

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по выполнению лабораторных работ
по дисциплине «ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ»
для студентов специальности 10.05.01 Компьютерная безопасность
специализация «Информационно-аналитическая и техническая
экспертиза компьютерных систем»

**Ставрополь
2026**

Содержание

ВВЕДЕНИЕ.....	4
Лабораторная работа 1.....	6
Лабораторная работа 2.....	12
Лабораторная работа 3.....	18
Лабораторная работа 4.....	23
Лабораторная работа 5.....	29
Лабораторная работа 6.....	35
Лабораторная работа 7.....	42
Лабораторная работа 8.....	48
Лабораторная работа 9.....	56
Лабораторная работа 10.....	61
Лабораторная работа 11.....	69
Лабораторная работа 12.....	73
Лабораторная работа 13.....	77
Лабораторная работа 14.....	81
Лабораторная работа 15.....	86
Лабораторная работа 16.....	90
Лабораторная работа 17.....	94
Лабораторная работа 18.....	99
Лабораторная работа 19.....	104
Лабораторная работа 20.....	108
Лабораторная работа 21.....	112
Лабораторная работа 22.....	117
Лабораторная работа 23.....	123
Лабораторная работа 24.....	128
Лабораторная работа 25.....	137
Лабораторная работа 26.....	144
Лабораторная работа 27.....	149

Лабораторная работа 28.....	154
Лабораторная работа 29.....	161
Лабораторная работа 30.....	167
Лабораторная работа 31.....	173
Лабораторная работа 32.....	180
Лабораторная работа 33.....	188
Лабораторная работа 34.....	196
СПИСОК ЛИТЕРАТУРЫ.....	203

ВВЕДЕНИЕ

Дисциплина «Технологии и методы программирования» является одной из основных при подготовке специалистов в области компьютерной безопасности. Материал, представленный в учебном пособии по дисциплине «Технологии и методы программирования» направлен на формирование набора общепрофессиональных и профессиональных компетенций специалистов, обучающихся по направлению 10.05.01 «Компьютерная безопасность», специализации «Информационно-аналитическая и техническая экспертиза компьютерных систем». В том числе:

ОПК-3 – способен на основании совокупности математических методов разрабатывать, обосновывать и реализовать процедуры решения задач профессиональной деятельности;

ОПК-7 – способен создавать программы на языках высокого и низкого уровня, применять методы и инструментальные средства программирования для решения профессиональных задач, осуществлять обоснованный выбор инструментария программирования и способов организации программ;

Главная задача учебного пособия – вооружить будущего специалиста в области компьютерной безопасности всеми необходимыми знаниями для создания профессиональных программных продуктов.

Задачи учебного пособия (лабораторного практикума) совпадают с задачами изучения дисциплины «Технологии и методы программирования» и могут быть сформулированы следующим образом:

- 1) применение современных методов и технологий программирования;
- 2) оценка качества программного обеспечения;
- 3) изучение структур данных;
- 4) изучение основных алгоритмов;
- 5) оценка сложности работы алгоритма;
- 6) основы структурного и объектно-ориентированного программирования;

7) порядок разработки современного безопасного программного обеспечения (ПО) и жизненный цикл ПО;

8) освоение современных тенденций разработки и сопровождения ПО.

Учебное пособие по дисциплине «Технологии и методы программирования» представляет собой лабораторный практикум по дисциплине «Технологии и методы программирования» и составлено в соответствии с рабочей программой дисциплины для специалистов 10.05.01 «Компьютерная безопасность», специализации «Информационно-аналитическая и техническая экспертиза компьютерных систем».

В методических указаниях приведены алгоритмы, структуры и технологии, широко используемые для разработки современного прикладного программного обеспечения. Структурно каждая из работ, в предлагаемом учебном пособии, состоит из краткого изложения теоретического материала, необходимого для ее выполнения, листингов с примерами исходного кода на языках C и C++, заданий к лабораторной работе, методики выполнения заданий, контрольных вопросов, списка литературы, рекомендуемой к использованию в данной теме.

Лабораторная работа 1

Исследование структур данных

Цель работы: исследование особенностей представления в памяти простых типов данных и структур.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Современные языки программирования, в том числе и C++ обладают возможностью определять *структуры* – набор переменных, объединённых одним именем. Это весьма удобное решение, листинг 1.1 предлагает пример структуры, предназначенной для хранения информации о студенте.

Листинг 1.1 – Определение структуры

```
struct student{  
    char[20] fname;           //имя  
    char[20] sname;          //фамилия  
    unsigned short int y,m,d; //год, месяц и день рождения  
};
```

Структура `student` объединяет в себе 6 полей, позволяющих сохранить имя, фамилию, дату рождения и номер курса.

Описанная в листинге 1.1 структура является не переменной, а **составным типом данных**, на основе которого можно объявлять другие переменные (листинг 1.2).

Листинг 1.2 – Объявление и инициализация переменных-структур

```
student st1={"Petr", "Petrov", 2001, 10, 5, 1};
student st2;
    st2.fname="Ivan";
    st2.fname="Ivanov";
    st2.y=1995;
    st2.m=1;
    st2.d=12;
    st2.academicyear=2;
```

Для вычисления размера, занимаемого в памяти любой переменной или структурой, следует обратиться за помощью к оператору `sizeof`. Листинг 1.3 демонстрирует несколько способов применения оператора.

Листинг 1.3 – Получение размера переменных и структур

```
int x;
    x=sizeof(int); //размер типа данных int
    x=sizeof(x);   //размер переменной x
float a[100];
    x=sizeof(a);   //размер массива a
    x=sizeof(a)/sizeof(float); //число эл-тов в массиве a
struct type_str{    //
double d;
    unsigned int i;
    char c;
};
x=sizeof(type_str); //размер структуры type_str
```

Представление массива – это отображение (абстрактного) массива компонент типа T в память, которая представляет собой массив компонент типа `word`.

Адрес, или индекс памяти, i -ой компоненты массива вычисляется с помощью линейной функции отображения:

$$i = i_0 + i \times S, \quad (1.1)$$

где: i_0 – адрес первой компоненты; S – число слов, которые занимает компонента.

Так как по определению `WORD` есть минимальная доступная единица памяти, то, по-видимому, желательно, чтобы S было целым числом. Если S не целое число слов памяти (а так бывает довольно часто), то S обычно округляется до ближайшего большего целого числа $[S]$. В этом случае каждая компонента массива занимает $[S]$ слов, причем часть слова величиной $[S] - S$ остается неиспользованной. Округление числа занимаемых слов до ближайшего целого называется **выравниванием**.

Для упаковки структур в C++ применяют директиву `#pragma pack`, например, так, как представлено в листинге 1.4.

Листинг 1.4 – C++ Упаковка структуры

```
#pragma pack(push, 1)
struct Foo{
    // ...
};
#pragma pack(pop)
```

Директива `pack` действует на первое объявление `struct`, `union` или `class` после этой директивы `#pragma`.

Отношение размера памяти, которая отводится для описания структуры данных, к размеру действительно занятой памяти называется коэффициентом использования памяти:

$$u = \frac{s}{s'} = \frac{s}{[s]}, \quad (1.2).$$

С одной стороны, разработчик стремится получить коэффициент использования памяти, близкий к 1. Но поскольку, с другой стороны, доступ к частям слова неясный и довольно неэффективный процесс, разработчику приходится идти на некоторый компромисс. Выравнивание понижает коэффициент использования памяти. Отказ от выравнивания может привести к неэффективному обращению к частям слова.

Записи отображаются в память простым объединением отображений их компонент. Адрес компоненты r_i относительно начального адреса записи называется смещением k_i :

$$k_i = S_1 + S_2 + \dots + S_{i-1}, \quad (1.3)$$

где S_i – размер компоненты.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задания

На языке программирования Си разработайте структуру данных согласно полученного варианта.

Варианты к заданию

1. Разработайте структуру данных позволяющую составлять прайс-лист магазина. В прайс-лист входят следующие позиции: название товара, цена за единицу, единица измерения, количество товара в магазине.
2. Подготовьте структуру данных позволяющую создавать расписание телепередач на неделю. В расписание входят следующие позиции: день недели, номер канала, время начала передачи, название передачи, жанр передачи.
3. Подготовьте структуру, предназначенную для описания информации в зачётной книжке студента. Необходимо предусмотреть следующие элементы: дисциплина, количество часов, вид отчётности (экзамен, зачёт, курсовая работа), оценка.
4. Подготовьте структуру для хранения данных о квартирах многоэтажного дома. В структуру следует включить следующие позиции: номер квартиры, число комнат, номер этажа, площадь, наличие балкона(ов), признак первого этажа, признак последнего этажа.
5. Подготовьте структуру для хранения сведений об автотранспортных средствах на стоянке: тип автомобиля (грузовой, легковой, ...), регистрационный номер, марка, когда поставлен на стоянку, ФИО владельца.
6. Подготовьте структуру для описания телевизора. В структуру должны входить: производитель, марка, диагональ, разрешение (HD, Full HD, 4K, и т.п.).

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и приведенные в ней листинги. В отчете следует схематично изобразить представление в памяти данных в упакованном и

неупакованном виде. Сделайте выводы об особенностях определения структур с учётом минимизации расхода памяти.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Какие скалярные типы данных вам известны?
2. Что такое выравнивание?
3. С помощью какого оператора можно выяснить размер переменной или структуры в памяти?
4. Как рассчитывается коэффициент использования памяти?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [2, 3];
дополнительная литература – [1].

Лабораторная работа 2

Алгоритмы работы с двоичными файлами

Цель работы: исследование особенностей последовательного хранения данных в бинарных файлах.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Если размер данных заранее неизвестен и может динамически изменяться в ходе выполнения кода, то наиболее простым способом их размещения в памяти станет последовательное хранение, будто бы это пачка листов бумаги, в которую вы каждый раз докладываете очередной лист.

В C++ определён ряд функций, позволяющий программисту осуществлять весь необходимый набор операций с файлом: создание файла, открытие существующего файла, запись данных в файл, чтение данных из файла, перемещение по элементам файла и завершение работы с файлом (заккрытие файла).

У большинства рассматриваемых в этой работе файловых функций имеется обязательный аргумент – указатель на определённую в файле `<stdio.h>` структуру `FILE`. Структура служит связующей нитью между программой и файлом, для этого она приспособлена для хранения всей необходимой информации об обрабатываемом файле, например, позиции курсора и сведений о статусе текущей операции. Для того чтобы служебные данные о файле стали доступны файловым функциям нам обязательно потребуется объявлять указатель на структуру (например, `FILE *f`).

Сразу после объявления с указателем `FILE *f` не будет сопоставлено никакого файла – он окажется неопределённым. Для того чтобы указатель стал приносить пользу, нам следует заполнить структуру `FILE` и получить её адрес.

Для того чтобы функция открытия файла приступила к выполнению своих обязанностей нам понадобится направить ей сведения об имени интересующего нас файла (для этого предназначен параметр `filename`) и конкретизировать режим доступа к файлу аргументом `mode`. Последний параметр весьма разносторонний, благодаря `mode` программист определяет концепцию всего дальнейшего алгоритма (табл. 2.1). Признаком успешного выполнения функции открытия файла будет значение отличное от `NULL`.

Вне зависимости от того для какой цели открывался файловый поток логическим завершением работы с файлом должно стать закрытие потока: `int fclose(FILE *stream)`. Единственный параметр `f` не что иное, как указатель на файловую структуру `FILE` полученный с помощью функции `fopen()` или `_wfopen()`. Признаком успешного выполнения операции станет нулевое значение, возвращаемое функцией.

Большинство файлов, которые вы обнаружите на дисках своего компьютера, обладают существенно более сложной структурой, чем текстовые файлы и содержат наборы самых разнообразных данных. При работе с файлами неизвестного формата или для обслуживания своих

собственных форматов данных следует применять приемы работы с *бинарными (двоичными) файлами*.

Таблица 2.1 – Варианты доступа к бинарному файлу

Режим	Описание	Доступ
"rb"	Открывает существующий бинарный файл	Только чтение
"wb"	Создаёт или перезаписывает существующий бинарный файл	Только запись
"ab"	Создаёт или открывает существующий бинарный файл	Только запись
"r+"	Открывает существующий текстовый файл	Чтение и запись
"w+"	Создаёт или перезаписывает существующий текстовый файл	Чтение и запись
"a+"	Создаёт или открывает существующий текстовый файл. Запись разрешена только в конец файла	Чтение и запись
"r+b" или "rb+"	Открывает существующий бинарный файл	Чтение и запись
"w+b" или "wb+"	Создаёт или перезаписывает существующий бинарный файл	Чтение и запись
"a+b" или "ab+"	Создаёт или открывает существующий бинарный файл	Чтение и запись

Бинарные файлы предоставляют нам наиболее общий и одновременно наиболее универсальный способ хранения данных. Для хранения целых чисел достаточно определить буфер соответствующего типа (например, `int`), для хранения вещественных – `float`. Двоичные файлы могут быть легко приспособлены для обслуживания более сложных типов данных, например, структур (листинги 2.1 и 2.2).

Листинг 2.1 – Запись структуры в бинарный файл

```
#include <stdio.h>
#include <fileapi.h>
struct rec { /*определяем структуру*/
    int a;
    float b;
    char c[16];
};
int main() {
```

```

    rec r={100,0.333,"Hello, World!"};
    FILE *f;
    const char *fn="c:\\rec.bin";
    f=fopen(fn,"wb+"); // режим создание-запись для бинарного
    файла
        if (f!=nullptr)
            { fwrite(&r, sizeof(r), 1, f);
              SetEndOfFile(f);
            }
    fclose(f);
    return 0;
}

```

Листинг 2.2 – Чтение структуры из бинарного файла

```

rec r; //переменная структурного типа
FILE *f;
const char *fn="c:\\rec.bin";
f=fopen(fn,"rb+"); // режим открытия-чтения бинарного файла
if (f!=nullptr)
{ fread(&r, sizeof(r), 1, f);;
  printf("%d %c", r.a, ' ');
  printf("%f %c", r.b, ' ');
  printf("%s", r.c);
}
else printf("Error!\n");
fclose(f);

```

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024 × 768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19², клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Воспользуйтесь разработанной вами в 1 лабораторной работой структурой и подготовьте консольное приложение, позволяющее пользователю сохранять, редактировать, удалять и отображать данные, сохраненные в **двоичном файле**. Приложение должно уметь осуществлять следующие операции:

1. Выводить все записи на экран.
2. Выводить отдельную запись на экран (например, по её индексу в файле).
3. Добавлять в файл новую запись.
4. Редактировать произвольную запись.
5. Удалять из файла произвольную запись.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и приведенные в ней листинги. Файловые операции должны быть организованы исключительно средствами структурного программирования (модуль `<stdio.h>`). Специализированные классы для работы с файлами из библиотеки объектно-ориентированного программирования использовать не рекомендуется.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. В чем отличие алгоритмов работы с текстовыми и бинарными файлами?
2. Что понимается под последовательным доступом?
3. Можно ли выделять последовательному файлу дополнительную память во время выполнения программы?
4. Можно ли выделять статическому массиву дополнительную память во время выполнения программы?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

- основная литература – [3];
дополнительная литература – [2].

Лабораторная работа 3

Оценка алгоритмов

Цель работы: исследование эмпирического и математического способов оценки алгоритмов.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Алгоритм (algorithm) – это любая корректно определённая вычислительная процедура, на вход которой подаётся некоторый набор значений, и результатом выполнения которой является выходной набор значений.

Эмпирический анализ алгоритма. Один из первых шагов в понимании производительности алгоритмов – это эмпирический анализ. Например, если имеется два алгоритма способные решить одну и ту же задачу. Мы их запускаем и сравниваем время их выполнения.

Самым простым способом сравнения времени выполнения алгоритма может стать использование двух меток времени T_1 и T_2 . Одна из них берётся в начале выполнения алгоритма, а вторая в конце. Разность $T_2 - T_1$ и есть время выполнения.

Способов получения отсчетов времени достаточно много, и они разнятся в различных операционных системах. В Windows для эмпирической оценки алгоритма лучше всего воспользоваться функциями: `QueryPerformanceCounter()` и `QueryPerformanceFrequency()`.

Первая из них позволяет обратиться к счетчику тиков процессора, который запускается в момент старта операционной системы и отсчитывает тики вплоть до выключения системы. Счетчик производительности работает с разрешением меньше микросекунды, поэтому поможет нам реализовать функционал очень точного хронометра. Вторая функция получает частоту счетчика производительности. Эта частота постоянна для компьютера и не меняется во время его работы. Зная частоту счетчика, мы сможем конвертировать тики процессора в микросекунды (листинг 3.1).

Листинг 3.1 – Шаблон кода для эмпирической оценки алгоритма

```
#include <stdio.h>
#include <iostream.h>
#include <windows.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[]){
    LARGE_INTEGER Frequency;
    QueryPerformanceFrequency(&Frequency); //частота счетчика
    LARGE_INTEGER StartTime, EndTime, ElapsedMicroseconds;
    QueryPerformanceCounter(&StartTime); //стартовое значение счетчика
    //здесь располагаем оцениваемый код
    QueryPerformanceCounter(&EndTime); //финишное значение счетчика
    ElapsedMicroseconds.QuadPart = EndTime.QuadPart -
    StartTime.QuadPart;
    cout<<ElapsedMicroseconds.QuadPart<<" ticks\n"; //количество
    тиков
    ElapsedMicroseconds.QuadPart *= 1000000;
    ElapsedMicroseconds.QuadPart /= Frequency.QuadPart;
```

```
cout<<ElapsedMicroseconds.QuadPart<<" microseconds\n";
system("pause">>void");
return 0;
}
```

Математический анализ алгоритма. В той ситуации, когда необходимо получить точную сравнительную оценку алгоритмов, решающих одну и ту же задачу, следует провести математический анализ алгоритма. Для этого понадобится осуществить ряд действий.

1. Изучить блок-схему (или исходный код) алгоритма и построить список абстрактных операций, входящих в состав алгоритма.
2. Подсчитать сколько раз в ходе исполнения алгоритма будет выполнена каждая из входящих в список операций.
3. Сформировать модель потока данных, поступающих на вход алгоритма, хорошей идеей окажется анализ реакции алгоритма не только на корректные, но и на заведомо ошибочные данные.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

На языке программирования Си разработайте функции согласно полученного варианта.

Варианты к заданию

Разработайте две функции поиска наибольшего общего делителя (НОД) произвольных целых чисел А и В. Первая процедура должна применять «лобовой» поиск (обычный перебор). Вторая процедура может использовать любой известный вам алгоритм поиска НОД, например, алгоритм Евклида.

Разработайте две функции получения факториала произвольного целого числа. Первая процедура основана на классическом решении (например, рекурсивном алгоритме), а вторая на формуле Джеймса Стирлинга: $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$, где: e – основание натурального алгоритма.

Разработайте две функции получения числа Фибоначчи. Первая процедура вычисляет запрошенное число с помощью классического рекуррентного соотношения: $F_0 = 0, F_1 = 1, F_i = F_{i-1} + F_{i-2}, i \geq 2$. Например, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Вторая процедура получает число Фибоначчи с помощью формулы Бине.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и приведенные в ней листинги. Перед защитой лабораторной работы студент должен осуществить эмпирическую и математическую оценку разработанных алгоритмов. Эмпирическая оценка должна включать не менее 10 измерений для разных входных значений.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Какими признаками должен обладать качественный алгоритм?
2. Какой алгоритм задаёт определённые действия, обозначая их в единственной последовательности?
3. Какой алгоритм предлагает программу решения задачи несколькими путями или способами?
4. Какие способы анализа алгоритмов вам известны?
5. Позволит ли модель вычисления RAM оценить как алгоритм будет работать на компьютере?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [1 – 4];

дополнительная литература – [2].

Лабораторная работа 4

Алгоритмы сортировки массивов N^2

Цель работы: исследование методов сортировки одномерных массивов временной сложности N^2 .

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Под сортировкой понимается процесс перестановки объектов какого-то множества в определённом порядке.

Алгоритмы сортировки обычно оцениваются по 3 параметрам:

1. **Время сортировки** – основной параметр, характеризующий быстродействие алгоритма.
2. **Память** – ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных.
3. **Устойчивость** – устойчивая сортировка не меняет взаимного расположения равных элементов. Такое свойство может быть очень полезным,

если они состоят из нескольких полей, а сортировка происходит по одному из них.

Если критерий экономии памяти принят за ограничение, то удобная мера эффективности получается при подсчёте числа C необходимых сравнений ключей и M пересылок элементов.

Числа M и C определяются функциями от числа N сортируемых элементов. Рассматриваемые в первой части лабораторной работы алгоритмы предусматривают N^2 сравнений. Это далеко не лучший показатель, поэтому все эти алгоритмы относятся к классу простых.

Сортировка простым обменом. Алгоритм сортировки простым обменом основан на следующем принципе: осуществляется просмотр всех элементов массива с целью выявления элемента с наименьшим весом, найденный элемент переносится в первую позицию массива, а основные элементы смещаются. На следующей итерации алгоритма процедура повторяется, но уже без учёта первого элемента, и т.п. Очень часто этот метод называют «**пузырьковым**», когда элемент с наименьшим весом всплывает на поверхность. Работа алгоритма становится нагляднее, если развернуть массив по вертикали (рис. 4.1, а).

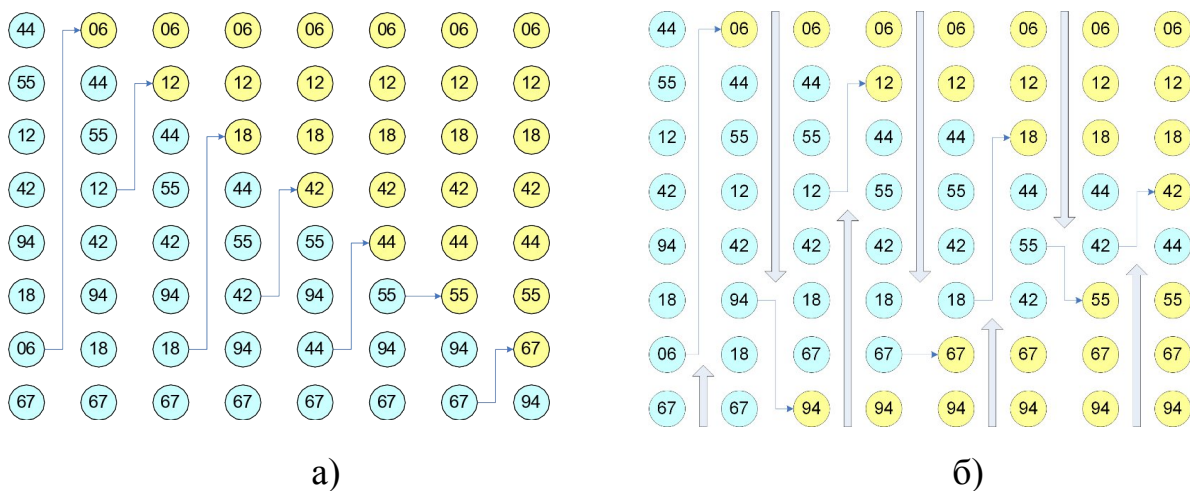


Рисунок 4.1 – Пример сортировки: а) простым обменом; б) Шейкер-сортировка

Шейкер-сортировка. Сортировка пузырьковым методом обладает примечательным недостатком – один неправильно расположенный «лёгкий пузырёк» расположенный в «тяжёлом» конце всплывёт на место за один проход. Например, 12,8,42,44,55,67,94,06. Неправильно расположенный элемент в «лёгком» конце массива будет опускаться за несколько проходов. Например, 94,06,12,8,42,44,55,67. Эта неестественная асимметрия подсказывает направление улучшения алгоритма – менять направление следующих друг за другом проходов. Работа такого алгоритма продемонстрирована рисунком 4.1, а.

В отличие от сортировки пузырьком (которая не находит места в серьёзных программах), шейкер-сортировка с успехом применяется в тех случаях, когда заранее известно, что все элементы почти упорядочены.

Сортировка простым выбором. Метод сортировки **простым выбором** основан на следующем правиле:

1. Находится элемент с наименьшим ключом.
2. Он меняется местами с первым элементом a_1 .

Эти операции повторяются с $n - 1$ элементами, затем с $n - 2$, и т.д., пока не останется только один элемент – наибольший (рис. 4.2, а).

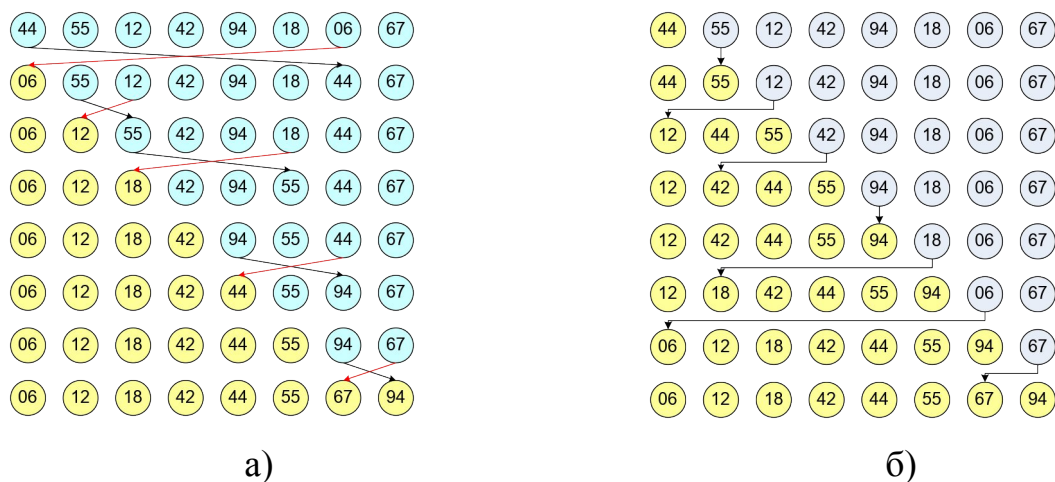


Рисунок 4.2 – Пример сортировки: а) простым выбором; б) простыми вставками

Сортировка простыми вставками. Рассмотрим метод сортировки **простыми вставками**. Элементы исходного массива условно разделяются на готовую последовательность $A[0], \dots, A[i]$ и входную последовательность $A[i + 1], \dots, A[n]$. На каждом шаге, начиная с $i = 1$ и увеличивая i на единицу берут i -й элемент входной последовательности и передают в готовую последовательность, вставляя его на подходящее место. Порядок работы алгоритма отражен на рисунке 4.2, б.

Сортировка бинарными вставками. Алгоритм сортировки простыми вставками можно улучшить, пользуясь тем, что готовая последовательность $A[0], \dots, A[i]$ (в которую нужно включить новый элемент) уже упорядочена. Поэтому место включения можно найти значительно быстрее. На этой идее основан алгоритм сортировки **бинарными вставками**.

Будем разбирать алгоритм, рассматривая его действия на i -м шаге. Как говорилось выше, последовательность к этому моменту разделена на две части: готовую $A[0], \dots, A[i]$ и неупорядоченную $A[i + 1], \dots, A[n]$. Выбираем любой элемент из неупорядоченной последовательности и сравниваем его с элементом, находящимся в центре уже отсортированной последовательности. Произведя сравнение, мы получаем важную информацию о том, в какой половине уже отсортированной последовательности должен оказаться очередной элемент. Далее операция повторяется с выбранной частью отсортированной последовательности до тех пор, пока наш элемент не попадёт в установленное место.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или

больше) цветами с диагональю не менее 19^2 , клавиатура, мышь. **Программное обеспечение** – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Подготовьте одномерный массив целых чисел произвольной размерности N и заполните его случайными значениями. Разработайте 2 функции сортировки (по вариантам). Осуществите эмпирическую и математическую оценку каждого из алгоритмов сортировки. Изменяя параметр N , убедитесь в соответствии алгоритма классу сортировки N^2 .

В отчете по работе для каждого из разработанных алгоритмов заполните следующую таблицу:

	Название алгоритма	
	Число сравнений	Число перестановок
N=10		
N=100		
N=1000		
N=10000		

Варианты к заданию

1. Сортировка простым обменом и сортировка простым выбором.
2. Шейкер-сортировка и сортировка простыми вставками.
3. Сортировка бинарными вставками и сортировка простым выбором.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе. Разработайте универсальную функцию, позволяющую осуществлять вывод значений массива на экран на любом этапе сортировки. Во время отладки проекта используйте разработанную функцию для проверки правильности работы алгоритма. Перед защитой лабораторной работы студент должен осуществить эмпирическую и математическую оценку разработанных алгоритмов.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Для чего предназначена O-нотация?
2. За счет чего можно ускорить алгоритм сортировки массива?
3. Как осуществляется математический анализ алгоритма сортировки?
4. Почему на практике не рекомендуется использовать алгоритмы N^2 ?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [1 – 4];

дополнительная литература – [2].

Лабораторная работа 5

Алгоритмы сортировки массивов $N(\log N)$

Цель работы: исследование методов сортировки одномерных массивов временной сложности $N(\log N)$.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Сортировка Шелла. Сортировка Шелла представляет собой несколько этапов сортировок, каждая из которых программируется, как классическая сортировка включениями. На рисунке 5.1 представлен процесс сортировки массива из 8-ми элементов способом Шелла.

На первом проходе отдельно группируются и сортируются элементы, отстоящие друг от друга на 4 позиции. После этого элементы объединяются в группы, отстоящие друг от друга на 2 позиции, и сортируются заново. На 3-м проходе элементы сортируются обычной сортировкой включениями. Сначала может показаться, что необходимость в нескольких проходах сортировки в

каждом из которых участвуют все элементы, потребует больше работы чем сэкономит. Однако на каждом шаге сортировки либо участвует мало элементов, либо они уже довольно хорошо упорядочены и требуют мало перестановок.

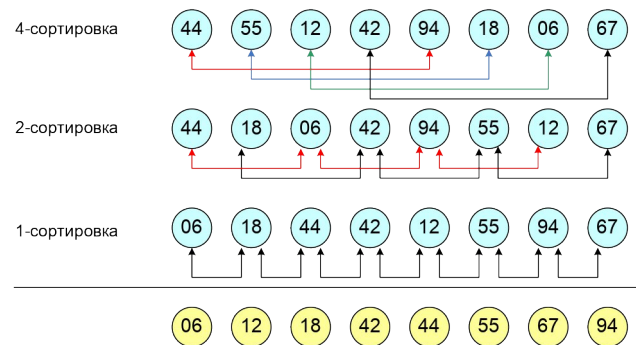


Рисунок 5.1 – Сортировка Шелла

Пирамидальная сортировка. Метод сортировки простым выбором основан на повторном выборе наименьшего ключа среди n элементов, затем среди $N - 1$ элементов, и т.д. Соответственно поиск наименьшего элемента из N элементов потребует $N - 1$ сравнений. Направление улучшения сортировки – получать от каждого прохода больше информации, чем просто указание на один наименьший элемент.

Пирамида – это бинарное дерево (каждый узел которого имеет два дочерних узла) высоты k , в котором:

1. Все узлы имеют глубину k или $k - 1$ - дерево сбалансированное.
2. При этом уровень $k - 1$ полностью заполнен, а уровень k заполнен слева направо.
3. Выполняется «свойство пирамиды»: каждый дочерний элемент меньше, либо равен родителю.

Как хранить пирамиду? Наименее хлопотно – поместить ее в массив. Соответствие между геометрической структурой пирамиды как дерева и массивом устанавливается по следующей схеме:

- в $A[0]$ хранится корень дерева;
- левый и правый сыновья элемента $A[i]$ хранятся, соответственно, в $A[2i + 1]$ и $A[2i + 2]$.

Пирамидальная сортировка состоит из двух фаз:

1. Построение пирамиды (просеивание) (см. рис. 5.2).
2. Фаза сортировки (см. рис. 5.3).

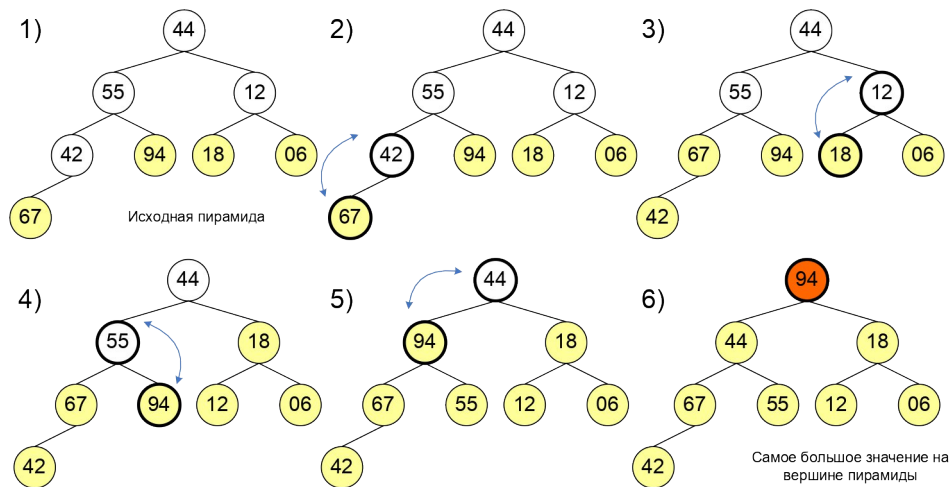


Рисунок 5.2 – Фаза просеивания элемента пирамиды

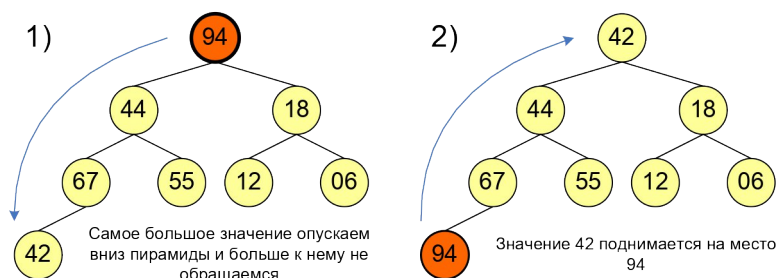


Рисунок 5.3 – Фаза сортировки пирамиды

Очевидно, в конец массива каждый раз попадает максимальный элемент из текущей пирамиды, поэтому в правой части постепенно возникает упорядоченная последовательность. Вычислительные затраты пирамидальной сортировки составляют $N \log(N)$.

Сортировка с разделением (быстрая сортировка). Сортировка с разделением, хоть и была разработана более 50 лет назад К. Хоором, является

наиболее широко применяемым и одним из самых эффективных алгоритмов. Сортировка основана на том принципе, что для достижения максимальной эффективности следует производить обмены элементов на больших расстояниях.

Метод основан на подходе «разделяй и властвуй». Общая схема такова:

1. Из центральной части массива случайным образом выбираем какой-то элемент, например X .
2. Двигаемся по массиву слева направо пока не находим элемент $A[i] > X$.
3. Двигаемся по массиву справа налево пока не находим элемент $A[j] < X$.
4. Элементы $A[i]$ и $A[j]$ меняем местами.
5. Вновь продолжаем процесс с обменом пока два просмотра не встретятся где-то в центре массива.

В результате массив разделится на две части, причём в левой его части будут находиться элементы меньше X , а в правой – больше X . После разделения массива вновь применяем механизм разделения, но на этот раз к частям массива. Процесс продолжается до тех пор, пока каждая из частей не будет состоять из одного единственного элемента.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Подготовьте одномерный массив целых чисел произвольной размерности N и заполните его случайными значениями. Разработайте функции сортировки (по вариантам). Осуществите эмпирическую и математическую оценку алгоритма сортировки. Изменяя параметр N , убедитесь в соответствии алгоритма классу сортировки N .

В отчете по работе для каждого из разработанных алгоритмов заполните следующую таблицу:

	Название алгоритма	
	Число сравнений	Число перестановок
N=10		
N=100		
N=1000		
N=10000		

Варианты к заданию

1. Сортировка с помощью включений (Шелла).
2. Пирамидальная сортировка.
3. Сортировка с разделениями (быстрая).

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе. Разработайте универсальную функцию, позволяющую осуществлять вывод значений массива на экран на любом этапе сортировки. Во время отладки проекта используйте разработанную функцию для проверки правильности работы алгоритма. Перед защитой лабораторной работы студент должен осуществить эмпирическую и математическую оценку разработанных алгоритмов.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы;
2. Ответов на контрольные вопросы;
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. По каким параметрам следует оценивать алгоритмы сортировки?
2. Каким образом осуществляется преобразование массива в пирамиду в пирамидальной сортировке?
3. Какими свойствами должна обладать пирамида?
4. Что понимается под термином «разделяй и властвуй» в программировании?
5. В чем принципиальное преимущество алгоритма быстрой сортировки над другими алгоритмами упорядочивания массивов?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

- основная литература – [1 – 4];
дополнительная литература – [2].

Лабораторная работа 6

Алгоритмы сортировки последовательностей

Цель работы: исследование методов сортировки последовательностей на примере бинарных файлов.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Все рассмотренные в предыдущих лабораторных работах сортировки неприменимы, если сортируемые данные не помещаются в оперативной памяти и находятся на внешнем запоминающем устройстве с последовательным доступом. В таком случае основной метод сортировки основан на идее слияния.

Сортировка простым слиянием. Идея метода сортировки простым слиянием заключается в следующем:

~ последовательность A разбивается на две половины B и C;

- ~ последовательности В и С сливаются при помощи объединения отдельных элементов в упорядоченные пары;
- ~ полученной последовательности присваивается имя А и шаги 1 и 2 повторяются, но на этот раз упорядоченные пары сливаются в упорядоченные четвёрки;
- ~ предыдущие шаги повторяются: четвёрки сливаются в восьмёрки и весь процесс продолжается пока не будет упорядочена вся последовательность (ведь длины слагаемых последовательностей всегда удваиваются).

Изложенная идея представлена на рисунке 6.1.

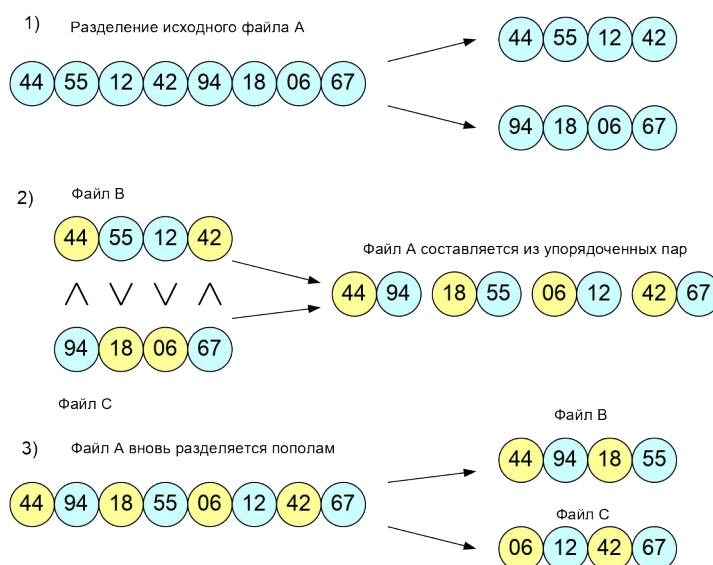


Рисунок 6.1 – Процесс разделения и слияния файлов

Чтобы получить ответ на вопрос: «Каким образом сливать упорядоченные 4-ки, 8-ки, и т.п.?» следует изучить рисунок 6.2.

Для слияния двух последовательностей мы производим сравнение первых элементов файлов В и С. Наименьший элемент следует в файл А, и процесс повторяется вновь до тех пор, пока упорядоченные четвёрки не превратятся в упорядоченные восьмёрки.

Сортировка простым слиянием достаточно эффективна. Общее число пересылок, осуществляемых при полной сортировке файла равно $M = N \cdot [\log_2 N]$.

Число сравнений S по значению ещё меньше, т.к. при копировании остатка последовательности сравнения не производятся. Хотя показатели сортировки слияния схожи с показателями производительности пирамидальной сортировки, но сортировка слиянием требует вдвое больше памяти ($2N$), что затрудняет её использование для больших массивов данных, расположенных в ОЗУ.

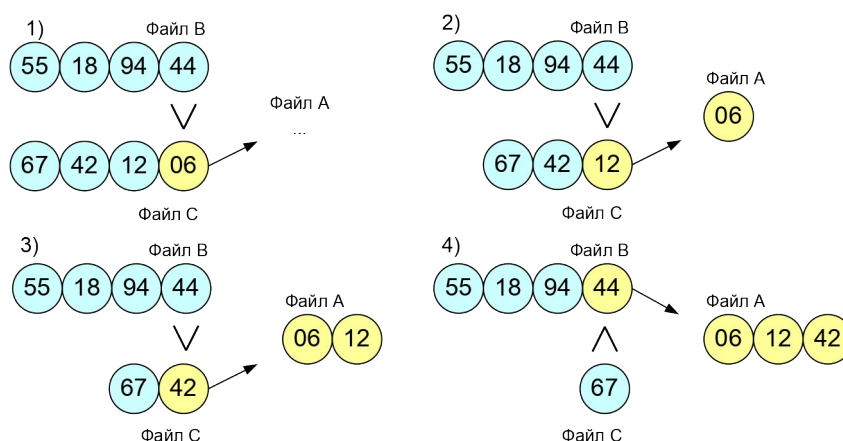


Рисунок 6.2 – Слияние двух последовательностей

Естественное слияние. В случае простого слияния мы ничего не выигрываем, если данные уже частично рассортированы. На k -м проходе длина всех рассортированных последовательностей уже меньше или равна 2^k и их уже можно объединять. Выполнение сортировки продемонстрировано на рисунке 6.3.

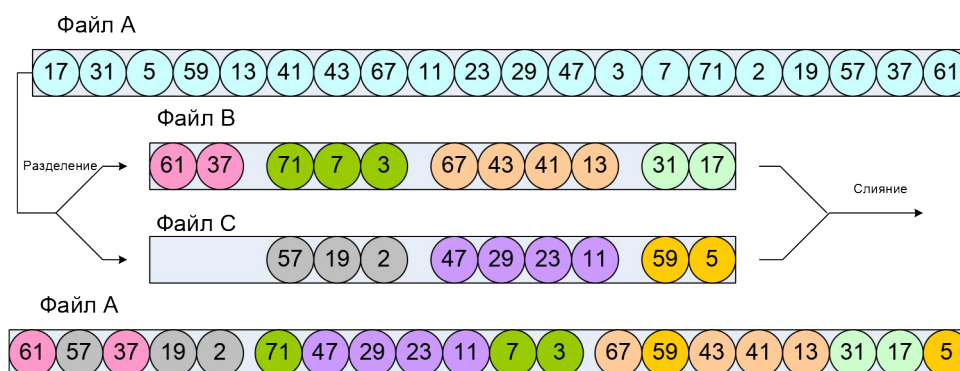


Рисунок 6.3 – Демонстрация алгоритма естественного слияния

Пусть исходная последовательность элементов задана в виде файла А. Кроме того, у нас имеется два вспомогательных файла В и С. Каждый проход алгоритма состоит из:

- ~ фазы распределения, которая распределяет серии из А в В и С;
- ~ фазы слияния, которая сливает серии из В и С в А.

Обратите внимание на то, что как во время распределения, так и в момент слияния алгоритм формирует упорядоченные подпоследовательности. Поэтому с каждым проходом длины упорядоченных подпоследовательностей увеличиваются, а их количество уменьшается.

Многопутевое слияние. Основное отличие сбалансированного многопутевого слияния от естественного двухфазного слияния, заключается в том, что это однофазная сортировка. Демонстрация алгоритма многопутевого слияния показана на рис. 6.4.

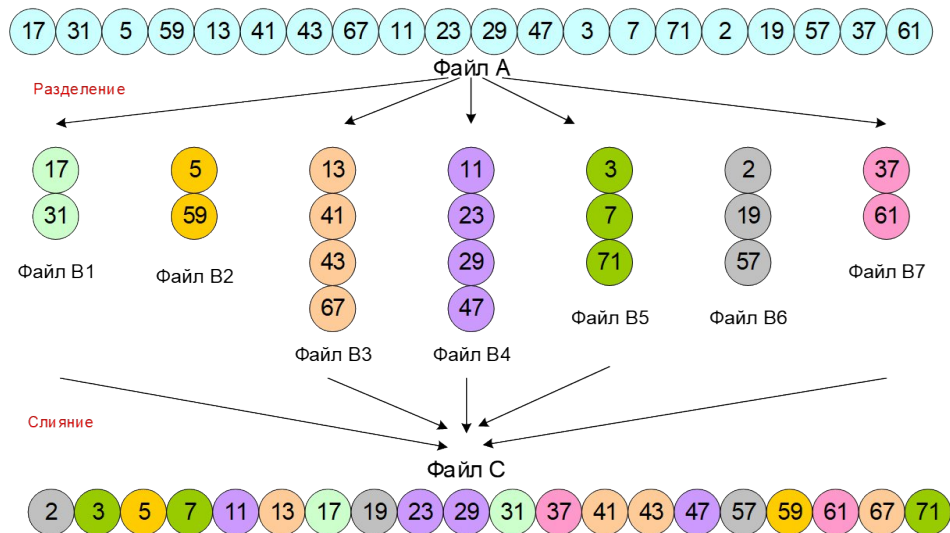


Рисунок 6.4 – Демонстрация алгоритма многопутевого слияния

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19^2 , клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Сформируйте двоичный файл размерности N , хранящий целые случайные числа. Разработайте функцию, позволяющую упорядочить исходный файл с использованием алгоритма вашего варианта.

Варианты к заданию

1. Метод сортировки простым слиянием.
2. Метод сортировки естественным слиянием.
3. Метод многопутевого слияния.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе. При разработке исходного кода предусмотрите возможность подсчета операций сравнений и пересылок, осуществляемых во время сортировки файла. Произведите эмпирическую оценку алгоритма на файлах разной размерности.

Предусмотрите возможность просмотра состояния сортируемого файла на экране.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Почему для упорядочивания данных в файлах не целесообразно использовать алгоритмы сортировки массивов?
2. Что может выступать мерой эффективности алгоритмов сортировки файлов?
3. Почему при разработке алгоритмов сортировки файлов не ограничивают использование вторичной памяти?
4. Какой из алгоритмов сортировки файлов на ваш взгляд самый эффективный?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление

программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [2– 4];

дополнительная литература – [2].

Лабораторная работа 7

Полустатические структуры данных

Цель работы: исследование организации и способов представления в памяти полустатических структур данных.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

В отличие от своих статических коллег (например, массивов) полустатические структуры имеют переменную длину, и что немаловажно, процедуры для её изменения, однако на размерность структуры накладываются ограничения, не давая ей превышать определенный лимит.

Классическими представителями полустатических структур выступают: стек, очередь и дек. Логическое представление перечисленных структур в большей степени соответствует спискам. Физическое представление полустатических структур зависит от программистов, это может быть массив, линейный связный список или даже класс.

Стек (*stack*) – структура, представляющая собой список элементов, организованных по принципу LIFO (*last in – first out*) – «последним пришел – первым вышел».

Как минимум в стеке должно быть реализовано две основных операции:

- ~ `push()` – помещает в стек новое значение;
- ~ `pop()` – извлекает из стека значение.

Кроме того, полезными могут оказаться вспомогательные операции, позволяющие:

- ~ `size()` – выяснить число элементов в стеке;
- ~ `empty()` – очистить стек;
- ~ `top()` – осуществить неразрушающее чтение из вершины.

Принцип включения элементов в стек и исключения элементов отражает рисунок 7.1.

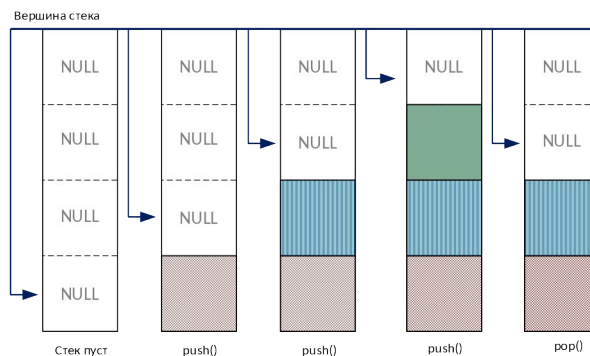


Рисунок 7.1 – Стек, включение `push()` и исключение `pop()` элементов

Указатель стека обычно адресует его первый элемент. После занесения элемента в стек указатель модифицируется – переходит на очередной свободный элемент. Исключение элемента из стека также приводит к модификации указателя, на этот раз он смещается на только что освободившийся слот в памяти. Отметим, что, применяя термин «указатель стека» мы далеко не всегда имеем в виду физический указатель на ячейку в

памяти. В ситуации, когда стек реализован на основе массива вместо указателей мы оперируем индексами ячеек массива. Реализуем программный стек на базе одномерного массива. Объявления констант, типов и структур данных необходимых для работы стека вы обнаружите в листинге 7.1.

Листинг 7.1 – Константы и структуры для стека

```
const int MAX_ITEMS = 10;           //размер стека
const int ERR_STACK_OVERFLOW  = -100; //ошибка переполнения
const int ERR_STACK_UNDERFLOW = -101; //ошибка чтения пустого
стека
typedef char T;                     //тип данных элемента стека
struct TStack {                     //структура стека
    T item[MAX_ITEMS];              //массив с элементами данных T
    unsigned int itemscout; //количество занятых элементов
};
```

Мы постарались создать максимально универсальный стек. Благодаря константе `MAX_ITEMS` определяется предельный размер полустатической структуры. Тип данных элемента стека `T` в нашем примере выбран символьным, но его при желании несложно поменять, для этого достаточно заменить `char`, на другой тип, допустим `int` или `float`.

Программная логика стека представлена в листинге 7.2. Здесь вы обнаружите реализацию операций помещения/извлечения значения, очистки стека и неразрушающего чтения элемента из вершины стека.

Листинг 7.2 – Функции для работы со стеком

```
void push(TStack *stack, const T value) {
    if (stack->itemscout < MAX_ITEMS) {
        stack->item[stack->itemscout] = value;
        stack->itemscout++;
    } else exit(ERR_STACK_OVERFLOW);
}

T pop(TStack *stack) {
    if (stack->itemscout > 0) {
        stack->itemscout--;
        return stack->item[stack->itemscout];
    }
```

```

        } else exit(ERR_STACK_UNDERFLOW);
    }
void empty(TStack *stack) {
    stack->itemcount = 0;
}
T top(const TStack *stack) {
    if (stack->itemcount > 0) {
        return stack->item[stack->itemcount-1];
    }
    else exit(ERR_STACK_UNDERFLOW);
}

```

Очередь (*queue*) – структура, представляющая собой список элементов, организованных по принципу FIFO (*First In – First Out*, «первым пришел – первым исключается»).

Основные операции, реализуемые в очереди аналогичны операциям стека, с той лишь разницей, что здесь включение и исключение осуществляется с разных сторон:

```

~      push() – помещает в хвост очереди новое значение;
~      pop() – извлекает из головы очереди значение.

```

К дополнительным операциям стоит отнести:

```

~      size() – число элементов в очереди;
~      empty() – очистка очереди;
~      top() – осуществить неразрушающее чтение элемента из головы.

```

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024 × 768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19², клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Разработайте полустатическую структуру руководствуясь вариантом задания.

Варианты к заданию

1. Разработайте стек на основе массива для хранения значений CHAR. Стек должен выполнять операции `push`, `pop`, `size`, `empty` и `top`.
2. Разработайте стек на основе массива для хранения значений INT. Стек должен выполнять операции `push`, `pop`, `size`, `empty` и `top`.
3. Разработайте очередь на основе массива для хранения значений UNSIGNED INT. Очередь должна выполнять операции `push`, `pop`, `size`, `empty` и `top`.
4. Разработайте стек на основе массива для хранения значений FLOAT. Стек должен должен выполнять операции `push`, `pop`, `size`, `empty` и `top`.
5. Разработайте очередь на основе массива для хранения значений FLOAT. Очередь должна выполнять операции `push`, `pop`, `size`, `empty` и `top`.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе. При выполнении задания не используйте приемы объектно-ориентированного программирования. При разработке структуры предусмотрите дополнительную операцию, позволяющую вывести на экран все содержимое структуры. Эта операция потребуется, для проверки корректности работы структуры

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. В чем заключается отличие между статическими и полустатическими структурами?
2. Какими признаками обладают полустатические структуры?
3. Где на практике применяются полустатические структуры?
4. Какими недостатками обладают полустатические структуры?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [2 – 4];

дополнительная литература – [2].

Лабораторная работа 8

Динамические структуры данных

Цель работы: исследование организации и способов представления в памяти динамических структур данных.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Динамические структуры по определению характеризуются отсутствием физической смежности элементов структуры в памяти, непостоянством и непредсказуемостью размера (числа элементов) структуры в процессе ее обработки. Связь между элементами динамических структур зачастую реализуется с помощью указателей.

С точки зрения хранения данных любая переменная или константа представляют собой область памяти, в которую занесено определенное значение. Например, при объявлении целочисленной переменной `x` типа `int` (`integer`) в памяти компьютера резервируется 4 байта для хранения

значения этой переменной. Для обращения к содержимому переменной можно пойти традиционным путем, просто вызвав переменную по ее имени. Однако существует еще один способ, позволяющий читать и записывать данные непосредственно в область памяти. Для этого предназначены указатели.

Указатель представляет собой адрес определенной области памяти. Ещё раз повторимся – указатель хранит *не содержимое* памяти, а *адрес к ячейкам памяти*, в которых размещены данные. Поэтому указатель не занимает никакого места, кроме того, которое необходимо для хранящегося в нем адреса. Таким образом, в 32-разрядных системах указатель затребует всего 4 байта памяти, а 64-разрядных – 8 байт.

Разрабатывая приложения на языке C++, мы очень часто будем получать доступ к структурам (и их полям) через указатели. В подобных ситуациях нам окажется полезной операция «->» называемая операцией разыменования указателей на структуры (листинг 8.1).

Листинг 8.1 – Объявление указателя на структуру

```
struct cur_pos {
    unsigned int x,y;
};
cur_pos ср,*рср; //определили переменную ср и указатель рср
рср=&ср;         //теперь указатель "смотрит" на ср
рср->x=10;        //запись в поля структуры через указатель
рср->y=20;        //операции ->
int a,b;
a=(*рср).x;      //альтернативный вариант доступа к элементам
структуры
(*рср).y=b;
```

В примере создается переменная структурного типа `ср` и указатель `рср`. Для получения адреса переменной в памяти задействуется уже знакомая схема – `рср=&ср`, в которой используется оператор взятия адреса `&`. Затем

наступает черед операции $\rightarrow$, благодаря ей мы разыменуем указатель `pcp` и получим доступ к полям `x` и `y`.

Обратите внимание (листинг 8.1), что в Си предусмотрен альтернативный способ решения задачи доступа к полям структуры через указатель – выражение `(*pcp).x` выступает полным аналогом выражению `pcp->x`.

Списком (*list*) называется упорядоченное множество, состоящее из переменного числа элементов, к которым применимы операции включения, исключения.

При таком подходе элемент динамической структуры должен состоять из двух полей:

1. Информационного поля (поля данных), в котором содержатся те данные, ради которых и создается структура.
2. Поле связей, в котором содержатся один или несколько указателей, связывающий данный элемент с другими элементами структуры.

В самом общем случае, если мы планируем создать список, специализирующийся на хранении целых чисел то элемент списка, может быть представлен в виде, предложенном в листинге 8.2.

Листинг 8.2 – Элемент односвязного списка

```
struct ItemData{ //структура для поля данных элемента списка
    int a;
    //...
};
struct Item{      //структура элемента списка
    ItemData d;   //поле данных
    Item *next;   //поле указателя (связок)
};
```

В зависимости от стоящих перед программистом задач в качестве поля данных может быть использована любая другая произвольная структура.

Ключевым преимуществом связанного списка (как динамической структуры) является то, что её размер ограничен лишь предельным значением адресного пространства программы и размерностью памяти компьютера.

В результате мы получаем возможность работать с данными с заранее неизвестной размерностью. Благодаря тому, что операции вставки (рис. 8.2), удаления (рис. 8.3) и перемещения элементов связанного списка осуществляются просто путем перестановки указателей эти действия происходят практически мгновенно. Процедуру добавления нового элемента в начало список раскрывает листинг 8.3.

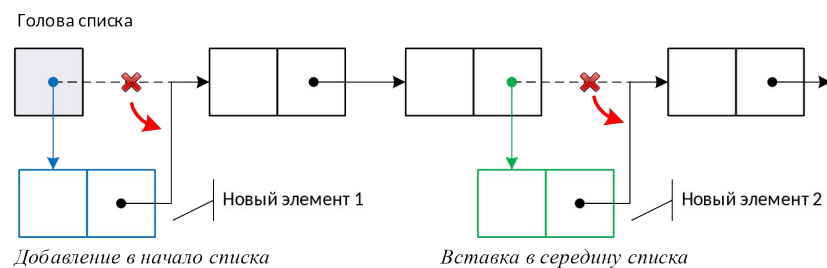


Рисунок 8.1 – Вставка элемента в список

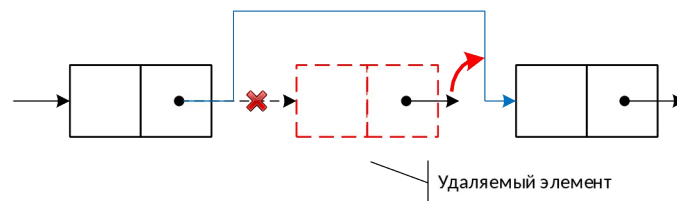


Рисунок 8.2 – Удаление элемента из списка

Листинг 8.3 – Вставка нового элемента в начало списка

```
void AddItem(Item **header, ItemData &d) {
    Item *NewItem = new Item;
    NewItem->d.a = d.a;
    NewItem->next = *header;
    *header = NewItem;
}
```

На вход функции поступает указатель на голову списка `header` и данные `d` для вновь создаваемого элемента. Функция создает новый элемент `NewItem` и переставляет указатели точно, так как представлено на рис. 8.2.

Для удаления элемента из списка (листинг 8.4) на вход функции подается указатель на голову списка `header` и содержимое поля данных `x` подлежащего удалению элемента.

Листинг 8.4 – Удаление элемента из списка

```
bool DeleteItem(Item **header, ItemData &x){
    if(*header== 0){          // исходный список пуст
        return false;
    }
    Item *t = *header;
    if(t->d.a == x.a){ // удаляется 1-й элемент
        *header = t->next;
        delete t;
        return true;
    }
    Item *t1 = t->next;
    while(t1){
        if(t1->d.a == x.a){ //удаляется элемент >1
            t->next = t1->next;
            delete t1;
            return true;
        }
        t = t1;
        t1 = t1->next;
    }
}
```

Алгоритм удаления элемента из списка немного сложнее. Для корректного решения задачи нам приходится предусмотреть три варианта развития событий:

1. Список пуст – удалять нечего.
2. Осуществляется удаление самого первого элемента, в этом случае нам надо поменять содержимое головного указателя.
3. Удаляется элемент с порядковым номером больше 1.

Кроме освобождения памяти, ставшего ненужным элементом, мы вновь переносим указатель.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Разработайте программу, позволяющую хранить данные типа T (см. варианты к заданию) в формате односвязного списка. Программа должна позволять: создавать список; добавлять элемент в конец списка; вставлять элемент в произвольное место списка; удалять любой элемент из списка; читать данные из списка; уничтожать весь список.

Варианты к заданию

1. $T = \text{char}$.
2. $T = \text{int}$.
3. $T = \text{short int}$.
4. $T = \text{float}$.
5. $T = \text{double}$.
6. $T = \text{unsigned int}$.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. При выполнении задания не используйте приемы объектно-ориентированного программирования.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Почему элементы динамической структуры располагаются по непредсказуемым адресам в памяти?
2. Какие поля должны входить в элемент динамической структуры?
3. Какими недостатками обладают динамические структуры?
4. Приведите примеры практического использования связанных списков?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно

содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [1,2,4];

дополнительная литература – [2].

Лабораторная работа 9

Двоичные деревья поиска

Цель работы: исследование способов реализации двоичных деревьев поиска.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Дерево – это непустое конечное множество элементов, один из которых называется корнем, а остальные делятся на несколько непересекающихся подмножеств, каждое из которых является деревом. Линия связи между парой узлов дерева называется обычно **ветвью**. Те узлы, которые не ссылаются ни на какие другие узлы дерева, называются **листьями** (или терминальными вершинами). Узел, не являющийся листом или корнем, считается промежуточным или узлом ветвления (нетерминальной или внутренней вершиной).

Существуют m -арные деревья, т.е. такие деревья, у каждой вершины которых от нуля до m дочерних вершин (где m может быть равно 0,1,2,3 и т.д.). При $m = 2$ такие деревья называются соответственно **бинарными** (**двоичными**).

Деревья можно представлять с помощью связанных списков. Чаще всего используется связанное представление деревьев, т.к. оно очень сильно напоминает логическое. Связное хранение состоит в том, что задается связь от родительской вершины к дочерней. Если двоичное дерево организовано таким образом, что для каждого узла p все значения (ключи) в левом поддереве меньше значения в узле p , а все значения в правом поддереве не меньше значения в p , то это дерево называется деревом поиска (ДДП).

Пример правильного и неправильного ДДП представлен на рисунке (рис. 9.1).

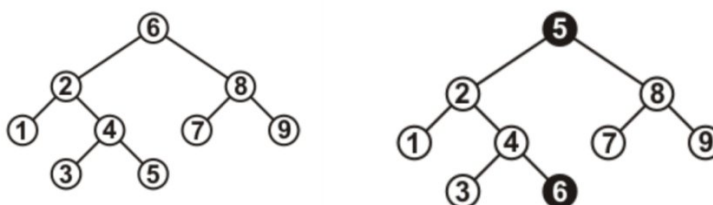


Рисунок 9.1 – Правильное (слева) и неправильное (справа) ДДП

ДДП должно позволять выполнять следующие основные операции:

- ~ создание дерева;
- ~ вставка вершины;
- ~ удаление вершины;
- ~ вывод (печать) дерева;
- ~ поиск вершины по ключу;
- ~ определение вершин с минимальным и максимальным значениями ключа;
- ~ обход дерева;

- ~ переход к последующей или предыдущей вершине, в порядке, определяемом ключами;
- ~ проверка пустоты дерева.

Для реализации ДДП программисты используют узлы, содержащие три указателя. Пример определения узла двоичного дерева поиска представлен в листинге 9.1.

Листинг 9.1 – Определение узла двоичного дерева поиска

```
typedef struct tree {
    item_type item; //элемент данных
struct tree *parent; //указатель на родительский узел
struct tree *left; //указатель на левый дочерний узел
struct tree *right; //указатель на правый дочерний узел
} tree;
```

Кроме поля ключа каждый узел двоичного дерева имеет поля `left`, `right` и `parent` которые содержат указатели на левый и правый дочерние узлы и на родительский узел соответственно.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024 × 768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19², клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Разработайте ДДП предназначенное для обслуживания типа данных T (см. варианты к заданию). Программа должна позволять выполнять

следующие операции: создание дерева; поиск вершины по ключу; определение вершин с минимальным и максимальным значениям ключа; обход дерева; переход к последующей или предыдущей вершине в порядке, определяемом ключами; вставка вершины; удаление вершины; проверка пустоты дерева.

Варианты к заданию

1. $T = \text{char}$.
2. $T = \text{int}$.
3. $T = \text{short int}$.
4. $T = \text{float}$.
5. $T = \text{double}$.
6. $T = \text{unsigned int}$.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. При выполнении задания не используйте приемы объектно-ориентированного программирования.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Почему ДДП называется деревом поиска?
2. Какие виды обхода ДДП вам известны?
3. Какие особенности удаления узла имеются у ДДП?

4. Приведите примеры практического использования ДДП?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [1,2,4];

дополнительная литература – [1, 3].

Лабораторная работа 10

Поиск данных по образцу

Цель работы: исследование методов поиска в упорядоченных и неупорядоченных последовательностях.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Двоичный (бинарный) поиск элемента в массиве. Если у нас есть массив, содержащий упорядоченную последовательность данных, то очень эффективен двоичный поиск.

Допустим, что переменные L и R содержат, соответственно, левую и правую границы отрезка массива, где находится нужный нам элемент. Мы начинаем всегда с исследования среднего элемента отрезка. Если искомое значение меньше среднего элемента, мы переходим к поиску в верхней половине отрезка, где все элементы меньше только что проверенного. Другими словами, значением R становится $(M-1)$ и на следующей итерации

мы работаем с половиной массива. Таким образом, в результате каждой проверки мы вдвое сужаем область поиска. Так, в нашем примере, после первой итерации область поиска – всего лишь три элемента, после второй остается всего лишь один элемент. Таким образом, если длина массива равна 6, нам достаточно трех итераций, чтобы найти нужное число.

Двоичный поиск – очень мощный метод. Если, например, длина массива равна 1023, после первого сравнения область сужается до 511 элементов, а после второй – до 255. Легко посчитать, что для поиска в массиве из 1023 элементов достаточно 10 сравнений.

Интерполяционный поиск элемента в массиве. Представьте себе, что вы ищете слово в словаре. Маловероятно, что вы сначала загляните в середину словаря, затем отступите от начала на $1/4$ или $3/4$ и т.д., как в бинарном поиске.

Если нужное слово начинается с буквы 'А', вы, наверное, начнете поиск где-то в начале словаря. Когда найдена отправная точка для поиска, ваши дальнейшие действия мало похожи на рассмотренный выше метод двоичного поиска.

Если вы заметите, что искомое слово должно находиться гораздо дальше открытой страницы, вы пропустите порядочное их количество, прежде чем сделать новую попытку. Это в корне отличается от алгоритма двоичного поиска, не делающего разницы между «много больше» и «чуть больше».

Мы приходим к алгоритму, называемому интерполяционным поиском. Если известно, что K лежит между K_l и K_u , то следующую пробу делаем на расстоянии $(u - 1)(K - K_l)/(K_u - K_l)$ от 1, предполагая, что ключи являются числами, возрастающими приблизительно в арифметической прогрессии.

На практике, интерполяционный поиск часто быстрее бинарного, так как с вычислительной стороны их отличают лишь применяемые арифметические операции: интерполирование – в интерполирующем поиске

и деление на два – в двоичном, а скорость их вычисления отличается незначительно. С другой стороны, интерполирующий поиск использует такое принципиальное свойство данных, как однородность распределения значений. Бинарный поиск учитывает лишь знак разности между ключом и текущим значением, а интерполирующий еще учитывает и модуль этой разности и по данному значению производит предсказание позиции следующего элемента для проверки. В среднем, интерполирующий поиск производит $\log(\log(N))$ операций, где N есть число элементов, среди которых производится поиск. Число необходимых операций зависит от равномерности распределения значений среди элементов. В плохом случае (например, когда значения элементов экспоненциально возрастают) интерполяционный поиск может потребовать до $O(N)$ операций.

Линейный алгоритм. Простейшую процедуру поиска подстроки в неупорядоченной текстовой строке можно интерпретировать графически как скольжение шаблона с искомой подстрокой по тексту. На рисунке 10.1 проиллюстрированы три последовательных положения шаблона с образцом искомого текста «ААВ» относительно текста «АСААВС».

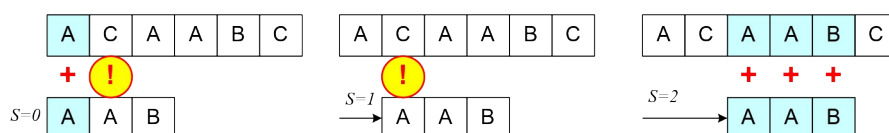


Рисунок 10.1 – Простейший алгоритм поиска подстроки для образца «ААВ» и текста «АСААВС»

Основной операцией простейшего алгоритма выступает сравнение символов, поэтому именно число сравнений и следует подсчитывать. В наихудшем случае при каждом проходе совпадают все символы за исключением последнего. Если длина подстроки равна S , а длина текста равна T , то, в наихудшем случае число сравнений будет равно $S(T - S + 1)$.

Проблема стандартного алгоритма заключается в том, что он затрачивает много усилий впустую.

Алгоритм Кнута-Морриса-Пратта. Алгоритм, изобретённый авторами приблизительно в 1970 году, требует порядка $N + M$ сравнений символов даже в самом плохом случае. Основной принцип алгоритма – не уничтожать ценную информацию предыдущего сравнения. После частичного совпадения начальной части образа с символами строки при каждом несовпадении двух символов образец сдвигается на максимально возможное количество символов $d \geq 1$ и сравнение строки с образцом (с начала образца) продолжается.

Алгоритм Кнута-Морриса-Пратта (КМП) дает выигрыш только тогда, когда неудаче предшествовало некоторое число совпадений. Лишь в этом случае образец сдвигается более чем на одну позицию.

Символ * означает любое количество любых символов. Величина $d[j]$ равна размеру самой длинной последовательности символов образца, непосредственно предшествующей j -му символу, которая совпадает с началом образца (для $j = 0$ принимают $d[j] = -1$). Величина $d[j]$ зависит только от образца и может быть вычислена заранее перед операцией поиска (см. пример в таблице 10.1).

Таблица 10.1 – Вычисление значений $d[j]$ для образца $P =$ «АБРАКАДАБРА»

Позиция несовпадающего символа – j	Совпавшая часть	Значение $d[j]$
0	-	-1
1	А	0
2	АБ	0
3	АБР	0
4	АБРА	1
5	АБРАК	0
6	АБРАКА	1
7	АБРАКАД	0
8	АБРАКАДА	1
9	АБРАКАДАБ	2
10	АБРАКАДАБР	3

Простейший вариант **алгоритма Бойера-Мура** состоит из следующих шагов:

1. Строим таблицу смещений для искомого образца.
2. Совмещаем начало строки и образца и начинаем проверку с последнего символа образца.
3. Если последний символ образца и соответствующий ему при наложении символ строки не совпадают, образец сдвигается относительно строки на величину, полученную из таблицы смещений, и снова проводится сравнение, начиная с последнего символа образца.
4. Если же символы совпадают, производится сравнение предпоследнего символа образца и т.д. Если все символы образца совпали с наложенными символами строки, значит мы нашли подстроку и поиск окончен.
5. Если же какой-то (не последний) символ образца не совпадает с соответствующим символом строки, мы сдвигаем образец на один символ вправо и снова начинаем проверку с последнего символа.

Правила построения таблицы смещений. Величина сдвига в случае несовпадения последнего символа вычисляется исходя из следующих соображений:

1. Сдвиг образца должен быть минимальным, таким, чтобы не пропустить вхождение образца в строке.
2. Если данный символ строки встречается в образце, мы смещаем образец таким образом, чтобы символ строки совпал с самым правым вхождением этого символа в образце.
3. Если же образец вообще не содержит этого символа, мы сдвигаем образец на величину, равную его длине, так что первый символ образца накладывается на следующий за проверявшимся символом строки.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

На основе рассмотренных на лекции и в текущей лабораторной работе алгоритмов разработайте программы, реализующие функционал согласно вариантам к заданию.

Варианты к заданию

1. Линейный алгоритм поиска (все варианты).
2. Поиск в упорядоченных последовательностях:
 - a. Двоичный поиск (вариант 1).
 - b. Интерполяционный поиск (вариант 2).
3. Поиск в неупорядоченных последовательностях:
 - a. Алгоритм Кнута-Морриса-Пратта (вариант 1).
 - b. Алгоритм Бойера-Мура (вариант 2).

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. Осуществите математическую оценку разработанных алгоритмов.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Дайте классификацию алгоритмов поиска.
2. Какой из алгоритмов поиска на упорядоченных последовательностях нашел применение в алгоритмах сортировки массивов?
3. За счет чего достигается преимущество у алгоритма Бойера-Мура?
4. Какой основной принцип алгоритма поиска Кнута-Морриса-Пратта?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [1,2, 4];

дополнительная литература – [1, 3].

Лабораторная работа 11

Красно-черные деревья

Цель работы: исследование способов реализации красно-черных деревьев.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма.

Теоретическая часть

Красно-черное дерево (red-black tree, КЧД) – это самобалансирующееся двоичное дерево поиска (ДДП), вершины которого разделены на красные и черные. Таким образом, каждая вершина хранит один дополнительный бит.

Пример определения узла КЧД предложен в листинге 11.1.

Листинг 11.1 – Определение узла красно-черного дерева

```
enum RBTCOLOR { Black, Red };
template<class KeyType>
struct RBTreeNode{
    KeyType key;
    RBTCOLOR color;
    RBTreeNode<KeyType> * left;
    RBTreeNode<KeyType> * right;
```

```

RBTNode<KeyType> * parent;
RBTNode(KeyType k, RBTCOLOR c, RBTNode* p, RBTNode*l,
RBTNode*r) : key(k), color(c), parent(p), left(l), right(r) { };
};

```

Двоичное дерево поиска называется красно-черным деревом, если оно обладает следующими свойствами (red-black properties):

1. Каждый узел – либо красный, либо черный.
2. Корень дерева является черным.
3. Каждый лист (NULL) – черный.
4. Если узел красный, то оба его дочерних узла черные.
5. Для каждого узла все пути от него до листьев, являющихся потомками данного узла, содержат одно и то же количество черных узлов.

Иллюстрация красно-черного дерева представлена на рисунке 11.1. Красно-черные деревья являются наиболее активно используемыми на практике самобалансирующимися деревьями поиска. В частности, ассоциативные контейнеры библиотеки STL (map, set, multiset, multimap) основаны на красно-черных деревьях.

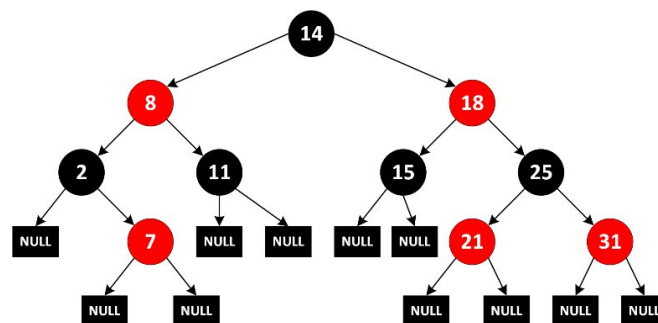


Рисунок 11.1 – Красно-черное дерево

Самое главное преимущество красно-черных деревьев в том, что при вставке выполняется не более $O(1)$ вращений. Процедуру балансировки практически всегда можно выполнять параллельно с процедурами поиска, так как алгоритм поиска не зависит от атрибута цвета узлов. Используется всего 1 бит дополнительной памяти для хранения цвета вершины.

Оборудование и материалы. Для выполнения лабораторной работы необходим компьютер со следующими характеристиками: процессор 32-битный или 64-битный с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа с разрешением 1024×768 (или выше) с диагональю не менее $19''$, клавиатура, мышь.

Программное обеспечение – операционная система Windows или Linux, среда программирования.

Задание

Разработайте красно-черное дерево, предназначенное для обслуживания типа данных T (см. варианты к заданию). Программа должна выполнять следующие операции: создание дерева; поиск вершины по ключу; определение вершин с минимальным и максимальным значениями ключа; обход дерева; переход к последующей или предыдущей вершине в порядке, определяемом ключами; вставка вершины; удаление вершины; проверка пустоты дерева; балансировка дерева.

Варианты к заданию

1. T = char.
2. T = int.
3. T = short int.
4. T = float.
5. T = double.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. Осуществите математическую оценку разработанных алгоритмов.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Какими свойствами должно обладать красно-черное дерево?
2. Какие поля должны входить в состав вершины красно-черного дерева?
3. Какие преимущества есть у красно-черного дерева над ДДП?
4. Для чего необходима процедура балансировки?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [1 – 4];

дополнительная литература – [1, 3].

Лабораторная работа 12

Представление графов в памяти

Цель работы: исследование способов представления графов в памяти.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Граф – это сложная нелинейная многосвязная динамическая структура, отображающая свойства и связи сложного объекта.

Самым привычным примером графа служит карта автодорог, на которой изображены перекрестки и связывающие их дороги. Перекрестки являются вершинами графа, а дороги – его ребрами. Иногда наши графы ориентированы (подобно улицам с односторонним движением) или взвешены – каждой дороге приписана стоимость путешествия по ней (если, например, дороги платные).

Как многосвязная структура граф обладает следующими свойствами:

1. На каждый элемент (узел, вершину) может быть произвольное количество ссылок.
2. Каждый элемент может иметь связь с любым количеством других элементов.
3. Каждая связка (ребро, дуга) может иметь направление и вес.

Обычно информацию о графах хранят двумя способами: в виде матрицы примыканий (смежностей) или в виде списка примыканий (смежностей).

Матрица примыканий графа $G = (V, E)$ с числом вершин записывается в виде двумерного массива размером $N \times N$. В каждой ячейке $[i, j]$ этого массива записано значение 0 за исключением лишь тех случаев, когда из вершины V_i в вершину V_j ведет ребро, и тогда в ячейке записано значение:

$$AdjMat[i, j] = \begin{cases} 1, & \text{если } V_i V_j \in E, \\ 0, & \text{если } V_i V_j \notin E. \end{cases} \quad (12.1)$$

Ячейка матрицы примыканий взвешенного графа или орграфа содержит бесконечность, если соответствующее ребро отсутствует, а во всех остальных случаях ее значение равно весу ребра. Диагональные элементы такой матрицы равны 0, поскольку путешествие из вершины в нее саму не стоит ничего (рис.12.1, а).

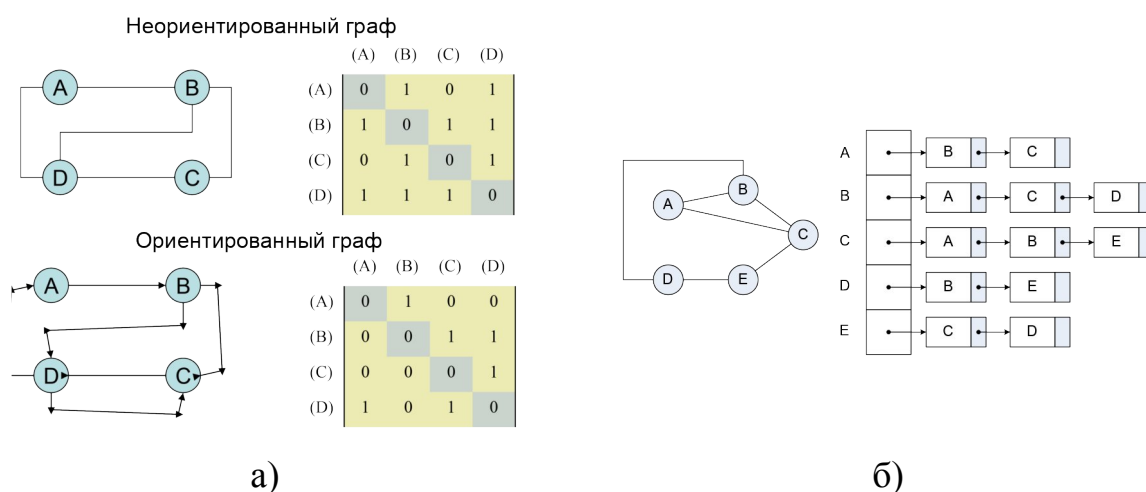


Рисунок 12.1 – Представление графа: а) в виде матрицы примыканий; б) в виде списка примыканий

Список примыканий графа $G = (V, E)$ с числом вершин $|V| = N$ записывается в виде одномерного массива длины N , каждый элемент которого представляет собой ссылку на список (рис. 12.1, б).

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Разработайте программу, позволяющую хранить информацию (матрица смежностей или список смежностей) об ориентированном графе произвольного размера. Программа должна содержать функции:

1. Добавления вершины в граф.
2. Удаления вершины из графа.
3. Вывода информации о составе графа.
4. Сохранения данных в файл и чтения данных из файла.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода.

Разработанная на данном занятии программа будет востребована в дальнейших работах, посвященных графам.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Каким образом можно классифицировать графы?
2. Как граф можно представить в памяти компьютера?
3. В каких случаях следует хранить граф в виде матрицы смежности?
4. В каких случаях следует хранить граф в виде списка смежности?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [1, 2];

дополнительная литература – [1, 3].

Лабораторная работа 13

Алгоритмы обхода графа

Цель работы: исследование алгоритмов обхода графа.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

При работе с графами часто приходится выполнять некоторое действие по одному разу с каждой из вершин графа. Например, некоторую порцию информации следует передать каждому из компьютеров в сети. При этом мы не хотим посещать какой-либо компьютер дважды. Подобный обход можно совершать двумя различными способами: **обходом в глубину** и **обходом по уровням**.

Алгоритм обхода в глубину. При обходе в глубину мы посещаем первый узел, а затем идем вдоль ребер графа, пока не упремся в тупик. После попадания в тупик мы возвращаемся назад вдоль пройденного пути пока не

обнаружим вершину, у которой есть еще не посещенный сосед, а затем двигаемся в этом новом направлении (рис. 13.1).

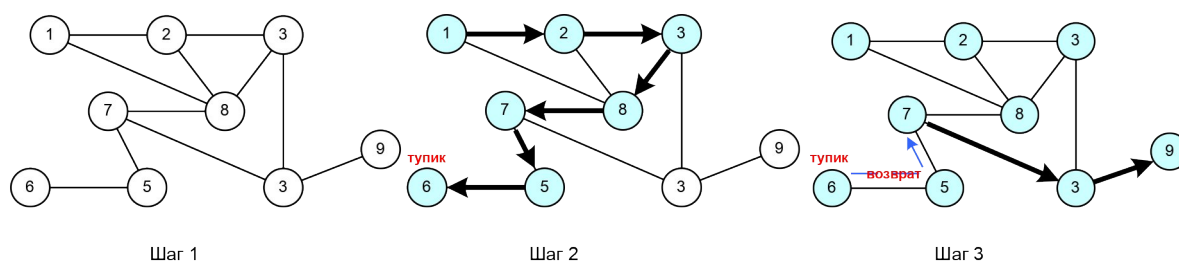


Рисунок 13.1 – Иллюстрация алгоритма обхода в глубину

Алгоритм обхода по уровням. Алгоритм обхода по уровням работает следующим образом (рис. 13.2):

1. При обходе графа по уровням мы, после посещения первого узла, посещаем все соседние с ним вершины.
2. При втором проходе посещаются все вершины на расстоянии двух ребер от начальной.
3. При каждом новом проходе обходятся вершины, расстояние от которых до начальной на единицу больше предыдущего.

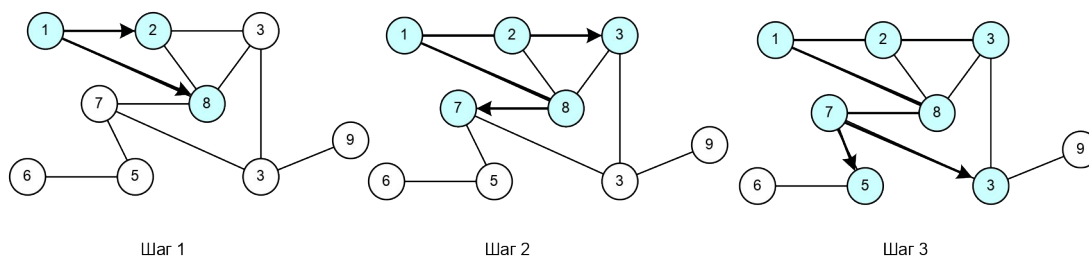


Рисунок 13.2 – Иллюстрация алгоритма обхода по уровням

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор

типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Доработайте приложение для работы с графами (лаб. работа №12), внедрив в него функцию для осуществления обхода графа с использованием алгоритмов:

1. Алгоритм обхода в глубину.
2. Алгоритм обхода по уровням.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Где на практике используются алгоритмы обхода графа?
2. Какой способ представления графа в памяти удобнее применять для алгоритма обхода в глубину?
3. Какой способ представления графа в памяти удобнее применять для алгоритма обхода по уровням?
4. Сравните эффективность алгоритмов обхода графа?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [1 – 4];

дополнительная литература – [1- 3].

Лабораторная работа 14

Минимальное остовное дерево графа

Цель работы: исследование алгоритмов построения минимального остовного дерева графа.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Минимальным остовным деревом (МОД) связного взвешенного графа называется его связный подграф, состоящий из всех вершин исходного дерева и некоторых его ребер, причем сумма весов ребер минимально возможная. В конце 1950-х годов Эдгар Дейкстра и Прим, работая и публикуя свои результаты независимо друг от друга, предложили следующий алгоритм построения МОД. Разобьем вершины графа на три класса:

- ~ вершины, вошедшие в уже построенную часть дерева;
- ~ вершины, окаймляющие построенную часть;
- ~ еще не рассмотренные вершины.

Начнем с произвольной вершины графа и включим ее в остовное дерево (рис. 14.1). Все вершины, соединенные с исходной вершиной, заносим в кайму. Затем выполняется цикл поиска ребра с наименьшим весом, соединяющего уже построенную часть остовного дерева с каймой. Это ребро вместе с новой вершиной добавляется в дерево и происходит обновление каймы. После того, как в дерево попадут все вершины, работа будет закончена.

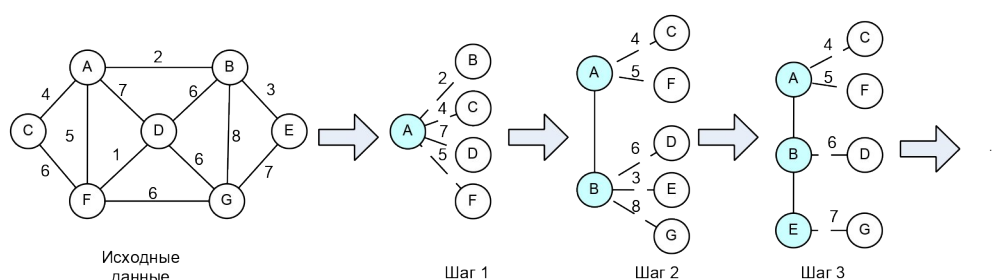


Рисунок 14.1 – Алгоритм построения МОД Дейкстры-Прима

Алгоритм Дейкстры-Прима начинает построение МОД с конкретной вершины графа и постепенно расширяет построенную часть дерева. В отличие от него алгоритм Крускала делает упор на ребрах графа. В этом алгоритме мы начинаем с пустого дерева и добавляем к нему ребра в порядке возрастания их весов пока не получим набор ребер, объединяющий все вершины графа.

Если ребра закончатся до того, как все вершины будут соединены между собой, то это означает, что исходный граф был несвязным, и полученный нами результат представляет собой объединение МОД всех его компонент связности.

На рисунке 14.2 представлены шаги алгоритма Крускала, создающего МОД для тех же исходных данных, что и в предыдущем примере.

Работа алгоритма Борувки состоит из нескольких итераций, каждая из которых состоит в последовательном добавлении рёбер к остовному лесу графа, до тех пор, пока лес не превратится в дерево, то есть, лес, состоящий

из одной компоненты связности. На каждом этапе мы отыскиваем наиболее короткое ребро, которое соединяет каждое поддерево с некоторым другим поддеревом (рис. 14.3).

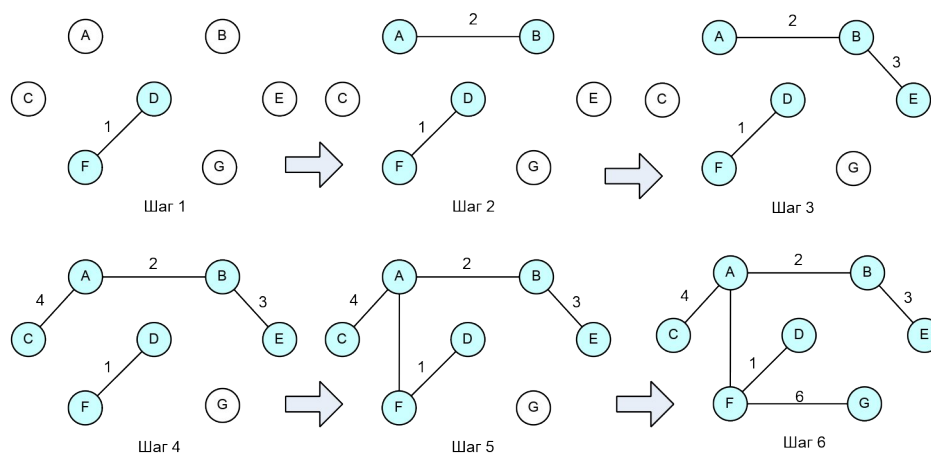


Рисунок 14.2 – Иллюстрация алгоритма Крускала

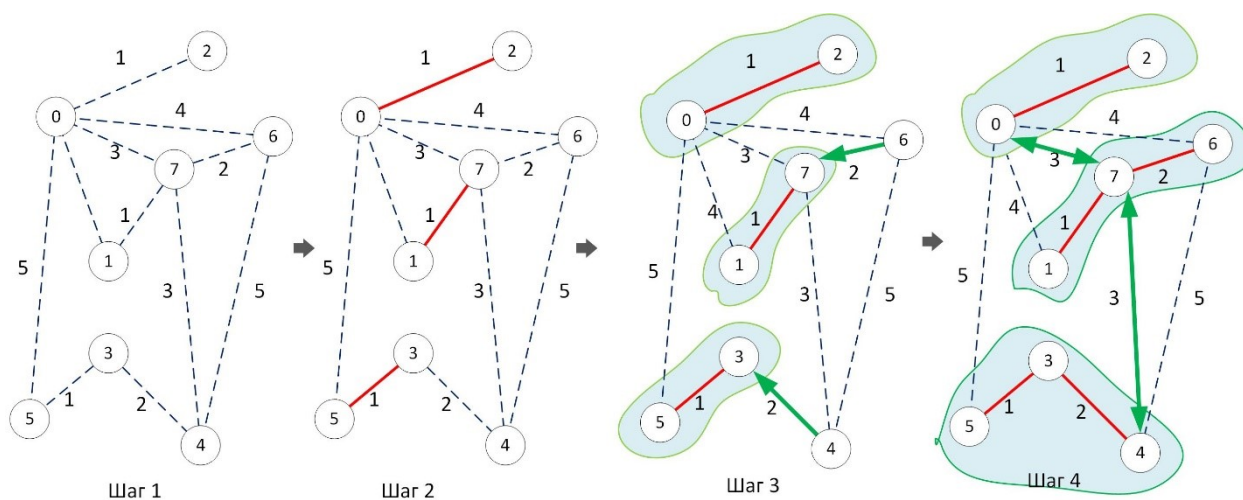


Рисунок 14.3 – Иллюстрация алгоритма Борувки

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или

больше) цветами с диагональю не менее 19^2 , клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Доработайте приложение для работы с графами (лаб. работа №12), внедрив в него функцию для создания минимального остовного дерева с использованием алгоритма согласно вашему варианту.

Варианты к заданию

1. Алгоритм Дейкстры-Прима.
2. Алгоритм Крускала.
3. Алгоритм Борувки.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Что понимается под минимальным остовным деревом связного графа?
2. Почему алгоритм Дейкстры-Прима считается «жадным»?
3. Какие разновидности остовных деревьев вам известны?
4. Сравните эффективность алгоритмов МОД?

5. Почему алгоритмы построения МОД не пригодны для поиска кратчайшего пути в графе?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [1,2, 4];

дополнительная литература – [1, 3].

Лабораторная работа 15

Поиск кратчайшего пути в графе

Цель работы: исследование алгоритмов поиска кратчайшего пути в графе.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Результатом алгоритма поиска кратчайшего пути является последовательность ребер, соединяющая две заданные вершины и имеющая наименьшую длину среди всех таких последовательностей. Если изменить рассмотренный ранее алгоритм Дейкстры-Прима построения МОД так, чтобы при выборе ребра, ведущего в кайму, он выбирал ребро, являющееся частью кратчайшего в целом пути из начальной вершины, то мы получим требуемый результат. На этой идее и основан алгоритм Дейкстры.

Допустим, что нам требуется построить кратчайший путь из вершины А в вершину G (рис. 15.1). На каждой итерации алгоритма Дейкстры в кайму

добавляется ребро с наименьшим весом относительно стартовой вершины. Так как наш алгоритм начинает работать из вершины А, то на первом шаге в кайму добавляется ребро, соединяющее узел А с узлом В, так как вес ребра самый малый. На втором шаге мы анализируем веса рёбер, исходящих уже из двух этих вершин. «Победителем» становится ребро с весом 4, оно соединяет вершины А и С. Ребро В-Е будет добавлено в кайму только на третьем этапе, заметьте, что при включении этого ребра в кайму следует учитывать суммарный вес пути $A-B-E=2+3=5$.

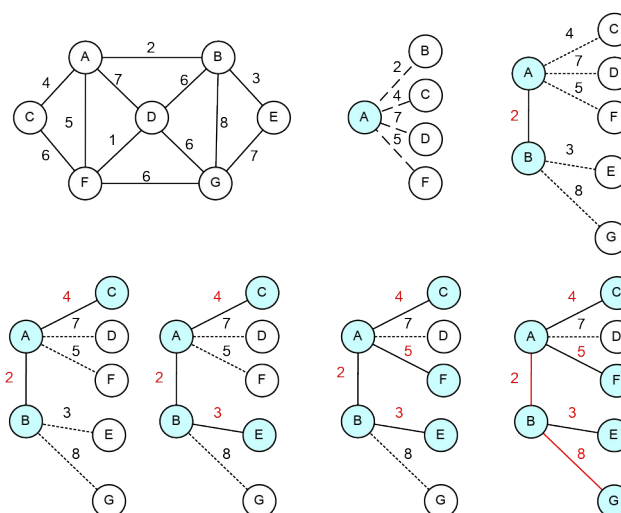


Рисунок 15.1 – Иллюстрация алгоритма Дейкстры при прокладке кратчайшего пути из А в G

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19^2 , клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Доработайте приложение для работы с графами (лаб. работа № 12), внедрив в него функцию для осуществления поиска кратчайшего пути с использованием алгоритма согласно вашему варианту.

Варианты к заданию

1. Алгоритм Дейкстры.
2. Алгоритм Флойда.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Раскройте принцип способа поиска кратчайшего пути в графе с помощью построения дерева решений?
2. Каким образом можно оценить эффективность алгоритма поиска кратчайшего пути Дейкстры?
3. Почему алгоритм Дейкстры может оказаться неэффективным в графах с отрицательными весами?
4. В чем заключается основная идея алгоритма Флойда?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для

пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [1,2,4];

дополнительная литература – [1, 3].

Лабораторная работа 16

Алгоритмы хеширования

Цель работы: исследование методов хеширования.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Хеширование (*hashing*) – преобразование входного массива данных произвольной длины в выходную битовую строку фиксированной длины. Такие преобразования обеспечиваются с помощью хеш-функций, а их результаты называют хеш-кодом. Хеш-таблицы являются ближайшими родственниками обычных таблиц. Таблица превращается в хэш-таблицу, когда к заданному элементу осуществляется непоследовательный доступ с помощью хеш-функции. В хеш-функцию, в общем, передается элемент данных, который она преобразовывается в целое число, которое используется как индекс в таблице. Например, вы имеете таблицу, в которую вы намерены сохранять значения ИНН персонала фирмы, но вместо истинных значений вы

разделите ИНН на какое-то число (допустим 3) и сохраните в таблице результат деления.

В случае прямой адресации элемент с ключом k хранится в ячейке k . При хешировании этот элемент хранится в ячейке $h(k)$, т.е. мы используем хеш-функцию h для вычисления ячейки для данного ключа k . Само собой разумеется, функция h должна быть детерминистической и для одного и того же значения и всегда давать одно и то же хеш-значение $h(k)$. Однако поскольку $|U| > m$, должно существовать как минимум два ключа, которые имеют одинаковое хеш-значение. Таким образом, полностью избежать коллизий невозможно в принципе, и хорошая хеш-функция в состоянии только минимизировать количество коллизий.

Идеальная функция хеширования должна обладать следующими свойствами:

1. Быть применимой к аргументу любого размера.
2. Выходное хеш-значение должно иметь фиксированный размер.
3. Скорость вычисления должна быть такой, чтобы опережать время, затрачиваемое на обработку исходного значения.
4. Хеш-функция должна быть чувствительной к всевозможным изменениям во входных данных.
5. Хеш-функция должна быть односторонней, то есть обладать свойствами необратимости.
6. Вероятность того, что значения хеш-функций двух различных входных данных совпадут, должна быть ничтожно мала.

Наиболее простейший метод хеширования заключается в выборе в качестве размера M хеш-таблицы простого числа и вычисление остатка от деления исходного значения k на M :

$$h(k) = k \bmod M. \quad (16.1).$$

Такая функция называется модульной хеш-функцией. Модульные хеш-функции хорошо подходят для хеширования целых чисел.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный с тактовой частотой 1,5 ГГц или выше, оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 с диагональю не менее $19''$, клавиатура, мышь. **Программное обеспечение** – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Создайте текстовый файл, содержащий 50-60 строк случайных латинских символов (от 1 до 20-ти символов в строке). Указанный файл будет служить источником данных для вашей дальнейшей работы.

1. Разработайте хеш-функцию и структуру хеш-таблицы для работы с текстовыми данными.
2. Реализуйте операции построчного чтения текстового файла, хеширования строк файла и заполнения значений в хеш-таблицу.
3. Реализуйте операции поиска данных в хеш-таблице следующим образом:
 - ~ пользователь вводит произвольную строку латинских символов;
 - ~ по нажатию клавиши ввод программа преобразует запрашиваемые данные в хеш-значение;
 - ~ в хеш-таблице находим совпадающее хеш-значение и узнаём номер строки в файле input.txt.
4. Внесите предложения по борьбе с коллизиями.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. В чем основной недостаток таблиц с прямой адресацией?
2. Какими свойствами должна обладать хеш-функция?
3. Какие современные функции хеширования вам известны?
4. В чем причина возникновения коллизий?
5. Как можно устранять последствия коллизий?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [2,4];
дополнительная литература – [1].

Лабораторная работа 17

Псевдослучайные последовательности

Цель работы: исследование методов генерации псевдослучайных значений, оценка качества ПСП.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Генератор псевдослучайных чисел (ГПСЧ) – алгоритм, генерирующий последовательность чисел, элементы которой почти независимы друг от друга и подчиняются заданному распределению (обычно равномерному).

Равномерным распределением на конечном множестве чисел называется такое распределение, при котором любое из возможных чисел имеет одинаковую вероятность появления.

Метод середин квадратов предложен Джоном фон Нейманом в 1946 году. Его идея заключалась в следующем:

1. Берём N -значное число, например, 5772156649.
2. Возводим число в квадрат: 33317792380594909201.
3. Из результата ($2N$ -значного числа) берём N средних цифр: 7923805949.
4. Возвращаемся к 1-му шагу.

Линейный конгруэнтный метод предложен Д.Г. Лехмером (D.H. Lehmer) в 1949 году. Он основан на математическом выражении:

$$X_{n+1} = (aX_n + c) \bmod m, \quad (17.1)$$

где: m – модуль $0 < m$; a – множитель $0 \leq a < m$; c – приращение $0 \leq c < m$; в момент запуска $X_n = X_0$; X_0 – начальное значение $0 \leq X_0 < m$.

Конгруэнтная последовательность всегда образует петли, т.е. обязательно существует цикл, который будет повторяться бесконечное количество раз. Поэтому для получения большого периода повторения m должно быть достаточно большим (табл. 17.1).

Таблица 17.1 – Рекомендуемые параметры линейного конгруэнтного генератора

(a, c, m)	(a, c, m)	(a, c, m)
(106,1283,6075)	(625,6571,31104)	(1277,24749,117128)
(211,1663,7875)	(1541,2957,14000)	(2041,25673,121500)
(421,1663,7875)	(1741,2731,12960)	(2311,25367,120050)
(430,2531,11979)	(1291,4621,21870)	(1597,51749,244944)
(936,1399,6655)	(205,29573,139968)	(2661,36979,175000)

Пример ПСП последовательности, сгенерированной в Mathcad, представлен на рисунке 17.1, на котором видно, что период линейной конгруэнтной последовательности с параметрами $a = 11$, $c = 67$ и $m = 255$ невелик. Поэтому такая ПСП вряд ли подойдёт для применения в криптографических системах.

Качественный генератор случайных чисел характеризуется тремя свойствами:

1. Равномерным распределением случайных чисел.
2. Непредсказуемостью значений.
3. Замкнутостью цикла (способностью генерировать весь перечень значений из заданного подмножества).

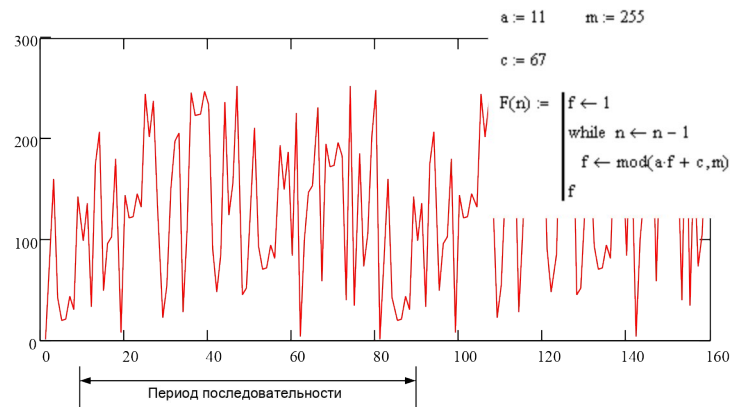


Рисунок 17.1 – Пример ПСП последовательности, сгенерированной в Mathcad

При оценке качества ПСП различают два вида критериев: эмпирические критерии, при которых компьютер манипулирует группами чисел последовательности и вычисляет определённые статистики, и теоретические критерии, для которых характеристики последовательности определяются с помощью теоретико-числовых методов, основанных на рекуррентных правилах, которые используются для образования последовательности.

Один из наиболее эффективных способов оценки последовательности на «качество случайности» основан на критерии «Хи-квадрат» – χ^2 . Однако в этой лабораторной работе вы проведёте анализ ПСП с помощью трёх эмпирических критериев:

- ~ тест на длительность периода повторения последовательности;
- ~ критерий равномерности (критерий частот);
- ~ тест на пропуски.

Критерий равномерности проверяет тот факт, что ПСП распределяет генерируемые значения по всему заданному диапазону.

Тест на пропуски гарантирует, что последовательность случайных чисел не будет сначала попадать в один поддиапазон, затем во второй, затем в третий несмотря на то, что в целом числа будут равномерно распределены по поддиапазону.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19², клавиатура, мышь. **Программное обеспечение** – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

1. Подготовьте программную реализацию трех различных генераторов псевдослучайных последовательностей (системного генератора, генератора на основе метода середин квадратов и линейного конгруэнтного метода).
2. Оцените качество разработанных генераторов ПСП по критериям: длительности периода повторения последовательности, равномерности и теста на пропуски.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Какие числовые характеристики случайных величин вам известны?
2. Где применяются случайные числа?
3. Опишите структуру генератора псевдослучайных чисел?
4. Какими чертами должен обладать генератор ПСП?
5. Как можно оценить качество генератора ПСП?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [4];

дополнительная литература – [1-3].

Лабораторная работа 18

Методы сжатия данных, алгоритм LZW

Цель работы: исследование методов сокращения избыточности текстовых данных.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Алгоритмы сжатия могут повышать эффективность хранения и передачи данных посредством сокращения их избыточности.

Избыточность в представлении строки S есть $L(S) - H(S)$, где $L(S)$ – это длина представления в битах, а $H(S)$ – энтропия, т.е. мера содержания информации, также выраженная в битах. Алгоритмов, которые могли бы без потери информации сжать строку к меньшему числу бит, чем составляет ее энтропия, не существует.

Идея словарных методов заключается в замене строк символов на такие коды, что их можно трактовать как индексы какого-то словаря.

Словарь – это **набор** таких **фраз**, которые как мы полагаем будут **встречаться** в обрабатываемой последовательности.

Индексы должны быть построены таким образом, чтобы в среднем их представление составляло **меньше места**, чем требуют замещаемые строки.

Классический представитель словарных методов сжатия данных – алгоритм LZW:

1. LZW отказывается от 8-мибитного кодирования и, например, переходит к 10-битному ($2^{10}=1024$). Значения кодов 0 - 255 соответствуют отдельным байтам, а коды 256-1024 соответствуют подстрокам.

2. В процессе кодирования LZW строит таблицу соответствия (словарь), содержащий следующие пары: сочетание символов $\rightarrow$ числовое значение.

3. Алгоритм LZW приступает к анализу исходной строки преобразуя последовательность сочетаний символов в коды подстрок. Полученные коды заносятся в таблицу соответствия.

Рассмотрим работу алгоритма на примере. Входная строка является кратким списком слов, разделенных символом «/». Указанная строка состоит из 19 символов, что соответствует $19 \cdot 8 = 152$ битам.

Работа начинается с того, что на первом шаге цикла алгоритм выполняет проверку на наличие строки «/W» в таблице словаря (табл. 18.1). Когда он не находит эту строку, то генерирует код для «/» и добавляет в таблицу строку «/W». Так как 256 символов уже определены для кодов 0-255, то первой определенной строке может быть поставлен в соответствие код 256. После этого система читает следующую букву («E»), добавляет вторую подстроку («WE») в таблицу и выводит код для буквы «W».

В результате работы алгоритма мы получаем строку из 12 символов, каждый из которых занимает 10 бит. Эффект от сжатия данных налицо, исходная строка занимала 152 бита, а сжатая – 120 бит. В результате мы получили выигрыш, превышающий 20%.

Таблица 18.1 – Построение словаря для последовательности

Вход (символы)	Выход (коды)	Новые коды и соответствующие строки
/W	/	256 = /W
E	W	257 = WE
D	E	258 = ED
/	D	259 = D/
WE	256	260 = /WE
/	E	261 = E/
WEE	260	262 = /WEE
/W	261	263 = E/W
EB	257	264 = WEB
/	B	265 = B/
WET	260	266 = /WET
<EOF>	T	

Теперь познакомимся с алгоритмом распаковки. Алгоритм распаковки получает выходной поток кодов от алгоритма сжатия и использует его для точного восстановления входного потока (табл. 18.2).

Таблица 18.2 – Алгоритм распаковки

Вход	Старый код	Выход	Символ	Новый вход таблицы
/	/	/		
W	/	W	W	256 = /W
E	W	E	E	257 = WE
D	E	D	D	258 = ED
256	D	/W	/	259 = D/
E	256	E	E	260 = /WE
260	E	/WE	/	261 = E/
261	260	E/	E	262 = /WEE
257	261	WE	W	263 = E/W
B	257	B	B	264 = WEB
260	B	/WE	/	265 = B/
T	260	T	T	266 = /WET

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Используя идею алгоритма «LZW» разработайте функции, кодирующие и декодирующие произвольную текстовую строку. Программа должна позволять выбирать размерность кодирования (в пределах 10-16 бит).

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Что такое энтропия информации?

2. Существуют ли алгоритмы, которые могли бы без потери информации сжать строку к меньшему числу бит, чем составляет ее энтропия?
3. В чем заключается идея словарных методов сжатия?
4. Приведите примеры современных программных продуктов, основанных на словарном методе сжатия?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

- основная литература – [1 – 4];
дополнительная литература – [1, 3].

Лабораторная работа 19

Алгоритм Хаффмана

Цель работы: исследование алгоритма оптимального префиксного кодирования алфавита с минимальной избыточностью.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Алгоритм Хаффмана (англ. Huffman) – жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью. Разработан в 1952 году доктором Массачусетского технологического института Дэвидом Хаффманом.

Этот метод кодирования состоит из двух основных этапов:

1. Построение оптимального кодового дерева.
2. Построение отображения код-символ на основе построенного дерева.

1 этап. Допустим, что мы имеем файл длиной в 100 байт и имеющий 6 различных символов в себе. Мы подсчитали вхождение каждого из символов в файл и получили следующий результат, представленный в таблице 19.1.

Таблица 19.1 – Результат анализа количества вхождений символов

Символ	A	B	C	D	E	F
Число вхождений	10	20	30	5	25	10

Выписываем символы первичного алфавита m_1 в порядке убывания вероятностей, как показано в таблице 19.2.

Таблица 19.2 – Результат сортировки вхождений символов

Символ	C	E	B	F	A	D
Число вхождений	30	25	20	10	10	5

Последние n_0 символов объединяют в новый символ, вероятность которого равна суммарной вероятности этих символов, удаляют эти символы и вставляют новый символ в список остальных на соответствующее место (по вероятности). n_0 вычисляется из системы:

$$\begin{cases} 2 \leq n_0 \leq m_2, \\ n_0 = m_1 - a(m_2 - 1), \end{cases} \quad (19.1)$$

где a – целое число, m_1 и m_2 – мощность первичного и вторичного алфавита соответственно.

2 этап. Каждый символ изначально представлялся 8-ю битами (один байт), и так как мы уменьшили число битов необходимых для представления каждого символа, мы, следовательно, уменьшили размер выходного файла. Таким образом:

1. Если мы будем учитывать все 256 символов, то для нас не будет разницы в сжатии текстового, EXE, графического или любого другого файла.

2. Для восстановления первоначального файла, мы должны иметь декодирующее дерево. Следовательно, мы должны сохранять дерево вместе с файлом.

3. Так как алгоритм «жадный», то нам потребуется 2 прохода (2 просмотра файла) для его реализации.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Подготовьте программную реализацию алгоритма Хаффмана. Ваша программа должна позволять:

1. Строить оптимальное кодовое дерево.
2. Строить отображение код-символ.
3. Кодировать и декодировать текстовый файл, содержащий не менее 100 символов произвольного алфавита.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Почему алгоритм Хаффмана считается «жадным»?
2. Почему восстановление закодированных данных невозможно без декодирующего дерева?
3. Возможно ли применение алгоритма Хаффмана не только к текстовым данным?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

- основная литература – [2];
дополнительная литература – [1].

Лабораторная работа 20

Алгоритм вывода отрезка

Цель работы: исследование базовых графических алгоритмов.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Вся растровая графика построена на идеи закрашивания элементарных графических объектов – пикселей. В этом случае экран компьютера можно рассматривать как двумерный массив пикселей, а операция рисования является операцией заполнения элементов этого массива.

Рассмотрим простейший способ вычисления координат закрашиваемых пикселей. Пусть заданы координаты начальной (x_1, y_1) и конечной (x_2, y_2) точек отрезка. Запишем отношения катетов для подобных треугольников:

$$\frac{x-x_1}{y-y_1} = \frac{x_2-x_1}{y_2-y_1}. \quad (20.1)$$

Тогда расчёт координат точки будет следующим:

$$x = x_1 + (y - y_1) \frac{x_2 - x_1}{y_2 - y_1}, y = y_1 + (x - x_1) \frac{y_2 - y_1}{x_2 - x_1}. \quad (20.2)$$

Положительные черты прямого вычисления координат сводятся к простоте алгоритма и возможности работы с дробными значениями координат. Недостатки рассмотренного метода более существенны. Во-первых, использование операций с плавающей точкой, операций деления и умножения обуславливают малую скорость алгоритма. Во-вторых, растровый дисплей способен работать только с целыми числами, а округление вещественных значений до целых вносит дополнительную погрешность.

Для устранения недостатков алгоритмов прямого вычисления координат придуманы инкрементные алгоритмы. Основной целью создания инкрементного алгоритма было построение таких циклов вычислений координат, чтобы в их основе лежали только целочисленные операции сложения/вычитания.

Инкрементные алгоритмы выполняются как последовательное вычисление координат соседних пикселей путём добавления приращения координат. Приращения рассчитываются на основе анализа функции погрешности.

Основная идея алгоритма Брезенхэма состоит в следующем. При черчении виртуальной прямой из точки (x_1, y_1) в точку (x_2, y_2) перед принятием решения на закрашивание очередного пикселя мы вычисляем угловой коэффициент прямой. Если угловой коэффициент прямой меньше 0.5, то естественно точку, следующую за точкой $(0,0)$, поставить в позицию $(1,0)$, а если угловой коэффициент больше 0.5, то в позицию $(1,1)$.

Для принятия решения куда заносить очередной пиксель вводится величина отклонения E точной позиции от середины между двумя возможными растровыми точками в направлении наименьшей относительной координаты. Знак E используется как критерий для выбора ближайшей растровой точки.

Чтобы реализация алгоритма Брезенхема была полной, необходимо обрабатывать отрезки во всех октантах. Модификацию легко сделать, учитывая в алгоритме номер квадранта, в котором лежит отрезок и его угловой коэффициент. Когда абсолютная величина углового коэффициента больше 1, y постоянно изменяется на единицу, а критерий ошибки Брезенхема используется для принятия решения об изменении величины x . Выбор постоянно изменяющейся (на +1 или -1) координаты зависит от квадранта.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Разработайте программу, позволяющую осуществлять графический вывод произвольного отрезка с помощью алгоритма:

1. Прямого вычисления координат.
2. Алгоритма Брезенхэма.

Сравните эффективность используемых в работе алгоритмов.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Почему графические алгоритмы как правило являются самыми ресурсоемкими?
2. В чем отличие растрового изображения от векторного?
3. Почему прямое вычисление координат считается не эффективным?
4. Опишите идею расчета точек отрезка в инкрементных алгоритмах.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [1];
дополнительная литература – [4, 5].

Лабораторная работа 21

Технологичность программного обеспечения

Цель работы: исследование особенностей проектирования технологичного ПО.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Под **технологичностью** понимают качество проекта программного продукта, от которого зависят трудовые и материальные затраты на его реализацию и последующие модификации. Хороший проект сравнительно быстро и легко кодируется, тестируется, отлаживается и модифицируется. Чем лучше проработана модель разрабатываемого программного обеспечения, тем четче определены подзадачи и структуры данных, хранящие входную, промежуточную и выходную информацию, тем проще их проектирование и реализация и меньше вероятность ошибок, для исправления которых потребуется существенно изменять программу.

Модулем называют автономно компилируемую программную единицу. Термин «модуль» традиционно используется в двух смыслах. Первоначально, когда размер программ был сравнительно невелик, и все подпрограммы компилировались отдельно, под модулем понималась подпрограмма, т.е. последовательность связанных фрагментов программы, обращение к которой выполняется по имени. Со временем, когда размер программ значительно вырос, и появилась возможность создавать библиотеки ресурсов: констант, переменных, описаний типов, классов и подпрограмм, термин «модуль» стал использоваться и в смысле автономно компилируемый набор программных ресурсов.

Практика показала, что чем выше степень независимости модулей, тем:

- ~ легче разобраться в отдельном модуле и всей программе и соответственно тестировать, отлаживать и модифицировать ее;
- ~ меньше вероятность появления новых ошибок при исправлении старых или внесении изменений в программу, т.е. вероятность появления «волнового» эффекта;
- ~ проще организовать разработку программного обеспечения группой программистов и легче его сопровождать.

Таким образом, уменьшение зависимости модулей улучшает технологичность проекта. Степень независимости модулей (как подпрограмм, так и библиотек) оценивают двумя критериями: сцеплением и связностью.

Сцепление является мерой взаимозависимости модулей, которая определяет, насколько хорошо модули отделены друг от друга. Модули независимы, если каждый из них не содержит о другом никакой информации. Чем больше информации о других модулях хранит модуль, тем больше он с ними сцеплен.

Различают пять типов сцепления модулей:

- ~ по данным;

- ~ образцу;
- ~ управлению;
- ~ общей области данных;
- ~ содержимому.

Связность модулей – мера прочности соединения функциональных и информационных объектов внутри одного модуля. Если сцепление характеризует качество отделения модулей, то связность характеризует степень взаимосвязи элементов, реализуемых одним модулем. Размещение сильно связанных элементов в одном модуле уменьшает межмодульные связи и соответственно взаимовлияние модулей. В то же время помещение сильно связанных элементов в разные модули не только усиливает межмодульные связи, но и усложняет понимание их взаимодействия. Объединение слабо связанных элементов также уменьшает технологичность модулей, так как такими элементами сложнее мысленно манипулировать.

Различают следующие виды связности (в порядке убывания уровня):

- ~ функциональную связность;
- ~ последовательную;
- ~ информационную (коммуникативную);
- ~ процедурную;
- ~ временную;
- ~ логическую;
- ~ случайную.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024 × 768 (или выше) с 24-битными (или

больше) цветами с диагональю не менее 19^2 , клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Подготовьте 2-мерный массив целых чисел произвольной размерности $N \times M$ и заполните его случайными числами. Разработайте функцию:

1. Вывода содержимого массива на экран.
2. Поиска минимального значения в массиве.
3. Поиска максимального значения в массиве.
4. Суммирования минимального и максимального значения.
5. Вычисления среднего арифметического значения.
6. Суммирования значений выше главной диагонали.
7. Суммирования значений ниже главной диагонали.
8. Суммирования значений выше главной диагонали и значений ниже главной диагонали.
9. Суммирования значений в заданном ряду массива.
10. Суммирования значений в заданном столбце массива.
11. Суммирования указанных пользователем рядов массива.
12. Суммирования указанных пользователем столбцов массива.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. При проектировании функций обеспечьте качественный уровень связности и сцепления модулей.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Дайте определение сцепления модулей.
2. Дайте определение связности модулей.
3. Почему технологичность программы зависит от характеристик сцепления и связности?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [5];
дополнительная литература – [5].

Лабораторная работа 22

Разработка технического задания

Цель работы: исследование особенностей разработки технического задания на ПО.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Техническое задание представляет собой документ, в котором сформулированы основные цели разработки, требования к программному продукту, определены сроки и этапы разработки и регламентирован процесс приемно-сдаточных испытаний. На техническое задание существует стандарт ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению». В соответствии с этим стандартом техническое задание должно содержать следующие разделы:

- ~ введение;
- ~ основания для разработки;

- ~ назначение разработки;
- ~ требования к программе или программному изделию;
- ~ требования к программной документации;
- ~ технико-экономические показатели;
- ~ стадии и этапы разработки;
- ~ порядок контроля и приемки.

Введение должно включать наименование и краткую характеристику области применения программы или программного продукта, а также объекта (например, системы), в котором предполагается их использовать. Основное назначение введения – продемонстрировать актуальность данной разработки и показать, какое место эта разработка занимает в ряду подобных.

Раздел «Основание для разработки» должен содержать наименование документа, на основании которого ведется разработка, организации, утвердившей данный документ, и наименование или условное обозначение темы разработки. Таким документом может служить план, приказ, договор и т. п.

Раздел «Назначение» должен содержать описание функционального и эксплуатационного назначения программного продукта с указанием категорий пользователей.

Раздел «Требования к программе» или программному изделию должен включать следующие подразделы:

- ~ требования к функциональным характеристикам;
- ~ требования к надежности;
- ~ условия эксплуатации;
- ~ требования к составу и параметрам технических средств;
- ~ требования к информационной и программной совместимости;
- ~ требования к маркировке и упаковке;
- ~ требования к транспортированию и хранению;
- ~ специальные требования.

Наиболее важным из перечисленных выше является подраздел «Требования к функциональным характеристикам». В этом разделе должны быть перечислены выполняемые функции и описаны состав, характеристики и формы представления исходных данных и результатов. В этом же разделе при необходимости указывают критерии эффективности: максимально допустимое время ответа системы, максимальный объем используемой оперативной и/или внешней памяти и др.

В разделе «Требования к программной документации» указывают необходимость наличия руководства программиста, руководства пользователя, руководства системного программиста, пояснительной записки и т.п. На все эти типы документов также существуют ГОСТы.

В разделе «Технико-экономические показатели» рекомендуется указывать ориентировочную экономическую эффективность, предполагаемую годовую потребность и экономические преимущества по сравнению с существующими аналогами.

В разделе «Стадии и этапы разработки» указывают стадии разработки, этапы и содержание работ с указанием сроков разработки и исполнителей.

В разделе «Порядок контроля и приемки» указывают виды испытаний и общие требования к приемке работы.

В приложениях при необходимости приводят: перечень научно-исследовательских работ, обосновывающих разработку; схемы алгоритмов, таблицы, описания, обоснования, расчеты и другие документы, которые следует использовать при разработке.

В зависимости от особенностей разрабатываемого продукта разрешается уточнять содержание разделов, т.е. использовать подразделы, вводить новые разделы или объединять их.

В случаях если какие-либо требования, предусмотренные техническим заданием, заказчик не предъявляет, следует в соответствующем месте указать «Требования не предъявляются».

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный с тактовой частотой 1,5 ГГц или выше, оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 с диагональю не менее 19", клавиатура, мышь. **Программное обеспечение** – операционная система семейства Windows или Linux.

Задание

Учебная группа студентов разделяется на 6 подгрупп из 2-3 студентов. В соответствии с вариантами задания обучаемыми формируются требования заказчика к проектируемому ПО и техническое задание на разработку ПО.

Варианты к заданию

1. Для обслуживания компьютеров вашего отдела необходимо разработать программное обеспечение, позволяющее пользователю очищать жесткие диски от неиспользуемых файлов (временные файлы, log-файлы, и т.п.). Сформулируйте ваши требования к перспективному ПО и подготовьте техническое задание.

2. Для повышения оперативности взаимодействия между сотрудниками вашего отдела необходимо приложение, позволяющее обеспечить обмен текстовыми сообщениями между компьютерами рабочей группы (домена) отдела. Сформулируйте ваши требования к перспективному ПО и подготовьте техническое задание.

3. На файловый сервер предприятия регулярно поступает большое число файлов, среди них зачастую имеются файлы-дубликаты. Поступающие файлы располагаются в произвольных каталогах и имеют разные имена, атрибуты и время создания. Сформулируйте ваши требования к перспективному ПО позволяющему находить, и при необходимости удалять с сервера файлы-дубликаты и подготовьте техническое задание.

4. Для экономического отдела предприятия необходимо разработать приложение-калькулятор позволяющее осуществлять основные арифметические операции с точностью не менее 4 знаков после запятой. Интерфейс калькулятора должен позволять пользователю осуществлять вычисления без использования мышки. Сформулируйте ваши требования к перспективному ПО и подготовьте техническое задание.

5. Для проведения занятий по астрономии с учениками средней школы необходимо разработать приложение моделирующее поведение планет Солнечной системы. Сформулируйте ваши требования к перспективному ПО и подготовьте техническое задание.

6. Для обработки простейших текстовых файлов и файлов в формате RTF вашему отделу необходимо разработать приложение-редактор. Сформулируйте ваши требования к перспективному ПО и подготовьте техническое задание.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. По окончании выполнения задания письменный отчёт о выполненной работе, включающий требования заказчика и техническое задание предоставляется для защиты преподавателю.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Каким документом определяется состав технического задания?
2. Какие разделы входят в ТЗ?
3. Кто должен разрабатывать ТЗ?
4. Что происходит с программным продуктом если он не соответствует ТЗ?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [5];
дополнительная литература – [5].

Лабораторная работа 23

Функциональная спецификация ПО

Цель работы: разработка спецификации ПО в соответствии с техническим заданием.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Спецификации представляют собой полное и точное описание функций и ограничений разрабатываемого программного обеспечения. При этом одна часть спецификаций (функциональные) описывает функции разрабатываемого программного обеспечения, а другая часть (эксплуатационные) определяет требования к техническим средствам, надежности, информационной безопасности и т.д.

Диаграмма переходов состояний является графической формой предоставления конечного автомата.

Для построения диаграммы переходов состояний необходимо в соответствии с теорией конечных автоматов определить: основные состояния, управляющие воздействия (или условия перехода), выполняемые действия и возможные варианты переходов из одного состояния в другое.

Для интерактивного программного обеспечения с развитым пользовательским интерфейсом основные управляющие воздействия – команды пользователя, для программного обеспечения реального времени – сигналы от датчиков и/или оператора производственного процесса. Общим для этих типов программного обеспечения является наличие состояния ожидания, когда программное обеспечение приостанавливает работу до получения очередного управляющего воздействия.

Функциональными называют диаграммы, в первую очередь отражающие взаимосвязи функций разрабатываемого программного обеспечения.

Отображение взаимосвязи функций активностной модели осуществляется посредством построения иерархии функциональных диаграмм, схематически представляющих взаимосвязи функций. Каждый блок такой диаграммы соответствует некоторой функции, для которой должны быть определены: исходные данные, результаты, управляющая информация и механизмы ее осуществления – человек или технические средства. Все перечисленные выше связи функции представляются дугами, причем тип связи и ее направление строго регламентированы. Дуги, изображающие каждый тип связей, должны подходить к блоку с определенной стороны (рис. 23.1, а), а направление связи должно указываться стрелкой в конце дуги.

Блоки на диаграмме размещают в соответствии с последовательностью их работы или доминированием, которое понимается как влияние, оказываемое одним блоком на другие. В функциональных диаграммах SADT различают пять типов влияний блоков друг на друга (рис. 23.2).

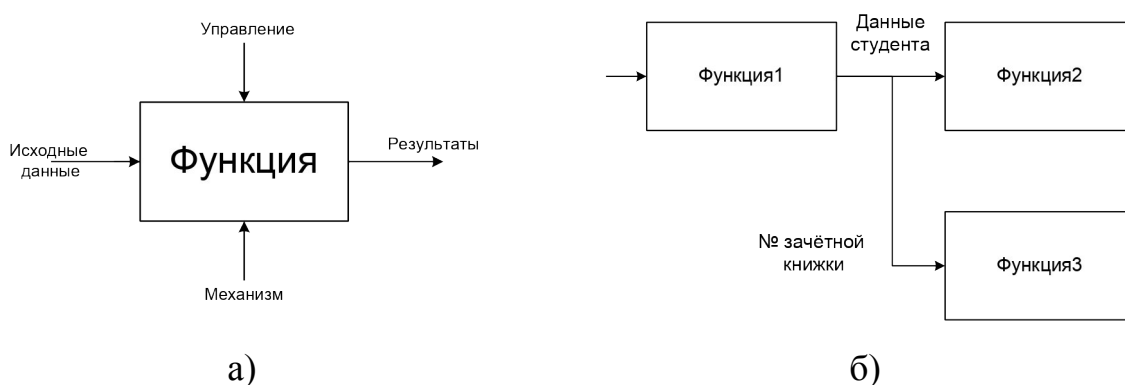


Рисунок 23.1 – Примеры: а) диаграммы переходов состояний интерактивного ПО; б) обозначения дуг при ветвлении

Дуги могут разветвляться и соединяться вместе различными способами. Разветвление означает, что часть или вся информация может использоваться в каждом ответвлении дуги. Дуга всегда помечается до ветвления, чтобы идентифицировать передаваемый набор данных. Если ветвь дуги после ветвления не помечена, то непомеченная ветвь содержит весь набор данных. Каждая метка ветви уточняет, что именно содержит данная ветвь (рис. 23.1, б).

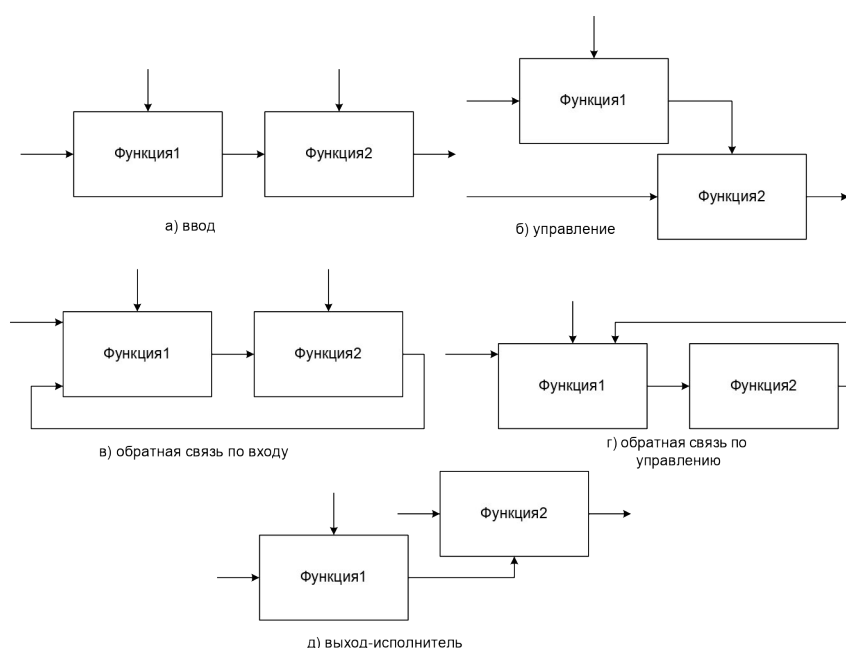


Рисунок 23.2 – Диаграмма переходов состояний интерактивного ПО

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 с диагональю не менее 19", клавиатура, мышь. **Программное обеспечение** – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Лабораторная работа выполняется в соответствии с вариантами задания из лабораторной работы №22 на основании подготовленного технического задания (ТЗ) на разработку ПО.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. Разработайте диаграммы переходов состояний и функциональные диаграммы, полно и точно описывающие разрабатываемое программное обеспечение. По окончании выполнения задания письменный отчет о выполненной работе предоставляется для защиты преподавателю.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Что такое «спецификации» при структурном подходе?
2. Что понимается под диаграммой переходов состояний?
3. Какие функциональные диаграммы вам известны?
4. Для чего необходимы диаграммы потоков данных?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [5];
дополнительная литература – [5].

Лабораторная работа 24

Проектирование классов

Цель работы: исследование основ объектно-ориентированного проектирования класса.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Основной структурной единицей всей концепции объектно-ориентированного программирования (ООП) выступает класс. *Класс* – это программное описание будущего объекта, включающее в себя определение полей и методов.

Основными строительными элементами класса выступают поля и методы. С некоторой степенью упрощения можно воспринимать поля как обычные переменные. Поля могут принимать любой тип данных за исключением типа класса, в котором они задействуются. Обычно поля объявляются в секции `protected`, что дает дополнительную степень защиты

от некорректного изменения их содержимого извне класса. Доступ к полям извне класса реализуется исключительно с помощью методов (свойств и событий). Разработаем класс «Автомобиль» (листинг 24.1).

Листинг 24.1 – Класс автомобиля (первая версия)

```
class TAutomobile{
protected :
    int fspeed;        //скорость движения автомобиля
    bool fhandbrake; //ручной тормоз включен/выключен
public :
    int setspeed(int value){
        if (fhandbrake==false){
            if (value>=0 && value<=200)
                fspeed=value;
            else throw(Exception("Недопустимое значение!"));
        }
        else throw(Exception("Включен стояночный тормоз!"));
        return fspeed;
    }
    int getspeed() { return fspeed; }
};
```

Если на основе класса планируется создать программный объект, то в классе должен существовать по крайней мере один специализированный метод, отвечающий за создание из класса полноценного объекта. Такой метод называется *конструктором*. Конструктор создает программный объект, вложенные объекты (если такие имеются) и инициализирует поля объекта. Для нашего автомобиля пока достаточно услуг простейшего конструктора (листинг 24.2).

Листинг 24.2 – Простой конструктор для класса TAutomobile

```
class TAutomobile{
protected :
    //...
public :
    TAutomobile(); //объявление прототипа конструктора
    //...
```

```
};
TAutomobile:: TAutomobile(){ //конструктор класса
    fspeed=0;
    fhandbrake=true;
}
```

Основная задача деструктора заключается в бесконфликтном освобождении всех ресурсов (в первую очередь памяти), захваченных объектом. По существующему соглашению имя деструктора должно начинаться с символа тильды «~» (листинг 24.3).

Листинг 24.3 – Деструктор для класса TAutomobile

```
class TAutomobile{
    protected :
        //...
    public :
        //...
        TAutomobile(); //прототип конструктора
        ~TAutomobile(); //прототип деструктора
        //...
};
TAutomobile:: ~TAutomobile()
{ // код деструктора}
```

Обратите внимание, что мы определили деструктор вне класса и для этого нам понадобился оператор разрешения видимости «: :».

Для разрушения заданного через указатель объекта деструктор должен вызываться неявно, с помощью оператора `delete`.

Нами разработан класс автомобиля, теперь стоит проверить его в действии. Единственный способ проверки – создание на основе класса TAutomobile действующего объекта – *экземпляра класса* (листинг 24.4).

Листинг 24.4 – Создание экземпляра класса

```
TAutomobile *a;           // указатель на будущий объект
a = new Tautomobile(); //вызов конструктора автомобиля
a->handbrake=false;       //сняли со стояночного тормоза
```

```

a->setspeed(100);      //набрали скорость 100 км/ч
//...
a->setspeed(0);        //остановились
a->handbrake=true;     //поставили на стояночный тормоз
delete a;             //вызов деструктора

```

Для обеспечения доступа к свойствам и методам класса в состав языка C++ введены два оператора: стрелка «->» и точка «.». В C++ наиболее востребован оператор стрелка – он необходим в ситуации, когда доступ осуществляется при посредничестве указателя на объект. Оператор точка решает ту же самую задачу, если доступ осуществляется непосредственно через объект.

Инкапсуляция объекта в объекте. Допустим, что в нашем распоряжении имеется класс TEngine, способный играть роль двигателя. Внедрим экземпляр класса TEngine в объект-автомобиль (листинг 24.7). Логика управления двигателем представлена в листинге 24.8.

Листинг 24.5 – Внедрение класса в класс

```

class TEngine{
    protected :
        bool factive; //двигатель включен/отключен
    public :
        //управление двигателем с помощью свойства
        __property bool active={read=factive, write=factive};
};

class TAutomobile{
    protected :
        TEngine *fengine; //указатель на двигатель
        //... другие поля ...
    public :
        TAutomobile(); //объявление конструктора
        ~TAutomobile(); //объявление деструктора
        ...
        //свойство предоставляет доступ к инкапс. объекту TEngine
        __property TEngine *engine = {read=fengine, write=fengine};
        //двигатель
};

TAutomobile:: TAutomobile()

```

```

    {fengine = new Tengine; //создаем объект-двигатель
      fengine->active=false; //двигатель выключен
      fspeed=0; //скорость автомобиля 0 км/ч
      fhandbrake=true; //включен ручной тормоз
    }
TAutomobile:: ~TAutomobile()
{
    delete fengine; //уничтожаем объект-двигатель
}

```

Листинг 24.6 – Управление двигателем

```

TAutomobile *a;
a = new TAutomobile(); //вызов конструктора
a->engine->active=true; //заводим двигатель

```

Создание класса-наследника. В основе концепции наследования лежит возможность строительства нового класса на базе уже существующего родительского класса. Благодаря наследованию дочерний класс впитывает в себя все родовые черты своего предка. Ограничений на длину цепочки наследования нет. Таким образом, самый последний в древе наследования класс вбирает в себя все лучшее от всех своих предшественников.

В нашем распоряжении уже имеется класс `TAutomobile`, олицетворяющий все наши знания в области ООП. Попробуем развить первоначальный успех и создать программный класс, способный смоделировать пассажирский автобус – класс `TAutobus`.

Автобус – это всего лишь разновидность автомобиля. У него, как и у автомобиля, есть двигатель, автобус также может набирать и снижать скорость движения. Одним словом, за основу нашего нового класса можно взять уже существующий `TAutomobile` и дополнить его всего одной возможностью – перевозить пассажиров (листинг 24.9).

Листинг 24.7 – Определение класса-наследника

```

class TAutobus : public TAutomobile{
    private :

```

```

unsigned char fpassengercount;
unsigned char setpassengercount(unsigned char value) {
    if (value>=0 && value<=50) fpassengercount=value;
    else throw(Exception(«Недопустимое число пассажиров!»));
};

public :
    __property unsigned char passengercount =
{read=fpassengercount, write=setpassengercount};
};

```

Как видите, механизм наследования позволяет нам строить иерархию классов, в которой дочерние классы вбирают в себя все черты своих классов-предков и имеют возможность развивать родовые черты.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024 × 768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19², клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

На языке программирования C++ разработайте классы согласно полученного варианта.

Варианты к заданию

Вариант 1.

- ~ класс «Воздушный шар». Умения класса: взлет и посадка;
- ~ класс «Дирижабль» и класс «Двигатель». Умения класса «Дирижабль»: взлет, посадка, движение со скоростью 0-100 км.\ч. Умения

класса «Двигатель»: вкл. и откл. Класс «Дирижабль» должен уметь отправлять сигналы «Взлет» и «Посадка»;

~ класс «Диспетчер». Обладает слотом способным получать сигналы от класса «Дирижабль».

Вариант 2.

~ класс «Батискаф». Умения класса: открыть люк для посадки экипажа, закрыть люк, погружение на глубину до 3000 м. Всплытие. Батискаф должен уметь отправлять сигналы об открытии и закрытии люка;

~ класс «Подводная лодка» и класс «Двигатель». Умения класса «Подводная лодка». Открыть люк, закрыть люк, погружение на глубину до 300 м. Всплытие. Движение под водой со скоростью 0-10 узлов в час, движение над водой 0-25 узлов в час. Умения класса «Двигатель»: вкл. и откл.;

~ класс «Капитанский мостик». Обладает слотом, способным получать сигналы о закрытии и открытии люка.

Вариант 3.

~ класс «Чайник». Умения класса: залить воду 0-2 литра, вылить воду;

~ класс «Электрочайник» и класс «Нагреватель». Умения класса «Электрочайник». Залить воду 0-2 литра, вылить воду. Нагреть воду до температуры 60-100 градусов. Умения класса «Нагреватель»: вкл. и откл. Класс «Электрочайник» должен уметь отправлять сигнал о том, что он включился и выключился;

~ класс «Наблюдатель». Обладает слотом способным получать сигналы от класса «Электрочайник».

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. Оформите

разработанные вами классы в отдельном программном модуле. Разработайте приложение, демонстрирующее работу классов.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Что в ООП понимается под термином «абстракция»?
2. Что такое инкапсуляция?
3. Каких преимуществ позволяет достичь наследование?
4. Почему доступ к полям объекта следует осуществлять с помощью методов?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [5];

дополнительная литература – [2].

Лабораторная работа 25

Объектный подход к проектированию ПО. Язык UML

Цель работы: исследование особенностей проектирования объектно-ориентированного ПО.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Спецификация разрабатываемого программного обеспечения при использовании UML объединяет несколько моделей:

1. Модель использования представляет собой описание функциональности программного обеспечения с точки зрения пользователя.
2. Логическая модель описывает ключевые абстракции программного обеспечения (классы, интерфейсы и т.п.), т.е. средства, обеспечивающие требуемую функциональность.
3. Модель реализации определяет реальную организацию программных модулей в среде разработки.

4. Модель процессов отображает организацию вычислений и оперирует понятиями «процессы» и «нити». Она позволяет оценить производительность, масштабируемость и надежность программного обеспечения.

5. Модель развертывания показывает особенности размещения программных компонентов на конкретном оборудовании.

Таким образом, каждая из указанных моделей характеризует определенный аспект проектируемой системы, а все они вместе составляют относительно полную модель разрабатываемого программного обеспечения.

Всего UML предлагает девять дополняющих друг друга диаграмм, входящих в различные модели:

- ~ диаграммы вариантов использования;
- ~ диаграммы классов;
- ~ диаграммы пакетов;
- ~ диаграммы последовательностей действий;
- ~ диаграммы кооперации;
- ~ диаграммы деятельности;
- ~ диаграммы состояний объектов;
- ~ диаграммы компонентов;
- ~ диаграммы размещения.

Диаграммы вариантов использования. Вариант использования представляет собой характерную процедуру применения разрабатываемой системы конкретным действующим лицом, в качестве которого могут выступать не только люди, но и другие системы или устройства.

В зависимости от цели выполнения конкретной процедуры различают следующие варианты использования:

- ~ основные – обеспечивают требуемую функциональность разрабатываемого программного обеспечения;

~ вспомогательные – обеспечивают выполнение необходимых настроек системы и ее обслуживание (например, архивирование информации и т.п.);

~ дополнительные – обеспечивают дополнительные удобства для пользователя (как правило, реализуются в том случае, если не требуют серьезных затрат каких-либо ресурсов ни при разработке, ни при эксплуатации).

Диаграммы вариантов использования позволяют наглядно представить ожидаемое поведение системы. Основными понятиями диаграмм вариантов использования являются: действующее лицо, вариант использования, связь. Пример диаграммы вариантов использования представлен на рисунке 25.1.

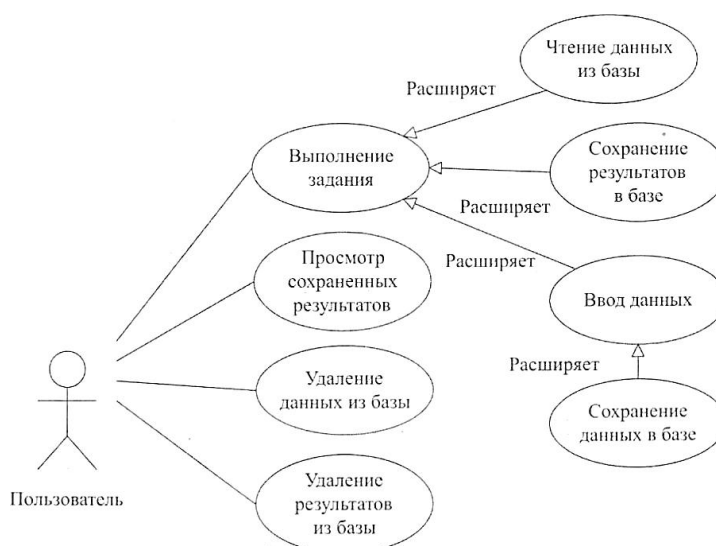


Рисунок 25.1 – Пример диаграммы вариантов использования

Диаграмма последовательностей системы. Диаграмма последовательностей системы – графическая модель, которая для определенного сценария варианта использования показывает генерируемые действующими лицами события и их порядок. При этом система рассматривается как единое целое.

Для построения диаграммы последовательностей системы необходимо:

- ~ представить систему как «черный ящик» и изобразить для нее линию жизни – вертикальную пунктирную линию, подходящую к блоку снизу;
- ~ идентифицировать каждое действующее лицо и изобразить для него линию жизни (много действующих лиц бывает в вариантах совместного использования программного обеспечения);
- ~ из описания варианта использования определить множество системных событий и их последовательность;
- ~ изобразить системные события в виде линий со стрелкой на конце между линиями жизни действующих лиц и системы, а также указать имена событий и списки передаваемых значений.

Пример диаграммы последовательностей системы представлен на рисунке 25.2.

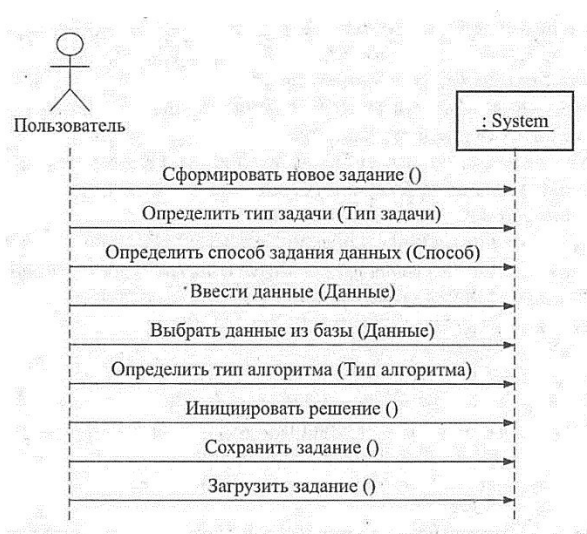


Рисунок 25.2 – Пример диаграммы последовательностей системы

В отличие от внутренних, события, которые генерируются для системы действующими лицами, называют системными. Каждую системную операцию необходимо описать. Обычно описание системной операции содержит:

- ~ имя операции и ее параметры;

- ~ описание обязанности;
- ~ указание типа;
- ~ названия вариантов использования, в которых она используется;
- ~ примечания для разработчиков алгоритмов и т.д.;
- ~ описание обработки возможных исключений;
- ~ описание вывода неинтерфейсных сообщений;
- ~ предположение о состоянии системы до выполнения операции (предусловие);
- ~ описание изменения состояния системы после выполнения операции (постусловие).

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

В соответствии с полученным вами техническим заданием разработайте диаграмму вариантов использования и диаграмму последовательностей системы.

По окончании выполнения задания письменный отчёт о выполненной работе предоставляется для защиты преподавателю.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Какие модели могут использоваться в UML?
2. Какие диаграммы применяются в UML?
3. Что такое «вариант использования»?
4. Для чего необходима концептуальная модель?
5. Что понимается под «деятельностью»?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [5];

дополнительная литература – [6].

Лабораторная работа 26

Оконное приложение на Windows API

Цель работы: исследование основ проектирования приложений с применением Windows API.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Обозначим этапы, через которые надо пройти программисту для того, чтобы создать простейшее жизнеспособное приложение для Windows:

- ~ описать макет будущего главного окна приложения. Для этого заполняются поля специализированной структуры `WNDCLASSEX`, называемой классом окна;
- ~ зарегистрировать этот класс в Windows (функция `RegisterClassEx`);
- ~ создать экземпляр главного окна приложения (`CreateWindow`);
- ~ отобразить окно на экране компьютера (`ShowWindow`);

~ организовать цикл обработки сообщений Windows;
 ~ описать оконную процедуру, в рамках которой наша программа станет обслуживать все поступающие в её адрес сообщения.

Пример приложения для Windows показан в листинге 26.1.

Листинг 26.1 – Приложение для Windows

```
#include <windows.h>
#include <Winapi.Windows.hpp>
#pragma hdrstop
#include <tchar.h>
HINSTANCE hInst;
TCHAR *AppName="demo";
ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPTSTR lpCmdLine, int nCmdShow){
    MSG msg;
    MyRegisterClass(hInstance);
    if (!InitInstance (hInstance, nCmdShow))
        return false;
    while (GetMessage(&msg, NULL, 0, 0)){
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return (int) msg.wParam;
}
ATOM MyRegisterClass(HINSTANCE hInstance){
    WNDCLASSEX wcex;
    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style= CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = WndProc;
    wcex.cbClsExtra = 0;
    wcex.cbWndExtra = 0;
    wcex.hInstance = hInstance;
    wcex.hIcon = LoadIcon(hInstance, IDI_APPLICATION);
    wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH) (COLOR_WINDOW+1);
    wcex.lpszMenuName = "";
    wcex.lpszClassName = AppName;
    wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(0));
    return RegisterClassEx(&wcex);
```

```

}
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow){
    HWND hWnd;
    hInst = hInstance;
    hWnd = CreateWindow(AppName, "приложение WinAPI",
WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL,
NULL, hInstance, NULL);
    if (!hWnd) return false;
}
ShowWindow(hWnd, nCmdShow);
UpdateWindow(hWnd);
return true;
}
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam){
    PAINTSTRUCT ps;
    HDC hdc;
    switch (message){
    case WM_COMMAND: break;
    case WM_PAINT: //обработка команд GDI
        hdc = BeginPaint(hWnd, &ps);
        //..
        EndPaint(hWnd, &ps);
        break;
    case WM_SIZE: break;
    case WM_SETFOCUS: break;
    case WM_DESTROY: PostQuitMessage(0);break;
    default: return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

```

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024 × 768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19², клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Опираясь на функции Win API разработайте простейшее приложение с графическим пользовательским интерфейсом для Windows. В том числе:

~ создайте класс окна кнопки, разместите кнопку в правом нижнем углу главного окна приложения и сделайте так, чтобы щелчок по кнопке закрывал главное окно приложения;

~ изучите какие сообщения поступают в адрес окна в момент изменения его размеров, сделайте так чтобы кнопка закрытия окна при изменении его размеров всегда находилась в правом нижнем углу окна;

~ создайте класс окна для строки ввода и разместите эту строку на главном окне;

~ создайте класс окна комбинированного списка, заполните список 3-4 произвольными строками и разместите список на главном окне;

~ сделайте так, чтобы при выборе пользователем элемента списка его текст отображался в строке ввода.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. Разработайте диаграммы переходов состояний и функциональные диаграммы, полно и точно описывающие разрабатываемое программное обеспечение. По окончании выполнения задания письменный отчет о выполненной работе предоставляется для защиты преподавателю.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. В чем на ваш взгляд заключается преимущество приложений, написанных на Windows API?
2. Почему разработка проектов с использованием Windows API требует от программиста глубоких знаний операционной системы?
3. Что такое сообщение Windows?
4. Какие сообщения Windows использовались в вашей программе?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [5];
дополнительная литература – [2].

Лабораторная работа 27

Управление процессами в Windows

Цель работы: исследование приемов работы с процессами в Windows.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Главную роль в формировании нового процесса в Windows выполняет функция `CreateProcess()`, создающая процесс и поток управления (листинг 27.1).

Листинг 27.1 – Описание функции `CreateProcess()`

```

BOOL WINAPI CreateProcess (
    __in          LPCTSTR lpApplicationName,
    __in_out      LPTSTR lpCommandLine,
    __in          LPSECURITY_ATTRIBUTES lpProcessAttributes,
    __in          LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in          BOOL bInheritHandles,
    __in          DWORD dwCreationFlags,
    __in          LPVOID lpEnvironment,

```

```

__in          LPCTSTR lpCurrentDirectory,
__in          LPSTARTUPINFO lpStartupInfo,
__out         LPPROCESS_INFORMATION lpProcessInformation
);

```

В листинге 27.2 приведен пример запуска процесса Блокнота (файл notepad.exe), в командной строке процессу передается указание открыть файл demo.txt.

Листинг 27.2 – Создание процесса

```

_SECURITY_DESCRIPTOR SD;
InitializeSecurityDescriptor(&SD, SECURITY_DESCRIPTOR_REVISION);
TSecurityAttributes SA;
    SA.nLength=sizeof(TSecurityAttributes);
    SA.lpSecurityDescriptor=&SD;
    SA.bInheritHandle=true;
STARTUPINFO SI;
ZeroMemory(&SI, sizeof(STARTUPINFO));
PROCESS_INFORMATION PI;
CreateProcess(L"c:\\windows\\notepad.exe", L"open c:\\demo.txt",
    &SA, &SA, false, NORMAL_PRIORITY_CLASS,
    NULL, NULL, &SI, &PI);

```

Доступ к процессу. Для получения доступа к определенному процессу, исполняющемуся в ОС, следует воспользоваться функцией `OpenProcess()` (листинг 27.3).

Листинг 27.3 – Описание функции `OpenProcess()`

```

HANDLE WINAPI OpenProcess(    __in          DWORD
dwDesiredAccess, __in      BOOL bInheritHandle, __in      DWORD
dwProcessId);

```

Приоритет процесса. Количество квантов процессорного времени, выделяемого процессу (точнее, потоку управления), определяется его приоритетом. Зависимость прямо пропорциональная – чем выше приоритет,

тем большего внимания его удостоит процессор (или процессоры) вашего компьютера.

Система Windows поддерживает шесть категорий приоритетов процессов:

- ~ фоновый (IDLE_PRIORITY_CLASS);
- ~ нормальный заднего плана (BELOW_NORMAL_PRIORITY_CLASS);
- ~ нормальный (NORMAL_PRIORITY_CLASS);
- ~ нормальный переднего плана (ABOVE_NORMAL_PRIORITY_CLASS);
- ~ высокий (HIGH_PRIORITY_CLASS);
- ~ процесс реального времени (REALTIME_PRIORITY_CLASS).

По умолчанию, при создании нового процесса ему присваивается нормальный приоритет.

Для системного программиста большой практический интерес предоставляет возможность ускорения или замедления выполняющегося процесса. Для достижения этой цели достаточно сделать два шага: во-первых, получить дескриптор процесса (с помощью функции `OpenProcess` или `GetCurrentProcess`) и, во-вторых, изменить его приоритет. Тем или иным способом приобретаем дескриптор интересующего нас процесса, приступаем к главному – меняем приоритет задачи. Для этого вызовем функцию `SetPriorityClass()` (листинг 27.4).

Листинг 27.4 – Вызов функции `SetPriorityClass()`

```
BOOL WINAPI SetPriorityClass(__in HANDLE hProcess,
                           __in DWORD dwPriorityClass);
```

Завершение процесса. Процесс может прекратить свое существование самостоятельно, для этого он должен вызвать функцию `VOID WINAPI ExitProcess( UINT uExitCode)`. Но если процесс не желает уходить из жизни добровольно, то применяют функцию `BOOL WINAPI TerminateProcess(HANDLE hProcess, UINT uExitCode)`. Функция может

быть вызвана родительским процессом или любым другим процессом, знающим дескриптор разрушаемого процесса. В обеих функциях параметр `uExitCode` содержит код завершения процесса. Этот код можно получить с помощью функции `GetExitCodeProcess()`.

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Разработайте приложение, позволяющее:

- ~ собирать сведения о выполняющихся в системе процессах;
- ~ останавливать пользовательский процесс из списка;
- ~ создавать произвольный процесс.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. По окончании выполнения задания письменный отчет о выполненной работе предоставляется для защиты преподавателю.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Почему истинная многозадачность возможна только в многопроцессорных системах?
2. Как регулируется приоритет процесса?
3. Что такое поток управления?
4. Как осуществляется сбор информации о процессах системы?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [5];
дополнительная литература – [2].

Лабораторная работа 28

Многопоточные приложения

Цель работы: исследование технологий проектирования многопоточных приложений.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

В момент старта процесса под управлением Windows обязательно создается главный поток процесса (поток управления). Перед главным потоком приложения поставлена серьезная задача – он отвечает за взаимодействие с ОС и за выполнение кода программы. Кроме обязательного потока управления процесс имеет возможность обладать некоторым набором вспомогательных потоков (нитей). Каждый из потоков выполняет поставленную перед ним подзадачу и при желании может быть приостановлен или запущен вновь.

Для программной реализации потока в VCL C++ Builder объявлен абстрактный класс `TThread`, на основе которого программист создаст свой полнофункциональный поток. Для того чтобы самое обычное приложение сделать многопоточным, требуется к уже существующему приложению добавить специальный программный модуль, шаблон которого представлен в листинге 28.1.

Листинг 28.1 – Модуль потока `TDemoThread`

```
#include <vcl.h>
#pragma hdrstop
#include "Unit2.h"
#pragma package(smart_init)
__fastcall TDemoThread::TDemoThread(bool CreateSuspended) :
TThread(CreateSuspended) {}
void __fastcall TDemoThread::Execute()
{
    //---- программный код потока ----
}
```

Очень часто проверка состояния свойства служит сигналом к выходу из цикла обработки задачи потока (листинг 28.2).

Листинг 28.2 – Шаблон кода основного метода потока

```
void __fastcall TDemoThread::Execute() {
    do {
        //код потока
    }
    while (Terminated);
}
```

Управление приоритетом потока. Обычный поток приложения (точно так, как и поток управления приложения) обладает определенным приоритетом, от которого зависит, сколько квантов времени будет выделено процессором компьютера для обработки подзадачи. Приоритет потока определяется значениями двух составляющих: приоритетом процесса-

владельца потока и собственно приоритетом этого потока. Поэтому приоритет обычного потока часто называют *относительным приоритетом*. Такое название объясняется тем, что два потока, скажем, с приоритетом `tpNormal`, но порожденные процессами с различными приоритетами, не смогут претендовать на одинаковое процессорное время. Реальный приоритет потока определяется приоритетом порождающего его процесса и своим приоритетом.

В C++ Builder приоритет потока определяется свойством, представленном в листинге 28.3.

Листинг 28.3 – Описание свойства Priority

```
__property TThreadPriority Priority; //по умолчанию tpNormal
enum TThreadPriority : unsigned char { tpIdle, // фоновый
    tpLowest, //низкий приоритет
    tpLower, //пониженный приоритет
    tpNormal, //нормальный приоритет
    tpHigher, //повышенный приоритет
    tpHighest, //высочайший приоритет
    tpTimeCritical}; //приоритет реального времени
```

Синхронный и асинхронный вызовы. Если из тела основного метода потока `Execute()` необходимо осуществить вызов внешнего метода (метода не принадлежащего потоку) или обратиться к свойству компонента библиотеки VCL, то не следует это делать напрямую, вместо этого надо использовать специализированные методы `Synchronize()` или `Queue()`. Задача методов – исключение взаимных блокировок потоков.

Синхронизация потоков. Одной из серьезных проблем, с которой сталкиваются разработчики многопоточных приложений, заключается в организации совместного доступа двух (и более) потоков к общему ресурсу. Конкурируя за обладание общим ресурсом, потоки могут инициировать трудно выявляемые ошибки. Поэтому при реализации взаимодействующих потоков программист обязан решить задачу корректного доступа потоков к

разделяемому ресурсу. Для исключения ошибок при подключении к общему ресурсу потокам необходимо синхронизировать свои действия так, чтобы ресурс принадлежал не более чем одному потоку одновременно. При попытке обращения к захваченному ресурсу остальных потоков они должны быть приостановлены до момента его освобождения.

Объект синхронизации обладает двумя устойчивыми состояниями: *сигнальным* (signaled) и *несигнальным* (nonsignaled). Сигнальное состояние разрешает приостановленному потоку приступить к работе, несигнальное состояние объекта синхронизации, напротив, предлагает ожидающему своей очереди потоку спать дальше.

Семафор TSemaphore. Семафор (semaphore) – может находиться в совместном использовании сразу нескольких потоков, но при этом он умеет управлять количеством этих потоков – выступает в роли ограничителя их «популяции». Для этого он снабжен счетчиком, определяющим максимальное число потоков, которым разрешено выполняться совместно. С запуском каждого нового потока значение счетчика уменьшается на единицу, и когда оно достигнет нуля, семафор перейдет в несигнальное состояние. Это означает, что последующие потоки приостанавливаются и будут дожидаться момента, пока какой-нибудь из потоков не освободит семафор. Семафор обладает сразу тремя перегружаемыми версиями конструктора (листинг 28.4).

Листинг 28.4 – Описание конструкторов семафора TSemaphore

```
TSemaphore(bool UseCOMWait);
TSemaphore(PSecurityAttributes SemaphoreAttributes, int
AInitialCount, int AMaximumCount, const UnicodeString Name, bool
UseCOMWait);
TSemaphore(unsigned DesiredAccess, bool InheritHandle, const
UnicodeString Name, bool UseCOMWait);
```

Несмотря на то, что семафор – достаточно сложный объект синхронизации, работа с ним не вызовет особых затруднений. Листинг 28.5 отражает процесс создания экземпляра семафора.

Листинг 28.5 – Создание семафора

```
TSemaphore *Semaphore;
void __fastcall TForm1::FormCreate(TObject *Sender){
int AMaximumCount=TThread::ProcessorCount*10;
    Semaphore=new TSemaphore(NULL, AMaximumCount-
1,AMaximumCount,"DemoSemaphor",false);
...
}
```

Обратите внимание на то, как в коде определяется максимальное число потоков, контролируемое семафором. Мы узнаем число процессоров в компьютере и, исходя из этого, ограничиваем количество потоков. Напомним, что не рекомендуется создавать более 16 потоков на одно ядро процессора. Листинг 28.6 демонстрирует работу с семафором потока.

Листинг 28.6 – Применение семафора

```
void __fastcall TDemoThread::Execute(){
    Semaphore->Acquire();
    //работа потока с ресурсом (данными)
    Semaphore->Release();
}
```

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024 × 768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19², клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Разработайте приложение, позволяющее создавать миниатюры изображений для файлов, расположенных в одной из папок вашего компьютера:

1. Реализуйте алгоритм сбора миниатюр в однопоточном режиме.
2. Реализуйте алгоритм сбора миниатюр в многопоточном режиме.
3. Ограничьте число одновременно выполняющихся потоков с помощью семафора.
4. Сравните время выполнения алгоритмов.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. По окончании выполнения задания письменный отчет о выполненной работе предоставляется для защиты преподавателю.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Как зависит приоритет потока от приоритета породившего его процесса?
2. Что понимается под взаимной блокировкой потоков?

3. Как осуществляется синхронизация потоков?
4. Назовите известные вам объекты синхронизации?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [5];
дополнительная литература – [2].

Лабораторная работа 29

Динамически подключаемые библиотеки

Цель работы: исследование методов разработки программных библиотек.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Создание проекта динамически подключаемой библиотеки начинается с обращения к традиционному пункту меню **File | New | Other**, затем на странице **New** диалогового окна **New Items** следует выбрать пункт **Dynamic-link Library**. В ответ на наши действия Delphi создаст шаблон кода, начинающийся с ключевого слова `library` – это и есть визитная карточка проекта библиотеки DLL. Прообраз нашей первой «тренировочной» библиотеки представлен в листинге 29.1.

Листинг 29.1 – Шаблон кода библиотеки

```
#pragma hdrstop
```

```
#pragma argsused
extern "C" int _libmain(unsigned long reason){
    return 1;
}
```

Объявление и экспорт функций в DLL. Для того чтобы стороннее приложение смогло воспользоваться описанными в библиотеке функциями, эти функции должны быть помечены, как экспортируемые. Синтаксис объявления экспортируемых библиотекой методов имеет ряд небольших отличий от уже привычного описания. Четко выделяются следующие особенности:

- ~ между экспортирующей функции библиотекой и выступающим в роли импортера приложением должно быть заключено *соглашение о вызовах*, оно необходимо для обеспечения корректной передачи параметров и возврата результата выполнения функции;

- ~ объявление экспортируемой функции с ключевым словом `export` и использование секции `exports` для указания имен экспортируемых процедур и функций.

Вызов библиотеки из приложения. Чтобы динамической библиотекой смогло воспользоваться приложение, нам следует выбрать для нее подходящий каталог. Для этого подходят:

- ~ каталог, из которого запускается основной исполняемый exe-файл;
- ~ текущий каталог;
- ~ системная папка Windows;
- ~ папка Windows;
- ~ каталоги, включенные в переменную окружения `PATH`.

В нашем случае во время разработки проекта наиболее целесообразно размещать свои библиотеки в том же каталоге, где находится исполняемый файл.

Неявное подключение DLL. Для того чтобы воспользоваться услугами динамической библиотеки, приложение должно отобразить DLL в свое

адресное пространство. Различают два способа решения этой задачи: *неявное* (статическое) и *явное* (динамическое) подключение библиотеки.

Неявная загрузка предназначена для тех случаев, когда услуги библиотеки необходимы на протяжении всего периода выполнения приложения. В этом случае библиотека подключается к процессу в момент его старта и отключается вместе с завершением работы. Организация неявной загрузки включает следующие этапы:

- ~ подключение к проекту файла .lib (файл создаётся автоматически в момент компиляции библиотеки);
- ~ определение в проекте прототипа импортируемой функции.

Для определения в проекте прототипа импортируемой функции нам понадобится одна строка кода `extern "C" float __declspec(dllimport) add(float x, float y)`, всё остальное вы найдёте в листинге 29.2.

Листинг 29.2 – Статическое подключение

```
#include <vcl.h>
#pragma hdrstop
#include "Unit2.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
extern "C" float __declspec(dllimport) add(float x, float y);
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner){}
void __fastcall TForm1::Button1Click(TObject *Sender){
float x=StrToFloat(Edit1->Text);
float y=StrToFloat(Edit2->Text);
float z=add(x,y);
Label2->Caption=FloatToStr(z);
}
```

Явное подключение DLL. Явное (динамическое) подключение библиотеки к проекту целесообразно применять в тех случаях, когда обращение к библиотеке происходит нерегулярно. В такой ситуации лучше

всего отображать образ библиотеки в адресное пространство процесса только на время обращения к той или иной функции и сразу выгружать библиотеку.

При организации явной загрузки программист должен сделать четыре шага:

- ~ загрузить DLL;
- ~ получить адрес импортируемой функции;
- ~ вызвать импортируемую функцию;
- ~ выгрузить библиотеку из памяти.

Практическая сторона вопроса явной загрузки DLL и обращения к функции суммирования отражена в листинге 29.3.

Листинг 29.3 – Явное подключение библиотеки

```
const wchar_t *libname={L"mathlib.dll"}; //имя библиотеки
const char *fname={"_add"}; //имя импортируемой функции
typedef float (__import fadd(float,float));
HMODULE hLib=LoadLibrary(libname); //загрузка библиотеки
if (hLib>0) { //если библиотека успешно загружена
    fadd *add;
    add=(fadd*)GetProcAddress(hLib,fname); //получаем адрес функции
    if (add) { //если получен адрес импортируемой функции
        float x=StrToFloat(Edit1->Text);
        float y=StrToFloat(Edit2->Text);
        float z=add(x,y);
        Label2->Caption=FloatToStr(z);
    }
    FreeLibrary(hLib); //выгружаем библиотеку
```

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024 × 768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19², клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Разработайте DLL позволяющую вычислять две любые из перечисленных ниже функций:

- ~ возведение значения в произвольную степень;
- ~ факториал числа;
- ~ расчёт гипотенузы прямоугольного треугольника по значениям катетов;
- ~ возврат значения синуса (косинуса, тангенса, и т.д.) угла по известным значениям катетов и гипотенузы;
- ~ преобразование градусов в радианы (или радианы в градусы);
- ~ корни квадратного уравнения;
- ~ разработайте DLL позволяющую хранить и выводить экранную форму, которая позволит пользователю сделать одно из перечисленных ниже действий:
 - ~ выбирать дату в календаре и возвращать выбранную дату в приложение;
 - ~ выбирать в цвет (из нескольких предложенных) и возвращать выбранное значение в приложение;
 - ~ вводить текстовую строку и возвращать её в приложение.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. По окончании выполнения задания письменный отчёт о выполненной работе предоставляется для защиты преподавателю.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Что понимается под модульным программированием?
2. В каких случаях целесообразно использовать неявное подключение библиотеки к приложению?
3. В каких случаях целесообразно использовать явное подключение библиотеки к приложению?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [5];
дополнительная литература – [2].

Лабораторная работа 30

Передача данных через буфер обмена

Цель работы: исследование особенностей межпрограммного взаимодействия с помощью буфера обмена.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Буфер обмена предназначен для передачи различного типа данных между приложениями или внутри одного приложения. Функционально буфер обмена представляет собой некоторый перечень методов Windows API, осуществляющих операции с областью памяти. Содержание операций заключается в действиях по загрузке в память и выгрузки из нее данных различного типа. Для обеспечения максимально удобной работы с буфером обмена в VCL введен специальный класс `TClipboard`, он описан в модуле `Vcl.Clipbrd.hpp` (который необходимо подключать к проекту). С

присоединением данного модуля программист получит доступ к буферу обмена при посредничестве функции `TClipboard* Clipboard(void)`.

Буфер обмена служит универсальным хранилищем данных и позволяет размещать в себе обычную текстовую информацию, растровые и векторные рисунки, объекты OLE и т.п. Однако для различных типов данных, направляемых в буфер, требуется уникальный способ их хранения. Этот способ определяется *форматом буфера обмена*. Для ключевых, наиболее часто используемых типов объектов в Windows заранее подготовлены специальные зарегистрированные форматы буфера обмена. Однако если приложение предполагает размещать в буфере какую-то экзотическую информацию, то программист имеет полное право описать и зарегистрировать в системе собственный – *пользовательский формат*.

Просмотрите код примера в листинге 30.1. В нашем распоряжении есть командный объект `acPaste:Taction`, отвечающий за вставку текста из буфера обмена. В рамках события `OnUpdate()` мы проверяем наличие в буфере обмена текстовые данные и активируем (или переводим в пассивный режим) команду.

Листинг 30.1 – Проверка наличия в буфере обмена текстовых данных

```
#include "clipbrd.hpp"
void __fastcall Tform1::acPasteUpdate(Tobject *Sender){
    acPaste->Enabled=Clipboard()->HasFormat(CF_TEXT); //проверка
буфера
}
```

Регистрация пользовательского формата буфера обмена. Для профессионального разработчика очень важно уметь создавать собственные форматы буфера обмена. Создайте новое приложение, состоящее из одной главной формы. При сохранении проекта я традиционно переименую форму в `frmMain` и соответствующий ей программный модуль в `main.cpp`. Подключите к проекту модуль `Clipbrd`, объявите структуру `TDBRecord`

(позднее она определит новый формат буфера обмена), объявите переменную CF_MYFORMAT (в ней окажется код нового зарегистрированного формата) и опишите секцию инициализации так, как предложено в листинге 30.2.

Листинг 30.2 – Подготовка модуля к работе с новым форматом буфера обмена

```
struct TDBRecord { //товар, производитель, единица измерения
wchar_t Name[40], Creator[40], Measure[5];
    Double Money; //цена за единицу
    Tdate DateIn; //дата поступления
};
Word CF_MYFORMAT; //код формата
__fastcall TForm1::Tform1(Tcomponent* Owner): Tform(Owner)
{ //регистрация формата
    CF_MYFORMAT=RegisterClipboardFormat(L"CF_MYFORMAT");
}
```

В секции публичных объявлений главной формы опишите заголовки методов копирования данных в буфер обмена и загрузки данных из буфера (листинг 30.3).

Листинг 30.3 – Объявление методов работы с буфером обмена

```
#include <Vcl.Clipbrd.hpp>
void __fastcall CopyToClipboard(TDBRecord Value){
    HANDLE h=GlobalAlloc(GMEM_MOVEABLE, sizeof(TDBRecord));
    Clipboard()->Open();
    __try
    {LPVOID FirstByte=GlobalLock(h);
        Move(&Value, FirstByte, sizeof(TDBRecord));
    Clipboard()->SetAsHandle(CF_MYFORMAT, (unsigned int) h);
    }
    catch (...)
    { GlobalUnlock(h);
        Clipboard()->Close();
        MessageBeep(0);
    }
    GlobalUnlock(h);
    Clipboard()->Close();
}
```

```

TDBRecord __fastcall LoadFromClipboard() {
    TDBRecord DBRecord;
    ZeroMemory(&DBRecord, sizeof(TDBRecord));
    if (Clipboard()->HasFormat(CF_MYFORMAT)) {
        Clipboard()->Open();
        HANDLE h=(void *) Clipboard()-> GetAsHandle(CF_MYFORMAT);
        if (h!=0) {    LPVOID FirstByte=GlobalLock(h);
            __try
            {
                Move(FirstByte, &DBRecord, sizeof(TDBRecord));
            }
            __finally
            {
                GlobalUnlock(h);
                Clipboard()->Close();
            }
        }
    }
    return (DBRecord);
}

```

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Разработайте приложение, позволяющее обмениваться данными через буфер обмена структурами из вариантов к заданию:

Варианты к заданию

1. Прайс-лист магазина. В прайс-лист входят следующие позиции: название товара, цена за единицу, единица измерения, количество товара в магазине.
2. Расписание телепередач на неделю. В расписание входят следующие позиции: день недели, номер канала, время начала передачи, название передачи, жанр передачи.
3. Зачётная книжка студента. Необходимо предусмотреть следующие элементы: дисциплина, количество часов, вид отчётности (экзамен, зачёт, курсовая работа), оценка.
4. Квартиры многоэтажного дома. В структуру следует включить следующие позиции: номер квартиры, число комнат, номер этажа, площадь, наличие балкона(ов), признак первого этажа, признак последнего этажа.
5. Структура для хранения сведений об автотранспортных средствах на стоянке: тип автомобиля (грузовой, легковой, ...), регистрационный номер, марка, когда поставлен на стоянку, ФИО владельца.
6. Структура для описания телевизора. В структуру должны входить: фирма-производитель, марка телевизора, диагональ, разрешение (HD, Full HD, 4K, и т.п.).

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. По окончании выполнения задания письменный отчёт о выполненной работе предоставляется для защиты преподавателю.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.

2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Какие форматы буфера обмена поддерживаются в системе по умолчанию?
2. В каких случаях целесообразно использовать неявное подключение библиотеки к приложению?
3. В каких случаях целесообразно использовать явное подключение библиотеки к приложению?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [5];
дополнительная литература – [2].

Лабораторная работа 31

Взаимодействие процессов с помощью сообщений

Цель работы: исследование особенностей межпрограммного взаимодействия с помощью сообщений.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Наиболее распространённым способом взаимодействия процессов в Windows является обмен сообщениями. *Сообщение* представляет собой средство, с помощью которого Windows уведомляет процесс о том, что в системе что-то произошло, например, пользователь щелкнул кнопкой мыши, была нажата клавиша или системный таймер произвел очередной отсчет интервала времени.

Для того чтобы приложение не прозевало предназначенное ей послание, оно периодически просматривает свой «почтовый ящик» на предмет новых поступлений. Для этого в любой программе (кстати, и в самой

Windows) организуется *цикл обработки сообщений* (message loop). И если по какой-то причине у приложения остановится этот цикл, то оно безнадежно «зависнет», а если остановится цикл операционной системы, то нас спасет только перезагрузка.

Поиск окна. Для того чтобы сообщение дошло до окна-получателя, мы должны знать его дескриптор. Если же дескриптор нам неизвестен, то можно воспользоваться функцией `HWND WINAPI FindWindow( __in_opt LPCTSTR lpClassName, __in_opt LPCTSTR lpWindowName)`. Задача функции – среди всех окон процессов попытаться найти именно то, которому предназначено наше послание.

Для поиска главного окна приложения первый параметр устанавливают в `NULL`, во второй параметр передается указатель на строку, заканчивающуюся нулем и содержащую название требуемого окна. В случае успеха функция вернет значение, отличное от нуля (листинг 31.1).

Листинг 31.1 – Поиск окна

```
HWND h=FindWindow(NULL,L"Project1") ;
    if (h>0) {
        //...
    }
```

Регистрация пользовательских сообщений. В Windows зарегистрирован впечатляющий перечень сообщений, отправляемых для активизации окна, минимизации или распахивания его на весь экран. Сообщения появляются на свет при нажатии и отпуске клавиш, при перемещении мыши или щелчке ее кнопки. Все сообщения описываются именованными константами, имена констант, как правило, начинаются с аббревиатуры «WM_» (сокращение от Windows Message). Кроме того, ряд элементов управления способен воспринимать специфичные сообщения. Например, сообщения для комбинированного списка начинаются с «CB_», для дерева – «TVM_», для кнопки – «BN_».

В адрес окна могут поступать сообщения от самых различных отправителей, как находящихся внутри этого же приложения, так и от внешних источников (других приложений и операционной системы). При поступлении сообщений к главному окну приложения у Application возникает событие `__property TMessageEvent OnMessage; typedef void (__closure *TMessageEvent)(tagMSG &Msg, bool &Handled)`. Содержимое сообщения передается в структуре `tagMSG &Msg` (листинг 31.2).

Листинг 31.2 – Объявление структуры tagMSG

```
typedef struct tagMSG {
    HWND      hwnd;      //дескриптор получателя сообщения
    UINT      message;    // идентификатор сообщения
    WPARAM    wParam;    //необязательный старший параметр
    LPARAM    lParam;    //необязательный младший параметр
    DWORD     time;       //время создания сообщения
    POINT      pt;        //экранные координаты указателя мыши
} MSG, *PMSG, NEAR *NPMSG, FAR *LPMSG;
```

Пример обмена сообщениями между процессами. Спроектируем два приложения, демонстрирующих процесс обмена сообщениями. Приложение-передатчик станет отправлять сообщение со своими экранными координатами в адрес приложения-приемника. Приложение-приемник (получив координаты) укажет пользователю направление на приложение-передатчик.

Начнем работу с приложения-передатчика. Создайте новый проект, на единственной форме проекта разместите компонент `TApplicationEvents *ApplicationEvents1`. Уменьшите размер формы до минимальных размеров. Сохраните проект под любым именем. Чтобы научить приложение отправлять сообщения, понадобится минимум усилий, весь исходный код главной формы проекта предложен в листинге 31.3.

Листинг 31.3 – Исходный код приложения-передатчика

```

#include <vcl.h>
#pragma hdrstop
#include "transmit.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmTransmit *frmTransmit;
unsigned WM_MSG; //здесь будем хранить идентификаторы сообщений
__fastcall TfrmTransmit::TfrmTransmit(TComponent* Owner) :
TForm(Owner) {
    WM_MSG=RegisterWindowMessage(L"WM_SENDPOSITION");
}
void __fastcall TfrmTransmit::ApplicationEvents1Idle(TObject
*Sender, bool &Done){//поиск главного окна приложения с
названием "RECEIVER"
HWND h=FindWindow(NULL,L"RECEIVER");
if (h>0){ // если окно обнаружено
    TPoint FormCenter=Point(frmTransmit->ClientWidth / 2,
                             frmTransmit->ClientHeight / 2);
    FormCenter=frmTransmit->ClientToScreen(FormCenter);
    Label1->Caption=Format("Координаты (%d,%d)",
                           ARRAYOFCONST((FormCenter.X,FormCenter.Y)));
    PostMessage(h, WM_MSG, FormCenter.X, FormCenter.Y);
}
}

```

Переходим ко второй части нашей работы – разработке приложения-приемника. Создаем новое приложение и сразу же в свойство `Caption` главной формы проекта заносим заголовок окна «RECEIVER». Разместите на форме единственный компонент – метку `TLabel *Label1` – и сохраните проект под любым именем. Нам осталось разобраться с исходным кодом (листинг 31.4).

Листинг 31.4 – Исходный код приложения-приемника

```

#include <vcl.h>
#include <Winuser.h>
#pragma hdrstop

#include "Unit1.h"
#pragma package(smart_init)
#pragma resource "*.dfm"

```

```

TForm1 *Form1;
TPoint  TransmitterPos; //координаты точки
unsigned WM_MSG; //идентификатор сообщения
__fastcall TForm1::TForm1(TComponent* Owner): TForm(Owner){
    WM_MSG=RegisterWindowMessage(L"WM_SENDPOSITION");
}
void __fastcall TForm1::OnMessage_Event(tagMSG &Msg, bool
&Handled) {
    if (Msg.message==WM_MSG)
    {
        Label1->Caption=Format("Координаты (%d,%d)",
            ARRAYOFCONST( (Msg.wParam,Msg.lParam) ));
        TransmitterPos.X=Msg.wParam;
        TransmitterPos.Y=Msg.lParam;
        Repaint();
    }
}
void __fastcall TForm1::FormCreate(TObject *Sender){
    Application->OnMessage=OnMessage_Event;
    Label1->Caption="";
}
void __fastcall TForm1::FormPaint(TObject *Sender){
    TPoint Center=Point(Form1->ClientWidth/2,
        Form1->ClientHeight/2);
    Form1->Canvas->Brush->Color=Form1->Color;
    Form1->Canvas->FillRect(Form1->ClientRect);
    Form1->Canvas->Brush->Color=Form1->Canvas->Pen->Color;
    Form1->Canvas->Ellipse(Center.X-3,Center.Y-
        3,Center.X+3,Center.Y+3);
    Form1->Canvas->MoveTo(Center.X, Center.Y);
    TransmitterPos=ScreenToClient(TransmitterPos);
    Form1->Canvas->LineTo(TransmitterPos.X, TransmitterPos.Y);
}

```

Для работы приемника понадобятся две глобальные переменные. В переменную `TransmitterPos` мы станем записывать поступающие от приложения-передатчика экранные координаты, в `WM_MSG` после инициализации приложения окажется числовой идентификатор сообщения «WM_SENDPOSITION». Обработка поступающих сообщений производится в рамках метода `OnMessage_Event()`. На этот раз мы отказались от

применения в проекте компонента `TApplicationEvents` и обеспечили реакцию на событие «вручную».

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19², клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Разработайте три приложения, демонстрирующие порядок обмена сообщениями между разными процессами.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. По окончании выполнения задания письменный отчёт о выполненной работе предоставляется для защиты преподавателю.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. Опишите состав структуры, отвечающей за отправку сообщений в Windows?
2. Для чего приложению нужен цикл обработки сообщений?
3. Как получить дескриптор окна для отправки ему сообщения?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [5];
дополнительная литература – [2].

Лабораторная работа 32

Сетевое взаимодействие с помощью почтовых слотов

Цель работы: исследование способов сетевого взаимодействия с помощью почтовых слотов.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Своим названием технология почтовых слотов обязана схожестью принципа работы с обычным почтовым ящиком. Серверное приложение имеет возможность создать собственный почтовый ящик, корреспонденцию в который имеет возможность отправить любой клиент (при условии обладания адресом слота).

Механизм mailslots односторонний. Передаваемое в виде дейтаграммы сообщение, попав (или не попав) в предназначенный для него почтовый слот, не уведомляет отправителя, что оно прибыло по назначению. Так что о гарантированной доставке почты речи идти не может.

Почтовые слоты часто используются для широковещательной рассылки небольших сообщений. Надо знать, что размер почтового отправления ограничен – он не должен превышать 424 байта.

Организация работы с почтовым слотом сравнительно проста. Сервер регистрирует файл-слот и размещает его в памяти компьютера. Правом читать данные из слота также обладает только сервер, причем для доступа к слоту применяются методы, весьма напоминающие процесс чтения из обычного файла. В свою очередь клиент имеет право лишь отправлять сообщения в почтовый слот.

Определение имени почтового слота. Во время регистрации почтового слота ему присваивается уникальный сетевой адрес. Существует правило, согласно которому склеивается строка с текстом адреса:

- ~ адрес начинается с двух наклонных черт, затем идет точка, вновь наклонная черта и ключевое слово «mailslot»;
- ~ необязательный путь, также начинающийся с наклонной черты;
- ~ адрес завершается наклонной чертой и именем слота.

Например, `const wchar_t *MailSlotName = L"\\\\.\\mailslot\\sample_slot"`. При обращении к почтовому ящику другого компьютера алгоритм создания адресной строки незначительно изменяется. В этом случае строка также начинается с двух наклонных черт, но теперь вместо точки указывается сетевое имя компьютера, к которому отправляется корреспонденция или имя домена, если мы одновременно обращаемся к целой группе компьютеров (листинг 32.1).

Листинг 32.1 – Исходный код приложения-приемника

```
\\имя_компьютера\mailslot\[путь]имя_слота
\\имя_домена\mailslot\[путь]имя_слота
```

Управление почтовым слотом. За создание почтового ящика отвечает приложение-сервер, оно обращается к функции прикладного программного

интерфейса Windows HANDLE WINAPI CreateMailslot(__in LPCTSTR lpName, __in DWORD nMaxMessageSize, __in DWORD lReadTimeout, __in_opt LPSECURITY_ATTRIBUTES lpSecurityAttributes). Здесь lpName – указатель на строку с адресом слота. nMaxMessageSize – ограничитель максимальной длины послания в байтах, если параметру присвоить нулевое значение, то ограничения снимаются. Очередной параметр lReadTimeout оказывает влияние на процесс чтения корреспонденции из создаваемого ящика. Он задает время ожидания в миллисекундах. Если передается 0, то при обращении к пустому ящику функция, читающая корреспонденцию, возвратит результат немедленно. В противном случае она выждет установленный таймаут в надежде, что за этот промежуток времени нам кто-нибудь напишет. Для вечного ожидания передайте константу MAILSLOT_WAIT_FOREVER. Последний параметр lpSecurityAttributes – указатель на структуру атрибутов безопасности, в простейшем случае в него передают NULL. В случае успешного выполнения метод возвращает дескриптор созданного слота, иначе – нулевое значение.

Кроме функции создания почтового слота, Windows API способен похвастаться еще двумя функциями работы с mailslot. Во-первых, это функция, позволяющая настраивать время таймаута lReadTimeOut в слоте, заданном дескриптором hMailSlot: BOOL WINAPI SetMailslotInfo(__in HANDLE hMailslot, __in DWORD lReadTimeout). Во-вторых, это функция, информирующая о состоянии почтового слота: BOOL WINAPI GetMailslotInfo(__in HANDLE hMailslot, __out LPDWORD lpMaxMessageSize, __out_opt LPDWORD lpNextSize, __out_opt LPDWORD lpMessageCount, __out_opt LPDWORD lpReadTimeout). Эта функция вновь требует указать дескриптор интересующего нас слота в параметре hMailslot. А в результате метод выдаст данные о: lpMaxMessageSize – максимальном размере сообщения в байтах; lpNextSize – размере следующего сообщения, и если такового нет, то в

параметре окажется константа `MAILSLOT_NO_MESSAGE`; `lpMessageCount` – общем числе сообщений в слоте; `lpReadTimeout` – выбранном времени таймаута почтового ящика.

Получение и отправка корреспонденции. Чтение находящихся в почтовом слоте данных возможно только из программы-сервера, создавшей этот самый слот. Зная дескриптор слота, приложение обращается к нему, как к обыкновенному файлу. Для операции чтения подходят как функции Windows API `ReadFile()` и `ReadFileEx()`, так и их наследник – функция из состава VCL `FileRead()`.

Для отправки почты клиентское приложение просто осуществляет запись в виртуальный файл, олицетворяющий почтовый слот сервера-получателя. Обычно для этого применяют методы `FileCreate()`, `FileWrite()` и `FileClose()`.

Пример почтового приложения. Программирование начнем с регистрации почтового слота. Для этого объявите константу `MailSlotName`, в которую мы внесем имя почтового слота, и обратитесь к событию `OnCreate()`, вызываемому в момент создания главной формы (листинг 32.2).

Листинг 32.2 – Создание приложения и регистрация слота

```
const wchar_t *MailSlotName=L"\\\\.\\mailslot\\sample_slot";
HANDLE hMailSlot;
void __fastcall TForm1::FormCreate(TObject *Sender){
    wchar_t buf[MAX_COMPUTERNAME_LENGTH];
    hMailSlot=CreateMailslot(MailSlotName, 0, MAILSLOT_WAIT_FOREVER,
    NULL);
    if (hMailSlot==INVALID_HANDLE_VALUE) { //если вызов метода
    завершился неудачей
        unsigned ErrorCode=GetLastError(); //узнаём код ошибки
        if (ErrorCode!=0) {
            String s=SysErrorMessage(ErrorCode); //расшифровка ошибки
            String err=Format("Ошибка %d %s",ARRAYOFCONST((ErrorCode,
            s)));
            MessageBox(Form1->Handle, err.w_str(), L"Ошибка",
            MB_ICONERROR |MB_OK);
        }
    }
```

```

}
unsigned long w;
GetComputerName(buf, &w);
StatusBar1->Panels->Items[0]->Text="\\\\\\"+StrPas(buf);
}

```

После создания жизненный цикл почтового слота будет продолжаться до тех пор, пока мы не обратимся к функции `CloseHandle()`, освобождающей экземпляр слота. Эту операцию мы проведем в момент закрытия главной формы (листинг 32.3).

Листинг 32.3 – Завершение работы приложения

```

void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction
&Action) {
    CloseHandle(hMailSlot);
}

```

Разработаем метод чтения почты из слота. Для этого в секции частных объявлений опишем заголовок метода `ReadMail()`, обладающий единственным параметром `aMailSlot`, в который следует передавать дескриптор почтового ящика, откуда будет прочитано почтовое отправление. Реализация метода представлена в листинге 32.4.

Листинг 32.4 – Получение почты

```

void __fastcall TForm1::ReadMail(HANDLE aMailSlot) {
unsigned long NextSize, MessageCount;
if (GetMailslotInfo(aMailSlot, NULL, &NextSize, &MessageCount,
NULL)) {
while (MessageCount>0) { //собираем все сообщения из ящика
    char *Buf=(char *)malloc(NextSize+1); //распределяем память
    DWORD NumberOfBytesRead;
    ReadFile(aMailSlot, Buf, NextSize + 1, &NumberOfBytesRead, NULL);
    MemoIn->Lines->Add(StrPas(Buf)); //сообщения в мемо-поле
    free(Buf); //освобождаем память
    MessageCount--; //уменьшаем счётчик сообщений
  } }
}

```

Технология почтовых слотов не предусматривает возможности автоматического контроля за получением корреспонденции. Поэтому, для того чтобы организовать динамический просмотр почтового ящика, в приложении-сервере придется применить таймер или поток.

В секции частных объявлений опишите очередной метод `SendMail()`. Новый метод возьмет ответственность за отправку текстового сообщения `MailText` корреспонденту с адресом `MailAddres`. Реализация метода предложена в листинге 32.5.

Листинг 32.5 – Отправка почты

```
void __fastcall TForm1::SendMail(String MailAddres,String
MailText){
char *Buf;
unsigned hFile=FileCreate(MailAddres);/*обращаемся к слоту как к
файлу используя имя слота*/
if (hFile>0){ //если получилось обратиться то:
    char *Buf=(char *)malloc(MailText.Length()+1);
    ZeroMemory(Buf,sizeof(Buf));
    StrPCopy(Buf,MailText);//помещаем в буфер сообщение
    FileWrite(hFile,Buf,MailText.Length()+1);//отправляем сообщение
    в файл
    FileClose(hFile); //закрываем файл
    SysFreeMem(Buf); //освобождаем буфер
}
}
```

Приложение уже почти готово. Нам осталось дважды щелкнуть по менеджеру команд `ActionManager1` и создать в нем команды `acSendMail` и `acReadMail`, отвечающие соответственно за отправку и получение почты. Листинг 32.6 раскрывает функциональную нагрузку этих команд.

Листинг 32.6 – Управление приложением

```
void __fastcall TForm1::acReadMailExecute(TObject *Sender){
ReadMail(hMailSlot);
}
void __fastcall TForm1::acSendMailExecute(TObject *Sender){
```

```

String MailAddress=MailSlotName;
MailAddress=cbComputers->Text+MailAddress.Delete(1,3);
SendMail(MailAddress, StatusBar1->Panels->Items[0]->Text+":
"+MemoOut->Text);
MemoOut->Clear();
}

```

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024 × 768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19², клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Разработайте приложение сетевого взаимодействия используя технологию почтовых слотов для Windows. Приложение должно позволять обмениваться данными в соответствии с вариантами.

Варианты к заданию

1. Текстовые строки.
2. Целочисленные значения.
3. Вещественные значения.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. По окончании выполнения задания письменный отчёт о выполненной работе предоставляется для защиты преподавателю.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. На каком уровне модели OSI работает технология почтовых слотов?
2. Возможна ли в технологии почтовых слотов широковещательная рассылка почты?
3. Какой максимальный размер текстового сообщения поддерживает технология почтовых слотов?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [1, 5];
дополнительная литература – [2].

Лабораторная работа 33

Сетевое взаимодействие с помощью именованных каналов

Цель работы: исследование способов сетевого взаимодействия с помощью именованных каналов.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Выбор термина «канал» (pipe) обусловлен тем, что связь между приложениями осуществляется по некоторому виртуальному каналу, проложенному в сети между двумя компьютерами. Каналы классифицируют как *неименованные* (anonymous pipe) и *именованные* (named pipe).

Определение имени именованного канала. Для определения имени канала во время разработки сервера программист должен придерживаться следующего синтаксиса: `\\.\pipe\<имя_канала>`.

Имя канала не чувствительно к регистру символов и не должно содержать более 256 символов. Например: `const wchar_t *PipeName = L"\\\\.\\pipe\\mypipe".`

При обращении к именованному каналу клиентские приложения заменяют точку в его имени на сетевое имя сервера, предоставляющего услугу именованных каналов: `const wchar_t *PipeName = L"\\\\mainserver\\pipe\\mypipe".`

Разработка класса сервера именованного канала. Листинг 33.1 содержит определение класса `TPipeServer`, способного создать отдельный именованный канал. Канал создается в рамках конструктора класса (листинг 33.2).

Листинг 33.1 – Объявление класса сервера именованного канала

```
class TPipeServer{
    private: HANDLE fHandle;
             _OVERLAPPED Overlapped;
             unsigned fTimeout; // = 0
    public:
        __fastcall TPipeServer(String PipeName, DWORD
Buffer Size);
        __fastcall ~TPipeServer();
        bool __fastcall Connect();
        int __fastcall Write(void *buf, int Count);
        int __fastcall Read(void *buf, int Count);
        String __fastcall GetClientComputerName();
        HANDLE __property Handle= {read=fHandle};
        unsigned __property Timeout= {read=fTimeout,
write=fTimeout};
};
```

Листинг 33.2 – Конструктор класса сервера именованного канала

```
__fastcall TPipeServer::TPipeServer(String PipeName, DWORD
Buffer Size){
    _SECURITY_DESCRIPTOR SD;
    InitializeSecurityDescriptor(&SD, SECURITY_DESCRIPTOR_REVISION);
    _SECURITY_ATTRIBUTES SA;
    SA.nLength=sizeof(_SECURITY_ATTRIBUTES);
```

```

SA.lpSecurityDescriptor=&SD;
SA.bInheritHandle=true;
ZeroMemory(&fOverlapped, sizeof(_OVERLAPPED));
fOverlapped.hEvent= CreateEvent (NULL, true, false, NULL);
BufferSize=4096;
TimeOut=0;
fHandle = CreateNamedPipe(PipeName.w_str(),
    PIPE_ACCESS_DUPLEX | FILE_FLAG_OVERLAPPED,
    PIPE_WAIT | PIPE_READMODE_MESSAGE |
PIPE_TYPE_MESSAGE,
    PIPE_UNLIMITED_INSTANCES,
    BufferSize,
    BufferSize,
    TimeOut,
    &SA);
if (Handle == INVALID_HANDLE_VALUE | Handle==0)
    { throw (Exception("Ошибка создания канала!")); }
}

```

После создания экземпляра канала наступает очередь метода `Connect()`, отвечающего за перевод сервера в режим готовности к соединению (листинг 33.3).

Листинг 33.3 – Метод соединения

```

bool __fastcall TPipeServer::Connect() {
bool Result=ConnectNamedPipe(Handle,&fOverlapped);
if (Result==false)
    switch (GetLastError())
    {
        case ERROR_PIPE_CONNECTED: Result= true; break;
        case ERROR_IO_PENDING      :
            if (WaitForSingleObject(fOverlapped.hEvent, INFINITE)
== WAIT_OBJECT_0) Result= true; break;
        default                  : throw Exception("Не готов к
соединению!");
    }
return (Result);
}

```

Настал черед методов, отвечающих за операции чтения/записи в канал. Для этого достаточно задействовать классические для Windows API функции чтения и записи в файл (листинги 33.4 и 33.5).

Листинг 33.4 – Метод чтения данных

```
int __fastcall TPipeServer::Read(void *buf, int Count){
unsigned long NumberOfBytesToRead=0;
if (!ReadFile(Handle, buf, Count, &NumberOfBytesToRead,
&fOverlapped)){
    switch (GetLastError()){
    { case ERROR_IO_PENDING:{
        WaitForSingleObject(fOverlapped.hEvent, INFINITE);
        GetOverlappedResult(Handle, &fOverlapped,
&NumberOfBytesToRead, true);
        break;}
    case ERROR_BROKEN_PIPE: { throw Exception("Канал
разорван!"); break; }
    default : throw Exception("Сбой чтения, ошибка"); break;
    }
};
return (NumberOfBytesToRead);
}
```

Листинг 33.5 – Метод записи данных

```
int __fastcall TPipeServer::Write(void *buf, int Count){
unsigned long NumberOfBytesToWrite=0;
if (!WriteFile(Handle, buf, Count, &NumberOfBytesToWrite,
&fOverlapped)){
    switch (GetLastError()){
    { case ERROR_IO_PENDING :
        WaitForSingleObject (fOverlapped.hEvent, INFINITE);
break;
    case ERROR_NO_DATA      : throw Exception("Ошибка записи");
break;
    }
    }
    GetOverlappedResult(Handle, &fOverlapped, &NumberOfBytesToWrite,
true);
return (NumberOfBytesToWrite);
}
```

Работа над классом завершается описанием канала деструктором (листинг 33.6). В коде деструктора предусматривается не только уничтожение экземпляра канала, но и очистка буфера с данными и разрушение события синхронизации.

Листинг 33.6 – Деструктор класса сервера именованного канала

```
__fastcall TPipeServer::~TPipeServer()
{
    FlushFileBuffers(Handle);
    DisconnectNamedPipe(Handle);
    CloseHandle(fOverlapped.hEvent);
    CloseHandle(Handle);
}
```

Разработка класса клиента именованного канала. Разработаем класс TPipeClient, предназначенный для обслуживания именованного канала на клиентской стороне. Клиентский класс более прост, т.к. пользуется услугами уже существующего канала, поэтому кроме конструктора и деструктора нам потребуются всего два метода чтения и записи (листинг 33.7).

Листинг 33.7 – Объявление класса клиента именованного канала

```
class TPipeClient
{
private:    HANDLE fHandle;
             _OVERLAPPED OverlappedRead, OverlappedWrite;
public:
    __fastcall TPipeClient(String PipeName);
    __fastcall ~TPipeClient();
    int __fastcall Write(void *buf, int Count);
    int __fastcall Read(void *buf, int Count);
    HANDLE __property Handle= {read=fHandle};
};
```

Основная особенность клиента именованного канала в том, что он работает с каналом так, как будто это обычный файл. Этим объясняется вся логика функционирования класса TPipeClient. В момент создания

экземпляра класса мы «обманиваем» клиента, предоставляя в его распоряжение дескриптор канала под видом дескриптора файла (листинг 33.8).

Листинг 33.8 – Конструктор класса клиента именованного канала

```
__fastcall TPipeClient::TPipeClient(String PipeName){
fHandle = CreateFile(PipeName.w_str(), GENERIC_READ |
GENERIC_WRITE, 0, NULL, OPEN_EXISTING, FILE_FLAG_OVERLAPPED, 0);
String msg;
if (Handle==NULL)
    switch (GetLastError()){
        case ERROR_SEEK_ON_DEVICE : {
            msg=Format("Канал %s не обнаружен",
ARRAYOFCONST((PipeName)));
            throw (Exception(msg));
            break;
        }
        case ERROR_FILE_NOT_FOUND : {
            msg=Format("Канал %s не обнаружен",
ARRAYOFCONST((PipeName)));
            throw (Exception(msg));
            break;
        }
        default : {
            msg=Format("Ошибка канала!",
ARRAYOFCONST((PipeName)));
            throw (Exception(msg));
            break;
        }
    }
ZeroMemory(&OverlappedRead, sizeof(OverlappedRead));
ZeroMemory(&OverlappedWrite, sizeof(OverlappedWrite));
OverlappedRead.hEvent = CreateEvent(NULL, true, false, NULL);
OverlappedWrite.hEvent= CreateEvent(NULL, true, false, NULL);
}
```

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 Гб (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024 × 768 (или выше) с 24-битными (или

больше) цветами с диагональю не менее 19", клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Разработайте приложение сетевого взаимодействия используя технологию именованных каналов для Windows. Приложение должно позволять обмениваться данными в соответствии с вариантами.

Варианты к заданию

1. Текстовые строки.
2. Целочисленные значения.
3. Вещественные значения.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. По окончании выполнения задания письменный отчет о выполненной работе предоставляется для защиты преподавателю.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. На каком уровне модели OSI работает технология именованных каналов?

2. В каких известных вам программных продуктах применяются именованные каналы?

3. Какие ограничения на применение именованных каналов вам известны?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [5];
дополнительная литература – [2].

Лабораторная работа 34

Сетевое взаимодействие с помощью сокетов

Цель работы: исследование способов сетевого взаимодействия с помощью сокетов.

Знания, умения и владения, приобретаемые обучающимся в результате освоения темы в рамках формируемых компетенций:

Знает: языки и системы программирования; методы построения алгоритма и проведения его анализа; механизмы установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Умеет: корректно использовать современные технологии программирования для разработки ПО; разработать алгоритм и провести его анализ; осуществлять наладку и тестирование современного общего и специального ПО.

Владеет: современными технологиями программирования для разработки программного обеспечения; приемами разработки и анализа алгоритма; навыками установки, наладки, тестирования и обслуживания современного общего и специального программного обеспечения.

Теоретическая часть

Одним из наиболее востребованных механизмов взаимодействия сетевых приложений считается технология *сокетов* (sockets). Важное преимущество сокетов в том, что они поддерживаются практически всеми современными операционными системами и поэтому позволяют организовывать сетевой обмен между ОС различных производителей.

Технология WinSock построена по принципу «клиент-сервер». Сразу после создания сервер переходит в режим ожидания связи (для этого предложен термин «*прослушивание*» – listening). Услышав «стук в дверь»,

сервер открывает сеанс с клиентом, в рамках которого осуществляется односторонний или двусторонний обмен данными.

Сокет-клиент. Каким образом научить клиентское приложение получать экранный снимок рабочего стола? На самом деле, в этом нет ничего сложного. Нам следует получить дескриптор дисплея, узнать размеры рабочего стола и с помощью функции `BitBlt()` скопировать пиксели экрана в битовый образ, созданный функцией `CreateCompatibleBitmap()`. Код функции, решающей эту задачу, едва превышает 10 строк (листинг 34.1), а в результате мы получаем дескриптор битового образа.

Листинг 34.1 – Получение дескриптора битового образа со снимком рабочего стола

```
HBITMAP GetScreenBitmap() {
    HDC DC=CreateDC(L"DISPLAY",NULL, NULL, NULL); //дескриптор
экрана
    HDC memDC=CreateCompatibleDC(DC); //совместимый дескриптор в
памяти
    int W=GetDeviceCaps(DC,HORZRES); //горизонтальный размер
    int H=GetDeviceCaps(DC,VERTRES); //вертикальный размер
    HBITMAP BITMAP=CreateCompatibleBitmap(DC,W,H); //создаем
битовый образ
    SelectObject(memDC,BITMAP); //передаем образ в
память
    BitBlt(memDC,0,0,W,H,DC,0,0,SRCCOPY); //копируем экран в
память
    DeleteDC(memDC);
    DeleteDC(DC);
    return (BITMAP);
}
```

После этого нам осталось написать функцию, позволяющую при посредничестве компонента `TCPCClient * TCPClient1` отправлять по заданному адресу снимок экрана. При разработке клиентской функции необходимо учесть тот факт, что растровая картинка 19-дюймового экрана с разрешением 1280x1024 пикселей с глубиной цвета 24 бит/пиксель займет

более 3,5 Мбайт памяти, поэтому нам следует задуматься над решением проблемы снижения передаваемого в сеть потока данных. В листинге 34.2 для этих целей мы воспользовались услугами графического формата JPEG, который позволяет существенно сжимать растровые рисунки.

Листинг 34.2 – Передача графических данных в адрес серверного сокета

```
const BufSize=0x3FFF; //размер буфера 2^18
void __fastcall TfrmClient::SendScreenShot(String RemoteHost,
String RemotePort){
    TcpClient1->RemoteHost=RemoteHost;
    TcpClient1->RemotePort=RemotePort;
    TcpClient1->Open();
    TBitmap *B;
    TJPEGImage *J;
    TMemoryStream *MS;
    if (TcpClient1->Connected==true){
        __try
        {
            B=new TBitmap;
            B->Handle=GetScreenBitmap();
            J=new TJPEGImage(); //с помощью JPEG сжимаем снимок
            J->Assign(B); //забираем исходную картинку
            J->ProgressiveEncoding=true; //прогрессивная кодировка
            J->CompressionQuality=50; //степень сжатия
            J->JPEGNeeded(); //сжимаем образ
            MS=new TMemoryStream();
            J->SaveToStream(MS); //загружаем снимок в поток MS
            MS->Seek(0,0); //позиционируемся на первом байте потока
            unsigned char Buf[BufSize];
            MS->ReadBuffer(Buf,MS->Size); //читаем данные из потока
            TcpClient1->SendBuf(Buf,MS->Size);
        }
        __finally{
            if (B!=NULL) delete B;
            if (J!=NULL) delete J;
            if (MS!=NULL) delete MS;
        }
    }
    TcpClient1->Close();
}
```

Сокет-сервер. Для реализации приложения-сервера нам потребуются три ингредиента:

- ~ сокет-сервер `TTCPServer`. Проконтролируйте, чтобы его свойство `BlockMode` было установлено в состояние `bmThreadBlocking`;
- ~ компонент `TImage* Image1`, способный отобразить поступивший из сети рисунок со снимком рабочего стола;
- ~ отдельный поток `class TJPEGThread : public TThread`, который позволит бесконфликтно передавать поступившие от клиентских приложений снимки в `Image1`.

Реализацию сервера начнем с объявления потока `TJPEGThread`. Это мы сделаем прямо в коде программного модуля `main.cpp`, соответствующего главной форме приложения `frmMain` (листинг 34.3).

Листинг 34.3 – Фрагмент модуля главной формы с объявлением потока

```
#include <Vcl.Imaging.jpeg.hpp>
class TJPEGThread : public TThread{
private:
protected:
    void __fastcall Execute();
public:
    TStringList *ListBuffer;
    TMemoryStream *MS;
    TfrmChild *frmChild;
    __fastcall TJPEGThread(bool CreateSuspended);
    __fastcall ~TJPEGThread();
    void __fastcall GetJpeg();
};
```

Помимо конструктора, деструктора и обязательного для потока метода `Execute()` объявление потока `TJPEGThread` включает функцию `GetJpeg()`, предназначенную для загрузки клиентского экранного снимка в компонент `Image1` (листинг 34.4).

Листинг 34.4 – Реализация потока `TJPEGThread`

```

__fastcall TJPEGThread::TJPEGThread(bool CreateSuspended) :
TThread(CreateSuspended) {
    MS=new TMemoryStream();
    FreeOnTerminate=true;
}
__fastcall TJPEGThread::~TJPEGThread() {
    delete MS;
    frmChild=NULL;
}
void __fastcall TJPEGThread::GetJpeg() {
    TJPEGImage *JPG;
    MS->Position=0;
    if (MS->Size>0)
        __try
        {
            JPG=new TJPEGImage();
            JPG->ProgressiveDisplay=true;
            JPG->LoadFromStream(MS);
            JPG->DIBNeeded();
            Image1->Picture->Bitmap->Assign(JPG);
        }
        __finally{
            delete JPG;
        }
}
void __fastcall TJPEGThread::Execute() {
    Synchronize(GetJpeg);
    Terminate();
}

```

Для понимания работы сервера нам осталось рассмотреть событие `OnAccept()`, возникающее у компонента `TcpServer1` в момент поступления заявки на обслуживание из сети от клиентского приложения (листинг 34.5).

Листинг 34.5 – Обслуживание клиентского приложения

```

const BufSize=0x3FFFF;
void __fastcall TfrmMain::TcpServer1Accept(TObject *Sender,
TCustomIpClient *ClientSocket){
    unsigned char Buf[BufSize];
    int x;
    TJPEGThread *JPEGThread=new TJPEGThread(true);
    x=ClientSocket->ReceiveBuf(Buf, sizeof(Buf));
    JPEGThread->MS->Write(Buf,x);
}

```

```
JPEGThread->Resume();  
}
```

Оборудование и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор 32-битный или 64-битный (x86-64, EM64T) с тактовой частотой 1,5 ГГц или выше (рекомендуется 2 ГГц), оперативная память – 2 ГБ (рекомендуется 4 Гб), свободное дисковое пространство – 1,75 Гб, монитор типа Super XGA с разрешением 1024×768 (или выше) с 24-битными (или больше) цветами с диагональю не менее 19², клавиатура, мышь.

Программное обеспечение – операционная система семейства Windows или Linux, среда программирования (по выбору студента).

Задание

Разработайте приложение сетевого взаимодействия используя технологию сокетов Windows. Приложение должно позволять обмениваться данными в соответствии с вариантами.

Варианты к заданию

1. Текстовые строки.
2. Целочисленные значения.
3. Вещественные значения.

Указания по порядку выполнения работы

Для выполнения задания необходимо изучить теоретическую часть к лабораторной работе и листинги с примерами исходного кода. По окончании выполнения задания письменный отчет о выполненной работе предоставляется для защиты преподавателю.

Содержание отчета

Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Ответов на контрольные вопросы.
3. Формулировки задания и порядка его выполнения.

Контрольные вопросы

1. На каком уровне модели OSI работает технология сокетов?
2. В каких операционных системах поддерживается технология сокетов?
3. Какие ограничения на применение именованных каналов вам известны?

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой. В частности: самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории; соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Список литературы, рекомендуемый к использованию по данной теме:

основная литература – [5];
дополнительная литература – [2].

СПИСОК ЛИТЕРАТУРЫ

Перечень основной литературы

1. Кнут Д. Искусство программирования для ЭВМ. т.1,2,3, Сортировка и поиск. - М.: Вильямс, Addison Wesley Longman, 2000 г.
2. Седжвик. Р. Фундаментальные алгоритмы на C++. Анализ/Структуры данных/Сортировка/Поиск: Пер. с англ./Роберт Седжвик. - К.: Издательство «ДиаСофт», 2001.- 688 с.
3. Вирт Н. Алгоритмы и структуры данных: Пер. с англ. – 2-е изд., испр. – СПб.: Невский Диалект, 2005. – 352 с.: ил.
4. Кормен. Т., Леверсон Ч., Ривест Р. Алгоритмы. Построение и анализ, 2-е издание.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2005. – 1296 с.: ил..
5. Орлов С. А. Программная инженерия. Технологии разработки программного обеспечения – Учебник для вузов. 5-е издание обновленное и дополненное – СПб.: Питер, 2021. – 640 с.

Перечень дополнительной литературы

1. Макконелл Дж. Основы современных алгоритмов. 2-е дополненное издание. – М.: Техносфера, 2004. – 370 с.
2. Осипов Д. Л. C++ Builder. Основы программирования. – [б.м.]: Издательские решения, 2016. – 780 с..
3. Уэзерелл Ч. Этюды для программистов: Пер. с англ. – М.: Мир, 1982. – 288 с.,ил.
4. Пореев В.Н. Компьютерная графика. – СПб.: БХВ-Петербург, 2004. – 432 с.: ил.
5. Майкл Ласло. Вычислительная геометрия и компьютерная графика на C++: Пер. с англ. – М.: «Издательство БИНОМ», 1997. – 304 с. ил.

6. Гагарина Л.Г. Технология разработки программного обеспечения. – ИНФРА-М, 2008. - 400 с.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по проведению практических занятий
по дисциплине «ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ»
для студентов специальности 10.05.01 Компьютерная безопасность
специализация «Информационно-аналитическая и техническая
экспертиза компьютерных систем»

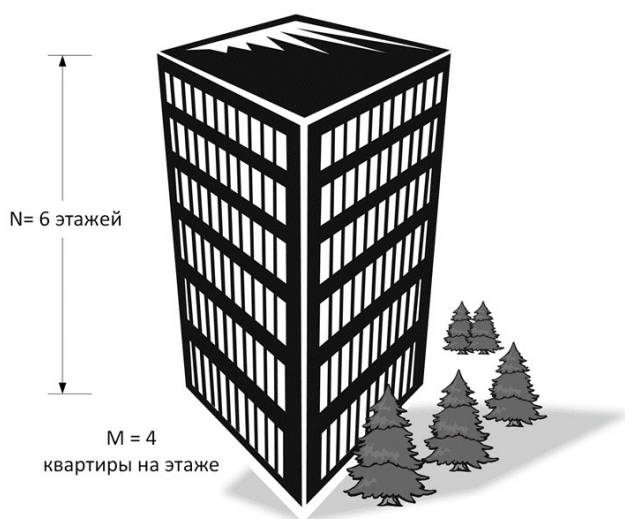
Ставрополь
2026

Практическое занятие № 1

Ваша консольная программа должна обладать интерактивным интерфейсом, для этого вспомните каким образом в языке Си осуществляются операции ввода-вывода.

Задание 1

Дано: дом с одним подъездом высотой в 6 этажей, на лестничной площадке по 4 квартиры.



Напишите программу для лифта, который после нажатия пассажиром кнопки выбора этажа высвечивает номера квартир на данном этаже, например так

Этаж 1 → 1 – 4

Этаж 2 → 5 – 8

Задание 2

Дано: решите задачу обратную заданию 1. На этот раз ваша программа должна подсказывать пользователю на каком этаже расположена та или иная квартира, например так

Кв 1 → 1 этаж

Кв 8 → 2 этаж

Задание 3

Дано: N-этажный дом с K-подъездами, на лестничной площадке по M квартир. Ваша программа должна подсказывать пользователю в каком подъезде и на каком этаже расположена та или иная квартира. Ваша программа должна предусматривать ввод пользователем исходных данных и вывод результата расчетов на экран.

Задание по вариантам

Напишите на языке Си программы осуществляющие следующие расчеты:

Вариант	Задание 1	Задание 2	Задание 3
1	$\frac{x^2 + 3x + y}{ax + e^y}$	$\frac{5,23 + 7,6^x + \sin \frac{\pi}{4}}{\sin \frac{2\pi}{7} + 3,1}$	$2^{-x} \sqrt{x + \sqrt[4]{ x }}$
2	$\frac{c + d - x}{ x + d}$	$6,673 \times 10^{-8} \frac{m_1 m_2}{r^2}$	$\sqrt{ x - 1 + \sin(x)}$
3	$\sqrt[8]{x^8 + 8^x}$	$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!}$	$\frac{xyz - 3,3 x + \sqrt[4]{y} }{10^7 + \sqrt{\lg 4!}}$
4	$\frac{xyz - 3,3 x + \sqrt[4]{y} }{10^7 + \sqrt{\lg 4!}}$	$x \cos x + \sin^3 x$	$6,673 \times 10^{-8} \frac{m_1 m_2}{r^2}$
5	$\frac{\beta + \sin^2 \pi^4}{\cos 2 + \operatorname{ctg} \gamma }$	$1 + \frac{1}{x} + \frac{1}{x^2}$	$\frac{c + d - x}{ x + d}$
6	$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!}$	$\sqrt[8]{x^8 + 8^x}$	$\frac{5,23 + 7,6^x + \sin \frac{\pi}{4}}{\sin \frac{2\pi}{7} + 3,1}$
7	$6,673 \times 10^{-8} \frac{m_1 m_2}{r^2}$	$2^{-x} \sqrt{x + \sqrt[4]{ x }}$	$\frac{\beta + \sin^2 \pi^4}{\cos 2 + \operatorname{ctg} \gamma }$
8	$e^{ x-y } + \ln(1 + e) \log_2 \operatorname{tg} 2$	$\frac{c + d - x}{ x + d}$	$\sqrt[8]{x^8 + 8^x}$
9	$\sqrt{ x - 1 + \sin(x)}$	$\frac{xyz - 3,3 x + \sqrt[4]{y} }{10^7 + \sqrt{\lg 4!}}$	$e^{ x-y } + \ln(1 + e) \log_2 \operatorname{tg} 2$
10	$2^{-x} \sqrt{x + \sqrt[4]{ x }}$	$\frac{\beta + \sin^2 \pi^4}{\cos 2 + \operatorname{ctg} \gamma }$	$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!}$

Практическое занятие № 2

1. Объявите одномерный массив типа данных T (см. таблицу) размерности N.
2. Заполните массив случайными значениями в диапазоне от MIN до MAX.
3. Выведите массив на экран.
4. Используя цикл for найдите сумму элементов массива.
5. Используя цикл while..do найдите среднее арифметическое значение из хранимых в массиве значений.
6. Используя цикл do..while найдите в массиве самое большое и самое маленькое значение.
7. Объявите 2-хмерный массив типа данных T размерности NxN.
8. Используя любой цикл, заполните 2-хмерный массив следующим образом:
 - a. В главную диагональ поместите значение $(MIN+MAX)/2$;
 - b. Ниже главной диагонали поместите значение MIN
 - c. Выше главной диагонали поместите значение MAX

Вариант	Тип данных T	Размерность N	MIN-MAX
1	T = short int	50	-20...20
2	T = int	20	-1000...1000
3	T = float	20	-10,00...10,00
4	T = double	20	-1,00...1,00
5	T = unsigned short int	255	10...90
6	T = unsigned int	50	0..100
7	T = short int	50	-25...25
8	T = unsigned short int	25	0..50
9	T = long int	25	-100..100
10	T = unsigned long int	25	0..200

Общее задание:

Напишите консольное приложение-калькулятор, которое позволит пользователю осуществлять следующие операции:

1. Сложение, вычитание, деление, умножение.
2. Возведение в квадрат.

3. Взятие квадратного корня.
4. Инвертировать введенное число (менять знак на противоположный).
5. Возвращать число π

Практическое занятие № 3

В качестве исходных данных воспользуйтесь одномерным массивом типа данных T размерности N (см. задание своего варианта в ЛР №2).

Напишите следующие функции:

1. Функция вывода содержимого массива на экран.
2. Функция, возвращающая сумму элементов массива.
3. Функция, возвращающая среднее арифметическое элементов массива.
4. Функция, замещающая значения всех элементов массива на значение типа данных T, передаваемое через аргумент функции.
5. Функция, изменяющая порядок следования элементов массива на обратный (элемент с индексом 0 попадает в ячейку N-1, 2 в N-2, и т.д.).
6. Разработайте шаблон функции, работающей с массивом любого типа данных. Задача функции – найти и вернуть индекс и значение минимального элемента в массиве.
7. Разработайте шаблон функции, работающей с массивом любого типа данных. Задача функции – любым известным вам способом упорядочить элементы массива по возрастанию.
8. Напишите функцию на вход которой подается произвольный массив символов `unsigned char`, функция должна преобразовать символы в их числовой код и вернуть сумму этих чисел.

Доступ к массиву должен осуществляться только через параметры функции.

Практическое занятие № 4

1. Напишите консольное приложение, позволяющее пользователю:
 - a. Создавать новый текстовый файл, путь к файлу и его имя вводятся пользователем с клавиатуры.
 - b. Открывать существующий текстовый файл для просмотра (вывода всего текста на экран), путь к файлу и его имя вводятся пользователем с клавиатуры.
 - c. Добавлять в конец открытого файла новые строки.
 - d. Удалять строки (пользователь указывает номер удаляемой строки).
2. Напишите консольное приложение, предоставляющее пользователю возможность работать с бинарным файлом, предназначенным для хранения данных типа T (см. табл.). Приложение должно позволять:
 - a. Создавать новый бинарный файл, путь к файлу и его имя вводятся пользователем с клавиатуры.
 - b. Открывать существующий файл для просмотра (вывода данных на экран), путь к файлу и его имя вводятся пользователем с клавиатуры.
 - c. Добавлять в любое место файла (номер позиции выбирается пользователем) новое значение.
 - d. Удалять из любого места файла (номер позиции выбирается пользователем) хранящееся там значение.

Вариант	Тип данных T
1	T = short int
2	T = int
3	T = float
4	T = double
5	T = unsigned short int
6	T = unsigned int
7	T = short int
8	T = unsigned char
9	T = char
10	T = unsigned long int

3. Напишите консольное приложение, позволяющее пользователю просмотреть состав папок и файлов, хранящихся в указанном им каталоге (путь к каталогу вводится с клавиатуры). На экран выводятся:
 - a. имена папок и даты их создания
 - b. имена файлов, даты их создания и размер.

Практическое занятие № 5

1. Разработайте класс для работы с файлами типа данных T. Он должен позволять осуществлять следующие операции с файлом:
 - a. Создавать новый или открывать существующий файл с именем определяемым пользователем.
 - b. Возвращать размер открытого файла.
 - c. Читать данные из любой позиции файла.
 - d. Записывать в файл новые данные.
 - e. Редактировать данные в любом месте файла.
 - f. Удалять данные из любого места файла.

Вариант	Тип данных T
1	T = short int
2	T = int
3	T = float
4	T = double
5	T = unsigned short int
6	T = unsigned int
7	T = short int
8	T = unsigned char
9	T = char
10	T = unsigned long int

2. Разработайте класс-наследник развивающий функционал класса из первого задания. Класс-наследник должен позволять:
 - a. Удалять подключенный файл с диска.
 - b. Перемещать подключенный файл в другое место на диске.
 - c. Присваивать файлу атрибут “ТОЛЬКО ДЛЯ ЧТЕНИЯ”, что автоматически запретит операции редактирования и удаления данных классом.
 - d. Снимать с файла атрибут “ТОЛЬКО ДЛЯ ЧТЕНИЯ”, что восстановит функциональность класса.

Практическое занятие № 6

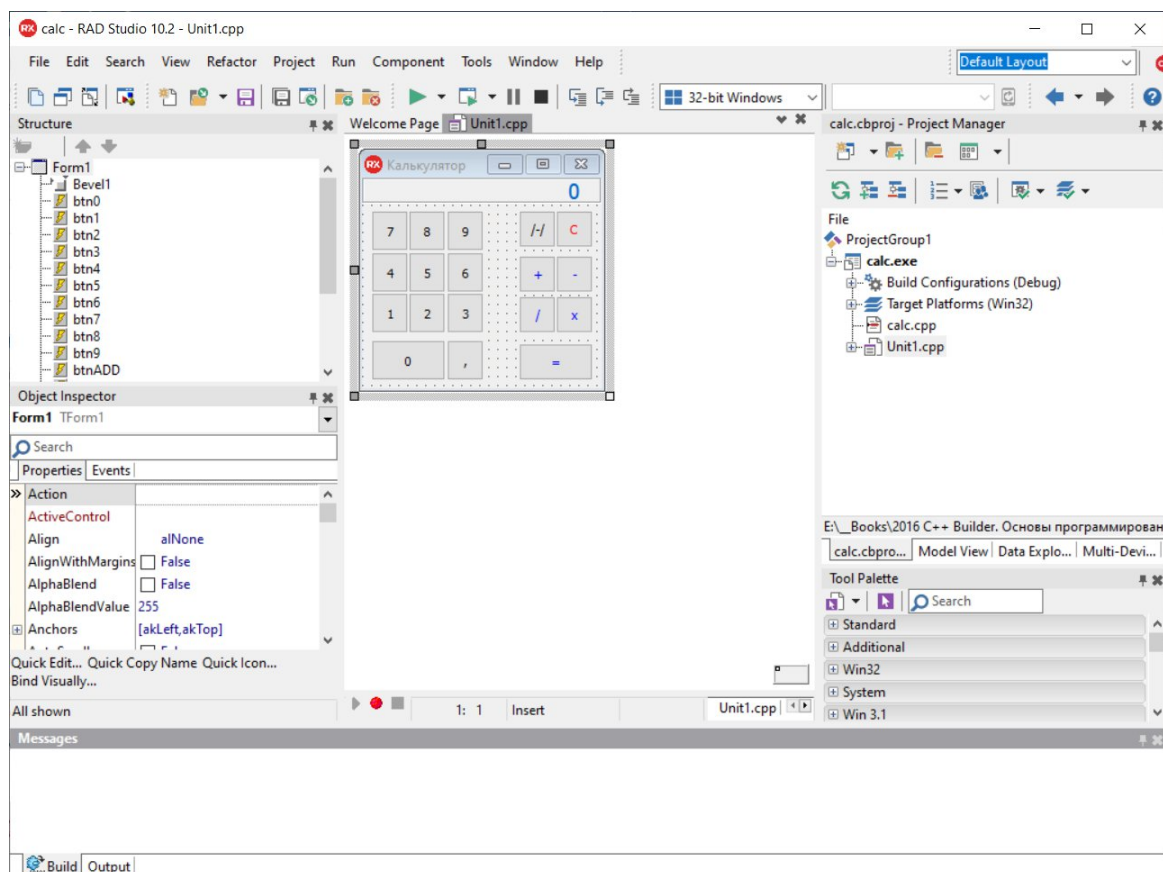
Разработайте класс-шаблон, предназначенный для работы с одномерным массивом типа данных `<class T>` размерности `unsigned int N` хранящий значения из диапазона чисел `aMIN...aMax`. Массив хранится в виде поля класса в защищенной секции **protected**, доступ к элементам массива осуществляется только методами и свойствами класса.

1. В момент вызова конструктора класса в него передается размер создаваемого массива `N` и диапазон допустимых значений `aMIN...aMax`. После выделения памяти массив заполняется нулевыми значениями.
2. Класс должен обладать следующими методами:
 - a. **методом чтения и методом записи** значения в указанную ячейку.
 - b. **методом поиска минимального значения**, с одновременным возвратом индекса этого значения в массиве.
 - c. **методом поиска максимального значения**, с одновременным возвратом индекса этого значения в массиве.
 - d. **методом сортировки** значений в массиве по возрастанию, допускается подключение библиотечных методов.
3. Класс должен обладать следующими свойствами (property):
 - a. Свойство возвращающее число элементов в массиве (только для чтения).
 - b. Свойство возвращающее значение `aMIN` (только для чтения).
 - c. Свойство возвращающее значение `aMax` (только для чтения).

Разработайте приложение, демонстрирующее работу класса.

Практическое занятие № 7

Разработайте приложение “Калькулятор”, выполняющее основные математические операции



Создайте приложение VCL с одной формой. Для создания интерфейса приложения вам понадобятся следующие компоненты:

- Кнопка TButton
- Метка TLabel для вывода результатов расчетов

Совет. В момент щелчка по кнопке генерируется простейшее событие OnClick(). Возможно, в рамках этого события вам и придется писать код.

Предположение. А нельзя ли создать для всех цифровых кнопок (0,1,..9) калькулятора единый обработчик события? (см. лекцию 8. Листинг 8.1)

Практическое занятие № 8

1. **События мышки.** Разместите на единственной форме проекта 5-6 произвольных элементов управления (кнопки TButton, метки TLabel, панели TPanel). Напишите код позволяющий сделать следующее:
 - a. Перемещать пользователю элементы управления по форме мышкой во время работы проекта.
 - b. Сохранить новое положение элементов управления в внешнем файле в момент закрытия формы (событие формы OnClose).
 - c. Восстанавливать сохраненное в файле положение элементов управления в момент вывода формы на экран (событие формы OnShow).

Совет 1. Возможно при реализации пунктов **b** и **c** вам поможет класс TIniFile (не забудьте подключить `#include <System.IniFiles.hpp>`).

Совет 2. Вам понадобится помощь событий OnMouseDown(), OnMouseMove() и OnMouseUp().

2. **Клавиатурные события.** Доработайте приложение “Калькулятор” из лаб. работы 7 таким образом, чтобы пользователь получил возможность работать с программой с клавиатуры без помощи мышки.

Совет 3. Чтобы форма TForm смогла перехватывать нажатия клавиш не забудьте воспользоваться услугами свойства KeyPreview=true.

Практическое занятие № 9

1. Воспользовавшись услугами класса TStringList напишите приложение с визуальным интерфейсом, позволяющее:
 - a. Добавлять в список и удалять из списка текстовые строки.
 - b. Сортировать строки в списке.
 - c. Искать в списке заданную строку.
 - d. Сохранять в файл и выгружать из файла список строк.

Совет. Для отображения списка строк можно воспользоваться услугами компонента TListBox или TMemo, передавая список в компоненты с помощью метода Assign().

2. Изучите пример ex_p_09_04 TComponentList. И ответьте на следующие вопросы:
 - a. Какую роль в приложении выполняет экземпляр TComponentList.
 - b. Как организовано “падение” шариков по экрану?
 - c. Как ведется учет пойманных и потерянных шариков?

Практическое занятие № 10

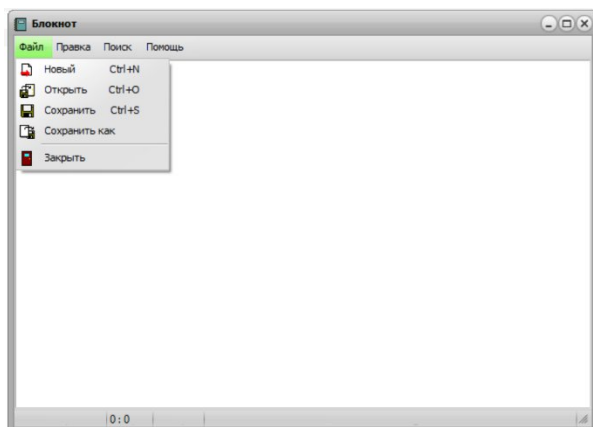
1. Разработайте приложение с интерфейсом MDI позволяющее пользователю открывать, просматривать в отдельных окнах файлы изображений в формате JPEG и закрывать окна.

Совет. Вам могут пригодиться следующие компоненты и модули:

- Главное меню TMainMenu.
 - Диалог выбора файла с изображением TOpenPictureDialog.
 - Компонент просмотра графических файлов TImage.
 - Для работы с графикой JPEG подключите модуль `#include <Jpeg.hpp>`.
2. Научите приложение упорядочивать дочерние окна каскадом, по горизонтали, по вертикали.
 3. Предусмотрите возможность того, чтобы приложение в отдельном текстовом файле вело протокол какие файлы и когда открывал пользователь.
 4. Предоставьте пользователю возможность выбирать предпочтительный стиль оформления приложения.

Практическое занятие № 11

Разработайте приложение “Блокнот” с интерфейсом SDI.

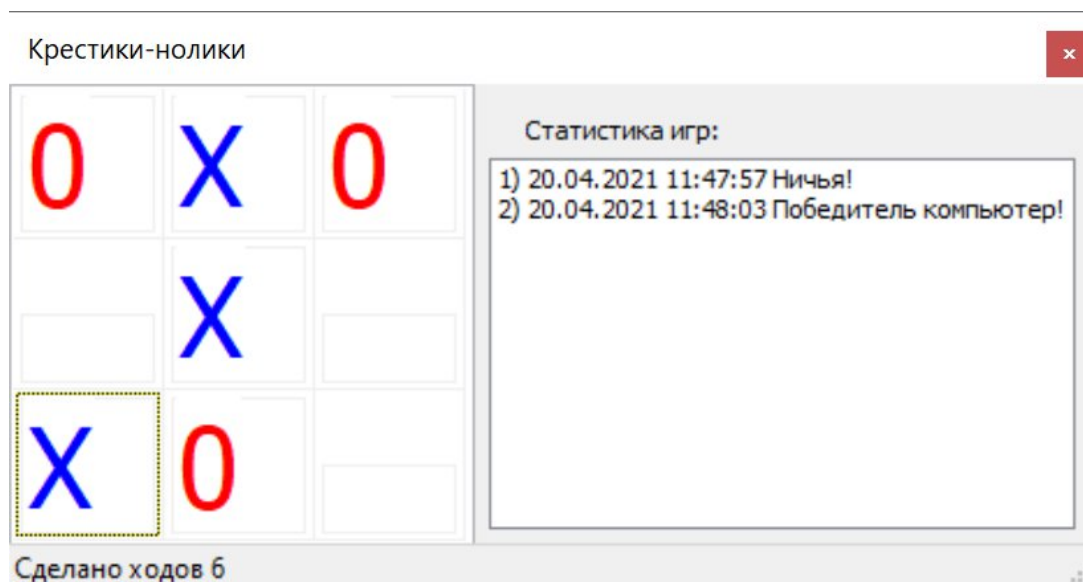


Вам понадобятся следующие компоненты:

1. Многострочный редактор TМемо.
2. Главное меню TMainMenu.
3. Контейнер изображений TImageList для подключения к пунктам меню пояснительных пиктограмм.
4. Диалог открытия текстового файла TOpenTextFileDialog.
5. Диалог сохранения текстового файла TSaveTextFileDialog.
6. И т.д.

Практическое занятие № 12

Разработайте приложение “Крестики-нолики” с интерфейсом SDI.



Игровое поле сделайте на основе компонента TDrawGrid или TStringGrid

Приложение должно позволять пользователю:

1. Выбирать символ (“X” или “O”) которым он будет играть.
2. Играть с компьютером в автоматическом режиме.
3. Вести статистику игр для текущей сессии.

Практическое занятие № 13

Дополните разработанное в 10 лабораторной работе MDI-приложение просмотра файлов изображений в формате JPEG:

- a. функцией сохранения файла, добавив пункт меню “Сохранить как”. Для определения имени сохраняемого файла воспользуйтесь компонентом-диалогом TSavePictureDialog.
- b. С помощью функции MessageDlg научите приложение запрашивать подтверждение у пользователя на завершение работы.
- c. С помощью функции InputBox (или InputQuery) научите приложение запрашивать новое имя для открытого файла (ограничение - переименовывать файл не обязательно).

Практическое занятие № 14

Воспользовавшись компонентом TTimer разработайте программу “Секундомер”.

Программа должна позволять:

1. Измерять интервал времени с точностью до 100 миллисекунд.
2. Запускать секундомер.
3. Приостанавливать секундомер.
4. Возобновлять работу приостановленного секундомера.
5. Остановка секундомера.
6. Сброс секундомера.
7. Вести протокол измерений, каждая запись протокола должна быть представлена в формате:
 - a. дата/время старта секундомера
 - b. дата/время остановки (или приостановки) секундомера
 - c. продолжительность прошедшего интервала времени с точностью до 100 миллисекунд.
8. Протокол измерений представляется в текстовом формате в компоненте TМето.

Практическое занятие № 15

Воспользуйтесь классом `TTreeView` и разработайте программу позволяющую представить в древовидном виде иерархическую структуру вашего института:

Институт

Дирекция

Директор

Заместитель

Кафедры

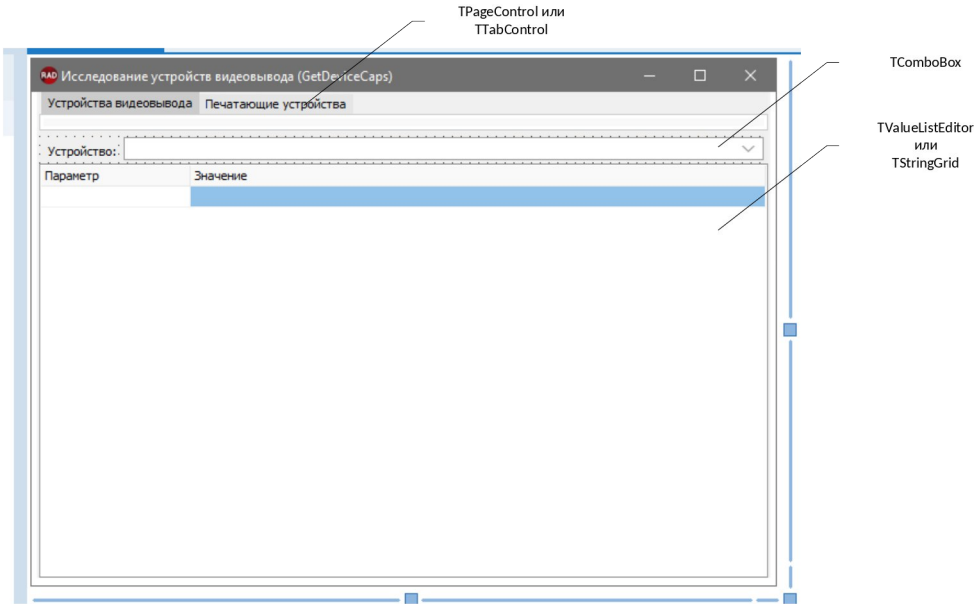
Кафедра 1

Кафедра 2

Программа должна позволять добавлять, редактировать, удалять узлы.
Сохранять в файл и загружать данные из файла.

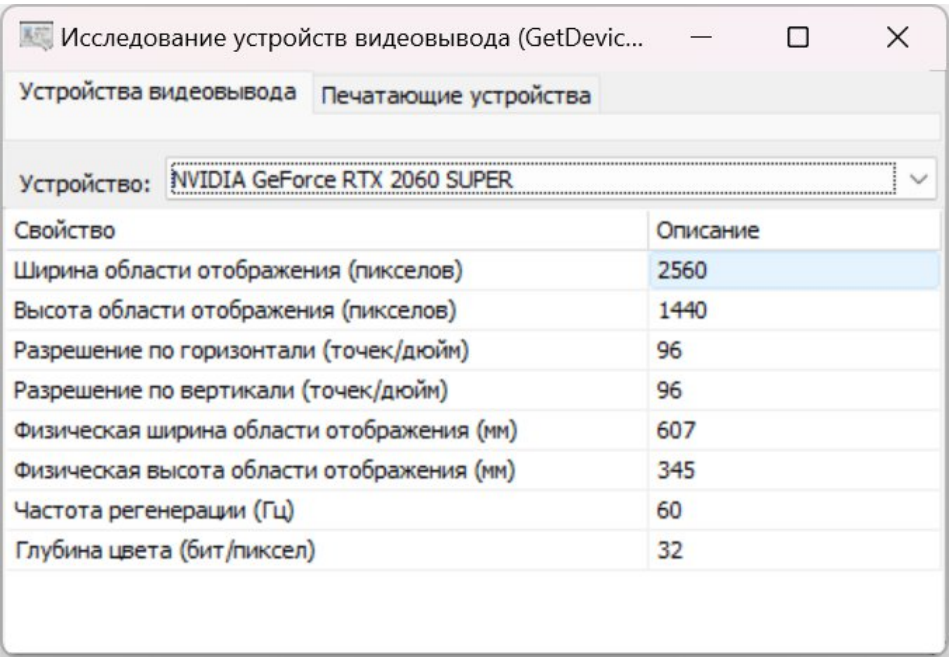
Практическое занятие № 16

Разработайте приложение, позволяющее пользователю получить сведения о подключенных к компьютеру графических устройствах.

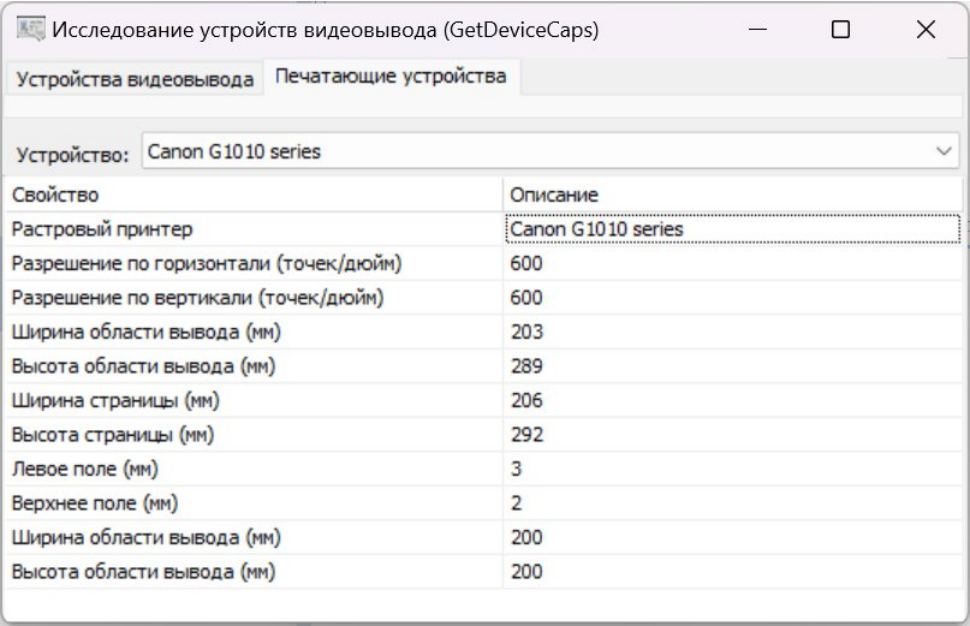


Для сбора сведений могут пригодиться функции Win API EnumDisplayDevices() и GetDeviceCaps().

Собираемая информация для видеокарт:



Собираемая информация для принтеров:



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ

Методические рекомендации по выполнению курсовых работ
Специальность 10.05.01 «Компьютерная безопасность»
Специализация «Информационно-аналитическая и техническая экспертиза
компьютерных систем»
Квалификация выпускника – специалист
Год начала обучения - 2026 г.
Форма обучения - очная

Ставрополь
2026

1 Общие положения

Курсовая работа по дисциплине «Технологии и методы программирования» представляет собой теоретическое и исследование и практическую (программную) реализацию базы данных по определённой для студента тематике. Курсовая работа оформляется в соответствии с общими требованиями и правилами, установленными ГОСТ 7.32-2001 Система стандартов по информатизации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления.

Курсовая работа выполняется студентом самостоятельно и является основанием для допуска к экзамену по дисциплине.

В соответствии с Государственными образовательными стандартами высшего профессионального образования по специальности 080801 «Прикладная информатика в экономике» курсовая работа имеет целью:

- систематизацию и углубление знаний студентов по дисциплине Базы данных (БД);
- совершенствование навыков и умений выпускника в решении сложных задач проектирования программного обеспечения (ПО).

Результаты работы оформляются в виде текстуальной части с приложением графиков, таблиц, чертежей, схем. Текстовые документы работы должны быть составлены и сброшюрованы в папки формата А4 по ГОСТ 2.105-95.

В процессе подготовки работы разрабатываются и последовательно реализуются следующие этапы:

- изучение литературных источников;
- планирование и выполнение исследований;
- работа над текстовой частью, чертежами, схемами и т.п.;
- оформление курсовой работы;
- защита курсовой работы .

За принятые решения и правильность всех выводов и фактических данных отвечает студент - автор работы.

2 Структура и содержание курсовой работы

Текстовый материал оформляется в следующей последовательности:

- титульный лист;
- задание;
- содержание;
- перечень условных обозначений, символов, единиц и терминов;
- введение;
- основная часть работы;
- заключение;
- список использованных источников;
- приложения.

3.1 Бланк титульного листа работы заполняется студентом самостоятельно.

3.2 Содержание включает наименование всех структурных элементов, разделов, подразделов, пунктов и подпунктов с указанием номера страницы, на которых размещается начало материала работы. В содержании указываются приложения с обозначением и наименованием. Слово **«СОДЕРЖАНИЕ»** записывают в виде заголовка (симметрично тексту) прописными буквами.

3.3 Перечень условных обозначений, символов, единиц и терминов далеко не всегда может присутствовать в работе. Как правило, он приводится в том случае, если в работе использованы малоизвестные сокращения и аббревиатуры. Причем если они встречаются в тексте довольно редко, перечень не составляется, а расшифровка приводится прямо в тексте, непосредственно после первого упоминания. Перечень печатается после

содержания, перед введением двумя колонками: в левой (узкой) перечислены сокращения, справа (в более широкой) - их расшифровка.

3.4 Введение содержит в сжатой форме все фундаментальные положения, обоснованию которых посвящена курсовая работа. Писать введение методологически лучше не в начале, а по окончании работы над курсовой работой, когда основная часть работы уже полностью готова. Введение начинается с небольшой преамбулы, где упоминается сущность и состояние исследуемой проблемы, ее значимость в современных условиях, указывается на необходимость научного исследования. Далее введение составляется в следующем порядке:

- актуальность темы;
- цель и задачи работы;
- объект и предмет исследования;
- методы исследования;
- новизна полученных результатов;

3.5 Основная часть работы должна содержать данные, отражающие существо, методику и основные результаты. Она включает:

– выбор направления работы, включающий обоснование выбора, методы решения задач, их сравнительную оценку. Это теоретический раздел работы дает теоретическое обоснование выбранной тематики и выполняется на основании литературных источников. Обучаемый должен представить анализ использованной литературы и дать свою точку зрения по проблеме исследования;

– второй раздел является аналитическим и охватывает основные технико-экономические показатели деятельности объекта исследования. Результаты анализа представляются в виде таблиц, диаграмм, графиков. Программное обеспечение может быть представлено в приложении. По аналитическому разделу работы представляются выводы в конце раздела;

– третий раздел рассматривает вопросы, связанные с темой работы. Обучаемый анализирует объект исследования и предлагает свои решения, исходя из результатов первых двух разделов.

3.6 В заключении к работе приводятся выводы, сделанные по результатам всей работы. В заключении целесообразно соотнести полученные выводы с целями и задачами, поставленными во введении. Здесь должны быть суммированы основные результаты исследования, сформулирована сущность решенной проблемы, даны грамотные и четкие предложения, касающиеся научного и практического использования полученных результатов.

3.7 В приложения рекомендуется включать материалы, связанные с выполненной работой, которые по каким-либо причинам не могут быть включены в основную часть. В приложения могут быть включены:

- материалы, дополняющие работу;
- промежуточные математические доказательства, формулы и расчеты;
- таблицы вспомогательных цифровых данных;
- методики, описания алгоритмов и программ задач, решаемых на ЭВМ, разработанных в процессе выполнения исследований;
- иллюстрации вспомогательного характера и др.

3.8 Сведения об источниках приводятся в соответствии с требованиями:

- ГОСТ 7.1-84 Система стандартов по информации, библиотечному и издательскому делу. Библиографическое описание документа. Общие требования и правила составления;

- ГОСТ 7.32-2001 Система стандартов по информатизации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления, п.6.1.

Порядок построения сведений об источниках – по мере упоминания в тексте. При ссылке в тексте номер источника из библиографического списка заключается в квадратные скобки, например, [1].

3.9 Приложения оформляют как продолжение курсовой работы на последующих листах. В тексте на все приложения должны быть даны ссылки. Приложения могут быть обязательными и информационными. Информационные приложения имеют рекомендуемый или справочный характер.

Каждое приложение следует начинать с новой страницы с указанием наверху посередине страницы слова «ПРИЛОЖЕНИЕ» и его обозначения, а под ним в скобках для обязательного приложения пишут слово «обязательное», а для информационного – «рекомендуемое» или «справочное».

Приложение должно иметь заголовок, который записывают симметрично относительно текста с прописной буквы отдельной строкой.

Приложения обозначают заглавными буквами русского алфавита начиная с «А», за исключением букв Е, З, И, О, Ч, Ъ, Ы, Ь. После слова «ПРИЛОЖЕНИЕ» следует буква, обозначающая его последовательность, например, «ПРИЛОЖЕНИЕ А».

Если в документе одно приложение, оно обозначается «ПРИЛОЖЕНИЕ А».

Приложения выполняют на листах формата А4, допускается оформление на листах формата А3 (297х420) мм по ГОСТ 2.301.

Приложения должны иметь общую с остальной частью работы сквозную нумерацию страниц.

3 Оформление курсовой работы

Курсовую работу оформляют с использованием компьютерной техники на одной стороне стандартного листа белой бумаги (А4, 210 х 297 мм).

Чертежи, большие таблицы и иллюстрации могут быть поданы в приложениях на формате А3.

Объем работы 20-30 листов, без учета приложений.

При наборе, печати работы делают следующие поля: сверху и снизу - 20 мм, справа - 10 мм, левое поле - 30 мм.

Предпочтительная гарнитура шрифта работы - Times New Roman, кегль - 14, цвет печати - черный, межстрочный интервал (интерлиньяж) - полуторный при печати на компьютере и двойной при печати на пишущей машинке. Плотность текста должна быть равномерной (без разрежений и уплотнений).

3.1 Изложение текста

3.1.1 Изложение текста работы лучше вести от третьего лица. Излагать мысли используя местоимение "я" не принято. В то же время, нельзя признать удачным излишний перегруз текста местоимением "мы", фразами "по нашему мнению", "считаем", "думаем" - это абсолютно нормально, но, в то же время, создает неосознанное впечатление, что работу писал коллектив авторов. Наиболее удачные в этом плане более неопределенные словесные обороты, например: "представляется, что...", "автор работы считает, что...", "разработанный подход позволяет...", "сначала производился опрос...", "данный вывод сделан на основании..." и т.д.

3.1.2 Текст работы должен быть кратким, четким и не допускать различных толкований. При изложении обязательных требований должны применяться слова «должен», «следует», «необходимо», «требуется», «не допускается», «запрещается», «не следует».

3.1.3 Изложение текста делается в безличной форме, например: «...значение коэффициента принято ...», или «принимается».

3.1.4 В тексте должны применяться научно-технические термины, обозначения и определения, установленные соответствующими стандартами, а при их отсутствии – общепринятые в научно-технической литературе.

3.1.5 В тексте не допускается:

- применять обороты разговорной речи;
- применять для одного и того же понятия различные научно-технические термины, близкие по смыслу (синонимы), а также иностранные слова при наличии равнозначных слов и терминов на русском языке;
- сокращать обозначения единиц физических величин, если они употребляются без цифр, за исключением единиц физических величин в головках и боковиках таблиц, в расшифровках буквенных обозначений, входящих в формулы и рисунки; применять математический знак минус (-) перед отрицательными значениями величин (следует писать слово «минус»);
- применять без числовых значений математические знаки, например, $>$ (больше), $<$ (меньше), $=$ (равно), $\neq$ (не равно), а также знака № (номер), % (процент); применять индексы стандартов без регистрирующего номера; перечень допускаемых сокращений слов установлен в ГОСТ 2.316, ГОСТ 7.12; в тексте следует применять стандартизованные единицы физических величин, их наименование и обозначение, установленные в ГОСТ 8.417;
- если в тексте документа приводится диапазон числовых значений физической величины, то обозначение единицы физической величины указывается, после последнего числового значения диапазона.

Примеры: 1. От 1 до 5 мм.

2. От плюс 10 до минус 40° С.

Недопустимо отделять единицу физической величины от числового значения (переносить их на разные строки или страницы), кроме единиц физических величин, помещенных в таблицах, выполненных машинописным способом.

3.2 Оформление графической части

3.2.1 Материал, выносимый на листы большого формата, должен быть обязательно представлен в пояснительной записке. Размеры листов чертежей и плакатов должны быть стандартные. Надписи, технические требования и

таблица на чертежах должны быть выполнены чертежным шрифтом. Блок – схемы алгоритмов и программ должны соответствовать требованиям ЕСКД.

3.2.2 Допускается представление графической части в виде презентации состоящей из 5-6 слайдов в Microsoft PowerPoint.

3.3 Вопросы нумерации

3.3.1 Страницы работы следует нумеровать арабскими цифрами, соблюдая сквозную нумерацию по всему тексту. Номер страницы проставляют внизу по центру страницы.

3.3.2 Титульный лист, задание включают в общую нумерацию страниц курсовой работы, но номера страниц на них не проставляются.

3.3.3 Структурные элементы работы не номеруются, записывают в виде заголовка (симметрично тексту) прописными буквами жирным шрифтом. Расстояние между заголовками структурных элементов и текстом должно быть 3 интервала.

3.3.4 Иллюстрации и таблицы, расположенные на отдельных листах, и распечатки с ЭВМ включают в общую нумерацию страниц.

3.3.5 Разделы, подразделы, пункты, подпункты нумеруют арабскими цифрами. Расстояние между заголовком раздела и подраздела – 3 интервала, а между подразделом и текстом – 1,5 интервала.

3.3.6 Разделы работы должны иметь порядковую нумерацию в пределах основной части, например, 1, 2, 3 и т.д.

3.3.7 Пункты должны иметь порядковую нумерацию в пределах каждого раздела или подраздела. Номер пункта включает номер раздела и порядковый номер подраздела или пункта, разделенные точкой, например, 1.1, 1.2 или 1.1.1, 1.1.2 и т.д.

3.3.8 Номер подпункта включает номер раздела, подраздела, пункта и порядковый номер подпункта, разделенные точкой, например, 1.1.1.1, 1.1.1.2 и т.д. Если раздел или подраздел имеет только один пункт или пункт имеет один подпункт, то нумеровать пункт (подпункт) не следует. Заголовки

подразделов следует начинать с абзацного отступа и печатать с прописной буквы в разрядку, не подчеркивая, без точки в конце.

3.3.9 Заголовки разделов следует располагать с абзацного отступа без точки в конце и писать прописными буквами, не подчеркивая. Если заголовок включает несколько предложений, их разделяют точками. Переносы слов в заголовках не допускаются.

3.3.10 Пункты и подпункты основной части следует начинать печатать с абзацного отступа.

3.3.11 Внутри пунктов или подпунктов могут быть приведены перечисления. Перед каждой позицией перечисления следует ставить дефис или при необходимости ссылки в тексте работы на одно из перечислений, арабскую цифру, после которой ставится скобка. Для дальнейшей детализации перечислений необходимо использовать строчные буквы, после которых ставится скобка, а запись производится с абзацного отступа, как показано в примере.

Пример:

- _____
- _____
 - 1) _____
 - 2) _____
 - а) _____
 - б) _____
 - в) _____

3.3.12 Примечания приводят в том случае, если необходимы пояснения или справочные данные к содержанию текста, таблиц или графического материала. Примечания не должны содержать требований. Примечания следует помещать непосредственно после текстового, графического материала или в таблице, к которым относятся эти примечания, и печатать с прописной буквы в разрядку, с абзаца. Если примечание одно, то

после слова «Примечание» ставится тире и текст примечания печатается тоже с прописной буквы. Одно примечание не нумеруют. Несколько примечаний нумеруют по порядку арабскими цифрами. Примечание к таблице помещают в конце таблицы под линией, обозначающей окончание таблицы.

Примеры:

П р и м е ч а н и е —

П р и м е ч а н и я

1 _____

2 _____

3.4 Формулы

3.4.1 Формулы пишутся отдельной строкой. Выше и ниже каждой формулы должно быть оставлено не менее одной свободной строки.

Знаки препинания ставят на продолжении основной строки формулы, непосредственно за ней, например:

Математическая модель данной экспертной системы выглядит следующим образом. Продукционная система

$$PS = \langle F, Z, I \rangle, \quad (4.1)$$

где F — рабочая память системы (называемая также базой данных), содержащая текущие данные (элементы рабочей памяти);

Z — база знаний, содержащая множество продукций (правил вида: "условие $\rightarrow$ действие");

I — интерпретатор (решатель), реализующий процедуры вывода.

3.4.2 После формулы помещают перечень всех принятых в формуле символов с расшифровкой их значений и указанием размерности (если в этом

есть необходимость). Буквенные обозначения дают в той же последовательности, в которой они приведены в формуле. Первая строка пояснения должна начинаться со слова «где» без двоеточия после него.

3.4.3 Переносы формул на другую строку допускаются на знаках неравенства, умножения, сложения, вычитания, деления, соотношений больше - меньше и других математических знаков.

3.4.4 При ссылке на формулу указывается ее полный номер в круглых скобках, например, ... в формуле (1.1).

3.4.5 Формулы нумеруются порядковой нумерацией в пределах всей работы арабскими цифрами в круглых скобках в крайнем правом положении на строке. Одну формулу обозначают – (1).

3.4.6 Формулы, помещенные в приложениях, должны нумероваться отдельной нумерацией арабскими цифрами в пределах каждого приложения с добавлением перед каждой цифрой обозначения приложения, например, формула (В. 1).

3.5 Иллюстрации

3.5.1 Все иллюстрации (графики, чертежи, схемы, фотографии) размещают сразу после первой ссылки на них. Например:

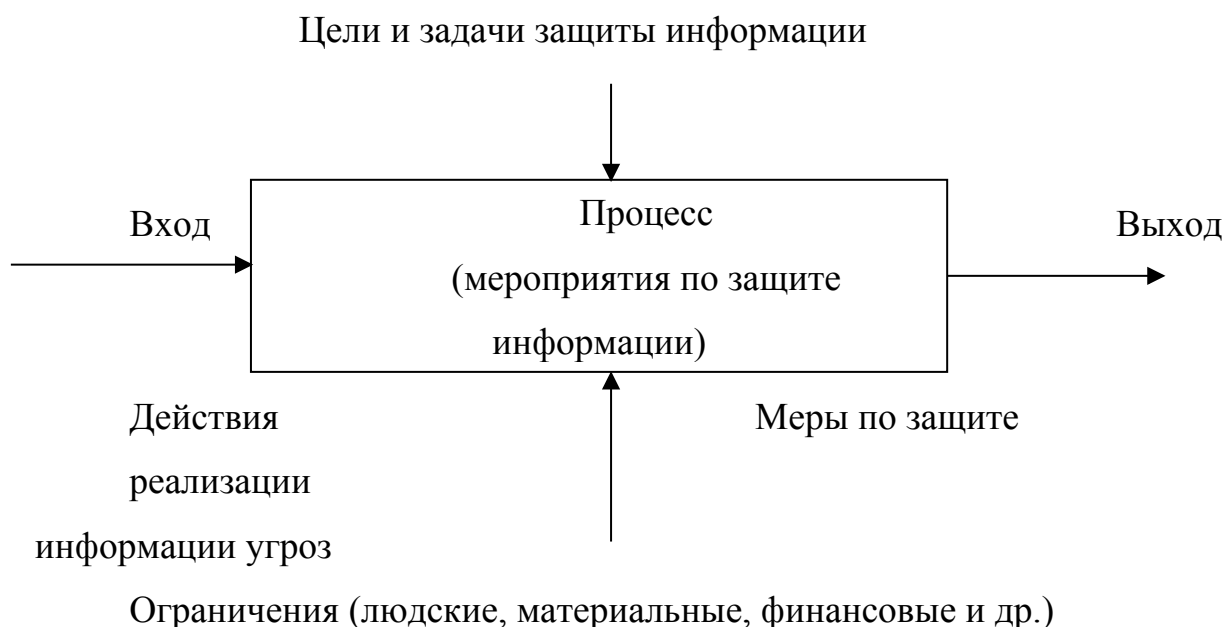


Рисунок 1 – Основные характеристики системы защиты

3.5.2 Иллюстрации сопровождаются:

- номером рисунка и наименованием (под рисунком посередине строки), например: Рисунок 1 – Основные характеристики системы защиты;
- поясняющими надписями, раскрывающими суть иллюстрации и детали содержания (они размещаются над словом рисунок и наименованием иллюстрации).

3.5.3 Иллюстрации следует нумеровать арабскими цифрами порядковой нумерацией в пределах работы, если иллюстрация одна, ее не нумеруют. При ссылках на иллюстрации следует писать «... в соответствии с рисунком 1». Иллюстрации должны выполняться на белой непрозрачной бумаге, не допускается калька и миллиметровая бумага.

3.6 Таблицы

3.6.1 Каждая таблица должна иметь порядковый номер в пределах всей работы. Название таблицы следует помещать над таблицей слева, без абзацного отступа в одну строку с ее номером через тире, например: «Таблица 1 – Наименование». При переносе части таблицы на другую страницу пишут «Продолжение», указывают номер таблицы, например: «Продолжение таблицы 1». В данном случае, нижняя черта проставляется на последней части таблицы. На все таблицы должна быть ссылка, при ссылке следует писать слово «таблица» с указанием ее номера. Таблица помещается после первого упоминания о ней в тексте так, чтобы ее можно было читать без поворота листа или с поворотом по часовой стрелке. Графа № п/п в таблицы не включается. Для облегчения ссылок по тексту в таблицах допускается нумерация граф.

3.6.2 В заголовках таблиц недопустимы нестандартные сокращения. В названиях граф надписи пишут в именительном падеже, единственном числе.

Таблица 1 –.....

Таблица 2 – Результаты структурирования информации

№ элемента информации	Наименование элемента информации	Гриф конфиденциальности информации	Цена информации	Наименование источника информации	Местонахождение источника информации
1	2	3	4	5	6

Продолжение таблицы 2

1	2	3	4	5	6

4 Примерный перечень тем курсовых работ

	Наименование темы КР
1.	Оценка способов отладки программного обеспечения в системе моделирования AnyLogic
2.	Анализ графических примитивов программными методами
3.	Сравнительный анализ алгоритмов построения минимального островного дерева
4.	Исследование алгоритмов поиска словесной информации
5.	Анализ способов представления графов в памяти ЭВМ
6.	Анализ метода турнирной сортировки при упорядочивании больших последовательностей
7.	Анализ быстрой сортировки Хоара
8.	Исследование алгоритмов поиска методом хеширования
9.	Сравнительный анализ методов сортировки больших массивов данных
10.	Сравнительный анализ методов сортировки файловых последовательностей
11.	Анализ методов нисходящего и восходящего программирования
12.	Анализ метода структурного программирования
13.	Сравнительный анализ алгоритмов поиска кратчайшего пути на графах
14.	Анализ программного обеспечения технологии аутентификации и авторизации пользователя в компьютерной системе
15.	Анализ применения технологии OLE Automation для программного ввода/вывода в приложения Excel
16.	Анализ алгоритмов генерации простых чисел
17.	Исследование возможностей графических алгоритмов библиотеки OpenGL при проектировании пользовательского интерфейса

	приложений
18.	Исследование графических алгоритмов, реализованных в стандартных библиотечных модулях языка C++ при проектировании пользовательского интерфейса приложений
19.	Анализ возможностей технологии модульного программирования при разработке защищённого программного обеспечения
20.	Анализ возможностей технологии событийно-ориентированного программирования при разработке защищённого программного обеспечения
21.	Анализ возможностей технологии параллельного программирования OpenMP
22.	Сравнительный анализ алгоритмов сжатия изображений
23.	Исследование достоинств и недостатков рекурсивных алгоритмов
24.	Исследование словарных методов сжатия
25.	Исследование алгоритмов черчения кривых линий
26.	Анализ способа сжатия данных на основе алгоритма контекстного моделирования (PPM)

ПРИЛОЖЕНИЕ А Образцы библиографической записи.

Книги.

Домашев А.В. Программирование алгоритмов защиты информации: учебное пособие. / Домашев А.В., Грунтович М.М. – М.: Нолидж, 2005.-416с.

Перроун П. Д. Создание корпоративных систем на базе Java2 Enterprise Edition: рук. разработчика : [пер. с англ.] / Поль Дж. Перроун, Венката С. Р., Р. Чаганти. – М. : Вильямс, 2005. – 1179 с.

Управление персоналом: учебное пособие / В.А.Якушев [и др.]; под ред. В.А.Якушева. – М.: Высшая школа, 2006. – 511с.

Стандарты и кодексы

ГОСТ Р 517721–2001. Аппаратура радиоэлектронная бытовая. Входные и выходные параметры и типы соединений. Технические требования [Текст]. – Введ. 2002–01–01. – М. : Изд-во стандартов, 2001. – IV, 27 с. : ил. ; 29 см.

ГОСТ 7. 53–2001. Издания. Международная стандартная нумерация книг [Текст]. – Взамен ГОСТ 7.53–86 ; введ. 2002–07–01. – Минск : Межгос. совет по стандартизации, метрологии и сертификации ; М. : Изд-во стандартов, сор. 2002. – 3 с. – (Система стандартов по информации, библиотечному и издательскому делу).

Гражданский кодекс РФ. Часть четвертая от 18 декабря 2006 г. № 230-ФЗ: принят Гос. Думой Федер. Собр. Рос. Федерации 24.11.2006 г. // Российская газета – 2006. – 22 дек., ч.1, ст 5444.

Статья из газет и журналов

Боголюбов А. Н. О вещественных резонансах в волноводе с неоднородным заполнением [Текст] / А. Н. Боголюбов, А. Л. Делицын, М. Д. Малых // Вестн. Моск. ун-та. Сер. 3, Физика. Астрономия. – 2001. – № 5. – С. 23–25.

Громов В.А. Информационная безопасность России / В.А.Громов // Известия. – 2008. – 2 марта. – С.2.

Словари и энциклопедии

Социальная философия: словарь / под общ.ред. В.А.Иванова. – М.: Академия, 2005.-353 с.

Экономическая энциклопедия / В.И.Иванов [и др.]. – М.: Экономика, 2007. – 233 с.

Статьи из сборников

Иванов В.Р. Конфликты в системе управления персоналом / В.Р.Иванов // Материалы конференции. – М.: Юрист, 2007. – С. 344-355.

Электронные ресурсы

Художественная энциклопедия зарубежного классического искусства

[Электронный ресурс]. – Электрон. текстовые, граф., зв. дан. и прикладная прогр. (546 Мб). – М. : Большая Рос. энцикл. [и др.], 2005. – 1 электрон. опт. диск (CD-ROM).

Логинов Л.Г. Информационная безопасность объектов / Безопасность технологий: Интернет-журнал. URL: <http://www.secur.ru> (дата обращения: 17.05.2005).

Иванов В.Е. Справочник по полупроводниковым приборам // Сервер радиолюбителей России. URL: <http://www.radio.ru> (дата обращения: 17.05.2005).

Информационная безопасность России: [сайт].[2004]. URL: <http://www.info.ru> (дата обращения: 17.05.2005).

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ТЕХНОЛОГИИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ

**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К САМОСТОЯТЕЛЬНОЙ
РАБОТЕ СТУДЕНТОВ**

Специальность 10.05.01 «Компьютерная безопасность»

Специализация «Информационно-аналитическая и техническая экспертиза
компьютерных систем»

Квалификация выпускника – специалист

Год начала обучения – 2026 г.

Форма обучения – очная

Ставрополь
2026

Оглавление

Пояснительная записка	248
План-график выполнения СРС по дисциплине.....	248
Методические рекомендации к СРС	248
Форма контроля СРС	251
Задания для СРС	251
Примерные темы рефератов.....	251
Примерные темы докладов.....	252
Примерные темы индивидуальных творческих заданий.....	252
Рекомендуемая литература и интернет-ресурсы.....	253

Пояснительная записка

В ходе изучения дисциплины студенты должны выполнить домашнее задание, способствующее приобретению навыков разработки и анализа алгоритмов. Студенту предлагается смоделировать предметную область и спроектировать приложение. По итогам выполнения работы преподавателю предоставляется набор листингов, разработанная студентом программа и оформленный отчёт с полученными результатами.

Текущий контроль усвоения знаний осуществляется путем прохождения 6 контрольных точек, подготовки и сдачи отчетов по итогам выполнения лабораторных работ, проверки выполнения домашнего задания, опросов на практических занятиях.

План-график выполнения СРС по дисциплине

№ контрольной точки	Вид самостоятельной работы	Форма контроля	Зачетные единицы (часы)
	5 семестр		
	6 семестр		
3,4	Подготовка к лабораторным занятиям	Защита лаб.работ	40
3	Самостоятельное изучение темы	Тестирование	20
4	Написание доклада	Проверка	24
	Итого за 6 семестр		84
	7 семестр		
5,6	Подготовка к лабораторным занятиям	Защита лаб.работ	6
5	Самостоятельное изучение темы	Тестирование	6
6	Индивидуальное творческое задание	Проверка	6
	Итого за 7 семестр		18
	Итого		102

Методические рекомендации к СРС

Тема 1. Фундаментальные структуры данных

Цели и задачи курса. Место дисциплины в учебном процессе. Методические рекомендации по изучению курса. Обзор литературы.

Концепция типа данных. Представление структур в памяти. Элементарные и простые структуры данных. Сложные структуры данных. Статические, полустатические и динамические структуры. Элементарные структуры данных (Строительные блоки, массивы, связные списки, обработка простых списков, распределение памяти под списки, строки, составные структуры данных).

Абстрактные типы данных (Абстрактные объекты и коллекции объектов, АД для стека магазинного типа, примеры программ-клиентов, использующих АД стека, реализации АД стека, создание нового АД,

очереди FIFO и обобщенные очереди, повторяющиеся и индексные элементы, АТД первого класса).

Понятие последовательности, последовательное представление данных. Операции над файлами. Базовые алгоритмы работы с файлами

Тема 2. Оценка качества и сложности алгоритмов, отладка и тестирование программного обеспечения

Понятие алгоритма с точки зрения методов программирования. Требования к алгоритму. Понятия модели вычислений и оценки сложности алгоритмов. Критерии оценки сложности алгоритмов. Методы оценки алгоритмов. Рост функций. О-нотация.

Отладка и тестирование программного обеспечения. Модульное тестирование, тестирование чёрного ящика. Создание проектной документации. Особенности сопровождения разработанного программного обеспечения. Принципы тестирования. Тестирование спецификаций. Тестирование на основании анализа текста программы. Тестовые данные. Этапы тестирования. Примеры тестирования циклов. Тестирование программ численных расчетов. Тестирование модулей. Средства тестирования Отладка.

Тема 3. Методы сортировки данных, алгоритмы поиска

Постановка задачи сортировки. Основные понятия и стратегии сортировки. Элементарные методы сортировки. Сортировка выбором. Сортировка вставками. Пузырьковая сортировка. Характеристики производительности элементарных методов сортировки. Сортировка методом Шелла. Сортировка других типов данных. Сортировка по индексам и указателям. Сортировка связанных списков. Метод распределяющего подсчета.

Быстрая сортировка. Базовый алгоритм. Характеристики производительности быстрой сортировки. Размер стека. Подфайлы небольших размеров. Метод деления с вычислением медианы из трех элементов. Дублированные ключи. Строки и векторы. Выборка.

Очереди по приоритетам и пирамидальная сортировка. Элементарные реализации. Пирамидальная структура данных. Алгоритмы для сортирующих деревьев. Пирамидальная сортировка. Абстрактный тип данных очереди по приоритетам. Очередь по приоритетам для индексных элементов. Биномиальные очереди

Сортировка последовательных файлов. Основные понятия внешней сортировки. Двухпутевое слияние. Абстрактное обменное слияние. Нисходящая сортировка слиянием. Усовершенствования базового алгоритма. Восходящая сортировка слиянием. Производительность сортировки слиянием. Реализация сортировки слиянием, ориентированной на связанные списки.

Основные понятия поиска. Алгоритмы исчерпывающего поиска. Поиск в последовательно организованном файле. Поиск в деревьях. Оптимальные деревья двоичного поиска. Сбалансированные деревья.

Тема 4. Алгоритмы на графах

Нелинейные структуры данных, графы и деревья. Представление графа в виде матрицы смежности и виде списка смежных вершин.

Поиск на графе. Поиск в глубину. Поиск в ширину. Отделимость и бисвязность. Алгоритмы поиска связных и двусвязных компонент в неориентированных графах. Алгоритм поиска сильносвязных компонент в ориентированных графах. Обобщённый поиск в графах.

Алгоритмы нахождения остовного дерева минимального веса. Алгоритмы Прима и поиск по приоритету. Алгоритмы Крускала. Алгоритм Борувки. Алгоритмы построения дерева MST.

Алгоритмы определения кратчайших расстояний между вершинами графа, нахождения транзитивного замыкания. Алгоритм Дейкстры. Кратчайшие пути между всеми парами. Кратчайшие пути в ациклических сетях. Эвклидовы сети. Сведение.

Тема 5. Генерация псевдослучайных последовательностей и алгоритмы порождения перестановок

Моделирование равномерно распределённых случайных величин. Методы моделирования дискретных и непрерывных случайных величин.

Детерминированные ГПСЧ. ГПСЧ с источником энтропии или ГСЧ. ГПСЧ в криптографии. Аппаратные ГПСЧ.

Тесты DIEHARD. Метод Монте-Карло. Случайное простое число.

Алгоритмы порождения перестановок в лексикографическом порядке, циклическим сдвигом и в порядке минимального изменения. Коды Грея.

Тема 6. Алгоритмы сжатия данных. Хеширование. Графические алгоритмы

Избыточность данных. Метод Running. Метод Huffman. Метод LZW.

Вопросы построения хеш-функций и хеш-таблиц. Недостатки таблиц с прямой адресацией. Раздельное связывание. Линейное зондирование. Двойное хеширование. Динамические хеш-таблицы.

Способы представления данных о цвете и цветовые модели. Алгоритмы вывода простейших графических примитивов. Пошаговая выборка. Способы пространственного разделения. Алгоритмы сканирования на плоскости. Удаление невидимых поверхностей. Двумерные аффинные преобразования.

Тема 7. Основные понятия и подходы разработки современного программного обеспечения

Этапы развития технологии программирования. Жизненный цикл и этапы разработки программного обеспечения (ПО). Оценка качества процессов создания ПО. Понятие защищённого ПО. Концепция безопасности SD3. Модули и их свойства.

Особенности создания программного продукта (принципы работы с требованиями к программному обеспечению, проблематика проектирования, оценка стоимости ошибок, управление требованиями, последовательность работы с требованиями, анализ проблемы, преграды на пути выявления требований)

Оценка качества процессов создания программного обеспечения (Серия стандартов ISO 9000, CMM, процесс сертификации программ на базе информации об их использовании)

Жизненный цикл программы (понятие технологии разработки программы, основа разработки программного обеспечения, модели жизненного цикла, Rational Objectory Process — модель жизненного цикла, жизненный цикл UML (Rational Objectory Process), специфицирование и планирование, процесс разработки, выпуск продукта и механизмы обратной связи)

Тема 8. Проектирование ПО при структурном подходе

Средства описания структурных алгоритмов. Классификация программных продуктов по функциональному признаку. Основные эксплуатационные требования к программным продуктам. Подготовка к проектированию ПО.

Проектирование программного обеспечения при структурном подходе (структурная схема разрабатываемого программного обеспечения, функциональная схема, метод пошаговой детализации при составлении алгоритмов, структурные карты Константайна, структурные карты Джексона, CASE-технологии, ускорение разработки программного обеспечения, методология RAD).

Тема 9. Проектирование ПО при объектном подходе

Концепция объектно-ориентированного программирования. Классы и объекты. Проектирование программного обеспечения при объектном подходе (разработка структуры программного обеспечения при объектном подходе, диаграммы кооперации)

Технология СОМ. Интерфейсы и СОМ-объекты. Особенности проектирования клиент-серверного и распределённого ПО. Параллельная обработка данных. Перспективные возможности современных языков программирования по созданию высокотехнологичного ПО. Совершенствование действующего ПО, рефакторинг кода. Управление командой разработчиков. Роль пользователя в жизненном цикле разработки ПО. Стоимость программного продукта.

Форма контроля СРС

Качество усвоения знаний студентами осуществляется в форме тестирования, индивидуального собеседования, проверки доклада, реферата и проверки индивидуального творческого задания.

Задания для СРС

Примерные темы рефератов

1. Понятие структуры данных. Стеки, деки, очереди, линейные списки и деревья.
2. Алгоритмы сортировки сравнениями. Вывод нижней оценки для трудоемкости работы алгоритмов данного типа.
3. Алгоритм сортировки вставками и вывод оценки его трудоемкости.

4. Обменная сортировка. Алгоритм быстрой сортировки.
5. Сортировка выбором. Турнирная и пирамидальная сортировки.
6. Сортировки слиянием и распределяющие сортировки.
7. Алгоритмы исчерпывающего поиска.
8. Алгоритмы поиска в последовательно организованных файлах.
9. Алгоритмы поиска в деревьях.
10. Хеширование и способы разрешения коллизий.
11. Способы задания графа. Остовное дерево. Алгоритм поиска в глубину.
12. Алгоритм поиска сильно связанных компонент.
13. Нахождение двусвязных компонент.
14. Алгоритм построения минимального остовного дерева.
15. Моделирование равномерно распределенных случайных величин.
16. Алгоритмы порождения перестановок в лексикографическом порядке и циклическим сдвигом.

Примерные темы докладов

1. Обменная сортировка со слиянием алгоритм Бэтчера.
2. Лексикографическая сортировка.
3. Внешняя сортировка.
4. Алгоритмы поиска с возвращениями.
5. Индексно-последовательный поиск.
6. Построение оптимальных бинарных деревьев поиска.
7. Алгоритм сортировки в дерево.
8. Балансировка деревьев по высоте.
9. Балансировка деревьев по весу.
10. Цифровой поиск.
11. Алгоритм поиска в ширину.
12. Алгоритм нахождения кратчайшего пути.
13. Алгоритм транзитивного замыкания.
14. Алгоритм нахождения кратчайших расстояний от источника до всех остальных вершин.
15. Моделирование равномерно распределенных дискретных случайных величин.
16. Моделирование равномерно распределенных дискретных случайных величин.
17. Алгоритмы порождения перестановок в порядке минимального изменения.
18. Графические алгоритмы заливки замкнутых областей.

Примерные темы индивидуальных творческих заданий

1. Напишите программу, реализующую один из алгоритмов сортировки.
2. Напишите программу, реализующую один из алгоритмов поиска.

3. Получить эмпирические оценки трудоемкости алгоритма сортировки или поиска.
4. Произвести сравнительный анализ трудоемкости работы нескольких алгоритмов.
5. Напишите программу, реализующую алгоритм поиска сильно связанных компонент в графе.
6. Напишите программу, реализующую алгоритм поиска двусвязных компонент в графе.
7. Напишите программу, реализующую алгоритм порождения перестановок в лексикографическом порядке.
8. Напишите программу, реализующую алгоритм порождения перестановок циклическим сдвигом.

Рекомендуемая литература и интернет-ресурсы

Основная литература:

1. Дональд Э. Кнут. Искусство программирования. Том 1. Основные алгоритмы 3-е издание - СПб.: Вильямс, 2011. - 720 с.
2. Дональд Э. Кнут. Искусство программирования, том 2. Получисленные алгоритмы 3-е издание. - СПб.: Вильямс, 2011. - 832 с.
3. Технология программирования: учебник/ Г.С. Иванова. - М.: КНОРУС, 2011. - 336 с.

Дополнительная литература:

1. Роберт Мартин. Чистый код: создание, анализ, рефакторинг. Библиотека программиста. - СПб.: Питер, 2010. – 464 с.
2. Кормен. Т., Левerson Ч., Ртвест Р. Алгоритмы. Построение и анализ. - М.; МЦНМО, 2006, 960 с.
3. Вирт Н. Алгоритмы и структуры данных: Пер. с англ. – 2-е изд., испр. – СПб.: Невский Диалект, 2007. – 352 с.: ил.
4. Глаголев В.А. Разработка технической документации: Руководство для технических писателей и локализаторов. - СПб.: Питер, 2008. - 192 с.

Интернет-ресурсы:

1. www.microsoft.com – сайт компании Microsoft
2. www.embarcadero.com – сайт компании Embarcadero
3. <http://algolist.manual.ru/> - сайт посвященный разработке алгоритмов