Министерство науки и высшего образования
Российской Федерации

КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МЕТОДЫ ПРОГРАММИРОВАНИЯ

Учебно-методическое пособие

Краснодар
2020

Министерство науки и высшего образования Российской Федерации
КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МЕТОДЫ ПРОГРАММИРОВАНИЯ

Учебно-методическое пособие

Краснодар
2020

УДК 004.432.2
ББК 32.973 (018.2)
М 44

Рецензенты:
Кандидат физико-математических наук, доцент
М.Е. Бегларян
Кандидат экономических наук, доцент
В.В. Ткаченко

М 44 Методы программирования: учеб.-метод. пособие / авт. В. В. Подколзин, А. Н. Полетайкин,Е. П. Лукащик, О. В. Гаркуша, С. Г. Синица, А. А. Полупанов, А. В. Харченко, А. В. Уварова, А. А. Михайличенко. – Краснодар: Кубанский гос. ун-т, 2020. – 174 с. – 500 экз.
ISBN 978-5-8209-1768-4

Содержит материал о методах программирования на языке C++, а также о базовых аспектах алгоритмизации.

Адресуется студентам I курса направления бакалавриата 01.03.02 «Прикладная математика и информатика», направления бакалавриата 09.03.03 «Прикладная информатика», направления бакалавриата 02.03.03 «Математическое обеспечение и администрирование информационных систем», направления бакалавриата 02.03.02 «Фундаментальные информатика и информационные технологии» для изучения практических основ программирования на языках высокого уровня.

УДК 004.432.2
ББК 32.973 (018.2)

ISBN 978-5-8209-1768-4

ПРЕДИСЛОВИЕ

Предлагаемое издание адресуется студентам направления «01.03.02 Прикладная математика и информатика», «02.03.02 Фундаментальная информатика и информационные технологии», «02.03.03 Математическое обеспечение и администрирование информационных систем», «09.03.03 Прикладная информатика», а также всем желающим изучить язык программирования C++. Пособие предлагается использовать при выполнении лабораторных работ студентам I курса факультета компьютерных технологий и прикладной математики по дисциплине «Методы программирования». Примеры, рассмотренные в данном издании, позволят студентам более полно разобраться в изучаемом материале, а решение задач по темам – закрепить полученные знания на практике.

Усвоение информации по основам программирования на языке C++ создаст базу для изучения материала по дисциплинам «Объектно-ориентированное программирование», «Основы программирования в RAD системах».

Структура учебно-методического пособия соответствует учебному плану рабочей программы дисциплины «Методы программирования», а представленный материал ориентирован на использование в рамках самостоятельной работы студентов. При подготовке к лабораторным занятиям студентам рекомендуется изучить теоретический материал и ознакомиться с примерами решения задач по темам. Контрольные задания, приведенные в конце каждой темы, могут использоваться преподавателями для текущего контроля уровня усвоения обучающимися материала, а также в качестве самостоятельной работы студентами.

Понятия, объекты, имена переменных или другие элементы, представленные в примерах и описаниях программного кода, подлежащие замене на конкретные значения, обозначаются в угловых скобках. Например:

```
<тело программы>
```

ВВЕДЕНИЕ

Среди современных языков программирования язык С является одним из наиболее распространенных. Язык С универсален, однако наиболее эффективно его применение в задачах системного программирования: разработке трансляторов, операционных систем, инструментальных средств. Язык С хорошо зарекомендовал себя эффективностью, лаконичностью записи алгоритмов, логической стройностью программ. Во многих случаях программы, написанные на языке С, сравнимы по скорости с программами, написанными на Ассемблере, при этом они более наглядны и просты в сопровождении.

Язык С имеет ряд существенных особенностей, которые выделяют его среди других языков программирования. В значительной степени на формирование идеологии языка повлияла цель, которую ставили перед собой его создатели – обеспечение системного программиста удобным инструментальным языком, который мог бы заменить Ассемблер. В результате появился язык программирования высокого уровня, обеспечивающий необычайно легкий доступ к аппаратным средствам компьютера. С одной стороны, как и другие современные языки высокого уровня, язык С поддерживает полный набор конструкций структурного программирования, модульность, блочную структуру программы. С другой стороны, язык С имеет ряд низкоуровневых черт, перечислим некоторые из них.

В языке С реализован ряд операций низкого уровня. Некоторые из таких операций напрямую соответствуют машинным командам, например, поразрядные операции, или операции ++ и --.

Базовые типы данных языка С отражают те же объекты, с которыми приходится иметь дело в программе на Ассемблере, – байты, машинные слова и т.д. Несмотря на наличие в языке С развитых средств построения составных объектов (массивов и структур), в нем практически отсутствуют средства для работы с ними как с единым целым.

Язык С поддерживает механизм указателей на переменные и функции. Указатель – это переменная, предназначенная для хранения машинного адреса некоторой переменной или функции. Поддерживается арифметика указателей, что позволяет осуществлять непосредственный доступ и работу с адресами памяти

практически так же легко, как на Ассемблере. Использование указателей дает возможность создавать высокоэффективные программы, однако требует от программиста особой осторожности.

Как никакой другой язык программирования высокого уровня, язык С «доверяет» программисту. Даже в таком существенном вопросе, как преобразование типов данных, налагаются лишь незначительные ограничения. Однако это также требует от программиста осторожности и самоконтроля.

Несмотря на эффективность и мощность конструкция языка С, он относительно мал по объему. В нем отсутствуют встроенные операторы ввода / вывода, динамического распределения памяти, управления процессами и т.п., однако в системное окружение языка С входит библиотека стандартных функций, в которой реализованы подобные действия.

Язык С++ – это язык программирования общего назначения, цель которого – сделать работу серьёзных программистов более приятным занятием. За исключением несущественных деталей язык С++ является надмножеством языка С. Помимо возможностей, предоставляемых языком С, язык С++ обеспечивает гибкие и эффективные средства определения новых типов.

Язык программирования служит двум взаимосвязанным целям: он предоставляет программисту инструмент для описания подлежащих выполнению действий и набор концепций, которыми оперирует программист, обдумывая, что можно сделать. Первая цель в идеале требует языка, близкого к компьютеру, чтобы все важные элементы компьютера управлялись просто и эффективно способом, достаточно очевидным для программиста. Язык С создавался на основе именно этой идеи. Вторая цель в идеале требует языка, близкого к решаемой задаче, чтобы концепции решения могли быть выражены понятно и непосредственно. Эта идея привела к пополнению языка С свойствами, превратившими его в язык С++.

Ключевое понятие в языке С++ – класс. Классы обеспечивают сокрытие информации, гарантированную инициализацию данных, неявное преобразование определяемых пользователем типов, динамическое определение типа, контроль пользователя над управлением памятью и механизм перегрузки операторов. Язык С++ предоставляет гораздо лучшие, чем язык С, средства для проверки типов и поддержки модульного программирования. Кроме того,

язык содержит усовершенствования, непосредственно не связанные с классами, такие как символические константы, встраивание функций вместо вызова, параметры функций по умолчанию, перегруженные имена функций, операторы управления свободной памятью и ссылки. Язык C++ сохраняет способность языка C эффективно работать с аппаратной частью на уровне битов, байтов, слов, адресов и т. д. Это позволяет реализовывать пользовательские типы с достаточной степенью эффективности.

В предлагаемом издании рассматриваются следующие теоретические и практические вопросы разработки программ на языке программирования C++: структурные, перечислимые типы в C++, линейные динамические информационные структуры, двоичные деревья, контейнеры, строки, файлы, методы представления и обработки графов.

1. СТРУКТУРЫ И ПЕРЕЧИСЛЕНИЯ В С++

Структура – это пользовательский тип данных, объединяемый в целое множество поименованных элементов, в общем случае разных типов.

Каждая структура включает один или несколько объектов (переменных, массивов, указателей, структур и т. д.), называемых элементами, или полями структуры (компонентами).

Поля структуры могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на него.

Структуры, так же, как и массивы, относятся к структурированным типам данных. Они отличаются от массивов тем, что, во-первых, к элементам структуры необходимо обращаться по имени, во-вторых, все поля структуры необязательно должны принадлежать одному типу.

Для объявления структур в языке С++ используется следующий формат:

```
struct [ имя_структуры ]
{
   тип_1 элемент_1;
   тип_2 элемент_2;
...
   тип_n элемент_n;
} [ список_описателей ];
```

Если список описателей отсутствует, описание структуры определяет новый тип данных, имя которого можно использовать в дальнейшем наряду со стандартными типами, например:

```
struct Worker { //описание нового типа Worker
   char fio[30];
   int age, code;
   double salary;
}; //описание заканчивается точкой с запятой!

Worker y; //определение переменной
Worker staff[100]; //определение массива
Worker *ps; //определение указателя
```

Если отсутствует имя типа, должен быть указан список описателей переменных, указателей или массивов. В этом случае описание структуры служит определением элементов этого списка, например:

```
    /*определение переменной, массива структур и
указателя на структуру*/
    struct {
       char fio[30];
       int age, code;
       double salary;
    } x, staff[100], *ps;
```

Имя структуры можно использовать сразу после его объявления (определение можно дать позднее) в тех случаях, когда компилятору не требуется знать размер структуры, к примеру:

```
    struct List; //объявление структуры List
    struct Link
    {
       List *p;          //указатель на структуру
List
       Link *prev; //указатель на структуру Link
    }

    struct List { // определение структуры List
       char fio[30];
       int age, code;

    };
```

Для инициализации структуры значения её элементов перечисляют в фигурных скобках в порядке их описания в самой структуре.

```
    struct Worker
    {
       char fio[30];
```

```
    int age, code;
    double salary;
};

Worker ivanov = {"Иванов И.И.", 31, 215,
5800.35};
```

При инициализации массивов структур следует заключать в фигурные скобки каждый элемент массива.

```
struct complex
{
    float re, im;
} compl [2] = {{1.3, 5.2}, {3.0, 1.5}};
```

Переменные структурного типа можно размещать и в динамической области памяти.

Доступ к полям структуры выполняется при помощи операций выбора . (точка) при обращении к полю через имя описателя, а также при помощи –> при обращении через указатель.

```
Worker worker, staff[100];

Worker *ps = new Worker;          /*создаёт
указатель на переменную структурного типа*/

Worker *mps = new Worker[5]; /*создаёт массив
структурного типа*/

worker.fio = "Петров С.С.";
staff[3] = worker;
staff[8].code = 123;
ps->salary = 4350.00;
ps->age = 41;
 (*ps).code   =   253;   /*обращение   через
разыменовывание указателя*/

mps[0].salary = 5800; /*обращение к 0 элементу
созданного массива через индекс*/
```

```
    (*(mps + 1)).salary = 4800; /*обращение к 1
элементу созданного массива через указатель*/


    //очистка занимаемой памяти
    delete ps;
    delete []mps;
```

Для переменных одного и того же структурного типа определена операция присваивания, при этом происходит поэлементное копирование. Но присваивание – это и все, что можно делать со структурами целиком. Другие операции, например сравнение на равенство или вывод, не определены. Структура интерпретируется компилятором как структурный тип, поэтому структуру можно использовать для передачи параметров функций и возвращать в качестве значения функции.

Размер структуры не обязательно равен сумме размеров её элементов, поскольку они могут быть выровнены по границам слова.

Если элементом структуры является другая структура (вложенные структуры), то доступ к её элементам выполняется через две операции выбора, например:

```
struct A
{
   int a;
   double x;
};

struct B
{
   A a;
   double x;
};

B x[2];

x[0].a.a = 1;
x[0].a.x = 35.15;
x[1].x = 0.1;
```

Как видно из примера, поля разных структур могут иметь одинаковые имена, поскольку у них разная область видимости.

Пример 1. Создать статический массив структур automobile из пяти элементов, содержащей поля: model, marka, year – для описания параметров автомобиля. Проинициализировать массив при его объявлении, распечатать все элементы массива на экране, используя функцию PrintAuto, выполнить сортировку элементов массива по убыванию года производства автомобилей, используя функцию SortYear, и распечатать полученную последовательность на экране.

```
#include <iostream>
#include <iomanip>

using namespace std;

struct automobile //структура автомобиля
{
    char model[10];//модель
    char marka[10];//марка
    int year;      //год производства
} car[5] = { { "opel", "astra", 2012 },
{ "opel", "omega", 2005 },
{ "honda", "civic", 2000 },
{ "kia", "ceed", 2012 },
{ "suzuki", "SX 4", 2010 } };

//вывод на экран массива структур
void PrintAuto(automobile car[], int size){
    for (int i = 0; i < size; i++)   {
cout   <<   "   Модель:   "   <<setw(10)   <<
car[i].model  <<  "  Марка:  "  <<  setw(10)  <<
car[i].marka << " Год: " << setw(5) << right <<
car[i].year << endl;
    } cout << endl;
}

void SortYear(automobile car[], int size){
    for (int i = size - 1; i >0 ; i--)
    {
```

```
            for (int j = 0; j < i; j++)
            {
                if (car[j].year < car[j + 1].year)
                {
                    automobile tmp = car[j];
                    car[j] = car[j + 1];
                    car[j + 1] = tmp;
                }
            }
        }
    }

    int main(){
        setlocale(LC_ALL, "Rus");
        cout << "Исходная база данных машин:\n";
        PrintAuto(&car[0], 5);
        cout << "Сортировка машин в порядке
убывания года производства:\n";
        SortYear(&car[0], 5);
        PrintAuto(car, 5);
        system("pause");
        return 0;
    }
```

Здесь в PrintAuto используются директивы вывода setw(N) и right, определенные в библиотеке iomanip. Директива setw(N) задает размер (количество символов) при выводе очередного значения (в случае необходимости заполняется пробелами), а right определяет выравнивание по правому краю внутри выделенного размера.

Пример 2. Создать динамический массив структур automobile из n элементов, содержащий поля: model, marka, year – для описания параметров автомобиля (модель, марка, год выпуска). Ввести значения полей переменной структурного типа при помощи функции InitStruct, распечатать все элементы структуры на экране, используя функцию PrintAuto, выполнить сортировку элементов массива по возрастанию года производства автомобилей, используя функцию SortYear, и распечатать полученный список структур на экране.

```
    #include <iostream>
```

```
#include <iomanip>

using namespace std;

struct automobile //структура автомобиля
{
    char model[10];
    char marka[10];
    int year;
};
//инициализируем массив структур
void InitStruct(automobile *car){
    cout << "Модель: "; cin >> car->model;
    cout << "Марка: "; cin >> car->marka;
    cout << "Год: "; cin >> car->year;
    cout << endl;
}

//вывод на экран массива структур
void PrintAuto(automobile *car){
cout << " Модель: " << setw(10) << car->model
<< " Марка: " << setw(10) << car->marka << " Год:
" << setw(5) << right << car->year << endl;
}

//сортировка в порядке возрастания года
производства
void SortYear(automobile *car, int size){
    for (int i = size - 1; i > 0;i--){
/*сортировка массива структур в порядке
возрастания */
        for (int j = 0; j < i; j++){
            if (car[j].year > car[j + 1].year){
                automobile tmp = car[j];
                car[j] = car[j + 1];
                car[j + 1] = tmp;
            }
        }
    }
}
```

```
int main(){
    setlocale(LC_ALL, "Rus");
    int size;//количество элементов

    cout << "Количество автомобилей: ";
    cin >> size;

    automobile *car = new automobile[size];
    cout << endl;
//ввод массива
    for (int i = 0; i < size; i++)
        InitStruct((car + i));

    cout << "Сортировка машин в порядке
возрастания года производства:\n";
    SortYear(car, size);

//вывод массива
    for (int i = 0; i < size; i++)
        PrintAuto((car + i));
    delete[] car;//освобождаем память
    system("pause");
    return 0;
}
```

При написании программ часто возникает потребность определить несколько именованных констант, для которых требуется, чтобы все они имели различные значения (при этом конкретные значения могут быть не важны). Для этого удобно воспользоваться перечисляемым типом данных, все возможные значения которого задаются списком целочисленных констант.

Для перечисляемого типа данных в языке C++ используется следующий формат:

```
enum [ имя типа ] { список_констант };
```

Имя типа задаётся в том случае, если в программе требуется определять переменные этого типа. Компилятор обеспечивает,

чтобы эти переменные принимали значения только из списка констант. Константы соответствуют целочисленным значениям и могут инициализироваться обычным образом. При отсутствии инициализатора первая константа будет рана нулю, а каждой следующей присваивается на единицу большее значение, чем предыдущей.

Допускается определять значения перечислителей. Они могут быть как положительными, так и отрицательными, или вообще иметь аналогичные другим перечислителям значения. Любые, не определённые пользователем перечислители, будут иметь значения на единицу больше, чем значения предыдущих перечислителей,

```
enum Err {ERR_READ, ERR_WRITE, ERR_CONVERT};

Err error;

switch (error){
    case ERR_READ:    /* операторы */ break;
    case ERR_WRITE:   /* операторы */ break;
    case ERR_CONVERT: /* операторы */ break;
}
```

Константам `ERR_READ`, `ERR_WRITE`, `ERR_CONVERT` присваиваются значения 0, 1 и 2 соответственно. Переменная `error` может принимать значения только этих констант.

Следующий пример демонстрирует возможность указания значений констант.

```
enum anothererr {two = -2, three, four,
ten = 10, eleven, fifty = ten + 40};
```

Константе `two` присваивается значение `-2`, константам `three` и `four` присваиваются значения `-3` и `-4`, константе `eleven` – 11. Имена перечисляемых констант должны быть уникальными, а значения могут совпадать.

При выполнении арифметических операций перечисления преобразуются в целые. Поскольку перечисления являются типами, определяемыми пользователем, для них можно вводить собственные операции.

Диапазон значений перечисления определяется количеством бит, необходимым для представления всех его значений. Любое значение целочисленного типа можно явно привести к типу перечисления, но при выходе за пределы его диапазона результат не определён.

Пример 3. используя перечисления в сочетании с оператором switch, разработать программу, которая по значению константы из перечисления MONTHS выводит название соответствующего месяца.

```
    #include <iostream>
    using namespace std;

    void main(void)
    {
        //перечисление, реализующее константы,
которые соответствуют месяцам года
        enum MONTHS {
            Jan = 1, Feb, Mar, Apr, May, Jun,
            Jul, Aug, Sep, Oct, Nov, Dec
        };

        MONTHS mn;

        mn = Mar; //mn = 3

        switch (mn){
        case Jan: cout << "January" << endl; break;
        case Feb: cout << "February" << endl;
break;
        case Mar: cout << "March" << endl; break;
        case Apr: cout << "April" << endl; break;
        case May: cout << "May" << endl; break;
        case Jun: cout << "June" << endl; break;
        case Jul: cout << "July" << endl; break;
        case Aug: cout << "August" << endl; break;
        case Sep: cout << "September" << endl;
break;
        case Oct: cout << "October" << endl; break;
```

```
        case Nov: cout << "November" << endl;
break;
        case Dec: cout << "December" << endl;
break;
        }
        system("pause");
    }
```

Контрольные вопросы и задания

1. Как определить (объявить) структуру?
2. Как проинициализировать структуру?
3. Какие варианты объявления с инициализацией вы знаете?
4. Как обратиться к полю структуры?
5. Как обратиться к полю структуры типа указатель?
6. Каким образом происходит обработка массивов структур?
7. Какие существуют способы ввода (задания) элементов структуры?
8. Как создать структуру в динамической памяти?
9. Как передать структуру в качестве параметра функции?
10. Как определить перечислимый тип?
11. Какие значения можно присваивать перечислителям?
12. Как автоматически присваиваются значения перечислителям?
13. Разработать структуру Student: Фамилия, Имя, Отчество, Дата рождения, Факультет, Курс. Создать массив структур не менее чем из трёх элементов. Вывести на экран:
    - список студентов заданного факультета;
    - списки студентов для каждого факультета и курса;
    - список студентов, родившихся после заданного года.
14. Разработать структуру Abiturient: Фамилия, Имя, Отчество, Адрес, Оценки. Создать массив структур не менее чем из трёх элементов. Вывести на экран:
    - список абитуриентов, имеющих неудовлетворительные оценки;
    - список абитуриентов, сумма баллов у которых не меньше заданной;

– выбрать N абитуриентов, имеющих самую высокую сумму баллов, и список абитуриентов, имеющих полупроходной балл.

15. Разработать структуру Aeroflot: Пункт назначения, Номер рейса, Тип самолета, Время вылета, Дни недели. Создать массив структур не менее чем из трёх элементов. Вывести на экран:

– список рейсов для заданного пункта назначения;
– список рейсов для заданного дня недели;
– список рейсов для заданного дня недели, время вылета для которых больше заданного.

16. Разработать структуру Book: Автор, Название, Издательство, Год, Количество страниц. Создать массив структур не менее чем из трёх элементов. Вывести на экран:

– список книг заданного автора;
– список книг, выпущенных заданным издательством;
– список книг, выпущенных после заданного года.

17. Разработать структуру Worker: Фамилия и инициалы, Должность, Год поступления на работу, Зарплата. Создать массив структур не менее чем из трёх элементов. Вывести на экран:

– список работников, стаж работы которых на данном предприятии превышает заданное число лет;
– список работников, зарплата которых больше заданной;
– список работников, занимающих заданную должность.

18. Разработать структуру Train: Пункт назначения, Номер поезда, Время отправления, Число общих мест, Купейных, Плацкартных. Создать массив структур не менее чем из трёх элементов. Вывести на экран:

– список поездов, следующих до заданного пункта назначения;
– список поездов, следующих до заданного пункта назначения и отправляющихся после заданного часа;
– список поездов, отправляющихся до заданного пункта назначения и имеющих общие места.

19. Разработать структуру Product: Наименование, Производитель, Цена, Срок хранения, Количество. Создать массив структур не менее чем из трёх элементов. Вывести на экран:

– список товаров для заданного наименования;

– список товаров для заданного наименования, цена которых не превышает указанной;
– список товаров, срок хранения которых больше заданного.

20. Разработать структуру Patient: Фамилия, Имя, Отчество, Адрес, Номер медицинской карты, Диагноз. Создать массив структур не менее чем из трёх элементов. Вывести на экран:
– список пациентов, имеющих данный диагноз;
– список пациентов, номер медицинской карты которых находится в заданном интервале.

21. Разработать структуру Bus: Фамилия и инициалы водителя, Номер автобуса, Номер маршрута, Марка, Год начала эксплуатации, Пробег. Создать массив структур не менее чем из трёх элементов. Вывести на экран:
– список автобусов для заданного номера маршрута;
– список автобусов, которые эксплуатируются больше 10 лет;
– список автобусов, пробег у которых больше 10 000 км.

22. Разработать структуру Customer: Фамилия, Имя, Отчество, Адрес, Телефон, Номер кредитной карточки. Создать массив структур не менее чем из трёх элементов. Вывести на экран:
– список покупателей в алфавитном порядке;
– список покупателей, номер кредитной карточки которых находится в заданном интервале.

23. Разработать структуру File: Имя файла, Размер, Дата создания, Количество обращений. Создать массив структур не менее чем из трёх элементов. Вывести на экран:
– список файлов, упорядоченный в алфавитном порядке;
– список файлов, размер которых превышает заданный;
– список файлов, число обращений к которым превышает заданное.

24. Разработать структуру Word: Слово, Номера страниц, на которых слово встречается (от 1 до 10), Число страниц. Создать массив структур не менее чем из трёх элементов. Вывести на экран:
– слова, которые встречаются более чем на N страницах;
– слова в алфавитном порядке;
– для заданного слова − номера страниц, на которых оно встречается.

25. Разработать структуру House: Адрес, Этаж, Количество комнат, Площадь. Создать массив структур не менее чем из трёх элементов. Вывести на экран:

- список квартир, имеющих заданное число комнат;
- список квартир, имеющих заданное число комнат и расположенных на этаже, который находится в определённом промежутке;
- список квартир, имеющих площадь, превосходящую заданную.

26. Разработать структуру Phone: Фамилия, Имя, Отчество, Адрес, Номер, Время внутригородских разговоров, Время междугородних разговоров. Создать массив структур не менее чем из трёх элементов. Вывести на экран:

- сведения об абонентах, время внутригородских разговоров которых превышает заданное;
- сведения об абонентах, воспользовавшихся междугородней связью;
- сведения об абонентах, выведенные в алфавитном порядке.

27. Разработать структуру Person: Фамилия, Имя, Отчество, Адрес, Пол, Образование, Год рождения. Создать массив структур не менее чем из трёх элементов. Вывести на экран:

- список граждан, возраст которых превышает заданный;
- список граждан с высшим образованием;
- список граждан мужского пола.

28. Разработать структуру Complex (действительная часть (re), мнимая часть (im)). Определить функции, выполняющие сложение, вычитание и умножение на действительную константу.

29. Разработать структуру Data (год, месяц, день). Определить функцию «дней с начала года», вычисляющую количество дней с начала года.

30. Разработать структуру Time (часы, минуты, секунды). Определить функцию «прошедшее время», определяющую интервал времени между t1 и t2 в минутах (округление производить в меньшую сторону).

31. Описать структуру Data (год, месяц, день). Определить функцию «дней до конца месяца», вычисляющую количество дней до конца месяца.

32. Описать структуру Decart для декартовых координат (x, y). Определить функцию для нахождения расстояния между двумя точками.

2. ЛИНЕЙНЫЕ ДИНАМИЧЕСКИЕ ИНФОРМАЦИОННЫЕ СТРУКТУРЫ

Динамические структуры данных — это любая структура данных, занимаемый объем памяти которой не является фиксированным.

Динамическая структура характеризуется следующими чертами:

1. Непостоянство и непредсказуемость размера (числа элементов) структуры в процессе ее обработки и компиляции. Количество элементов динамической структуры может изменяться от нуля до некоторого значения, определяемого спецификой соответствующей задачи или доступным объемом машинной памяти.

2. Отсутствие физической смежности элементов структуры в памяти. Логическая последовательность элементов задается в явном виде с помощью одного или нескольких указателей, или связок, хранящихся в самих элементах. Вследствие отсутствия физической смежности элементов память, занимаемая структурой, не представляет собой непрерывную область, т. е. элементы динамической структуры могут располагаться в памяти произвольным образом. Следствие такой особенности – усложнение процедур доступа к элементам динамической структуры по сравнению со статическими и полустатическими структурами.

Часто динамические структуры физически представляются в форме связных списков, поэтому их часто называют списковыми структурами. Связный список – такая структура, элементами которой служат записи с одним и тем же форматом, связанные друг с другом с помощью указателей, хранящихся в самих элементах. Простейшими связными списками являются линейные связные списки – односвязный список и двусвязный список.

2.1. ЛИНЕЙНЫЙ ОДНОСВЯЗНЫЙ СПИСОК

Наиболее простой способ объединить или связать некоторое множество элементов – представить его в «линейном» виде, т.е. организовать список. В этом случае каждому элементу нужно сопоставить ссылку на следующий элемент.

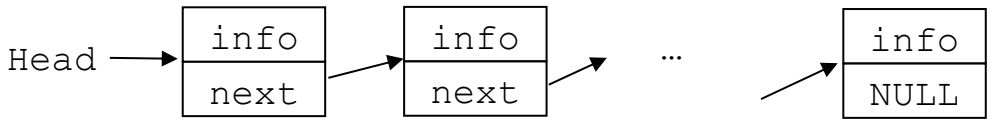
В односвязном списке каждый элемент состоит из двух различных по назначению составляющих: информационных полей и поля указателя на следующий элемент списка. В простейшем случае содержательное поле элемента списка хранит простое данное того или иного типа, например, целое число. Поле указателя хранит адрес следующего элемента списка. Пользуясь указателем, можно получить доступ к следующему элементу списка по его адресу, а из следующего элемента — к очередному элементу и т. д., пока не будет достигнут последний элемент. Поле указателя последнего элемента должно содержать признак пустого указателя `NULL`, свидетельствующий о конце списка. Так как каждый элемент списка содержит указатель только на следующий элемент, то для доступа ко всем его элементам выделяют специальную переменную, указывающую на первый элемент списка (голову списка).

Для иллюстрации определим следующий тип:

```
struct list  /*структура «линейный список»
             целых чисел*/
{
    int info;     //информационное поле
    list *next;   //указатель на следующий
элемент
};
```

Здесь `list` – идентификатор структурного типа данных, формально представляющего узел списка; `info` – поле структуры, принимающее данные для хранения (в данном примере это целое число); `next` – указатель на потенциальный соседний узел списка, который представлен аналогичным структурным типом данных `list`.

Графически линейный список можно представить так:



При описании любой структуры данных, в том числе и динамической, необходимо определить возможные операции над ее

элементами. Так, с элементами линейного списка выполняются следующие элементарные операции:

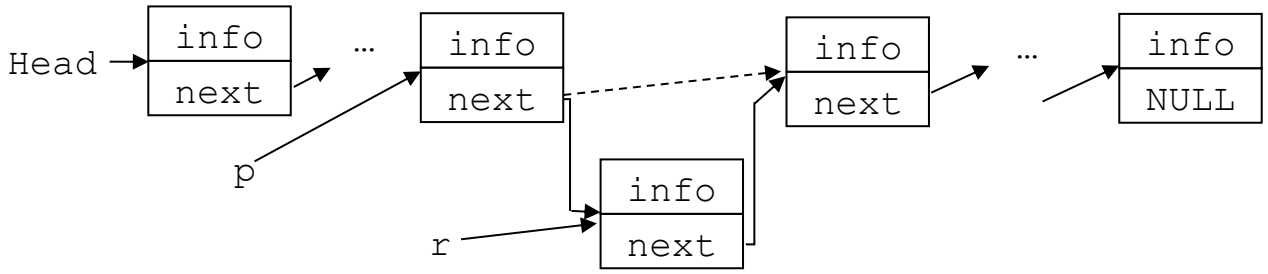
– включение нового элемента в заданное место списка;

– удаление элемента из списка;

– поиск элемента в списке и пр.

При написании процедур, реализующих основные операции над списками, будем предполагать наличие приведенных ранее описаний типов.

Включение нового элемента. Вставка элемента r в линейный список после элемента со ссылкой p приведет к следующему переопределению ссылок:

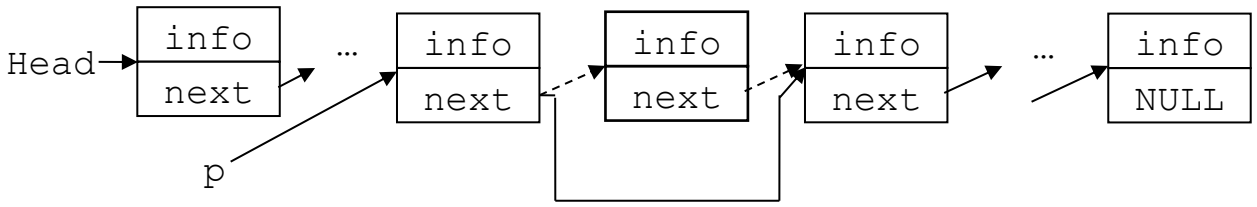


Для полноценного понимания организации работы с ЛДИС необходимо изучить тему «Указатели» [1, с. 121].

Процедура включения нового элемента после элемента со ссылкой p может выглядеть следующим образом:

```
/* Включение элемента в список после элемента
со ссылкой p */
void InsertElement(int data, list *&p){
    list *r = new(list);
    r->inf = data;
    r->next = p->next;
    p->next = r;
    p=r;   // p указывает на новый элемент
}
```

Удаление элемента из списка графически можно представить так:



И соответственно процедура удаления:

```
/* Удаление элемента, следующего за элементом,
на который указывает p */
    void Delete_ Element(list *p){
        if ((p->next)==NULL) return; /*проверяем
             существование следующего элемента*/
        list *r = p->next;  /* Сохраняем ссылку на
                        удаляемый элемент */
        p->next = r->next;
        delete r;
    }
```

Вывод списка на экран:

```
    void Print_List(list *p){ /*p – указывает на
                              голову списка*/
        while (p != NULL){
            cout << p->info << '\t';
            p = p->next;
        }
    }
```

В силу того что все элементы списка находятся в динамической памяти, обязательно освобождение. Освобождение памяти, выделенной для списка, может быть рализовано в следующем виде:

```
    void Delete_List(list *p){ /*p – указывает на
                              голову списка*/
        while (p != NULL){
            list *r = p;
            p = p->next;
            delete r;
        }
```

```
}
```

Пример 1. Сформировать список целых чисел. Признак завершения ввода — 0. Добавить в полученный список еще один элемент. Удалить из списка указанное значение.

```
#include <iostream>
using namespace std;

struct list{  /*структура «линейный список»
              целых чисел*/
    int info;    //информационное поле
    list  *next;    //указатель  на  следующий
элемент
};

void InsertElement(int data, list *&p){
    list *r = new(list);
    r->info = data;
    r->next = p->next;
    p->next = r;
    p = r;   // p указывает на новый элемент
}

void Delete_Element(list *p){
    if ((p->next) == NULL) return; /*проверяем
           существование          следующего
элемента*/
    list *r = p->next;  /* Сохраняем ссылку на
                      удаляемый элемент */
    p->next = r->next;
    delete r;
}

void Print_List(list *p){/*p - указывает на
                           голову списка*/
    while (p != NULL){
        cout << p->info << '\t';
        p = p->next;
    }
```

```
    }

    void Delete_List(list *p){
        while (p != NULL){
            list *r = p;
            p = p->next;
            delete r;
        }
    }

    int main(){
        setlocale(LC_ALL, "rus");
        list *head = NULL;
        int x;
        cout   <<   "Введите   элементы   (признак
завершения 0)" << endl;
        cin >> x;
        if (x != 0) {
            head = new (list); // Создание первого
            head->info = x;    //элемента
            head->next = NULL;
            cin >> x;
            list *p = head;
            while (x != 0) {
                InsertElement(x, p);
                cin >> x;
            }
        }
        cout << "Итоговый список: " << endl;
        Print_List(head);
        cout << endl;

        cout << "Введите добавляемый элемент x=";
    cin >> x;
        if (head == NULL) { //если список пуст
            head = new (list); // Создание первого
            head->info = x;    //элемента
            head->next = NULL;
        }
        else {
```

```
            list *p = head;    /* Поиск указателя
на последний элемент*/
            while ((p->next != NULL)) {
                p = p->next;
            }
            InsertElement(x,  p);  //добавление  в
конец
        }

        cout << "Новый список: " << endl;
        Print_List(head);  cout << endl;

        cout << "Элемент для удаления x=";   cin
>> x;
        if (head->info == x) {
            // Если это первый элемент, удаляем его
            list *r = head;
            head = head->next;
            delete r;
        }
        else {  // Ищем среди остальных...
            list *p = head;    // Поиск ссылки на
элемент x
            while ((p->next != NULL)
                && (p->next->info != x)) {
                p = p->next;
            }
            if (p->next != NULL) {
                Delete_Element(p);
            }
            else {
                cout  <<  "x="  <<  x  <<  "  не
найден!\t";
            }
        }

        cout << "Итоговый список: " << endl;
        Print_List(head);
        cout << endl;
```

```
    system("pause");
    Delete_List(head);
    return 0;
}
```

Следует обратить внимание, что в процедуре `InsertElement` формальный параметр `p` указан как ссылка и он изменяется, что также влияет на фактический параметр. Этот факт учитывается в цикле ввода и формирования списка основной программы.

Пример 2. Сформировать список целых чисел, отсортированный по неубыванию. Признак завершения ввода — 0. Определить функции добавления и удаления значения.

```
#include <iostream>
using namespace std;

struct list {
    int info;
    list *next;
};

list *InsertValue(int data, list *&head){
/*функция возвращает указатель на созданный элемент*/
    if (head == NULL || head->info>= data) {
/*первый элемент списка или вставить перед первым*/
        list *p = head;
        head = new(list);
        head->info = data;
        head->next = p;
        return head;
    }
    list *p;
    /* ищем элемент (указатель p), после которого нужно поставить новый*/
    for (p = head; p->next != NULL && (p->next)->info < data; p = p->next);

    list *r=new(list);
```

```
        r->info = data;
        r->next = p->next;
        p->next = r;
        return r;
    }

    void DeleteValue(int data, list *&head) {
        if (head == NULL) return; /*проверка
списка на пустоту*/
        if (head->info == data) { /*удаляется
первый элемент списка*/
            list *p=head;
            head = head->next;
            delete p;
            return;
        }
        list *p;
        /* ищем элемент(указатель p), после
которого возможно стоит удаляемый*/
        for (p = head; p->next != NULL &&
                (p->next)->info < data; p = p-
>next);
        if (p->next != NULL && (p->next)->info ==
data) { /*если значение присутствует в списке*/
            list *r= p->next;
            p->next = r->next;
            delete r;
        }
    }

    void PrintList(list *head){
        for (list *p = head; p != NULL; p = p-
>next){
            cout << p->info << '\t';
        }
    }

    void DeleteList(list *&head){
        while (head != NULL) {
            list *r = head;
```

```
            head = head->next;
            delete r;
        }
    }

    int main(){
        setlocale(LC_ALL, "rus");
        list *head = NULL;
        int x;
        cout << "Введите элементы (признак
завершения 0)" << endl;
        for (cin >> x; x != 0; cin >> x)
InsertValue(x, head);

        cout << endl << "Итоговый список: " <<
endl;
        PrintList(head);

        cout << endl<< "Введите удаляемое
значение: "; cin >> x;
        DeleteValue(x, head);

        cout << endl << "Новый список: " << endl;
        PrintList(head);

        cout << endl; system("pause");
        DeleteList(head);
        return 0;
    }
```

Обратите внимание, что в функциях `InsertValue` и `DeleteValue` для поиска указателя `p` на нужный элемент используется аналог цикла `while`, реализованный в виде цикла `for` с пустым телом.

2.2. СТЕК

Стек — это ЛДИС, доступ к элементам которого определяется по принципу LIFO (Last In – First Out: последним пришел, первым ушел).

Элемент, или узел, который добавляется в стек последним, называется верхушкой стека. Позицию этого элемента также называют верхушкой. Элемент, или узел, помещенный в стек первым, называют дном стека. В любой момент в стеке доступен только один элемент — верхушка стека.

Функционально стек похож на стопку тарелок. Когда мы заносим элемент в стек (добавлям тарелку в стопку), то он помещается поверх прежней вершины (кладем на самую верхнюю тарелку) и теперь сам находится на вершине стека. При выборе элемента из стека, он удаляется, а находящийся под ним становится верхушкой стека.

Над стеком обычно определены операции:

– создать стек;

– добавить элемент;

– извлечь удалить;

– проверка на пустоту (булевская операция, возвращает true, если в стеке есть хотя бы один узел).

Стек как абстрактное хранилище информации имеет бесконечный объем. Однако на практике имеющийся в распоряжении программы объем памяти всегда конечен и, следовательно, можно использовать только стек ограниченного объема, так называемый ограниченный стек. Очевидно, что для любого стека операция «извлечь из стека» определена только тогда, когда стек не пуст, а операция «добавить в стек» является частичной для ограниченного стека. Ситуация, в которой делается попытка добавить в ограниченный стек элемент, а имеющаяся память уже исчерпана, называется операцией переполнения.

Стек — это частная реализация линейного односвязного списка. Рассмотрим пример реализации стека целых чисел. Определим структуру:

```
struct stack{
    int info;
    stack *next;
};
```

Рассмотрим реализацию основных операций над стеком.

Создание стека:

```
stack *CreateStack(){
    return NULL;
}
```

Проверка стека на пустоту:

```
bool IsEmptyStack(stack *top){
    return  top == NULL;
}
```

Здесь и далее указатель `top` определяет стек и указывает на элемент, являющийся его верхушкой.

Включение нового элемента в стек:

```
void StackPush (int data, stack *&top){
    stack *r = new(stack);
    r->info = data;
    r->next = top;
    top = r;
}
```

Извлечение элемента из стека:

```
int StackPop (stack *&top)
{
    if (!IsEmptyStack(top)) {
        stack *r=top;
        int x = r->info;
        top = top ->next;
        delete r;
        return x;
    }
    return NAN;
}
```

При выполнении этой операции информационная часть элемента, находящегося на вершине стека, возвращается функцией, а сам элемент удаляется. Если стек пуст, то возвращается пустое значение.

Пример 3. Сформировать стек целых чисел. Признак завершения ввода — 0. Вывести содержимое стека с удалением.

```
#include <iostream>
using namespace std;

struct stack {
    int info;
    stack *next;
};

stack *CreateStack() {
    return NULL;
}

bool IsEmptyStack(stack *top) {
    return  top == NULL;
}

void StackPush(int data, stack *&top) {
    stack *r = new(stack);
    r->info = data;
    r->next = top;
    top = r;
}

int StackPop(stack *&top){
    if (!IsEmptyStack(top)) {
        stack *r = top;
        int x = r->info;
        top = top->next;
        delete r;
        return x;
    }
    return NAN;
}

void PrintStack(stack *&top){
    while (!IsEmptyStack(top)){
        cout << StackPop(top) << '\t';
```

```
        }
    }

    void DeleteStack(stack *&top){
        while (!IsEmptyStack(top)){
            StackPop(top);
        }
    }

    int main(){
        setlocale(LC_ALL, "rus");
        stack *Top = CreateStack();
        int x;
        cout << "Введите элементы (признак
завершения 0)" << endl;
        for (cin >> x; x != 0; cin >> x)
StackPush(x, Top);

        cout << endl << "Элементы стека: " << endl;
        PrintStack(Top);
        system("pause");
        return 0;
    }
```

Пример 4. Вводится последовательность символов `')'`, `'('`, `']'`, `'['`. Определим правильную скобочную последовательность по следующему правилу:

1) пустая строка является правильной скобочной последовательностью;

2) если `A` — правильная скобочная последовательность, то `(A)` и `[A]` — правильные скобочные последовательности.

Определить, является ли введенная последовательность скобок правильной скобочной последовательностью.

```
    #include <iostream>
    using namespace std;

    struct stack { //определяем ЛДИС символов
        char symbol;
```

```
    stack *next;
};

stack *CreateStack() {
    return NULL;
}

bool IsEmptyStack(stack *top) {
    return  top == NULL;
}

void StackPush(char data, stack *&top) {
    stack *r = new(stack);
    r->symbol = data;
    r->next = top;
    top = r;
}

char StackPop(stack *&top)
{
    if (!IsEmptyStack(top)) {
        stack *r = top;
        char x = r->symbol;
        top = top->next;
        delete r;
        return x;
    }
    return 0; /*если стек пуст, то возращается
пустой символ с кодом 0*/
}

void DeleteStack(stack *&top) {
    while (!IsEmptyStack(top)) {
        StackPop(top);
    }
}

int main(){
    setlocale(LC_ALL, "rus");
    stack *Top = CreateStack();
```

```
        bool flag = true; /*флаг проверки, равен
false если хоть раз произошло нарушение*/
        for (char x = getchar(); x != '\n'; x =
getchar()) { //считываем посимвольно до Enter
            switch (x) {
            case '(':case'[': {StackPush(x, Top);
break; }
            case      ')':      {flag=flag      &&
(StackPop(Top)=='('); break; }
            case    ']':    {flag    =    flag    &&
(StackPop(Top) == '['); break; }
            default   :   flag   =   false;/*введен
неверный символ*/
            }
        }
        flag  =  flag  &&  IsEmptyStack(Top);  /*
проверка не остались ли еще открытые скобки*/
        DeleteStack(Top);
        if (flag) cout<<"Да"<<endl;
        else cout << "Нет" << endl;
        system("pause");
        return 0;
    }
```

2.3. ОЧЕРЕДЬ

Очередь — это ЛДИС, доступ к узлам которой осуществляется по схеме FIFO (First In — First Out: первым пришел, первым ушел). Можно говорить, что у очереди есть начало и есть конец. Добавляются элементы в начало очереди, а удаляются из конца. Над очередью определяются операции, аналогичные операциям над стеком. Аналогично стеку определяется понятие ограниченной очереди.

Очередь — это частная реализация линейного односвязного списка. Рассмотрим пример реализации очереди целых чисел. Определим структуру:

```
    struct queue{
        int info;
        queue *next;
```

```
};
```

Рассмотрим реализацию основных операций над очередью.
Создание очереди:

```
queue *CreateQueue(){
    return NULL;
}
```

Проверка очереди на пустоту:

```
bool IsEmptyQueue (queue *begin){
    return begin == NULL;
}
```

Здесь и далее указатель `begin` определяет указатель на первый элемент очереди.
Включение нового элемента в очередь:

```
void QueuePushBack (int data, queue *&begin){
    queue *r = new(queue);
    r->info = data;
    r->next = NULL;
    if (begin==NULL){//очередь пуста
        begin=r;
        return;
    }
    queue *p = begin;
    while (p->next!=NULL) p = p->next; /*ищем
последний элемент очереди*/
    p->next=r;
}
```

Извлечение элемента из очереди:

```
int QueuePop (queue *&begin){
    if (!IsEmptyQueue(begin)) {
        queue *r= begin;
        int x = r->info;
```

```
        begin = begin ->next;
        delete r;
        return x;
    }
    return NAN;
}
```

При выполнении этой операции информационная часть элемента, находящегося в начале очереди, возвращается функцией, а сам элемент удаляется. Если очередь пуста, то возвращается пустое значение.

Пример 5. Сформировать очередь целых чисел. Признак завершения ввода — 0. Вывести содержимое очереди после первого отрицательного.

```
#include <iostream>
using namespace std;

struct queue {
    int info;
    queue *next;
};

queue *CreateQueue() {
    return NULL;
}

bool IsEmptyQueue(queue *begin) {
    return begin == NULL;
}

void QueuePushBack(int data, queue *&begin) {
    queue *r = new(queue);
    r->info = data;
    r->next = NULL;
    if (begin == NULL) {
        begin = r;
        return;
    }
```

```
        queue *p = begin;
        while (p->next != NULL) p = p->next; /*ищем
последний элемент очереди*/
        p->next = r;
    }

    int QueuePop(queue *&begin){
        if (!IsEmptyQueue(begin)) {
            queue *r = begin;
            int x = r->info;
            begin = begin->next;
            delete r;
            return x;
        }
        return NAN;
    }


    void DeleteQueue(queue *&begin) {
        while (!IsEmptyQueue(begin)) {
            QueuePop(begin);
        }
    }

    int main(){
        setlocale(LC_ALL, "rus");
        queue *Begin = CreateQueue();
        int x;
        cout   <<   "Введите   элементы   (признак
завершения 0)" << endl;
        for  (cin  >>  x;  x  !=  0;  cin  >>  x)
QueuePushBack(x, Begin);

        while       (!IsEmptyQueue(Begin)       &&
QueuePop(Begin) >= 0);
        cout << "Ответ: "<<endl;
        while (!IsEmptyQueue(Begin)) {
            cout << QueuePop(Begin) << '\t';
        }
        cout << endl;
```

```
    system("pause");
    return 0;
}
```

Пример 6. Дана последовательность целых чисел. Признак завершения ввода — 0. Определить, является ли она симметричной.

Заметим, что для решения этой задачи можно использовать факт равенства содержимого стека и очереди для данной последовательности.

```
#include <iostream>
using namespace std;

/* определение очереди*/
struct queue {
    int info;
    queue *next;
};

queue *CreateQueue() {
    return NULL;
}

bool IsEmptyQueue(queue *begin) {
    return begin == NULL;
}

void QueuePushBack(int data, queue *&begin) {
    queue *r = new(queue);
    r->info = data;
    r->next = NULL;
    if (begin == NULL) {
        begin = r;
        return;
    }
    queue *p = begin;
    while (p->next != NULL) p = p->next; /*ищем
последний элемент очереди*/
    p->next = r;
}
```

```
int QueuePop(queue *&begin){
    if (!IsEmptyQueue(begin)) {
        queue *r = begin;
        int x = r->info;
        begin = begin->next;
        delete r;
        return x;
    }
    return NAN;
}


void DeleteQueue(queue *&begin) {
    while (!IsEmptyQueue(begin)) {
        QueuePop(begin);
    }
}

/* определение стека*/
struct stack {
    int info;
    stack *next;
};

stack *CreateStack() {
    return NULL;
}

bool IsEmptyStack(stack *top) {
    return  top == NULL;
}

void StackPush(int data, stack *&top) {
    stack *r = new(stack);
    r->info = data;
    r->next = top;
    top = r;
}
```

```
int StackPop(stack *&top) {
    if (!IsEmptyStack(top)) {
        stack *r = top;
        int x = r->info;
        top = top->next;
        delete r;
        return x;
    }
    return NAN;
}

void DeleteStack(stack *&top) {
    while (!IsEmptyStack(top)) {
        StackPop(top);
    }
}

int main(){
    setlocale(LC_ALL, "rus");
    queue *Begin = CreateQueue();
    stack *Top = CreateStack();
    int x;
    cout << "Введите элементы (признак
завершения 0)" << endl;
    for (cin >> x; x != 0; cin >> x) {
        QueuePushBack(x, Begin);
        StackPush(x, Top);
    }

    while (!IsEmptyQueue(Begin) ||
!IsEmptyStack(Top)) {
        if (QueuePop(Begin) != StackPop(Top))
{
            cout << "НЕТ" << endl;
            DeleteQueue(Begin);
            DeleteStack(Top);
            system("pause");
            return 0;
        }
    }
```

```
    cout << "ДА"<<endl;
    system("pause");
    return 0;
}
```

Контрольные вопросы и задания

1. Описать процедуру, которая:
   – вставляет в список L новый элемент A за каждым вхождением элемента E;
   – удаляет из списка L все отрицательные элементы.
2. Описать процедуру, которая:
   – вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом;
   – удаляет из списка L за каждым вхождением элемента E один элемент, если такой есть и он отличен от E.
3. Описать процедуру, которая:
   – вставляет в непустой список L, элементы которого упорядочены по неубыванию, новый элемент E так, чтобы сохранилась упорядоченность;
   – удаляет из списка L первый отрицательный элемент.
4. Описать процедуру, которая:
   – проверяет, есть ли в списке L хотя бы два одинаковых элемента;
   – добавляет в конец списка L1 все элементы списка L2.
5. Описать процедуру, которая вставляет в список L за первым вхождением элемента E все элементы списка L1, если E входит в L.
6. Описать процедуру, которая:
   – переворачивает список L, т.е. изменяет ссылки в этом списке так, чтобы его элементы оказались расположенными в обратном порядке;
   – подсчитывает число вхождений элемента E в список L.
7. Описать процедуру, которая:
   – формирует список L, включив в него по одному разу элементы, которые входят хотя бы в один из списков L1 и L2;
   – удаляет из непустых слов списка L их первые буквы.
8. Описать процедуру, которая:

– формирует список L, включив в него по одному разу элементы, которые входят одновременно в оба списка L1 и L2;
– определяет количество слов в непустом списке L, отличных от последнего.

9. Описать процедуру, которая:
– формирует список L, включив в него по одному разу элементы, которые входят в список L1, но не входят в список L2;
– в списке L переставляет местами первое и последнее непустые слова, если в L есть хотя бы два непустых слова.

10. Описать процедуру, которая:
– формирует список L, включив в него по одному разу элементы, которые входят в один из списков L1 и L2, но в то же время не входят в другой из них;
– подсчитывает количество слов списка L, у которых первая и последняя буквы совпадают.

11. Описать процедуру, которая объединяет два упорядоченных по неубыванию списка L1 и L2 в один упорядоченный по неубыванию список, меняя соответствующим образом ссылки в L1 и L2 и присвоив полученный список параметру L1.

12. Описать процедуру, которая в списке L заменяет первое вхождение списка L1 (если такое есть) на список L2.

13. Описать процедуру, которая проверяет на равенство два многочлена (указывается коэффициент и степень каждого элемента).

14. Описать процедуру, которая вычисляет значение многочлена в некоторой целочисленной точке.

15. Описать процедуру, которая строит многочлен p — сумму многочленов q и r.

3. ЛИНЕЙНЫЕ ДВУНАПРАВЛЕННЫЕ СВЯЗНЫЕ СПИСКИ

Линейный двунаправленный список является разновидностью связных списков и характеризуется наличием пары указателей в каждом элементе: на предыдущий элемент и на следующий, как показано на рис. 1.

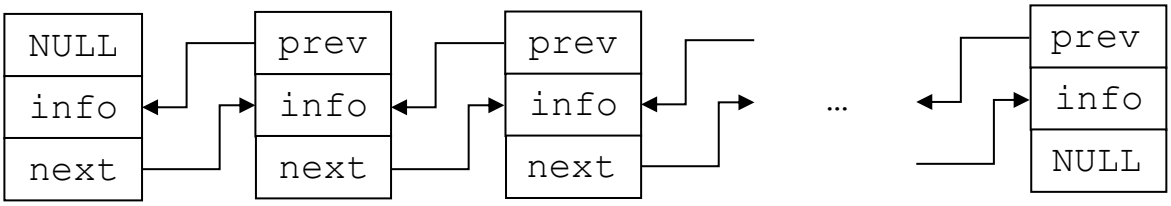


Рис. 3.1. Представление двунаправленного списка в памяти

Преимущество: от данного элемента структуры возможно перейти в обе стороны, что упрощает некоторые операции, например, перестановку элементов; для доступа к элементам списка достаточно знать адрес любого его элемента.

Недостаток: на «обратные» указатели выделяется дополнительная память, что при нелокальном их распределении в памяти ЭВМ снижает эффективность обработки данных.

Базовая статическая структура, описывающая узел двунаправленного связного списка имеет вид:

```
struct list {
    int info;
    list *prev;
    list *next;
};
```

Здесь `list` – идентификатор структурного типа данных, формально представляющего узел списка; `list` – поле структуры, принимающее данные для хранения (в данном примере это целое число); `next` и `prev` – указатели на потенциальные соседние узлы списка, которые представлены аналогичным структурным типом данных `list`.

Данные поля info могут быть любого типа, в том числе и структурного. В последнем случае каждый узел может хранить типовой набор данных.

Как и в случае с однонаправленным списком, создание двунаправленного списка сводится к последовательному созданию его отдельных узлов. При этом предварительно формируется элемент данных для записи в новый узел списка. В силу того что по двунаправленному списку можно «передвигаться в обе стороны», то для доступа к его элементам достаточно знать указатель на любой из них. Первым (головой) элементом двунаправленного списка считается эелемент, у которого поле `prev` имеет значение `NULL`, соответственно последним – у которого `next` имеет значение `NULL`. Пример кода программы, создающей первый узел списка:

```
        list *element = new list;
        element ->info = 0;
        element  ->prev  =  NULL;  /*  признак
головного узла (начала списка)*/
        element ->next = NULL; /* признак конца
списка*/
```

Здесь в качестве данных в узел заносится константа `0`, а поля-указатели инициализируются значениями `NULL`. Последнее означает, что данный узел является одновременно и головой, и концом списка.

Дальнейшее формирование двунаправленного списка также может осуществляться добавлением нового узла в любой заданной допустимой позиции, в том числе в начало или в конец списка. Однако в отличие от однонаправленного списка, двунаправленный список предполагает установление связи встраиваемого элемента и с предшествующим ему элементом (если таковой имеется), а также со следующим за ним элементом (если таковой имеется) и этого следующего со встраиваемым. То есть в данном случае устанавливаются двунаправленные связи.

При добавлении нового узла в начало списка указатель `prev` текущего головного элемента получает значение ссылки на новый элемент, а указатель `next` нового элемента инициализируется ссылкой на текущую голову. Указатель `prev` нового узла остается

пустым, тем самым делая его новой головой списка. В данном случае также устанавливаются две противоположные связи между новым узлом и текущим головным элементом. Код функции AddToBegin, реализующей данную операцию:

```
list *AddToBegin(list *elem, int data) {
    if (elem==NULL){
elem=new list;
        elem ->info = data;
        elem ->prev = NULL;
        elem ->next = NULL;
        return elem;
    }
    while (elem->prev!= NULL)//ищем первый
     elem = elem->prev;       //эл-т списка
    list *p = new list;
    p->info = data;
    p->prev = NULL;
    p->next = elem;
    elem->prev = p;
    elem= p;
    return p;
}
```

В качестве параметров AddToBegin получает указатель на произвольный элемент двунаправленного списка и добавляемое значение.

В начале функции проверяется на пустоту списка. Здесь и далее параметр elem указывает на произвольный (существующий) элемент списка. Если elem равен NULL, то считается что список пуст. Цикл while обеспечивает переход к первому элементу списка, указатель на который хранится в elem. Далее создается указатель p на новый элемент типа list, информационное поле которого инициализируется элементом данных data, передаваемым из основной программы. Указатель next получает ссылку на первый элемент текущего списка, а указатель prev получает значение NULL, превращая таким образом новый элемент данных в первый элемент списка. Указатель prev элемента списка, бывшего первого текущего списка, и представленный переменной elem, получает

значение указателя на новый элемент, что достигается в предпоследней строке тела функции AddToBegin.

Так, при добавлении нового узла в конец списка, показанного на рис. 1, указатель next последнего элемента примет значение ссылки на новый элемент типа list, размещенный в памяти ЭВМ и представляющий новый узел списка. При этом поля-указатели этого нового элемента инициализируются следующим образом: указатель prev получает значение ссылки на текущий последний элемент списка, а указатель next остается пустым (NULL). Таким образом, новый узел становится последним.

Код функции AddToEnd, реализующей данную операцию (data – элемент данных целого типа):

```
list *AddToEnd(list *elem, int data) {
    if (elem==NULL){
elem=new list;
        elem ->info = data;
        elem ->prev = NULL;
        elem ->next = NULL;
        return elem;
    }
    while (elem->next != NULL)//ищем последний
     elem = elem->next;        //эл-т списка
    list *p = new list;
    p->info = data;
    p->prev = elem;
    p->next = NULL;
    elem->next = p;
    return p;
}
```

В качестве параметров AddToEnd получает указатель на произвольный элемент двунаправленного списка и добавляемое значение.

Цикл while обеспечивает переход к последнему элементу списка, указатель на который хранится в elem. Далее создается указатель p на новый элемент типа list, информационное поле которого инициализируется элементом данных data, передаваемым из основной программы. Указатель prev получает ссылку на

последний элемент текущего списка, а указатель next получает значение NULL, превращая таким образом новый элемент данных в последний элемент списка. Указатель же next элемента списка, бывшего последним текущего списка и представленный переменной elem, получает значение указателя на новый элемент, что достигается в предпоследней строке тела функции AddToEnd.

Добавление элемента в произвольную позицию списка должно быть конкретизировано дополнительным условием. В качестве такого условия могут выступать: номер позиции (индекс) нового элемента в списке; указатель на элемент, до / после котого необходимо добавить элемент; условие на информационное поле элемента, перед / после которым добавляется элемент.

В случае добавления нового элемента в заданную позицию внутри списка фактически выполняется разрыв списка в заданной позиции и переопределение значений указателей текущих соседних элементов значениями ссылки на новый элемент в памяти ЭВМ. Поля же указателей prev и next нового элемента инициализируются соответственно ссылками на предшествующий ему (сосед слева) и последующий ему (сосед справа) элемент списка. Код функции AddToPos, реализующей данную операцию:

```
    list *AddToPos(list *elem, int data, int pos){
        while (elem->prev!= NULL)//ищем первый
         elem = elem->prev;          //эл-т списка
        if (pos==1)
            return AddToBegin(elem, data);
        for (int i=2; elem->next!=NULL && i<pos;
i++)
            elem = elem ->next;
        if (elem->next == NULL)
            return AddToEnd(elem, data);

        list* p = new list;
        p->info = data;
        p->prev = elem;
        p->next = elem ->next;
        elem ->next = p;
        p->next->prev = p;
        return p;
```

```
}
```

В данной функции указан параметр `pos`, определяющий позицию вставки нового узла, если `pos` превышает количество элементов в списке, то добавление производится в конец списка. Цикл `for` обеспечивает переход к позиции вставки. Дальнейшие действия по инициализации полей `prev` и `next` нового узла привязывают его к текущему и следующему за ним узлам. После цикла `elem` указывает на элемент, после которого будет добавлен новый. Последние две строки тела функции устанавливают обратные связи с новым узлом.

Расширенный таким образом исходный список (рис. 1) показан на рис. 2, где добавленный узел указан во второй позиции и выделен заливкой. В данном случае было установлено две пары противоположных связей между новым узлом и бывшими соседними узлами списка.

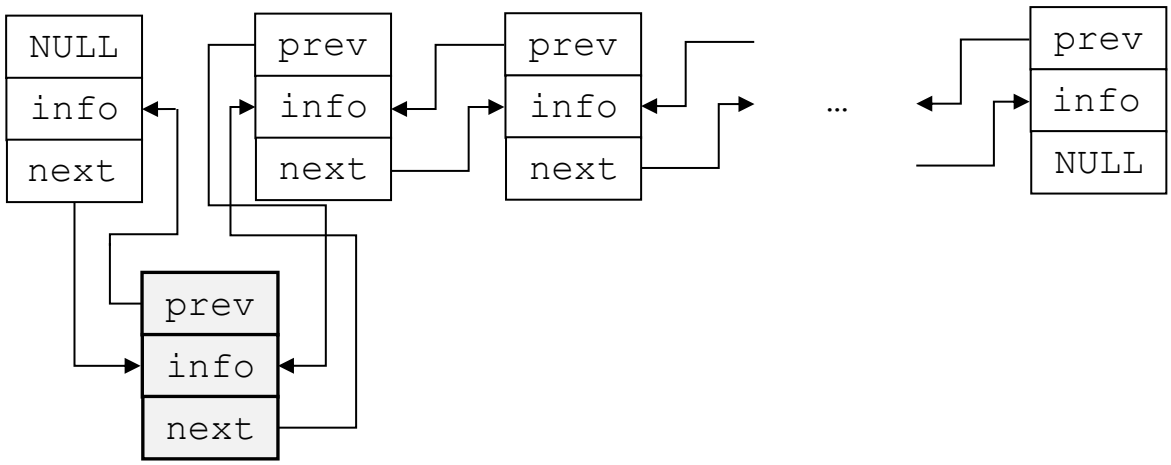


Рис. 2. Представление двунаправленного списка в памяти после добавления элемента во вторую позицию

Вывод элементов списка аналогичен выводу однонаправленного списка и имеет вид:

```
void PrintList(list * elem){
    while (elem->prev!= NULL)
     elem = elem->prev;
    while (elem != NULL){
        cout << elem ->info << '\t';
        elem = elem ->next;
    }
}
```

Изменение элемента в заданной позиции, если такая существует, имеет вид:

```
list *EditByPos(list *elem, int data, int pos){
        while (elem->prev!= NULL)//ищем первый
         elem = elem->prev;          //эл-т списка
        for (int i=1; elem!=NULL && i<pos; i++)
            elem = elem ->next;
        if (elem == NULL)
            return NULL;
        elem ->info = data;
        return elem;
    }
```

Функция возвращает либо указатель на измененный элемент в указанной позиции, либо NULL, если элемента в заданной позиции не существует.

Удаление элемента в заданной позиции реализуется посредством замыкания полей–указателей соседних элементов по отношению к удаляемому узлу друг на друга. Код функции, реализующей удаление:

```
list *DeleteByPos(list *elem, int pos) {
    if (elem==NULL) return NULL; //список пуст
    while (elem->prev!= NULL)
        elem = elem->prev;
    list *tmp=elem; //запоминаем первый эл-т
    for (int i=1; elem!=NULL && i<pos; i++)
        elem = elem ->next;
    if (elem==NULL) // нет эл-та в позиции pos
        return tmp;
    if (elem->prev ==NULL){ //удаляется первый
        tmp=elem;
        elem = elem->next;
        if (elem !=NULL)
            elem->prev =NULL;
        delete tmp;
        return elem;
    }
```

```
        if   (elem->next   ==NULL){   //удаляется
последний
            tmp=elem;
            elem = elem->prev;
            if (elem !=NULL)
                elem->next =NULL;
            delete tmp;
            return elem;
        }
        tmp=elem; //удаляется «внутренний» эл-т
        elem->prev->next = elem->next;
        elem->next->prev = elem->prev;
        elem = elem->prev;
        delete tmp;
        return elem;
    }
```

В данной функции различаются случаи: список пуст; указана неверная позиция удаления; удаляется первый элемент списка; удаляется последний элемент списка; удаляется «внутренний» элемент списка. Результатом работы функции `DeleteByPos` является либо указатель на существующий элемент списка, либо `NULL`.

Удаление линейного двунаправленного списка может быть организовано аналогично удалению однонаправленного списка и имеет вид:

```
    void DeleteList(list *elem) {
        if (elem==NULL) return;
        while (elem->prev!= NULL)
            elem = elem->prev;
        while (elem!= NULL){
            list *tmp=elem;
            elem = elem->next;
            delete tmp;
        }
```

Обработка информации с применением двунаправленного списка имеет преимущество по сравнению с использованием

однонаправленного списка, которое заключается в возможности перемещения по списку в обоих направлениях.

Пример 1. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Удалить из списка все элементы, значения информационного поля которых меньше среднего арифметического значения всех элементов списка. Вывести результирующий список в обратном порядке.

```
#include <iostream>
using namespace std;

struct list {
    int info;
    list *prev;
    list *next;
};

list *AddToEnd(list *elem, int data) {
    if (elem == NULL) {
        elem = new list;
        elem->info = data;
        elem->prev = NULL;
        elem->next = NULL;
        return elem;
    }
    while (elem->next != NULL)
        elem = elem->next;
    list *p = new list;
    p->info = data;
    p->prev = elem;
    p->next = NULL;
    elem->next = p;
    return p;
}

void PrintList(list * elem) {
    while (elem->prev != NULL)
        elem = elem->prev;
    while (elem != NULL) {
```

```
            cout << elem->info << '\t';
            elem = elem->next;
        }
    }

    void PrintReverseList(list * elem) {
        while (elem->next != NULL)
            elem = elem->next;
        while (elem != NULL) {
            cout << elem->info << '\t';
            elem = elem->prev;
        }
    }

    float AvrageList(list *elem) {
        if (elem == NULL) return NAN;
        while (elem->prev != NULL)
            elem = elem->prev;
        int s,k;
        for(s=0,k=0;elem!=NULL;elem=elem->next){
            s += elem->info;
            ++k;
        }
        return float(s)/k;
    }

    list *DeleteByPointer(list *elem) {
        if (elem == NULL) return elem;
        if   (elem->next==NULL)   {   //последний
элемент
            list *tmp = elem;
            elem = elem->prev;
            elem->next = NULL;
            delete tmp;
            return elem;
        }
        list *tmp = elem;
        elem = elem->next;
        elem->prev = tmp->prev;
        if (elem->prev != NULL)
```

```
            elem->prev->next = elem;
        delete tmp;
        return elem;
    }

    list *DeleteLessAvrageList(list *elem) {
        if (elem == NULL) return NULL;
        float avg = AvrageList(elem);
        while (elem->prev != NULL)
            elem = elem->prev;
        list *tmp = elem;
        while (tmp != NULL) {
            if (tmp->info < avg) {
                if (elem == tmp)
                    elem=tmp                                    =
DeleteByPointer(tmp);
                else
                    tmp = DeleteByPointer(tmp);
            }
            else
                tmp = tmp->next;
        }
        return elem;
    }

    void DeleteList(list *elem) {
        if (elem == NULL) return;
        while (elem->prev != NULL)
            elem = elem->prev;
        while (elem != NULL) {
            list *tmp = elem;
            elem = elem->next;
            delete tmp;
        }
    }

    int main() {
        setlocale(LC_ALL, "RUSSIAN");
        list *E=NULL;
```

```
        cout << "Введите список(признак окончания
0):" << endl;
        int x;
        for (cin >> x; x; cin >> x) {
            E = AddToEnd(E, x);
        }
        cout << endl;
        PrintList(E);
        cout << endl;
        E = DeleteLessAvrageList(E);
        cout << "Результат:\n";
        PrintReverseList(E);
        cout << endl;
        system("pause");
        return 0;
        DeleteList(E);
    }
```

В добавление к описанным функциям в данном примере заданы `PrintReverseList`, `AvrageList`, `DeleteByPointer`, `DeleteLessAvrageList`.

Функция `PrintReverseList` выводит элементы списка в обратном порядке. Так как `elem` указывает на произвольный элемент списка, то вначале ищется последний элемент списка, а затем (второй цикл) перебираются все элементы от последнего до первого и выводится на экран их информационное поле.

Функция `AvrageList` возвращает среднее арифметическое информационных полей элементов списка.

Функция `DeleteByPointer` удаляет элемент списка, на который указывает значение параметра `elem`, и возвращает указатель на элемент, который стоял в списке за удаляемым. Если `elem` указывал на последний элемент списка, то возвращается указатель элемент, стоящий в списке перед удаляемым.

Функция `DeleteLessAvrageList` просматривает все элементы списка с первого до последнего и удаляет все его элементы, меньшие среднего арифметического. Функция возвращает указатель на первый неудаленный элемент списка.

Пример 2. Сформировать двунаправленный список различных целых чисел, отсортированный по возрастанию. Признак

завершения ввода — 0. Определить функции добавления и удаления значения (замечание: повторяющиеся элементы в список не добавляются).

```
#include <iostream>
using namespace std;

struct list {
    int info;
    list *prev;
    list *next;
};

list *AddToBegin(list *elem, int data) {
    if (elem == NULL) {
        elem = new list;
        elem->info = data;
        elem->prev = NULL;
        elem->next = NULL;
        return elem;
    }
    while (elem->prev != NULL)//ищем первый
        elem = elem->prev;       //эл-т списка
    list *p = new list;
    p->info = data;
    p->prev = NULL;
    p->next = elem;
    elem->prev = p;
    elem = p;
    return p;
}

list *InsertAfter(list *elem, int data) {
    if (elem == NULL) {
        elem = new list;
        elem->info = data;
        elem->prev = NULL;
        elem->next = NULL;
        return elem;
    }
    list *p = new list;
```

```
        p->info = data;
        p->prev = elem;
        p->next = elem->next;
        elem->next = p;
        if (p->next != NULL)
            p->next->prev = p;
        return p;
    }

    list *AddSortByValue(list *elem, int data) {
        if (elem == NULL) { //список пуст
            elem = new list;
            elem->info = data;
            elem->prev = NULL;
            elem->next = NULL;
            return elem;
        }

        while(elem->info<data && elem->next!=
NULL && elem->next->info <= data)
            elem = elem->next; /*просматриваем в
сторону возрастания*/

        while(elem->info>data && elem-
>prev!=NULL)
            elem = elem->prev; /*просматриваем в
сторону убывания*/

        if (elem->info == data)
            return elem;
        if (elem->info > data)
            return AddToBegin(elem, data);
        else
            return InsertAfter(elem, data);
    }

    void PrintList(list * elem) {
        while (elem->prev != NULL)
            elem = elem->prev;
        while (elem != NULL) {
```

```
            cout << elem->info << '\t';
            elem = elem->next;
        }
    }

    list *DeleteByPointer(list *elem) {
        if (elem == NULL) return elem;
        if(elem->next == NULL){//последний
элемент
            list *tmp = elem;
            elem = elem->prev;
            elem->next = NULL;
            delete tmp;
            return elem;
        }
        list *tmp = elem;
        elem = elem->next;
        elem->prev = tmp->prev;
        if (elem->prev != NULL)
            elem->prev->next = elem;
        delete tmp;
        return elem;
    }

    list *DeleteByValue(list *elem, int data) {
        if (elem == NULL) return elem;
        while (elem->prev != NULL)//ищем первый
            elem = elem->prev;    //эл-т списка

        for(list *tmp=elem; tmp!=NULL;
tmp=tmp->next)
            if (tmp->info == data)
                return DeleteByPointer(tmp);
        return elem;
    }

    void DeleteList(list *elem) {
        if (elem == NULL) return;
        while (elem->prev != NULL)
            elem = elem->prev;
```

```
        while (elem != NULL) {
            list *tmp = elem;
            elem = elem->next;
            delete tmp;
        }
    }

    int main() {
        setlocale(LC_ALL, "RUSSIAN");
        list *E=NULL;
        cout << "Введите список(признак окончания
0):" << endl;
        int x;
        for (cin >> x; x; cin >> x) {
            E = AddSortByValue(E, x);
        }
        cout << endl;
        PrintList(E);
        cout << endl;
        cout << "Введите удаляемое значение:";
        cin >> x;
        E = DeleteByValue(E,x);
        cout << "Результат:\n";
        PrintList(E);
        cout << endl;
        system("pause");
        return 0;
        DeleteList(E);
    }
```

В примере дополнительно описаны функции `InsertAfter`, `AddSortByValue`, `DeleteByValue`.

Функция `InsertAfter` создает новый элемент списка с информационным значением, равным `data`, и устанавливает его в списке после элемента, на который указывает `elem`. Функция возвращает указатель на созданный элемент.

Функция `AddSortByValue` добавляет в отсортированный список элемент с информационным полем, равным `data`. Прежде всего проверяется список на пустоту, и если он пуст, то создается

первый элемент списка и возвращается указатель на него. Для непустого отсортированного списка ищется указатель на элемент, после которого нужно добавить новый. Если найти таковой не удается, то новый элемент создается как первый элемент списка. Функция возвращает указатель на созданный элемент.

Функция `DeleteByValue` просматривает список с первого до последнего элемента, и если найдется элемент с информационным полем, равным `data`, то он удаляется. Функция возвращает указатель на любой неудаленный элемент или `NULL`.

Контрольные вопросы и задания

1. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Найти значение минимального элемента в списке.

2. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Найти значение среднего арифметического нечетных отрицательных элементов.

3. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Найти количество элементов, значения которых больше заданного.

4. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Удалить из списка все максимальные элементы.

5. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Найти количество повторяющихся элементов.

6. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Найти количество неповторяющихся четных элементов.

7. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Расположить отрицательные элементы вначале списка с сохранением их исходного порядка ввода.

8. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Упорядочить элементы списка по убыванию.

9. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Проходя список слева направо, после каждого второго элемента добавить элемент, значение которого есть разность двух предыдущих элементов.

10. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Удалить из списка все элементы с нечетными значениями.

11. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Определить, является ли заданная последовательность симметричной.

12. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Оставить в списке все элементы с четной суммой цифр.

13. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Проходя список справа налево, после каждого элемента добавить элемент, значение которого есть произведение предшествующего элемента и минимального элемента в списке.

14. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Удалить из списка все элементы, содержащие цифру 0.

15. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Элементы, меньшие среднего арифметического значений элементов списка, расположить в конце списка с сохранением исходного порядка.

16. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Положительные элементы списка расположить в начале списка.

17. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Проходя список слева направо, удалить из списка повторяющиеся элементы.

18. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Проходя список слева направо, найти количество элементов, кратных трем. Двигаясь в обратном направлении, удалить из списка элементы, кратные шести.

19. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Удалить из списка элементы с одинаковыми значениями и равноудаленные от концов списка.

20. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Удалить из списка нечетные элементы со значениями, большими минимального среди кратных 8.

4. КОЛЬЦЕВЫЕ СПИСКИ

Разновидностью рассмотренных видов связных списков является кольцевой список, который может быть организован на основе как однонаправленного, так и двунаправленного списков:

1. В однонаправленном списке указатель последнего элемента должен указывать на первый элемент этого же списка. Графически линейный список можно представить так:

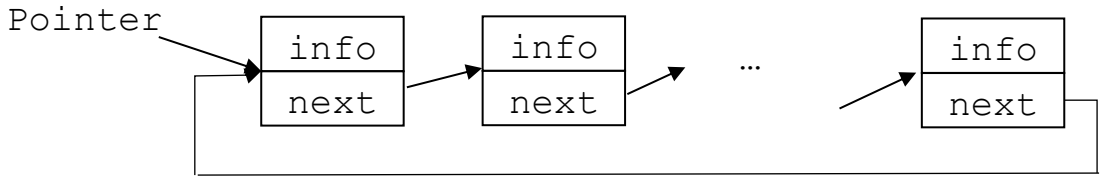


Рис. 4.1. Структура кольцевого однонаправленного списка

2. В двунаправленном списке в первом и последнем элементах соответствующие указатели переопределяются, как показано на рис. 4.2.

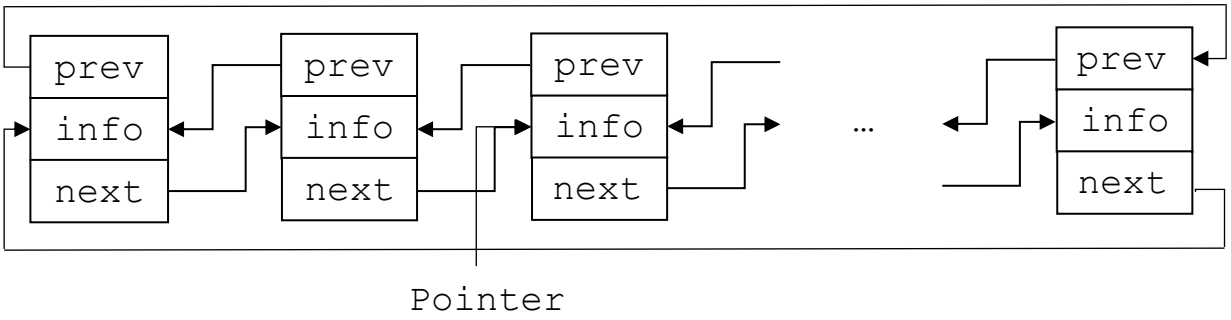


Рис. 4.2. Структура кольцевого двунаправленного списка

Базовая статическая структура, описывающая элемент кольцевого списка, аналогична структуре, рассмотренной ранее при описании однонаправленного или двунаправленного списка.

При работе с такими списками несколько упрощаются некоторые процедуры. Просмотр такого списка способом, описанным ранее для линейных списков, ввиду отсутствия указателей со значением `NULL` влечет попадание в бесконечный цикл. В кольцевом списке нет понятия головы. Кольцевой список характеризуется указателем на любой его элемент. Если в списке один элемент, то его поля указателей содержат его же адрес.

Поэтому кольцевой список следует просматривать, начиная от заданного начального элемента списка и до ему предшествующего.

Например, вывод кольцевого списка может иметь вид:

```
void Output(list *Pointer) {
    if (Pointer == NULL)
        return;
    list *tmp = Pointer;
    cout << "Элементы: ";
    while (tmp->next != Pointer) {
        cout << tmp->info << " ";
        tmp = tmp->next;
    }
    cout << tmp->info << endl;
}
```

В данном примере показана функция `Output` с единственным параметром – указателем на элемент, с которого начинается просмотр. Для обхода списка организуется цикл с предусловием, которое обеспечивает продвижение по списку, пока следующий элемент не является заданным. Подобным же образом следует организовывать просмотр кольцевого списка при обработке его данных.

Остальные операции – вставка / модификация / удаление узла в заданной позиции – выполняются по аналогии с соответствующим линейным списком с ориентацией на заданную позицию `pos` посредством параметрического цикла.

Пример 1. Дана последовательность целых чисел. Признак завершения ввода — 0. Определить однонаправленный кольцевой список для хранения последовательности. Отсортировать элементы списка по возрастанию.

```
#include <iostream>

struct ring{
    int info;
    ring *next;
};
```

```
    void InsertAfter(ring *&p,int data) {
    /* p указывает на элемент, после которого
добавляется*/
        if (p == NULL) {
            p = new ring;
            p->info = data;
            p->next = p;
        }
        else {
            ring *r = new ring;
            r->info = data;
            r->next = p->next;
            p->next = r;
            p = r;
        }
    }

    void DestroyList(ring *p) {
        if (p == NULL) return;
        while (p != p->next) {
            ring *r = p->next;
            p->next = r->next;
            delete r;
        }
        delete p;
    }

    void SwapNeighbors(ring *p) {
    /*обменивается текущий и следующий за ним*/
        if (p == NULL) return;
        ring *r = p;
        while (r->next != p) {
            r = r->next; //ищем элемент перед p
        }
        r->next = p->next;
        p->next = p->next->next;
        r->next->next = p;
    }

    void SwapNeighborsByValue(ring *p) {
```

```
/*обмениваются значения текущего и следующего за
ним*/
        if (p == NULL) return;
        int x;
        x = p->info;
        p->info = p->next->info;
        p->next->info = x;
    }

    void Output(ring *p) {/*p – указывает на
    первый выводимый элемент списка*/
        ring *r = p;
        while (r->next != p) {
            std::cout << r->info << '\t';
            r = r->next;
        }
        std::cout << r->info << '\t';
    }

    void Sort(ring *&p) {
        if (p == NULL) return;
        ring *min = p;
        for(ring *r= p->next; r != p; r = r-
>next)
            if (r->info < min->info)
                min = r;
        p = min;/* p указывает на элемент с
минимальным значением */

        bool wasSwap = true;
        while (wasSwap) {
            ring *r = p;
            wasSwap = false;
            while (r->next != p) {
                if (r->info > r->next->info) {
                    wasSwap = true;
                    SwapNeighbors(r);
                }
                else
                 r = r->next;
```

```
            }
        }
    }

    int main() {
        setlocale(LC_ALL, "rus");
        ring *elem = NULL;
        int x;
        std::cout << "Введите элементы (признак
завершения 0)\n";
        std::cin >> x;
        while (x != 0) {
            InsertAfter(elem,x);
            std::cin >> x;
        }
        std::cout<<"Список: \n" ;
        if (elem!=NULL)
            Output(elem->next);
        std::cout << "\n";

        Sort(elem);

        std::cout << "Новый список: \n";
        if (elem != NULL)
            Output(elem);
        std::cout << "\n";
        system("pause");
        DestroyList(elem);
        return 0;
    }
```

Функция `InsertAfter` в качестве параметров получает ссылку `p` на указатель на элемент, после которого добавляется (вставляется) новый элемент и значение `data` – информационное поле создаваемого элемента. После выполнения функции `p` указывает на созданный элемент.

Функция `DestroyList` освобождает память, выделенную для хранения кольцевого списка.

Функция SwapNeighbors обменивает местами в кольцевом списке элемент, на который указывает p и следующий за ним.

Функция SwapNeighborsByValue приведена для сравнения с предыдущей, не обменивает местами в элементы, а обменивает значения их информационных полей (по аналогии обмена в массивах).

Функция Output выводит на экран элементы кольцевого списка, начиная с данного в качестве параметра.

Функция Sort сортирует заданный кольцевой однонаправленный список. В качестве параметра p задается ссылка на указатель произвольного элемента списка. После выполнения тела функции p указывает на элемент списка с минимальным значением. Сортировка производится циклично. На каждой итерации цикла осуществляется полный обход кольцевого списка и сравниваются два соседних элемента. В случае необходимости производится обмен местами соседних элементов и отмечается, что хотя бы один раз обмен был выполнен (wasSwap=true). Цикл останавливается, если на очередной итерации не было проведено ни одного обмена.

Пример 2. Реализовать стек на основе двунаправленного списка. Определить операции добавления элемента в стек, извлечение верхушки стека и опустошения стека.

```
#include <iostream>
using namespace std;

struct stack{
    int info;
    stack *prev, *next;
};

void Push(stack *&Top, int data) {
    if (Top == NULL) {
        Top = new stack;
        Top->info = data;
        Top->prev = Top;
        Top->next = Top;
    }
    else {
```

```
            stack *p = new stack;
            p->info = data;
            p->next = Top;
            p->prev = Top->prev;
            Top = Top->prev = Top->prev->next =
p;
        }
    }

    int Pop(stack *&Top) {
        if (Top == NULL) {
            return NAN;
        }
        else {
            int k= Top->info;
            if (Top->next == Top)
                Top=NULL;
            else {
                stack *p = Top;
                Top = Top->next;
                p->prev->next = Top;
                Top->prev = p->prev;
                delete p;
                return k;
            }
        }
    }

    void Output(stack *Top) {
        if (Top == NULL) cout << "Стек пуст" <<
endl;
        else {
            stack *tmp = Top;
            cout << "Элементы: ";
            while (tmp->next != Top) {
                cout << tmp->info << " ";
                tmp = tmp->next;  // Продвижение
по стеку вглубь
            }
            cout << tmp->info << endl;
```

```
        }
    }

    void Destroy(stack *PElement) {
        while (PElement != NULL)
            Pop(PElement);
    }

    void main() {
        setlocale(LC_ALL, "RUS");
        int i = 1;
        stack *TopStack = NULL;
        while (i) {
            cout << "\nВведите команду :\n\n > 1
- Загрузка данных в стек\n > 2 - Извлечение
данных из стека\n > 3 - Очистка стека\n > 4 -
Вывод содержимого стека\n > 0 - Выход из
меню\n\n";
            cin >> i;
            cout << endl;
            switch (i) {
            case 0: { break; }
            case 1: {
                cout << "Ведите элемент данных:
";
                int x;
                cin >> x;
                Push(TopStack, x);
                break;
            }
            case 2: {
                if (TopStack == NULL)
                    cout << "Стек пуст";
                else
                    cout <<"Извлеченное значение:"
<< Pop(TopStack) << endl;
            }
            case 3: {
                Destroy(TopStack);
                TopStack = NULL;
```

```
                break;
            }
            case 4: {
                Output(TopStack);
                break;
            }
            default: { cout << "Ошибка: неверная
команда"; }
            }
        }
        Destroy(TopStack);
    }
```

В функции `main` реализовано стандартное текстовое меню на основе цикла `while`. В теле цикла осуществляется вывод меню на консоль. Для осуществления выбора введена целочисленная переменная `i`, которая инициализируется посредством потокового ввода с клавиатуры пользователем номера выбранного пункта. В соответствии с введенным значением вычисления направляются в один из блоков оператора `switch`.

Блок 0 обеспечивает выход из цикла и завершение программы.

Блок 1 запрашивает ввод с клавиатуры элемента данных для загрузки в стек, после чего вызывается функция `Push`, в которую введенный элемент передается в качестве аргумента.

Блок 2 вызывает функцию `Pop` для извлечения элемента данных из верхушки стека.

Блок 3 вызывает функцию `Destroy` для очистки стека путем последовательного извлечения верхушки стека.

Блок 4 вызывает функцию `Output` для вывода всех элементов данных, содержащихся в стеке, без его разрушения (очистки).

После выполнения указанного пользователем блока цикл повторяется снова и пользователю снова предлагается меню, чтобы сделать новый выбор.

Если пользователь вводит число, не принадлежащее интервалу 0…4, то он получает сообщение о том, что такой команды не существует, и цикл повторяется снова.

Все функции для работы со стеком по сути являются модификациями рассмотренных функций для работы с линейным двунаправленным списком. Основное отличие заключается в том,

что ни один из указателей базовой структуры – элемента списка никогда не получает значение NULL. Даже при инициализации головы стека в функции Push оба указателя prev и next получают значения ссылки на вершину. Впоследствии это свойство кольцевого списка используется в функции Destroy для идентификации факта извлечения последнего элемента: Top->next==Top.

Реализацию вывода данных из стека с его разрушением можно реализовать на базе функции Output, после вызова которой обнулить указатель на голову стека. В программе функция Destroy реализована на базе функции Pop, чем достигается полная корректность извлечения данных из стека, несмотря на значительные вычислительные издержки по сравнению с использованием варианта на базе функции Output.

Также следует отметить, что параметр Top функций Push и Pop является ссылкой на указатель, чем обеспечивается его изменяемость значения и корректность дальнейшей работы со стеком.

Контрольные вопросы и задания

1. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать однонаправленный кольцевой список на основе данной последовательности. Найти значение минимального элемента в списке.
2. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный кольцевой список на основе данной последовательности. Найти значение среднего арифметического нечетных отрицательных элементов.
3. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать однонаправленный кольцевой список на основе данной последовательности. Найти количество элементов, значения которых больше заданного.
4. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный кольцевой список на основе данной последовательности. Удалить из списка все повторяющиеся элементы.
5. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать однонаправленный кольцевой список на

основе данной последовательности. Найти количество повторяющихся элементов.

6. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать однонаправленный кольцевой список на основе данной последовательности. Найти количество неповторяющихся четных элементов.

7. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный кольцевой список на основе данной последовательности. Расположить отрицательные элементы вначале списка с сохранением их исходного порядка ввода.

8. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать однонаправленный кольцевой список на основе данной последовательности. Упорядочить элементы списка по убыванию.

9. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать однонаправленный кольцевой список на основе данной последовательности. Последовательно удалить элементы из списка так, чтобы в нем не было элементов, равных разности двух соседних элементов.

10. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный кольцевой список на основе данной последовательности. Удалить из списка все элементы с нечетными значениями.

11. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный список на основе данной последовательности. Определить, есть ли в списке два равных элемента, расположенных на диаметре кольца.

12. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать однонаправленный кольцевой список на основе данной последовательности. Определить, можно ли разделить список на две части с равным количеством элементов таких, что суммы их элементов равны.

13. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать однонаправленный кольцевой список на основе данной последовательности. Найти в списке наиболее длинную возрастающую подпоследовательность.

14. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный кольцевой список на основе

данной последовательности. Найти в списке наиболее длинную симметричную подпоследовательность.

15. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать однонаправленный кольцевой список на основе данной последовательности. Найти в списке наиболее длинную подпоследовательность, элементы которой меньше среднего арифметического значений всех элементов списка.

16. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный кольцевой список на основе данной последовательности. Сформировать однонаправленный список отсортированных четных чисел из элементов двунаправленного списка.

17. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать однонаправленный кольцевой список на основе данной последовательности. Найти в списке количество симметричных подпоследовательностей заданной длины.

18. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный кольцевой список на основе данной последовательности. Найти в списке наиболее длинную подпоследовательность, состоящую из элементов с нечетной суммой цифр.

19. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать однонаправленный кольцевой список на основе данной последовательности. Удалить из списка все максимальные по длине не пересекающиеся (слева направо) подпоследовательности, состоящие из нечетных чисел.

20. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать двунаправленный кольцевой список на основе данной последовательности. Удалить из списка нечетные элементы со значениями, большими минимального среди кратных 8, встречающиеся не более трех раз.

5. ДВОИЧНЫЕ ДЕРЕВЬЯ

Двоичное (бинарное) дерево — это иерархическая структура данных, в которой каждый элемент (узел, вершина) имеет значение (информационное поле) и ссылки на левого и правого потомка. На рис. 5.1 вершина 1 имеет потомков (непосредственных потомков) 3 и 4. Вершина, не являющаяся чьим-либо потомком, называется корнем. Узлы, не имеющие потомков, называются листьями. На рис. 5.1 вершина 0 является корнем дерева, а вершины 7…14 – листьями.

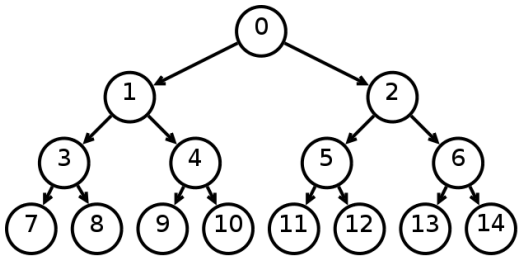


Рис. 5.1. Пример двоичного дерева

Если вершина B является потомком вершины A, то говорят, что A – предок (или родитель) B. В данном случае можно сказать, что A – непосредственный предок B. Отношение «предок» транзитивно, т.е. если A предок B и B предок C, то A предок C. На рис. 5.1 вершина 2 является непосредственным предком вершин 5 и 6, а также является предком для вершин 5, 6, 11, 12, 13, 14.

Если вершина A является непосредственным предком вершины B, то говорят, что существует ребро (A, B), соединяющее вершины A и B. В дереве существует путь между вершинами X_0 и X_n, если в дереве имеются ребра $(X_0, X_1), (X_1, X_2), \ldots, (X_{n-1}, X_n)$, и говорят, что данная последовательность ребер образует путь между вершинами X_0 и X_n. Количество ребер пути суть длина пути. В двоичных деревьях существуют только пути от предка к потомку.

Вершины, равноудаленные от корня, образуют ярус. Номер яруса определяется длиной пути от корня до любой вершины яруса. На
рис. 5.1 вершины 1 и 2 образуют первый ярус, а вершины 7…14 – третий.

Двоичное дерево поиска (ДДП) — это бинарное дерево, обладающее дополнительными свойствами: значение левого

потомка меньше значения предка, а значение правого потомка больше значения предка для каждого узла дерева. При каждой операции вставки нового или удаления существующего узла порядок дерева сохраняется.

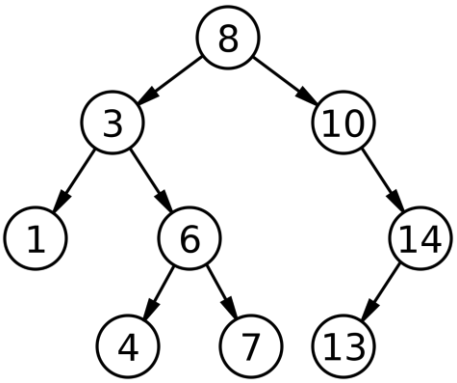


Рис. 5.2. Пример двоичного дерева поиска

Для моделирования деревьев определим тип:

```
    struct Tree{
        int info;  //информационное поле
        Tree *left; //указатель на левый потомок
        Tree *right; //указатель на правый
потомок
    };
```

Для данной модели листьями являются экземпляры, у которых оба поля `Left` и `Right` равны NULL.

Большинство функций, работающих с деревьями, рекурсивны, поскольку дерево по своей сути является рекурсивной структурой данных. Другими словами, каждое поддерево в свою очередь является деревом. Поэтому функции, работающие с деревьями, в большинстве являются рекурсивными. Существуют и нерекурсивные версии этих функций, но их код намного сложнее.

Составим процедуру, которая к уже имеющемуся ДДП добавляет новый узел, с информационной составляющей `data` так, что вновь образованное дерево также являлось ДДП.

```
    void Add(Tree *root, int data)
    {
```

```
    if (data  < root->info)
        if (root->left == NULL) {
            Tree *q;
            q = new(Tree);
            q->info = data;
            q->left = NULL;
            q->right = NULL;
            root->left = q;
        }
        else Add(root->left, data);
    else
        if (root->right == NULL) {
            Tree *q;
            q = new(Tree);
            q->info = data;
            q->left = NULL;
            q->right = NULL;
            root->right = q;
        }
        else Add(root->right, data);
}
```

В зависимости от траекторий выделяют два типа обхода: горизонтальный (в ширину, Breadth-first search, BFS) и вертикальный (в глубину, Depth-first search, DFS).

При горизонтальном обходе просмотр вершин дерева осуществляется по уровням (level-ordered) – вначале обрабатываются все узлы текущего уровня, после чего осуществляется переход на нижний уровень (рис. 5.3). Обход в ширину эффективно реализуется с помощью очереди.

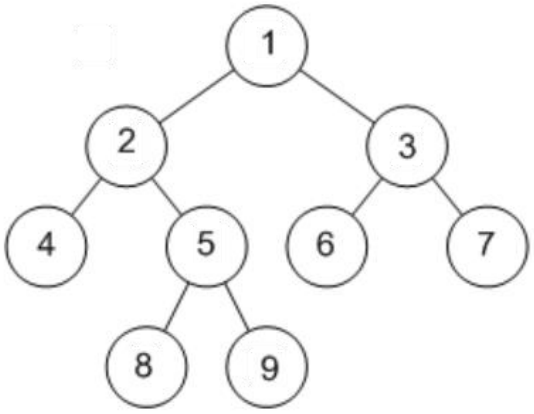


Рис. 5.3. Порядок обхода в ширину вершин двоичного дерева

При вертикальном обходе порядок обработки вершины и его правого и левого поддеревьев (потомков) следующий:

– прямой (префиксный, pre-ordered, NLR): вершина – левое поддерево – правое поддерево (рис. 5.4);

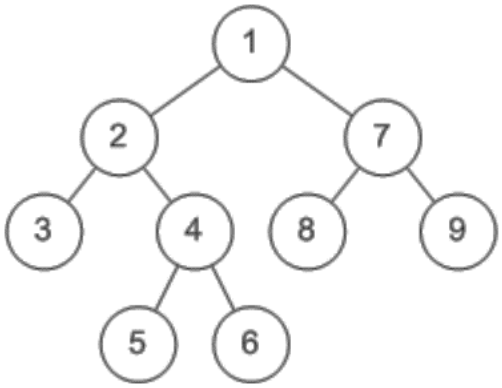


Рис. 5.4. Порядок прямого обхода вершин двоичного дерева

– центрированный (инфиксный, симметричный, in-ordered, LNR): левое поддерево – вершина – правое поддерево (рис. 5.5);

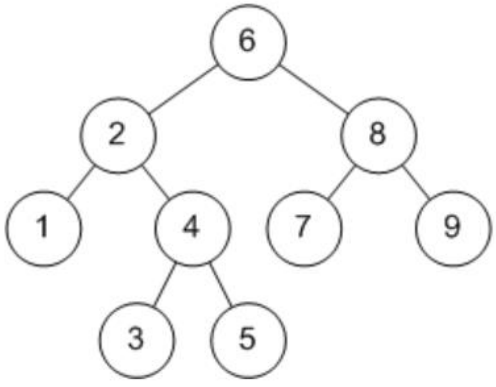


Рис. 5.5. Порядок центрированного обхода вершин двоичного дерева

– концевой (постфиксный, post-ordered, LRN): левое поддерево – правое поддерево – вершина (рис. 5.6).

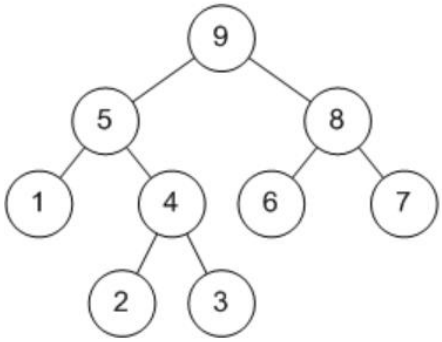


Рис. 5.6. Порядок концевого обхода вершин двоичного дерева

Алгоритмы обхода дерева в глубину рассмотрим на примере вывода значений информационных полей вершин.

Рекурсивная реализация прямого обхода вершин двоичного дерева имеет вид:

```
void DFS_NLR(Tree* root) {
    if (root!=NULL) {
        cout<<root->info<<' ';
        DFS_NLR (root->left);
        DFS_NLR (root->right);
    }
}
```

Рекурсивная реализация центрированного обхода вершин двоичного дерева имеет вид:

```
void DFS_LNR(Tree* root) {
    if (root!=NULL) {
        DFS_LNR(root->left);
        cout<<root->info<<' ';
        DFS_LNR(root->right);
    }
}
```

Рекурсивная реализация концевого обхода вершин двоичного дерева имеет вид:

```
void DFS_LRN (Tree* root) {
    if (root!=NULL) {
        DFS_ LRN (root->left);
        DFS_ LRN (root->right);
        cout<<root->info<<' ';
    }
}
```

Согласно данному определению дерева, все вершины дерева хранятся в динамической памяти, а следовательно, необходимо определить алгоритм ее освобождения. Здесь удобен концевой обход дерева. Реализация алгоритма удаления дерева имеет вид:

```
void Destroy(Tree *root) {
    if (root == NULL)
        return;
    Destroy(root->left);
    Destroy(root->right);
    delete root;
}
```

Пример 1. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Найти сумму листьев дерева.

```
#include <iostream>
using namespace std;

struct Tree {
    int info;
    Tree *left;
    Tree *right;
};

void Add(Tree *root, int data)
{
    if (data  < root->info)
        if (root->left == NULL) {
            Tree *q;
```

```
            q = new(Tree);
            q->info = data;
            q->left = NULL;
            q->right = NULL;
            root->left = q;
        }
        else Add(root->left, data);
    else
        if (root->right == NULL) {
            Tree *q;
            q = new(Tree);
            q->info = data;
            q->left = NULL;
            q->right = NULL;
            root->right = q;
        }
        else Add(root->right, data);
}

int Sum(Tree *root){
    if (root == NULL)
        return 0;
    if (root->left==NULL && root-
>right==NULL)
        return root->info;
    else
        return
         Sum(root->left)+Sum(root->right);
}

void Destroy(Tree *root) {
    if (root == NULL)
        return;
    Destroy(root->left);
    Destroy(root->right);
    delete root;
}

int main(){
    setlocale(LC_ALL, "RUSSIAN");
```

```
        Tree *root = NULL;
        int x;
        cout << "Введите элементы (признак
завершения 0)\n";
        cin >> x;
        if (x != 0) {
            root = new Tree;
            root->info = x;
            root->left = NULL;
            root->right = NULL;
        }
        for (cin >> x; x != 0; cin >> x)
            Add(root, x);

        cout << "Сумма листьев =" << Sum(root);
        system("pause");
        Destroy(root);
        return 0;
    }
```

Пример 2. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Используя алгоритм обхода дерева в ширину, найти количество четных чисел.

```
#include <iostream>
using namespace std;

struct Tree {
    int info;
    Tree *left;
    Tree *right;
};

void Add(Tree *root, int data)
{
    if (data  < root->info)
        if (root->left == NULL) {
            Tree *q;
```

```
            q = new(Tree);
            q->info = data;
            q->left = NULL;
            q->right = NULL;
            root->left = q;
        }
        else Add(root->left, data);
    else
        if (root->right == NULL) {
            Tree *q;
            q = new(Tree);
            q->info = data;
            q->left = NULL;
            q->right = NULL;
            root->right = q;
        }
        else Add(root->right, data);
}

void Destroy(Tree *root) {
    if (root == NULL)
        return;
    Destroy(root->left);
    Destroy(root->right);
    delete root;
}

/* определение очереди*/
struct queue {
    Tree *info;
    queue *next;
};

queue *CreateQueue() {
    return NULL;
}

bool IsEmptyQueue(queue *begin) {
    return begin == NULL;
}
```

```
void QueuePushBack(queue *&begin, Tree *data) {
    queue *r = new(queue);
    r->info = data;
    r->next = NULL;
    if (begin == NULL) {
        begin = r;
        return;
    }
    queue *p = begin;
    while (p->next != NULL) p = p->next; /*ищем
последний элемент очереди*/
    p->next = r;
}

Tree *QueuePop(queue *&begin) {
    if (!IsEmptyQueue(begin)) {
        queue *r = begin;
        Tree *x = r->info;
        begin = begin->next;
        delete r;
        return x;
    }
    return NULL;
}

void DeleteQueue(queue *&begin) {
    while (!IsEmptyQueue(begin)) {
        QueuePop(begin);
    }
}

int BFScount(Tree *root) {
    if (root == NULL) return 0;
    int sum = 0;
    queue *q = CreateQueue();
    QueuePushBack(q, root); //помещаем корень в
очередь
    while (!IsEmptyQueue(q)) {
```

```
        Tree *node = QueuePop(q); /*извлекаем
вершину из очереди*/
        if (node->left != NULL)
            QueuePushBack(q, node->left);
/*помещаем левого потомка в очередь*/
        if (node->right != NULL)
            QueuePushBack(q, node->right);
/*помещаем правого потомка в очередь*/
        if ((node->info & 1)== 0) /*проверяем
четность*/
            ++sum;
    }
    return sum;
}

int main(){
    setlocale(LC_ALL, "RUSSIAN");
    Tree *root = NULL;
    cout << "Введите элементы (признак завершения
0)\n";
    int x;
    cin >> x;
    if (x != 0) {
        root = new Tree;
        root->info = x;
        root->left = NULL;
        root->right = NULL;
    }
    for (cin >> x; x != 0; cin >> x)
        Add(root, x);
    cout << "Ответ =" << BFScount(root) << endl;
    system("pause");
    Destroy(root);
    return 0;
}
```

Пример 3. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Найти наибольший элемент на заданном уровне дерева.

Решение данной задачи подразумевает отслеживание уровня дерева, на котором находится вершина. Существует два подхода определения уровня вершины: определять уровень при обходе дерева и хранить уровень вершины.

Вариант программной реализации с подсчетом уровня вершины при обходе дерева в глубину имеет вид:

```
#include <iostream>
#include <algorithm>
using namespace std;

struct Tree {
    int info;
    Tree *left;
    Tree *right;
};

void Add(Tree *root, int data){
    if (data  < root->info)
        if (root->left == NULL) {
            Tree *q;
            q = new(Tree);
            q->info = data;
            q->left = NULL;
            q->right = NULL;
            root->left = q;
        }
        else Add(root->left, data);
    else
        if (root->right == NULL) {
            Tree *q;
            q = new(Tree);
            q->info = data;
            q->left = NULL;
            q->right = NULL;
            root->right = q;
        }
        else Add(root->right, data);
}
```

```
int DFSMaxByLevel(Tree *root, int currentLevel,
int Level){
    if (root == NULL)
        return -INFINITY;
    if (currentLevel == Level)
        return root->info;
    return max(
 DFSMaxByLevel(root->left,currentLevel+1,Level),
 DFSMaxByLevel(root-
>right,currentLevel+1,Level));
}

void Destroy(Tree *root){
    if (root == NULL)
        return;
    Destroy(root->left);
    Destroy(root->right);
    delete root;
}

int main() {
    setlocale(LC_ALL, "RUSSIAN");
    Tree *root = NULL;
    int x;
    cout << "Введите элементы (признак завершения
0)\n";
    cin >> x;
    if (x != 0) {
        root = new Tree;
        root->info = x;
        root->left = NULL;
        root->right = NULL;
    }
    for (cin >> x; x != 0; cin >> x)
        Add(root, x);

    cout << "Введите уровень: ";
    cin >> x;
    cout << "\nМаксимальный элемент уровня =" <<
DFSMaxByLevel(root,0,x);
```

```
    system("pause");
    Destroy(root);
    return 0;
}
```

Обратим внимание на функцию `DFSMaxByLevel`, она имеет три параметра: указатель на текущую вершину (корень текущего поддерева) `root`, уровень текущей вершины `currentLevel` и искомый уровень `Level`. Если уровень недостижим, то функция вернет значение «минус бесконечность» (`-INFINITY`). Если уровень достигнут в текущей вершине, то ее значение объявляется как максимум. Для любой вершины, уровень которой меньше заданного, ответ ищется как наибольшее значение (функция `max` из библиотеки `algorithm`) среди вершин искомого уровня ее левого и правого поддеревьев.

В случае, когда необходимо хранить значение уровня вершины дерева, следует модифицировать структуру данных, добавив еще одно поле – уровень дерева.

```
    struct Tree{
        int info;
        int level;
        Tree *left;
        Tree *right;
    };
```

Соответственно потребуется модификация функция добавления вершины в дерево.

Вариант программной реализации с обходом вершин дерева в глубину имеет вид:

```
#include <iostream>
#include <algorithm>
using namespace std;

struct Tree {
    int info;
    int level;
    Tree *left;
    Tree *right;
};
```

```
void Add(Tree *root, int data){
    if (data  < root->info)
        if (root->left == NULL) {
            Tree *q;
            q = new(Tree);
            q->info = data;
            q->level = root->level+1;
            q->left = NULL;
            q->right = NULL;
            root->left = q;
        }
        else Add(root->left, data);
    else
        if (root->right == NULL) {
            Tree *q;
            q = new(Tree);
            q->info = data;
            q->level = root->level+1;
            q->left = NULL;
            q->right = NULL;
            root->right = q;
        }
        else Add(root->right, data);
}

int DFSMaxByLevel(Tree *root,  int Level){
    if (root == NULL)
        return -INFINITY;
    if (root->level == Level)
        return root->info;
    return max(
        DFSMaxByLevel(root->left, Level),
        DFSMaxByLevel(root->right, Level));
}

void Destroy(Tree *root) {
    if (root == NULL)
        return;
    Destroy(root->left);
```

```
    Destroy(root->right);
    delete root;
}

int main() {
    setlocale(LC_ALL, "RUSSIAN");
    Tree *root = NULL;
    int x;
    cout << "Введите элементы (признак завершения
0)\n";
    cin >> x;
    if (x != 0) {
        root = new Tree;
        root->info = x;
        root->level = 0;
        root->left = NULL;
        root->right = NULL;
    }
    for (cin >> x; x != 0; cin >> x)
        Add(root, x);
    cout << "Введите уровень: ";
    cin >> x;
    cout << "\nМаксимальный элемент уровня =" <<
DFSMaxByLevel(root,x);
    system("pause");
    Destroy(root);
    return 0;
}
```

Контрольные вопросы и задания

1. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Написать функцию, подсчитывающую сумму элементов дерева.

2. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Написать функцию, которая находит наибольший элемент дерева.

3. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Написать функцию, которая находит наименьший четный элемент дерева.

4. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Заменить четные элементы дерева нулем.

5. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Написать функцию, которая выводит на экран элементы из всех листьев дерева.

6. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Написать функцию, которая определяет глубину (уровень) заданного элемента дерева, информационное поле которого равно заданному значению, и возвращает –1, если такого элемента нет.

7. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Написать функцию, которая находит глубину дерева.

8. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Написать функцию, которая по заданному числу `n` определяет количество всех вершин, находящихся на уровне с номером `n` в дереве.

9. Дан одномерный массив. Отсортировать массив, используя дерево двоичного поиска.

10. Дана последовательность символов. Определить частоту вхождения каждого из символов в последовательность, используя дерево двоичного поиска.

11. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Вывести количество вершин дерева, значение которых не кратно заданному числу `K`.

12. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Найти сумму значений всех четных вершин данного дерева.

13. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Найти количество вершин дерева, являющихся левыми потомками (корень дерева не учитывать).

14. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Найти количество листьев дерева.

15. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Найти сумму нечетных значений всех листьев дерева.

16. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Вывести количество листьев дерева, являющихся правыми потомками.

17. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Для каждого из уровней дерева, начиная с нулевого, вывести сумму вершин, находящихся на этом уровне.

18. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Для каждого из уровней дерева, начиная с нулевого, вывести количество четных значений вершин, находящихся на этом уровне.

19. Дана последовательность целых чисел, оканчивающаяся нулем, и число `N` (`N > 0`), не превосходящее количество чисел в последовательности. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Нумеруя вершины в инфиксном порядке, вывести значения всех вершин с порядковыми номерами от `1` до `N`.

20. Дана последовательность целых чисел, оканчивающаяся нулем, и число `N` (`N > 0`), не превосходящее количество чисел в

последовательности. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Нумеруя вершины в постфиксном порядке, вывести значения всех вершин с порядковыми номерами от N до максимального номера.

21. Дана последовательность целых чисел, оканчивающаяся нулем, и число N (N > 0), не превосходящее количество чисел в последовательности. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Нумеруя вершины в префиксном порядке, вывести значения всех вершин с порядковыми номерами от N1 до N2.

22. Дана последовательность целых чисел, оканчивающаяся нулем, и число L. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Используя любой из способов обхода дерева, вывести значения всех вершин уровня L, а также их количество (если дерево не содержит вершин уровня L, то вывести 0).

23. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Вывести максимальное значение и количество вершин дерева, имеющих это максимальное значение.

24. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Вывести минимальное из значений всех вершин дерева и количество листьев, имеющих это минимальное значение (данное количество может быть равно 0).

25. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Вывести максимальное значение внутренних вершин дерева (т. е. вершин, не являющихся листьями).

26. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Удвоить значение каждой положительной вершины дерева.

6. КОНТЕЙНЕРЫ

Стандартная библиотека шаблонов C++ (STL) содержит набор шаблонов контейнерных классов, алгоритмов и итераторов. Контейнерные классы (контейнеры) позволяют легко реализовывать наиболее популярные структуры данных, такие как вектора, списки, стеки, очереди.

Все контейнеры библиотеки STL делятся на два типа: последовательные и ассоциативные. Последовательные контейнеры обеспечивают хранение конечного количества однотипных объектов в виде непрерывной последовательности. К базовым последовательным контейнерам относятся векторы (`vector`) и двусторонние очереди (`deque`). Ассоциативные контейнеры обеспечивают быстрый доступ к данным по ключу. Они построены на основе сбалансированных деревьев. К ассоциативным контейнерам относят словари (`map`) и множества (`set`). Контейнер стек (`stack`) относится к специализированным последовательным контейнерам, которые реализованы на основе базовых.

Итератор – это объект, который позволяет последовательно перебирать элементы контейнерного класса без необходимости пользователю знать реализацию определённого контейнерного класса, переходя от одного элемента к другому.

Соответственно про итератор можно сказать, что это специальный объект, который позволяет получить доступ к элементам структуры данных, не раскрывая её внутреннего устройства, используя определённый абстрактный интерфейс.

В C++ в качестве интерфейса итераторов используется семантика указателей. Существенно то, что данный интерфейс един для всех контейнеров – пользователи работают именно с этим интерфейсом и ничего не знают о внутреннем устройстве контейнера, а значит, такие алгоритмы, как поиск максимального элемента, могут быть обобщены для массивов, связных списков и т.д.

Разные контейнеры STL по-разному реализуют доступ к своим элементам. У `vector` они расположены последовательно в памяти, у `list` они связаны через указатели и т.д. А итератор унифицирует эти различия за обобщенным значением указателя. Благодаря

итераторам можно использовать один и тот же алгоритм для разных контейнеров, не меняя при этом ни кода.

Итератор — это указатель на определённый элемент контейнерного класса с дополнительным набором перегруженных операторов для выполнения чётко определённых функций:

Оператор * возвращает элемент, на который в данный момент указывает итератор.

Оператор ++ перемещает итератор к следующему элементу контейнера. Большинство итераторов также предоставляют оператор -- для перехода к предыдущему элементу.

Операторы == и != используются для определения того, указывают ли два итератора на один и тот же элемент или нет. Для сравнения значений, на которые указывают два итератора, нужно сначала разименовать эти итераторы, а затем использовать оператор == или !=.

Оператор = присваивает итератору новую позицию (обычно начало или конец элементов контейнера). Чтобы присвоить значение элемента, на который указывает итератор, другому объекту, нужно сначала разименовать итератор, а затем использовать оператор = или использовать адресные переменные.

Каждый контейнерный класс имеет 4 основных метода для работы с оператором =:

`begin()` возвращает итератор, представляющий начало элементов контейнера.

`end()` возвращает итератор, представляющий элемент, который находится после последнего элемента в контейнере.

`cbegin()` возвращает константный (только для чтения) итератор, представляющий начало элементов контейнера.

`cend()` возвращает константный (только для чтения) итератор, представляющий элемент, который находится после последнего элемента в контейнере.

Заметим, что `end()` указывает не на последний элемент контейнера, а на элемент (область памяти), расположенный после последнего. Это сделано в целях упрощения использования циклов: цикл перебирает элементы до тех пор, пока итератор не достигнет `end()`.

Также все контейнеры предоставляют два типа итераторов:

`container::iterator` — итератор для чтения / записи;

container::const_iterator — итератор только для чтения.

Контейнер vector является аналогом динамического массива, память под который выделяется и освобождается автоматически по мере необходимости. Для использования контейнера vector в программу необходимо включить заголовочный файл <vector>.

```
#include <vector>
```

Далее можно воспользоваться одним из предложенных способов для создания экземпляра vector.

1. Создать пустой контейнер, используя конструкцию вида

```
vector < <тип_ элемента> > <имя>;
```

Например,

```
vector <int> v1;
vector <double> v2;
```

2. Создать контейнер заданного размера, используя конструкцию вида

```
vector < <тип_ элемента> > <имя>(<размер>);
```

Например, запись вида

```
vector <int> v1 (10);
```

выделяет память под 10 элементов типа int. В дальнейшем размер вектора может быть изменен, в отличие от обычных массивов.

3. Создать контейнер заданного размера и инициализировать его элементы заданным значением:

```
vector                                  <<тип_элемента>>
<имя>(<размер>,<значение>);
```

Например, запись

```
vector <int> v1 (10, 2);
```

выделяет память под 10 элементов типа int и инициализирует их двойками.

4. Создать контейнер заданного размера и инициализировать его различными значениями, аналогично массивам.

```
vector <int> v1 = {1, 2, 3, 4, 5, 6};
```

5. Создать контейнер и инициализировать его элементы значениями диапазона `[first, last)` элементов другого контейнера.

```
int arr[7] = { 15, 2, 19, -3, 28, 6, 8 };
vector<int> v1(arr, arr + 7);
```

6. Создать контейнер и инициализировать его элементы значениями элементов другого однотипного контейнера.

```
vector <int> v1;
// ... добавление элементов в v1
vector <int> v2(v1);
```

Для обращения к отдельному элементу вектора можно использовать скобки `[]`, так же, как и в обычном массиве. Другой способ обращение к отдельному элементу – метод `at(int p)`, который позволяет обратиться к элементу вектора, расположенному на позиции, заданной параметром. При этом проверяется, существует ли элемент с индексом `p`.

Нумерация элементов вектора начинается с 0.

Например,

```
vector <int> v (10);
v[0] = 3; /*обращение к элементу вектора без
проверки диапазона*/
```

```
v.at(1) = 4; /*обращение к элементу вектора с
проверкой диапазона*/
```

Еще одним способом обращения к отдельным элементам вектора является использования цикла `for`, основанного на диапазоне. Рассмотрим несколько примеров:

1. Доступ к элементам вектора по значению.

```
vector <int> v = {0, 10, 20, 30, 40, 50};
for (auto i : v)
    cout << i << ' ';
```

В этом случае переменная `i` имеет тип элементов вектора `v` (в данном случае `int`) и ей последовательно присваиваются значения элементов вектора. Изменение значения переменной `i` в теле цикла допустимо, но это не повлечет изменения значений элементов вектора. Следующий код эквивалентен приведенному:

```
vector <int> v = {0, 10, 20, 30, 40, 50};
for (int k=0; k<v.size(); ++k){
    int i=v[k];
    cout << i << ' ';
}
```

2. Доступ к элементам вектора по константной ссылке.

Следующие программные коды эквивалентные:

```
vector <int> v = {0, 10, 20, 30, 40, 50};
for (const int &i : v)
    cout << i << ' ';
```

и

```
vector <int> v = {0, 10, 20, 30, 40, 50};
for (const auto &i : v)
    cout << i << ' ';
```

Переменная `i` имеет ссылочный тип и последовательно ссылается на элементы вектора. В первом случае тип ссылки указан

в явном виде и должен совпадать с типом элементов вектора v (в данном случае int), а во втором – тип определяется компилятором. Изменение значения переменной i в теле цикла недопустимо.

3. Доступ к элементам вектора по ссылке.

Следующие программные коды эквивалентные:

```
vector <int> v = {0, 10, 20, 30, 40, 50};
for (int &i : v)
    cout << i << ' ';
```

и

```
vector <int> v = {0, 10, 20, 30, 40, 50};
for (auto &i : v)
    cout << i << ' ';
```

Переменная i имеет ссылочный тип и последовательно ссылается на элементы вектора. В первом случае тип ссылки указан в явном виде и должен совпадать с типом элементов вектора v (в данном случае int), а во втором – тип определяется компилятором. Изменение значения переменной i в теле цикла допустимо и любое ее изменение повлечет изменение значения соответствующего элемента вектора.

Класс vector содержит методы для получения двух видов итераторов: обычного и реверсивного.

Метод begin() возвращает прямой итератор на первый элемент вектора; метод end() возвращает прямой итератор на элемент вектора, следующий за последним; метод rbegin() возвращает обратный итератор на первый элемент вектора; метод rend() возвращает обратный итератор на элемент, следующий за последним.

Пример 1. Ввести 5 целых чисел и записать их в вектор. Вывести содержимое вектора в прямом и обратном порядке.

```
#include <iostream>
#include <vector>
using namespace std;
```

```
int main(){
    vector<int> myVector;
    for (int i = 0; i < 5; ++i){
        int x;
        cin>>x;
        myVector.push_back(x);
    }
    for(vector<int>::const_iterator
it=myVector.begin(); it != myVector.end(); ++it)
        cout << *it << " ";
    cout << '\n';
    for (vector<int>::reverse_iterator
it=myVector.rbegin(); it != myVector.rend();
++it)
        cout << *it << " ";
    cout << '\n';
    return 0;
}
```

Аналогичный программный код без использования итераторов:

```
#include <iostream>
#include <vector>
using namespace std;

int main(){
    vector<int> myVector;
    for (int i = 0; i < 5; ++i){
        int x;
        cin>>x;
        myVector.push_back(x);
    }
    for (int i = 0; i < 5; ++i)
        cout << v[i] << " ";
    cout << '\n';
    for (int i = 4; i > -1; --i)
        cout << v[i] << " ";
    cout << '\n';
    return 0;
```

```
}
```

Размер вектора – это количество элементов в контейнере, а объем – это размер памяти, выделенной для элементов контейнера. Если при вставке в вектор новых элементов его размер становится больше его объёма, происходит перераспределение памяти.

В классе `vector` определены следующие методы для работы с объемом и размером вектора:

`empty()` – возвращает `true`, если вектор пуст, и `false` в противном случае;

`size()` – возвращает количество элементов в векторе;

`max_size()` – озвращает максимально возможное количество элементов в векторе;

`reserve()` – устанавливает минимально возможное количество элементов в векторе;

`capacity()` – возвращает количество элементов, которое может содержать вектор до того, как ему потребуется выделить больше памяти.

Над векторами применимы все виды операций сравнения: ==, !=, <, <=, >, >= и прямого присваивания одному вектору значения другого.

Следующие методы позволяют модифицировать значения элементов вектора, поэтому их также называют методами-модификаторами:

`clear()` – очищает контейнер, т.е. удаляет все элементы. При этом размерность вектора остается без изменений;

`insert()` – вставляет элемент в произвольную позицию.

Существует несколько модификаций этого метода с разными списками параметров:

– вставить элемент, имеющий значение `value`, в позицию `it` (итератор)

```
<имя>.insert(<it>, <value>)
```

В результате выполнения следующего кода

```
vector<int> v = {0,1,2,3,4,5,6,7,8,9};
v.insert(v.begin()+2,100);
```

вектор v будет содержать следующие элементы:

```
 0 1 100 2 3 4 5 6 7 8 9
```

– вставить `count` элементов, имеющие значение `value`, начиная с позиции `it` (итератор)

```
<имя>.insert(<it>,<count>,<value>)
```

В результате выполнения следующего кода

```
vector<int> v = {0,1,2,3,4,5,6,7,8,9};
v.insert(v.begin()+2,5,200);
```

вектор v будет содержать следующие элементы:

```
0 1 200 200 200 200 200 2 3 4 5 6 7 8 9
```

– вставить несколько элементов из интервала `[first,last)`, имеющие значение `value`, начиная с позиции `it` (здесь `first`, `last`, `it` – итераторы)

```
<имя>.insert(<it>,<first>,<last>)
```

В результате выполнения следующего кода

```
vector<int> v = {0,1,2,3,4,5,6,7,8,9};
v.insert(v.begin()+3,v.begin()+5,v.begin()+9)
;
```

вектор v будет содержать следующие элементы:

```
0 1 2 5 6 7 8 3 4 5 6 7 8 9
```

`erase()` – удаляет элемент из произвольной позиции. Существует несколько модификаций этого метода с разными списками параметров:

– удалить элемент в позиции `it` (итератор)

```
<имя>.erase(<it>)
```

В результате выполнения следующего кода

```
vector<int> v = {0,1,2,3,4,5,6,7,8,9};
v.insert(v.begin()+3,v.begin()+5,v.begin()+9)
;
```

вектор v будет содержать следующие элементы:

```
 0 1 2 3 5 6 7 8 9
```

– удалить элемент s из интервала [first, last) (здесь first, last итераторы)

```
<имя>.erase(<first>, <last>)
```

В результате выполнения следующего кода

```
vector<int> v = {0,1,2,3,4,5,6,7,8,9};
v.erase(v.begin() + 4, v.begin() + 7);
```

вектор v будет содержать следующие элементы:

```
0 1 2 3 7 8 9
```

`push_back()` – добавляет элемент в конец;

`pop_back()` – удаляет последний элемент;

`resize()` – изменяет размер контейнера. Существует несколько модификаций этого метода с разными списками параметров:

– увеличить размер вектора до величины `count`, если текущий размер меньше, чем `count`. Если текущий размер больше `count`, контейнер сводится к размеру первых `count` элементов.

```
<имя>.resize(<count>)
```

– увеличить размер вектора до величины `count`, если текущий размер меньше, чем `count` и инициализирует новые элементы значением `value`. Если текущий размер больше `count`, контейнер сводится к размеру первых `count` элементов.

```
<имя>.resize(<count>, <value>)
```

`swap()` – обменивает содержимое двух векторов. Типы векторов при этом должны совпадать:

```
<вектор1>.swap(<вектор2>)
```

Пример 2. Дана последовательность целых чисел, оканчивающаяся нулем. Сформировать вектор на основе данной последовательности. Удалить из вектора все четные элементы. Вывести результирующий вектор в обратном порядке.

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> v ;
    setlocale(LC_ALL, "RUSSIAN");
    cout << "Введите список(признак окончания
0):" << endl;
    int x;
    for (cin >> x; x; cin >> x) {
        v.push_back(x);
    }
    cout << endl;
    for (int i=0; i<v.size();){
        if (v[i] % 2 == 0)
            v.erase(v.begin() + i);
        else
            ++i;
    }
    for (vector<int>::reverse_iterator
it=v.rbegin(); it != v.rend(); ++it)
        cout << *it << ' ';
```

```
    return 0;
}
```

Контейнер стека stack позволяет реализовать структуру данных, основанную на принципе LIFO (Last In First Out). Для работы с контейнером stack необходимо подключить соответствующую библиотеку:

```
#include <stack>
```

Объявить переменную типа stack можно с помощью команды вида

```
stack < <тип_ элемента> > <имя_стека>;
```

Например,

```
stack <int> st;
```

позволяет объявить стек st, состоящий из целых значений.

В контейнере stack есть предопределенные методы для работы со стеком. Рассмотрим эти методы:

push() позволяет добавить элемент на верхушку стека;

pop() позволяет извлечь элемент с верхушки стека, при этом сам элемент удаляется;

top() позволяет получить элемент, расположенный на верхушке стека, при этом сам элемент остается в стеке;

size() возвращает количество элементов в стеке;

empty() возвращает true, если стек пуст и false, если стек не пуст.

Пример 3. Вводится последовательность символов ')', '(', ']', '['. Определим правильную скобочную последовательность по следующему правилу:

1) пустая строка является правильной скобочной последовательностью;

2) если A — правильная скобочная последовательность, то (A) и [A] — правильные скобочные последовательности.

Определить, является ли введенная последовательность скобок правильной скобочной последовательностью.

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
    setlocale(LC_ALL, "rus");
    stack <char> st ;
    bool flag = true; /*флаг проверки, равен
false если хоть раз произошло нарушение*/
    for (char x=getchar(); x!='\n'; x=getchar()){
//считываем посимвольно до Enter
        switch (x) {
        case '(':case'[': {st.push(x); break; }
        case ')': {flag = flag && (st.top() ==
'('); st.pop(); break; }
        case ']': {flag = flag && (st.top() ==
'['); st.pop(); break; }
        default: flag = false;/*введен неверный
символ*/
        }
    }
    flag = flag && st.empty(); /* проверка не
остались ли еще открытые скобки*/
    if (flag) cout << "Да" << endl;
    else cout << "Нет" << endl;
    system("pause");
    return 0;
}
```

Контейнер очереди `queue` позволяет реализовать структуру данных, основанную на принципе FIFO (First In First Out). Для работы с контейнером queue необходимо подключить соответствующую библиотеку:

```
    #include <queue>
```

Объявить переменную типа queue можно с помощью команды вида

```
queue < <тип_ элемента> > <имя_очереди>;
```

Например,

```
queue <int> q;
```

позволяет объявить очередь q, состоящую из целых значений.

В контейнере queue предопределены следующие методы для работы с очередью:

push() позволяет добавить элемент в конец очереди;

pop() позволяет удалить элемент из начала очереди;

front() позволяет получить элемент, расположенный в начале очереди, при этом сам элемент остается в очереди;

back() позволяет получить элемент, расположенный в конце очереди, при этом сам элемент остается в очереди;

size() возвращает количество элементов в очереди;

empty() возвращает true, если очередь пуста и false, если очередь не пуста.

Пример 4. Дана последовательность целых чисел, оканчивающаяся нулем. Написать программную реализацию хранения данной последовательности в виде дерева двоичного поиска. Используя алгоритм обхода дерева в ширину, найти количество четных чисел.

```
#include <iostream>
#include <queue>
using namespace std;

struct Tree {
    int info;
    Tree *left;
    Tree *right;
};

void Add(Tree *root, int data)
{
```

```
    if (data  < root->info)
        if (root->left == NULL) {
            Tree *node;
            node = new(Tree);
            node->info = data;
            node->left = NULL;
            node->right = NULL;
            root->left = node;
        }
        else Add(root->left, data);
    else
        if (root->right == NULL) {
            Tree *node;
            node = new(Tree);
            node->info = data;
            node->left = NULL;
            node->right = NULL;
            root->right = node;
        }
        else Add(root->right, data);
}

void Destroy(Tree *root) {
    if (root == NULL)
        return;
    Destroy(root->left);
    Destroy(root->right);
    delete root;
}


int BFScount(Tree *root) {
    if (root == NULL) return 0;
    int counter = 0;
    queue <Tree *> q;
    q.push(root); //помещаем корень в очередь
    while (!q.empty()) {
        Tree *node = q.front();//извлекаем
вершину
        q.pop();
```

```
        if (node->left != NULL)
            q.push(node->left); /*помещаем левого
потомка в очередь*/
        if (node->right != NULL)
            q.push(node->right); /*помещаем
правого потомка в очередь*/
        if ((node->info & 1) == 0) /*проверяем
четность*/
            ++counter;
    }
    return counter;
}

int main() {
    setlocale(LC_ALL, "RUSSIAN");
    Tree *root = NULL;
    cout << "Введите элементы (признак завершения
0)\n";
    int x;
    cin >> x;
    if (x != 0) {
        root = new Tree;
        root->info = x;
        root->left = NULL;
        root->right = NULL;
    }
    for (cin >> x; x != 0; cin >> x)
        Add(root, x);
    cout << "Ответ =" << BFScount(root) << endl;
    system("pause");
    Destroy(root);
    return 0;
}
```

Контейнер дек deque представляет собой реализацию динамического массива, где добавление и извлечение элементов возможно с обеих сторон. Для использования дека необходимо подключить заголовочный файл <deque>:

```
#include <deque>
```

Способы создания объекта дек аналогичны способам создания вектора. Стандартная библиотека STL C++ предоставляет специальные средства работы с деком. Методы класса `deque` позволяют за константное время осуществлять вставку и удаление элементов. Методы контейнера `deque` следующие:

`front()` позволяет получить значения первого элемента;
`back()` позволяет получить значения последнего элемента;
`push_front()` добавление элемента в начало;
`push_back()` добавление элемента в конец;
`pop_front()` удаление первого элемента;
`pop_back()` удаление последнего элемента;
`size()` возвращает текущее количество элементов дека;
`clear()` очистка дека.

Пример 5. Дана последовательность целых чисел, оканчивающаяся нулем. Определить является ли данная последовательность симметричной.

```
#include <iostream>
#include <deque>
using namespace std;

int main() {
    setlocale(LC_ALL, "RUSSIAN");
    cout << "Введите элементы (признак завершения
0)\n";
    int x;
    deque <int> d;
    for (cin >> x; x != 0; cin >> x)
        d.push_back(x);
    while (!d.empty() && d.front() == d.back()) {
        d.pop_front();
        if (!d.empty())
            d.pop_back();
    }
    cout <<endl<< (d.empty()?"да" : "нет")
<<endl;
```

```
    cout << endl;
    system("pause");
    return 0;
}
```

Контейнер `set` представляет собой реализацию математического множества, которое состоит из различных элементов заданного типа и поддерживает операции добавления элемента, удаления элемента и проверки принадлежности элемента множеству. В множестве не может быть повторяющихся элементов.

Контейнер `set` реализован при помощи сбалансированного двоичного дерева поиска (красно-черного дерева), поэтому множества в STL хранятся в виде упорядоченной структуры, что позволяет перебирать элементы множества в порядке возрастания их значений. Сортировка элементов происходит автоматически при добавлении в множество.

Для использования контейнера `set` необходимо подключить заголовочный файл `<set>`

```
#include <set>
```

Для создания пустого множества можно воспользоваться конструкцией вида

```
set <type> <имя>;
```

Для работы с размером и объемом `set` определены следующие методы:

`empty()` возвращает `true`, если множество пусто и `false` в противном случае;

`size()` возвращает текущее количество элементов в множестве;

`max_size()` возвращает максимально возможное количество элементов в множестве;

Методы, которые позволяют модифицировать множество:

`clear()` очищает множество;

`insert()` позволяет добавить элементы;

`erase()` позволяет удалить элементы;

`swap()` обменивает содержимое двух множеств.

К наиболее важному аспекту работы с `set` относится поиск элементов, который осуществляется очень эффективно, поэтому `set` имеет несколько собственных методов поиска:

`count()` возвращает количество элементов, соответствующих определенному ключу;

`find()` находит элемент с конкретным ключом. Возвращает константный итератор позиции элемента или итератор `end()`, если таковой не найден;

`lower_bound()` возвращает итератор на первый элемент, который меньше, чем определенный ключ;

`upper_bound()` возвращает итератор на первый элемент, который больше, чем определенный ключ.

Пример 6. Дана последовательность целых чисел, оканчивающаяся нулем. Вывести в начале положительные значения, отсортированные по возрастанию, а затем – отрицательные, отсортированные по убыванию.

```
#include <iostream>
#include <set>
using namespace std;

int main() {
    setlocale(LC_ALL, "RUSSIAN");
    cout << "Введите элементы (признак завершения
0)\n";
    int x;
    set <int> positive,negative;
    for (cin >> x; x != 0; cin >> x)
        if (x > 0)
            positive.insert(x);
        else
            negative.insert(x);

    cout << "Ответ:" << endl;
```

```
    for (set <int>::iterator
it = positive.begin(); it != positive.end();
++it)
        cout << *it << ' ';

    for (set <int>::reverse_iterator
it=negative.rbegin(); it != negative.rend();
++it)
        cout << *it << ' ';
    cout << endl;
    system("pause");
    return 0;
}
```

Контейнер `map` (карта, словарь, ассоциативный массив) так же, как и множество, реализован на основе сбалансированного дерева двоичного поиска. Однако его отличием является то, что в словаре хранится набор элементов, состоящий из пар «ключ – значение», при этом ключи должны быть уникальны в пределах словаря и иметь упорядоченный тип.

Для работы с классом map необходимо подключить соответствующий заголовочный файл

```
#include <map>
```

Объекты класса `map` можно получить с помощью следующего описания:

```
map <<тип ключа>, <тип значения> > <имя>;
```

Например,

```
map<string, int> ar;
```

Для доступа к элементам словаря можно использовать `[]`, при этом в качестве индекса указывается значение ключа. Если же элемента с таким ключом нет в словаре, то он будет создан со значением, равным нулю. Второй способ получить доступ к

элементам словаря – использование метода `at()`, где в качестве параметра передается значение ключа. Помимо указанных способов доступа можно использовать итераторы для доступа к элементам словаря. С помощью итератора можно обратиться к ключу (`it->first`) и к значению (`it->second`).

Методы для работы со словарем являются такими же, как и методы для работы с множествами.

Пример 7. Дано n целых чисел (n<=1000000). Найти наиболее часто встречающееся число.

```
#include <iostream>
#include <map>
using namespace std;

int main() {
 setlocale(LC_ALL, "RUSSIAN");

 int n;
 cin >> n;
 map <long, long> m;

 for (long i = 0; i<n; ++i) {
      long x;
      cin >> x;
      ++m[x];
 }
 if (n == 0) {
      cout << "не найдено";
      return 0;
 }
 pair<long, long>max=make_pair(m.begin()-
>first,m.begin()->second);
 for (map <long, long>::iterator
it=m.begin(); it != m.end(); ++it) {
   if(max.second < it->second)
      max=make_pair(it->first,it->second);
 }
```

```
        cout << "значение "<<max.first<<
" встречается "<<max.second<<" раз(а)"<<endl;
    system("pause");
    return 0;
   }
```

В данном примере значение последовательности определяет ключ словаря `m`, а количество повторений – его значение.

Контрольные вопросы и задания

1. Что представляет собой контейнер?
2. Какие типы контейнеров вам известны и в чем их различия?
3. Что такое итератор?
4. Чем отличается метод `size()` от метода `capacity()`? Приведите пример.
5. По какому принципу строится работа со стеком?
6. Что такое дек?
7. Перечислите основные методы работы с контейнером deque.
8. В чем основное различие между контейнерами set и map?
9. Что обычно используют для последовательного доступа к элементам контейнера?
10. Какие типичные методы работы с контейнерами вам известны?
11. Перечислите известные вам способы доступа к элементу контейнера.
12. Дан целочисленный вектор размера N. Увеличить все четные числа, содержащиеся в векторе, на исходное значение первого четного числа. Если четные числа в векторе отсутствуют, то оставить вектор без изменений.
13. Дан вектор A размера N и целые числа K и L ($1 \le K < L \le N$). Переставить в обратном порядке элементы вектора, расположенные между элементами A[K] и A[L], не включая эти элементы.
14. Дан целочисленный вектор размера N. Удалить из вектора все элементы, встречающиеся более одного раза, и вывести размер полученного вектора и его содержимое.
15. Дан целочисленный вектор размера N. Вычислить минимальное значение элементов вектора; умножить каждый

элемент вектора на 5 и добавить в конец вектора элемент, равный вычисленному минимуму.

16. Какие операции необходимо запретить в контейнере deque, чтобы фактически он работал как stack?

17. Сформировать множество из 20 случайных чисел. Определить, входят ли в это множество 2 заданных пользователем числа.

18. Создать множество чисел (set <int>), записать в него 20 различных чисел, а затем удалить из множества все числа больше 10.

19. Создать словарь (map <string, string>), занести в него десять записей по принципу «Фамилия» – «Имя», после чего проверить, сколько людей имеют совпадающие с заданным имя или фамилию.

20. Создать словарь (map <string, string>), занести в него десять записей по принципу «Фамилия» – «Имя». Удалить из словаря людей с повторяющимися именами.

21. Дан набор целых чисел с четным количеством элементов. Заполнить вектор V исходными числами и вывести вначале вторую половину элементов вектора V, а затем первую половину (в каждой половине порядок элементов не изменять).

22. Дан набор целых чисел с четным количеством элементов. Заполнить дек D исходными числами и вывести первую половину элементов дека D в обратном порядке, а затем — вторую половину (также в обратном порядке).

23. Дан набор целых чисел, количество которых делится на 3. Заполнить список L исходными числами и вывести вначале первую треть элементов списка L в исходном порядке, затем вторую треть элементов в обратном порядке, а затем последнюю треть (также в обратном порядке).

24. Даны вектор V, дек D и список L. Каждый исходный контейнер содержит не менее трех элементов, количество элементов является нечетным. Удвоить значения первого, среднего и последнего элемента каждого из исходных контейнеров.

25. Даны вектор V, дек D и список L. Каждый исходный контейнер содержит не менее двух элементов, количество элементов является четным. Поменять значения двух средних элементов каждого из исходных контейнеров.

26. Дан вектор V с четным количеством элементов. Добавить в середину вектора 5 нулевых элементов.

27. Даны вектор V и список L. Каждый исходный контейнер содержит не менее 5 элементов. Вставить после элемента списка с порядковым номером 5 первые 5 элементов вектора в обратном порядке.

28. Дан вектор V. Вставить после каждого элемента исходного вектора число −1.

29. Дан вектор V. Удалить все элементы исходного вектора с нечетными порядковыми номерами (считая, что начальный элемент вектора имеет порядковый номер 1).

30. Дан список L. Удалить все элементы исходного списка с четными порядковыми номерами (считая, что начальный элемент списка имеет порядковый номер 1).

31. Дан список L с количеством элементов, кратным 4. Удалить в первой половине исходного списка все элементы с нечетными порядковыми номерами (считая, что начальный элемент списка имеет порядковый номер 1).

7. СТРОКИ

Строка — это последовательность символов (`char`). Константная строка заключается в двойные кавычки ("). Признаком конца строки является нулевой символ `\0`.

В C++ строки допустимо описывать с помощью массива символов (массив элементов типа `char`), в котором следует предусмотреть место для хранения признака конца строки. Использование массива `char` для хранения строк унаследовано от языка Си. Такие массивы символов также называют строковыми массивами, строковыми переменными или строками.

Описание массива для хранения строки из не более чем пяти символов имеет вид:

```
char stroka[6]
```

При описании необходимо учитывать, что в массиве последний элемент представляет нулевой символ '\0' (нуль-символ).

Строка при объявлении может быть инициализирована начальным значением:

```
char stroka [6] = "hello";
char stroka [] = {'h','e','l','l','o','\0'};
```

Как и для массивов, при объявлении строки допустимо не указывать её размер, но тогда обязательно её инициализация начальным значением. В этом случае размер строки определится автоматически и в конец строки добавится нуль-терминатор:

```
char stroka [] = "hello";
```

Потоковый ввод строки осуществляется либо с помощью команды `gets_s`, либо с помощью команды `getline`.

Команда `gets_s` имеет вид:

```
gets_s(<имя строковой переменной>);
```

В результате выполнения данной команды из входного потока считываются все символы до `'\n'` (клавиша Enter) либо до конца

входного потока, включая '\n'; символ '\n' заменяется на '\0' либо добавляется символ '\0'; полученная последовательность записывается в строковую переменную, указанную как параметр. Размер входной последовательности, включая '\0', не должен превышать размер строкового массива.

Использование стандартного ввода сроки вида

```
cin>>stroka;
```

позволяет ввести в переменную stroka последовательность входных символов до первого пробела или до символа '\n' либо до конца входного потока. Существенным здесь является отсутствие возможности ввести строки, содержащие пробелы, а также неизвлечение '\n' символа во входном потоке.

Для ввода последовательностей символов, включающих ' ' (пробел), до символа '\n' либо до конца входного потока используется команда cin.getline. Символ '\n' удаляется из входного потока. Допустимы следующие вызовы команды:

cin.getline(<имя>,<размер>); заносит в переменную, указанную как параметр <имя>, не более <размер> символов либо до символа '\n', если он встретится раньше, либо до конца входного потока, если входной поток меньше <размер>. Обычно <размер> соответствует размеру массива <имя>.

cin.getline(<имя>,<символ>); заносит в переменную, указанную как параметр <имя>, последовательность символов либо до символа <символ>, либо до символа '\n', либо до конца входного потока. Размер входной последовательности, включая '\0', не должен превышать размер строкового массива.

Обратим внимание, что как команда ввода cin оставляет символ '\n' во входном потоке, что приводит к некорректному вводу.

Например, для следующего программного кода

```
char stroka[100];
int x;
cin >> x;
```

```
cin.getline(stroka,100);
```

если пользователь после указания цифр (ввод значения целочисленной переменной x) нажмет клавишу Enter, то символ '\n' (соответствует коду клавиши Enter) останется во входном потоке. Последний символ будет считан командой cin.getline и в stroka будет находиться пустая строка.

Для исключения описанной ситуации предназначена команда cin.ignore. Допустимы следующие вызовы команды:

cin.ignore(); удаляет первый символ из входного потока;

cin.ignore(<количество>,<разделитель>); удаляет из входного потока, в зависимости, что произойдет раньше, либо <количество> символов либо все символы до первого вхождения <разделитель> символа включительно.

Таким образом, для приведенного примера корректный программный код имеет вид

```
char stroka[100];
int x;
cin >> x;
cin.ignore();
cin.getline(stroka,100);
```

Определены следующие функции работы со строковыми массивами:

strlen(<имя_строки>) определяет длину указанной строки, без учёта нуль-символа.

strcpy(<строка1>, <строка2>) выполняет копирование символов из строки <строка2> в строку <строка1>.

strncpy (<строка1>, <строка2>, <кол-во>) выполняет копирование <кол-во> символов из строки <строка2> в строку <строка1>.

strcat(<строка1>, <строка2>) объединяет строку <строка2> со строкой <строка1>, результат сохраняется в <строка1>.

strncat(<строка1>, <строка2>, <кол-во>) объединяет <кол-во> символов строки <строка2> со строкой <строка1>, результат сохраняется в <строка1>.
strcmp(<строка1>, <строка2>) сравнивает строку <строка1> со строкой <строка2> и возвращает результат типа int: 0 – если строки эквивалентны, положительное значение – если <строка1> в лексикографическом порядке больше <строка2>, отрицательное значение — если <строка1> в лексикографическом порядке меньше <строка2> с учётом регистра.
strncmp(<строка1>, <строка2>, <кол-во>) сравнивает <кол-во> символов строки <строка1> со строкой <строка2> и возвращает результат типа int: 0 – если строки эквивалентны, положительное значение – если <строка1> в лексикографическом порядке больше <строка2>, отрицательное значение — если <строка1> в лексикографическом порядке меньше <строка2> с учётом регистра.
stricm(<строка1>, <строка2>) сравнивает строку <строка1> со строкой <строка2> и возвращает результат типа int: 0 –если строки эквивалентны, положительное значение – если <строка1> в лексикографическом порядке больше <строка2>, отрицательное значение — если <строка1> в лексикографическом порядке меньше <строка2> без учёта регистра.
strnicmp(<строка1>, <строка2>, <кол-во>) сравнивает <кол-во> символов строки <строка1> со строкой <строка2> и возвращает результат типа int: 0 – если строки эквивалентны, положительное значение – если <строка1> в лексикографическом порядке больше <строка2>, отрицательное значение — если <строка1> в лексикографическом порядке меньше <строка2> без учёта регистра.
isalnum(<символ>) возвращает значение true, если <символ> является буквой или цифрой, и false в других случаях.

isalpha(<символ>) возвращает значение true, если <символ> является буквой, и false в других случаях.
isdigit(<символ>) возвращает значение true, если <символ> является цифрой, и false в других случаях.
islower(<символ>) возвращает значение true, если <символ> является буквой нижнего регистра, и false в других случаях.
isupper(<символ>) возвращает значение true, если <символ> является буквой верхнего регистра, и false в других случаях.
isspace(<символ>) возвращает значение true, если <символ> является пробелом, и false в других случаях.
toupper(<символ>) если символ <символ>, является символом нижнего регистра, то функция возвращает преобразованный символ <символ> в верхнем регистре, иначе символ возвращается без изменений.
strchr(<строка>, <символ>) поиск первого вхождения символа <символ> в строке <строка>. В случае удачного поиска возвращает указатель на место первого вхождения символа <символ>. Если символ не найден, то возвращается NULL.
strcspn(<строка1>, <строка2>) определяет длину начального сегмента строки <строка1>, содержащего те символы, которые не входят в строку <строка2>.
strspn(<строка1>, <строка2>) возвращает длину начального сегмента строки <строка1>, содержащего только те символы, которые входят в строку <строка2>.
strprbk(<строка1>, <строка2>) Возвращает указатель первого вхождения любого символа строки <строка2> в строке <строка1>.
atof(<строка>) преобразует строку <строка> в тип double.
atoi(<строка>) преобразует строку <строка> в тип int.
atol(<строка>) преобразует строку <строка> в тип long.

В MS VisualStudio ряд функций над строковыми массивами признаются как небезопасные с точки зрения работы с памятью ЭВМ. Для таких функций введены новые, более безопасные. Имена таких функций совпадают со стандартными с добавлением суффикса «_s», например, для функции `strcpy` определен безопасный аналог `strcpy_s`.

Пример 1. Дано n строк (n<=10000) длиной не более 200 символов каждая. Найти самую длинную строку, в которой цифр больше, чем букв.

```
#include <iostream>
using namespace std;

bool MoreDigits(char *s) {
 int digits = 0, letters = 0;
 for (int i = 0; i < strlen(s); ++i) {
      if (isdigit(s[i])) //подсчет цифр
           ++digits;
      if (isalpha(s[i])) //подсчет букв
           ++letters;
 }
 return digits > letters;
}

int main() {
 setlocale(LC_ALL, "RUSSIAN");
 int n;
 cin >> n;
 cin.ignore();
 char maxstroka[200] = "\0";
for (int i = 0; i < n; ++i) {
      char str[200];
      cin.getline(str, 200);
      if (MoreDigits(str) &&
strlen(maxstroka) < strlen(str))
           strcpy_s(maxstroka,str);
 }
 cout << "Найденная строка: " << maxstroka
<< endl;
```

```
 system("pause");
 return 0;
}
```

Кроме возможности работы со строками, как с массивами, в C++ определен контейнерный класс `string`. Для работы с ним необходимо подключить модуль `string` и использовать пространство имен `std`:

```
#include <string>
```

Объявление переменной типа `string` осуществляется точно так же как и обычной переменной. Возможен вариант объявления с одновременной инициализацией. Контейнерный класс `string` близок по функционалу к классу `vector`, поддерживает все функции, характерные для контейнерного класса, в нем добавлены специфические функции работы со строками.

Класс `string` удобен тем, что позволяет манипулировать строками, используя стандартные (перегруженные) операторы.

С объектами класса `string` можно использовать следующие операторы:

= присваивание;
+ конкатенация (объединение строк);
+= присваивание с конкатенацией;
== равенство;
!= неравенство;
< меньше;
<= меньше или равно;
> больше;
>= больше или равно;
[] индексация.

При написании программ на C++ следует отдавать предпочтение типу `string`, а не массиву символов. Однако, например, при передаче строк по сети могут возникать ситуации, когда со строкой необходимо работать, как с массивом байтов. Для этого в классе `string` есть специальные методы:

`c_str()` возвращает указатель на массив, содержащий байтовое представление символов строки, оканчивающееся `'\0'`. Аналогичный адрес возвращает выражение `&s[0]`, где `s` — это переменная типа `string`. Массив по этому адресу предполагается использовать только для чтения.

`copy(<dest>, <count>, <pos>)` копирует в строковый массив `<dest>` из текущей строки `<count>` символов, начиная с `<pos>`, либо до конца строки. Возвращает количество скопированных символов. Значение `<pos>` должно быть меньше длины строки. Символ `'\0'` в конце `<dest>` не добавляется. Последнее следует учитывать, если подразумевается дальнейшая работа со строковым массивом `<dest>`. Пример корректного кода:

```
char buffer[20];
string str = "Test string...";
size_t length = str.copy(buffer, 6, 5);
buffer[length] = '\0';
```

Ввод значений типа `string` эквивалентен описанным способам ввода строковых массивов. Кроме этого в библиотеке `string` введена команда `getline` для ввода строк. Допустимы следующие вызовы команды:

`getline(<stream>, <str>, <delim>);` вводит строку в переменную `<str>` из входного потока `<stream>` до достижения символа `<delim>`. Символ `<delim>` из входного потока удаляется, но в `<str>` не попадает.

`getline(<stream>, <str>);` вводит строку в переменную `<str>` из входного потока `<stream>` до достижения символа `'\n'`. Символ `'\n'` из входного потока удаляется, но в `<str>` не попадает.

Для перевода числа в строку определена функция

```
to_string(<value>);
```

возвращающая строковое представление значения `<value>`.

В класс `string` добавлены следующие функции для работы со строками:

length() возвращает количество символов в строке, для с однобайтовых кодировок символов эквивалентен size().

append добавляет в конец строки последовательность символов. Определены следующие спецификации:

append(<str>) добавляет к текущей строке содержимое <str>. <str> – либо константная строка, либо переменная типа string, либо строковый массив (в этом случае символ '\0' не добавляется).

append(<str>, <pos>, <len>) добавляет к текущей строке <len> символов строки <str> (либо до конца строки, если <len>+<pos> больше длины <str>), начиная с символа в позиции <pos>. <str> – либо константная строка, либо переменная типа string.

append(<str>, <len>) добавляет к текущей строке <len> символов строки <str> (либо до конца строки, если <len> меньше длины <str>). <str> – строковый массив (в этом случае символ '\0' не добавляется).

append(<count>, <symbol>) добавляет к текущей строке <count> символов <symbol>.

assign присваивает текущей строковой переменной последовательность символов. Определены следующие спецификации:

assign(<str>) присваивает содержимое <str>. <str> – либо константная строка, либо переменная типа string, либо строковый массив (в этом случае символ '\0' не добавляется).

assign(<str>, <pos>, <len>) присваивает <len> символов строки <str> (либо до конца строки, если <len>+<pos> больше длины <str>), начиная с символа в позиции <pos>. <str> – либо константная строка, либо переменная типа string.

assign(<str>, <len>) присваивает <len> символов строки <str> (либо до конца строки, если <len> меньше длины <str>). <str> – строковый массив (в этом случае символ '\0' не добавляется).

assign(<count>, <symbol>) присваивает <count> символов <symbol>.

insert вставляет в строку последовательность символов непосредственно перед символом в позиции <s_pos>. Определены следующие спецификации:

insert(<s_pos>, <str>) вставляет содержимое <str>. <str> – либо константная строка, либо переменная типа string, либо строковый массив (в этом случае символ '\0' не добавляется).

insert(<s_pos>, <str>, <pos>, <len>) вставляет <len> символов строки <str> (либо до конца строки, если <len>+<pos> больше длины <str>), начиная с символа в позиции <pos>. <str> – либо константная строка, либо переменная типа string.

insert(<s_pos>, <str>, <len>) вставляет <len> символов строки <str> (либо до конца строки, если <len> меньше длины <str>). <str> – строковый массив (в этом случае символ '\0' не добавляется).

insert(<s_pos>, <count>, <symbol>) вставляет <count> символов <symbol>.

erase(<pos>, <count>) удаляет из строки <count>, начиная с символа в позиции <pos> (либо до конца строки, если <count>+<pos> больше длины строки). Если <pos> не указан, то значение считается равным нулю.

replace заменяет в строке последовательность символов длины <s_len>, непосредственно начиная с символа в позиции <s_pos>. Определены следующие спецификации:

replace(<s_pos>, <s_len>, <str>) заменяет содержимым <str>. <str> – либо константная строка, либо переменная типа string, либо строковый массив (в этом случае символ '\0' не добавляется).

replace(<s_pos>, <s_len>, <str>, <pos>, <len>) заменяет на <len> символов строки <str> (либо до конца строки, если <len>+<pos> больше длины <str>), начиная с символа в позиции <pos>. <str> – либо константная строка, либо переменная типа string.

replace(<s_pos>, <s_len>, <count>, <symbol>) заменяет на <count> символов <symbol>.

substr(<pos>, <count>) возвращает <count> строки, начиная с символа в позиции <pos> (либо до конца строки, если <count>+<pos> больше длины строки). Если <pos> не указан, то значение считается равным нулю.

compare сравнивает текущую строку (или ее часть) с последовательностью символов <str> (или ее частью). Возвращает 0, если строковые последовательности совпадают; положительное значение, если последовательность текущей строки в лексикографическом порядке больше последовательности <str>; отрицательное значение, если последовательность текущей строки в лексикографическом порядке меньше последовательности <str>. Определены следующие спецификации:

compare(<str>) сравнивает текущую строку с последовательностью символов <str>. <str> – либо константная строка, либо переменная типа string, либо строковый массив (в этом случае символ '\0' игнорируется).

compare(<s_pos>, <s_len>, <str>) сравнивает подстроку длины <s_len>, начиная с <s_pos> текущей строки со строкой <str>. <str> – либо константная строка, либо переменная типа string, либо строковый массив (в этом случае символ '\0' не добавляется).

compare(<s_pos>, <s_len>, <str>, <pos>, <len>) сравнивает подстроку длины <s_len>, начиная с <s_pos> текущей строки с подстрокой длины <len>, начиная с < pos> строки <str>. <str> – либо константная строка, либо переменная типа string.

compare(<s_pos>, <s_len>, <symbol>, <count>, <len>) сравнивает подстроку длины <s_len>, начиная с <s_pos> текущей строки с последовательностью из <count> символов <symbol>.

find осуществляет поиск, начиная с символа в позиции <pos> в текущей строке первого вхождения последовательности символов, заданной в качестве параметра, и возвращает номер символа текущей строки первого найденного вхождения. Если вхождение не найдено, то возвращает -1. Если <pos> не

указан, то поиск осуществляется по всей строке. Определены следующие спецификации:

`find (<str>, <pos>)` ищет содержимое `<str>`. `<str>` – либо константная строка, либо переменная типа `string`, либо строковый массив (в этом случае символ `'\0'` не добавляется), либо символ.

`find (<str>, <pos>, <count>)` ищет содержимое первых `<count>` символов `<str>`. `<str>` – строковый массив (в этом случае символ `'\0'` не добавляется), указание `<pos>` обязательно.

`rfind` осуществляет поиск, начиная с символа в позиции `<pos>` в текущей строке последнего вхождения последовательности символов, заданной в качестве параметра, и возвращает номер символа текущей строки найденного вхождения. Спецификации аналогичны `find`.

`find_first_of` осуществляет поиск, начиная с символа в позиции `<pos>` в текущей строке номера первого символа, который содержится в последовательности символов, заданной в качестве параметра. Спецификации аналогичны `find`.

`find_last_of` осуществляет поиск, начиная с символа в позиции `<pos>` в текущей строке номера последнего символа, который содержится в последовательности символов, заданной в качестве параметра. Спецификации аналогичны `rfind`.

`find_first_not_of` осуществляет поиск, начиная с символа в позиции `<pos>` в текущей строке номера первого символа, который не содержится в последовательности символов, заданной в качестве параметра. Спецификации аналогичны `find_first_of`.

`find_last_not_of` осуществляет поиск, начиная с символа в позиции `<pos>` в текущей строке номера последнего символа, который не содержится в последовательности символов, заданной в качестве параметра. Спецификации аналогичны `find_last_of`.

Пример 2. Дано n строк (n<=10000). Найти две строки, в которых нет общих символов и суммарная длина максимальна.

```
#include <iostream>
#include <string>
```

```
#include <vector>
using namespace std;

int main() {
 setlocale(LC_ALL, "RUSSIAN");
 vector<string> v;
 int n;
 cin >> n;
 cin.ignore();
 v.resize(n);
 for (size_t i = 0; i < n; ++i)
      getline(cin,v[i]);
 string s1 = "", s2 = "";
 for (size_t i = 0; i < n; ++i)
      for (size_t j = i+1; j < n; ++j)
           if (v[i].find_first_of(v[j]) == -1
&& v[i].length() + v[j].length() > s1.length() +
s2.length()) {
                s1 = v[i];
                s2 = v[j];
           }
 if (s1.length() == 0)
      cout << "\nискомых строк нет";
 else
      cout << '\n' + s1 + '\n' + s2 + '\n';
 system("pause");
 return 0;
}
```

Контрольные вопросы и задания

1. Дана строка. Входят ли знаки «+», «—», «*» в первую половину строки?

2. Дана строка S. Преобразовать строку, заменив в ней каждую точку многоточием.

3. Дана строка. Если длина строки четна, то найти количество букв, иначе количество цифр.

4. Дана строка. Из второй половины каждой строки удалить все «.».

5. Дана строка. Если длина строки нечетна, то удалить средний символ, иначе вставить в середину строки «abc».

6. Дана строка. Если в строке больше двух цифр, то удалить первый и последний символы строки,

7. Дана строка. Если длина строки четна, то вставить в середину строки первый символ.

8. Дана строка. Заменить в строке все цифры на первую букву строки.

9. Дана строка S. Исключить из строки группы символов, расположенные между скобками (,). Сами скобки тоже должны быть исключены. Предполагается, что внутри каждой пары скобок нет других скобок.

10. Дана строка. Если буква «а» входит в первую половину строки, то вставить в начало строки «xyz», иначе удалить первый символ строки.

11. Дана строка. Напечатать «да», если вторая половина строки не содержит цифр.

12. Дана строка. Заменить все цифры, находящиеся в первой половине строки на «*».

13. Дана строка. Напечатать «да», если строка не содержит цифр.

14. Дана строка. Напечатать «да», если строка содержит только цифры.

15. С клавиатуры вводится целое число. В десятичной записи числа заменить все вхождения цифры 0 на 99, к результирующему числу прибавить 1 и вывести результат на экран.

16. Дана строка, содержащая предложение на английском языке. Слова разделены пробелом либо запятой. Определить количество слов в предложении.

17. Дана строка. Проверить, является ли она палиндромом с точностью до вхождения пробелов.

18. С клавиатуры вводится количество и считывается указанное количество строк. Сформировать новую строку, равную конкатенации введенных строк.

19. С клавиатуры вводятся две строки. Заменить в первой строке все вхождения второй строки без пересечений на количество таких вхождений.

20. Задан массив строк. Отсортировать строки лексикографически (в алфавитном порядке).

21. Дана строка, содержащая запись выражения, состоящего из целых чисел и символов операций «+», «-», «*» и «:». Вычислить значения выражения с учетом приоритета математических операций.

22. Написать программу, генерирующую слова на вымышленном языке конкатенацией от одного до пяти суффиксов из списка заданных суффиксов и одного или двух корней из списка допустимых корней в случайном порядке.

8. ФАЙЛЫ

Файлом называют способ хранения информации на физическом устройстве. Файл — это понятие, которое применимо ко всему — от файла на диске до терминала. Конкретно под файлом понимается некоторая последовательность байтов, которая имеет своё уникальное имя, например file.txt. В одной директории не могут находиться файлы с одинаковыми именами. Под именем файла понимается не только его название, но и расширение, например: file.txt и file.dat — разные файлы, хоть и имеют одинаковые названия. Существует такое понятие, как полное имя файлов, – это полный адрес к директории файла с указанием имени файла, например: D:\examples\file.txt.

В C++ отсутствуют операторы для работы с файлами. Все необходимые действия выполняются с помощью функций, включенных в стандартную библиотеку. Они позволяют работать с различными устройствами, такими, как диски, принтер, коммуникационные каналы и т.д. Эти устройства сильно отличаются друг от друга. Однако файловая система преобразует их в единое абстрактное логическое устройство, называемое потоком.

Текстовый поток — это последовательность символов. При передаче символов из потока на экран, часть из них не выводится (например, символ возврата каретки, перевода строки).

Двоичный (бинарный) поток — это последовательность байтов, которые однозначно соответствуют тому, что находится на внешнем устройстве.

Файловый потоковый ввод / вывод аналогичен стандартному потоковому вводу / выводу, единственное отличие – это то, что ввод / вывод выполнятся не на экран, а в файл. Если ввод / вывод на стандартные устройства выполняется с помощью объектов cin и cout, то для организации файлового ввода / вывода создаются собственные объекты, которые можно использовать аналогично операторам cin и cout.

Для работы с файлами необходимо подключить заголовочный файл `<fstream>`**.** В `<fstream>` определены несколько классов и подключены заголовочные файлы `<ifstream>` — файловый ввод и `<ofstream>` — файловый вывод.

Наиболее частые операции с файлами следующие:

– операторы перенаправления ввода / вывода << и >>;

– методы записи и чтения строк `getline()`, `get()`, `put()`;

– потоковая запись и чтение методами `write()` и `read()`;

– методы открытия / создания и закрытия файлов `open()` и `close()`;

– метод проверки открыт ли файл `is_open()`;

– метод проверки достигнут ли конец файла `eof()`;

– настройка форматированного вывода для >> с помощью `width()` и `precision()`;

– операции позиционирования `tellg()`, `tellp()` и `seekg()`, `seekp()`.

При работе с файлом можно выделить следующие этапы:

– создать объект класса `fstream` (возможно, `ofstream` или `ifstream`);

– связать объект класса `fstream` с файлом, который будет использоваться для операций ввода-вывода;

– осуществить операции ввода-вывода в файл;

– закрыть файл.

Пример 1. Создать текстовый файл и записать в него строку "Файлы в С++".

```
#include <fstream> // подключаем библиотеку
using namespace std;

int main(){
 ofstream fout("first.txt");/* открыли файл для записи*/
 fout <<"Файлы в С++"<< endl;
 fout.close();// закрыли файл
 return 0;
}
```

Для того чтобы прочитать данные из файла, понадобится выполнить следующие шаги:

– создать объект класса `ifstream` и связать его с файлом, из которого будет производиться считывание;

– прочитать данные из файла;

– закрыть файл.

Пример 2. Дан текстовый файл. Считать из файла первое слово и строку.

```
#include <fstream>
#include <iostream>
using namespace std;

int main(){
 setlocale(LC_ALL, "rus");
 char buff[50];
 ifstream fin("first.txt");/* открыли файл
для чтения*/
 fin >> buff;// считали первое слово из файла
 cout << buff << endl;
 fin.getline(buff, 50);// считали строку
 fin.close();// закрыли файл
 cout << buff << endl;
 system("pause");
 return 0;
}
```

В программе показаны два способа чтения из файла, первый – используя операцию передачи в поток >>, второй – используя функцию `getline()`. В первом случае считывается только первое слово, а во втором – строка длиной до 50 символов. Но так как в файле осталось меньше 50 символов, то считываются символы включительно до конца строки. Обратите внимание, что считывание во второй раз продолжилось после первого слова, а не с начала, так как первое слово было прочитано ранее.

Однако если файл не будет найден, то и прочитать данные из него невозможно. Поэтому компилятор проигнорирует строки, где выполняется работа с файлом. В результате корректно завершится работа программы, но ничего на экране показано не будет. Чтобы обработать данное событие, в C++ предусмотрена функция `is_open()`, которая возвращает целые значения: 1 – если файл был успешно открыт, 0 – если файл открыт не был.

Пример 3. Дан текстовый файл. Считать из файла первое слово и строку из файла "second.doc", если он существует.

```
#include <fstream>
#include <iostream>
using namespace std;

int main(){
setlocale(LC_ALL,"rus");
char buff[50]; // буфер
ifstream fin("second.doc");
if (!fin.is_open())
  cout << "Файл не может быть открыт!\n";
else {
 fin >> buff;
 cout << buff << endl;
 fin.getline(buff, 50);
 fin.close(); // закрываем файл
 cout << buff << endl;
}
 system("pause");
 return 0;
}
```

Режимы открытия файлов устанавливают характер использования файлов. Для установки режима предусмотрен класс `ios`, в котором предусмотрены константы, определяющие режим открытия файлов:

`ios::in` открыть файл для чтения;

`ios::out` открыть файл для записи;

`ios::ate` при открытии переместить указатель в конец файла;

`ios::app` открыть файл для записи в конец файла;

`ios::trunc` удалить содержимое файла, если он существует;

`ios::binary` открытие файла в двоичном режиме.

Режимы открытия файлов можно устанавливать непосредственно при создании объекта или при вызове метода `open()`, например:

```
ofstream fout("file.txt", ios::app);
fout.open("file.txt", ios::app);
```

Режимы открытия файлов можно комбинировать с помощью поразрядной логической операции ИЛИ |, например:

`ios::out | ios::in` – открытие файла для записи и чтения.

Объекты класса `ofstream` при связке с файлами по умолчанию содержат режимы открытия файлов `ios::out | ios::trunc`, т. е. файл будет создан, если не существует. Если же файл существует, то его содержимое будет удалено, а сам файл будет готов к записи.

Объекты класса `ifstream`, связываясь с файлом, имеют по умолчанию режим открытия файла `ios::in` — файл открыт только для чтения.

Режимы `ate` и `app` по описанию очень похожи, они оба перемещают указатель в конец файла, но режим `app` позволяет производить запись только в конец файла, а режим `ate` просто переставляет указатель файла в конец файла и не ограничивает места записи.

Потоки для работы с текстовыми файлами представляют объекты, для которых не задан режим открытия `ios::binary`.

Пример 4. Файл содержит несколько строк, в каждой из которых записано единственное выражение вида a#b (без ошибок), где a, b – целочисленные величины, # – операция +, -, /, *. Вывести каждое из выражений и их значения.

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;
int main(){
 setlocale(LC_ALL, "RUSSIAN");
 cout << "Укажите имя файла: ";
 string s;
 cin >> s;
 ifstream f;
 f.open(s);
 if (!f.is_open()) {
```

```
        cout << "ошибка открытия файла\n";
        system("pause");
        return 0;
    }
    while (!f.eof()){
        getline(f,s);
        int i = s.find_first_of("+-*/");
        cout << s << " = ";
        if (i == -1) {
            cout << "ошибка\n";
            continue;
        }
        long a = atol(s.substr(0, i).c_str());
        long b = atol(s.substr(i+1,
s.length()).c_str());
        switch (s[i]) {
        case '+': a += b; break;
        case '-': a -= b; break;
        case '/': a /= b; break;
        case '*': a *= b; break;
        }
        cout << a << endl;
    }
    f.close();
    system("pause");
    return 0;
}
```

Пример 5. В заданном файле целых чисел посчитать количество компонент, кратных 3.

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main(){
 setlocale(LC_ALL, "RUSSIAN");
 cout << "Укажите имя файла: ";
```

```
 ifstream f;
 { // s существует только в этом блоке
      string s;
      cin >> s;
      f.open(s);
 }
 if (!f.is_open()) {
      cout << "ошибка открытия файла\n";
      system("pause");
      return 0;
 }
 int counter = 0;
 while (!f.eof()){
      int n;
      f>>n;
      if (n % 3 == 0)
           ++counter;
 }
 f.close();
 cout << "Ответ: " << counter<<endl;
 system("pause");
 return 0;
}
```

Пример 6. Записать в текстовый файл табличное задание функции fun на отрезке [a,b].

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

double fun(double x);

int main(){
 setlocale(LC_ALL, "RUSSIAN");
 cout << "Укажите имя файла: ";
 ofstream f;
 {
```

```
        string s;
        cin >> s;
        f.open(s);
    }
    double a, b, h;
    cout << "Укажите начало, конец отрезка и
шаг: ";
    cin >> a >> b >> h;
    for (double x = a; x <= b; x += h){
        f.width(10); f << x;
        f.width(15); f << fun(x) << endl;
    }
    f.close();
    return 0;
}

double fun(double x){
 return x*x + 5 * x;
}
```

Использование бинарных файлов (с режимом `ios::binary`) дает возможность производить работу с данными на уровне битов, что позволяет производить запись или считывание различных управляющих символов, например, `'\n'`, который заканчивает строку и начинает новую. В отличие от текстовых файлов доступ к элементам бинарных файлов может выполняться в произвольном порядке, а не последовательно. Поэтому бинарные файлы называют файлами произвольного доступа. Заметим, что текстовые файлы являются частным случаем бинарных, в силу чего инструментарий для бинарных файлов можно применять и при работе с текстовыми файлами.

Приложение, в котором предполагается использовать файлы произвольного доступа, должно их создавать при соблюдении некоторых правил. Все записи в таком файле должны быть одинаковой фиксированной длины. В этом случае данные могут быть добавлены в файл прямого доступа без разрушения других данных, изменены или удалены без перезаписи всего файла.

Для получения доступа к бинарному файлу (потоку) необходимо выполнить следующие действия:

1. Создать поток соответствующего типа:
   `ifstream` для ввода из файла ;
   `ofstream` для вывода в файл;
   `fstream` для обмена с файлом в двух направлениях.
2. Связать его с файлом данных и открыть (`open`) для работы в определенном режиме, с обязательным указанием двоичного режима `ios::binary` (по умолчанию потоки открываются в текстовом режиме).
3. Для чтения из потока используется метод `read`:

```
read(const char_type *Str, streamsize Count);
```

4. Для записи в поток в C++ используется метод `write`:

```
write(const char_type *Str, streamsize Count);
```

Здесь `Count` – число выводимых (вводимых) в поток байт, `Str` – символы, выводимые(вводимые) в (из) поток(а).

Пример кода для считывания из файла `N` байт:

```
int n=10;
char* buffer=new char[n+1];
buffer[n]=0;
file.read(buffer,n);
…
delete [] buffer;
```

При работе с бинарными файлами данные произвольного типа `x` необходимо предварительно представить как последовательность байтов:

```
тип x;
// x переводим в строку байтов
(char*)&x  // указатель на начало строки
sizeof(x)  // размер в байтах
```

Пример 7. Запись и чтение числовых данных из бинарного файла.

```
#include <iostream>
#include <fstream>
#include <string>
```

```
using namespace std;

int main(){
 setlocale(LC_ALL, "RUSSIAN");
 cout << "Укажите имя файла: ";
 string s;
 cin >> s;
 /*Запись в бинарный файл*/
 ofstream f_out;
 f_out.open(s, ios::binary);
 int x = 100;
 double y = 5.988;
 f_out.write((char*)&x, sizeof(x));
 f_out.write((char*)&y, sizeof(y));
 f_out.close();
 /*Чтение из бинарного файла*/
 int x1 = 0;
 double y1 = 0;
 ifstream f_in(s, ios::binary);
 f_in.read((char*)&x1, sizeof(x1));
 f_in.read((char*)&y1, sizeof(y1));
 f_in.close();
 cout << x1 << '\n' << y1 << '\n';
 system("pause");
 return 0;
}
```

Пример 8. Чтение и запись структур в бинарный файл.

```
#include <iostream>
#include <fstream>

using namespace std;

struct Worker {
      char name[255]; //Фамилия
      float salary;  //Зар. плата
      int age; //Возраст
 };
```

```
int main(){
 setlocale(LC_ALL, "RUSSIAN");
 cout << "Укажите имя файла: ";
 const string FName ="worker.bin";

 Worker teacher;
 strcpy_s(teacher.name,"Pupkin");
 teacher.age = 30;
 teacher.salary = 1523.99;
 fstream f1(FName, ios::binary | ios::out);
 f1.write((char*)&teacher, sizeof(teacher));
 f1.close();

 Worker w;
 fstream f2(FName, ios::binary | ios::in);
 f2.read((char*)&w, sizeof(w));
 f2.close();

 cout << w.name << '\t' << w.age << '\t' <<
w.salary << '\n';

 system("pause");
 return 0;
}
```

Система ввода – вывода C++ обрабатывает два указателя, ассоциированные с каждым файлом:

- get pointer – определяет, где именно в файле будет производиться следующая операция ввода;
- put pointer – определяет, где именно в файле будет производиться следующая операция вывода.

Всякий раз, когда осуществляются операции ввода или вывода, соответствующий указатель автоматически перемещается.

Доступ к файлу в произвольном месте можно получить с помощью методов `seekg()` и `seekp()`:

```
ifstream &seekg(Смещение, Позиция);
ofstream &seekp(Смещение, Позиция);
```

Здесь Смещение определяет область значений в пределах файла (long int). Позиция смещения определяется как:

ios::beg начало файла;
ios::cur текущее положение;
ios::end конец файла.

Например:

```
file.seekg(0,ios::end); //Стать в конец файла
file.seekg(10,ios::end); /*Стать на 10 байтов с конца*/
file.seekg(30,ios::beg); //Стать на 31-й байт
file.seekg(3,ios::cur); /*перепрыгнуть через 3 байта*/
file.seekg(3); /*перепрыгнуть через 3 байта*/
```

Текущую позицию файлового указателя можно определить, используя следующие функции:

streampos tellg() позиция для ввода;
streampos tellp() позиция для вывода.

Метод tellg() возвращает значение, которое показывает, сколько в файле уже пройдено в байтах. Его можно использовать в паре с методом seekg(), чтобы получать размер файла:

```
//становимся в конец файла
file.seekg(0,ios::end);
//Получаем текущую позицию
cout << "Размер файла (в байтах): " << file.tellg();
```

Пример 9. Функция для чтения count символов-байт из файла f с указанной позиции position:

```
char *my_read(fstream f, int position, int count){
```

```
    if (!f.is_open())
        return NULL;
    f.seekg(position);
    if (f.eof())
        return NULL;
    char *buffer = new char[count];
    f.read(buffer, count);
    return buffer;
}
```

Пример 10. Прямой доступ к содержимому текстового файла. Здесь один и тот же файл используется как для записи, так и для чтения.

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    setlocale(LC_ALL, "RUSSIAN");
    fstream inout; // объект файл без указания
метода доступа

    inout.open("file.txt", ios::out); // файл
открыт для записи
    inout << "строчка текста" << endl;
    inout.seekp(8, ios::beg);
    inout << "еще строчка текста";
    inout.close();

    inout.open("file.txt", ios::in); // файл
открыт для чтения
    inout.seekg(-16, ios::end);
    char s[80];
    inout.getline(s,80);
    inout.close();
    cout << s<<endl;
    system("pause");
    return 0;
```

```
}
```

В результате работы программы файл «file.txt» будет содержать `строчка тееще строчка текста`, а на экран будет выведено `е строчка текста`.

Контрольные вопросы и задания

1. Радиотелескоп пытается получать и анализировать сигналы из космоса. Все сигналы представляются в виде вещественного неотрицательного числа и сохраняются последовательно в текстовом файле. Характеристикой определенного района космоса считается число, равное максимальному произведению, которое можно получить, перемножая значения сигналов, приходящих из этого района. То есть требуется выбрать такое непустое подмножество сигналов (в него может войти как один сигнал, так и все поступившие сигналы), произведение значений у которого будет максимальным. Если таких подмножеств несколько, то выбрать можно любое из них. Напишите программу, которая будет обрабатывать результаты эксперимента, находя искомое подмножество.

2. Каждая запись текстового файла содержит текст на английском языке. Требуется написать программу, которая для каждой записи файла будет определять, можно ли переставить латинские буквы каждой записи так, чтобы получился палиндром (палиндром читается одинаково слева направо и справа налево). В выходном файле записать ответ «Да» или «Нет», а в случае ответа «Да» – еще и сам полученный палиндром (первый в алфавитном порядке).

3. Файл содержит текст заклинания. Гарри Поттеру нужно зашифровать его следующим образом. В каждом абзаце Гарри сначала определяет количество букв в самом коротком слове, обозначив полученное число через K. Затем он заменяет каждую английскую букву в заклинании на букву, стоящую в английском алфавите на K букв ранее (алфавит считается циклическим, т. е. перед буквой A стоит буква Z), оставив другие символы неизменными. Строчные буквы при этом остаются строчными, а прописные – прописными. Написать программу для Гарри Поттера, которая будет выводить в файл текст зашифрованного заклинания.

4. Файл содержит результаты голосования избирателей за несколько партий, в виде списка названий данных партий. В каждой строке файла записано название партии, за которую проголосовал данный избиратель. Название партии может содержать буквы, цифры, пробелы и прочие символы. Количество партий определяется на основе данных файла. Программа должна вывести в выходной файл список названий всех партий, встречающихся в исходном списке, в порядке убывания количества голосов, отданных за эту партию. При этом название каждой партии должно быть выведено ровно один раз, вне зависимости от того, сколько голосов было отдано за данную партию.

5. Входной файл содержит текст, в котором нужно зашифровать все английские слова. Каждое слово шифруется с помощью циклического сдвига на длину этого слова. Например, если длина слова равна K, каждая буква в слове заменяется на букву, стоящую в английском алфавите на K букв дальше (алфавит считается циклическим, т. е. за буквой Z стоит буква A). Строчные буквы при этом остаются строчными, а прописные – прописными. Символы, не являющиеся английскими буквами, не изменяются. Зашифрованный текст разместить в выходном файле.

6. Каждая запись входного файла содержит последовательность символов. Написать программу, которая определяет, есть ли в этой последовательности десятичные цифры, и формирует наибольшее число, которое можно составить из этих цифр. Ведущих нулей в числе быть не должно (за исключением числа 0, запись которого содержит ровно одну цифру). Если цифр нет, программа должна вывести в выходной файл сообщение «NO», а если есть – полученное число.

7. Каждая запись входного текстового файла содержит последовательность цифр. Написать программу, которая цифры, встречающиеся во входной записи, в порядке увеличения частоты их встречаемости выводит в соответствующую запись выводного файла. Если какие-то цифры встречаются одинаковое число раз, они выводятся в порядке возрастания.

8. Каждая запись входного текстового файла содержит последовательность символов. Среди символов обязательно встречаются десятичные цифры. Написать программу, которая из цифр одной записи файла составляет число-палиндром максимальной длины (которое читается одинаково слева направо и

справа налево) и записывает его в выводной файл. Если таких чисел несколько, нужно вывести минимальное из них. Все имеющиеся цифры использовать не обязательно, но количество цифр в ответе должно быть максимально возможным.

9. Каждая запись входного текстового файла представляет предложение на английском языке. Написать программу, которая из английских букв предложения строит палиндром – слово, которое читается одинаково слева направо и справа налево. Строчные и прописные буквы не различаются. Если этого сделать нельзя, программа должна вывести в запись выводного файла слово «NO», а если можно – слово «YES» и затем искомое слово прописными буквами. Если таких слов несколько, нужно вывести последнее в алфавитном порядке слово.

10. Каждая запись входного файла представляет собой набор символов. Напишите программу, которая сначала будет определять, есть ли в этом наборе символы, соответствующие десятичным цифрам. Если такие символы есть, то можно ли переставить их так, чтобы полученное число было симметричным (читалось одинаково как слева направо, так и справа налево). Ведущих нулей в числе быть не должно, исключение – число 0, запись которого содержит ровно один ноль. Если требуемое число составить невозможно, то программа должна вывести в запись выводного файла слово «NO». А если возможно, то сначала вывести слово «YES», а затем – искомое симметричное число. Если таких чисел несколько, то программа должна выводить максимальное из них.

11. Каждая запись входного текстового файла содержит текст на английском языке. Требуется написать программу, которая для каждой записи вводного файла будет определять английскую букву, встречающуюся в этой записи чаще всего, и количество там таких букв. Строчные и прописные буквы при этом считаются неразличимыми. Если искомых букв несколько, то программа должна выводить первую из них по алфавиту. Найденный результат помещать в выводном файле.

12. Каждая запись входного текстового файла содержит произвольные алфавитно-цифровые символы. Написать программу, которая для каждой записи файла будет определять строчные английские буквы, встречающиеся в ней, и частоты их повторения. Результат поместить в выходной файл, буквы выводить в алфавитном порядке.

13. Каждая запись входного файла представляет собой последовательность символов, среди которых могут быть и цифры. Написать программу, которая составляет и выводит минимальное число из тех цифр, которые не встречаются во входной последовательности. Ноль не используется. Если во входной последовательности встречаются все цифры от 1 до 9, то следует вывести «0». Полученные минимальные числа записать во входной файл.

14. Дед Мороз и Снегурочка приходят на детские утренники с мешком конфет. Сведения об утренниках записаны во входном файле. Каждая запись файла описывает один утренник и содержит количество конфет и количество детей на этом празднике. Дед Мороз делит конфеты поровну между всеми присутствующими детьми, а оставшиеся конфеты отдает Снегурочке. Снегурочка каждый раз записывает в блокнот количество полученных конфет. Если конфеты разделились между всеми детьми без остатка, Снегурочка ничего не получает и ничего не записывает. Когда утренники закончились, Деду Морозу стало интересно, какое число чаще всего записывала Снегурочка. Напишите программу, которая будет решать эту задачу.

15. Популярная газета объявила конкурс на выбор лучшего финалиста программы «Голос». На выбор зрителям было предложено 10 претендентов. Необходимо написать программу, которая будет статистически обрабатывать результаты sms-голосования по этому вопросу, чтобы определить популярность того или иного исполнителя. В каждой записи входного файла записано имя финалиста – результат одного голосования. Программа должна вывести в выходной файл список всех финалистов, встречающихся в списке, в порядке убывания (невозрастания) количества отданных за них голосов с указанием этого количества голосов. Имя каждого финалиста должно быть выведено только один раз.

16. Вводной бинарный файл содержит сведения об учащихся, участвовавших в олимпиаде. Каждая запись файла содержит фамилию учащегося и номер школы. Написать программу, которая будет размещать в выводной файл информацию, для каждой школы, указанной во входном файле, количество участников олимпиады.

17. Вводной бинарный файл содержит сведения о студентах некоторого вуза. Каждая запись файла содержит данные о конкретном студенте: фамилия, имя, курс и размер получаемой им

стипендии. Требуется написать программу, которая для каждого курса будет в выводной файл размещать фамилии и имена студентов, имеющих максимальные стипендии.

18. Вводной бинарный файл содержит сведения о сдаче экзаменов студентами некоторой группы. Каждая запись файла содержит фамилию учащегося и три оценки. Написать программу, которая будет сохранять в выводном файле фамилии и имена трех худших по среднему баллу студентов. Если среди остальных есть студенты, набравшие тот же средний балл, что и один из трех худших, то следует вывести и их фамилии.

19. Входной бинарный файл содержит информацию о среднесуточной температуре некоторых дней определенного года. Каждая запись файла содержит номер месяца, номер дня, значение температуры. Хронологический порядок данных не соблюдается. Написать программу, которая будет выводить в выводной файл информацию о месяце (месяцах), среднемесячная температура у которого (которых) наименее отклоняется от среднегодовой. В первой записи выводного файла вывести среднегодовую температуру. Найденные значения для каждого из месяцев следует выводить в отдельных записях в виде: номер месяца, значение среднемесячной температуры, отклонение от средней температуры.

20. Каждая запись входного бинарного файла содержит фамилию и имя студента. Написать программу, которая формирует и печатает уникальный логин для каждого студента по следующему правилу: если фамилия встречается первый раз, то логин – это данная фамилия, если фамилия встречается второй раз, то логин – это фамилия, в конец которой приписывается число 2 и т.д. Полученные логины записать в выводной файл.

21. На городской олимпиаде по информатике участникам было предложено выполнить 3 задания. Каждая запись входного бинарного файла содержит фамилию, имя участника и баллы, полученные им за каждое задание. Напишите программу, которая будет выводить в выводной файл фамилию и имя участника, набравшего максимальное количество баллов. Если есть несколько участников, набравших такое же количество баллов, то их фамилии и имена также следует вывести. При этом имена и фамилии можно выводить в произвольном порядке.

22. Входной бинарный файл содержит сведения о результатах соревнований по многоборью. Каждая запись файла содержит

сведения о фамилии, имени участника и баллы, полученные им по каждому из четырех видов спорта. Победители определяются по наибольшей сумме набранных баллов. Напишите программу, которая будет записывать в выводной файл фамилии и имена трех лучших участников многоборья. Если среди остальных участников есть ученики, набравшие то же количество баллов, что и один из трех лучших, то их фамилии и имена также следует записать в файл. При этом имена и фамилии можно выводить в произвольном порядке.

23. Входной бинарный файл содержит сведения о результатах предварительного тестирования, на основании которого абитуриенты могут быть допущены к сдаче вступительных экзаменов в первом потоке. Тестирование проводится по двум предметам, по каждому предмету абитуриент может набрать от 0 до 100 баллов. При этом к сдаче экзаменов в первом потоке допускаются абитуриенты, набравшие по результатам тестирования не менее 30 баллов по каждому из двух предметов. Каждая запись вводного текстового файла содержит фамилию, имя абитуриента и результаты по каждому из двух предметов. Напишите программу, которая будет выводить в выводной файл фамилии и имена абитуриентов, потерпевших неудачу, т. е. не допущенных к сдаче экзаменов в первом потоке. При этом фамилии должны выводиться в алфавитном порядке.

24. Входной бинарный файл содержит сведения о телефонах всех сотрудников некоторого учреждения. Каждая запись файла содержит сведения о фамилии, имени сотрудника и номере рабочего телефона его отдела. Сотрудники одного отдела имеют один и тот же номер телефона. Номера телефонов в учреждении отличаются только двумя последними цифрами. Написать программу, которая определяет количество отделов в данном учреждении, а в выводном файле записывает количество сотрудников в каждом отделе.

25. Во входном бинарном файле содержится информация о наличии в молочных магазинах города X сметаны жирностью 15, 20 и 25% с указанием цены на эту продукцию. Каждая запись вводного файла содержит данные: наименование фирмы производителя, адрес магазина (улица, номер дома), процент жирности сметаны и цена. Напишите программу, которая будет определять для каждого вида сметаны, сколько магазинов продают ее дешевле всего. Программа должна выводить 3 числа – количество магазинов, продающих

дешевле всего сметану жирностью 15, 20 и 25%. Если какой-то вид сметаны нигде не продавался, то следует вывести 0.

26. Входной бинарный файл содержит список сотрудников с указанием их фамилии, имени и даты рождения. Каждая запись файла содержит сведения о конкретном сотруднике: фамилия, имя дата рождения. Напишите программу, которая будет определять самого старшего человека из этого списка и выводить его фамилию и имя, а если имеется несколько самых старших людей с одинаковой датой рождения, то определять их количество.

27. Вводной бинарный файл содержит сведения о наличии и ценах на автозаправочных станциях (АЗС) бензина с маркировкой 92, 95 и 98. Каждая запись файла имеет формат: компания, улица АЗС, марка, цена за литр. Напишите программу, которая будет определять для каждого вида бензина, сколько АЗС продают его дешевле всего. Программа должна выводить 3 числа – количество АЗС, продающих дешевле всего 92-й, 95-й и 98-й бензин соответственно. Если бензин какой-то марки нигде не продавался, то следует вывести 0.

28. Каждая запись бинарного вводного файла содержит результат одного из участников универсиады: фамилия, имя, курс и количество набранных баллов. Для определения призеров универсиады сначала отбираются 25% участников, показавших лучшие результаты. Если у последнего участника, входящего в 25%, оказывается такое же количество баллов, как и у следующих за ним в итоговой таблице, все они считаются призерами только тогда, когда набранные ими баллы больше половины максимально возможных; иначе все они не считаются призерами. Напишите программу, которая по результатам универсиады будет определять минимальный балл призера универсиады и количество призеров в каждой параллели (среди 1-х, 2-х, 3-х и 4-х курсов отдельно).

29. Напишите программу, которая для каждой из представленных во входном файле групп определяет двух студентов, которые лучше всех сдали информатику, и записывает номер группы, их фамилии и имена в выводной файл. Если наибольший балл набрали более двух человек, нужно вывести только их количество. Если наибольший балл набрал один человек, а следующий балл набрало несколько человек, нужно вывести только фамилию и имя лучшего.

30. Каждая запись бинарного вводного файла содержит результат одного из участников универсиады: фамилия, имя, курс и количество набранных баллов. Победителем универсиады становится участник, набравший наибольшее количество баллов, при условии, что он набрал более 200 баллов. Если такое количество баллов набрали несколько участников, то все они признаются победителями при выполнении условия, что их доля не превышает 20% от общего числа участников. Победителем универсиады не признается никто, если нет участников, набравших больше 200 баллов, или больше 20% от общего числа участников набрали одинаковый наибольший балл. Напишите программу, которая будет определять фамилию и имя лучшего участника, не ставшего победителем универсиады. Если таких участников несколько, т.е. если следующий за баллом победителей один и тот же балл набрали несколько человек, или, если победителей нет, а лучших участников несколько (в этом случае именно они являются искомыми), то выдается только количество искомых участников.

9. ОБРАБОТКА ГРАФОВ

Среди возможных представлений графов воспользуемся списком смежности как наиболее оптимальной по памяти моделью. Каждая вершина в данной модели характеризуется номером и списком ее соседних вершин (соседей). Здесь и далее будем понимать, что вершина B является соседней (соседом) для вершины A тогда и только тогда, когда существует ориентированное или неориентированное ребро (A, B). Таким образом, модель графа – суть множество вершин, его составляющих, так как ребра определены как характеристика (поле) вершин.

В самом простом виде вершина графа может быть описана следующим образом:

```
struct Vertex{
    vector <int> neighbors;
};
```

А граф может быть представлен как вектор вершин:

```
typedef vector<Vertex> Graph;
```

В приведенном описании нумерация вершин графа начинается с нуля.

При решении задач на графы часто возникает необходимость иметь дополнительную информацию о вершинах: наименование, различные весовые характеристики, признаки посещения (просмотра) вершины при выполнении различных алгоритмов и т.д. Также возможен ввод информации о ребрах, например, вес ребра – некоторая числовая характеристика ребра, такой граф называют нагруженным. В этом случае необходимо изменить структуру Vertex, например, следующим образом:

```
struct Vertex{
    string info;
    long weight;
    int visited;
    vector <int> neighbors;
    vector <int> edge_weight;
```

```
};
```

Здесь и далее, если не указано явно, что граф нагруженный, то граф считается ненагруженным.

При вводе графовых структур указывают количество вершин графов и информацию о них, если необходимо, а также вводят информацию ребрах: пару вершин, которые ребро соединяет, и, если необходимо, вес ребра и его ориентацию.

Граф, у которого все ребра неориентированные, называется неориентированным графом. Граф, у которого все ребра ориентированные, – ориентированный граф. Граф, у которого имеются и ориентированные и неориентированные ребра, – смешанный граф.

Пример 1. Дан неориентированный нагруженный граф. Найти вершину с максимальной суммой весов ребер, которым она принадлежит.

```
#include <iostream>
#include <vector>
using namespace std;

struct Vertex {
    vector <int> neighbors;
    vector <int> edge_weight;
};

typedef vector<Vertex> Graph;
Graph graph; //глобальная переменная

int main() {
    setlocale(LC_ALL, "RUSSIAN");
    cout << "Введите количество вершин и
количество ребер графа: ";
    int n, m;
    cin >> n >> m;
    graph.resize(n);
    cout << "\nВведите информацию о ребрах
графа\n";
    for (int i = 0; i < m; ++i) {
        int a, b, w;
```

```
        cin >> a >> b >> w;//ребро (a,b) с весом
w
        graph[a - 1].neighbors.push_back(b - 1);
//вершина a имеет соседа b
        graph[a - 1].edge_weight.push_back(w); //
вес соответствующего ребра
        graph[b - 1].neighbors.push_back(a - 1);
//вершина b имеет соседа a
        graph[b - 1].edge_weight.push_back(w); //
вес соответствующего ребра
    }

    int maxSum = -1, vertex;
    for (int i = 0; i < graph.size(); ++i) {
//перебираем вершины
        int sum = 0;
        for (int j = 0; j <
graph[i].edge_weight.size(); ++j)
//просматриваем веса всех ребер
            sum += graph[i].edge_weight[j];
        if (sum > maxSum) {
            maxSum = sum;
            vertex = i;
        }
    }
    cout << "Искомая вершина: " <<
vertex+1<<endl;
    system("pause");
    return 0;
}
```

Обратим внимание, что обычно в задачах на графы вершины нумеруются с единицы, а индексация в векторах – с нуля, поэтому при вводе необходимо номера вершин декрементировать, а при выводе – инкрементировать.

При вводе неориентированного ребра `(a, b)` необходимо указывать, что вершина `b` смежная вершине `a` (добавить вершину `b` в список соседей вершины `a`) и, наоборот (добавить вершину `a` в список соседей вершины `b`).

Для нагруженных графов вместо двух числовых списков можно использовать один список пар (pair), в котором первый элемент пары определяет смежную вершину, а второй элемент пары – вес ребра.

Для решения рассматриваемой задачи в структуру Vertex можно ввести еще одно поле – сумма весов инцидентных ребер, значение которого определяется на этапе ввода.

С учетом сказанного, решение задачи примера 1 можно представить в виде:

```
#include <iostream>
#include <vector>
using namespace std;

struct Vertex {
    int weight_sum; //сумма весов ребер
    vector <pair<int,int>>
neighbors;//вершина,вес
};

typedef vector<Vertex> Graph;
Graph graph; //глобальная переменная

int main() {
    setlocale(LC_ALL, "RUSSIAN");
    cout << "Введите количество вершин и
количество ребер графа: ";
    int n, m;
    cin >> n >> m;
    graph.resize(n);
    for (int i = 0; i < n; ++i)
        graph[i].weight_sum = 0;
    cout << "\nВведите информацию о ребрах
графа\n";
    for (int i = 0; i < m; ++i) {
        int a, b, w;
        cin >> a >> b >> w;
        graph[a-1].neighbors.push_back(
make_pair(b - 1,w));
```

```
        graph[a - 1].weight_sum += w;
        graph[b-1].neighbors.push_back(
make_pair(a - 1, w));
        graph[b - 1].weight_sum += w;
    }

    int vertex=0;
    for (int i = 0; i < graph.size(); ++i) {
//перебираем вершины
        if (graph[vertex].weight_sum <
graph[i].weight_sum)
            vertex = i;
    }
    cout << "Искомая вершина: " <<
vertex+1<<endl;
    system("pause");
    return 0;
}
```

Большинство алгоритмов обработки графов, представленных списками смежности, основаны на обходах в глубину и в ширину (см. гл. 5).

Пример 2. Дан неориентированный граф. Для заданной пары вершин определить, существует ли путь, их соединяющий.

Решение с использованием обхода в глубину может иметь вид:

```
#include <iostream>
#include <vector>
using namespace std;

struct Vertex {
    int visited; //вершина просмотрена?
    vector <int> neighbors;
};

typedef vector<Vertex> Graph;
Graph graph; //глобальная переменная

bool FindPath(int from_vertex, int to_vertex) {
    graph[from_vertex].visited = 1;
```

```
    for (int i = 0; i <
graph[from_vertex].neighbors.size(); ++i) {
        int current_vertex =
graph[from_vertex].neighbors[i];
        if (current_vertex == to_vertex ||
graph[current_vertex].visited == 0 &&
FindPath(current_vertex, to_vertex))
            return true;
    }
    return false;
}


int main() {
    setlocale(LC_ALL, "RUSSIAN");
    cout << "Введите количество вершин и
количество ребер графа: ";
    int n, m;
    cin >> n >> m;
    graph.resize(n);
    for (int i = 0; i < n; ++i)
        graph[i].visited = 0;
    cout << "\nВведите информацию о ребрах
графа\n";
    for (int i = 0; i < m; ++i) {
        int a, b;
        cin >> a >> b;
        graph[a - 1].neighbors.push_back(b - 1);
        graph[b - 1].neighbors.push_back(a - 1);
    }
    cout << "\nВведите пару вершин:\n";
    int a,b;
    cin >> a>> b;

    if (FindPath(a-1,b-1))
        cout << "Путь существует"<<endl;
    else
        cout << "Путь не найден" << endl;
    system("pause");
    return 0;
```

```
}
```

Функция `FindPath` возвращает `true`, если между парой вершин `from_vertex` и `to_vertex` существует путь. Прежде всего вершина `from_vertex` помечается как посещенная. Затем просматриваются все непосещенные соседи `from_vertex` и, если среди них встретится вершина `to_vertex` или существует путь от соседа до `to_vertex`, то функция возвращает `true`. В противном случае выводится информация, что путь не найден.

Решение данной задачи для ориентированного или смешанного графа отличается от приведенного только вводом ребер графа.

Пример 3. Дан неориентированный граф. Найти количество компонент связанности.

Решение с использованием обхода в глубину может иметь вид:

```
#include <iostream>
#include <vector>
using namespace std;

struct Vertex {
    int visited;
    vector <int> neighbors;
};

typedef vector<Vertex> Graph;
Graph graph;

void MakeComponent(int vertex, int &component) {
    graph[vertex].visited = component;
    for (auto i =
graph[vertex].neighbors.begin(); i !=
graph[vertex].neighbors.end(); ++i) {
        if (graph[*i].visited == 0)
            MakeComponent(*i, component);
    }
}

int main() {
    setlocale(LC_ALL, "RUSSIAN");
```

```
    cout << "Введите количество вершин и
количество ребер графа: ";
    int n, m;
    cin >> n >> m;
    graph.resize(n);
    for (int i = 0; i < n; ++i)
        graph[i].visited = 0;
    cout << "\nВведите информацию о ребрах
графа\n";
    for (int i = 0; i < m; ++i) {
        int a, b;
        cin >> a >> b;
        graph[a - 1].neighbors.push_back(b - 1);
        graph[b - 1].neighbors.push_back(a - 1);
    }

    int c = 0;
    for (int i = 0; i < graph.size(); ++i) {
        if (graph[i].visited == 0)
            MakeComponent(i, ++c);
    }
    cout << "Количество компонент: " << c <<
endl;
    system("pause");
    return 0;
}
```

Поле `visited` структуры `Vertex` в данном решении указывает, что вершина не была посещена, если значение равно нулю, и определяет номер компоненты связанности в противном случае.

Функция `MakeComponent` с параметрами `vertex` и `component` указывает принадлежность вершины `vertex` компоненте с номером `component`, а также всем вершинам, достижимым из данной.

В теле программы для каждой непосещенной вершины вызывается функция `MakeComponent` с очередным номером компоненты.

Решение этой задачи с помощью поиска в ширину имеет вид:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

struct Vertex {
    int visited; //вершина просмотрена?
    vector <int> neighbors;
};

typedef vector<Vertex> Graph;
Graph graph; //глобальная переменная

void MakeComponent(int vertex, int &component) {
    queue<int> q;
    q.push(vertex);
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        graph[v].visited = component;
        for (auto i = graph[v].neighbors.begin();
i != graph[v].neighbors.end(); ++i)
            if (graph[*i].visited == 0)
                q.push(*i);
    }
}

int main() {
    setlocale(LC_ALL, "RUSSIAN");
    cout << "Введите количество вершин и
количество ребер графа: ";
    int n, m;
    cin >> n >> m;
    graph.resize(n);
    for (int i = 0; i < n; ++i)
        graph[i].visited = 0;
    cout << "\nВведите информацию о ребрах
графа\n";
    for (int i = 0; i < m; ++i) {
```

```
        int a, b;
        cin >> a >> b;
        graph[a - 1].neighbors.push_back(b - 1);
        graph[b - 1].neighbors.push_back(a - 1);
    }

    int c = 0;
    for (int i = 0; i < graph.size(); ++i) {
        if (graph[i].visited == 0)
            MakeComponent(i, ++c);
    }
    cout << "Количество компонент: " << c <<
endl;
    system("pause");
    return 0;
}
```

Пример 4. Дан неориентированный нагруженный граф с положительными весами. Найти длину кратчайшего пути от заданной вершины до всех остальных вершин графа.

```
#include <iostream>
#include <vector>
#include <set>
using namespace std;

struct Vertex {
    int visited;
    int weight;
    vector <pair<int, int>> neighbors;
};

typedef vector<Vertex> Graph;
Graph graph; //глобальная переменная

void Dijkstra(int vertex) {
    set<pair<int,int>> s;
    s.insert(make_pair(0, vertex));
    while (!s.empty()) {
        pair<int, int> p = *(s.begin());
```

```
        s.erase(s.begin());
        if (graph[p.second].visited == 1)
            continue;
        graph[p.second].visited = 1;
        graph[p.second].weight = p.first;
        for (auto
i=graph[p.second].neighbors.begin();
i!=graph[p.second].neighbors.end(); ++i)
            if (graph[i->first].visited == 0)

    s.insert(make_pair(graph[p.second].weight
+(i->second), i->first));
    }
}

int main() {
    setlocale(LC_ALL, "RUSSIAN");
    cout << "Введите количество вершин и
количество ребер графа: ";
    int n, m;
    cin >> n >> m;
    graph.resize(n);
    int INFweight = 1;
    cout << "\nВведите информацию о ребрах
графа\n";
    for (int i = 0; i < m; ++i) {
        int a, b, w;
        cin >> a >> b>>w;
        INFweight += w;
        graph[a - 1].neighbors.push_back(
make_pair(b - 1,w));
        graph[b - 1].neighbors.push_back(
make_pair(a - 1, w));
    }
    for (int i = 0; i < n; ++i) {
        graph[i].visited = 0;
        graph[i].weight = INFweight;
    }

    cout << "\nВведите номер вершины:";
```

```
    int v;
    cin >> v;
    Dijkstra(v-1);
    cout << "\nОтвет:\n";
    for (int i = 0; i < graph.size(); ++i) {
        if (graph[i].weight==INFweight)
            cout << "Вершина " << i + 1 << "
недостижима" << endl;
        else
            cout << "Длина пути до вершины " <<
i+1 << ": " << graph[i].weight << endl;

    }
    system("pause");
    return 0;
}
```

Предложенное решение основано на алгоритме Дейкстры с использованием множества пар. Множество `s` в функции `Dijkstra` состоит из пар, в которых первый элемент – найденная длина пути до вершины, указанной как второй ее элемент. Изначально в качестве «бесконечного» значения алгоритма для поля `weight` вершин графа определено значение `INFweight`, равное сумме весов всех ребер, увеличенное на 1.

Пример 5. Дан неориентированный нагруженный граф с положительными весами. Найти кратчайший путь для заданной пары вершин.

```
#include <iostream>
#include <vector>
#include <set>
#include <stack>

using namespace std;

struct Vertex {
    int visited;
    int weight;
    int prev; //вершина, из которой пришли
```

```
    vector <pair<int, int>> neighbors;
};

typedef vector<Vertex> Graph;
Graph graph;

void Dijkstra(int from_vertex, int to_vertex) {
    struct SetVertex {
        int index; //номер вершины
        int weight; //найденный вес
        int prev; //вершина, из которой пришли
    };
    struct cmp { //сравнение для SetVertex
        bool operator() (SetVertex a, SetVertex
b) const {
            return a.weight < b.weight;
        }
    };
    set<SetVertex,cmp> s;
    SetVertex v;
    v.index = from_vertex;
    v.weight = 0;
    v.prev = -1;
    s.insert(v);
    while (!s.empty()) {
        v = *(s.begin());
        s.erase(s.begin());
        if (graph[v.index].visited == 1)
            continue;
        graph[v.index].visited = 1;
        graph[v.index].weight = v.weight;
        graph[v.index].prev = v.prev;
        if (to_vertex == v.index)
            return;
        for (auto i =
graph[v.index].neighbors.begin(); i !=
graph[v.index].neighbors.end(); ++i)
            if (graph[i->first].visited == 0) {
                SetVertex w;
                w.index = i->first;
```

```
                w.weight = (i->second)+v.weight;
                w.prev = v.index;
                s.insert(w);
            }
    }
}
int main() {
    setlocale(LC_ALL, "RUSSIAN");
    cout << "Введите количество вершин и
количество ребер графа: ";
    int n, m;
    cin >> n >> m;
    graph.resize(n);
    int INFweight = 1;
    cout << "\nВведите информацию о ребрах
графа\n";
    for (int i = 0; i < m; ++i) {
        int a, b, w;
        cin >> a >> b>>w;
        INFweight += w;
        graph[a -
1].neighbors.push_back(make_pair(b - 1,w));
        graph[b -
1].neighbors.push_back(make_pair(a - 1, w));
    }
    for (int i = 0; i < n; ++i) {
        graph[i].visited = 0;
        graph[i].weight = INFweight;
    }

    cout << "\nВведите номера вершин:";
    int a,b;
    cin >> a>>b;
    Dijkstra(--a,--b);
    if (graph[b].weight==INFweight)
            cout << "Путь не существует" << endl;
    else{
        stack<int> st;
        for (int i=b;i!=-1;i=graph[i].prev)
            st.push(i);
```

```
        cout << "Найденный путь:\n";
        while (!st.empty()) {
            cout << st.top()+1 << ' ';
            st.pop();
        }
    }
    system("pause");   return 0;
}
```

Для восстановления пути в предложенном решении используется поле `prev`, хранящее номер вершины, из которой пришли в данную. Для начальной вершины пути `prev` равен `-1`.

Для определения элемента множества `s` в функции `Dijkstra` введена структура `SetVertex`. В силу того, что множество в C++ есть упорядоченная структура, то для сравнения значений типа `SetVertex` определена операция сравнения (структура `cmp`), которая указана при описании `s`.

Контрольные вопросы и задания

1. Дан связанный неориентированный нагруженный граф. Найти две наиболее удаленные вершины.
2. Дан связанный неориентированный граф. Определить, есть ли в нем цикл.
3. Дан связанный неориентированный граф. Найти цикл максимальной длины.
4. Дан связанный неориентированный нагруженный граф. Найти минимальное остовое дерево.
5. Дан связанный неориентированный нагруженный граф. Найти кратчайшее расстояние между всеми парами вершин.
6. Дан связанный ориентированный граф. Определить, описывает ли данный граф отношение частичного порядка.
7. Дан связанный ориентированный граф. Определить, описывает ли данный граф отношение линейного порядка.
8. Дан неориентированный граф. Найти компоненту связанности с наибольшим количеством вершин.
9. Дан связанный неориентированный граф. Найти все мосты графа.

10. Дан связанный смешанный нагруженный граф. Найти все пути между заданной парой вершин.

11. Дан связанный неориентированный нагруженный граф. Найти центр графа.

12. Дан связанный ориентированный граф. Укажите минимальное множество ребер, смена направления которых приведет к тому, что между заданной парой вершин не будет пути.

13. Дан связанный неориентированный граф. Найти все циклы заданной длины.

14. Дано нагруженное дерево. Найти вершину, у которой максимальная разность весов ее поддеревьев минимальна.

15. Дан связанный неориентированный граф. Найти раскраску графа.

16. Дан связанный смешанный граф. Вершины графа покрашены одним их двух цветов. Определить количество вершин, до которых нет пути из заданной вершины, проходящих через `n` вершин заданного цвета.

17. Дан связанный неориентированный граф. Является ли он двудольным?

18. Дан связанный неориентированный граф. Найти клику графа.

19. Дан связанный неориентированный граф. Найти его точки сочленения.

20. Дан ориентированный граф. Является ли он сильносвязанным?

ЗАКЛЮЧЕНИЕ

Освещение теоретических и практических вопросов разработки программ на языке программирования C++, представленных в данном пособии, помогает студентам создавать программные приложения в среде Microsoft Visual Studio, понимать структуру программы, грамотно использовать структурные элементы языка и реализовывать алгоритмы с использованием структурных, перечислимых типов, линейных динамических информационных структур, двоичных деревьев, контейнеров, строк, файлов, методов представления и обработки графов.

При дальнейшем изучении языка программирования C++ следует уделить внимание классам, объектно-ориентированному подходу разработки приложений.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Основы программирования на языке С++: учеб.-метод. пособие / В. В. Подколзин, Е. П. Лукащик, Н. Ю. Добровольская, О. В. Гаркуша, С. Г. Синица, А. А. Полупанов, А. Н.Полетайкин, Р. Х. Багдасарян, А. В. Харченко, А. А. Михайличенко, А. В. Уварова. Краснодар, 2019.
2. Седжвик Р. Алгоритмы на С++. 2-е изд., испр. М., 2016.
3. Страуструп Б. Язык программирования С++ для профессионалов. М., 2006.
4. Синицын С. В., Хлытчиев О. И. Основы разработки программного обеспечения на примере языка С. 2-е изд., испр. М., 2016.
5. Белоцерковская И. Е., Галина Н. В., Катаева И. Е. Алгоритмизация. Введение в язык программирования С++. 2-е изд., испр. М., 2016.
6. Павловская Т. А.С/С++. Процедурное и объектно-ориентированное программирование: учебник. СПб., 2019.
7. Ишкова Э. А. Изучаем С++ на задачах и примерах. СПб., 2016.
8. Лубашева Т.В., Железко Б.А. Основы алгоритмизации и программирования: учеб. пособие. Минск, 2016.
9. Информатика и программирование: учеб. пособие / Р.Ю. Царев, А.Н. Пупков, В.В. Самарин, Е.В. Мыльникова. Красноярск, 2014.
10. Информатика: учеб. пособие. Тамбов, 2015.
11. Костюкова Н. И. Графы и их применение. Комбинаторные алгоритмы для программистов: учеб. пособие. М., 2010.
12. Сухан И. В. , Иванисова О. В., Кравченко Г. Г. Графы: учеб. пособие. 2-е изд. испр. и доп. Краснодар, 2015.
13. Алгоритмы на языке С++. URL: http://e-maxx.ru/algo/
14. C++ reference. URL:https://en.cppreference.com/w/

СОДЕРЖАНИЕ

Учебное издание

Авторы: Подколзин Вадим Владиславович
Полетайкин Алексей Николаевич
Лукащик Елена Павловна
Гаркуша Олег Васильевич
Синица Сергей Геннадьевич
Полупанов Алексей Александрович
Харченко Анна Владимировна
Уварова Анастасия Викторовна
Михайличенко Анна Александровна

МЕТОДЫ ПРОГРАММИРОВАНИЯ

Учебно-методическое пособие

Подписано в печать __.__.__. Печать цифровая.
Формат 60×84 1/16. Уч.-изд. л. 11,2.
Тираж 500 экз. Выход в свет __.__.__. Заказ №

Кубанский государственный университет
350040, г. Краснодар, ул. Ставропольская, 149.

Издательско-полиграфический центр
Кубанского государственного университета
350040, г. Краснодар, ул. Ставропольская, 149