Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
“Севастопольский государственный университет”

Методические указания
к лабораторным работам по дисциплине
“Алгоритмизация и программирование”
для студентов дневной и заочной формы обучения направлений
09.03.02 – “Информационные системы и технологии”
и 09.03.03 – “Прикладная информатика”

2 часть

Севастополь
2019

УДК 004.42 (075.8)

Методические указания к лабораторным работам по дисциплине “Алгоритмизация и программирование” для студентов дневной и заочной формы обучения направлений 09.03.02 — “Информационные системы и технологии” и 09.03.03 — “Прикладная информатика”, часть 2 / Сост. В. Н. Бондарев, Т. И. Сметанина, А. К. Забаштанский. — Севастополь: Изд-во СевГУ, 2019. — 118 с.

Методические указания предназначены для проведения лабораторных работ и обеспечения самостоятельной подготовки студентов по дисциплине “Алгоритмизация и программирование”. Целью методических указаний является обучение студентов основным принципам структурного программирования, навыкам разработки программ на языках Паскаль и C/C++.

Методические указания составлены в соответствии с требованиями программы дисциплины “Алгоритмизация и программирование” для студентов направлений 09.03.02 — “Информационные системы и технологии” и 09.03.03 — “Прикладная информатика ” и утверждены на заседании кафедры «Информационные системы» протокол № ____ от

Допущено научно-методической комиссией института информационных технологий и систем управления в технических системах в качестве методических указаний.

Рецензент: Кожаев Е.А., к.т.н., доцент кафедры информационных технологий и компьютерных систем

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цели и задачи лабораторных работ

Основная цель выполнения настоящих лабораторных работ — получение практических навыков создания и отладки программ на языке C/C++, использующих сложные типы данных. В результате выполнения лабораторных работ студенты должны углубить знания основных теоретических положений дисциплины “Информатика и программирование”, решая практические задачи на ЭВМ.

Студенты должны получить практические навыки работы в интегрированной системе программирования Borland C/C++ (или Eclipse CDT), навыки разработки программ, использующих текстовые и типизированные файлы, структуры, динамические списковые и древовидные структуры данных, графическую библиотеку.

Выбор вариантов и график выполнения лабораторных работ

В лабораторных работах студент решает заданную индивидуальным вариантом задачу. Варианты заданий приведены к каждой лабораторной работе и уточняются преподавателем.

Лабораторная работа выполняется в два этапа. На первом этапе — этапе самостоятельной подготовки — студент должен выполнить следующее:

- изучить основные теоретические положения лабораторной работы и подготовить ответы на контрольные вопросы, используя конспект лекций и рекомендованную литературу;
- разработать алгоритм решения задачи и составить его структурную схему;
- описать схему алгоритма;
- написать программу на языке C/C++ и описать ее;
- оформить результаты первого этапа в виде заготовки отчета по лабораторной работе.

На втором этапе, выполняемом в лабораториях кафедры, студент должен:

- отладить программу;
- разработать и выполнить тестовый пример;
- подготовить и защитить отчёт по лабораторной работе.

Студенты дневного отделения должны выполнять и защищать работы строго по графику. График уточняется преподавателем, ведущим лабораторные занятия.

Требования к оформлению отчета

Отчёты по лабораторной работе оформляются каждым студентом индивидуально. Отчёт должен включать: название и номер лабораторной работы; цель работы; вариант задания и постановка задачи; разработку и описание алгоритма решения задачи; текст и описание программы; результаты выполнения программы; выводы по работе; приложения. Содержание отчета указано в методических указаниях к каждой лабораторной работе.

Постановка задачи представляет изложение содержания задачи, цели её решения. На этом этапе должны быть полностью определены исходные данные, перечислены задачи по преобразованию и обработке входных данных, дана характеристика результатов, описаны входные и выходные данные.

Разработка алгоритмов решения задачи предполагает математическую формулировку задачи и описание решения задачи на уровне схем алгоритмов. Схемы алгоритмов должны быть выполнены в соответствии с требованиями ГОСТ и сопровождаться описанием.

Описание программы включает описание вычислительного процесса на языке C/C++. Кроме текста программы, в отчёте приводится её пооператорное описание.

В приложении приводится текст программы, распечатки, полученные во всех отладочных прогонах программы с анализом ошибок, результаты решений тестового примера.

1 ЛАБОРАТОРНАЯ РАБОТА №1
“ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ И РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ”

1.1 Цель работы

Изучение структуры С-программы.

Формирование навыков программирования алгоритмов линейной и разветвляющейся структуры на языке С.

Исследование особенностей ввода-вывода значений стандартных типов в языках С/С++.

1.2 Краткие теоретические сведения

Для выполнения лабораторной работы необходимо внимательно изучить структуру С-программы, основные типы данных и операторы языка С, управляющие инструкции, позволяющие реализовывать программы линейной и разветвляющейся структуры, а также ознакомиться со средствами ввода-вывода языков С/С++ и математическими функциями, описанными в файле **<math.h>**.

1.2.1 Структура С-программы

Любая *С-программа* состоит из *функций* и *переменных*. Функции содержат *инструкции*, описывающие вычисления, которые необходимо выполнить, а переменные хранят значения, используемые в процессе этих вычислений. Любая программа начинает свои вычисления с первой инструкции функции `main`. Таким образом, каждая С-программа должна содержать функцию `main`. Эта функция может не иметь параметров, и тогда ее заголовок — `main()`. Если функция `main` имеет параметры, то эти параметры выбираются из *строки вызова* (*командной строки*).

Левая фигурная скобка (`{`) должна начинать *тело* каждой функции. Соответствующая *правая фигурная скобка* (`}`) должна заканчивать каждую функцию. Так что тело функции представляет собой *блок*, ограниченный фигурными скобками `{}`. Блок содержит *выражения*, заканчивающиеся *точкой с запятой* (`;`), которые в языке С называют *инструкциями*. Общая структура функции `main` такова:

```
main()
{   инструкция_1
    инструкция_2
    ............
    инструкция_n
}
```

Помимо функции `main` в тексте программы могут располагаться определения *вспомогательных функций*.

В системах программирования С перед началом этапа компиляции выполняется программа предварительной *(препроцессорной)* обработки. Эта программа подчиняется специальным командам *(директивам препроцессора)*, которые ука-

зывают, что в программе перед ее компиляцией нужно выполнить определенные преобразования. Обычно эти преобразования состоят во включении других текстовых файлов в файл, подлежащий компиляции, и выполнении различных текстовых замен. Так, например, появление директивы

```
#include <stdio.h>
```

приводит к тому, что *препроцессор* подставляет на место этой директивы текст файла `<stdio.h>`, т. е. происходит включение информации о *стандартной библиотеке ввода-вывода*.

1.2.2 Переменные и основные типы данных языка С

В С любая *переменная* должна быть описана раньше, чем она будет использована; обычно все переменные описываются в начале функции перед первой исполняемой инструкцией. В *декларации (описании)* объявляются свойства переменных. Декларация состоит из названия типа и списка переменных, например:

```
int width, height;
float distance;
int exam_score;
char c;
```

В первом описании имеется *список переменных*, содержащий два имени (`width` и `height`). Обе переменные описываются как целые (`int`). Целочисленная переменная `exam_score` описана отдельно, хотя ее можно добавить к первому списку целых переменных.

Переменные можно инициализировать в месте их описания, например:

```
float distance=15.9;
```

Здесь переменной `distance` присваивается начальное значение `15.9`.

Имя переменной — это любая последовательность символов, содержащая *буквы*, *цифры* и *символы подчеркивания* (_), которая не начинается с цифры. Язык С *чувствителен к регистру* — прописные и строчные буквы различаются.

В С существует всего несколько *базовых типов*:

`char` — единичный байт, который может содержать одну литеру;

`int` — целое;

`float` — число с плавающей точкой одинарной точности;

`double` — число с плавающей точкой повышенной точности.

Имеется также несколько *квалификаторов*, которые можно использовать вместе с указанными базовыми типами. Например, квалификаторы `short` (короткий) и `long` (длинный) применяются к целым:

```
short int counter; long int amount;
```

В таких описаниях слово `int` можно опускать, что обычно и делается.

Квалификаторы `signed` (со знаком) и `unsigned` (без знака) можно применять к типу `char` и любому целому типу. Тип `long double` предназначен для арифметики с плавающей точкой повышенной точности.

Если операнды оператора принадлежат разным типам, то они *приводятся* к общему типу. Автоматически производятся лишь те преобразования, которые без

какой-либо потери информации превращают операнды с меньшим диапазоном значений в операнды с большим диапазоном значений.

1.2.3 Основные операторы языка С

В таблице 1.1. показаны *приоритеты* и *порядок вычисления* всех операторов языка С. Операторы, перечисленные на одной строке, имеют одинаковый приоритет; строки упорядочены по убыванию приоритетов. *Унарные операторы* +, –, и * имеют более высокий приоритет, чем те же операторы в *бинарном* варианте.

Таблица 1.1 — Приоритеты и порядок вычисления операторов

Операторы	Порядок выполнения
`() [] -> .`	слева направо
`! ~ ++ -- + - * & (тип) sizeof`	справа налево
`*  /  %`	слева направо
`+  -`	слева направо
`<<  >>`	слева направо
`<  <=  >  >=`	слева направо
`==  !=`	слева направо
`&`	слева направо
`^`	слева направо
`|`	слева направо
`&&`	слева направо
`||`	слева направо
`?:`	справа налево
`= += -= *= /= %= &= ^= |= <<= >>=`	справа налево
`,`	слева направо

1.2.3.1 Арифметические операторы

К *арифметическим операторам* относятся: *сложение* (+), *вычитание* (–), *деление* (/), *умножение* (*) и *остаток* (%). Все операции (за исключением остатка) определены для переменных типа `int`, `char` и `float`. Остаток не определен для переменных типа `float`. Деление целых сопровождается отбрасыванием дробной части.

Все арифметические операции с плавающей точкой производятся над операндами *двойной точности* (`double`). После того, как получен результат с двойной точностью, он приводится к типу левой части выражения, если левая часть присутствует.

1.2.3.2 Операторы отношения

В языке С определены следующие *операторы отношения*: *проверка на равенство* (==), *проверка на неравенство* (!=), *меньше* (<), *меньше или равно* (<=), *больше* (>), *больше или равно* (>=).

Все перечисленные операторы вырабатывают результат целого типа (`int`). Если данное отношение между операндами ложно, то значение этого целого —

ноль. Значения, отличные от нуля, интерпретируются как истинные.

1.2.3.3 Логические операторы

К *логическим операторам* относятся:

`&&` — *логическое И* (and);

`||` — *логическое ИЛИ* (or);

`!` — *логическое НЕ* (not).

Операндами логических операторов могут быть любые числа. Результат логического выражения — единица, если истина, и ноль, если ложь. Вычисление выражений, содержащих логические операторы, производится слева направо и прекращается, как только удается определить результат.

1.2.3.4 Операторы присваивания

К *операторам присваивания* относятся =, +=, −=, *= и /=, а также *префиксные* и *постфиксные* операторы ++ и – –. Все операторы присваивания присваивают переменной результат вычисления выражения.

В одном выражении оператор присваивания может встречаться несколько раз. Вычисления производятся *справа налево*. Например: `a=(b=c)*d;` вначале переменной `b` присваивается значение переменной `c`, затем выполняется операция умножения на `d`, и результат присваивается переменной `a`.

Типичный пример использования *многократного присваивания*:

```
a=b=c=d=e=f=0;
```

Операторы +=, −=, *=, /= являются *укороченной* формой записи оператора присваивания. Например:

```
a+=b; // a=a+b;          a*=b; // a=a*b;
a−=b; // a=a−b;          a/=b; // a=a/b;
```

Многие компиляторы генерируют код, который выполняется быстрее при использовании таких операторов присваивания.

Префиксные и постфиксные операторы присваивания ++ и − − используют для увеличения *(инкремент)* и уменьшения *(декремент)* на единицу значения переменной. Эти операторы можно использовать как в префиксной форме (помещая перед переменной, например, `++n`), так и в постфиксной форме (помещая после переменной: `n++`).

1.2.4 Основные средства ввода-вывода языков C/C++

1.2.4.1 Основные средства ввода-вывода языка C: функции `printf` и `scanf`

В языке С ввод-вывод данных реализуется с помощью *внешних функций*. Одними из наиболее универсальных и полезных функций ввода и вывода являются соответственно функции `scanf` и `printf`, описанные в головном файле `<stdio.h>`.

Функцию `printf` можно использовать для вывода любой комбинации символов, целых и вещественных чисел, строк, беззнаковых целых, длинных целых и беззнаковых длинных целых.

Типичный пример использования функции `printf`:

```
#include <stdio.h>
main()
{int age=22; // возраст Эрика
 float income=534.72; // доход Эрика
 printf("\nВозраст Эрика: %d. Его доход: $%.2f.\n",
           age,income);
 return 0;
}
```

Функция `printf` выводит на экран значения своих аргументов `age` и `income` в формате, который определяется *управляющей строкой*, являющейся первым параметром этой функции. Все символы этой строки, кроме *спецификаций формата* (`%d` и `%.2f`) и *управляющей последовательности* (`\n`), выводятся на экран без изменений.

Таким образом, при выполнении функции `printf` произойдет следующее. Последовательность символов `\n` переведет курсор на новую строку. В результате последовательность символов «`Возраст Эрика: `» будет выведена с начала новой строки. Смволы `%d` — это спецификация формата для целой переменной. Вместо `%d` будет подставлено значение переменной `age`. Далее будет выведена литеральная строка «`. Его доход: $`». `%.2f` — это спецификация формата для вещественной переменной, а также указание формата для вывода только двух цифр после десятичной точки. Вместо `%.2f` будет выведено значение переменной `income` в указанном формате. В заключение управляющая последовательность `\n` вызовет переход на новую строку.

Как видно из рассмотренного примера, спецификации формата помещаются внутри печатаемой строки. Вслед за этой строкой должен стоять ноль или более переменных, разделенных запятыми. Каждой спецификации в управляющей строке функции `printf` должна соответствовать переменная адекватного типа. Если используется несколько спецификаций, то всем им должны соответствовать переменные того типа, который задается спецификацией.

Формально спецификацию формата можно определить следующим образом: `%[флаг][ширина][.точность][h|l|L]символ_формата`,
где `ширина` — минимальное количество позиций, отводимых под выводимое значение, `точность` — количество позиций, отводимых под дробную часть числа.

Модификатор `h` указывает, что соответствующий аргумент должен печататься как `short` или `unsigned short`; `l` сообщает, что аргумент имеет тип `long` или `unsigned long`; `L` информирует, что аргумент принадлежит типу `long double`.

Возможные значения полей `флаг` и `символ_формата` приведены в таблицах 1.2 и 1.3.

Функция `scanf` служит для ввода данных. Подобно функции `printf`, `scanf` использует спецификацию формата, сопровождаемую списком вводимых переменных. Каждой вводимой переменной в функции `scanf` должна соответствовать спецификация формата.

Таблица 1.2 — Описание значений поля флаг

Флаг	Назначение
-	выравнивание по левому краю поля
+	печать числа всегда со знаком
Пробел	если первая литера — не знак, то числу должен предшествовать пробел

Таблица 1.3 — Описание значений поля символ_формата

Символ формата	Тип выводимого объекта
C	char; единичная литера
S	char *; строка
d,i	int; знаковая десятичная запись
o	int; беззнаковая восьмеричная запись
u	int; беззнаковое десятичное целое
x,X	int; беззнаковая шестнадцатеричная запись
f	double; вещественное число с фиксированной точкой
e,E	double; вещественное число с плавающей точкой
g,G	double; вещественное число в виде %f или %e в зависимости от значения
p	void *; указатель
%	знак процента %

Перед именами переменных следует ставить символ &. Этот символ означает *«взять адрес переменной»*. Ниже приведен пример программы, использующей функцию scanf:

```
#include <stdio.h>
main()
{ int weight, // вес
      height; // рост
  printf("Введите Ваш вес:\n"); scanf("%d",&weight);
  printf("Введите Ваш рост:\n"); scanf("%d",&height);
  printf("\nВаш вес = %d;\nВаш рост = %d.\n",
              weight,height);
  return 0;
}
```

Здесь &weight и &height — это адреса переменных weight и height соответственно.

1.2.4.2 Средства ввода-вывода языка C++: объекты cin и cout

В C++ вывод на экран выполняется с помощью *объекта стандартного потока вывода* cout, а ввод с клавиатуры осуществляется с помощью *объекта*

стандартного потока ввода cin. Для использования этих объектов необходимо подключить головной файл <iostream.h>. Для обработки форматного ввода-вывода при помощи так называемых *манипуляторов потока* необходимо подключить головной файл <iomanip.h>.

Рассмотрим пример использования объекта cout:

```
#include <iomanip.h>
#include <iostream.h>
main()
{   int age=22; // возраст Эрика
    float income=534.72; // доход Эрика
    cout<<'\n'<<"Возраст Эрика: "<<age<<'.'
        <<" Его доход: $"<<setprecision(2)
         <<income<<'.'<<endl;
    return 0;
}
```

Вывод в поток выполняется с помощью *операции поместить в поток* <<. В рассматриваемом примере объекту cout с помощью операции << передаются значения, которые необходимо вывести на экран. Операция << является «более интеллектуальной» по сравнению с функцией printf, так как определяет тип выводимых данных. Для вывода нескольких объектов операция << используется в *сцепленной форме*.

Манипулятор потока endl вызывает переход на новую строку и очищает *буфер вывода*, т. е. заставляет буфер немедленно вывести данные, даже если он полностью не заполнен. *Очистка (сброс)* буфера вывода может быть также выполнена манипулятором flush.

При выводе значения переменной income используется манипулятор потока setprecision. Использование setprecision(2) — это указание формата для вывода только двух цифр после десятичной точки. Поскольку манипулятор setprecision принимает параметр, он называется *параметризированным манипулятором потока*.

Для установки ширины поля вывода может быть использован манипулятор потока setw, принимающий в качестве параметра число символьных позиций, в которые будет выведено значение.

Ввод потока осуществляется с помощью *операции взять из потока* >>. Эта операция применяется к объекту стандартного потока ввода cin. Рассмотрим пример использования объекта cin:

```
#include <iostream.h>
main()
{   int weight, // вес
        height; // рост
    cout<<"Введите Ваш вес:"<<endl;
    cin>>weight;
    cout<<"Введите Ваш рост:"<<endl;
```

```
    cin>>height;
    cout<<"\nВаш вес = "<<weight<<';'<<endl
        <<"Ваш рост = "<<height<<'.'<<endl;
    return 0;
}
```

Здесь считываемые значения размещаются в переменных `weight` и `height`. В процессе ввода последовательность символов, набранная на клавиатуре, преобразуется в необходимое *внутреннее представление* (в рассматриваемом примере целочисленное).

1.2.5 Управляющие инструкции языка C: `if-else`, условное выражение, `switch`

1.2.5.1 Инструкция `if-else`

Инструкция `if-else` используется для принятия решения. Ниже представлен ее синтаксис:

```
if (выражение)
    инструкция_1
else
    инструкция_2
```

причем `else`-часть может быть опущена. Сначала вычисляется `выражение`, и, если оно истинно, то выполняется `инструкция_1`. Если `выражение` ложно и существует `else`-часть, то выполняется `инструкция_2`. Необходимо отметить, что `else`-*часть всегда относится к ближайшей* `if`-*части*.

1.2.5.2 Условное выражение

Условное выражение, написанное с помощью *тернарного оператора* «`?:`», представляет собой другой способ записи инструкции `if-else`. В выражении

```
выражение1 ? выражение2 : выражение3
```

первым вычисляется `выражение1`. Если его значение отлично от нуля, то вычисляется `выражение2` и значение этого выражения становится значением всего условного выражения. В противном случае вычисляется `выражение3`, и его значение становится значением условного выражения. Таким образом, чтобы установить в `z` наибольшее из `a` и `b`, можно записать:

```
z = (a > b) ? a : b; // z=max(a,b)
```

1.2.5.3 Инструкция `switch`

Инструкция `switch` используется для выбора одного из многих путей. Она проверяет, совпадает ли значение выражения с одним из значений, входящих в некоторое множество целых констант, и выполняет соответствующую этому значению ветвь программы:

```
switch (выражение){
  case конст_выражение_1: последовательность_инструкций_1
  case конст_выражение_2: последовательность_инструкций_2
  ..................................................
  case конст_выражение_n: последовательность_инструкций_n
```

```
default: последовательность_инструкций_n+1
```

Константные выражения всех `case`-ветвей должны быть целочисленными и должны отличаться друг от друга.

Инструкция `switch` выполняется следующим образом. Вначале вычисляется `выражение`, и результат сравнивается с каждой `case`-константой. Если одна из `case`-констант равна значению выражения, управление переходит на последовательность инструкций с соответствующей `case`-меткой. Если ни с одной из `case`-констант нет совпадения, управление передается на последовательность инструкций с `default`-меткой, если такая имеется, в противном случае ни одна из последовательностей инструкций `switch` не выполняется. Ветви `case` и `default` можно располагать в любом порядке.

Для иллюстрации инструкции `switch` рассмотрим программу, которая определяет вид литеры, введенной пользователем с клавиатуры. Вид литеры — это цифра (digit), *пробельная литера* (white space) или другая литера (other). К пробельным литерам относят пробел (`' '`), *литеру табуляции* (`'\t'`) и *литеру новая-строка* (`'\n'`). Ниже представлен текст программы.

```
#include <iostream.h>
main()
// определение вида литеры, введенной с клавиатуры
// (цифра, пробельная литера или другая литера)
{   char c; // литера, вводимая с клавиатуры
    c=cin.get(); // ввод литеры с клавиатуры
    switch(c){
    case '0': case '1': case '2': case '3': case '4':
    case '5': case '6': case '7': case '8': case '9':
        // цифра
        cout<<"digit"<<endl; break;
    case ' ': case '\t': case '\n':
        // пробельная литера
        cout<<"white space"<<endl; break;
    default:
        // другая литера
        cout<<"other"<<endl; break;
    }
   return 0;
}
```

Литера, заключенная в одиночные кавычки, представляет целое значение, равное коду этой литеры (в кодировке, принятой на данной машине). Это так называемая *литерная константа*. Например, `'A'` есть литерная константа; в наборе ASCII ее значение равняется `65`. `'\t'` и `'\n'` обозначают коды литеры табуляции и литеры новая-строка соответственно.

Функция-элемент `get` объекта `cin` вводит одиночный символ из потока и возвращает этот символ в качестве *значения вызова функции*. Для вывода одиноч-

ного символа в поток используется функция-элемент `put`, которая в качестве аргумента принимает значение кода символа. Следует отметить, что стандартная библиотека ввода-вывода `<stdio.h>` включает функции для чтения и записи одной литеры `getchar` и `putchar`, аналогичные функциям-элементам `get` и `put` соответственно.

Инструкция `break` вызывает немедленный выход из инструкции `switch`. Поскольку выбор ветви `case` реализуется как переход на метку, то после выполнения одной ветви `case`, если ничего не предпринять, программа «провалится вниз» на следующую ветвь. Инструкция `break` — наиболее распространенное средство выхода из инструкции `switch`.

1.2.6 Математические функции файла `<math.h>`

Ниже приведен перечень основных математических функций, описанных в головном файле `<math.h>`. Аргументы `x` и `y` функций имеют тип `double`; все функции возвращают значения типа `double`. Углы в тригонометрических функциях задаются *в радианах*.

`sin(x)` — *синус* x;
`cos(x)` — *косинус* x;
`tan(x)` — *тангенс* x;
`asin(x)` — *арксинус* x в диапазоне $[-\pi/2, \pi/2]$, $x \in [-1, 1]$;
`acos(x)` — *арккосинус* x в диапазоне $[0, \pi]$, $x \in [-1, 1]$;
`atan(x)` — *арктангенс* x в диапазоне $[-\pi/2, \pi/2]$;
`atan2(y,x)` — *арктангенс* y/x в диапазоне $[-\pi, \pi]$;
`sinh(x)` — *гиперболический синус* x;
`cosh(x)` — *гиперболический косинус* x;
`tanh(x)` — *гиперболический тангенс* x;
`exp(x)` — *экспоненциальная функция* e^x;
`log(x)` — *натуральный логарифм* $\ln(x)$, $x > 0$;
`log10(x)` — *десятичный логарифм* $\lg(x)$, $x > 0$;
`pow(x,y)` — x^y;
`sqrt(x)` — $\sqrt{x}$, $x \geq 0$;
`ceil(x)` — *наименьшее целое* в виде `double`, которое $\geq x$;
`floor(x)` — *наибольшее целое* в виде `double`, которое $\leq x$;
`fabs(x)` — *абсолютное значение* $|x|$;
`fmod(x,y)` — *остаток от деления* x *на* y в виде числа с плавающей точкой.

1.2.7 Пример программы разветвляющейся структуры

Ниже представлен текст С-программы, вычисляющей значение функции

$$z = f(x) = \begin{cases} \sin(2x + \pi/7), \text{ если } x \leq a, \\ x^{3,21} + \tan(x), \text{ если } a < x < b, \\ \sqrt{x}\lg(x), \text{ если } x \geq b. \end{cases}$$

Значения параметров *a*, *b* и аргумента *x* вводятся с клавиатуры. Результаты вычислений выводятся на дисплей в *формате с плавающей точкой*.

```
#include <conio.h>
#include <math.h>
#include <stdio.h>
#define PI 3.14159265
main()
// вычисление значения функции z=f(x)
{
   float a, // параметр a
         b, // параметр b
         x, // аргумент x
         z; // значение функции z
    clrscr();
    // ввод значений параметров a, b и аргумента x
    printf("Введите значение параметра a: ");
    scanf("%f",&a);
    printf("Введите значение параметра b: ");
    scanf("%f",&b);
    printf("Введите значение аргумента x: ");
    scanf("%f",&x);
    // вычисление значения функции z
    if (x<=a) z=sin(2*x+PI/7);
    else if (x<b) // a<x<b
         z=pow(x,3.21)+tan(x);
    else // x>=b
         z=sqrt(x)*log10(x);
        // вывод значения функции z в формате
        //с плавающей точкой
    printf("Значение функции z = %e\n",z);
    printf("Нажмите любую клавишу...");
    getch();
    return 0;
}
```

С помощью директивы

```
#define PI 3.14159265
```

оказывается возможным указать препроцессору, чтобы он при любом появлении идентификатора `PI` в тексте программы заменял его на значение `3.14159265` ($\approx \pi$).

Функция `clrscr` библиотеки `<conio.h>` выполняет очистку экрана и перемещение курсора в левый верхний угол.

Функция `getch` библиотеки `<conio.h>` используется для ввода символов с клавиатуры без их высвечивания на экране. С помощью `getch` можно считать коды основных клавиш (ASCII-коды) и *расширенные коды*. Расширенные коды — это коды верхнего ряда клавиш, коды правой части клавиатуры и коды комбинаций клавиш `Alt`, `Ctrl`, `Shift` с другими клавишами. В случае считывания расширенных кодов при первом обращении функция `getch` возвращает нулевое значение, а ее повторный вызов позволяет получить расширенный код.

Используя возможности языка C++, вышеприведенную программу можно переписать следующим образом:

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
const double pi=3.14159265;
main()
{    // вычисление значения функции z=f(x)
    // объявление переменных и очистка экрана
    ................................
    // ввод значений параметров a, b и аргумента x
    cout<<"Введите параметр a: ";
    cin>>a;
    cout<<"Введите параметр b: ";
    cin>>b;
    cout<<"Введите аргумент x: ";
    cin>>x;
    // вычисление значения функции z
    ................................
        // вывод значения функции z в формате
        //с плавающей точкой
    cout.setf(ios::scientific, ios::floatfield);
    cout<<"Значение функции z = f(x) = "<<z<<endl;
    cout<<"Нажмите любую клавишу...";
    getch();
    return 0;
}
```

Как видно, в C++ вместо *символических констант*, которые создаются директивой препроцессора `#define`, имеется возможность использования *константных переменных*. Строка `const double pi=3.14159265;` использует *спецификацию* `const` для объявления *константы* `pi`, имеющей значение `3.14159265`. После определения константной переменной изменить ее значение нельзя. Следует отметить, что в C++ отдается предпочтение использованию константных переменных, а не символических констант. Константные переменные, в отличие от символических констант, являются данными определенного типа, и их имена видны отладчику.

Строка `cout.setf(ios::scientific,ios::floatfield);` устанавливает *экспоненциальный формат* вывода вещественных чисел (*формат с плавающей точкой*). Для отображения чисел в *формате с фиксированной точкой* следует записать `cout.setf(ios::fixed,ios::floatfield);` здесь функция-элемент `setf` объекта `cout` используется для управления установкой *флагов формата* вывода вещественных чисел `ios::scientific` или `ios::fixed`, которые содержатся в *статическом элементе данных* `ios::floatfield`.

1.3 Варианты заданий

Составить структурную схему алгоритма и написать на языке С программу вычисления функции $z = f(x)$. Варианты функций по указанию преподавателя выбирать либо из приведенных ниже, либо в соответствии с вариантами задания к лабораторной работе №5 методических указаний [1]. Значения параметров a, b и аргумента x вводятся с клавиатуры. Результаты вычислений выводятся на дисплей в *формате с плавающей точкой*.

1) $$z=\begin{cases}\left(1+x^2\right)/\left(2-11\sin x\right)+e^{\sinh x}, & \text{если } x\le a\\ \sqrt{1{,}57-x^3\sin^2 x}+4{,}1e^{2x}, & \text{если } a<x<b\\ \left(\arcsin x+\arccos x+\ln x\right)^{\tan x}, & \text{если } x\ge b\end{cases}$$

2) $$z=\begin{cases}e^x/\left(\sqrt{\sinh x}+8{,}9x+9\right), & \text{если } x\le a\\ \left(\cos^2 x+1{,}1\tan x\right)^{2{,}1}, & \text{если } a<x<b\\ \left|\ln x+\cos x^{2{,}4}-1{,}2x\right|x^{4{,}8}, & \text{если } x\ge b\end{cases}$$

3) $$z=\begin{cases}\sqrt[3]{e^{x-1/\arctan x}}+\tan^2 x+\ln x+6, & \text{если } x\le a\\ \left(\arcsin x+\sinh x\right)^{\cos x+x^2+e}, & \text{если } a<x<b\\ \left|x^{1{,}3/x}-\lg\left(1+x\right)\right|^{1{,}3+x^2}+x^5+x, & \text{если } x\ge b\end{cases}$$

4) $$z=\begin{cases}\ln\left(1{,}2x+2{,}4\right)+e^{3x}\sinh x, & \text{если } x\le a\\ 5^{x^2+1}\sqrt{1{,}2x^{3{,}5}+\sqrt{\left|1-x\right|}}, & \text{если } a<x<b\\ \left(1+\tan^2 x\right)^{\arctan x+\ln x+1}+\lg x, & \text{если } x\ge b\end{cases}$$

5) $$z=\begin{cases}\lg\left(\sqrt[3]{x^2}+\sqrt{x}+\sin x+\cos x\right), & \text{если } x\le a\\ \left(\tan^2 x+1{,}3e^{\cosh x+\tanh x}\right)^{x^2}, & \text{если } a<x<b\\ \left|\sin x-0{,}11\right|\arccos x+x^{\arcsin x}, & \text{если } x\ge b\end{cases}$$

6) $$z=\begin{cases}\left|\cos x+\sin x\right|^{1+3\tan^2 x}+x^{1{,}31}, & \text{если } x\le a\\ 8^{2{,}1x}\sqrt{\tanh x+\sqrt[7]{\left|1-x\right|}}, & \text{если } a<x<b\\ \lg\left(\sqrt[3]{x^{1{,}1}}+\sqrt{x}+6{,}1x+7\right), & \text{если } x\ge b\end{cases}$$

7) $$z=\begin{cases}4{,}5\sin\left(x+\pi/9\right)+1{,}2e^{-0{,}2x-1}, & \text{если } x\le a\\ \ln\left(x+e+x^2\right)+e^{2x}\cos x, & \text{если } a<x<b\\ x+8x^2+9x^3+8x^4+\cosh x, & \text{если } x\ge b\end{cases}$$

8) $$z=\begin{cases}\ln\left(x+5{,}7\right)+3{,}8e^{3x}\sin x, & \text{если } x\le a\\ \sqrt[7]{5{,}9^{x-1/\arctan x}}+\tan^2 x, & \text{если } a<x<b\\ \left|\cos x-0{,}5\right|^{1{,}1}+5\arccos x, & \text{если } x\ge b\end{cases}$$

9) $$z=\begin{cases}6{,}3e^{3{,}2x+7{,}9}\sin\left(6x+\pi/7\right)+5, & \text{если } x\le a\\ \left|\sin x-3{,}2\right|\cosh^2 x+x^{5{,}4}, & \text{если } a<x<b\\ \lg\left(11\pi+\arcsin x+\arccos x\right), & \text{если } x\ge b\end{cases}$$

10) $$z=\begin{cases}\left[1+\sinh^2\left(x^2-x-3\right)\right]x^{-4}, & \text{если } x\le a\\ 4.5\ln x+e^x/32+3.5x, & \text{если } a<x<b\\ \left(1+2\tan^4 x+\tan^8 x\right)^{\sqrt{|x|+17}}, & \text{если } x\ge b\end{cases}$$

11) $$z=\begin{cases}\left(1+\cosh x+\tanh x^2\right)^{\sin x+\cos x}, & \text{если } x\le a\\ 7{,}3x^{3{,}17x+5}+\arctan x^{-2\ln x}, & \text{если } a<x<b\\ \left|\arcsin x+e^x\right|^{1+5\sin^2 x}+\sqrt{\left|1-x\right|}, & \text{если } x\ge b\end{cases}$$

12) $$z=\begin{cases}16e^{-2.5\cos x}+2.5\ln\left(1+x^{2{,}8}\right), & \text{если } x\le a\\ \left(1+x^5\right)/\left(1{,}7-\cosh x\right), & \text{если } a<x<b\\ \sqrt[7]{x^3+\sqrt[5]{1.8+x^{1.3}}+\tan^{2{,}3} x}, & \text{если } x\ge b\end{cases}$$

13) $$z=\begin{cases}\left(\cosh x+\sinh x+x^2\right)^{11+x^2}, & \text{если } x\le a\\ \left|x^{1{,}3/x}-\sqrt[3]{x/1{,}3}+x^{x^{2{,}11}}\right|, & \text{если } a<x<b\\ \lg\left(\sqrt{e^{x+13}}+\ln x+\cosh x\right), & \text{если } x\ge b\end{cases}$$

14) $$z=\begin{cases}\sqrt{5\left(\sqrt[3]{x+1.2}+\sqrt[5]{x^2-4.6}\right)}, & \text{если } x\le a\\ 1.2e^{\cos 5x-1}+\left(x-1\right)^{4.5x+1}, & \text{если } a<x<b\\ tg^2 x/\left(1/2+\sinh^{\pi} x+\ln x\right), & \text{если } x\ge b\end{cases}$$

15) $$z=\begin{cases}\left(\sqrt{x-\tanh x+x^4}\right)\Big/\cos x^2, & \text{если } x\le a\\ e^{-2\sin 4x}+\lg x+\arctan x, & \text{если } a<x<b\\ 3.7x^{7.3}\sin\left|x^2\right|+4.5\cosh x^3, & \text{если } x\ge b\end{cases}$$

16) $$z=\begin{cases}\left|x^{5,7/x}-\sqrt[5]{\arctan x}+\sin x^2\right|, & \text{если } x\le a\\ \ln\left(e^{x^2}+x\lg x+\cos x\right), & \text{если } a<x<b\\ \left(1+2x^2+3x^3\right)\sinh x^{x^{1,5}+7,3}, & \text{если } x\ge b\end{cases}$$

17) $$z=\begin{cases}5.7\cos\left(3.4x-\pi/6\right)+10.2, & \text{если } x\le a\\ \left(\cos x+2.8\lg x\right)^{\arccos 15x}, & \text{если } a<x<b\\ 2.7+1.1x+4.2x^2+5.8x^{3,1}, & \text{если } x\ge b\end{cases}$$

18) $$z=\begin{cases}\sqrt{1,414-x^{3,5}\sin x^{2,2}}+\ln x, & \text{если } x\le a\\ 2e^{3,14x}\sin\left(2,1x-\pi/5\right), & \text{если } a<x<b\\ \left|x^{5,1}\cosh x-1,4\right|\arctan x^{2,6}, & \text{если } x\ge b\end{cases}$$

19) $$z=\begin{cases}\ln\sin 5.9x+3.6x^{1.2}+\cos x, & \text{если } x\le a\\ 3^{5x^{4.4}}+\cosh x^{-3.31}+\lg x, & \text{если } a<x<b\\ \sqrt{\sinh^2\arctan x^{3,94}+9\tan x}, & \text{если } x\ge b\end{cases}$$

20) $$z=\begin{cases}\ln\left(x-\sinh x\right)+\arccos 5,1x, & \text{если } x\le a\\ \sin^2 2,45x+3,81e^{x^2+x+1}, & \text{если } a<x<b\\ \left(1+x^2\right)\Big/\left(\cosh^2 x^3+x^{2x+9}\right), & \text{если } x\ge b\end{cases}$$

21) $$z=\begin{cases}2,56e^{\sinh x-11}+\cosh\sqrt{x}+\pi, & \text{если } x\le a\\ \ln\left(2,44x+3,7\right)+\sin x, & \text{если } a<x<b\\ \left|x^{1,2}-\sqrt[5]{1+x^3}\right|+\arctan x^{2,11}, & \text{если } x\ge b\end{cases}$$

22) $$z=\begin{cases}\sqrt[4]{3x^2+\sqrt[3]{1+\sin^2 x^5}+e^{2,91x}}, & \text{если } x\le a\\ \left(1+\sinh x+\lg^2 x\right)^{x^2+x+9}, & \text{если } a<x<b\\ \left|x-\arctan x+\tan^{2e^x+15x+3}x\right|, & \text{если } x\ge b\end{cases}$$

23) $$z=\begin{cases}2,2+2,4x+4,8x^2+9,6x^3, & \text{если } x\le a\\ \sqrt{\cosh^2\arctan x^5+1,32}, & \text{если } a<x<b\\ \lg\left(\sin x+\cos x+\tan^{2e}x^{2\pi}\right), & \text{если } x\ge b\end{cases}$$

24) $$z=\begin{cases}4,1+7x^2+\sin\left(8,2x+\pi/6\right), & \text{если } x\le a\\ \sqrt{\cos^2\arctan^2 x^2+e^{3x+10}}, & \text{если } a<x<b\\ \ln\left(\arcsin x+\arccos x+x^{3,2}\right), & \text{если } x\ge b\end{cases}$$

25) $$z=\begin{cases}\arccos^2 x+\arctan x+x^4+1, & \text{если } x\le a\\ e^{\pi x}+\pi^{ex}+5e^{x+1}\cos x^{x+1}, & \text{если } a<x<b\\ \lg\left(\sqrt[5]{x^2}+\sqrt{\left|3,2-x\right|}+1,73\right), & \text{если } x\ge b\end{cases}$$

26) $$z=\begin{cases}\sqrt{1,414-x^2\sin^3 x^2}+\lg^{1,21}x, & \text{если } x\le a\\ \tan^2 x+\tan x+4^{x^{1,212}}+2, & \text{если } a<x<b\\ \pi-2x+3x^2-4x^3+x^4-x^6, & \text{если } x\ge b\end{cases}$$

27) $$z=\begin{cases}\left(1+\arcsin x\right)\Big/\left(1-\arccos x\right), & \text{если } x\le a\\ e^{x-2,11}+\left(x-1\right)^{x+1}\ln\sin x, & \text{если } a<x<b\\ \sqrt{\sinh^2 x+\tanh x^2+\cos x^{4,1}}, & \text{если } x\ge b\end{cases}$$

28) $$z=\begin{cases}e^{2x^2+x+5}\Big/\left(1,5+3\sin x+\lg x\right), & \text{если } x\le a\\ \left|\cosh^{2,1}x-1,2\right|\arcsin^2 x, & \text{если } a<x<b\\ \left(\arctan x+\ln x\right)^{\cos x+2\tan x}+\pi, & \text{если } x\ge b\end{cases}$$

29) $$z=\begin{cases}\left|\cosh x+\sinh x\right|^{9+2\tan^2 x}+\lg x, & \text{если } x\le a\\ 7^{-x-7}\sqrt{\sin x+\left|1-x-x^2\right|}, & \text{если } a<x<b\\ e^x\left(1+\arctan^2 x\right)^{\sqrt{|x|+3}}+\ln^e x, & \text{если } x\ge b\end{cases}$$

30) $$z=\begin{cases}x^{1,3}\ln\left(2\sinh x^2+7\cosh x^2\right), & \text{если } x\le a\\ \left(\arctan x+1\right)\Big/\left(1+x^{x+5}\right), & \text{если } a<x<b\\ \sqrt{\left|0.5-\sin x\right|}+e^{2,6x}\cos 4,1x, & \text{если } x\ge b\end{cases}$$

1.4 Порядок выполнения работы

1.4.1 Проработать необходимый теоретический материал, опираясь на сведения и указания, представленные в предыдущем пункте.

1.4.2 Ответить на контрольные вопросы.

1.4.3 Внимательно изучить постановку задачи.

1.4.4 Выполнить анализ *области определения* и *области значений* вычисляемой функции z. Желательным является построение графика функции.

1.4.5 Разработать структурную схему алгоритма решения задачи.

1.4.6 Разработать *тестовые примеры*, которые должны включать, по крайней мере, по два значения аргумента из каждого интервала кусочно-заданной функции z. Тестовые примеры должны также отражать поведение функции z в граничных точках.

1.4.7 Написать *две* программы вычисления функции z. Одна программа для ввода-вывода данных должна использовать функции `scanf` и `printf`, а другая — объекты `cin` и `cout`. Программы *обязательно* должны содержать комментарии.

1.4.8 Выполнить *тестирование и отладку* написанных программ, используя разработанные тестовые примеры. Особое внимание следует обратить на значения функции z в граничных точках.

1.4.9 Подготовить отчет по работе.

1.5 Содержание отчета

Цель работы; постановка задачи и вариант задания; краткие теоретические сведения; математическое обоснование задачи (включает анализ области определения и области значений функции, подлежащей вычислению, а также, факультативно, график этой функции); структурная схема алгоритма решения задачи; тестовые примеры; тексты программ; результаты тестирования и отладки программ; выводы.

1.6 Контрольные вопросы

1.6.1 Опишите структуру С-программы.
1.6.2 В чем состоит препроцессорная обработка программы?
1.6.3 Перечислите и охарактеризуйте базовые типы языка С.
1.6.4 Перечислите и охарактеризуйте квалификаторы языка С.
1.6.5 Что такое приведение типа?
1.6.6 Перечислите и охарактеризуйте арифметические операторы.
1.6.7 Перечислите и охарактеризуйте операторы отношения.
1.6.8 Перечислите и охарактеризуйте логические операторы.
1.6.9 Перечислите и охарактеризуйте операторы присваивания.
1.6.10 Опишите средства ввода-вывода языка С.
1.6.11 Опишите средства ввода-вывода языка С++.
1.6.12 Опишите инструкцию if-else.
1.6.13 Что такое условное выражение?
1.6.14 Опишите инструкцию switch.
1.6.15 Опишите математические функции файла <math.h>.

2 ЛАБОРАТОРНАЯ РАБОТА №2
“ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ЦИКЛИЧЕСКОЙ СТРУКТУРЫ”

2.1 Цель работы

Получение навыков программирования алгоритмов циклической структуры на языке С. Исследование эффективности применения различных видов циклов в задаче табулирования функции.

2.2 Краткие теоретические сведения

2.2.1 Циклы в языках C/C++

Цикл представляет собой участок программы, повторяемый многократно. В языках C/C++ имеется три разновидности циклов: **`while`**, **`do-while`** и **`for`**.

2.2.1.1 Цикл `while`

Цикл `while` имеет следующий синтаксис:

```
while (<выражение>) <инструкция>
```

В цикле `while` вначале вычисляется `<выражение>`. Если его значение отлично от нуля (т. е. имеет значение истина), то выполняется `<инструкция>` и вычисление выражения повторяется. Этот цикл продолжается до тех пор, пока `выражение` не станет равным нулю (примет значение ложь), после чего вычисления возобновятся с точки, расположенной сразу за инструкцией.

Перед входом в цикл `while` обычно инициализируют одну или несколько переменных для того, чтобы `<выражение>` имело какое-либо конкретное значение. Инструкция или последовательность инструкций (*составная инструкция*), составляющих *тело цикла*, должны, как правило, изменять значения одной или нескольких переменных, входящих в `выражение`, чтобы за конечное число *итераций,* `выражение` приняло нулевое значение и цикл завершился. Цикл `while` — это *цикл с предусловием*; это значит, что решение о выполнении еще одной итерации цикла принимается *перед* началом цикла.

Цикл `while` завершается в следующих случаях:

- обращение в ноль выражения в *заголовке цикла*;
- выполнение в теле цикла инструкции `break`;
- выполнение в теле цикла инструкции `return`.

В первых двух случаях управление передается в точку, расположенную сразу за циклом. В третьем случае происходит выход из функции.

2.2.1.2 Цикл `do-while`

Цикл `do-while` имеет следующий синтаксис:

```
do
    <инструкция>
while (<выражение>);
```

В цикле do-while сначала выполняется <инструкция>, затем вычисляется <выражение>. Если оно истинно (отлично от нуля), то инструкция выполняется снова и т. д. Когда <выражение> становится ложным (обращается в ноль), цикл заканчивает работу. Цикл do-while завершается в тех же случаях, что и цикл while. Цикл do-while — это *цикл с постусловием*, т.е. проверка условия продолжения цикла (<выражение>) выполняется *после* каждого прохождения тела цикла (<инструкция>).

2.2.1.3 Цикл for

Цикл for имеет следующий синтаксис:

```
for (<выражение1>; <выражение2>; <выражение3>)
    <инструкция>
```

Каждое из трех выражений <выражение1>, <выражение2>, <выражение3> можно опускать, но точку с запятой опускать нельзя. При отсутствии *<выражения1>* или *<выражения3>* считается, что их нет в конструкции цикла; при отсутствии *<выражения2>* полагается, что его значение всегда истинно.

Обычно <выражение1> служит для инициализации *счетчика цикла*, <выражение2> — для выполнения проверки на окончание цикла, <выражение3> — для модификации счетчика цикла.

Инструкция for эквивалентна конструкции

```
<выражение1>;
while (<выражение2>)
     {
       <инструкция>
       <выражение3>;
     }
```

В языке C++ переменную можно объявить в *части инициализации* (выражение1) инструкции for. Например:

```
for (int counter=1;counter<=10;counter++)
{     // тело цикла
     ..............
}
```

2.2.1.4 Инструкции break и continue

Инструкции break и continue изменяют поток управления. Когда инструкция break выполняется в инструкциях switch, while, do-while или for, происходит немедленный выход из соответствующей инструкции, и управление передается в точку, расположенную сразу за инструкцией. Обычное назначение инструкции break — досрочно прервать цикл или пропустить оставшуюся часть инструкции switch. Необходимо заметить, что инструкция break вызывает немедленный выход из *самого внутреннего* из объемлющих ее циклов.

Инструкция continue в циклах while, do-while или for вызывает пропуск оставшейся части тела цикла, после чего начинается выполнение следующей итерации цикла. В циклах while и do-while после выполнения ин-

струкции `continue` осуществляется проверка условия продолжения цикла. В цикле `for` после выполнения инструкции `continue` выполняется *выражение приращения* (`выражение3`), а затем осуществляется проверка условия продолжения цикла (`выражение2`). Таким образом, инструкция `continue` вынуждает ближайший объемлющий ее цикл начать следующий шаг итерации.

2.2.1.5 Цикл `for` и оператор запятая

Иногда в цикле `for <выражение1>` и `<выражение3>` представляются как *списки выражений*, разделенных запятыми. В этом случае запятая используется как *оператор запятая* или *операция следования*, гарантирующая, что список выражений будет вычисляться *слева направо*. Оператор запятая имеет самый низкий приоритет (см. таблицу 2.1 в лабораторной работе №2). Значение и тип списка выражений, разделенных запятыми, равны значению и типу самого правого выражения в списке.

Оператор запятая наиболее часто применяется в цикле `for`. Этот оператор дает возможность использовать несколько выражений задания начальных значений и (или) несколько выражений приращения переменных, что позволяет вести несколько индексов (управляющих переменных) параллельно, например:

```
for (int left=0,right=n-1;left<right;left++,right--)
{     // тело цикла
     ..............
}
```

Кроме цикла `for`, оператор запятая можно использовать в тех случаях, когда последовательность инструкций мыслится как одна операция, например:

```
temp=a[i],a[i]=a[i+1],a[i+1]=temp; // обмен
```

2.2.2 Пример программы табулирования функции

2.2.2.1 Постановка задачи

Написать программу табулирования (печати таблицы значений) кусочно-заданной функции $z = f(x)$ на интервале от $x_{нач}$ до $x_{кон}$ с шагом Δx. Таблицу снабдить заголовком и шапкой. Вид функции определяется формулой

$$z = f(x) = \begin{cases} a^2, \text{ если } x < 0, \\ ax, \text{ если } 0 \le x < 10, \\ 5a, \text{ если } x \ge 10. \end{cases}$$

Если $a > 10$, то значения функции должны выводиться в виде целых чисел. Значения параметра a, а также $x_{нач}$, $x_{кон}$ и Δx вводятся с клавиатуры. Результаты вычислений выводятся в формате с фиксированной точкой.

2.2.2.2 Описание алгоритма решения задачи в словесной форме

В *словесной форме* алгоритм решения задачи можно сформулировать следующим образом:

1) Ввести с клавиатуры исходные данные: a, $x_{нач}$, $x_{кон}$ и Δx.
2) Вывести заголовок и шапку таблицы.

3) Положить $x = x_{нач}$.

4) *Вычислить значение функции* z *по вышеприведенной формуле.*

5) *Если* $a > 10$*, то привести значение функции* z *к целому типу.*

6) *Вывести на экран строку таблицы значений функции.*

7) *Увеличить значение* x *на величину* Δx.

8) Если значение x не превышает $x_{кон}$, то перейти к пункту 4, иначе закончить выполнение программы.

Так как пункты 4 — 7 алгоритма выполняются многократно, то для их выполнения необходимо организовать цикл. Напишем два варианта программы с использованием циклов `while` и `for`.

2.2.2.3 Использование цикла `while`

Ниже представлена программа табулирования функции с использованием цикла `while`:

```
#include <conio.h>
#include <stdio.h>
main()
// программа табулирования функции z=f(x)
// на интервале от xn до xk с шагом dx
{   float a, // параметр
          x, // аргумент функции z
          xn, // начальное значение аргумента x
          xk, // конечное значение аргумента x
          dx, // шаг
          z; // значение функции z
    clrscr();
    // ввод a, xn, xk, dx
  printf("Введите параметр a: "), scanf("%f",&a);
  printf("Введите xn: "), scanf("%f",&xn);
  printf("Введите xk: "), scanf("%f",&xk);
  printf("Введите шаг dx: "), scanf("%f",&dx);
  // вывод заголовка и шапки таблицы
  printf("Таблица значений функции z=f(x)\n");
  printf("  ┌──────────┬──────────┐\n");
  printf("  │    x     │  z = f(x)│\n");
  printf("  ├──────────┼──────────┤\n");
  // табулирование функции z
  x=xn; // начальное значение аргумента x
  while (x<=xk)
  { // вывод строки таблицы
   printf("  │ %-9.3f│",x); // вывод аргумента x
      // вычисление значения функции z
    (x<0) ? (z=a*a) : ((x<10) ? (z=a*x) : (z=5*a));
      // вывод значения функции z
    if (a>10)
```

```
        printf("  %-10d│\n",(int)z); // целочисленное z
        else
        printf("  %-10.3f│\n",z); // вещественное z
      x+=dx; // приращение аргумента x
     }
   printf("  └──────────┴──────────┘\n");
   printf("Нажмите любую клавишу...");  getch();
   return 0;
}
```

Поскольку формулы, определяющие функцию *z*, являются достаточно короткими, то вычисление значения функции *z* целесообразно осуществлять не с помощью инструкции `if-else`, а с помощью условного выражения.

Вывод таблицы значений функции осуществляется средствами функции `printf` стандартной библиотеки ввода-вывода `<stdio.h>`.

Воспроизвести символ большинства кодов на экране можно, нажав соответствующую ему клавишу. Но этого нельзя сделать, например, для *кодов псевдографики*, с помощью которых возможно построение рамки таблицы. Любой из символов, имеющих коды от *1* до *255*, можно воспроизвести на экране, дополнительно используя клавишу `Alt`. Для этого в среде Turbo (Borland) C++ надо установить режим работы с *цифровой клавиатурой* (правая часть клавиатуры), нажав клавишу `Num Lock`, что фиксируется индикатором `Num Lock`. Затем надо нажать клавишу `Alt`, и, не отпуская ее, на цифровой клавиатуре набрать код символа, после чего отпустить клавишу `Alt`. В результате на экране воспроизведется символ, код которого был набран. Коды символов псевдографики представлены в таблице 2.1.

Таблица 2.1 — Коды символов псевдографики (от *176* до *223*)

код	с	код	с	код	с	код	с	код	с	код	с	Код	с	код	с	код	с	код	с	код	с	код	с
176	░	180	┤	184	╕	188	╝	192	└	196	─	200	╚	204	╠	208	╨	212	╘	216	╪	220	▄
177	▒	181	╡	185	╣	189	╜	193	┴	197	┼	201	╔	205	═	209	╤	213	╒	217	┘	221	▌
178	▓	182	╢	186	║	190	╛	194	┬	198	╞	202	╩	206	╬	210	╥	214	╓	218	┌	222	▐
179	│	183	╖	187	╗	191	┐	195	├	199	╟	203	╦	207	╧	211	╙	215	╫	219	█	223	▀

2.2.2.4 Использование цикла `for`

Ниже представлена программа табулирования функции с использованием цикла **`for`**:

```
#include <conio.h>
#include <iomanip.h>
#include <iostream.h>
main()
// программа табулирования функции z=f(x)
// на интервале от xn до xk с шагом dx
{    // объявление переменных и очистка экрана
```

```
    ..............................
  // ввод a, xn, xk, dx
    cout<<"Введите параметр a:  ", cin>>a;
    cout<<"Введите xn: ", cin>>xn;
    cout<<"Введите xk: ", cin>>xk;
    cout<<"Введите шаг dx: ", cin>>dx;
  // вывод заголовка и шапки таблицы
    cout<<"Таблица значений функции z=f(x)"<<endl
        <<"  ┌──────────────────────────┐"<<endl
        <<"  │     x     │   z = f(x)   │"<<endl
        <<"  ├───────────┼──────────────┤"<<endl;

  // табулирование функции z
    cout.precision(3), cout.setf(ios::showpoint);
    cout.setf(ios::left,ios::adjustfield);
    cout.setf(ios::fixed,ios::floatfield);
  for(x=xn;x<=xk;x+=dx)
{ // вывод строки таблицы
  cout<<"  │  "<<setw(9)<<x<<'│'; // вывод аргумента x
  // вычисление значения функции z
  (x<0) ? (z=a*a) : ((x<10) ? (z=a*x) : (z=5*a));
        // вывод значения функции z
        cout<<"  "<<setw(10);
        // целочисленное или вещественное z
       (a>10) ? cout<<(int)z : cout<<z;
        cout<<'│'<<endl;
}
    cout<<"  └───────────┴──────────────┘"<<endl;
    cout<<"Нажмите любую клавишу...";
    getch();
    return 0;
}
```

В вышеприведенной программе ввод-вывод данных осуществляется с помощью объектов `cin` и `cout`. Последовательность инструкций

```
cout.precision(3), cout.setf(ios::showpoint);
cout.setf(ios::left,ios::adjustfield);
cout.setf(ios::fixed,ios::floatfield);
```

задает формат для вывода значения аргумента x и значения функции y в строке таблицы. Для установки ширины поля используется манипулятор потока `setw`. *Функция-элемент* объекта `cout precision` позволяет задать точность печатаемого вещественного числа (т. е. число разрядов справа от десятичной точки). Функция-элемент `setf` используется для управления *флагами состояния формата*.

Флаг `ios::showpoint` устанавливается для вывода чисел с обязательной печатью десятичной точки и нулевых младших разрядов (нулей в конце числа). Так, например, значение `79.0` без установки `ios::showpoint` будет напечатано как `79`, а с установкой `ios::showpoint` — как `79.00000` (количество нулей определяется заданной точностью).

Флаги `ios::left` и `ios::right` содержатся в *статическом элементе данных* `ios::adjustfield` и позволяют выравнивать печать соответственно по левой или правой границам поля.

Флаги `ios::fixed` и `ios::scientific`, управляющие форматом вывода вещественных чисел, описаны в лабораторной работе №1 настоящих методических указаний.

2.3 Варианты заданий

Вычислить и вывести на экран в виде таблицы значения функции $z = f(x)$ на интервале от $x_{нач}$ до $x_{кон}$ с шагом Δx. Таблицу снабдить заголовком и шапкой. Вид функции z выбирать в соответствии с вариантами задания к лабораторной работе №1 настоящих методических указаний. Значения параметров a, b, а также $x_{нач}$, $x_{кон}$ и Δx вводятся с клавиатуры. Результаты вычислений выводятся в формате с фиксированной точкой.

2.4 Порядок выполнения работы

2.4.1 Проработать необходимый теоретический материал, опираясь на сведения и указания, представленные в предыдущем пункте, а также в литературе [7, 8, 9].

2.4.2 Ответить на контрольные вопросы.

2.4.3 Внимательно изучить постановку задачи.

2.4.4 Выполнить анализ *области определения* и *области значений* вычисляемой функции z. Желательным является построение графика функции. Для выполнения этого пункта можно воспользоваться результатами предыдущей лабораторной работы.

2.4.5 Разработать структурную схему алгоритма решения задачи.

2.4.6 Разработать *тестовые примеры* (таблицы значений функции). При разработке тестовых примеров целесообразно использовать *отлаженную* программу из предыдущей лабораторной работы для вычисления значений функции z.

2.4.7 Написать *два* варианта программы табулирования функции z, используя циклы **`while`** и **`for`**. В одном из вариантов ввод-вывод должен базироваться на функциях **`scanf`** и **`printf`**, а в другом — на объектах **`cin`** и **`cout`**. Программы *обязательно* должны содержать комментарии.

2.4.8 Выполнить *тестирование и отладку* написанных программ, используя разработанные тестовые примеры.

2.4.9 Подготовить отчет по работе.

2.5 Содержание отчета

Цель работы; постановка задачи и вариант задания; краткие теоретические сведения; математическое обоснование задачи (включает анализ области определения и области значений функции, подлежащей вычислению, а также, факультативно, график этой функции); структурная схема алгоритма решения задачи; тестовые примеры; тексты программ; результаты тестирования и отладки программ; выводы.

2.6 Контрольные вопросы

2.6.1 Что называют циклом?

2.6.2 Какие три элемента должны входить в цикл?

2.6.3 Перечислите типы циклов в языках C/C++.

2.6.4 Охарактеризуйте цикл с предусловием.

2.6.5 Охарактеризуйте цикл с постусловием.

2.6.6 Охарактеризуйте цикл **`while`**.

2.6.7 Охарактеризуйте цикл **`do-while`**.

2.6.8 Охарактеризуйте цикл **`for`**.

2.6.9 Опишите инструкции **`break`** и **`continue`**.

2.6.10 В каких случаях целесообразно применять оператор запятая?

2.6.11 Какой тип цикла лучше использовать в задаче табулирования функции? Почему?

2.6.12 Дайте характеристику флагам состояния формата, используемым в функции **`setf`** объекта **`cout`**.

3 ЛАБОРАТОРНАЯ РАБОТА №3
“ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ОБРАБОТКИ ОДНОМЕРНЫХ МАССИВОВ”

3.1 Цель работы

Изучение особенностей представления и обработки одномерных массивов в языках C/C++ с учетом связи указателей и массивов. Получение практических навыков реализации алгоритмов обработки одномерных массивов средствами языков C/C++. Исследование особенностей обработки одномерных динамических массивов.

3.2 Краткие теоретические сведения

Для выполнения лабораторной работы необходимо изучить особенности представления и обработки одномерных массивов в языках C/C++ с учетом связи указателей и массивов, а также алгоритмы сортировки массивов методами прямого обмена, вставки и прямого выбора.

3.2.1 Одномерные массивы

Одномерный массив — это набор объектов одинакового типа, расположенных в *последовательной* группе ячеек памяти, доступ к которым осуществляется *по индексу*. В языке C одномерные массивы описываются следующим образом:

```
тип имя_массива[размер_массива];
```

В результате такого объявления компилятор отводит под массив память размером `sizeof(тип)*размер_массива` байтов. Операция (функция) `sizeof`, операндами (аргументами) которой могут являться константы, типы и переменные, в качестве результата возвращает количество байт, занимаемых ее операндом (аргументом).

Используя имя массива и индекс, можно обращаться к элементам массива: `имя_массива[индекс]`. Все элементы массива индексируются, *начиная с нуля* и заканчивая величиной, на единицу меньшей, чем размер массива, указанный при его описании. Например, если массив `data` объявлен как

```
double data[245];
```

то его `245` элементами типа `double` будут: `data[0]`, `data[1]`, ..., `data[244]`. Индекс массива может быть как целым числом, так и целочисленным выражением. Квадратные скобки, внутри которых записывается индекс массива, рассматриваются как *оператор индексации*. Оператор индексации имеет самый высокий приоритет (см. таблицу 1 в лабораторной работе №1).

Элементам массива можно присваивать начальные значения (*инициализировать* их) при объявлении массива с помощью списка начальных значений, разделенных запятыми, заключенного в фигурные скобки:

```
int n[10]={32,27,64,18,95,14,90,70,60,37};
```

Если начальных значений меньше, чем элементов в массиве, то оставшиеся элементы автоматически получают нулевые начальные значения. Например, эле-

ментам массива **n** можно присвоить нулевые начальные значения с помощью объявления

```
int n[10]={0};
```

Если размер массива не указан в объявлении со списком инициализации, то количество элементов массива будет равно количеству элементов в списке начальных значений. Например, объявление

```
int n[]={1,2,3,4,5};
```

создает массив из пяти элементов.

3.2.2 Указатели

Указатели — это переменные, которые содержат в качестве своих значений *адреса памяти*. В общем случае переменная типа *указатель* объявляется следующим образом:

```
тип *переменная_указатель;
```

Такая запись означает, что `переменная_указатель` является указателем на объект типа `тип`. Так, объявление

```
int *countPtr,count=5;
```

объявляет переменную `countPtr` типа `int *` (т. е. указатель на целое число) и читается как «`countPtr` является указателем на целое число» или «`countPtr` указывает на объект типа `int`». Однако переменная `count` объявлена как целое число, но не как указатель на целое число. Смвол `*` в объявлении относится только к `countPtr`. *Каждая переменная, объявляемая как указатель, должна иметь перед собой знак звездочки* (`*`).

Указатели должны инициализироваться либо при своем объявлении, либо с помощью оператора присваивания. Указатель может получить в качестве начального значения `0`, `NULL` или адрес. Указатель с начальными значениями `0` или `NULL` ни на что не указывает и часто используется как *ограничитель в динамических структурах*. `NULL` — это *символическая константа*, определенная в головном файле `<iostream.h>`, а также в нескольких головных файлах стандартной библиотеки языка С.

Унарный оператор адресации `&` возвращает адрес своего операнда. Например, в результате выполнения инструкции

```
countPtr=&count;
```

адрес переменной `count` будет запомнен в указателе `countPtr`.

Оператор `*`, называемый *оператором раскрытия ссылки* или *оператором разыменования (разадресации)*, возвращает значение объекта, на который указывает указатель. Например, инструкция

```
cout<<*countPtr<<endl;
```

выводит на экран значение переменной `count`, а именно `5`.

Два оператора языка C++ `new` и `delete` позволяют управлять временем жизни какой-либо переменной. Переменная может быть создана в любой точке программы с помощью оператора `new` и затем при необходимости уничтожена с помощью оператора `delete`.

В результате выполнения инструкции

```
countPtr=new int;
```

переменной-указателю `countPtr` присваивается адрес начала участка памяти размером `sizeof(int)` байт. Память выделяется *динамически* из специальной области, которая называется *куча (heap)*. Оператор `delete` освобождает ранее занятую память, возвращая ее в кучу.

Наряду с операторами `new` и `delete` в языках C/C++ существуют две функции для захвата/освобождения памяти в куче: `malloc` и `free` соответственно. Эти функции описаны в головном файле `<stdlib.h>`, а также в `<alloc.h>`. Функция `malloc(size)` запрашивает память размером `size` байт из кучи и возвращает указатель на первый байт этого участка. Функция `malloc` всегда возвращает указатель на тип `void`, поэтому для получения адреса объекта определенного типа необходимо выполнить соответствующее *преобразование типа*, например:

```
int *x;
x=(int*)malloc(sizeof(int));
```

Здесь преобразование типа `(int*)` приводит значение, возвращаемое функцией `malloc` к адресу указателя на целочисленную переменную.

Память, выделенную в куче с помощью функции `malloc`, следует освобождать с помощью функции `free`. Единственным аргументом функции `free` является указатель, ссылающийся на освобождаемую память.

При работе с указателями возможны ошибки. Рассмотрим фрагмент программы

```
int *p=new int, *q=new int;
*p=255, *q=127;
p=q, *p=63;
```

Присвоение `p=q` означает, что `p`, начиная с этого момента, указывает на ту же область памяти, что и `q`. Когда по адресу, содержащемуся в `p`, заносится значение `63` (для этого используется инструкция `*p=63;`), значение, на которое ссылается `q`, также будет равно `63` (`*q==63`). Кроме того, после присвоения `p=q` значение `*p==255` оказывается *потерянным*. Не существует доступа к этой области памяти. Она остается захваченной до конца выполнения программы. Такие явления называют *утечкой памяти*.

Если указатель является *локальной переменной*, т. е. описан в некотором блоке программы, как это имеет место во фрагменте

```
{    char *q1=new char;
    *q1='c';
}
```

то при выходе из блока он *перестанет существовать* и память, на которую он ссылается, окажется недоступной. Эта память не помечается как свободная и поэтому не может быть использована в дальнейшем.

Еще один источник ошибок — *неосвобождение памяти*, запрошенной ранее с помощью оператора `new` или функции `malloc`. Сстема не способна автоматически освобождать память в куче. Неосвобождение памяти может остаться

незамеченным в небольших программах, однако часто приводит к отказам в серьезных разработках и является плохим стилем программирования.

3.2.3 Массивы и указатели

Имя массива является *константой-указателем* и содержит адрес области памяти, которую занимает набор последовательных ячеек, соответствующих массиву.

Декларация

```
double a[10];
```

определяет массив a, состоящий из десяти последовательных объектов с именами a[0], a[1], ..., a[9]. Если pa есть указатель на double, т. е. определен как double *pa, то в результате присваивания

```
pa=a;
```

или

```
pa=&a[0];
```

pa будет указывать на нулевой элемент массива a; иначе говоря, pa будет содержать адрес элемента a[0].

Если pa указывает на некоторый элемент массива, то pa+1 *по определению* указывает на следующий элемент, pa-1 указывает на предыдущий элемент, pa+i — на *i*-й элемент после pa, а pa-i — на *i*-й элемент перед pa. Таким образом, если pa указывает на a[0] (pa==&a[0]), то *pa есть содержимое a[0], *(pa+1) есть содержимое a[1], а *(pa+i) — содержимое a[i]. Сделанные замечания верны безотносительно к типу и размеру элементов массива a. Однако нельзя выходить за границы массива и тем самым ссылаться на несуществующие объекты.

Если p и q указывают на элементы *одного и того же* массива, то к ним можно применять операторы отношения ==, !=, <, <=, >, >=. Например, отношение вида p<q истинно, если p указывает на «более ранний» элемент массива, чем q. Любой указатель всегда можно сравнить на равенство и неравенство с нулем.

Допускается также вычитание указателей. Например, если p и q ссылаются на элементы *одного и того же* массива и p<q, то q-p+1 есть число элементов от p до q включительно.

Если до начала работы программы количество элементов в массиве неизвестно, то следует использовать *динамические массивы*. Память под них выделяется во *время выполнения* программы с помощью оператора new или функции malloc, а освобождается с помощью оператора delete или функции free соответственно, например:

```
#include <stdlib.h>
main
{
    int n;
    cout<<"Введите количество элементов: ", cin>>n;
```

```
        int *a=new int[n];
        double *b=(double*)malloc(n*sizeof(double));
        // обработка массивов a и b
        ..............................
        delete []a;
        free(b);
        return 0;
    }
```

3.2.4 Пример программы обработки динамических массивов

Рассмотрим пример работы с динамическими массивами. Ниже представлен текст программы, которая в целочисленном массиве, не все элементы которого одинаковы, заменяет набор элементов, расположенных между максимальным и минимальным элементами массива (максимальный и минимальный элементы не включаются в набор), на единственный элемент, равный сумме положительных элементов в заменяемом наборе. После этого выполняется сортировка полученного массива методом прямого обмена.

```
    #include <conio.h>
    #include <iostream.h>
    main()
    // пример программы обработки динамических массивов
    {int n, // кол-во эл-тов в исходном массиве
          m; // кол-во эл-тов в результирующем массиве
     int *a, // указатель на массив (исходный
            // и результирующий)
         *temp_a; // указатель на временный
                  // (промежуточный) массив
     int i,j; // счетчики циклов
     int imax, // индекс макс. эл-та массива
         imin; // индекс мин. эл-та массива
     int ibeg, // индекс начала заменяемого набора эл-тов
         iend; // индекс конца заменяемого набора эл-тов
    int s_pos; // сумма полож. эл-тов в заменяемом наборе
    int temp; // буфер для сортировки методом
              //прямого обмена
        clrscr();
        //---------------------------------------

        // формирование исходного массива
  cout<<"Введите количество элементов массива: ", cin>>n;
     a=new int[n];
     cout<<"Введите "<<n<<" элемента(ов) массива: ";
     for(i=0;i<n;i++) cin>>a[i];
      cout<<"Исходный массив:"<<endl;
      for(i=0;i<n;i++) cout<<"a["<<i<<"]="<<a[i]<<" ";
      cout<<endl;
    // поиск индексов макс. и мин. эл-тов массива
     for(imax=imin=0,i=1;i<n;i++)
        {
```

```
            if(a[i]>a[imax]) imax=i;
            if(a[i]<a[imin]) imin=i;
        }
    cout<<"Максимальный элемент:a[" << imax << "]=" <<
    a[imax] << endl  <<"Минимальный элемент: a[" << imin
    << "]=" << a[imin] << endl;
// поиск индексов начала и конца заменяемого набора эл-тов
    if(imax<imin) ibeg=imax+1, iend=imin-1;
    else ibeg=imin+1, iend=imax-1;
    cout<<"Заменяемый набор элементов:"<<endl;
     for(i=ibeg;i<=iend;i++)
        cout<<"a["<<i<<"]="<<a[i]<<" ";
        cout<<endl;
    // расчет суммы полож. эл-тов заменяемого набора
        for(i=ibeg,s_pos=0;i<=iend;i++)
            if(a[i]>0) s_pos+=a[i];
            cout<<"Сумма положительных элементов заменяе-
мого набора: "<<s_pos<<endl;
        temp_a=a; // адрес исходного массива
        m=n+ibeg-iend; // кол-во эл-тов
                       //в результирующем массиве
        a=new int[m]; // результирующий массив
    /*---- формирование результирующего массива -------*/
    // запись эл-тов, расположенных до начала
    // заменяемого набора
        for(i=0,j=0;i<ibeg;i++,j++) a[j]=temp_a[i];
        // запись суммы полож. эл-тов заменяемого набора
        a[j++]=s_pos;
        // запись эл-тов, расположенных после
        //конца заменяемого набора
        for(i=iend+1;i<n;i++,j++) a[j]=temp_a[i];
        // освобождение памяти из-под исходного массива
        delete []temp_a;

        // вывод на экран результирующего массива
        cout<<"Результирующий массив:"<<endl;
        for(i=0;i<m;i++) cout<<"a["<<i<<"]="<<a[i]<<" ";
        cout<<endl;
        // сортировка массива методом прямого обмена
        for(i=m-1;i;i--)
            for(j=0;j<i;j++)
                if(a[j]>a[j+1])
                    temp=a[j],a[j]=a[j+1],a[j+1]=temp;
        // вывод на экран отсортированного массива
        cout<<"Отсортированный массив:"<<endl;
        for(i=0;i<m;i++) cout<<"a["<<i<<"]="<<a[i]<<" ";
        cout<<endl;
        cout<<"Нажмите любую клавишу...";
        getch();
        delete []a; // освобождение памяти
        return 0;
```

```
}
```

Исходный целочисленный массив формируется динамически, т. е. во время выполнения программы; при этом используется значение размера массива, введенное пользователем. После того, как введены элементы массива, происходит поиск индексов максимального и минимального элементов `imax` и `imin` соответственно.

Так как порядок расположения элементов в массиве заранее не известен, то сначала может следовать как максимальный (`imax<imin`), так и минимальный элемент (`imin<imax`). С учетом этого обстоятельства осуществляется поиск индексов начала и конца заменяемого набора `ibeg` и `iend` соответственно. Далее производится расчет суммы положительных элементов заменяемого набора, т. е. тех элементов исходного массива, индексы которых лежат в диапазоне от `ibeg` до `iend` включительно.

Затем динамически происходит создание результирующего массива. При этом адрес исходного массива запоминается, чтобы после того, как формирование результирующего массива будет окончено, освободить память, которую занимает исходный массив. Для вычисления размера результирующего массива необходимо из размера исходного массива (`n`) вычесть количество элементов в заменяемом наборе (`iend-ibeg+1`) и добавить единицу, соответствующую элементу, заменяющему набор:

```
n-(iend-ibeg+1)+1==n+ibeg-iend
```

Далее происходит формирование результирующего массива. Следует обратить внимание, что при копировании элементов из исходного массива в результирующий массив используются *разные* счетчики цикла, так как копируемым элементам соответствуют *различные* индексы в соответствующих массивах. После того как результирующий массив сформирован, память из-под исходного массива освобождается. В конце результирующий массив сортируется методом прямого обмена.

3.3 Варианты заданий

Даны натуральные числа n, p, q,причем $1<p<q<n$. Также дана последовательность чисел $a_1, a_2, \ldots, a_n$.

Требуется ввести с клавиатуры исходные данные, выполнить их обработку в соответствии с вариантом задания и вывести результаты обработки на экран.

Программу оформить в виде функций, выполняющих:

- выделение памяти и заполнения массива (с клавиатуры);
- обработку массива способом согласно варианту;
- сортировку массива способом согласно варианту;
- вывод массива на экран;
- освобождение памяти.

Вызов функций осуществлять из интерактивного меню.

Вариант 1

Последовательность состоит из целых чисел. Необходимо найти количество четных элементов последовательности и определить значение наименьшего из них.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 2

Последовательность состоит из целых чисел. Необходимо найти сумму положительных четных элементов последовательности.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 3

Последовательность состоит из целых чисел. Необходимо найти количество отрицательных элементов последовательности, кратных 3 и 7.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 4

Последовательность состоит из целых чисел. Необходимо найти количество элементов последовательности таких, что верно $sin(a_i) < 0$, $i=1,2...n$.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 5

Последовательность состоит из целых чисел. Необходимо найти количество элементов последовательности, отвечающих условию: $a_{i-1}<a_i>a_{i+1}$, при $i=2,3...n-1$.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 6

Последовательность состоит из целых чисел. Необходимо найти количество нечетных элементов последовательности и определить значение наибольшего из них.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 7

Последовательность состоит из вещественных чисел. Необходимо ввести два числа bи c. Все элементы последовательности, значения которых попадают в интервал $[b, c]$, заменить на 0.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 8

Последовательность состоит из вещественных чисел. Необходимо ввести число b. Определить элемент последовательности, наиболее близкий к числу b.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 9

Последовательность состоит из вещественных чисел. Необходимо поменять местами элементы с четными и нечетными индексами (для n кратного двум).

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 10

Последовательность состоит из вещественных чисел. Необходимо найти количество элементов последовательности таких, что верно $cos(a_i) > 0$, $i=1,2…n$.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 11

Последовательность состоит из вещественных чисел. Необходимо найти произведение тех элементов последовательности, значения которых лежат на отрезке$[b, c]$ (границы отрезка вводит пользователь).

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 12

Последовательность состоит из вещественных чисел. Необходимо найти такое a_i, что выполнится следующее: $max[cos(a_1), cos(a_2)… cos(a_n)] = cos(a_i)$.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 13

Последовательность состоит из целых чисел. Необходимо найти сумму отрицательных нечетных элементов последовательности.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 14

Последовательность состоит из целых чисел. Необходимо «перевернуть» последовательность (записать ее в обратном порядке).

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 15

Последовательность состоит из целых чисел. Необходимо найти сумму нечетных элементов последовательности, имеющих четные порядковые номера (индексы).

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 16

Последовательность состоит из целых чисел. Необходимо найти количество элементов последовательности таких, что верно условие $a_i=(a_{i-1}+a_{i+1})/2$, $i=2,3…n-1$.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 17

Последовательность состоит из целых чисел. Необходимо определить, есть ли в последовательности три положительных элемента, идущих подряд.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 18

Последовательность состоит из целых чисел. Необходимо указать, каких элементов в последовательности больше: четных или нечетных, положительных или отрицательных.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 19

Последовательность состоит из вещественных чисел. Необходимо найти количество элементов последовательности, отвечающих условию: $a_{i-1}<a_i<a_{i+1}$, при $i=2,3…n-1$.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 20

Последовательность состоит из вещественных чисел. Необходимо переместить элементы последовательности со значением 0 в ее начало.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 21

Последовательность состоит из вещественных чисел. Необходимо найти такое a_i, что выполнится следующее: $min[tg(a_1), tg(a_2)… tg(a_n)]= tg(a_i)$.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 22

Последовательность состоит из вещественных чисел. Необходимо проверить, является ли последовательность упорядоченной по убыванию.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 23

Последовательность состоит из вещественных чисел. Необходимо вычислить значение выражения $\sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 24

Последовательность состоит из вещественных чисел. Необходимо найти число с наибольшей дробной частью.

Упорядочить элементы последовательности a_p, a_{p+1}, …, a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

Вариант 25

Последовательность состоит из символов. Необходимо найти количество гласных букв в последовательности.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом прямого обмена.

Вариант 26

Последовательность состоит из символов. Необходимо все заглавные буквы (если они есть) заменить на соответствующие строчные.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом прямого обмена.

Вариант 27

Последовательность состоит из символов. Необходимо найти количество символов, не являющихся буквами.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом вставки.

Вариант 28

Последовательность состоит из символов. Необходимо найти количество групп символов в последовательности (группа – это несколько одинаковых символов, расположенных рядом; например, в последовательности “aabcccaaddd” четыре группы символов).

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом вставки.

Вариант 29

Последовательность состоит из символов. Необходимо найти самую длинную последовательность цифр в исходном массиве.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по убыванию, используя алгоритм сортировки методом прямого выбора.

Вариант 30

Последовательность состоит из символов. Необходимо разделить символы последовательности так, чтобы в левой ее части располагались гласные буквы, а в правой – согласные.

Упорядочить элементы последовательности a_p, a_{p+1}, ..., a_q по возрастанию, используя алгоритм сортировки методом прямого выбора.

3.4 Порядок выполнения работы

3.4.1 Проработать необходимый теоретический материал, опираясь на сведения и указания, представленные в предыдущем пункте и литературе [5,7,9,10].

3.4.2 Ответить на контрольные вопросы.

3.4.3 Внимательно изучить постановку задачи.

3.4.4 Разработать структурную схему алгоритма решения задачи.

3.4.5 Разработать *тестовые примеры*.

3.4.6 Написать программу, решающую поставленную задачу. Программа *обязательно* должна содержать комментарии.

3.4.7 Выполнить *тестирование и отладку* написанной программы, используя разработанные тестовые примеры.

3.4.8 Подготовить отчет по работе.

3.5 Содержание отчета

Цель работы; постановка задачи и вариант задания; краткие теоретические сведения; структурная схема алгоритма решения задачи; тестовые примеры; текст программы; результаты тестирования и отладки программы; выводы.

3.6 Контрольные вопросы

3.6.1 Как в языках C/C++ описываются одномерные массивы?

3.6.2 Каковы особенности инициализации массива при его объявлении?

3.6.3 Как объявить переменную типа указатель?

3.6.4 Охарактеризуйте оператор адресации **`&`** и оператор раскрытия ссылки **`*`**.

3.6.5 Охарактеризуйте операторы **`new`** и **`delete`**.

3.6.6 Опишите функции **`malloc`** и **`free`**.

3.6.7 Перечислите наиболее распространенные ошибки, связанные с использованием указателей.

3.6.8 Охарактеризуйте связь между массивами и указателями.

3.6.9 Какие действия можно выполнять над указателями?

3.6.10 Что такое динамические массивы?

3.6.11 Как выделить память под динамический массив?

3.6.12 Опишите алгоритмы сортировки массивов методами прямого обмена, вставки и прямого выбора. Каковы особенности реализации данных алгоритмов средствами языков C/C++?

4 ЛАБОРАТОРНАЯ РАБОТА №4 “ОБРАБОТКА ДВУМЕРНЫХ МАССИВОВ С ПОМОЩЬЮ ФУНКЦИЙ”

4.1 Цель работы

Изучить основные принципы работы с массивами в языках C/C++.
Исследовать способы передачи параметров в функции.

4.2 Краткие теоретические сведения

4.2.1 Описание и обработка двумерных массивов

Двумерный массив в C/C++ представляется как массив, состоящий из массивов. Для этого при описании массива в квадратных скобках указывают вторую размерность:

```
int a[3][5]; // матрица 3x5
```

Такой массив хранится в непрерывной области памяти по строкам (рисунок 4.1):

a_{00} a_{01} a_{02} a_{03} a_{04}	a_{10} a_{11} a_{12} a_{13} a_{14}	a_{20} a_{21} a_{22} a_{23} a_{24}
← 0-ая строка →	← 1-ая строка →	← 2-ая строка →

Рисунок 4.1 — Хранение двумерного массива в памяти ЭВМ

В памяти сначала располагается одномерный массив `a[0]`, т.е. нулевая строка, затем массив `a[1]` — первая строка и т.д.

Для доступа к отдельному элементу массива применяется конструкция вида `a[i][j]`, где `i` — номер строки, `j` — номер столбца. Каждый индекс изменяет свои значения, начиная с нуля. Можно обратиться к элементу массива и с помощью указателей, например, `*(*(a+i)+j)` или `*(a[i]+j)`. Следует обратить внимание на то, что `*(a+i)` должно быть указателем.

При описании массивов можно задавать начальные значения его элементов. Элементы массива инициализируются в порядке их расположения в памяти:

```
int a[3][5]={1,2,3,4,5,1,2,3,4,5,1,2,3,4,5};
```

Если количество значений в фигурных скобках превышает количество элементов в массиве, то выдается сообщение об ошибке. Если значений меньше, то оставшиеся элементы массива инициализируются значениями по умолчанию (для основных типов это ноль). Можно задать начальные значения не для всех элементов массива. Для этого список значений для каждой строки массива заключается в дополнительные фигурные скобки:

```
int a[3][5]={{1,2}, {1,2}, {1,2}};
```

Здесь нулевой и первый столбцы матрицы заполняются единицами и двойками. Остальные элементы массива обнуляются.

При явном задании хотя бы одного инициализатора для каждой строки количество строк в описании массива можно не указывать:

```
int a[][5]={{1, 1, 1, 1, 1}, {2, 2, 2}, {3, 3}};
```

Напишем программу, которая для целочисленной матрицы 10x20 определяет среднее арифметическое ее элементов и количество положительных элементов в каждой строке. Алгоритм решения этой задачи очевиден. Для вычисления среднего арифметического элементов массива требуется найти их общую сумму, после чего разделить ее на количество элементов. Порядок просмотра массива роли не играет. Определение положительных элементов каждой строки требует просмотра матрицы по строкам. Обе величины вычисляются при одном просмотре матрицы.

```
#include <iostream.h>
void main()
{   const int nrow=10, ncol=20;
    int a[nrow][ncol];
    int i,j;
    cout<<"Введите элементы массива: "<<endl;
    for(i=0;i<nrow;i++)
        for(j=0;j<ncol;j++) cin>>a[i][j];
    int n; // счетчик положительных элементов
    float s=0; // сумма элементов
    for(i=0;i<nrow;i++)
      { n=0;
        for(j=0;j<ncol;j++)
         {
            s+=a[i][j];
            if(a[i][j]>0) n++;
         }
       cout<<"В строке "<<i
       << " кол-во положительных элементов: "<<n<<endl;
      }
    s/=nrow*ncol;
    cout<<"Среднее арифметическое равно: "<<s<<endl;
 }
```

Здесь размерности массива заданы именованными константами nrow и ncol, что позволяет легко их изменять. Для упрощения отладки рекомендуется задать небольшие значения этих констант. При вычислении количества положительных элементов для каждой строки выполняются однотипные действия: обнуление счетчика n, просмотр каждого элемента строки и сравнение его с нулем, при необходимости увеличение счетчика на единицу, а после окончания вычислений — вывод результирующего значения счетчика.

Рекомендуется после ввода матрицы выполнять ее контрольный вывод на экран. Для того чтобы элементы матрицы располагались один за другим, следует использовать манипулятор setw(), устанавливающий ширину поля для выводимого значения. Для использования манипулятора следует подключить заголовочный файл <iomanip.h> и дополнить программу следующим кодом:

```
#include <iomanip.h>
```

```
.................................
for(i=0;i<nrow;i++)
{  for(j=0;j<ncol;j++) cout<<setw(6)<<a[i][j];
   cout<<endl;
}
```

Здесь после вывода каждой строки матрицы выводится символ перевода строки `endl`. Необходимый форматированный вывод матрицы можно также выполнить с помощью функции `printf`.

4.2.2 Динамические массивы

В динамической области памяти можно создавать двумерные массивы с помощью операции `new` или функции `malloc`. Рассмотрим первый способ. При выделении памяти сразу под весь массив количество строк можно задавать с помощью переменной, а количество столбцов должно быть определено с помощью константы. После слова new записывается тип элементов массива, а затем в квадратных скобках его размерности, например [10]:

```
int n;
const int m=5;
cin>>n;
int (*a)[m]=new int[n][m]; // 1
int **b=(int **) new int[n][m]; // 2
```

Здесь в строке 1 адрес начала выделенного с помощью `new` участка памяти присваивается переменной `a`, определенной как указатель на массив из `m` элементов типа `int`. Именно такой тип значения возвращает в данном случае операция `new`. Скобки для `(*a)` необходимы, поскольку без них конструкция интерпретировалась бы как массив из указателей.

В строке 2 адрес начала выделенного участка памяти присваивается переменной `b`, которая описана как указатель на указатель типа `int`. Поэтому требуется выполнить преобразование типа `(int **)`.

Обращение к элементам динамических массивов производится точно так же, как к элементам статических массивов, например, `a[i][j]`.

Для того чтобы понять, отчего динамические массивы описываются так, напомним, что доступ к элементу двумерного массива можно выполнить с помощью конструкции `*(*(a+i)+j)`. Поскольку здесь применяются две операции раскрытия ссылки, то переменная, в которой хранится адрес начала массива, должна быть указателем на указатель.

Когда обе размерности массива задаются на этапе выполнения программы, то память под двумерный массив можно выделить следующим образом:

```
int nrow,ncol;
cout<<"Введите количество строк и столбцов: "
cin>>nrow>>ncol;
int **a=new int *[nrow]; // 1
for(int i=0;i<nrow;i++)  // 2
    a[i]=new int[ncol];  // 3
```

В строке 1 объявляется переменная a типа указатель на указатель типа int и выделяется память под массив, каждый элемент которого есть указатель на строку матрицы. Всего элементов nrow. В строке 2 организуется цикл для выделения памяти под строки. В строке 3 каждому указателю на строку присваивается адрес начала области памяти, достаточной для хранения ncol элементов типа int.

4.2.3 Функции

Функция — это группа операторов, вызываемая по имени и возвращающая в точку вызова предписанное значение. Формат простейшего заголовка функции: <тип> <имя>(<список формальных параметров>)

Например, заголовок функции main обычно имеет вид

```
int main();
```

Это означает, что никаких параметров функции не передается, а возвращает она одно значение типа int. Функция может и не возвращать значения, в этом случае должен быть указан тип void.

Имена формальных параметров при задании *прототипа функции* можно не указывать. Достаточно указать их тип: `double sin(double);`

Здесь записано, что функция имеет имя sin, вычисляет значение синуса типа double и для этого ей надо передать аргумент типа double.

Хороший стиль программирования требует, чтобы все, что передается в функцию и обратно, отражалось в ее заголовке.

Заголовок задает *объявление функции*. *Определение функции*, кроме заголовка, включает ее тело, т.е. операторы, которые выполняются при вызове функции, например:

```
int sum(int a,int b)
{    return a+b; // тело функции
}
```

Тело функции представляет собой блок, заключенный в фигурные скобки. Для возврата результата, вычисленного в функции, служит оператор return. После него указывается выражение, значение которого и передается в точку вызова функции. Функция может иметь несколько операторов возврата.

Для вызова функции указывают ее имя, а также передают ей значения *фактических параметров*:

```
<имя функции>(<список фактических параметров>);
```

Порядок следования аргументов в списке фактических параметров и их типы должны строго соответствовать *списку формальных параметров*. Пример обращения к определенной выше функции sum:

```
int summa,a=5;
summa=sum(a,5);
```

Рассмотрим *механизм передачи параметров* в функцию. Он весьма прост. Параметры передаются в функцию как *параметры-значения*. Иными словами, функция работает с копией значений, которые ей передаются. Кстати, именно по-

этому на месте таких параметров можно при вызове задавать выражения, например: `summa=sum(a*10+5,a+3);`

Для передачи в функцию параметров способом *«параметр-переменная»* в языке С используют указатели. Определим функцию `swap`, осуществляющую обмен значениями двух переменных.

```
void swap(int *x,int *y)
{   int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
```

Здесь параметры `x` и `y` являются указателями и позволяют функции осуществлять доступ к ячейкам памяти вызвавшей ее программы и менять их значения местами. Чтобы функция `swap` выполняла желаемые действия, необходимо при вызове на место параметров `x` и `y` подставить адреса соответствующих ячеек памяти. Для этого используется операция взятия адреса `&`:

`swap(&a,&b);` в этом случае функция `swap` поменяет местами значения переменных `a` и `b`.

В языке С++ для передачи параметров способом «параметр-переменная» можно использовать *ссылочные переменные*. *Ссылка* — это переменная, которая ассоциирована с другой переменной и содержит ее адрес:

```
int ivar=1234;
int &iref=ivar; // ссылка iref
```

Здесь `iref` — это ссылка, которой присваивается адрес переменной `ivar`. Ссылки должны инициализироваться при их объявлении.

Не существует операторов, непосредственно производящих действия над ссылками. Все операции выполняются над ассоциированными значениями. Так, оператор `iref++` выполняет инкремент значения `ivar`.

Сами по себе ссылки применяются редко. Их обычно используют в качестве параметров функций. Так, функцию `swap` можно переписать в виде

```
void swap(int &x,int &y)
{   int temp;
    temp=x;
    x=y;
    y=temp;
}
```

При этом упрощается вызов функции: `swap(a,b);`

В этом случае `x` будет ссылаться на `a`, а `y` — на `b`, и функция `swap` выполнит обмен значениями `a` и `b`.

При определении функции входные данные обычно передаются по значению, а результаты ее работы — по ссылке или указателю.

Следует иметь ввиду, что возвращение ссылки или указателя из функции может привести к проблемам, если переменная, на которую делается ссылка, вышла из области видимости. Например,

```
int & func()
{  int x;
    x=5;
    return x;
}
```

В этом случае попытка возвратить ссылку на локальную переменную приведет к ошибке.

Эффективность передачи в функцию адреса вместо самой переменной ощутима и в скорости работы, особенно если используются массивы.

Если в функцию требуется передать достаточно большой объект, однако его модификация не предусматривается, то используется *константный указатель* или ссылка:

```
const int * func(const int * number);
const int & func(const int & number);
```

Здесь функция `func` принимает и возвращает указатель или ссылку на константный объект типа `int`. Любая попытка модифицировать такой объект внутри функции вызовет сообщение компилятора об ошибке.

В заключение рассмотрим передачу массивов функциям. Пусть требуется написать функцию, суммирующую элементы массива. Предположим, что имеются следующие описания:

```
#define MAX 100;
int a[MAX];
```

Тогда можно объявить функцию со следующим прототипом:

```
int SumOfA(int a[MAX]);
```

Однако будет лучше оставить квадратные скобки пустыми и добавить второй аргумент, определяющий размер массива:

```
int SumOfA(int a[],int n);
```

Необходимо помнить, что в этом случае реально в функцию передается указатель на тип элемента массива. Таким образом, изменение любого элемента массива внутри функции повлияет и на оригинал. Поэтому лучше сразу передать в функцию указатель на нулевой элемент константного массива, например:

```
int SumOfA(const int *a,int n);
```

Аналогичным образом поступают и при передаче двумерных массивов в функцию. При передаче двумерного массива в функцию в списке формальных параметров обычно указывают только количество столбцов массива. Количество строк передают в виде дополнительного параметра:

```
int SumOfMatrix(int matrix[][10],int nrow);
```

Поскольку доступ к двумерному массиву можно получить с помощью указателя на указатель, то прототип функции можно объявить и так:

```
int SumOfMatrix(int **matrix,int nrow,int ncol);
```

Здесь `nrow` — количество строк матрицы, а `ncol` — количество столбцов.

4.2.4 Обработка матриц с помощью функций

Пусть требуется написать программу, которая упорядочивает строки прямоугольной целочисленной матрицы по возрастанию сумм их элементов. Начинать решение задачи необходимо с четкого описания того, что является ее исходным данными и результатами, и каким образом они будут представлены в программе.

Исходные данные. Размерность матрицы будем задавать с помощью символических констант. В качестве типа значений элементов матрицы будем использовать тип `int`.

Результаты. Результатом является та же матрица, но упорядоченная. Это значит, что не следует отводить для результата новую область памяти, а необходимо упорядочить матрицу на том же месте.

Промежуточные величины. Кроме конечных результатов, в любой программе есть промежуточные, а также служебные переменные. Следует выбрать их тип и способ хранения. Очевидно, что если требуется упорядочить матрицу по возрастанию сумм элементов ее строк, эти суммы необходимо вычислить и где-то хранить. Поскольку все они потребуются при упорядочивании, их запишем в массив, количество элементов которого соответствует количеству строк матрицы, а `i`-ый элемент содержит сумму элементов `i`-ой строки. Сумма элементов строки может превысить диапазон значений, допустимых для отдельного элемента строки, поэтому для элементов этого массива необходимо выбрать тип `long`.

После того, как выбраны структуры данных, можно приступить разработке алгоритма, поскольку алгоритм зависит от того, каким образом представлены данные.

Алгоритм работы программы. Для сортировки строк воспользуемся алгоритмом прямого выбора. При этом будем одновременно с перестановкой элементов массива выполнять обмен двух строк матрицы. Вычисления в данном случае состоят из 2-х шагов: формирование сумм элементов каждой строки и упорядочивание матрицы. Упорядочивание состоит в выборе наименьшего элемента и обмена с первым из рассматриваемых. Разработанные алгоритмы полезно представить в виде блок-схемы. Функционально завершение части алгоритма отделяется пустой строкой и/или комментарием. Не следует стремиться написать всю программу сразу. Сначала рекомендуется написать и отладить фрагмент, содержащий ввод исходных данных. Затем целесообразно перейти к следующему фрагменту алгоритма.

Ниже приведен текст программы сортировки строк матрицы по возрастанию сумм элементов.

```
#include <iostream.h>
#include <iomanip.h>
// прототипы функций
// функция, суммирующая элементы строк
void SummaStrok(int **a,int m,int n,long *v);
// функция, выполняющая вывод матрицы и
```

```
// суммы элементов в строках
void Vyvod(int **a,int m,int n,long *v);
// функция, выполняющая сортировку строк
void Sort(int **a,int m,int n,long *v);
int main()
{ int m,n,i,j;
cout<<"Введите количество строк и столбцов матрицы:";
    cin>>m>>n;
    int **a=new int *[m]; // выделение памяти
    for(i=0;i<m;i++)      // под матрицу
        a[i]=new int[n];
    for(i=0;i<m,i++) // ввод матрицы
        for(j=0;j<n;j++) cin>>a[i][j];
    int *v=new long[m]; // вспомогательный массив
 SummaStrok(a,m,n,v); // суммирование элементов строк
 Vyvod(a,m,n,v); // контрольная печать
 Sort(a,m,n,v); // сортировка
 Vyvod(a,m,n,v); // вывод результатов
 for(i=0;i<m;i++)      // под матрицу
    delete []a[i];  // освобождение памяти
    delete []a;
return 0;
}
  //---------------------------------------
void SummaStrok(int **a,int m,int n,long *v)
{ int i,j;
  for(i=0;i<m;i++)
     {
       v[i]=0;
       for(j=0;j<n;j++)  v[i]+=a[i][j];
    }
}
  //---------------------------------------
void Vyvod(int **a,int m,int n,long *v)
{  int i,j;
   for(i=0;i<m;i++)
   { for(j=0;i<n;j++)
        cout<<setw(4)<<a[i][j]<<" ";
        cout<<endl;
   }
  for(i=0;i<m;i++)
      cout<<setw(4)<<v[i]<<" ";
  cout<<endl;
}
 //---------------------------------------
// сортировка строк матрицы методом прямого выбора
void Sort(int **a,int m,int n,long *v)
{    long buf_sum;
```

```
     int min,buf_a;
     int i,j;
for(i=0;i<m-1;i++)
   {
    min=i;
    for(j=i+1;j<m;j++) // поиск минимального элемента
      if(v[j]<v[min]) min=j;
       buf_sum=v[i]; // обмен значений v
       v[i]=v[min];
       v[min]=buf_sum;
       for(j=0;j<n;j++)//перестановка строк матрицы a
         {  buf_a=a[i][j];
            a[i][j]=a[min][j];
            a[min][j]=buf_a;
         }
   }
}
```

В функции `Sort` используются две буферные переменные: `buf_sum`, через которую осуществляется обмен двух значений сумм (имеет такой же тип, что и сумма), `buf_a` для обмена значений элементов массива (того же типа, что и элементы массива).

4.3 Варианты заданий

Каждый пункт нижеприведенного задания оформить в виде функции. Все необходимые данные для функций должны передаваться им в качестве параметров. Ввод-вывод данных и результатов также организовать с помощью соответствующих функций. Интерфейс пользователя программы оформить в виде меню. Матрицу представить в виде динамического массива, не забывать освобождать выделенную память.

Вариант 1

Определить количество строк матрицы, не содержащих ни одного нулевого элемента.

Осуществить циклический сдвиг элементов прямоугольной матрицы на k элементов вправо, k может быть больше количества элементов в строке или столбце.

Вариант 2

Определить количество строк матрицы, содержащих хотя бы один отрицательный элемент.

Характеристикой столбца матрицы назовем сумму его элементов, кратных 3. Переставляя столбцы заданной матрицы, расположить их в порядке возрастания характеристик.

Вариант 3

Определить номер строки матрицы, содержащей максимальное количество четных элементов.

Определить номер столбца, в котором находится самая длинная серия одинаковых элементов.

Вариант 4

Определить количество строк матрицы с положительной суммой элементов.

Поменять местами две строки и два столбца (номера строк и столбцов вводит пользователь).

Вариант 5

Посчитать сумму элементов всех строк, не содержащих положительных элементов.

Соседями элемента a_{ij} в матрице A назовем элементы a_{kl}, для которых верно следующее: $i-1 \le k \le i+1$, $j-1 \le l \le j+1$. Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов заданной матрицы размером 5x5.

Вариант 6

Определить количество столбцов матрицы, не содержащих ни одного нулевого элемента.

Для заданной квадратной матрицы найти такие k, что k-я строка матрицы совпадает с k-м столбцом.

Вариант 7

Определить количество столбцов матрицы, содержащих хотя бы один отрицательный элемент.

Характеристикой строки целочисленной матрицы назовем сумму ее положительных четных элементов. Переставляя строки заданной матрицы, расположить их в порядке возрастания характеристик.

Вариант 8

Определить номер столбца матрицы, содержащего минимальное количество четных элементов.

Определить максимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

Вариант 9

Определить количество столбцов матрицы с положительной суммой элементов.

Осуществить циклический сдвиг элементов прямоугольной матрицы на k элементов вверх, k может быть больше количества элементов в строке или столбце.

Вариант 10

Посчитать сумму элементов всех столбцов, состоящих только из положительных элементов.

Соседями элемента a_{ij} в матрице A назовем элементы a_{kl}, для которых верно следующее: $i-1 \le k \le i+1$, $j-1 \le l \le j+1$. Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов заданной матрицы размером 5x5.

Вариант 11

Определить количество строк матрицы, не содержащих ни одного четного элемента.

Осуществить циклический сдвиг элементов прямоугольной матрицы на k элементов влево, k может быть больше количества элементов в строке или столбце.

Вариант 12

Определить количество строк матрицы, содержащих хотя бы один нулевой элемент.

Характеристикой столбца матрицы назовем сумму модулей его отрицательных элементов. Переставляя столбцы заданной матрицы, расположить их в соответствии с убыванием характеристик.

Вариант 13

Определить номер строки матрицы, содержащей максимальное количество отрицательных элементов.

Найти максимальное из чисел, встречающихся в заданной матрице более одного раза.

Вариант 14

Определить количество строк матрицы с четной суммой элементов.

Осуществить циклический сдвиг элементов прямоугольной матрицы на k элементов вниз, k может быть больше количества элементов в строке или столбце.

Вариант 15

Определить сумму и количество четных элементов матрицы.

Найти позиции всех седловых точек матрицы.

Примечание. Матрица A имеет седловую точку a_{ij}, если a_{ij} является минимальным элементом в i-й строке и максимальным в j-м столбце.

Вариант 16

Определить количество столбцов матрицы, не содержащих ни одного элемента, кратного 3.

Осуществить циклический сдвиг элементов прямоугольной матрицы на k элементов влево или вверх (в зависимости от введенного режима), k может быть больше количества элементов в строке или столбце.

Вариант 17

Определить количество столбцов матрицы, содержащих хотя бы один нулевой элемент.

Характеристикой строки матрицы назовем ее максимальный по модулю элемент. Переставляя строки заданной матрицы, расположить их в соответствии с убыванием характеристик.

Вариант 18

Определить номер столбца матрицы, содержащего минимальное количество отрицательных элементов.

Определить номер строки, в которой находится самая длинная серия отрицательных элементов.

Вариант 19

Определить количество столбцов матрицы с нечетной суммой элементов.

Транспонировать матрицу, учесть, что матрица не обязательно квадратная.

Вариант 20

Определить количество и сумму нечетных отрицательных элементов.

Соседями элемента a_{ij} в матрице A назовем элементы a_{kl}, для которых верно следующее: $i-1 \le k \le i+1$, $j-1 \le l \le j+1$. Элемент матрицы называется локальным максимумом, если он строго больше всех имеющихся у него соседей. Подсчитать количество локальных максимумов заданной матрицы размером 5x5.

Вариант 21

Определить количество строк матрицы, не содержащих ни одного отрицательного элемента.

Путем перестановки элементов квадратной вещественной матрицы добиться того, чтобы ее максимальный элемент находился в левом верхнем углу, следующий по величине — в позиции (2,2), следующий по величине — в позиции (3,3) и т. д., заполнив таким образом всю главную диагональ.

Вариант 22

Определить количество строк матрицы, содержащих хотя бы один четный положительный элемент.

Характеристикой столбца матрицы назовем его минимальный нечетный элемент. Переставляя столбцы заданной матрицы, расположить их в порядке роста характеристик.

Вариант 23

Определить номер строки матрицы, содержащей максимальное количество нулевых элементов.

Определить минимум среди сумм модулей элементов диагоналей, параллельных побочной диагонали матрицы.

Вариант 24

Для каждой строки найти сумму элементов и определить номер строки, содержащую максимальную сумму.

Проверить является ли матрица симметричной относительно главной диагонали.

Вариант 25

Посчитать сумму элементов таких строк матрицы, в которых сумма всех элементов не превышает максимальный по модулю элемент строки.

Соседями элемента a_{ij} в матрице A назовем элементы a_{kl}, для которых верно следующее: $i-1 \le k \le i+1$, $j-1 \le l \le j+1$. Операция сглаживания матрицы дает новую матрицу того же размера, каждый элемент которой получается как среднее арифметическое имеющихся соседей соответствующего элемента исходной матрицы. Построить результат сглаживания заданной вещественной матрицы размером 5x5.

Вариант 26

Определить количество столбцов матрицы, не содержащих ни одного положительного элемента.

Уплотнить заданную матрицу, удаляя из нее строки и столбцы, заполненные одинаковыми элементами. Повторять до тех пор, пока таковых в матрице не останется или не останется только один элемент.

Вариант 27

Определить количество столбцов матрицы, содержащих хотя бы один отрицательный элемент, кратный 3.

Характеристикой строки матрицы назовем сумму ее положительных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с убыванием характеристик.

Вариант 28

Определить номер столбца матрицы, содержащего минимальное количество нулевых элементов.

Определить, является ли заданная квадратная матрица симметрической (если выполняется условие $A^T=A$).

Вариант 29

Определить количество столбцов матрицы с отрицательной суммой элементов.

Определить, является ли заданная квадратная матрица антисимметрической (если выполняется условие $A^T=-A$).

Вариант 30

Посчитать сумму элементов таких столбцов матрицы, в которых количество четных элементов больше, чем нечетных.

Найти позиции всех седловых точек матрицы.

Примечание. Матрица A имеет седловую точку a_{ij}, если a_{ij} является максимальным элементом в i-й строке и минимальным в j-м столбце.

4.4 Порядок выполнения работы

4.4.1 Выбрать типы данных для хранения исходных данных, промежуточных величин и результатов.

4.4.2 Разработать алгоритм решения задачи в соответствии с вариантом.

4.4.3 Составить программу, оформив ввод данных, вывод результатов, необходимые вычисления в виде функций.

4.4.4 Разработать тестовые примеры, которые проверяют все ветви алгоритма и возможные диапазоны данных. В качестве тестового примера рекомендуется ввести значения элементов массива, близкие к максимально возможным. Дополнительно рекомендуется проверить работу программы, когда массив состоит из одной или двух строк (столбцов), поскольку многие ошибки при написании циклов связаны с неверным указанием их граничных значений.

4.4.5 Выполнить отладку программы.

4.4.6 Получить результаты для всех вариантов тестовых примеров.

4.5 Содержание отчета

Цель работы, вариант задания, структурные схемы алгоритмов всех функций, описание тестовых примеров, текст программы, анализ результатов вычислений, выводы.

4.6 Контрольные вопросы

4.6.1 Как в языках C/C++ представляются двумерные массивы?

4.6.2 Как осуществляется доступ к элементам двумерного массива?

4.6.3 Каким образом осуществляется инициализация двумерных массивов?

4.6.4 Как выделить память под динамический двумерный массив?

4.6.5 Приведите формат заголовка функции.

4.6.6 Что является телом функции?

4.6.7 В чем состоит механизм передачи параметров в функцию?

4.6.8 Что такое ссылочные переменные? В каких случаях их целесообразно использовать?

4.6.9 Что такое аргументы по умолчанию?

4.6.10 Как осуществляется передача массивов в функции?

4.6.11 Напишите функцию, выполняющую поиск номера строки матрицы, в которой находиться минимальный элемент.

4.6.12 Напишите функцию, выполняющую перестановку 2-х заданных строк (столбцов) матрицы.

5 ЛАБОРАТОРНАЯ РАБОТА №5

ПРОГРАММИРОВАНИЕ ОПЕРАЦИЙ НАД СТРОКАМИ И ФАЙЛАМИ

5.1 Цель работы

Изучение основных операций над строками и файлами, программирование операций обработки строк текстовых файлов, исследование свойств файловых указателей.

5.2 Краткие теоретические сведения

5.2.1 Строки

В языке С *строка* представляет собой массив символов, завершающийся литерой ‘\0’. При объявлении строкового массива необходимо принимать во внимание наличие терминатора (‘\0’) в конце строки, отводя под строку на один байт больше. Удобно задавать длину строки именованной константой:

```
const int len_str = 80;
char stroka[len_str];
```

В переменной **stroka** можно хранить не 80 символов, а только 79.

Строки можно при описании *инициализировать* строковой константой. При этом нуль-символ (‘\0’) добавляется в строку автоматически:

```
char a[10] = “язык си”;
```

Если строка при определении инициализируется, её длину можно не указывать:

```
char a[] = “язык си ”;
```

В этом случае компилятор сам выделит память достаточную для размещения всех символов и нуль-литеры.

Для размещения строки в *динамической памяти* необходимо описать указатель на **char**, а затем выделить память с помощью функции **malloc()** или **new**:

```
char *s1 = (char*) malloc(m*sizeof(char));
char *s2 =  new char[m];
```

Здесь **m** – переменная, определяющая длину строки.

Для ввода-вывода строк можно использовать как *объекты* **cin** и **cout** языка С++, так и функции языка С. Рассмотрим сначала первый способ:

```
#include <iostream.h>
int main() {
  const int n = 80;
  char s[n];
  cin>>s;   cout<<s<<endl;  }
```

Таким образом, строка вводится аналогично числовым переменным. Однако, если ввести строку, состоящую из нескольких слов, разделенных пробелами, то будет введено только первое слово. Это связано с тем, что ввод выполняется до первого пробельного символа (**‘ ’, ‘\t’, ‘\n’**). Поэтому для ввода строки лучше использовать функции **getline** или **get**:

```
#include <iostream.h>
int main() {
  const int n = 80;
  char s[n];
  cin.getline(s,n);  cout<<s<<endl;
  cin.get(s,n);  cout<<s<<endl;
}
```

Функция **getline** считывает из входного потока **n**-1 символов или менее (если символ **'\n'** встретится раньше) и записывает их в строковую переменную **s**. Сам символ **'\n'** тоже считывается, но не записывается в **s**, вместо него записывается нуль-литера (**'\0'**). Если в потоке более **n**-1 символа, то следующий ввод будет выполняться, начиная с первого несчитанного символа.

Функция **get** работает аналогично, но оставляет в потоке символ '**\n**'. Поэтому перед повторным вызовом **get** необходимо удалять символ '**\n**' из входного потока. Для этого нужно вызвать **get** без параметров – **cin.get()**.

В языке С для ввода и вывода строк используются функции **scanf** и **printf** со спецификацией **%s**:

```
#include <stdio.h>
int main() {
  const int n = 40;
  char s[n];
  scanf("%s",s); printf("%s",s);
}
```

В функции **scanf** перед **s** операция взятия адреса (**&**) опущена, потому что имя массива является указателем на его начало. Функция **scanf** выполняет ввод до первого пробельного символа. Чтобы ввести строку, состоящую из нескольких слов, используйте спецификацию **%c**:

```
scanf("%20c",s);
```

В этом случае количество считываемых символов может быть только константой, и короткие строки придется при вводе дополнять пробелами до требуемой длины строки.

Библиотека языка С содержит функции, специально предназначенные для ввода (**gets**) и вывода (**puts**) строк.

Функция **gets(s)** читает символы с клавиатуры до появления символа '**\n**' и помещает их в строку **s** (сам символ '**\n**' в строку не включается, вместо него вносится '**\0**'). Функция возвращает указатель на **s**, а случае ошибки или конца файла – **NULL**. Однако функция **gets** не контролирует количество введенных символов в **s**, что может приводить к выходу за границы **s.**

Функция **puts(s)** выводит строку **s** на стандартное устройство вывода, заменяя завершающий символ '**\0**' на символ новой строки.

Большинство функций работы со строками описаны в головном файле **string.h.** Некоторые из этих функций указаны в таблице 1.1. Здесь аргументы ***s*** и ***t*** принадлежат типу **char***, ***cs*** и ***ct*** – типу **const char***, ***n*** – типу **size_t**, ***c*** – типу **int**, приведенному к **char**.

Например, чтобы строке **s** присвоить значение строки **t** можно использовать функции **strcpy** или **strncpy**:

```
char t[] = " Язык Си ";
char *s = new char[m];
strcpy(s,t);
strncpy(s,t,strlen(t)+1);
```

Функция **strcpy(s,t)** копирует все символы из строки **t**, включая и нуль-литеру, в строку, на которую указывает **s**. Функция **strncpy(s,t,n)** выполняет то же самое, но не более **n** символов. При этом если нуль-символ встретится раньше, копирование прекращается, и оставшиеся символы строки **s** заполняются нуль-символами. Если **n** меньше или равно длине строки **t**, завершающая нуль-литера не добавляется. Обе функции возвращают указатель на результирующую строку.

Функция **strlen(t)** возвращает фактическую длину строки **t**, не включая нуль-литеру.

Таблица 5.1 — Функции для обработки строк

char* strcpy(s,ct)	копирует стринг ***ct*** в стринг ***s***; возвращает ***s***
char* strcat(s,ct)	соединяет строку ***s*** и ***ct***
char* strcmp(cs,ct)	сравнивает ***cs*** и ***ct***
char* strchr(cs,c)	возвращает указатель на первое ***c*** в ***cs***
char* strrchr(cs,c)	возвращает указатель на последнее ***c*** в ***cs***
size_t strspn(cs,ct)	возвращает число символов в начале ***cs***, ни один из которых не входит в строку ***ct***
size_t strcspn(cs,ct)	возвращает число символов в начале ***cs***, которые принадлежат множеству символов из строки ***ct***
char* strpbrk(cs,ct)	возвращает указатель в ***cs*** на первую литеру, которая совпала с одной из литер, входящих в ***ct***
char* strstr(cs,ct)	возвращает указатель на первое вхождение ***ct*** в ***cs***
size_t strlen(cs)	возвращает длину ***cs***
char* strok(s,ct)	ищет в ***s*** лексему, ограниченную литерами из ***ct***

5.2.2 Функции файлового ввода-вывода

Язык C++ унаследовал от языка C библиотеку стандартных функций ввода-вывода. Функции ввода-вывода объявлены в заголовочном файле **<stdio.h>**. Операции ввода-вывода осуществляются с файлами. Файл может быть *текстовым* или *бинарным* (двоичным). Различие между ними заключается в том, что текстовый файл разбит на строки. Признаком конца строки является пара символов **'\r'**(возврат каретки) и **'\n'**(перевод строки). При вводе или выводе значений из текстового файла они *требуют преобразований* во внутреннюю форму хранения этих значе ний в соответствии с их типами, объявленными в программе. Бинарный файл – это просто последовательность символов. При вводе-выводе информации в бинарные файлы никаких преобразований не выполняется. Поэтому работа с бинарными файлами выполняется быстрее.

В языке C реализован так называемый *потоковый ввод-вывод*. Термин поток происходит из представления процесса ввода-вывода информации в файл в

виде последовательности или потока байтов. Все потоковые функции ввода-вывода обеспечивают буферизированный, *форматированный* или *неформатированный ввод-вывод*.

Перед тем, как выполнять операции ввода или вывода в файловый поток его следует открыть с помощью функции **fopen()**. Эта функция имеет следующий прототип:

FILE *fopen(const char *filename, const char *mode);

Функция **fopen()** открывает файл с именем **filename** и возвращает указатель на файловый поток, используемый в последующих операциях с файлом. Параметр **mode** является строкой, задающей режим, в котором открывается файл. Он может принимать значения, указанные в таблице 2.2

Таблица 5.2 — Режимы открытия файлов

Режим	Описание режима
R	Файл открывается только для чтения
W	Файл создается только для записи. Если файл с этим именем уже существует, он будет перезаписан. Чтение из файла не разрешено.
A	Режим добавления записей (append). Файл открывается только для записи в конец или создается только для записи, если он еще не существует. Чтение не разрешено.
r+	Существующий файл открывается для обновления (считывания и записи)
w+	Создается новый файл для обновления. Перезаписывается любой существующий файл с тем же именем.
a+	Файл открывается для добавления в конец; если файл не существует, он создается, и любой существующий файл с тем же именем перезаписывается.

Чтобы указать, что данный файл открывается или создается как текстовый, добавьте символ **t** в строку режима (например, **“rt”**,”**w+t**” и т.п.). Аналогично можно сообщить, что файл открывается или создается как бинарный. Для этого добавьте в строку режима символ **b** (например, “**wb**”, “**a+b**”).

В случае успеха функция **fopen** возвращает указатель на открытый поток, в случае ошибки – **NULL**. Например:

FILE *fptr = fopen(“mytxt.txt”,”rt”);

Здесь объявляется переменная **fptr** – указатель на файловый поток и выполняется её инициализация с помощью функции **fopen()**.

Для завершения работы с потоком он должен быть закрыт с помощью функции **fclose()**:

fclose(fptr);

Рассмотрим функции, осуществляющие ввод-вывод в файловый поток.
Функция **fgetc()** имеет следующий прототип:

int fgetc(FILE *fptr);

Она осуществляет ввод символа из файлового потока **fptr**. В случае успеха функция возвращает код символа. Если делается попытка прочесть конец файла или произошла ошибка, то возвращается **EOF.**

Функция **fputc()** имеет следующий прототип:

int fputc(int c,FILE *fptr);

Она осуществляет вывод символа в поток.

Функция **fgets()** имеет следующий прототип:

char *fgets(char *s,int n,FILE *fptr);

Она осуществляет чтение строки символов из файлового потока в строку **s.** Функция прекращает чтение, если прочитано **n**-1 символов или встретится символ перехода на новую строку **‘\n’**. Если этот символ встретился, то он сохраняется в переменной **s**. В обоих случаях в переменную **s** добавляется символ **‘\0’**. В случае успеха функция возвращает указатель на считанную строку **s**. Если произошла ошибка или считана метка **EOF**, то она возвращает **NULL**.

Для записи строки в файл можно использовать функцию **fputs**. При этом символ перехода на новую строку не добавляется, и завершающая нуль-литера в файл не копируется. Функция **fputs()** имеет следующий прототип:

int fputs(const *char, FILE *fptr);

В случае успеха функция **fputs()** возвращает неотрицательное значение. В противном случае она возвращает **EOF.**

Форматированный ввод-вывод текстовых файлов организуется в языке С с помощью функций **fscanf()** и **fprintf()**. Эти функции аналогичны функциям **scanf()** и **printf()** соответственно, с той лишь разницей, что их первым аргументом является указатель на файл, открытый в соответствующем режиме.

Функция **feof()** осуществляет проверку достижения метки конца файла. Она имеет прототип:

int feof(FILE *fptr);

Функция возвращает **0**, если конец файла не достигнут.

Рассмотрим пример файлового ввода и вывода с помощью функций **fscanf()** и **fprintf()**:

```
#include <stdio.h>
int main() {
  int i,x;
  FILE *in,*out; // описание указателей на файлы
  if ((in = fopen(“c:\\old.dat”,”rt”))== NULL)
    { fprintf(stderr,” Не могу открыть входной файл \n”);
          return 1;
    }
  if ((out = fopen(“c:\\new.dat”,”wt”))== NULL)
   { fprintf(stderr,”Не могу открыть выходной файл \n”);
          return 1;
   }
  i = 0;
```

```
    while (fscanf(in,"%d",&x)!=EOF)
          { i++;
           fprintf(out,"%d\t%d\n",i,x);
          }
    fclose(in);
    fclose(out);
    return 0;
  }
```

Программа копирует целые числа из входного файла **old.dat** в выходной файл **new.dat**. При этом в выходной файл записываются также порядковые номера чисел. Копирование выполняется до тех пор, пока не будет встречена метка конец файла **EOF**. Если входной или выходной файл нельзя открыть, то программа выводит соответствующее сообщение в стандартный поток **stderr**, который по умолчанию связан с пользовательским терминалом и предназначен для вывода сообщений об ошибках.

Для осуществления *неформатированного ввода-вывода* (без преобразований) применяются функции **fread()** и **fwrite()**. Эти функции имеют следующие прототипы:

size_t fread(void *ptr, size_t size, size_t n, FILE *fptr);
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fptr);

Функция **fread()** считывает блоки данных из файлового потока, на который указывает **fptr**, в буфер, доступ к которому выполняется через указатель **ptr**, а функция **fwrite()** выполняет обратную операцию, то есть переписывает блоки данных из буфера в файловый поток. При этом копируется **n** блоков данных, каждый из которых содержит **size** байтов. В случае успеха функции возвращают число скопированных блоков. В случае ошибки возвращается **0** или число полностью скопированных блоков. Если **n** больше значения, которое вернула функция **fread**, то это говорит о том, что встретилась метка конец файла.

Когда файл открывается для чтения или записи, с ним на самом деле связывается структура **FILE**, определенная в заголовочном файле **<stdio.h>**. Эта структура для каждого открытого файла содержит *счетчик положения текущей записи*. Сразу после открытия файла его значение равно **0**. Каждая операция ввода-вывода вызывает приращение этого счетчика на число записанных или считанных байтов из файла. Функции позиционирования – **fseek()**, **ftell()** и **rewind()** позволяют изменять или получать значение счетчика, связанного с файлом.

Функция

long int ftell(FILE *fptr);

возвращает текущее значение счетчика связанного с файлом. В случае ошибки она возвращает **-1L.**

Функция

int fseek(FILE *fptr, long offset, int from);

изменяет позиционирование файлового потока **fptr** на **offset** байтов относительно позиции, определяемой параметром **from**. Параметр **from** может принимать значения:

SEEK_SET — начало файла, 0;

SEEK_CURR — текущая позиция в файле, 1;

SEEK_END — конец файла, 2.

Функция **fseek()** возвращает **0**, если счетчик текущей записи успешно изменен.

Функция

void rewind(FILE *fptr);

устанавливает счетчик текущей позиции на начало потока.

5.2.3 Файловый ввод-вывод с использованием классов языка C++

Для реализации ввода-вывода с использованием классов языка C++ в программу следует включить заголовочный файл **<fstream.h>**. Для осуществления операций с файлами библиотека ввода-вывода C++ предусматривает три класса:

ifstream – потоковый класс для ввода из файла;

ofstream – потоковый класс для вывода в файл;

fstream – потоковый класс для ввода-вывода в файл.

Для создания потока ввода, открытия файла и связывания его с потоком можно использовать следующую форму конструктора класса **ifstream**:

ifstream fin(“text.txt”, ios::in);

Здесь определяется объект **fin** класса входных потоков **ifstream**. С этим объектом можно работать так же, как со стандартными объектами **cin** и **cout**, то есть использовать операции помещения в поток << и извлечения из потока >>, а также рассмотренные выше функции **get, getline**. Например:

```
const len = 80;
char s[len];
fin.getline(s,len);
```

Предполагается, что файл с именем **text.txt** находится в том же каталоге, что и текст программы, иначе следует указать полный путь, дублируя символ обратной косой черты, так как иначе он будет иметь специальное значение, например:

ifstream fin(“c:\\work\\text.txt”, ios::in);

Второй параметр этого конструктора означает режимы открытия файла. В таблице 1.3 приведены возможные режимы открытия и их значение.

Например, для открытия файла вывода в режиме добавления можно использовать инструкцию:

ofstream fout(“out.txt”,ios::out | ios::app);

Файлы, которые открываются для вывода, создаются, если они не существуют.

Таблица 5.3 — Режимы открытия и их значения

Режим	Назначение
ios::in	Открыть для чтения
ios::out	Открыть для записи
ios::ate	Начало вывода устанавливается в конец файла (допускает изменение позиции вывода)
ios::app	Открыть для добавления в конец (безотносительно к операциям позиционирования)
ios::trunc	Открыть, удалив содержимое файла
ios::binary	Двоичный режим операций

Если открытие файла завершилось неудачей, объект, соответствующий потоку, будет возвращать значение **false**. Например, если предыдущая инструкция завершилась неудачей, то это можно проверить так:

```
if (!fout)
{ cout<<” Файл не открыт \n”;}
```

Если при открытии файла не указан режим **ios::binary**, то файл открывается как текстовый. В этом случае можно использовать при работе с файлом функции ввода-вывода, принятые в языке С, такие, как **fprintf()** и **fscanf()**.

Для проверки достигнут ли конец файла или нет, можно использовать функцию **eof()** класса **ios**.

Завершив операции ввода-вывода, необходимо закрыть файл, вызвав функцию **close()**:

```
fout.close();
```

Закрытие файла происходит автоматически при выходе объекта потока из области видимости.

Произвольный доступ к записям файлов реализуется с помощью функций (методов) **seekg()** и **seekp()**, используемых, соответственно, для позиционирования входного и выходного потоков. Например:

```
fin.seekg(0, ios::end);
```

Функция **seekg(offset, org)** перемещает текущую позицию чтения в файле на **offset** байтов относительно **org.** Параметр **org** может принимать одно из трех значений:

ios::beg — от начала файла;

ios::cur — от текущей позиции;

ios::end — от конца файла.

Получить текущее значение позиции в потоке ввода и вывода можно с помощью функций **tellg()** и **tellp()**, соответственно. Например, вызов **fin.tellg()** вернет значение счетчика текущей позиции.

Для осуществления *неформатированного ввода-вывода* при работе с потоками в языке С++ применяются функции **read()** и **write()**. Эти функции характеризуются прототипами:

istream_type& read(char *s, streamsize n);
ostream_type& write(const char *s, streamsize n);

Здесь параметр **s** указывает на буфер, в который соответственно помещаются или из которого извлекаются символы, а **n** означает число читаемых или записываемых символов. Примеры вызова этих функций приведены ниже в тексте программы.

5.2.4 Пример программы обработки текстового файла

Написать программу, которая считывает текст из файла и выводит в выходной файл только вопросительные предложения из этого текста.

Исходные данные и результаты.

Исходные данные хранятся в текстовом файле неизвестного размера. Предложение может занимать несколько строк текстового файла, поэтому ограничиться буфером на одну строку в данной задаче нельзя. Для этого выделим буфер, в который поместится весь файл.

Результаты являются частью исходных данных, поэтому дополнительно под них пространство выделять не требуется.

Для организации вывода предложений понадобятся переменные целого типа, в которых будем хранить позиции начала и конца предложения.

Алгоритм решения задачи.

1. Открыть файл.
2. Определить его длину.
3. Выделить в динамической памяти соответствующий буфер.
4. Считать файл с диска в буфер.
5. Анализируя буфер посимвольно, выделять предложения. Если предложение оканчивается вопросительным знаком выводить его в файл.

Детализируем последний пункт алгоритма. Для вывода предложения необходимо хранить позиции его начала и конца. Предложение может оканчиваться точкой, восклицательным или вопросительным знаком. В двух первых случаях предложение пропускается. Это выражается в том, что значение позиции начала предложения обновляется. Она будет соответствовать позиции символа, следующего за текущим символом, и просмотр буфера продолжается.

В программе будем использовать функции неформатированного чтения блоков данных из входного файла, так как применение функций посимвольного чтения неэффективно.

Для определения длины файла воспользуемся функциями **seekg()** и **tellg()** класса **ifstream**. Для этого переместим указатель текущей позиции в конец файла с помощью **seekg()**, а затем с помощью **tellg()** получим значение конечной позиции, запомнив его в переменной **len**.

```
#include <fstream>
#include <stdio>
int main(){
//создание  входного потока и открытие файла
ifstream fin("c:\\bc\\work\\text.txt", ios::in);
if (!fin) {
```

```
        cout<<"Can't open input file"<< endl;
         return 1;
    }

    fin.seekg(0,ios::end);              //указатель в конец файла
    long len = fin.tellg();             //запомнить длину файла
    char *buf = new char [len +1];      //выделить память под буфер
    //неформатированное чтение текстового файла
    fin.seekg(0,ios::beg);              //указатель в начало файла
    fin.read(buf,len);                  //скопировать len символов из fin  в буфер
    buf[len] = '\0';                    //поместить в буфер нуль-литеру
    //создание выходного потока и открытие файла
    ofstream fout("c:\\bc\\work\\textout.txt", ios::out);
    if (!fout) {
        cout<<"Can't open output file"<< endl;
         return 1;
    }
    long n=0,                           //индекс символа начала предложения
         i=0,                           //индекс символа  конца предложения
         j=0;                           //текущий индекс символа вопросит. предл.
    while(buf[i]) {                     //просмотр символов в буфере
       if( buf[i] == '?') {             //Если i-ый символ – вопрос,
          for(j=n;j<=i;j++)
                fout<<buf[j];           //то вывод предложения в поток,
          n=i+1;                        //обновление индекса начала предложения.
       }
       if(buf[i]=='.'||buf[i]=='!')     //Если предложение не вопросительное,
          n=i+1;                        //то только обновление индекса n .
       i++;
    }
    fin.close();
    fout.close();
    cout<<endl;
    return 0;
}
```

Вариант решения этой же задачи с использованием функций языка С.

```
#include <stdio.h>
int main(){
    //открытие  входного файла
    FILE *fin;
    fin=fopen("text.txt","r"); // файл хранить в папке проекта

    if (!fin) {
      puts("Can't open input file");
      return 1;
```

```
  }
fseek(fin,0,SEEK_END);              //указатель в конец файла
long len=ftell(fin);                //запомнить длину файла
char *buf=new char[len+1];          //выделить память под буфер
//неформатированное чтение текстового файла (поблочное)
const int l_block=1024;                     //задать длину блока для чтения
int num_block=len/l_block;                  //определить число блоков
rewind(fin);                                //указатель в начало файла
fread(buf,l_block,num_block+1,fin);         //чтение блоков из файла
buf[len]='\0';                              //поместить в буфер нуль-литеру
//создание выходного файла
FILE *fout;
fout = fopen("textout.txt","w");
if (!fout) {
  puts("Can't open output file");
  return 1;
  }
long  n=0,                      //индекс символа начала предложения
      i=0,                      //индекс символа  конца предложения
      j=0;                      //текущий индекс символа вопросит. предл.
while(buf[i]) {                 //просмотр символов в буфере
   if(buf[i]=='?') {            //Если i-ый символ – вопрос,
      for(j=n;j<=i;j++)
      putc(buf[j],fout);        //то вывод предложения в поток,
      n=i+1;                    //обновление индекса начала предложения.
   }
   if (buf[i]=='.'||buf[i]=='!')  //Если предложение не вопросительное,
      n=i+1;                    //то обновление только индекса n .
   i++;
}
fclose(fin);
fclose(fout);
printf("\n");
return 0;
}
```

В заключение отметим, что файл с тестовым примером должен содержать предложения различной длины – от нескольких символов до нескольких строк.

Программа, написанная с использованием функций ввода-вывода языка С, а не классов языка С++ часто является более быстродействующей, но менее защищенной от ошибок ввода данных.

5.3 Варианты заданий

Написать две программы: в первой ввод-вывод осуществлять с помощью средств С, во второй ввод-вывод осуществлять с помощью классов С++.

Вариант 1

Написать программу, которая считывает из текстового файла три предложения и выводит их в обратном порядке.

Вариант 2

Написать программу, которая считывает текст из файла и выводит на экран только предложения, содержащие введенное с клавиатуры слово.

Вариант 3

Написать программу, которая считывает текст из файла и выводит на экран только строки, содержащие двузначные числа.

Вариант 4

Написать программу, которая считывает английский текст из файла и выводит на экран слова, начинающиеся с гласных букв.

Вариант 5

Написать программу, которая считывает текст из файла и выводит его на экран, меняя местами каждые два соседних слова.

Вариант 6

Написать программу, которая считывает текст из файла и выводит на экран только предложения, не содержащие запятых.

Вариант 7

Написать программу, которая считывает текст из файла и определяет, сколько в нем слов, состоящих не более чем из четырех букв.

Вариант 8

Написать программу, которая считывает текст из файла и выводит на экран только цитаты, то есть предложения, заключенные в кавычки.

Вариант 9

Написать программу, которая считывает текст из файла и выводит на экран только предложения, состоящие из заданного количества слов.

Вариант 10

Написать программу, которая считывает английский текст из файла и выводит на экран слова текста, начинающиеся и оканчивающиеся на гласные буквы.

Вариант 11

Написать программу, которая считывает текст из файла и выводит на экран только строки, не содержащие двузначных чисел.

Вариант 12

Написать программу, которая считывает текст из файла и выводит на экран только предложения, начинающиеся с тире, перед которым могут находиться только пробельные символы.

Вариант 13

Написать программу, которая считывает английский текст из файла и выводит его на экран, заменив каждую первую букву слов, начинающихся с гласной буквы, на прописную.

Вариант 14

Написать программу, которая считывает текст из файла и выводит его на экран, заменив цифры от 0 до 9 на слова «ноль», «один»...«девять», начиная каждое предложение с новой строки.

Вариант 15

Написать программу, которая считывает текст из файла, находит самое длинное слово и определяет, сколько раз оно встретилось в тексте.

Вариант 16

Написать программу, которая считывает текст из файла и выводит на экран сначала вопросительные, а затем восклицательные предложения.

Вариант 17

Написать программу, которая считывает текст из файла и выводит его на экран, после каждого предложения добавляя, сколько раз встретилось в нем введенное с клавиатуры слово.

Вариант 18

Написать программу, которая считывает текст из файла и выводит на экран все его предложения в обратном порядке.

Вариант 19

Написать программу, которая считывает текст из файла и выводит на экран сначала предложения, начинающиеся с однобуквенных слов, а затем все остальные.

Вариант 20

Написать программу, которая считывает текст из файла и вычисляет количество открытых закрытых скобок. Дописать вычисленные значения в конец каждой строки. Результаты записать в новый файл.

Вариант 21

Написать программу, которая считывает текст из файла и выводит на экран предложения, содержащие максимальное количество знаков пунктуации.

Вариант 22

Даны два текстовых файла. Удалить из этих файлов строки, которые имеют одинаковые номера, но сами не являются одинаковыми. Результаты записать в новый файл.

Вариант 23

В каждой строке текстового файла найти наиболее длинную последовательность цифр. Значение ее длины преобразовать в строку, которую записать в начале строки файла. Результаты записать в новый файл.

Вариант 24

Даны два текстовых файла. Создать третий файл из символов, которые записаны в позициях с одинаковыми номерами, но различаются между собой.

Вариант 25

Дан текстовый файл с программой. Исключить комментарии в тексте С программы.

Вариант 26

Написать программу, которая подсчитывает количество пустых строк в текстовом файле.

Вариант 27

Написать программу, которая находит максимальную длину строки текстового файла и печатает эту строку.

Вариант 28

Пусть текстовый файл разбит на непустые строки. Написать программу для подсчета числа строк, которые начинаются и оканчиваются одной и той же литерой.

Вариант 29

Пусть текстовый файл разбит на непустые строки. Написать программу для подсчета числа строк, которые состоят из одинаковых литер.

Вариант 30

Написать программу, переписывающую содержимое текстового файла t2 в текстовый файл t1 (с сохранением деления на строки).

5.4 Порядок выполнения работы

5.4.1 В ходе самостоятельной подготовки изучить основы работы со строками и файлами в языке C/C++.

5.4.2 Выбрать способ представления исходных данных задачи и результатов.

5.4.3 Разработать алгоритм решения задачи, разбив его при необходимости на отдельные функции.

5.4.4 Разработать программу на языке C/C++.

5.4.5 Разработать тестовые примеры, следуя указаниям, изложенным в разделе 3.

5.4.6 Выполнить отладку программы.

5.4.7 Получить результаты работы программы и исследовать её свойства для различных режимов работы, сформулировать выводы.

5.5 Содержание отчета

Цель работы, вариант задания, структурная схема алгоритма решения задачи с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, выводы.

5.6 Контрольные вопросы

5.6.1 Приведите примеры описания и инициализации строк в языке C.

5.6.2 Как хранится строка языка C в памяти ЭВМ?

5.6.3 Объясните назначение и запишите прототипы функций **gets, puts**.

5.6.4 Как выделить динамическую память под строку?

5.6.5 Как выполнить ввод и вывод строки с помощью функций **scanf()** и **printf()**?

5.6.6 Как выполнить ввод и вывод строк с помощью объектов **cin** и **cout**?

5.6.7 Как ввести строку с помощью функций **getline** и **get** объекта **cin**?

5.6.8 Для чего предназначены функции **strcpy()** и **strncpy()**? Как их вызвать?

5.6.9 Для чего предназначена функция **strcmp()**? Как её вызвать? Какие результаты она возвращает?

5.6.10 Объясните, как открыть файл? Какие имеются режимы открытия файлов в языке С?

5.6.11 Опишите функции **fgets** и **fputs**. В чем их отличие от функций **gets** и **puts**?

5.6.12 Приведите примеры чтения и записи значений в файл с помощью функций **fscanf()** и **fprintf()**?

5.6.13 Приведите примеры вызова функций **fread()** и **fwrite()**. Объясните назначение аргументов этих функций.

5.6.14 Как выполняется изменение и чтение счетчика текущей позиции файла в языке С?

5.6.15 Как открыть файловый поток в режиме чтения в языке С++?

5.6.16 Как открыть файловый поток в режиме записи в языке С++?

5.6.17 Объяснить назначение функций **seekg()** и **seekp()**. Приведите примеры их вызова.

5.6.18 Приведите примеры вызова функций **read()** и **write()** языка С++.

6 ЛАБОРАТОРНАЯ РАБОТА №6

ПРОГРАММИРОВАНИЕ ОПЕРАЦИЙ НАД СТРУКТУРАМИ И БИНАРНЫМИ ФАЙЛАМИ

6.1 Цель работы

Изучение способов описания структур данных на языке С. Исследование особенностей обработки бинарных файлов, хранящих структурные типы данных.

6.2 Краткие теоретические сведения

6.2.1 Структуры

Структуры языка С, аналогично комбинированному типу языка Паскаль, объединяют в себе компоненты (поля) разных типов. Описание структуры начинается с ключевого слова **struct** и содержит список описаний полей в фигурных скобках. За словом **struct** может следовать имя структуры, которое рассматривается как имя структурного типа:

```
struct <имя структуры>
{  <тип1> <поле 1>;
  <тип2> <поле 2>;
  . . .
  <тип n> <поле n>;
}
```

Например, пусть требуется фиксировать сведения о расписании работы секций конференции. При этом для каждого мероприятия фиксируется время, тема, фамилия организатора и количество участников. Для этого можно описать следующую структуру:

```
struct section
{  int hour, min;
     char theme[100];
     char name[30];
     int num;
  } s1,s[10];
```

Здесь **section —** это *имя типа*. Переменные структурного типа описываются либо одновременно с описанием структуры (см. выше), либо отдельно. Например:

```
section s1,s[10];   // структура и массив структур
```

При совместном описании структурного типа и переменных допускается имя структуры не указывать, если оно нигде не используется.

Переменные структурного типа можно размещать и в динамической области памяти, для этого надо описать указатель на структуру:

```
section *sptr=new section;          //указатель на структуру
section *mptr=new section[m];       //указатель на массив структур
```

Поля структуры могут быть любого основного типа, массивом, указателем, структурой или объединением. Для доступа к полям структуры используется операция выбора, обозначаемая точкой:

<имя структурной переменной>.<имя поля>

Например:

```
s1.hour=9;
s1.min=30;
strcpy(s1.theme,”Архитектура компьютеров”);
s[1].hour=13;
s[1].min=0;
strcpy(s[1].theme,”Программное обеспечение”);
```

Если структуры размещены в динамической памяти, то доступ к полям выполняют через указатели:

```
(*sptr).hour=9;
(*sptr).min=30;
```

Скобки здесь необходимы, поскольку приоритет операции **“.”** выше, чем у операции раскрытия ссылки. Существует еще одна форма доступа к полям структуры при работе с указателями на структуру:

```
sptr->num=30;
```

Если имеется указатель на массив структур, то доступ к ним выполняется так:

```
mptr[2].hour=15; //(*(mptr+2)).hour=15;
```

Структуры одного типа можно присваивать друг другу:

```
*sptr=s1;
mptr[1]=s1;
mptr[2]=s[0];
```

Ввод-вывод структур, как и массивов, выполняется поэлементно. Например, ввод-вывод структуры **s1** с использованием классов языка C++ можно выполнить следующим образом:

```
cin >> s1.hour >> s1.min;
cin.getline(s1.theme,100);
cout << s1.hour << s1.min << ‘ ‘ << s1.theme << endl;
```

Этот же ввод-вывод можно выполнить и с помощью функций ввода-вывода языка C:

```
scanf(“%d%d”,&s1.hour,&s1.min);
gets(s1.theme);
printf(“%d%d%s”,s1.hour,s1.min,s1.theme);
```

Статические структуры можно инициализировать перечислением значений их полей:

```
section s2={10,30,”Архитектура компьютеров”,25};
```

6.2.2 Пример программы обработки структур

Имеются записи о сотрудниках, содержащие фамилию и инициалы, год рождения и оклад сотрудника. Требуется написать программу с использованием самостоятельно определяемых функций, которая

- вводит записи о сотрудниках с клавиатуры в бинарный файл и упорядочивает его по фамилии;
- осуществляет поиск сотрудников в файле по фамилии и вычисляет средний оклад этих сотрудников;
- отображает содержимое бинарного файла.

Исходные данные и результаты.

Исходные данные:

Сведения об одном сотруднике будем представлять в виде структуры. При этом фамилию и инициалы представим строкой из 15 символов, год рождения – целым типом, оклад – вещественным типом. Поскольку количество записей о сотрудниках не оговорено будем хранить все сведения в бинарном файле, а не массиве.

Результаты:

В результате работы программы требуется вывести на экран требуемые элементы файла. Так как результаты представляют собой выборку из исходных данных, дополнительная память для них не отводится. Кроме того, необходимо подсчитать средний оклад для найденных сотрудников. Для этого введем переменную вещественного типа.

Алгоритм решения задачи.

1. Ввести с клавиатуры сведения о сотрудниках в бинарный файл. При этом будем прекращать ввод, если пользователь введет символ ‘*’.
2. Выполнить сортировку файла по Ф.И.О., используя возможность прямого доступа к элементам бинарного файла. В качестве метода сортировки будем использовать метод прямого обмена (пузырьковую сортировку);
3. Обеспечить вывод сведений о сотруднике:
   a) ввести с клавиатуры фамилию;
   b) выполнить поиск сотрудника в файле по фамилии;
   c) увеличить суммарный оклад и счетчик количества сотрудников;
   d) вывести сведения о сотруднике или сообщение о том, что его нет в списке;
   e) вывести средний оклад.
4. Отобразить содержимое бинарного файла.

Каждый из указанных пунктов алгоритма представим в виде функции, которую можно будет вызывать из основной программы. При этом контроль за порядком вызова функций полностью возлагается на пользователя.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <iostream.h>
```

```
//----------------------константы и структуры-----------------------------
const int len_fio=15;                   //длина строки с Ф.И.О.
struct person
{       char fio[len_fio];              // Ф.И.О
        int year;                       //год рождения
        float salary;                   //зарплата
};
const int size_p=sizeof(person);        //размер структуры
//------------------------прототипы функций-------------------------------
int create_file(FILE *fbin);            //запись в файл
int sort_file(FILE *fbin);              //сортировка файла
int print_file(FILE *fbin);             //вывод файла
int search_persons(FILE *fbin);         //поиск сотрудника и вычисления
//-------------------------основная функция-------------------------------
int main()
{       FILE *fbin;
        char c;
        fbin=fopen("person.dat","r+b");   //Открытие существующего файла
                                          //для чтения и записи в конец
        if (!fbin) {
                fbin=fopen("person.dat","w+b");  //Создание нового файла
                                                 //для обновления
                    if(!fbin){
                    puts("Не могу открыть (создать) файл\n");
                    return 1;
                }
 }//вывод меню и запуск соответствующих функций
  while (1)
    {           clrscr();
                puts("1- Запись в файл");
                puts("2- Сортировка файла");
                puts("3- Вывод файла");
                puts("4- Поиск и вычисления”);
                puts("5- Выход");
                puts("_____");
                puts("Введите номер пункта меню\n");
                c=getch();
                switch (c)
              {  case '1':create_file(fbin);break;
                 case '2':sort_file(fbin);break;
                 case '3':print_file(fbin);break;
                 case '4':search_persons(fbin);break;
                 case '5':return 0;
               }
        }
```

```
}
//-------------------------запись в файл----------------------------------
int create_file(FILE *fbin){
        person elem;
        fseek(fbin,0,SEEK_END);     //указатель в конец файла
        puts("Ввод данных о сотрудниках");
        puts("Для выхода введите символ *");
        puts("_____________________________________\n");
        while (1) {
                puts("Введите Ф.И.О. (обязательно с инициалами)");
                getline(elem.fio,len_fio);        //ввод Ф.И.О.
                if (!strcmp(elem.fio,"*"))        //если введена “*”,
                        return 1;                 // то выход
                puts("Введите год рождения");
                scanf("%i",&elem.year);           //ввод года рождения
                puts("Введите зарплату");
                scanf("%f",&elem.salary);         //ввод зарплаты
                fwrite(&elem,size_p,1,fbin);      //запись структуры в файл
        }
}
//-------------------------вывод файла-------------------------------------
int print_file(FILE *fbin){
        person elem;
        int n;
        clrscr();
        rewind(fbin);                             //указатель в начало файла
        puts("Ф.И.О.        Год   Зарплата");
        do {
                n=fread(&elem,size_p,1,fbin);//чтение структуры из файла
                if (n<1) break;                   //если n<1, то конец файла
                printf("%-15s%-6i%-8.2f\n",elem.fio,elem.year,elem.salary);
           } while (1);
        puts("_____________________________");
        puts("Нажмите любую клавишу");
        getch();
        return 0;
}
//-------------------------сортировка записей в файле----------------------
int sort_file(FILE *fbin) {
        long i,j;
        person elem1,elem2;

        puts("Для сортировки нажмите любую клавишу");
        getch();
        fseek(fbin,0,SEEK_END);     //указатель в конец
```

```
        long len=ftell(fbin)/size_p;     //определяем длину файла
        rewind(fbin);                    //указатель в начало
        //пузырьковая сортировка
        for(i=len-1;i>=1;i--)
          for (j=0;j<=i-1;j++) {
              fseek(fbin,j*size_p,SEEK_SET);     //указатель на j-ую запись
              fread(&elem1,size_p,1,fbin);       //читаем запись j в elem1
              fread(&elem2,size_p,1,fbin);       //читаем след. запись в elem2
              if (strcmp(elem1.fio,elem2.fio)>=1) {      //сравниваем Ф.И.О.
                 fseek(fbin,(-2)*size_p,SEEK_CUR);   //указатель на 2 поз. назад
                 //обмен значений
                 fwrite(&elem2,size_p,1,fbin);      //сначала записываем elem2
                 fwrite(&elem1,size_p,1,fbin);      // затем записываем elem1
              }
          }
        puts("Для выхода нажмите любую клавишу");
        getch();
        return 0;
}
// ———поиск сотрудников и вычисление средней зарплаты———
int search_persons(FILE *fbin){
        int not_found;                   //флаг  поиска
        char s[len_fio];                 //строка для ввода фамилии
        int n_persons=0;                 //счетчик сотрудников
        int n;
        float summa_salary=0;            //сумма зарплат
        person elem;
        while (1) {
              puts("Введите фамилию или  * ");
              cin.getline(s,len_fio);            //запоминаем фамилию в строке s
              if (!strcmp(s,"*"))  break;        //выход, если ввели *
              rewind(fbin);                      //указатель в начало файла
              not_found=1;                       //флаг – сотрудник не найден
              do {
                n=fread(&elem,size_p,1,fbin);    //читаем запись
                if (n<1) break;                  // если n<1, то конец файла
                if (strstr(elem.fio,s))          //ищем строку s в поле fio
                  if (elem.fio[strlen(s)]==' ') {//есть пробел после фамилии ?
                    strcpy(s,elem.fio);          //копируем fio в s
                    puts("_____________________________");
                    puts("Ф.И.О.        Год   Зарплата");
                    printf("%-15s%-6i%-8.2f\n",elem.fio,elem.year,elem.salary);
                    puts("_____________________________");
                    n_persons+=1;                //счетчик сотрудников
                    summa_salary+=elem.salary;   //суммирование зарплат
```

```
                not_found=0;                    //сотрудник найден
              }
        } while (1);
        if (not_found)
        puts("Такого сотрудника нет в файле");
    }
    if (n_persons>0)                    //вычисление средней зарплаты
        printf("Средняя зарплата=%8.2f\n",summa_salary/n_persons);
    puts("_____");
    puts("Нажмите любую клавишу");
    getch();
    return 0;
}
```

Программа достаточно подробно прокомментирована. В тексте основной программы осуществляется либо открытие файла (режим “**r+b**”), либо создание файла (режим “**w+b**”) для обновления, если файл “**person.dat**” на диске не обнаружен. Это позволяет один раз ввести исходные данные в файл, а затем его многократно использовать. Если потребуется начать работу с новым файлом, то старый файл следует удалить с диска средствами операционной системы.

Все необходимые действия с файлом выполняются с помощью функций, вызываемых из пунктов меню, отображаемых на экране. При заполнении файла требуется обязательно вводить инициалы, которые должны отделяться от фамилии пробелом. Фамилия должна начинаться с первой позиции каждой строки. Это свойство используется в процедуре поиска сотрудников.

6.3 Варианты заданий

Вариант 1

Описать структуру с именем STUDENT, содержащую следующие поля:

- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по возрастанию номера группы;
- чтение данных из этого файла, корректировку данных в файле по номеру записи;
- вывод на дисплей фамилий и номеров групп для всех студентов, если средний балл студента больше 4.0;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 2

Описать структуру с именем STUDENT, содержащую следующие поля:

- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по возрастанию среднего балла;
- чтение данных из этого файла;
- вывод на дисплей фамилий и номеров групп для всех студентов, имеющих оценки 4 и 5;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 3

Описать структуру с именем STUDENT, содержащую следующие поля:

- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку 2;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 4

Описать структуру с именем AEROFLOT, содержащую следующие поля:

- название пункта назначения рейса;
- номер рейса;
- тип самолета.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа AEROFLOT; записи должны быть упорядочены по возрастанию номера рейса;
- чтение данных из этого файла;
- вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры;
- если таких рейсов нет, выдать на дисплей соответствующее сообщение.

Вариант 5

Описать структуру с именем AEROFLOT, содержащую следующие поля:

- название пункта назначения рейса;
- номер рейса;
- тип самолета.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа AEROFLOT; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;

- чтение данных из этого файла;
- вывод на экран пунктов назначения и номеров рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры;
- если таких рейсов нет, выдать на дисплей соответствующее сообщение.

Вариант 6

Описать структуру с именем WORKER, содержащую следующие поля:
- фамилия и инициалы работника;
- название занимаемой должности;
- год поступления на работу.

Написать программу, выполняющую следующие действия с помощью функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа WORKER; записи должны быть размещены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры;
- если таких работников нет, вывести на дисплей соответствующее сообщение.

Вариант 7

Описать структуру с именем TRAIN, содержащую следующие поля:
- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия с помощью функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;
- чтение данных из этого файла;
- вывод на экран информации о поездах, отправляющихся после введенного с клавиатуры времени;
- если таких поездов нет, выдать на дисплей соответствующее сообщение.

Вариант 8

Описать структуру с именем TRAIN, содержащую следующие поля:
- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия с помощью функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN; записи должны быть упорядочены по времени отправления поезда;
- чтение данных из этого файла;
- вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры;

- если таких поездов нет, выдать на дисплей соответствующее сообщение.

Вариант 9

Описать структуру с именем TRAIN, содержащую следующие поля:

- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN; записи должны быть упорядочены по номерам поездов;
- чтение данных из этого файла;
- вывод на экран информации о поезде, номер которого введен с клавиатуры;
- если таких поездов нет, выдать на дисплей соответствующее сообщение.

Вариант 10

Описать структуру с именем MARSH, содержащую следующие поля:

- название начального пункта маршрута;
- название конечного пункта маршрута;
- номер маршрута.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа MARSH; записи должны быть упорядочены по номерам маршрутов;
- чтение данных из этого файла;
- вывод на экран информации о маршруте, номер которого введен с клавиатуры;
- если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

Вариант 11

Описать структуру с именем MARSH, содержащую следующие поля:

- название начального пункта маршрута;
- название конечного пункта маршрута;
- номер маршрута.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа MARSH; записи должны быть упорядочены по номерам маршрутов;
- чтение данных из этого файла;
- вывод на экран информации о маршрутах, которые начинаются или оканчиваются в пункте, название которого введено с клавиатуры;
- если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

Вариант 12

Описать структуру с именем NOTE, содержащую следующие поля:

- фамилия, имя;

- номер телефона;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа NOTE; записи должны быть упорядочены по датам рождения;
- чтение данных из этого файла;
- вывод на экран информации о человеке, номер телефона которого введен с клавиатуры;
- если такого нет, выдать на дисплей соответствующее сообщение.

Вариант 13

Описать структуру с именем NOTE, содержащую следующие поля:

- фамилия, имя;
- номер телефона;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа NOTE; записи должны быть размещены по алфавиту;
- чтение данных из этого файла;
- вывод на экран информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры;
- если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 14

Описать структуру с именем NOTE, содержащую следующие поля:

- фамилия, имя;
- номер телефона;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа NOTE; записи должны быть упорядочены по трем первым цифрам номера телефона;
- чтение данных из этого файла;
- вывод на экран информации о человеке, чья фамилия введена с клавиатуры;
- если такого нет, выдать на дисплей соответствующее сообщение.

Вариант 15

Описать структуру с именем ZNAK, содержащую следующие поля:

- фамилия, имя;
- знак Зодиака;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK; записи должны быть упорядочены по датам рождения;

- чтение данных из этого файла;
- вывод на экран информации о человеке, чья фамилия введена с клавиатуры;
- если такого нет, выдать на дисплей соответствующее сообщение.

Вариант 16

Описать структуру с именем ZNAK, содержащую следующие поля:
- фамилия, имя;
- знак Зодиака;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK; записи должны быть упорядочены по датам рождения;
- чтение данных из этого файла;
- вывод на экран информации о людях, родившихся под знаком, название которого введено с клавиатуры;
- если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 17

Описать структуру с именем ZNAK, содержащую следующие поля:
- фамилия, имя;
- знак Зодиака;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK; записи должны быть упорядочены по знакам Зодиака;
- чтение данных из этого файла;
- вывод на экран информации о людях, родившихся в месяц, значение которого введено с клавиатуры;
- если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 18

Описать структуру с именем PRICE, содержащую следующие поля:
- название товара;
- название магазина, в котором продается товар;
- стоимость товара в грн.

Написать программу, выполняющую следующие действия с помощью функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа PRICE; записи должны быть размещены в алфавитном порядке по названиям товаров;
- чтение данных из этого файла;
- вывод на экран информации о товаре, название которого введено с клавиатуры;
- если таких товаров нет, выдать на дисплей соответствующее сообщение

Вариант 19

Описать структуру с именем PRICE, содержащую следующие поля:

- название товара;
- название магазина, в котором продается товар;
- стоимость товара в грн.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа PRICE; записи должны быть размещены в алфавитном порядке по названиям магазинов;
- чтение данных из этого файла;
- вывод на экран информации о товарах, продающихся в магазине, название которого введено с клавиатуры;
- если такого магазина нет, выдать на дисплей соответствующее сообщение.

Вариант 20

Описать структуру с именем ORDER, содержащую следующие поля:

- расчетный счет плательщика;
- расчетный счет получателя;
- перечисляемая сумма в грн.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа ORDER; записи должны быть размещены в алфавитном порядке по расчетным счетам плательщиков;
- чтение данных из этого файла;
- вывод на экран информации о сумме, снятой с расчетного счета плательщика, введенного с клавиатуры;
- если такого расчетного счета нет, выдать на дисплей соответствующее сообщение.

Вариант 21

Описать структуру с именем STUDENT, содержащую следующие поля:

- номер;
- фамилия и имя;
- год рождения;
- год поступления в университет;
- структура OCENKI, содержащая четыре поля: физика, математика, программирование, история;

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов отличников;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 22

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены номеру;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших одну оценку 3;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 23

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по фамилиям;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших все двойки;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 24

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по номеру;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших все пятерки;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 25

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших одну оценку 4, а все остальные – 5;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 26

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, которые начинаются с литеры ‘А’, и их оценки;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 27

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по номеру;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, которые начинаются с литеры ‘Б’, и год их рождения;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 28

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по фамилиям;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, которые начинаются с литеры ‘Б’или ‘Г’, и год их поступления;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 29

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по фамилиям;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, имеющих средний балл выше среднего балла группы;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 30

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по номеру;
- чтение данных из этого файла;
- вывод на дисплей фамилий и года рождения студентов, не получивших ни одной тройки;
- если таких студентов нет, вывести соответствующее сообщение.

6.4 Порядок выполнения работы

6.4.1 В ходе самостоятельной подготовки изучить основы работы со структурами и бинарными файлами в языках C/C++.

6.4.2 Выбрать способ представления исходных данных задачи и результатов Разработать алгоритм решения задачи, разбив его на отдельные функции.

6.4.3 Разработать программу на языке C/C++.

6.4.4 Разработать тестовые примеры, следуя указаниям раздела 3.

6.4.5 Выполнить отладку программы.

6.4.6 Получить результаты работы программы и исследовать её свойства в различных режимах работы, сформулировать выводы.

6.5 Содержание отчета

Цель работы, вариант задания, структурная схема алгоритма решения задачи с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, выводы.

6.6 Контрольные вопросы

6.6.1 Что называют структурой в языке C/C++?

6.6.2 Приведите пример описания структурного типа.

6.6.3 Как описать структурный тип совместно и раздельно с соответствующей переменой?

6.6.4 Как описать массив структур? Для чего его применяют?

6.6.5 Как выполняется доступ к полям статических структурных переменных?

6.6.6 Как выполняется доступ к полям динамических структурных переменных?

6.6.7 Как выделяют динамическую память под структуры?

6.6.8 Покажите на примерах ввод-вывод структур с помощью функций языка C.

6.6.9 Покажите на примерах ввод-вывод структур с помощью классов языка C++.

6.6.10 Как выполняется инициализация структур?

6.6.11 Как выполнить чтение и запись структурных переменных в бинарные файлы?

6.6.12 Как обнаружить конец файла при использовании функции **fread**?

6.6.13. Как определить длину файла с помощью функций **seek** и **ftell**?

7 ЛАБОРАТОРНАЯ РАБОТА №7
ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ СПИСКОВ НА ЯЗЫКЕ С/С++

7.1 Цель работы

Изучение списковых структур данных и приобретение навыков разработки и отладки программ, использующих динамическую память. Исследование особенностей организации списков средствами языка С/С++.

7.2 Краткие теоретические сведения

7.2.1 Динамические структуры данных

В предыдущих лабораторных работах были рассмотрены задачи, в которых объем памяти, необходимый программе, был известен либо до компиляции программы, либо на этапе ввода данных. В первом случае память резервировалась с помощью операторов описания, во втором случае — с помощью функций выделения памяти. В каждом из этих случаев выделялся непрерывный участок памяти.

Если до начала работы программы невозможно определить, сколько памяти потребуется для хранения данных, то память выделяется по мере необходимости отдельными блоками, которые связывают друг с другом при помощи указателей. Такой способ выделения памяти предполагает организацию в памяти ЭВМ *динамических структур данных*, поскольку их размер изменяется во время выполнения программы. Из динамических структур в программах чаще всего используются различные линейные списки, стеки, очереди, деревья. Они различаются способами связи отдельных элементов и допустимыми операциями над ними. Динамическая структура может занимать несмежные участки оперативной памяти. В процессе работы программы элементы структуры могут по мере необходимости добавляться и удаляться.

Напомним, что *линейным списком* называется структура данных, при которой логический порядок следования элементов задается путем ссылок, т.е. каждый элемент списка содержит указатель на следующий элемент (предыдущий элемент). Доступ к первому элементу списка выполняется с помощью специального указателя – указателя на начало (голову) списка.

Односвязным линейным списком называют список, в котором предыдущий элемент ссылается на следующий. *Двусвязный линейный список* – это список, в котором предыдущий элемент ссылается на следующий, а следующий – на предыдущий. *Односвязный циклический список* – это односвязный линейный список, в котором последний элемент ссылается на первый. *Стек* – это односвязный список, в котором компоненты добавляются и удаляются только со стороны вершины списка. *Очередь* – это односвязный список, в котором компоненты добавляются в конец списка, а удаляются со стороны вершины списка.

Элемент любой динамической структуры данных состоит из полей, часть из которых предназначена для связи с соседними элементами. Например, элемент линейного списка, предназначенного для хранения целых чисел, можно описать следующим образом:

```
struct Element {
int d;
struct element *next;
}
```

Здесь поле **next** является указателем на структуру своего собственного типа и предназначено для связи элементов друг с другом.

Определив необходимые указатели, напишем фрагмент программы, выполняющей организацию списка:

```
element *beg=NULL;   //указатель на начало списка
element *temp;
int x;
while (1)
 { scanf("%d",&x);            //ввод x
   if (x==9999) break;
   temp=new element;   //выделение памяти под элемент
   temp->d=x;          //присваивание значений  полю d
   temp->next=beg;     //установление связи с предыдущим элементом
  beg=temp;            //beg указывает на последний введенный эл-т
}
```

Здесь в цикле с клавиатуры вводятся числа. Если введено значение 9999, то происходит выход из цикла. Затем с помощью оператора **new** динамически выделяется память под один элемент списка, и адрес этой области памяти запоминается в указателе **temp**. В поле **d** вновь созданного элемента копируется значение **x**, а в поле **next** – значение указателя **beg**. После этого указателю **beg** присваивается значение **temp**. В итоге в памяти создается список, на начало которого указывает **beg** (рисунок 7.1).

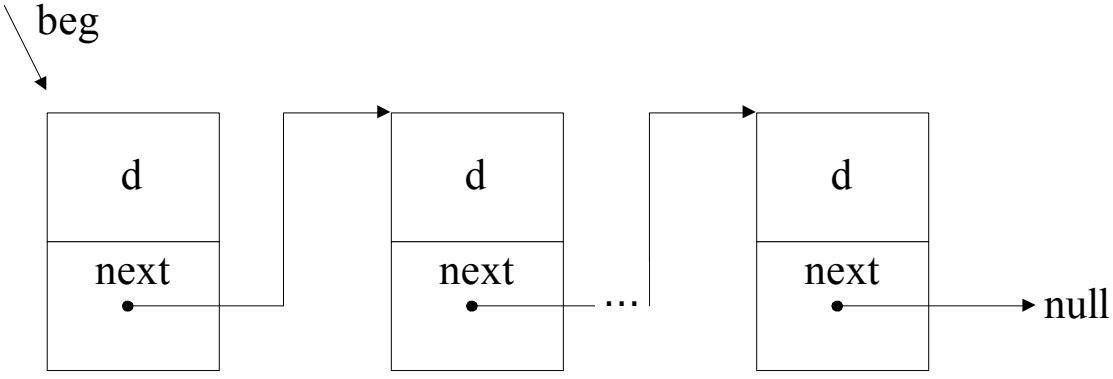

Рисунок 7.1 — Линейный список

Над списками выполняют следующие операции:

- добавление элемента в список;
- исключение элемента из списка;
- просмотр значений элементов списка;

- поиск заданного значения в списке и др.

Например, добавление элемента в указанный список можно выполнить так, как показано в приведенном выше фрагменте программы.

Исключение элемента, на который указывает указатель **beg,** выполняется так:

```
temp=beg;          //запоминание указателя
beg=beg->next;     //перемещение beg на следующий элемент
delete temp;             //освобождение памяти
```

Просмотр значений элементов списка можно выполнить следующим образом:

```
temp=beg;                    //копирование указателя beg
while (temp) {               // пока temp не равен NULL
printf("%d\n",temp->d);      //вывод значения поля d по указателю temp
temp=temp->next;             //перемещение temp на следующий элемент
}
```

7.2.2 Пример программы обработки очереди

Рассмотрим программу, выполняющую организацию очереди, добавление элемента в очередь, удаление элемента из очереди, просмотр очереди.

Очередь – это линейный список, добавление элемента в который выполняется с одной стороны, а исключение – с другой. При этом используются специальные указатели начала и конца очереди. В примере элементами очереди являются структуры данных о заявках на ремонт автомобилей. В заявке указывается фамилия владельца, марка автомобиля, вид работы, дата приема заказа и стоимость работы. После выполнения работы распечатывается квитанция.

Интерфейс программы организуем в виде меню:

1) добавление заявки в очередь;
2) печать и удаление заявки из очереди;
3) просмотр списка заявок;
4) сохранение списка заявок в файле;
5) выход.

Для задания длины строк определим символические константы, чтобы при необходимости можно было легко их изменять.

Для работы с очередью введем два указателя: **beg** – указатель начала очереди и **end** – указатель конца очереди.

Работу с очередью выполняют три функции. Функция **dob_first** формирует первый элемент очереди и возвращает указатель на него. Функция **dob** выполняет добавление элемента в конец очереди, поэтому ей передается указатель на конец очереди и элемент, который следует добавить, а возвращает она измененный указатель на конец очереди.

Удаление элементов выполняется с головы (начала) очереди функцией **udal**. Для этого функции передается указатель **beg**. После удаления элемента функция возвращает измененное значение **beg**. В ходе удаления элемента вызывается функция **print**, осуществляющая вывод на экран сведений о выполненном за-

казе. При этом продемонстрирована работа с датами с помощью функций, содержащихся в головном файле **<time.h>**.

В функциях **dob_first**, **dob** и **print** параметр-структура передается как константная ссылка. Это исключает лишнее копирование структур и их возможную модификацию в теле функций.

```
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <iomanip.h>
#include <conio.h>
#include <iostream>
using namespace std;
//--------------------------константы-----------------------------------------
const int d_n=20,                   //длина Ф.И.О.
          d_m=20,                   //длина названия модели автомобиля
          d_w=80;                   //длина строки описания работ

//----------------------структура элемента-------------------------------------
struct zakaz
{       char name[d_n],             //Ф.И.О.
             model[d_m],            //название модели автомобиля
             work[d_w];             //описание работы
        time_t time;                //время приема заказа
        float price;                //стоимость
        zakaz* next;                //указатель на следующий элемент
};
//-----------------------прототипы функций---------------------------------------
zakaz* dob(zakaz* end, const zakaz& z); //добавление элемента в очередь
zakaz* dob_first(const zakaz& z);       //добавление первого элемента
zakaz* udal(zakaz* beg);                //удаление элемента из очереди
zakaz vvod_zakaza();                    //ввод значений элемента
int  menu();                            //отображение меню и ввод его пункта
void print(const zakaz& z);             //вывод значений  элемента
void prosmotr(zakaz* beg);              //просмотр очереди
int  read_file(char* filename, zakaz** beg, zakaz** end);   //чтение файла
int  write_file(char* filename, zakaz* temp);               //запись в файл
//-----------------------основная функция-----------------------------
int main()
{    zakaz *beg=0,                          //указатель начала очереди
           *end=0;                          //указатель конца очереди
     char *filename="zakaz.txt";            //имя файла
     time_t now;                            // текущее время
     tm t;                                  // дата
     read_file(filename,&beg, &end);        //чтение данных из файла в очередь
```

```
    while (1)
{       switch (menu())//отобразить меню и сделать выбор
   {    case 1:                          //пункт меню 1 – добавление элемента
          if (beg)                       //если очередь не пустая,
               end=dob(end,vvod_zakaza());//то добавляем элемент в конец,
          else {                         //иначе
               beg=dob_first(vvod_zakaza());//создаем первый элемент очереди.
               end=beg;                   //указатели начала и конца равны
               }
          break;
        case 2:                          //пункт меню 2 –
          beg=udal(beg);                 //печать и удаление элемента
          now = time(0);                 //получить текущее время в сек.
          t = *(localtime(&now));        //преобразовать  в дату
          cout<<"Дата выполнения заказа: " //вывод даты
               <<t.tm_mday<<'.'<<t.tm_mon+1<<'.'<<1900+t.tm_year<<endl;
          cout<<"Нажмите любую клавишу"<<endl;
          cin.get();
          break;
        case 3:                          // пункт меню 3 –
          prosmotr(beg);                 //просмотр очереди
          break;
        case 4:                          //пункт меню 4 –
          write_file(filename,beg);      //запись в файл
          break;
        case 5:                          //пункт 5 –выход
          return 0;
        default:                         //если неверно введен номер пункта
          cout<<"Вам следует ввести номер от 1 до 5"<< endl;
          cin.get();
          break;
        }
    }
}
//----------------------добавление элемента----------------------------
zakaz* dob(zakaz *end,const zakaz& z)    //возвращает указатель на доб. эл-т
{       zakaz *newE=new zakaz;           //выделение памяти под элемент
        *newE=z;                         //присвоить значения элементу
        newE->next=0;                    //ссылка на следующий эл-т нулевая
        end->next=newE;                  //добавить эл-т в очередь
        end=newE;                        //установить указатель end на newE
        return end;                      //вернуть измененное значение end
}
//----------------------создание первого элемента-----------------------
zakaz* dob_first(const zakaz& z)
```

```
{       zakaz *beg=new zakaz;           //выделение памяти под элемент
        *beg=z;                         //присвоить значения элементу
        beg->next=0;                    //ссылка на следующий эл-т нулевая
        return beg;                     //вернуть указатель на элемент
}
// ---------- печать и удаление элемента-------------------------
zakaz* udal(zakaz *beg)
{       zakaz *temp;
        if (!beg) { cout<<"Очередь пустая"<<endl; return 0; }
        cout<<"============================="<<endl;
        print(*beg);                    //печать значений головного эл-та
        cout<<"============================="<<endl;
        temp=beg;                       //запомнить указатель на голову
        beg=beg->next;                  //перевод beg на следующий эл-т
        delete temp;                    //удаление элемента
        return beg;                     //вернуть измененное значение beg
}
//------------------ввод значений элемента----------------------------
zakaz vvod_zakaza()
{   char buf[10];
        zakaz z;
        cout<<"Введите Ф.И.О."<<endl;
        cin.getline(z.name,d_n);
        cout<<"Введите модель автомобиля"<<endl;
        cin.getline(z.model,d_m);
        cout<<"Введите описание работы"<<endl;
        cin.getline(z.work,d_w);
        z.time=time(0);                 //зафиксировать время в поле time
        do                              //проверка ввода числа
         {  cout<<"Введите стоимость работы"<<endl;
           cin>>buf;
        } while (!(z.price=(float)atof(buf)));
        return z;
}
//----------------отображение меню и  ввод пункта меню----------------------
int menu() {
        char buf[10];
        int item;
        do
         {clrscr();
           cout<<endl;
           cout<<"===СИСТЕМА УЧЕТА ЗАКАЗОВ==="<<endl<<endl;
           cout<<"1- Добавление элемента в очередь"<<endl;
           cout<<"2- Печать и удаление элемента"<<endl;
           cout<<"3- Просмотр очереди"<<endl;
```

```
        cout<<"4- Запись данных в файл"<<endl;
        cout<<"5- Выход"<<endl;
        cout<<"============================"<<endl;
        cout<<"Введите номер пункта меню"<<endl;
        cin>>buf;                       //прочитать введенное значение
        cin.get();
        item=atoi(buf);                 //преобразовать его в целое
        if (!item) {                    //анализ ошибки ввода
                 cout<<"Вам следует вводить число от 1 до 5"<<endl;
                cin.get();
                }
    } while (!item);            //повторять пока не введет число
    return item;                //вернуть номер введенного пункта меню
}
// ----------------------------печать заказа----------------------------
void print(const zakaz& z)
{   tm t=*(localtime(&z.time));
    cout<<"Ф.И.О.: " <<z.name<<endl;
    cout<<"Модель автомобиля: "<<z.model<<endl;
    cout<<"Описание работ: "<<z.work<<endl;
    cout<<"Дата  приема: "
       <<t.tm_mday<<'.'<<t.tm_mon+1<<'.'<<1900+t.tm_year<<endl;
    cout<<"Стоимость: "<<z.price<<endl;
}
//----------------------------просмотр очереди--------------------------
void prosmotr(zakaz *beg)
{   if (!beg) { cout<<"Очередь пустая"<<endl; return;  }
    zakaz *temp=beg;            //указатель temp устанавливаем в начало
    cout<<"============================="<<endl;
    while (temp) {              //просматриваем пока temp!=0
        print(*temp);           //печатаем значения элемента по указателю
        cout<<"============================="<<endl;
        cout<<"Нажмите любую клавишу"<<endl;
        cin.get();
        temp=temp->next;        //перемещаем temp на следующий эл-т
    }
}
// ----------------------------чтение из файла----------------------------
int read_file(char* filename,zakaz** beg, zakaz** end)
{   ifstream fin(filename,ios::in | ios::nocreate);     //открытие файла
    if (!fin) {cout<<"Нет файла"                        //выход если нет файла
                   <<filename<<endl; return 1;}
    zakaz z;
    *beg = 0;
    while (fin.getline(z.name,d_n))     //чтение фио пока не конец файла
```

```
        {   fin.getline(z.model,d_m);          //чтение марки  автомобиля
            fin.getline(z.work,d_w);           //чтение описания работы
            fin>>z.time>>z.price;              //чтение времени и стоимости
            fin.get();
            if (*beg)                          //если очередь не пустая,
                 *end=dob(*end,z);             //то добавляем элемент в конец,
            else                               //иначе
                {*beg=dob_first(z); *end=*beg;} //создаем первый элемент
        }
        return 0;
}
// ---------------------------запись в файл ---------------------------
int write_file(char* filename, zakaz* temp)
{       ofstream fout(filename);               //открытие файла
        if (!fout) {cout<<"Не могу открыть файл для записи"<<endl; return 1;}
        while (temp)                   //пока temp!=0 выводим эл-ты в файл
          {  fout<<temp->name<<endl;   //вывод фио
             fout<<temp->model<<endl;  //вывод марки автомобиля
             fout<<temp->work<<endl;   //вывод описания работ
             fout<<temp->time<<' '<<temp->price<<endl;      //время и стоимость
             temp=temp->next;          //переместить указатель на след. эл-т
        }
        cout<<"Данные сохранены в файле: "<<filename<<endl;
        cout<<"============================="<<endl;
        cout<<"Нажмите любую клавишу"<<endl;
        cin.get();
        return 0; }
```

Текущую дату и время можно получить с помощью функции **time**. Она возвращает ее в виде секунд, прошедших с полуночи 1 января 1970г. Функция **localtime** преобразует эту величину в стандартную структуру **tm**, определенную в том же головном файле, и возвращает указатель на эту структуру. Поля структуры содержат: год (количество лет, прошедших с 1900г.), месяц (от 0 до 11), день (от 1 до 31), час (0-23),минуту (0-59) и секунду (0-59).

Ввод-вывод организован с помощью классов C++. Обратите внимание на использование в ряде функций метода **cin.get()**. Он необходим для считывания из входного потока признака конца строки.

Функция **read_file** обеспечивает чтение элементов очереди из файла и её организацию в памяти ЭВМ. В качестве входного параметра ей передается указатель на имя файла, а в качестве результата она возвращает через список своих параметров значения указателей **beg** и **end**. Поэтому эти два параметра передаются в функцию по адресу, т.е. как указатель на указатель. Альтернативный способ передачи указателя по адресу – с использованием ссылки на указатель приведен в примере к следующей лабораторной работе.

В программе предусмотрена защита от неправильного ввода данных. Если пользователь введет неверный пункт меню или нечисловые символы там, где это

делать нельзя, программа корректно обработает эту ситуацию (см. функции **menu** и **vvod_zakaza**).

7.3 Варианты заданий

Представить одну из приведенных ниже таблиц в виде линейного списка L, элементами которого являются строки таблицы. Написать функции организации, добавления элемента в список, исключения элемента из списка, просмотра списка, а также одну из функций в соответствии с вариантом, приведенным ниже.

Значения и количество записей в таблице студент выбирает самостоятельно. Исходные данные после организации списка должны сохраняться в файле и при повторном запуске программы считываться из файла. Количество строк таблицы не задается.

Таблица 7.1 — Ведомость

N	Фамилия	Имя	Отчество	Оценки		
				Математика	История	Физика

Таблица 7.2 — Расписание

N Поезда	Станция отправления	Станция назначения	Время отправления	Время прибытия	Стоимость билета

Таблица 7.3 — Анкета

N	ФИО	Год рождения	Пол	Семейное состояние	Количество детей	Оклад

Вариант 1

Таблица 7.1 Функцию, которая вставляет в начало очереди новый элемент.

Вариант 2

Таблица 7.2 Функцию, которая вставляет в начало очереди новый элемент.

Вариант 3

Таблица 7.3 Функцию, которая вставляет в начало очереди новый элемент.

Вариант 4

Таблица 7.1 Функцию, которая вставляет в конец стека новый элемент.

Вариант 5

Таблица 7.2 Функцию, которая вставляет в конец стека новый элемент.

Вариант 6

Таблица 7.3 Функцию, которая вставляет в конец стека новый элемент.

Вариант 7

Таблица 7.1 Функцию, которая вставляет новый элемент E после первого элемента непустого списка L.

Вариант 8

Таблица 7.2 Функцию, которая вставляет новый элемент E после первого элемента непустого списка L.

Вариант 9

Таблица 7.3 Функцию, которая вставляет новый элемент E после первого элемента непустого списка L.

Вариант 10

Таблица 7.1 Функцию, которая вставляет в список L новый элемент E1 за каждым вхождением элемента E.

Вариант 11

Таблица 7.2 Функцию, которая вставляет в список L новый элемент E1 за каждым вхождением элемента E.

Вариант 12

Таблица 7.3 Функцию, которая вставляет в список L новый элемент E1 за каждым вхождением элемента E.

Вариант 13

Таблица 7.1 Функцию, которая вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом.

Вариант 14

Таблица 7.2 Функцию, которая вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом.

Вариант 15

Таблица 7.3 Функцию, которая вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом.

Вариант 16

Таблица 7.1 Функцию, которая вставляет в непустой список L, элементы которого упорядочены по возрастанию значений одного из полей таблицы, новый элемент E так, чтобы сохранилась упорядоченность.

Вариант 17

Таблица 7.2 Функцию, которая вставляет в непустой список L, элементы которого упорядочены по возрастанию значений одного из полей таблицы, новый элемент E так, чтобы сохранилась упорядоченность.

Вариант 18

Таблица 7.3 Функцию, которая вставляет в непустой список L, элементы которого упорядочены по возрастанию значений одного из полей таблицы, новый элемент E так, чтобы сохранилась упорядоченность

Вариант 19

Таблица 7.1 Функцию, которая удаляет из непустого списка L первый элемент.

Вариант 20

Таблица 7.2 Функцию, которая удаляет из непустого списка L первый элемент.

Вариант 21

Таблица 7.3 Функцию, которая удаляет из непустого списка L первый элемент.

Вариант 22

Таблица 7.1 Функцию, которая удаляет из непустого списка L второй элемент, если такой есть.

Вариант 23

Таблица 7.2 Функцию, которая удаляет из непустого списка L второй элемент, если такой есть.

Вариант 24

Таблица 7.3 Функцию, которая удаляет из непустого списка L второй элемент, если такой есть.

Вариант 25

Таблица 7.1 Функцию, которая удаляет из непустого списка L за каждым вхождением элемента E один элемент, если такой есть и он отличен от E.

Вариант 26

Таблица 7.2 Функцию, которая удаляет из непустого списка L за каждым вхождением элемента E один элемент, если такой есть и он отличен от E.

Вариант 27

Таблица 7.3 Функцию, которая удаляет из непустого списка L за каждым вхождением элемента E один элемент, если такой есть и он отличен от E.

Вариант 28

Таблица 7.1 Функцию, которая проверяет, есть ли в списке L хотя бы два элемента с равными значениями второго поля.

Вариант 29

Таблица 7.2 Функцию, которая проверяет, есть ли в списке L хотя бы два элемента с равными значениями второго поля.

Вариант 30

Таблица 7.3 Функцию, которая проверяет, есть ли в списке L хотя бы два элемента с равными значениями второго поля.

7.4 Порядок выполнения работы

7.4.1 В ходе самостоятельной подготовки изучить основы работы с динамическими списковыми структурами в языке C/C++.

7.4.2 Выбрать вид линейного списка, подходящий для решения задачи.

7.4.3 Разработать алгоритм решения задачи, разбив его на отдельные процедуры и функции, так, как указано в варианте задания.

7.4.4 Разработать программу на языке C/C++.

7.4.5 Разработать тестовые примеры, которые предусматривают проверку корректности работы программы.

7.4.6 Выполнить отладку программы.

7.4.7 Получить результаты работы программы и исследовать её свойства в различных режимах работы, и сформулировать выводы.

7.5 Содержание отчета

Цель работы, вариант задания, структурные схемы алгоритмов с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, выводы.

7.6 Контрольные вопросы

7.6.1 Какие операции определены над указателями в языке C/C++?

7.6.2 Что называется списковой структурой данных?

7.6.3 Определите различные виды линейных односвязных списков.

7.6.4 Как описать элемент списковой структуры на языке C/C++?

7.6.5 Напишите на языке C/C++ следующие функции для работы с однонаправленными списками:

- создание списка;
- добавления элемента в список;
- удаления элемента из списка;
- просмотра списка.

8 ЛАБОРАТОРНАЯ РАБОТА №8 ПРОГРАММИРОВАНИЕ НЕЛИНЕЙНЫХ СТРУКТУР ДАННЫХ НА ЯЗЫКЕ С/С++

8.1 Цель работы

Изучение нелинейных структур данных и приобретение навыков разработки и отладки программ, использующих древовидные структуры. Исследование особенностей работы с поисковыми бинарными деревьями на языке С/С++.

8.2 Краткие теоретические сведения

8.2.1 Бинарные деревья и алгоритмы их обработки

В лабораторной работе требуется выполнить операции над бинарными деревьями. В бинарном дереве каждый узел содержит указатель на левое и правое поддерево. Поэтому узел дерева представляют в виде структуры с тремя полями:

```
struct tree
{   int  data;
   struct tree *left;
   sruct tree *riqht;
}
```

Здесь данные, содержащиеся в узле, представлены целыми значениями. При решении конкретной задачи необходимо внести соответствующие изменения в описание поля данных.

Обычно над бинарными деревьями выполняют следующие операции: организацию бинарного дерева, добавления узла к дереву, удаление узла из дерева, просмотр дерева, поиск соответствующего узла в дереве. Алгоритмы этих операций аналогичны рассмотренным ранее алгоритмам обработки деревьев средствами языка Паскаль [5].

Рассмотрим в качестве примера функцию, выполняющую добавление узла в дерево. Процесс добавления узла в дерево является рекурсивным. Если добавленный узел меньше, чем корневой элемент, то добавляем элемент в левое поддерево, иначе добавляем его в правое поддерево. Добавление узла выполняется в этом случае на уровне листьев дерева.

```
struct  tree  *addtree(struct tree *top, int newnode)
{  if (!top) {                      //если находимся на уровне листа,
        top=new tree;               //то выделить память под узел
        if (!top){
              printf("Исчерпана память     \n");
              return NULL;          //выход, если нет памяти
              }
        top->data=newnode;          //запись значения в узел
        top->left=NULL;             //обнуление указателей поддеревьев
        top->riqht=NULL;
}
else                                //иначе
        if (top->data<newnode)      //сравниваем значение узла с
```

```
                                        //добавляемым и добавляем узел
        top->left=addtree(top->left,newnode);     //либо в левое поддерево
        else
        top->riqht=addtree(top->riqht,newnode); //либо в правое поддерево
    return top;                          //возвращаем указатель на корень
    }
```

Если вызов **new** возвращает нулевой указатель, то функция **addtree** завершает свою работу сообщением "Исчерпана память" и возвращает нулевой указатель.

Применяя функцию **addtree** многократно можно разместить в узлах дерева необходимые данные, а затем организовать их обработку в соответствии с вариантом задания.

8.2.2 Пример программы обработки бинарного дерева

Требуется написать функции: создания, добавления листа в бинарное дерево, просмотра бинарного дерева, отображения структуры дерева, а также рекурсивную функцию, которая подсчитывает число вершин на n-ом уровне непустого дерева T (корень считать вершиной 1-го уровня).

Ниже приведен текст соответствующей программы с комментариями. С каждым узлом дерева связана структура, состоящая из фамилии и адреса грузополучателя. Дерево упорядочено по фамилиям.

Программа позволяет сохранять дерево в файле и восстанавливать дерево при её повторном запуске. При этом, в отличие от примера в предыдущей лабораторной работе, для передачи указателя по адресу соответствующий параметр (top) функции read_file объявляется как ссылка на указатель. Это упрощает обращение к функции (отпадает необходимость в операции взятия адреса &) и са текст функции (не нужна операция разадресации *).

```
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include <iomanip.h>
#include <conio.h>
//----------------------константы---------------------------------------
const int     d_f=20,              //длина строки с ф.и.о.
              d_a=80;              //длина строки с адресом
//--------------------структуры-----------------------------------------
struct gruz_p                      // Описание  записи о грузополучателе
      {char fio[d_f],
      address[d_a];
      };
struct node                  // Описание узла дерева
     { gruz_p data;
      node* left;
      node* right;
      };
```

```
//------------прототипы функций-----------------------------------
node* addtree(node *top,const gruz_p& newnode);   //добавление узла
int menu();                                        //отображение меню
int node_count(node *top,int level,int &n);       //подсчета количества вершин
                                                   //уровня Level
void otobr(node *top, int otstup);                 //отображение структуры //дерева
void prosmotr(node *top);                          //просмотр  значений узлов
                                                   //дерева слева направо
int write_file(ofstream &f, node* top);            //запись в файл
int read_file(char* filename, node* &top);         //чтение из файла
gruz_p vvod();                                     //ввод данных
//-----------------------основная функция-----------------------------
int main()
{     node *top=0;
     char *filename="gruz.txt";
     ofstream fout;                    //объявление выходного потока
     read_file(filename,top);          //чтение данных из файла
                                       //и создание дерева
     while (1)
     {   switch (menu())               //отображение и ввод пункта меню
        {case 1:                       //пункт 1
            top=addtree(top,vvod());   //ввод и  добавление элемента
            break;
        case 2:                        //пункт 2 – отображение структуры дерева
           otobr(top,1);
           cout<<"Нажмите любую клавишу "<<endl;
           cin.get();
           break;
        case 3:                        //пункт 3 – просмотр значений в узлах
           prosmotr(top);
           cout<<"Нажмите любую клавишу "<<endl;
           cin.get();
           break;
        case 4:                        //пункт 4 – подсчет количества вершин
           int level;                  //на уровне level дерева
           int n=0;
           cout<<"Введите значение уровня"<<endl;
           cin>>level;                 //ввод значения уровня
           cin.get();
           cout<<"На данном уровне:"<<node_count(top,level,n)
               <<"вершин"<<endl;
           cout<<"Нажмите любую клавишу "<<endl;
           cin.get();
           break;
        case 5:                        //пункт 5 – запись в файл
```

```
            fout.open(filename);          //открытие файла
            if (!fout){
                cout<<"Ошибка открытия файла"<<endl; return 1;}
            write_file(fout,top);         //сохранение данных в файле
            cout<<"Данные сохранены в файле: "<<filename<<endl;
            cout<<"============================="<<endl;
            fout.close();
            cout<<"Нажмите любую клавишу "<<endl;
            cin.get();
            break;
        case 6:                           //пункт 6 – выход
            return 0;
        default:                          //если неверно введен пункт меню
            cout<<"Вам следует ввести число от 1 до 6"<< endl;
            cin.get();
            break;
        }
    }
}
//----------------------добавление узла в дерево-------------------------
node* addtree(node *top,const gruz_p& newnode)
{       if (!top)                         //если находимся на уровне листа,
            {top=new node;                //то выделить память под узел
            if (!top) {cout<<"Не хватает памяти"<<endl;
                    return NULL;          //выход если память не выделена
            }
            top->data=newnode;            //запись данных в узел
            top->left=NULL;               //обнуление указателей
            top->right=NULL;
        }
        else                                                //иначе
            if (strcmp(top->data.fio,newnode.fio)>0)        //сравниваем значение в
                                                            узле с добавляемым и
            top->left=addtree(top->left,newnode);           //добавляем в левое
                                                            //поддерево
            else
            top->right=addtree(top->right,newnode);//или правое поддерево
return top;                           //возвращаем указатель на корень дерева
}
// ----------------отображение структуры дерева--------------------------
//      Функция  отображения структуры дерева.
//     Дерево отображается повернутым на 90 градусов против
//     часовой стрелки. Узлы дерева, находящиеся на одном
//     уровне, отображаются с  одинаковым отступом от края
//     экрана.}
```

```
void otobr(node *top, int otstup)
{       if (top)
                {otstup+=3;                     //отступ от края экрана
                otobr(top->right,otstup);       //обход правого поддерева
                //вывод значения фамилии, соответствующего узла
                cout<<setw(otstup)<<'*'<<top->data.fio<<endl;
                otobr(top->left,otstup);        //обход левого поддерева
        }
}
// ----------------просмотр дерева---------------------------------------
void prosmotr(node *top)                        //просмотр сверху вниз
{       if (top)
                {cout<<top->data.fio<<endl;     //вывод значения корня
                cout<<top->data.address<<endl;
                prosmotr(top->left);            //обход левого поддерева
                prosmotr(top->right);           //обход правого поддерева
        }
}
//-------------------ввод данных-----------------------------------------
gruz_p vvod()
{       gruz_p p;
        cout<<"Введите ф.и.о."<<endl;
        cin.getline(p.fio,d_f);
        cout<<"Введите адрес"<<endl;
        cin.getline(p.address,d_a);
        return p;
}
//---------------отображение и ввод пунктов меню-----------------
int menu()
{       char buf[10];   int item;
        do
          {clrscr();
           cout<<endl;
           cout<<"==========ДЕРЕВЬЯ==========="<<endl<<endl;
           cout<<"1- Добавить элемент в дерево"<<endl;
           cout<<"2- Отобразить структуру дерева"<<endl;
           cout<<"3- Просмотр дерева"<<endl;
           cout<<"4- Подсчет узлов на заданном уровне"<<endl;
           cout<<"5- Запись данных в файл"<<endl;
           cout<<"6- Выход"<<endl;
           cout<<"============================"<<endl;
           cout<<"Введите номер пункта меню"<<endl;
           cin>>buf;                            //ввод номера пункта
           cin.get();
           item=atoi(buf);                      //преобразовать его в целое
```

```
        if (!item)                                  //если ошибка
           {  cout<<"Вам следует ввести число от 1 до 6"<<endl;
              cin.get();
           }
      } while (!item);          //повторять пока не будет введено правильно
      return item;              //вернуть номер введенного пункта меню
}
//----------подсчет узлов на заданном уровне дерева------------
int node_count(node *top,int level,int &n)      //n передается по ссылке
{                                               //т.к. модифицируется
      if ((level>=1)&&top)
       {    if (level==1) n++;                  //увеличить счетчик, если
                                                //на уровне level есть вершина.
            n=node_count(top->left,level-1,n);  //подсчет узлов в левом
                                                //поддереве
            n=node_count(top->right,level-1,n);//подсчет узлов в правом
                                                //поддереве
      }
      return n;
}
// ----------------------------чтение файла-----------------------------
int read_file(char* filename, node* &top )
{     ifstream fin(filename,ios::in);   //открытие файла
      if (!fin) {cout<<"Не найден файл"<<filename<<endl; return 1;}
      gruz_p p;
      top = 0;
      while (fin.getline(p.fio,d_f))          //чтение ф.и.о. пока не конец файла
      {     fin.getline(p.address,d_a);       //чтение адреса
            top=addtree(top,p);               //добавить эл-т в дерево
      }
      return 0;
}
// --------------------запись данных в файл-----------------------------
int write_file(ofstream &f, node* top)
{     if (top)
         {  f<<top->data.fio<<endl;           //запись корня под(дерева)
            f<<top->data.address<<endl;
            write_file(f,top->left);          //запись левого поддерева
            write_file(f,top->right);         //запись правого поддерева
         }
      return 0;
}
```

Функции **write_file** передается ссылка на открытый выходной поток, так как она является рекурсивной. Это исключает возможность открытия потока внутри функции по аналогии с функцией **read_file**.

8.3 Варианты заданий

Представить приведенную в предыдущей работе таблицу в виде бинарного дерева. Написать функции создания и обхода дерева, а также одну из функций, приведенных ниже. Значения полей и количество записей в таблице студент выбирает самостоятельно. Программа должна сохранять дерево в файле и создавать его заново при её повторном запуске.

Вариант 1

Таблица 7.1 Функция, которая присваивает параметру E элемент из самого левого листа непустого дерева T.

Вариант 2

Таблица 7.2 Функцию, которая присваивает параметру E элемент из самого левого листа непустого дерева T.

Вариант 3

Таблица 7.3 Функцию, которая присваивает параметру E элемент из самого левого листа непустого дерева T.

Вариант 4

Таблица 7.1 Функцию, которая определяет уровень, на котором находится элемент E в дереве T.

Вариант 5

Таблица 7.2 Функцию, которая определяет уровень, на котором находится элемент E в дереве T.

Вариант 6

Таблица 7.3 Функцию, которая определяет уровень, на котором находится элемент E в дереве T.

Вариант 7

Таблица 7.1 Функцию, которая вычисляет среднее арифметическое всех элементов непустого дерева T (по одному из полей таблицы, которое имеет числовое значение)

Вариант 8

Таблица 7.2 Функцию, которая вычисляет среднее арифметическое всех элементов непустого дерева T (по одному из полей таблицы, которое имеет числовое значение).

Вариант 9

Таблица 7.3 Функцию, которая вычисляет среднее арифметическое всех элементов непустого дерева T (по одному из полей таблицы, которое имеет числовое значение).

Вариант 10

Таблица 7.1 Функцию, которая заменяет в дереве T все элементы меньшие, чем некоторое положительное число A, на это число (по одному из полей таблицы, которое имеет числовое значение).

Вариант 11

Таблица 7.2 Функцию, которая заменяет в дереве T все элементы меньшие, чем некоторое положительное число A, на это число (по одному из полей таблицы, которое имеет числовое значение).

Вариант 12

Таблица 7.3 Функцию, которая заменяет в дереве T все элементы меньшие, чем некоторое положительное число A, на это число (по одному из полей таблицы, которое имеет числовое значение).

Вариант 13

Таблица 7.1 Функцию, которая печатает элементы всех листьев дерева.

Вариант 14

Таблица 7.2 Функцию, которая печатает элементы всех листьев дерева.

Вариант 15

Таблица 7.3 Функцию, которая печатает элементы всех листьев дерева.

Вариант 16

Таблица 7.1 Функцию, которая находит в непустом дереве T длину (число ветвей) пути от корня до вершины с элементом E.

Вариант 17

Таблица 7.2 Функцию, которая находит в непустом дереве T длину (число ветвей) пути от корня до вершины с элементом E.

Вариант 18

Таблица 7.3 Функцию, которая находит в непустом дереве T длину (число ветвей) пути от корня до вершины с элементом E.

Вариант 19

Таблица 7.1 Функцию, которая подсчитывает число вершин на n-ом уровне непустого дерева T.

Вариант 20

Таблица 7.2 Функцию, которая подсчитывает число вершин на n-ом уровне непустого дерева T.

Вариант 21

Таблица 7.3 Функцию, которая подсчитывает число вершин на n-ом уровне непустого дерева T.

Вариант 22

Таблица 7.1 Написать рекурсивную функцию, которая определяет, входит ли элемент в дерево T.

Вариант 23

Таблица 7.2 Написать рекурсивную функцию, которая определяет, входит ли элемент в дерево T.

Вариант 24

Таблица 7.3 Написать рекурсивную функцию, которая определяет, входит ли элемент в дерево T.

Вариант 25

Таблица 7.1 Написать рекурсивную функцию, которая определяет число вхождений элемента E в дерево T.

Вариант 26

Таблица 7.2 Написать рекурсивную функцию, которая определяет число вхождений элемента E в дерево T.

Вариант 27

Таблица 7.3 Написать рекурсивную функцию, которая определяет число

вхождений элемента E в дерево T.

Вариант 28

Таблица 7.1 Написать рекурсивную функцию, которая вычисляет сумму элементов (по одному из полей таблицы) непустого дерева T.

Вариант 29

Таблица 7.2 Написать рекурсивную функцию, которая вычисляет сумму элементов (по одному из полей таблицы) непустого дерева T.

Вариант 30

Таблица 7.3 Написать рекурсивную функцию, которая вычисляет сумму элементов (по одному из полей таблицы) непустого дерева T.

8.4 Порядок выполнения работы

8.4.1 Изучить в ходе самостоятельной подготовки функции обработки бинарных деревьев на языке C/C++.

8.4.2 Описать элемент бинарного дерева в соответствии с вариантом задачи.

8.4.3 Разработать алгоритм решения задачи, разбив его на отдельные функции, так как указано в варианте задания.

8.4.4 Разработать программу на языке C/C++.

8.4.5 Разработать тестовые примеры, которые предусматривают проверку корректности работы программы в разных режимах.

8.4.6 Выполнить отладку программы

8.4.7 Получить результаты работы программы и исследовать её свойства в различных режимах работы, сформулировать выводы.

8.5 Содержание отчета

Цель работы, вариант задания, структурные схемы алгоритмов с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, выводы.

8.6 Контрольные вопросы

8.6.1 Что понимают под нелинейной структурой данных?

8.6.2 Что называется бинарным деревом данных?

8.6.3 Как построить упорядоченное бинарное дерево?

8.6.4 Отчего зависит эффективность поиска в бинарном дереве?

8.6.5 Что понимают под симметричным и выровненным деревом?

8.6.6 Как формируется выровненное дерево?

8.6.7 Напишите на языке C/C++ функции для работы с бинарными деревьями:

- создания упорядоченного бинарного дерева;
- добавления элементов в дерево;
- удаления листа дерева;
- обхода дерева.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Грошев, А.С. Информатика [Электронный ресурс] : учебник / А.С. Грошев, П.В. Закляков. — Электрон. дан. — Москва : ДМК Пресс, 2018. — 672 с. — Режим доступа: https://e.lanbook.com/book/108131.

2. Вирт, Н. Алгоритмы и структуры данных. Новая версия для Оберона [Электронный ресурс] : учебное пособие / Н. Вирт. — Электрон. дан. — Москва : ДМК Пресс, 2010. — 272 с. — Режим доступа: https://e.lanbook.com/book/1261.

3. Медведик, В.И. Практика программирования на языке Паскаль (задачи и решения) [Электронный ресурс] : учебное пособие / В.И. Медведик. — Электрон. дан. — Москва : ДМК Пресс, 2013. — 590 с. — Режим доступа: https://e.lanbook.com/book/58700.

4. Подбельский, В.В. Курс программирования на языке Си [Электронный ресурс] : учебник / В.В. Подбельский, С.С. Фомин. — Электрон. дан. — Москва : ДМК Пресс, 2012. — 384 с. — Режим доступа: https://e.lanbook.com/book/4148.

5. Кудинов, Ю.И. Практикум по основам современной информатики [Электронный ресурс] : учебное пособие / Ю.И. Кудинов, Ф.Ф. Пащенко, А.Ю. Келина. — Электрон. дан. — Санкт-Петербург : Лань, 2011. — 352 с. — Режим доступа: https://e.lanbook.com/book/68471.

6. Солдатенко, И.С. Практическое введение в язык программирования Си [Электронный ресурс] : учебное пособие / И.С. Солдатенко, И.В. Попов. — Электрон. дан. — Санкт-Петербург : Лань, 2018. — 132 с. — Режим доступа: https://e.lanbook.com/book/109619.

Заказ № _____ от «_____» _____2019г. Тираж _____экз.
Изд-во СевГУ