МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ВОЛЖСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО
УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Игумнов А. Ю.

Основы программирования

(Язык C++)

Электронное учебное пособие

Волжский

2021

УДК 004.43(07)
ББК 32.97я73
И 286

Рецензенты:
заведующий кафедрой компьютерных наук и экспериментальной математики Волгоградского государственного университета
Клячин В.А.;
кандидат физико-математических наук, доцент кафедры информационных систем и математического моделирования Волгоградского института управления – филиала РАНХиГС при Президенте Российской Федерации
Савушкин А.Ю.

Издается по решению редакционно-издательского совета
Волгоградского государственного технического университета

Игумнов, А.Ю.
Основы программирования (язык C++) [Электронный ресурс] : учебное пособие / А.Ю. Игумнов ; Министерство науки и высшего образования Российской Федерации, ВПИ (филиал) ФГБОУ ВО ВолгГТУ. – Электрон. текстовые дан. (1 файл: 678 КБ). – Волжский, 2021. – Режим доступа: http://lib.volpi.ru. – Загл. с титул. экрана.
ISBN 978-5-9948-4059-7

Учебное пособие содержит изложение основ программирования на C++ в рамках курса «Программирование и основы алгоритмизации». Предназначено для студентов высших учебных заведений, обучающихся по направлению 15.03.04 «Автоматизация технологических процессов и производств».

Ил. 2, табл. 1, библиограф.: 4 назв.

ISBN 978-5-9948-4059-7

Оглавление

1. Краткие сведения о языке С и С++

Язык С начал разрабатываться в 1970-х годах. Наряду с возможностями использования конструкций языков верхнего уровня предоставляет непосредственный доступ к аппаратной части вычислительного устройства. Последнее обстоятельство способствовало широкому распространению языка, но также способствовало появлению в программах труднораспознаваемых ошибок. В последующих версиях языка (язык С++, С#) реализован ряд средств, обеспечивающих выявление ошибок на стадии составления программы.

2. Основные понятия программирования

Программа – описание последовательности действий.
Это описание может быть графическим (блок-схема) или текстовым.
По техническим причинам наиболее распространено текстовое описание (ввиду удобства приведения к машиночитаемому виду).
Программирование – деятельность человека по составлению программ.
Текст программы должен содержать обозначения операций (команды) и указания аргументов этих операций. Текст программы может содержать различного рода отладочную информацию и пояснения.
Язык программирования – это совокупность правил, по которым составляется текст программы.
Первичный порядок исполнения действий – линейный: действия исполняются в порядке расположения (написания/прочтения) их обозначений по строкам сверху вниз и в строке слева направо.
В большинстве языков программирования первичный порядок может быть изменен посредством специальных указаний (команды перехода).
Исполнение программы включает в себя ее преобразование в машинно-исполняемый вид (компиляция). Результатом такого преобразования является файл с машинными кодами команд, указанных в программе (exe-файл). Собственно исполнение программы заключается в обработке содержимого exe-файла вычислительным устройством.
В настоящее время разработаны средства, облегчающие процесс набора текста программы (текстовые редакторы), компиляции и отладки (программные оболочки).
В настоящем пособии рассматриваются только тексты программ безотносительно к программным оболочкам.
Исполнение программы удобно описывать в терминах действий исполнителя.
Исполнитель – это человек, организация, робот, исполняющий определенный заранее оговоренный набор действий. В техническом представлении исполнитель-робот может: читать текст программы, производить какие-то выкладки на листе бумаги (на школьной доске), рисовать на доске и стирать с доски какие-либо объекты (в виде прямоугольников) и т.п.

3. Общая структура текста программы в простейшем случае

В начале текста программы прописаны указания на специальные файлы, содержимое которых включает в себя различные функции (#include <iostream>) и учетные записи (using namespace std;) для контроля правильности употребления обозначений объектов. Дальнейший текст – основной – представляет собой совокупность именованных блоков-функций и различных программистских конструкций (цикл, выбор). Обязательная часть этих блоков – функция main. Исполнение программы начинается с этой функции. Помимо указанного в программе могут быть строки-комментарии, начинающиеся с символов // .

Помимо указанного, текст может быть разбит на блоки, обозначаемые фигурными скобками {,} . Блоки в тексте либо следуют один за другим, либо располагаются один внутри другого. Такие блоки используются для группирования элементарных операций в составные (например, тело цикла).

Минимальный текст программы с пустым основным текстом приведен ниже:

```
#include <iostream>
using namespace std;
int main()
{
}
```

Действия исполнителя: исполнитель просматривает текст в обычном порядке – по строкам сверху вниз и по строке слева направо – в поисках слова main. Найдя это слово, исполнитель определяет начало и конец main-блока (по фигурным скобкам). Далее исполнитель выполняет действия, прописанные в этом блоке. Так как в блоке ничего нет, то при исполнении этой программы ничего не произойдет.

В любом месте программы может быть размещен комментарий.

Комментарий – это текст, который не принимается во внимание исполнителем. Комментарий представляет собой произвольный (в смысле несоблюдения правил языка) текст. Содержательно – составитель программы пишет в комментарии пояснения к некоторой части программы (комментарий и соответствующий текст программы пишутся рядом друг с другом). Однострочный комментарий обозначается символом // в начале строки. Многострочный комментарий обозначается символом /* в начале текста и символом */ в конце текста.

Примеры «пустых» программ:

```
#include <iostream>
//Программа, состоящая из пустого блока
int main()
{ }
```

```
#include <iostream>
/*Программа, состоящая из нескольких
```

```
пустых блоков,
расположенных друг за другом.*/
int main()
{ }
{ }
{ }

#include <iostream>
//Описание: в главном блоке размещены подряд
//три блока, в третьем блоке – один вложенный блок.
int main()
{
        {       }
        {       }
        {       }
        {
                {       {       }       }
        }
}
```

Элементарной смысловой единицей языка является предложение – набор символов, завершаемый точкой с запятой. При выполнении программы исполнитель просматривает текст, вычленяя в нем предложения. При вычленении предложения производится его анализ на предмет соответствия правилам языка. В случае положительного результата предложение выполняется.

4. Объекты (переменные), основные типы данных

Исполнение программы представляет собой пересылку содержимого одних ячеек памяти в другие и их преобразование. Для правильного исполнения этих операций нужно соблюдать определенные правила согласованности объектов (например, нельзя применять математические операции к знакам алфавита).

Тип объекта – это указание множества, которому принадлежит содержимое объекта, и набора операций, производимых над элементами данного множества.

Основные типы языка C++ следующие:

Тип	Описание
int	Целочисленный тип 16 либо 32 бит
long int	Целочисленный тип 32 бит
short	Целочисленный тип 8 либо 16 бит
char	Символьный тип 8 бит
float	Вещественный тип 32 бит
double	Вещественный тип 64 бит

Для того чтобы иметь возможность работать с тем или иным типом данных необходимо задать переменную соответствующего типа. Эта процедура называется объявлением переменной и указывается следующим образом:

<тип переменной> < название переменной>; .

Например, в строке int arg; объявляется целочисленная переменная с названием arg.

Порядок исполнения этой операции следующий.

Исполнитель рисует на доске прямоугольник, надписывает его как arg и делает в своих учетных записях пометку, что содержимое этого прямоугольника должно трактоваться как целое число.

При составлении названия объекта можно использовать как верхний, так и нижний регистры букв латинского алфавита. При этом первым символом обязательно должна быть буква или символ подчеркивания ‘_’. Примеры:

Правильные названия	Неправильные названия
arg	&arg
cnt	$cnt
bottom_x	bottom-x
Arg	2Arg
don_t	don’t

В приведенных примерах переменные arg и Arg считаются разными, т.к. в языке C++ при объявлении переменных различаются верхний и нижний регистры.

В языке C++ переменные могут объявляться в любом месте программы.

5. Основные операторы и операции языка

Оператор вывода

Оператор вывода данных обозначается символом ″<<″, при использовании в инструкции *cout* он обеспечивает отображение информации на экране компьютера.

Пример. Вывод на экран текстового сообщения.

В рамках данного примера первые три строки программы рассматриваются без пояснений как обязательная часть.

```
#include <iostream>
using namespace std;
int main()
{
        cout<<"12345";
}
```

Результат исполнения программы: появление в окне вывода текста 12345.

В данном примере действием является вывод на экран набора символов. Командой на исполнение этого действия является команда cout<< . Аргументом является набор символов, указанный в двойных кавычках.

Оператор присваивания

Оператор присваивания представляет собой указание на пересылку вида a=b; .

Порядок исполнения: исполнитель находит на доске прямоугольники a и b, содержимое прямоугольника a стирает и записывает вместо стертого содержимое прямоугольника b. Перед исполнением названия a и b проверяются на согласованность типов (объекты должны занимать одинаковое количество байтов в памяти).

Варианты использования оператора присваивания:

вместо названия объекта-источника может указываться непосредственно содержимое, оператор присваивания может сочетаться с арифметическими действиями и с объявлением переменной.

Примеры:

```
i=20;
i=a+20;
int i=20
```

Операция приведения типов

Рассмотрим пример. Допустим, что имеются две переменные разного типа: short A = 10; long B; и выполняется оператор присваивания B = A; .

В результате переменная B будет иметь значение 10, при исполнении присваивания будет выполнено автоматическое преобразование типов и потери данных не происходит. В другом случае, когда

```
float A = 8.7; int B; B = A;
```

при исполнении присваивания произойдет потеря данных, т.к. целое число не может представлять числа, стоящие после десятичной точки. В результате переменная A будет иметь значение 8. Для корректного преобразования одного типа данных в другой используется операция приведения типов, имеющая следующий синтаксис:
<название переменной1> = (тип_данных)<название переменной2>; .
Например, B = (long)A; .

Оператор ввода. Подключение кодировки букв русского алфавита

Оператор ввода с клавиатуры имеет вид cin >> ⋯.; , где многоточие означает обозначение простого или составного объекта символьного типа.
Подключение кодировки русского алфавита производится следующими операциями (см. пример):
#include <Windows.h>, SetConsoleCP(1251), SetConsoleOutputCP(1251).

```
//Вывод/ввод русских букв
#include <iostream>
#include <Windows.h> // Обязательно для SetConsoleCP() и
//SetConsoleOutputCP()
using namespace std;
int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	char name[12];
	cout << "Введите свое имя: ";
	cin >> name;
	cout << "Ваше имя " << name;
	return 0;
}
```

Описание выполнения программы.
Команды SetConsoleCP(1251), SetConsoleOutputCP(1251) означают переключение на кодировку с русским алфавитом при исполнении операций ввода и вывода. Выполняя инструкцию char name[12], исполнитель чертит на листе бумаги ряд из 12 клеток, в которые должны записываться символы, и обозначает этот ряд как name. Далее на экран выводится фраза-приглашение. Затем: текст, набранный на клавиатуре, переписывается в ряд клеток name; на экран выводится фраза «Ваше имя» и содержимое ряда name. В заключение командой return в операционную систему посылается сигнал 0, означающий нормальное завершение программы.

6. Арифметические операции. Особенность деления целых чисел

Элементарные математические операции следующие: сложение, вычитание, умножение и деление. Очередность выполнения операций определяется по правилам обычной арифметики: умножение, деление; сложение, вычитание. Для изменения очередности используются скобки.

Пример. Пусть в программе заданы две переменные int a, b; с начальными значениями a=4; b=8; тогда операции сложения, вычитания, умножения и деления будут выглядеть следующим образом:

```
int c; c = a+b; //сложение двух переменных
c = a-b; //вычитание
c = a*b; //умножение
c = a/b; //деление.
```

Указанные операции можно выполнять также и с конкретными числами, например, c = 10+5; c = 8*4; float d; d = 7/2; .

Результатом первых двух арифметических операций будут числа 15 и 32 соответственно, но при выполнении операции деления в переменную d будет записано число 3, а не 3,5, поскольку число 7 по правилам языка C++ будет истолковываться как целочисленная величина, и, следовательно, не может содержать дробной части.

Для правильного выполнения деления одного числа на другое следует использовать такую запись: d = 7.0/2; или d = (float)7/2; .

В первом случае вещественное число делится на два и вещественный результат присваивается вещественной переменной d. Во втором варианте выполняется приведение типов: целое число 7 приводится к вещественному типу float, а затем делится на 2.

Операция деления по модулю

Тип аргументов этой операции – только целочисленный.

Результатом является остаток от деления одного целого числа на другое. Так выражение int a = 13 % 4; означает, что число 13 делится по модулю 4. Так как 13=4*3+1, то значение a будет равно 1.

Ниже приведен полный пример программы с операциями целочисленного деления и деления по модулю:

```
#include <iostream>
using namespace std;
//Описание: Рассматривается операция
//деления по модулю.
int main()
{int x,y;
        x=12; y=4;
        cout<<x/y;
//Результат операции - неполное частное.
        cout<<"\n";
```

```
        cout<<x%y;
//Результат операции – остаток от деления.
return 0;
}
```
(return 0 – сигнал для операционной системы о нормальном завершении программы)

Инкремент и декремент

Инкремент (декремент) означает увеличение (уменьшение) значения переменной на 1. Данные операторы могут быть записаны в виде:

```
i++; // операция инкремента (постфиксная форма)
++i; // операция инкремента (префиксная форма)
i--; // операция декремента (постфиксная форма)
--i; // операция декремента (префиксная форма).
```

В следующем примере показана разница между первой и второй формами записи:

```
int i=10,j=10;
int a = i++; //значение a = 10; i = 11;
int b = ++j; //значение b = 11; j = 11;
```

7. Типовые управляющие предложения (управляющие конструкции)

Управляющие предложения – это средство изменения порядка выполнения команд по сравнению с порядком их расположения в тексте программы.
В записи этих конструкций используются объекты и выражения типа bool.
Тип **bool** – это множество {0,1}, на котором определены логические операции – унарные и бинарные. Наглядное представление объекта такого типа: прямоугольник с названием, в котором записано либо 0, либо 1. Синонимами этих значений являются слова false, true соответственно. Логические операции определяются следующими таблицами истинности:
операция отрицания:

p	!p
0	1
1	0

операции логического сложения и умножения:

p	q	p || q	p && q
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Конструкции выбора

Конструкция если – то – иначе
Запись предложения на языке С++ следующая.
if(условие)
{Действия$_{да}$}
else
{Действия$_{нет}$}
Здесь условие представляет собой такое математическое выражение отношения равенства или неравенства само по себе или в сочетании с выражениями, содержащими логические операции, про которое имеет смысл говорить, истинно оно или ложно. Соответственно 1 или 0 будет значением истинности этого выражения. Также в роли выражения может выступать логическая переменная или константа. Например:

выражение	5=3	5>3	(5<3)&&(1=1)
значение истинности	0	1	0

Значение истинности выражения, содержащего переменные (например, a=b+5), зависит от значений входящих в него переменных (a, b).
Если блок {Действия$_{нет}$} пустой, то указанная конструкция выбора может быть записана в сокращенной форме:
if(выражение)
{Действия$_{да}$}

Условие, заданное без использования логических операций, называется простым, с использованием логических операций – составным.

Порядок действий исполнителя:

- прочитав слово if исполнитель определяет границы блока {Действия$_{да}$} и блока {Действия$_{нет}$} (при наличии слова else);
- исполнитель читает выражение при слове if и определяет его значение истинности;
- если выражение имеет значение истинности 1, то
  - в конструкции со словом else: исполнитель выполняет команды блока {Действия$_{да}$}, пропускает слово else и блок {Действия$_{нет}$} и выполняет команды написанные после блока {Действия$_{нет}$}
  - в конструкции без слова else: исполнитель выполняет команды блока {Действия$_{да}$} и далее исполняет команды, написанные ниже по тексту;
- если выражение имеет значение истинности 0, то
  - в конструкции со словом else: исполнитель пропускает блок {Действия$_{да}$}, выполняет блок {Действия$_{нет}$} и далее команды ниже по тексту
  - в конструкции без слова else: исполнитель пропускает блок {Действия$_{да}$} и далее выполняет команды ниже по тексту.

Пример программы с конструкцией выбора с простым условием:

```
#include <iostream>
using namespace std;
//Описание: рассмотрение управляющей конструкции если – то – иначе
int main()
{float x,y;
        x=3.5;
        if(x>0)
                {cout<<"+++++"; }
                else
                {cout<<x;    }
        x=-3.5;
        if(x>0)
                {cout<<"+++++"; }
                else
                {cout<< "x="<<x; }
 return 0;
}
```

Описание фрагмента программы:

```
if(x>0)
                {cout<<"+++++"; }
                else
                {cout<<x;    }
```

- При положительном значении x на экран выводятся знаки +++++, при отрицательном значении – на экран выводится само значение x.
- При исполнении первой конструкции значение x положительно, на экран будет выведено +++++.
- При исполнении второй конструкции значение x отрицательно и на экран будет выведено число -3.5.

Фрагмент программы с конструкцией выбора с составным условием.
Описание: если выполнено хотя бы одно из условий a=5 или b=8, то на экран выводится символьный набор +++++.

```
if((a=5)||(b=8))
{cout<<"+++++"; }
```

В любом из блоков {Действия$_{да}$}, {Действия$_{нет}$} могут быть любые конструкции, допустимые по правилам языка программирования, в частности, также конструкция выбора. Для примера рассмотрим программу для решения линейного уравнения ax=b. Если a ≠0, то решение единственное: x=b/a. Если a=0 и b≠0, то решений нет. Если a=0 и b=0, то решением является любое число. Программа может быть такой (значения a, b заданы в тексте программы и обозначены многоточием):

```
#include <iostream>
#include <Windows.h> // Обязательно для SetConsoleCP() и
//SetConsoleOutputCP()
using namespace std;
int main()
{
SetConsoleCP(1251);
SetConsoleOutputCP(1251);
float a,b;
a=…;
b=…;
if (a!=0)
{cout<<"x="; cout<<x;}
if(a=0)
  {if (b=0)
    {cout<<"x- любое";}
     else
    {cout<<"решений нет";}
   }
}
```

Конструкции цикла

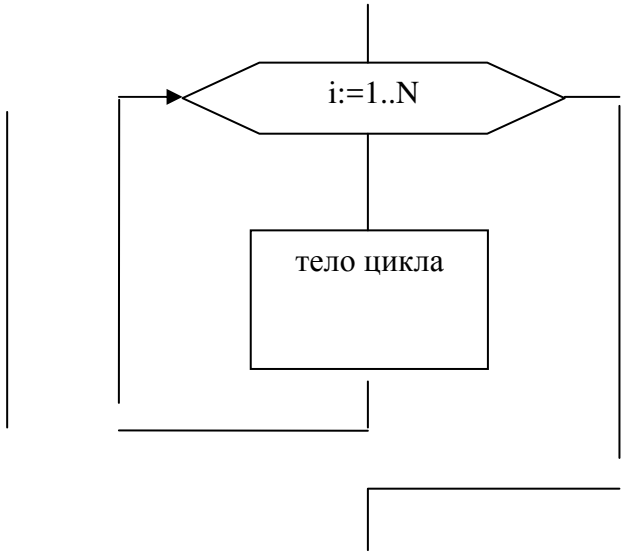
Циклом называется слитная группа команд, которая может выполняться несколько раз (особый случай – бесконечно много раз) подряд при

выполнении определенного условия. Это условие называется условием выполнения цикла. Способ обозначения условия называется *заголовок цикла*. Остальная часть команд цикла называется *тело цикла*.

Основных видов цикла – два: цикл по перечислению и цикл по условию (цикл с предусловием).

Цикл по перечислению

Блок схема цикла по перечислению имеет следующий вид:



Здесь: i – счетчик цикла (переменная целочисленного типа), блок – заголовочный блок, число N≥i (N – также переменная целочисленного типа). Порядок исполнения цикла следующий:

1) при первом исполнении заголовочного блока счетчику цикла придается значение 1 и проверяется выполнение условия принадлежности значения i целочисленному отрезку [1,N]

2) если это условие выполнено, то исполняется блок «тело цикла», после чего значение счетчика увеличивается на 1

3) исполняется заголовочный блок при новом значении счетчика; если это значение принадлежит отрезку [1,N], то снова исполняется шаг 2), если не принадлежит, то исполняются команды, указанные ниже по блок-схеме.

В теле цикла может быть команда **выход из цикла**. Ее исполнение заключается в переходе к исполнению команд, указанных по блок-схеме ниже тела цикла.

Соответствующий блок-схеме программный текст следующий:

```
for(i=1; i<=N; i:=i+1)
{……}
```

(многоточием обозначены команды тела цикла).

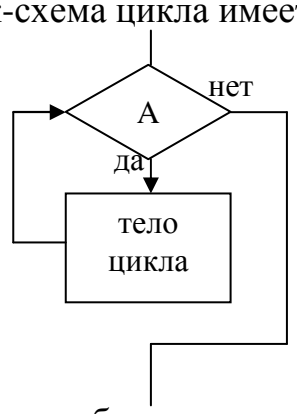
Далее из всех целочисленных типов будет использоваться только тип long int (диапазон: −2147483647. . 2147483647) .

Пример. Найти сумму 10 чисел вводимых с клавиатуры. После ввода 10-го числа исполнение программы должно завершиться с выводом значения суммы на экран. В программном фрагменте показан возможный вариант решения.

```
s=0;
for(i=1;i<=10;i=i+1)
{cout<<"a"<<i<<"= "; cin >> a;
s=s+a;}
cout << "сумма="<<s<< "\n";
```

Цикл по условию

Блок-схема цикла имеет следующий вид:



Описание блок-схемы

В блоке ветвления записано условие A, при истинности которого тело цикла выполняется. Правила составления условия A те же, что и для команд ветвления. Если при очередной проверке условия A оно оказывается ложным, то следующей после выполнения блока ветвления выполняется команда, написанная ниже по схеме после блока «тело цикла». Чтобы цикл завершился, тело цикла должно быть составлено так, чтобы при очередном его выполнении условие A стало ложным.

Соответствующий программный текст:

```
while(A)
{……..}
```

Описание блок-схемы и программного текста

Обозначением блока ветвления является слово while. Тело цикла указано многоточием в фигурных скобках.

Пример. Суммировать числа, вводимые с клавиатуры, до тех пор, пока не будет введено число большее 9.5. Последнее в сумму не включать. Решение дано в следующем фрагменте программы:

```
s=0; a=0;
while(a<=9.5)
{cout<<"a= "; cin >> a;
s=s+a;}
cout << "сумма="<<s<< "\n";
```

Конструкция (команда) безусловного перехода

Команда имеет вид: goto L, где L – некоторый идентификатор, расположенный в тексте программы. При этом идентификатор L вне команды должен быть с двоеточием.

Порядок исполнения команды следующий: исполнитель, прочтя текст команды, находит в текущем блоке запись «L:» и продолжает исполнение команд с той строки где написана указанная запись, при этом пропустив ее саму. **Пример**:

```
#include <iostream>
using namespace std;
//Описание:
//Бесконечный цикл с
//использованием безусловного перехода
int main()
{M: cout<<"*";
        goto M;
}
```

Описание программы ясно из комментария. Результатом исполнения будет безостановочный вывод на экран символа "*".

8. Обработка массивов

Определения.

Множество I={k,k+1,…,k+l}, состоящее из последовательных целых чисел, называется целочисленным отрезком. Количество элементов в нем обозначается |I (|I|=l+1).

Отображение φ:I→A, где A – некоторое множество, называется одномерным рядом элементов множества A. Элементы ряда обозначаются a_i.

Отображение $\varphi:I_1xI_2x\ldots xI_r\rightarrow A$ прямого произведения отрезков $I_1,I_2,\ldots,I_r$ в множество A называется многомерным рядом. Элементы ряда обозначаются $a_{ij\ldots}$.

В программировании ряды называются массивами (неправильный перевод с англ. слова array). В языке C++ элементы одномерного массива обозначаются как a[i], многомерного – a[i][j]… . Значения индексов начинаются с 0.

Одномерный массив объявляется следующим образом: T a[n], где T – называние типа, n – длина массива (количество элементов в нем). Наглядное представление – ряд ячеек памяти a[0] a[1] … a[n-1]. Содержимое ячейки a[i] называется значением элемента a[i].

Двумерный массив объявляется следующим образом: T a[n] [m]. Его наглядное представление – таблица, состоящая из n-1 строки и m-1 столбца.

Массивы более высокой размерности определяются аналогично.

С элементами массива возможны все операции, допустимые типом T. Аргументы операций указываются названием массива и номером (набором номеров) элемента в нем.

Типовыми задачами обработки массива являются, в частности, следующие.

Пример. Дано: одномерный числовой массив a. Требуется найти в массиве элемент со значением b и индекс его элемента. Если такого элемента нет, вывести на экран соответствующее сообщение.

Программный текст может быть следующим.

```
#include <iostream>
#include <Windows.h> // Обязательно для SetConsoleCP() и SetConsoleOutputCP()
using namespace std;
int main()
{
    1. SetConsoleCP(1251);
    2. SetConsoleOutputCP(1251);
    3. double a[5], b;
    4. long int i,p,flazhok;
    5. a[0]=3.5; a[1]=5.2; a[2]=-1.3; a[3]=7.4; a[4]=7;
    6. b=5.2;
    7. flazhok=0;
    8. for(i=0;i<5;i=i+1)
    9. {if(a[i]==b)
   10.{p=i;flazhok=1;break;}
```

```
    11.}
    12.if(flazhok==1)
    13.{cout << "Индекс искомого элемента: "<<p;}
    14.else
    15.{cout << "Элемент не найден";}
    16.return 0;
}
```

Описание программы

- В строках 1-2 описана перекодировка.
- В строках 3-4 объявлены массив и переменные. Назначение переменных:
- b – искомое значение,
- i – счетчик цикла,
- p – значение индекса найденного элемента,
- flazhok – флажковая переменная, имеет значение 0, если элемент не найден, значение 1, если элемент найден.
- В строках 5-7 задаются значения элементов массива, искомое значение и начальное состояние флажка – 0.
- В строках 8-10 описан цикл: элементы массива просматриваются подряд, начиная с начала (строка 8); если для очередного элемента обнаружено совпадение его значения с b (строка 9), то значение индекса этого элемента записывается как значение переменной p, флажок устанавливается в 1 и цикл прекращается (строка 10); в противном случае просматривается следующий элемент массива.
- По окончании цикла проверяется состояние флажка (строка 12):
- если флажок установлен (значение 1), то на экран выводится значение индекса найденного элемента – как значение переменной p – и поясняющий текст (строка 13);
- в противном случае (строка 14) на экран выводится фраза «Элемент не найден”).

Содержимое диалогового окна в случае b=5.2:

```
Индекс искомого элемента: 1
--------------------------------
Process exited after 0.8692 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Содержимое диалогового окна в случае b=6:

```
Элемент не найден
--------------------------------
Process exited after 1.194 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Пример. Поиск максимального элемента массива.

Дано: массив a. Требуется найти значение максимального элемента массива и его индекс.

Программный текст:

```
#include <iostream>
#include <Windows.h> // Обязательно для SetConsoleCP() и SetConsoleOutputCP()
using namespace std;
int main()
{
    1. SetConsoleCP(1251);
    2. SetConsoleOutputCP(1251);
    3. double a[5], m;
    4. long int i,p;
    5. a[0]=3.5; a[1]=5.2; a[2]=-1.3; a[3]=7.4; a[4]=7;
    6. p=0; m=a[p];
    7. for(i=0;i<5;i=i+1)
    8. {if(a[i]>m)
    9. {p=i;m=a[p];}
    10.}
    11.cout << "Максимальный элемент: a["<<p<<"]="<<a[p];
    12.return 0;
}
```

Описание программы

- В строках 1-5 описана вводная часть. Назначение переменных:
- i – счетчик цикла; p – значение индекса текущего максимального элемента; m – значение максимального на момент просмотра элемента.
- Перед началом просмотра массива (строка 6) за максимальный элемент принимается a[0]; значение индекса запоминается как значение p, значение самого элемента – как m.
- При просмотре каждого элемента в цикле (строки 7-10), начиная с a[0], просматриваемый элемент сравнивается с текущим значением максимума (строка 8): если элемент больше текущего значения, то значение его индекса и значение самого элемента записываются как p и m соответственно (строка 9).
- По окончании цикла (строка 11) на экран выводится значение индекса максимального элемента и значение самого максимума с поясняющим текстом.

Содержимое диалогового окна по завершении исполнения:

```
Максимальный элемент: a[3]=7.4
--------------------------------
Process exited after 0.8249 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Пример. Сортировка массива по убыванию.

Общее описание алгоритма. Сортируемый массив A представляется разделенным на две части A'A", размер которых изменяется по ходу исполнения сортировки: часть A' увеличивается, часть A" уменьшается.
Перед исполнением сортировки A' – пустое множество, A"=A.
Далее:

- в массиве A ищется максимальный элемент и переставляется с элементом a[0] – начальным элементом массива A (он же есть часть A"); за A' принимается a[0], за A" – остальная часть массива.
- в A" ищется максимальный элемент и переставляется с его начальным элементом – a[1]; за A' принимается a[0], a[1], за A" – остальная часть массива.
- И так далее.

9. Подпрограммы (функции) в языке С++. Упрощенное описание. Правила видимости переменных

Подпрограмма в языке С++ – это обособленный именованный блок команд со следующей структурой (описание подпрограммы):
заголовок
{список команд}

Обособленность означает следующее:

- такие блоки расположены вне блока main
- исполнение команд данного блока возможно только при прочтении исполнителем команды, инициирующей исполнение подпрограммы, в списке команд другого блока или этого же самого
- по исполнении последней (по очередности исполнения) команды подпрограммы исполнитель продолжает исполнение с команды, указанной непосредственно после того места, где расположена инициирующая команда.

В языке С++ подпрограммы называются функциями. Объявление подпрограммы называется объявление функции.
Наиболее полная запись заголовка имеет вид:
Т название функции (T_1 x_1, T_2 x_2, …, $T_к$ $x_к$), где x_1, x_2, …, $x_к$ – наименования переменных, T_1, T_2… , $T_к$ соответственно – их типы, Т – тип значения функции (тип ответа). Указанное оформление списка переменных называется передачей параметров по значению.
Обязательная часть заголовка функции – название (составляется по тем же правилам, что и идентификатор переменной). Функции делятся на категории: вырабатывающие/не вырабатывающие ответ (значение), с аргументами/без аргументов.
В соответствии с этим заголовок функции может содержать либо не содержать список аргументов (аналог аргументов функции в математическом смысле). Тип ответа указывается всегда; если значение функцией не вырабатывается, то указывается тип void (пустой).
Командой, инициирующей исполнение функции является запись ее заголовка без указания типа значения.
Наличие в заголовке функции непустого типа значения означает, что заголовок функции (без указания типа значения) может записываться в составе различных выражений как переменная (с учетом непротиворечивости типа этой переменной и типов других переменных, входящих в данное выражение).
С некоторыми упрощениями структуру программы с функциями можно представить в следующем виде:
Вводная часть
……..
Объявление функции1

Объявление функции2

……..
Объявление функцииК

Блок main

(Здесь «Вводная часть» – это указания по подключению заголовочных файлов и иные записи.)
Структура списка команд такая же, как и в блоке main. Последней исполняемой командой функции является команд return (выход из подпрограммы). Если функция вырабатывает значение, то команда имеет параметр – переменная, значение, выражение соответствующего типа. Если функция не вырабатывает значение, то команда параметров не имеет.
В настоящие время составлены библиотеки функций различных областей применения (например, математические). Указания по подключению файлов с библиотеками прописываются в вводной части.

Простейшие примеры.
Пример. Функция без аргументов, не вырабатывающая ответ.

```
#include <iostream>
using namespace std;
1. void Pechat()
2. {cout<<"********"; return;}
3. int main()
4. {Pechat(); return 0;}
```

Описание программы.

- В строке 1 написан заголовок функции, название функции – Pechat().
- В строке 2 описан блок функции, состоящий из двух команд.
- В строке 3 написан заголовок основной функции.
- В строке 4 приведены команды основного блока.

Описание исполнения.

1. Исполнитель просматривает текст и находит main-блок.
2. Исполнитель читает слово Pechat как наименование команды, запоминает его местонахождение и ищет слово в заголовках блоков в программе.
3. Найдя слово Pechat (строка 1) исполнитель выполняет команды блока: сначала печатает ряд символов «*», затем, исполняя команду return (в строке 2), исполнитель возвращается к просмотру списка команд основного блока сразу после команды «Pechat();» в строке 4.
4. Исполнитель выполняет команду «return 0;» в строке 4 и заканчивает работу.

Результат исполнения: печать на экране нескольких символов «*».

Пример. Функция с аргументами, не вырабатывающая ответ.

```
#include <iostream>
```

```
#include <Windows.h>
1. void summa(double x, double y, double z)
2. {cout<<x+y+z; return;}
3. int main()
4. {summa(3,-2.5,1.5); return 0;}
```

Описание программы аналогично предыдущему примеру.

Описание исполнения.

1. Исполнитель, найдя основной блок, читает запись summa(3,-2.5,1.5); и запоминает ее местонахождение.
2. Найдя в заголовках блоков слово summa, исполнитель проверяет соответствие количество и тип аргументов при слове summa в строке 1 и строке 4. Так как эти списки согласованы (по типу и количеству аргументов), исполнитель приравнивает значения x,y,z из строки 1 соответственно числам 3,-2.5,1.5 из списка в строке 4, и приступает к выполнению первой команды блока функции summa - cout<<x+y+z; .
3. Вычисляя значение выражения x+y+z исполнитель берет значения, сопоставленные переменным из списка в строке 2. Вычислив значение суммы, исполнитель печатает его на экране и выполняет команду «return;» в строке 2. Дальнейший ход исполнения такой же как в предыдущем примере.

Пример. Функция с аргументами, вырабатывающая ответ.

```
#include <iostream>
#include <Windows.h>
//функция с параметрами, вырабатывающая ответ
using namespace std;
1. double summa(double x, double y, double z)
2. {return x+y+z;}
3. int main()
4. {double a;
5. a=summa(3,-2.5,1.5)+summa(8,-2.5,5.5);
6. cout<<a;
7. return 0;}
```

Порядок исполнения.

Исполнитель находит основной блок (строка 3).

Исполняя строку 4 исполнитель формирует объект «a», содержимым которого должно быть вещественное число.

Исполнение строки 5 происходит следующим образом.

Исполнитель читает левую часть выражения (до знака =) и распознает обозначение объекта a, как уже сформированного объекта;

читает слово summa, находит блок с таким названием и определяет, что это подпрограмма вырабатывающая ответ типа вещественного типа;

проверяет согласованность списков переменных при названии функции в строке 5 и в строке 1, и подставляет числовые значения 3,-2.5,1.5 вместо double x, double y, double z в строке 1;

далее исполняется функция: в строке 2 вместо обозначений x, y, z подставляются числа 3, -2.5, 1.5 соответственно; вычисляется их сумма (число 2) и подставляется в строку 5 в качестве значения summa(3,-2.5,1.5).
Аналогично обрабатывается вторая запись summa(8,-2.5,5.5) в строке 5 (результат – 11).
По-прежнему в строке 5 производится сложение 2 и 11 и результат (13) записывается в объект a.
В строке 6 производится вывод на печать содержимого объекта a.
Исполнение программы завершается строкой 7.

При более сложных вычислениях значение функции записывается в некоторую переменную, а ее название записывается как аргумент операции return. В этом случае текст функции мог бы быть следующим:

```
1. double summa(double x, double y, double z)
2. {double s;
3. s= x+y+z;
4. return s;}
```

Правила видимости переменных. Локальные и глобальные переменные.
Правилами видимости переменных называются правила, по которым определяется соответствие между названием переменной и ее местом в памяти. В упрощенном виде правила выглядят так.
Каждый блок имеет индивидуальное название – либо явное, либо неявное (если блок обозначен только фигурными скобками).
К каждому обозначению объекта скрытно дописывается (как индекс) название блока в котором объявлен объект. Обозначение объекта вместе со скрытым индексом называется полным наименованием объекта.
Объекты считаются различными если различны их полные наименования.
Объявление объекта должно предшествовать его использованию.
Объекты, объявленные в блоках, называются локальными. Объекты, объявленные вне блоков, называются глобальными.
Пример программы с одинаковыми обозначениями объектов в разных блоках.

```
1. double summa(double x, double y, double z)
2. {double  a;
3. a= x+y+z;
4. return a;}
5. int main()
6. {double a;
7. a=summa(3,-2.5,1.5)+summa(8,-2.5,5.5);
8. cout<<a;
9. return 0;}
```

Полное наименование объекта a:
в блоке summa - a_{summa} ; в блоке main - a_{main} .

Пример. Программа с глобальными и локальными переменными.
Здесь переменная x упоминается несколько раз:
в строке 1 x – это глобальная переменная, в строках 2 и 3 x – это локальная переменная (внутри подпрограммы). При исполнении строки 7 на экран будет выведено значение 3.

```
#include <iostream>
using namespace std;
1. double x=3;
2. double summa(double x, double y, double z)
3. {return x+y+z;}
4. int main()
5. {double a;
6. a=summa(3,-2.5,1.5)+summa(8,-2.5,5.5);
7. cout<<x;
8. return 0;}
```

10. Передача массива в функцию

Простейший способ передачи массива в функцию – указание его типа, наименования и размера в списке параметров. При этом значения элементов массива могут быть изменены операциями, указанными в теле функции.

Пример.

```
#include <iostream>
#include <Windows.h>
using namespace std;
1. void f(double a[3])
2. {a[0]=0; a[1]=0;a[2]=0;return;}
3. int main()
4. {double a[3];
5. a[0]=10;  a[1]=20; a[2]=30;
6. f(a);
7. cout<<a[0];cout<<a[1];cout<<a[2];
8. return 0;}
```

Описание. Функция f описана в строках 1 и 2. Ее действие заключается в обнулении элементов массива a. В основной программе элементы массива заданы как ненулевые (строка 5). При исполнении функции (строка 6) элементы массива обнуляются, что проверяется выводом их значений на экран (строка 7).

11. Обработка строк

Строка – это символьный массив, завершающийся нулем. Пример объявления строки:

char a[11] .

Результатом обработки этого объявления исполнителем является ряд в машинной памяти из 11 ячеек. В этот ряд может быть записан набор из не более чем 10 символов. После символьного набора автоматически (т.е. без явного указания этого действия в программе) записывается значение 0.

Пример. Простейший способ ввода строки с клавиатуры – использование команды cin.

```
#include <iostream>
#include <Windows.h> //
using namespace std;
int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	char str[80];
	cout << "Введите строку: ";
	cin >> str;
	cout << "Введенная строка " << str;
	return 0;
}
```

В объявлении строки в данном примере указано число 80. Т.е. длина строки не может превышать 79 символов. Ввод с клавиатуры будет прерван при получении пробельного символа – пробел, символы табуляции, переход на новую строку.

Пример ввода строк, содержащих пробелы. Здесь используется функция gets (требуется подключение заголовочного файла cstdio).

```
#include <iostream>
#include <cstdio>
#include <Windows.h> //
using namespace std;
int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	char str[80];
	cout << "Введите строку: ";
	gets(str);
	cout << "Введенная строка " << str;
	return 0;
}
```

В этом случае ввод строки прекращается при нажатии на клавишу «Enter». Если с клавиатуры будет введено больше 79 символов, то память за пределами массива str будет затерта «лишними» символами.

Некоторые библиотечные функции обработки строк.
Вызов функции: strcpy(*получатель*, *отправитель*).
Описание: содержимое строки *отправитель* копируется в строку *получатель*.
Вызов функции: strcat(s1,s2).
Описание: строка s2 дописывается к строке s1.
Вызов функции: strcmp(s1,s2) – функция, вырабатывающая ответ.
Описание: если строки равны, то значением функции будет 0; если s1 лексикографически (т.е. в алфавитном порядке) больше s2, то ответ – положительное число, если меньше – то отрицательное.
Вызов функции: strlen(s) – функция, вырабатывающая ответ.
Описание: ответ – количество символов в строке s.
Пример обработки строк.

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <Windows.h>
SetConsoleOutputCP()
using namespace std;
int main()
{
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);
    1.  char stroka1[80], stroka2[80];
    2.  strcpy(stroka1,"abcd");
    3.  strcpy(stroka2,"123456789");
    4.  cout << "Длины строк: "<< strlen(stroka1);
    5.  cout << " ";
    6.  cout << strlen(stroka2)<<'\n';
    7.  if(!strcmp(stroka1,stroka2))
    8.  cout<<"строки совпадают"<<'\n';
    9.  else
    10.cout<<" строки не совпадают "<<'\n';
    11.strcat(stroka1,stroka2);
    12.cout << stroka1<<'\n';
    13.strcpy(stroka2,stroka1);
    14.cout<<"stroka1="<<stroka1<<" stroka2="<<stroka2<<'\n';
    15.if(!strcmp(stroka1,stroka2))
    16.cout<<"Строки 1 и 2 одинаковы"<<"\n";
    17.return 0;
}
```

Описание программы.

В строке 1 объявляются строковые переменные длиной 79 знаков с названиями stroka1 b stroka2. После исполнения строк 2 и 3 значения переменных следующие: stroka1=,"abcd", stroka2="123456789". При исполнении команд в строках 4-6 длины строковых переменных выводятся на экран. В строках 7-10 определяется, равны ли длины строковых переменных, и соответствующие заключения выводятся на экран. В строке 11 к массиву stroka1 дописывается массив stroka2 и значения этих строковых переменных становятся следующими:
stroka1=”abcd123456789” (изменилось), stroka2=123456789 (осталось прежним). Объединенная строка выводится на экран (строка 12 текста программы). В строке 13 значение строковой переменной stroka1 копируется в строковую переменную stroka2, в результате чего значения строковых переменных становятся одинаковыми: ”abcd123456789”. В строке 14 эти значения выводятся на экран. В строке 15 проверяется совпадение значений строковых переменных, и в случае совпадения на экран выводится фраза-заключение (строка 16).
Содержимое диалогового окна после выполнения программы:

```
Длины строк: 4 9
строки не совпадают
abcd123456789
stroka1=abcd123456789 stroka2=abcd123456789
Строки 1 и 2 одинаковы

--------------------------------
Process exited after 0.91 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Пример. Передача строки в функцию.
Строка представляет собою массив символов (тип char). Передается в функцию так же, как массив иного типа.
Пример.

```
#include <iostream>
#include <Windows.h>
using namespace std;
void f(char a[3])
{a[0]='z'; a[1]='q';a[2]='r';  return;}
int main()
{char a[3];
 a[0]='a';  a[1]='b'; a[2]='c';
 f(a);
 cout<<a[0];cout<<a[1];cout<<a[2];
 return 0;}
```

Действие функции заключается в изменении элементов строки. В основной программе задано значение строки (посимвольно). После исполнения функции строка посимвольно выводится на экран.

Из строк можно составлять массивы, как показано в следующем примере.
Пример. Имитация телефонной книжки.

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <Windows.h> // Обязательно для SetConsoleCP() и SetConsoleOutputCP()
using namespace std;
int main()
{
    1. SetConsoleCP(1251);
    2. SetConsoleOutputCP(1251);
    3. long int i;
    4. char stroka[80];
    5. char spisok[6][80]=
    6. {"Фамилия1", "Номер1",
    7. "Фамилия2", "Номер2",
    8. "Фамилия3", "Номер3"};
    9. cout << "Введите фамилию: ";
    10.cin >> stroka;
    11.for(i=0;i<6;i=i+2)
    12.if(!strcmp(stroka,spisok[i]))
    13.{cout<<"Фамилия найдена"<< "Телефон="<<spisok[i+1]<<"\n"; break;}
    14.if(i==6) cout<<"Фамилия не найдена";
    15.return 0;
}
```

Описание программы. Телефонная книжка представлена как массив из шести строк длиной до 79 символов. Запись в телефонной книжке – это пара строк с номерами 0-1, 2-3 и т.д. В первой строке пары записана фамилия, во второй – телефонный номер. В строках 9-10 введенная с клавиатуры фамилия запоминается как значение переменной stroka. В строках 11-13 в цикле просматриваются первые компоненты строковых пар. В случае совпадения просматриваемого содержимого с введенным с клавиатуры, на экран выводится подтверждающее сообщение и соответствующий номер телефона. После цикла – в строке 14 – проверяется, произошел ли выход за пределы массива. В этом случае выводится сообщение об отрицательном результате поиска.

12. Структуры

Структура – множество именованных объектов, объединенных в одно целое. Объекты называются компонентами (элементами) структуры. Объединенность в одно целое означает наличие общего названия у этой группы объектов и возможность оперирования с каждым объектом группы. Указанные два свойства присущи массивам. Но, в отличие от массива, элементы структур могут быть разных типов и обязательно должны иметь разные названия. Наглядное представление структуры – разграфленный лист бумаги с подписанными графами. Графы называют полями, их названия – названиями полей. Структура представляет собою средство составления новых типов (производных типов) на основании имеющихся. Рассмотрим простейшие примеры действий со структурами.

Пример. Объявление структуры и объектов объявленного типа.

```
#include <iostream>
#include <Windows.h>
using namespace std;
int main()
{
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);
    1.  struct kniga
    2.  {char avtor[80];
    3.  char nazvanie[80];
    4.  long int god;
    5.  };
    6.  kniga kn1, kn2, kn3;
    7.  strcpy(kn1.avtor,"Л.Н.Толстой");
    8.  strcpy(kn1.nazvanie,"Война и мир");
    9.  kn1.god=2010;
    10.cout << "Название книги: "<<kn1.nazvanie<<"\n";
    11.cout << "Автор: "<<kn1.avtor<<"\n";
    12.cout << "Год издания: "<<kn1.god<<"\n";
    13.return 0;
}
```

Описание программы.

В строках 1-5 объявляется производный тип kniga, представляющий собой два текстовых поля и одно числовое. Эти строки представляют собою план для составления новых объектов, при их прочтении исполнитель не предпринимает никаких действий с памятью (т.е. ничего не пишет и не стирает). В строке 6 объявляются объекты kn1, kn2, kn3 типа kniga. При прочтении этой строки исполнитель выделяет в памяти три места, каждое из которых состоит из трех частей – полей. Поля надписаны обозначениями avtor, nazvanie, god; сами места – обозначениями kn1, kn2, kn3. Содержимое всех полей пусто. В строках 7-9 заполняются поля объекта kn1. В строках 10-

12 содержимое этих полей выводится на экран. С объектами kn2, kn3 действий не производится.
Результат выполнения программы:
Название книги: Война и мир
Автор: Л.Н.Толстой
Год издания: 2010

Process exited after 1.792 seconds with return value 0
Для продолжения нажмите любую клавишу . . .

Структуры и массивы
Структуры могут быть элементами других структур, элементами массивов. Обратно, массив может быть элементом структуры, т.е. может быть ее полем. Ниже рассматривается пример использования структуры в качестве элементов массива.
Пример. Приведенная ниже программа представляет собой видоизменение предыдущего примера: здесь объекты типа kniga собраны в массив длины 3 (строка 6).

```
#include <iostream>
#include <Windows.h> //
using namespace std;
int main()
{
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);
    1.  struct kniga
    2.  {char avtor[80];
    3.  char nazvanie[80];
    4.  long int god;
    5.  };
    6.  kniga kn[3];
    7.  strcpy(kn[0].avtor,"Автор1");  strcpy(kn[0].nazvanie,"Название1");
        kn[0].god=2010;
    8.  strcpy(kn[1].avtor," Автор 2");strcpy(kn[1].nazvanie,"      Название      2");
        kn[1].god=2020;
    9.  strcpy(kn[2].avtor," Автор 3");strcpy(kn[2].nazvanie,"      Название      3");
        kn[2].god=2030;
    10.for(long int i=0;i<3;i=i+1)
    11.{cout << "Название книги: "<<kn[i].nazvanie<<"\n";
    12.cout << "Автор: "<<kn[i].avtor<<"\n";
    13.cout << "Год издания: "<<kn[i].god<<"\n";}
    14.return 0;
}
```

После заполнения полей (строки 7-9) их содержимое выводится в цикле на экран (строки 10-13).
Результат исполнения программы:

```
Название книги: Название1
Автор: Автор1
Год издания: 2010
Название книги: Название2
Автор: Автор2
Год издания: 2020
Название книги: Название3
Автор: Автор3
Год издания: 2030

--------------------------------
Process exited after 1.463 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

Примеры программных текстов некоторых алгоритмов поиска и сортировки

Последовательный поиск

Источник: https://prog-cpp.ru/search-serial/

Последовательный поиск предполагает последовательный просмотр всех записей множества, организованного как массив.

Пример. Найти в массиве элемент со значением, равным 3.

```
#include <stdio.h>
#include <stdlib.h> // для переключения на русский язык - функция system()
// Функция поиска в массиве k размерности n  элемента со значением key
int search(int *k, int n, int key)
{
  for (int i = 0; i < n; i++) // просматриваем все элементы в цикле
  {
    if (k[i] == key)      // если находим элемент со значением key,
      return i;     // возвращаем его индекс
  }
  return -1;  // возвращаем -1 - элемент не найден
}
int main()
{
  int k[8];    // описываем массив из 8 элементов
  int point;    // индекс элемента, равного указанному значению (3)
  system("chcp 1251"); // переходим на страницу 1251 для поддержки русского
языка
  system("cls");       // очищаем окно консоли
  // В цикле вводим элементы массива
  for (int i = 0; i<8; i++)
  {
    printf("Введите k[%d]: ", i);
    scanf("%d", &k[i]);
  }
  // Вызываем функцию поиска в массиве элемента, равного 3
  point = search(k, 8, 3);
  if (point == -1) // если функция вернула -1, такого элемента в массиве нет
    printf("Элементов равных 3 в массиве нет!\n");
  else // иначе выводим полученный индекс элемента
    printf("Элемент с индексом %d равен 3", point);
  getchar(); getchar();
  return 0;
}
```

Результат выполнения

Метод транспозиции

Источник: https://prog-cpp.ru/search-serial/

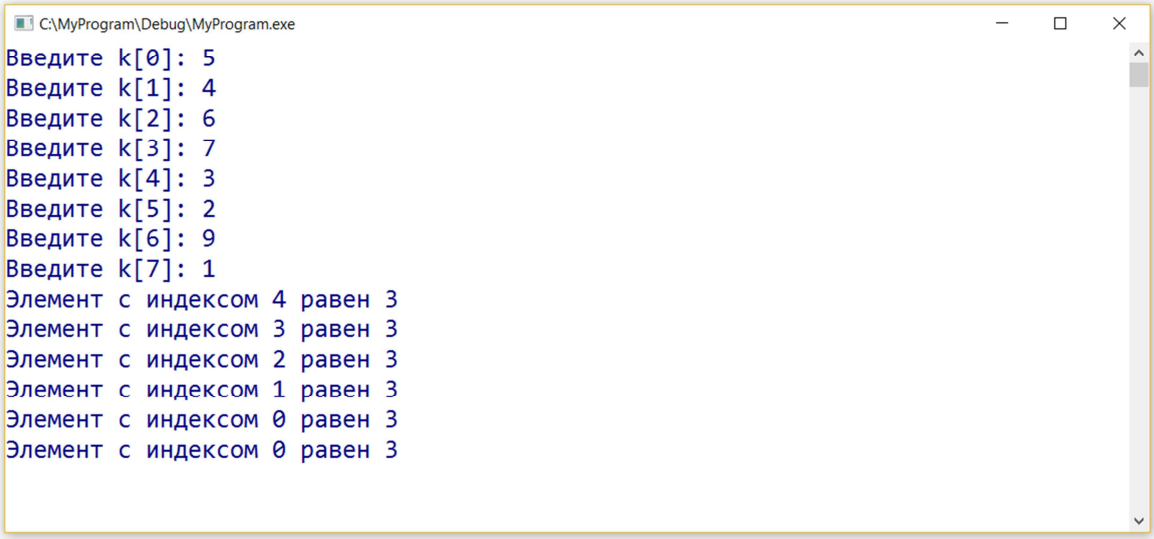
Улучшением рассмотренного метода является метод транспозиции: каждый запрос к записи сопровождается сменой мест этой и предшествующий записи; в итоге наиболее часто используемые записи постепенно перемещаются в начало таблицы; и при последующем обращении к ним, эти записи находятся почти сразу.

Программный текст.

```
#define _CRT_SECURE_NO_WARNINGS // для корректной работы scanf в Visual Studio последних версий
#define _CRT_SECURE_NO_WARNINGS // для корректной работы scanf()
#include <stdio.h>
#include <stdlib.h>    // для переключения на русский язык – функция system()
// Функция поиска в массиве k размерности n
// элемента со значением key с использованием транспозиции
int search(int *k, int n, int key)
{
 int temp;          // вспомогательная переменная для обмена
 for (int i = 0; i<n; i++) // просматриваем все элементы в цикле
 {
  if (k[i] == key)    // если находим элемент со значением key,
  {
   if (i == 0)      // если индекс равен нулю, возвращаем его,
    return i;     // потому что смещаться ближе к началу массива невозможно
temp = k[i];      // меняем местами найденный элемент с предыдущим
   k[i] = k[i - 1];
   k[i - 1] = temp;
   return i;         // возвращаем найденный индекс элемента
  }
 }
 return -1;  // если элемент не найден, возвращаем -1
}
int main()
{
```

```
  int k[8];    // описываем массив из 8 элементов
  int point;    // индекс элемента, равного указанному значению (3)
  system("chcp 1251"); // переходим на страницу 1251 для
//поддержки русского языка
  system("cls");       // очищаем окно консоли
  // В цикле вводим элементы массива
  for (int i = 0; i<8; i++)
  {
    printf("Введите k[%d]: ", i);
    scanf("%d", &k[i]);
  }
  // Повторяем процедуру поиска 6 раз (с учетом транспозиции)
  for (int j = 0; j < 6; j++)
  {
    point = search(k, 8, 3); // вызов функции поиска
    // Вывод индекса элемента, равного 3
    if (point == -1)
      printf("Элементов равных 3 в массиве нет!\n");
    else
      printf("Элемент с индексом %d равен 3\n", point);
  }
  getchar(); getchar();
  return 0;
}
```

Результат выполнения



Метод перемещения в начало

Источник: https://prog-cpp.ru/search-serial/

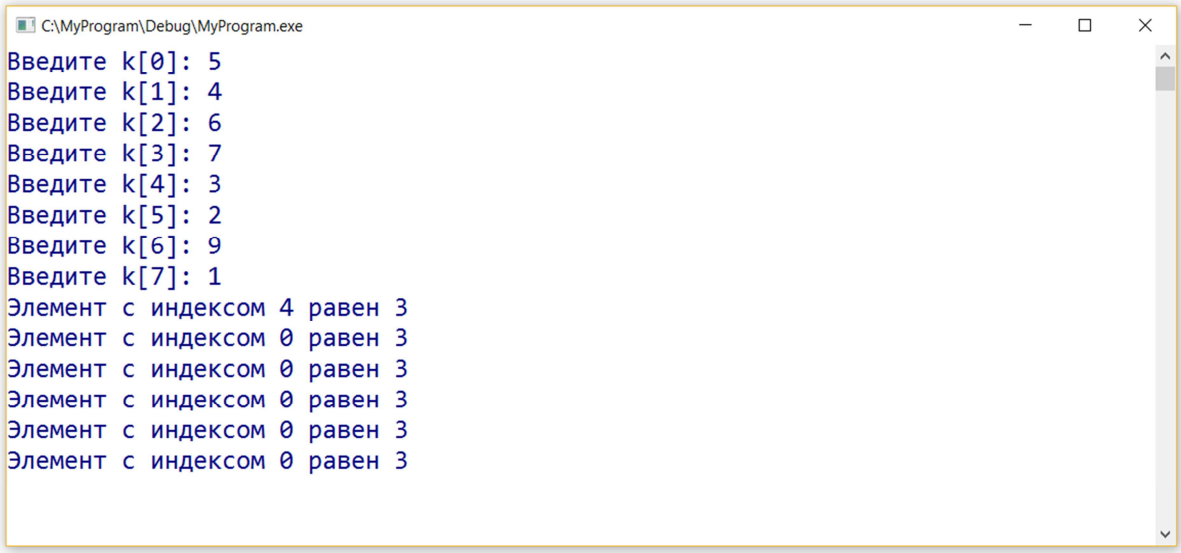
В этом методе каждый запрос к записи сопровождается её перемещением в начало таблицы. В итоге в начале таблицы оказывается запись, используемая в последний раз.
Программный текст.

```
#define _CRT_SECURE_NO_WARNINGS // для корректной работы scanf()
#include <stdio.h>
#include <stdlib.h>    // для переключения на русский язык – функция system()
// Функция поиска в массиве k размерности n
// элемента со значением key с использованием транспозиции
int search(int *k, int n, int key)
{
  int temp;          // вспомогательная переменная для обмена
  for (int i = 0; i<n; i++) // просматриваем все элементы в цикле
  {
    if (k[i] == key)    // если находим элемент со значением key,
    {
      temp = k[i];      // меняем местами найденный элемент с начальным
      k[i] = k[0];
      k[0] = temp;
      return i;         // возвращаем найденный индекс элемента
    }
  }
  return -1;  // если элемент не найден, возвращаем -1
}

int main()
{
  int k[8];    // описываем массив из 8 элементов
  int point;    // индекс элемента, равного указанному значению (3)
  system("chcp 1251"); // переходим на страницу 1251 для
//поддержки русского языка
  system("cls");       // очищаем окно консоли
  // В цикле вводим элементы массива
  for (int i = 0; i<8; i++)
  {
    printf("Введите k[%d]: ", i);
    scanf("%d", &k[i]);
  }
  // Повторяем процедуру поиска 6 раз (с учетом транспозиции)
  for (int j = 0; j < 6; j++)
  {
    point = search(k, 8, 3); // вызов функции поиска
    // Вывод индекса элемента, равного 3
    if (point == -1)
      printf("Элементов равных 3 в массиве нет!\n");
```

```
    else
      printf("Элемент с индексом %d равен 3\n", point);
  }
  getchar(); getchar();
  return 0;
}
```

Результат выполнения



Индексно-последовательный поиск

Источник: https://prog-cpp.ru/search-index/.

Для индексно-последовательного поиска в дополнение к отсортированной таблице заводится вспомогательная таблица, называемая индексной. Каждый элемент индексной таблицы состоит из ключа и указателя на запись в основной таблице, соответствующей этому ключу. Элементы в индексной таблице, как элементы в основной таблице, должны быть отсортированы по этому ключу. Если индекс имеет размер, составляющий 1/8 от размера основной таблицы, то каждая восьмая запись основной таблицы будет представлена в индексной таблице.

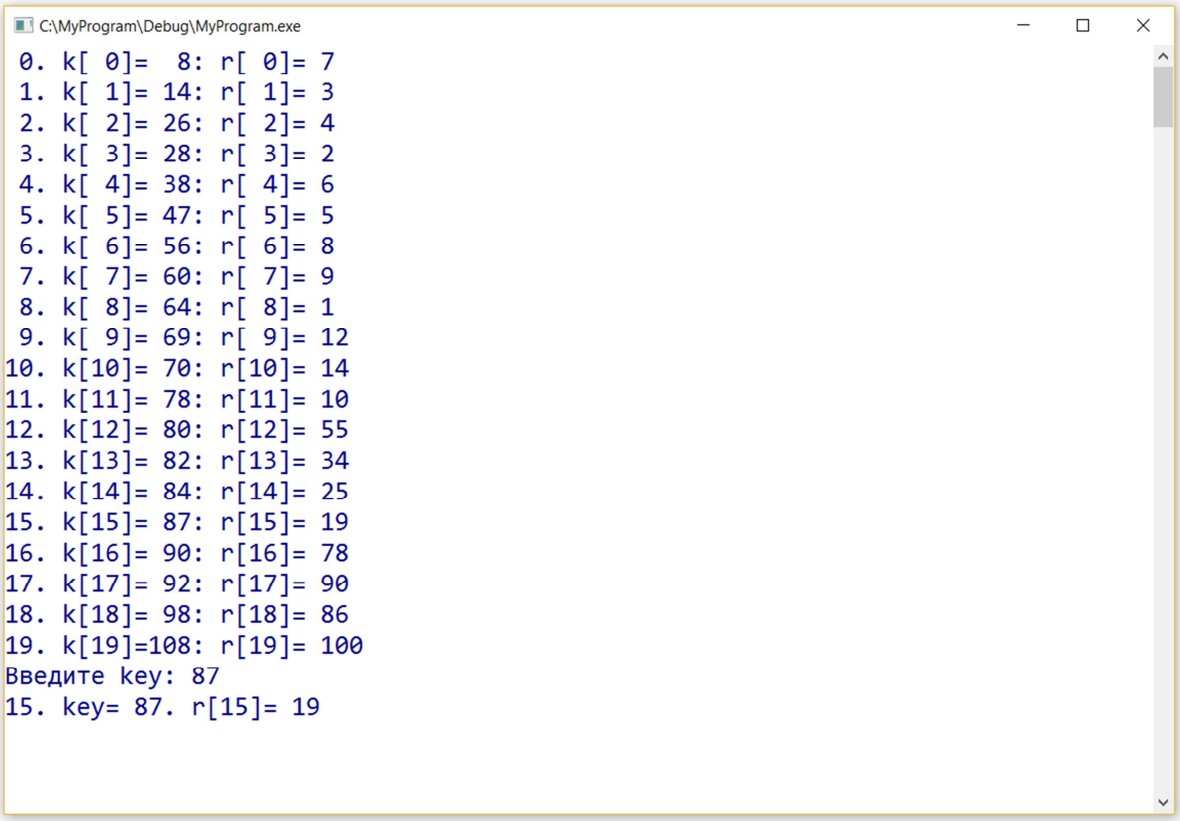
Если размер основной таблицы – n, то размер индексной таблицы ind_size=8. Достоинство алгоритма индексно-последовательного поиска заключается в том, что сокращается время поиска, так как последовательный поиск первоначально ведется в индексной таблице, имеющей меньший размер, чем основная таблица. Когда найден правильный индекс, второй последовательный поиск выполняется по небольшой части записей основной таблицы.

Программный текст.

```
#define _CRT_SECURE_NO_WARNINGS // для корректной работы scanf()
#include <stdio.h>
#include <stdlib.h> // для использования функций system()
int main()
{
 int k[20]; // массив ключей основной таблицы
 int r[20]; // массив записей основной таблицы
 int i, j, ind_size;
 int key;
 int kindex[3]; // массив ключей индексной таблицы
 int pindex[3]; // массив индексов индексной таблицы
 system("chcp 1251"); // перевод русского языка в консоли
 system("cls");     // очистка окна консоли
 // Инициализация ключевых полей упорядоченными значениями
 k[0] = 8;  k[1] = 14;
 k[2] = 26;  k[3] = 28;
 k[4] = 38;  k[5] = 47;
 k[6] = 56;  k[7] = 60;
 k[8] = 64;  k[9] = 69;
 k[10] = 70; k[11] = 78;
 k[12] = 80; k[13] = 82;
 k[14] = 84; k[15] = 87;
 k[16] = 90; k[17] = 92;
 k[18] = 98; k[19] = 108;
 // Ввод записей
 for (i = 0; i < 20; i++)
 {
  printf("%2d. k[%2d]=%3d: r[%2d]= ", i, i, k[i], i);
  scanf("%d", &r[i]);
 }
 // Формирование индексной таблицы
 for (i = 0, j = 0; i < 20; i = i + 8, j++)
 {
  kindex[j] = k[i]; // переносим каждый 8-й ключ в индексную таблицу
  pindex[j] = i;    // запоминаем текущий индекс
 }
 ind_size = j;  // запоминаем размер индексной таблицы
```

```
  pindex[j] = 20; // последний индекс равен 20
  // Поиск
  printf("Введите key: "); // вводим ключевое поле
  scanf("%d", &key);
  // Просматриваем элементы индексной таблицы
  for (j = 0; j < ind_size; j++)
  {
    if (key < kindex[j]) // если находим ключ меньше введенного,
      break;          // выходим из цикла – мы нашли нужную область основной
таблицы
  }
  if (j == 0)  i = 0;      // присваиваем i начальный индекс диапазона поиска в
основной таблице
  else  i = pindex[j - 1];
  for (i; i<pindex[j]; i++) // осуществляем поиск в основной таблице
  {                         //до следующего индекса индексной таблицы
    if (k[i] == key)  // если найдено введенное значение, выводим его
      printf("%2d. key=%3d. r[%2d]=%3d", i, k[i], i, r[i]);
  }
  getchar(); getchar();
  return 0;
}
```

Результат выполнения



Бинарный поиск

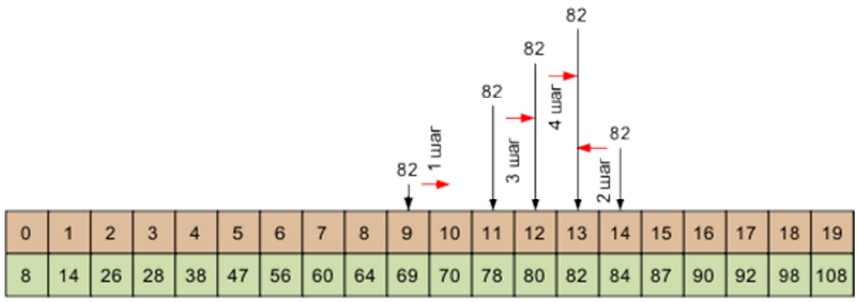
Источник: https://prog-cpp.ru/search-binary.

Бинарный поиск производится в упорядоченном массиве. При бинарном поиске искомый ключ сравнивается с ключом среднего элемента в массиве. Если они равны, то поиск успешен. В противном случае поиск осуществляется аналогично в левой или правой частях массива. Алгоритм может быть определен в рекурсивной и нерекурсивной формах. Бинарный поиск также называют поиском методом деления отрезка пополам или дихотомии. Количество шагов поиска определится как $\log_2 n\uparrow$,

где n-количество элементов, ↑ – округление в большую сторону до ближайшего целого числа. На каждом шаге осуществляется поиск середины отрезка по формуле

$$mid = (left + right)/2$$

Если искомый элемент равен элементу с индексом mid**,** поиск завершается. В случае если искомый элемент меньше элемента с индексом mid, на место mid перемещается правая граница рассматриваемого отрезка, в противном случае – левая граница.



Перед началом поиска устанавливается левая и правая границы массива:

$$left = 0, right = 19$$

Шаг 1. Ищется индекс середины массива (округляется в меньшую сторону):

$$mid = (19+0)/2=9$$

Сравнивается значение по этому индексу с искомым: $69 < 82$

Сдвигается левая граница: $left = mid = 9$

Шаг 2. Ищется индекс середины массива (округление в меньшую сторону):

$$mid = (9+19)/2=14$$

Сравнивается значение по этому индексу с искомым: $84 > 82$

Сдвигается правая граница: $right = mid = 14$

Шаг 3. Ищется индекс середины массива (округление в меньшую сторону):

$$mid = (9+14)/2=11$$

Сравнивается значение по этому индексу с искомым: $78 < 82$

Сдвигается левая граница: $left = mid = 11$

Шаг 4. Ищется индекс середины массива (округление в меньшую сторону):

$$mid = (11+14)/2=12$$

Сравнивается значение по этому индексу с искомым: $80 < 82$

Сдвигается левая граница: $left = mid = 12$

Шаг 5. Ищется индекс середины массива (округление в меньшую сторону):

$$mid = (12+14)/2=13$$

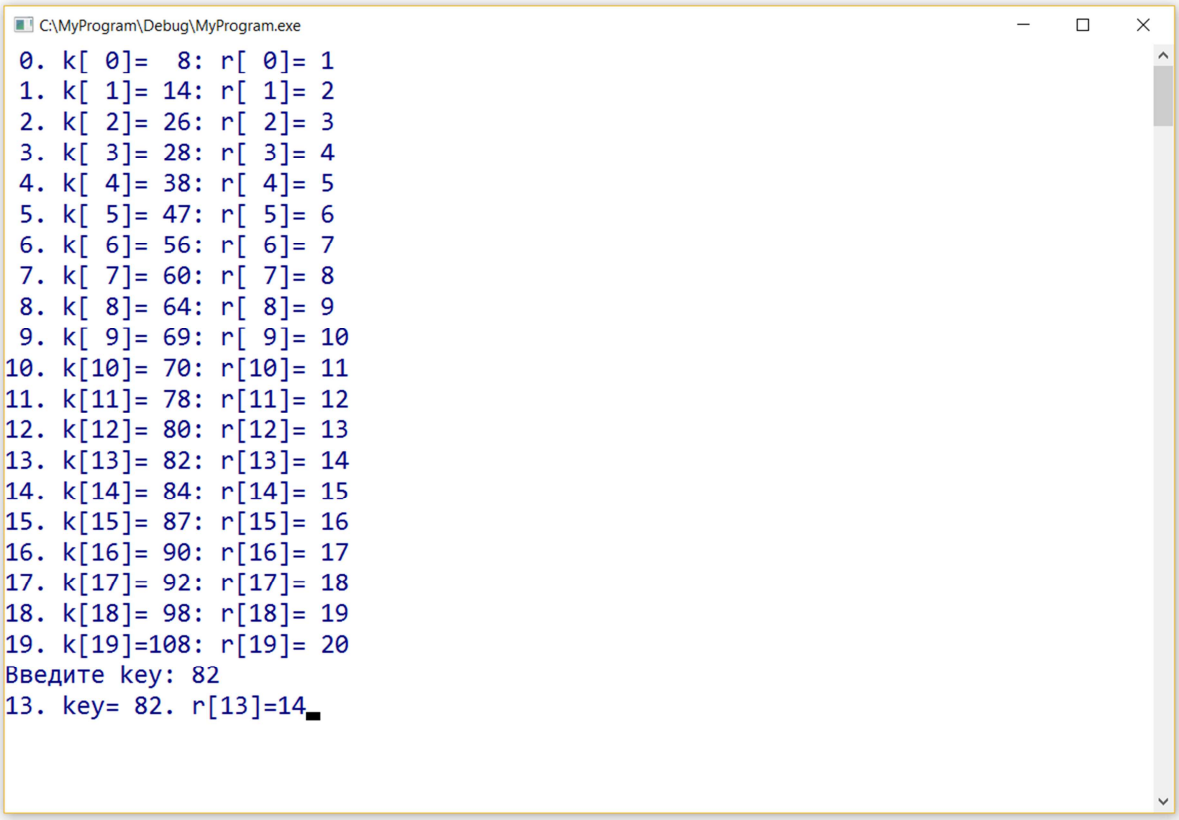
Сравнивается значение по этому индексу с искомым: $82 = 82$

Чтобы уменьшить количество шагов поиска, можно сразу смещать границы поиска на элемент, следующий за серединой отрезка: left=mid+1, right=mid-1.
Программный текст.

```
#define _CRT_SECURE_NO_WARNINGS // для корректной работы scanf()
#include <stdio.h>
#include <stdlib.h> // для использования функций system()
int main()
{
  int k[20]; // массив ключей основной таблицы
  int r[20]; // массив записей основной таблицы
  int key, i;
  system("chcp 1251"); // перевод русского языка в консоли
  system("cls");     // очистка окна консоли
  // Инициализация ключевых полей упорядоченными значениями
  k[0] = 8;  k[1] = 14;
  k[2] = 26;  k[3] = 28;
  k[4] = 38;  k[5] = 47;
  k[6] = 56;  k[7] = 60;
  k[8] = 64;  k[9] = 69;
  k[10] = 70; k[11] = 78;
  k[12] = 80; k[13] = 82;
  k[14] = 84; k[15] = 87;
  k[16] = 90; k[17] = 92;
  k[18] = 98; k[19] = 108;
  // Ввод записей
  for (i = 0; i < 20; i++)
  {
    printf("%2d. k[%2d]=%3d: r[%2d]= ", i, i, k[i], i);
    scanf("%d", &r[i]);
  }
  printf("Введите key: "); // вводим искомое ключевое поле
  scanf("%d", &key);
  int left = 0; // задаем левую и правую границы поиска
  int right = 19;
  int search = -1; // найденный индекс элемента равен -1 (элемент не найден)
  while (left <= right) // пока левая граница не "перескочит" правую
  {
    int mid = (left + right) / 2; // ищем середину отрезка
    if (key == k[mid]) {  // если ключевое поле равно искомому
      search = mid;     // мы нашли требуемый элемент,
      break;            // выходим из цикла
    }
    if (key < k[mid])     // если искомое ключевое поле меньше найденной
середины
```

```
    right = mid - 1;  // смещаем правую границу, продолжим поиск в левой части
   else               // иначе
    left = mid + 1;   // смещаем левую границу, продолжим поиск в правой части
  }
 if (search == -1)     // если индекс элемента по-прежнему -1, элемент не найден
   printf("Элемент не найден!\n");
 else          // иначе выводим элемент, его ключ и значение
   printf("%d. key= %d. r[%d]=%d", search, k[search], search, r[search]);
 getchar(); getchar();
 return 0;
}
```

Результат выполнения

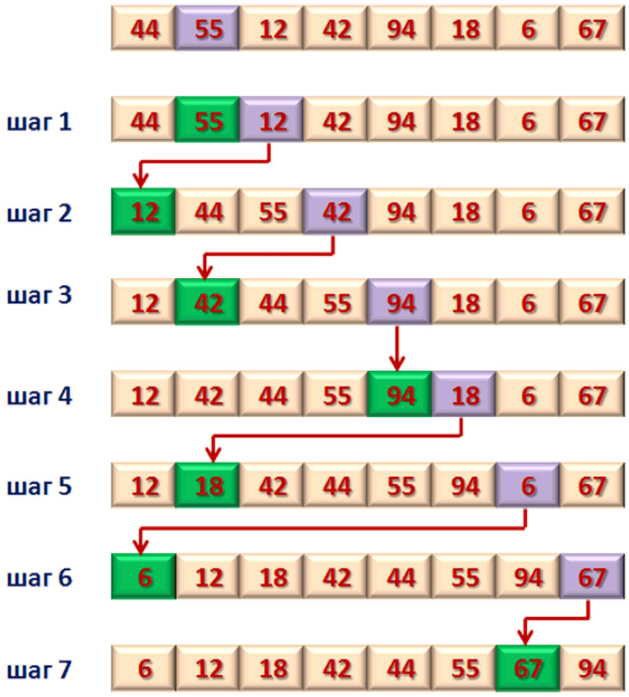


Алгоритмы сортировки

Источник: https://prog-cpp.ru/sort-include/

Элементы массива условно разделяются на готовую последовательность a_1, a_2, …, a_{i-1} и входную последовательность a_i, a_{i+1}, …, a_n.

На каждом шаге i-й элемент помещается на подходящее место в готовую последовательность.



Программный текст сортировки прямыми включениями

```
#define _CRT_SECURE_NO_WARNINGS // для корректной работы scanf()
#include <stdio.h>
// Функция сортировки прямыми включениями
void inclusionSort(int *num, int size)
{
  // Для всех элементов кроме начального
  for (int i = 1; i < size; i++)
  {
    int value = num[i]; // запоминаем значение элемента
    int index = i;     // и его индекс
    while ((index > 0) && (num[index - 1] > value))
    {   // смещаем другие элементы к концу массива пока они меньше index
      num[index] = num[index - 1];
      index--;    // смещаем просмотр к началу массива
    }
    num[index] = value; // рассматриваемый элемент помещаем на
освободившееся место
  }
}
int main()
{
  int a[10]; // Объявляем массив из 10 элементов
```

```
  // Вводим значения элементов массива
  for (int i = 0; i < 10; i++)
  {
    printf("a[%d] = ", i);
    scanf("%d", &a[i]);
  }
  inclusionSort(a, 10);  // вызываем функцию сортировки
  // Выводим отсортированные элементы массива
  for (int i = 0; i<10; i++)
    printf("%d ", a[i]);
  getchar(); getchar();
  return 0;
}
```

Результат выполнения

Анализ выполнения

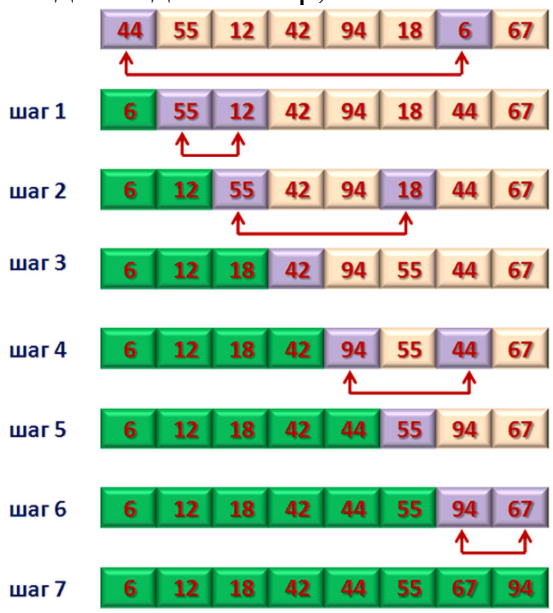
Число сравнений ключей C_i при i-м просеивании составляет самое большое i-1, самое меньшее – 1. Если предположить, что все перестановки из n ключей равновероятны, то среднее число сравнений – i/2. Число пересылок: $M_i=C_i+2$. Общее число сравнений и пересылок таковы: $C_{min}=n-1$; $M_{min}=3(n-1)$; $C_{cp}=(n^2+n-2)/4$; $M_{cp}=(n^2+9n-10)/4$; $C_{max}=(n^2+n-4)/4$; $M_{max}=(n^2+3n-4)/2$. Минимальные оценки встречаются в случае уже упорядоченной исходной последовательности элементов, наихудшие оценки – когда элементы первоначально расположены в обратном порядке.

Сортировка прямым выбором

Источник: https://prog-cpp.ru/sort-select/

При прямом включении на каждом шаге рассматривается только один очередной элемент входной последовательности и все элементы готовой последовательности для нахождения места включения. При прямом выборе для поиска одного элемента с наименьшим ключом просматриваются все элементы входной последовательности и найденный элемент помещается как очередной элемент в конец готовой последовательности. Метод сортировки прямым выбором основан на следующих правилах. Выбирается элемент с наименьшим ключом. Он меняется местами с первым элементом a_0. Затем

эти операции повторяются с оставшимися n-1 элементами, n-2 элементами и так далее до тех пор, пока не останется один, самый большой элемент.



Программный текст.

```
#define _CRT_SECURE_NO_WARNINGS // для корректной работы scanf()
#include <stdio.h>
// Функция сортировки прямым выбором
void selectionSort(int *num, int size)
{
  int min, temp; // для поиска минимального элемента и для обмена
  for (int i = 0; i < size - 1; i++)
  {
    min = i; // запоминаем индекс текущего элемента
    // ищем минимальный элемент чтобы поместить на место i-ого
    for (int j = i + 1; j < size; j++)  // для остальных элементов после i-ого
    {
      if (num[j] < num[min]) // если элемент меньше минимального,
        min = j;       // запоминаем его индекс в min
    }
    temp = num[i];      // меняем местами i-ый и минимальный элементы
    num[i] = num[min];
    num[min] = temp;
  }
}
int main()
{
  int a[10]; // Объявляем массив из 10 элементов
  // Вводим значения элементов массива
  for (int i = 0; i < 10; i++)
  {
    printf("a[%d] = ", i);
    scanf("%d", &a[i]);
```

```
  }
  selectionSort(a, 10);  // вызываем функцию сортировки
  // Выводим отсортированные элементы массива
  for (int i = 0; i<10; i++)
    printf("%d ", a[i]);
  getchar(); getchar();
  return 0;
}
```

Результат выполнения

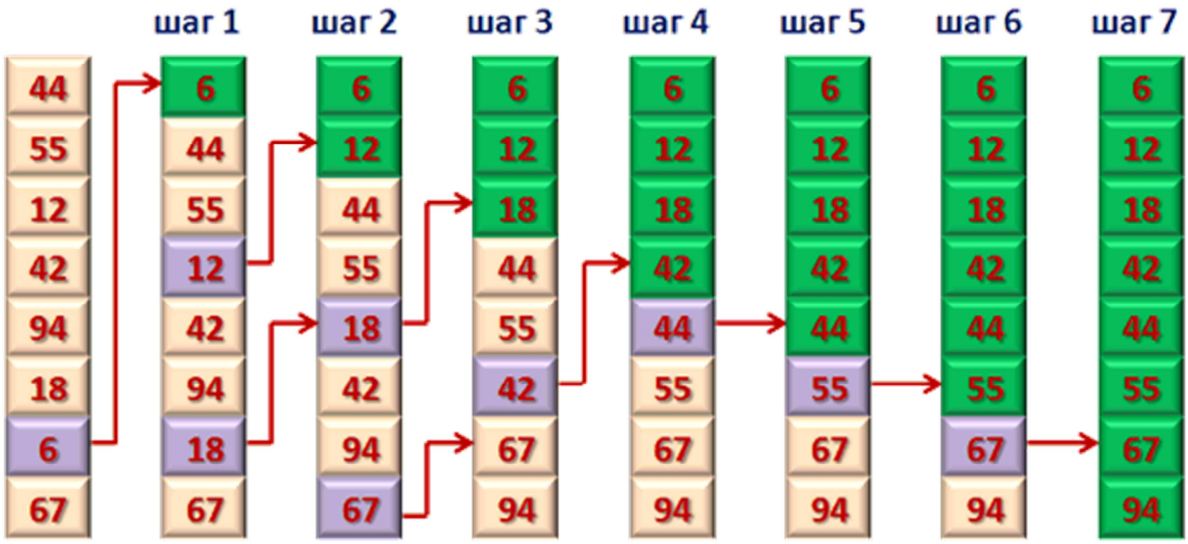
Анализ алгоритма прямым выбором.

Число сравнений ключей C не зависит от порядка ключей: $C=½(n^2-n)$. Число перестановок минимально $M_{min}=3(n-1)$ в случае изначально упорядоченных ключей и максимально $M_{max}= n2/4 +3(n-1)$, если первоначально ключи располагаются в обратном порядке. Среднее число пересылок $M_{cp}≈n(\ln n+g)$, где g = 0,577216... — константа Эйлера.

Сортировка прямым обменом (метод «пузырька»)

Источник: https://prog-cpp.ru/sort-bubble/

Алгоритм сортировки прямым обменом основан на принципе сравнения и обмена пары соседних элементов до тех пор, пока не будут отсортированы все элементы. Как и в методе прямого выбора, совершаются проходы по массиву, сдвигая каждый раз наименьший элемент оставшейся последовательности к началу массива. Если рассматривать массивы как вертикальные, а не горизонтальные построения, то элементы можно интерпретировать как пузырьки в банке с водой, причем вес каждого соответствует его ключу. В этом случае при каждом проходе один пузырек как бы поднимается до уровня, соответствующего его весу. Такой метод известен под названием «пузырьковая сортировка».



Программный текст.

```
#define _CRT_SECURE_NO_WARNINGS // для корректной работы scanf()
#include <stdio.h>
// Функция сортировки прямым обменом (метод "пузырька")
void bubbleSort(int *num, int size)
{
  // Для всех элементов
  for (int i = 0; i < size - 1; i++)
  {
    for (int j = (size - 1); j > i; j--) // для всех элементов после i-ого
    {
      if (num[j - 1] > num[j]) // если текущий элемент меньше предыдущего
      {
        int temp = num[j - 1]; // меняем их местами
        num[j - 1] = num[j];
        num[j] = temp;
      }
    }
  }
}
int main()
{
  int a[10]; // Объявляем массив из 10 элементов
  // Вводим значения элементов массива
  for (int i = 0; i < 10; i++)
  {
    printf("a[%d] = ", i);
    scanf("%d", &a[i]);
  }
  bubbleSort(a, 10);  // вызываем функцию сортировки
  // Выводим отсортированные элементы массива
  for (int i = 0; i<10; i++)
```

```
    printf("%d ", a[i]);
  getchar(); getchar();
  return 0;
}
```

Результат выполнения

Анализ алгоритма.

Число сравнений в алгоритме прямого обмена C = $(n^2- n)/2$; минимальное, среднее и максимальное число перемещений элементов равно соответственно $M_{min} = 0$, $M_{cp} = 3(n2 - n)/2$, $M_{max} = 3(n2 - n)/4$.

Шейкер-сортировка

Источник: https://prog-cpp.ru/sort-shaker/

Шейкер-сортировка (сортировка взбалтыванием) является усовершенствованным методом пузырьковой сортировки. Анализируя метод пузырьковой сортировки, можно отметить два обстоятельства:

- если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, ее можно исключить из рассмотрения;
- при движении от конца массива к началу минимальный элемент «всплывает» на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо.

Эти две идеи приводят к модификациям в методе пузырьковой сортировки.

- От последней перестановки до конца (начала) массива находятся отсортированные элементы. Учитывая данный факт, просмотр осуществляется не до конца (начала) массива, а до конкретной позиции. Границы сортируемой части массива сдвигаются на 1 позицию на каждой итерации.
- Массив просматривается поочередно справа налево и слева направо.
- Просмотр массива осуществляется до тех пор, пока все элементы не встанут в порядке возрастания (убывания).
- Количество просмотров элементов массива определяется моментом упорядочивания его элементов.

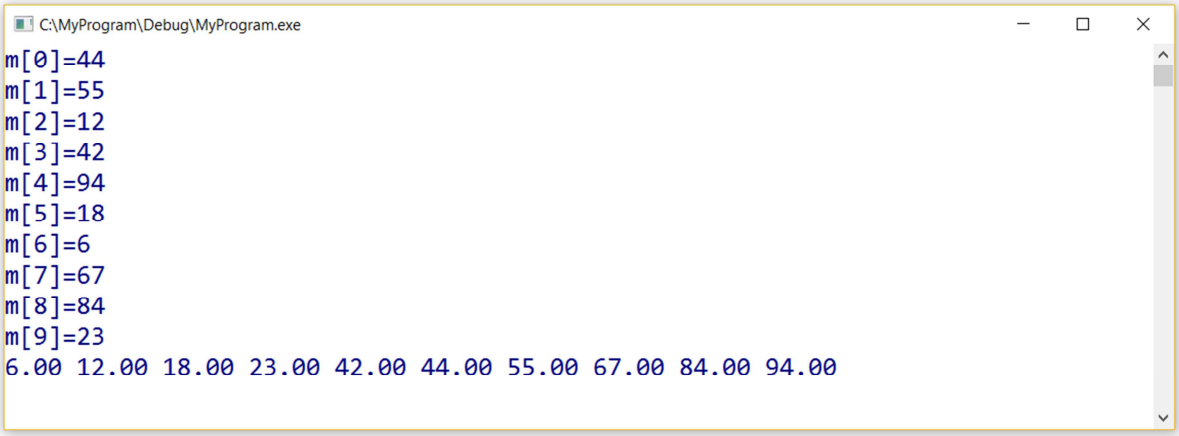
Рассмотрим алгоритм Шейкер-сортировки на примере. Дана последовательность

Программный текст.

```
#define _CRT_SECURE_NO_WARNINGS // для корректной работы scanf()
#include <stdio.h>
// Функция Шейкер-сортировки
void shekerSort(double *mass, int count)
{
  int left = 0, right = count - 1; // левая и правая границы сортируемой области массива
  int flag = 1;  // флаг наличия перемещений
  // Выполнение цикла пока левая граница не сомкнётся с правой
  // и пока в массиве имеются перемещения
  while ((left < right) && flag > 0)
  {
    flag = 0;
    for (int i = left; i<right; i++)  //двигаемся слева направо
    {
      if (mass[i]>mass[i + 1]) // если следующий элемент меньше текущего,
      {             // меняем их местами
        double t = mass[i];
        mass[i] = mass[i + 1];
        mass[i + 1] = t;
        flag = 1;      // перемещения в этом цикле были
      }
    }
```

```
    right--; // сдвигаем правую границу на предыдущий элемент
    for (int i = right; i>left; i--)  //двигаемся справа налево
    {
      if (mass[i - 1]>mass[i]) // если предыдущий элемент больше текущего,
      {           // меняем их местами
        double t = mass[i];
        mass[i] = mass[i - 1];
        mass[i - 1] = t;
        flag = 1;    // перемещения в этом цикле были
      }
    }
    left++; // сдвигаем левую границу на следующий элемент
  }
}
int main() {
  double m[10];
  // Вводим элементы массива
  for (int i = 0; i<10; i++) {
    printf("m[%d]=", i);
    scanf("%lf", &m[i]);
  }
  shekerSort(m, 10); // вызываем функцию сортировки
  // Выводим отсортированные элементы массива
  for (int i = 0; i<10; i++)
    printf("%.2lf ", m[i]);
  getchar(); getchar();
  return 0;
}
```

Результат выполнения

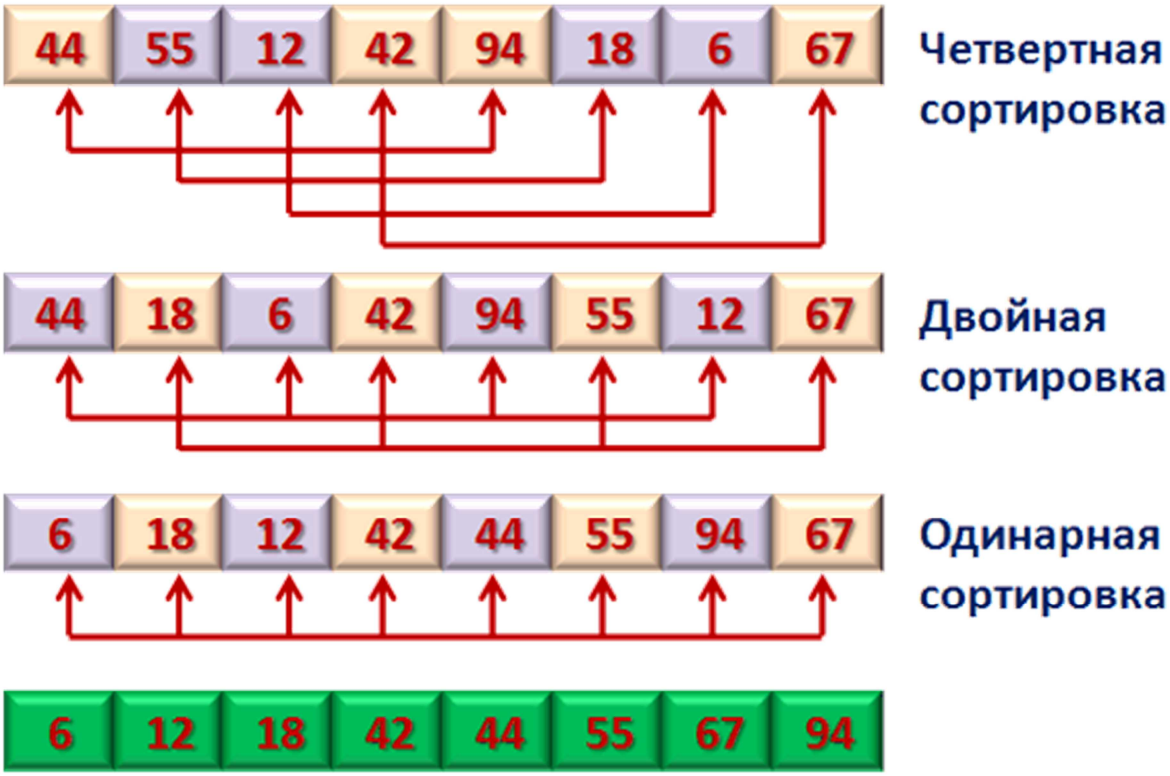


Сортировка включениями с убывающими приращениями

Источник: https://prog-cpp.ru/sort-shell/

Краткое описание. Сначала отдельно группируются и сортируются элементы, отстоящие друг от друга на 4 позиции. Такой процесс называется четвертной

сортировкой. После первого прохода элементы перегруппировываются – теперь каждый элемент группы отстоит от другого на 2 позиции – и вновь сортируются (двойная сортировка). На третьем проходе идет обычная сортировка.



На каждом этапе либо сортируется относительно мало элементов, либо элементы уже довольно хорошо упорядочены и требуется сравнительно немного перестановок. Такой метод в результате дает упорядоченный массив, и каждый проход от предыдущих только выигрывает (так как каждая i-сортировка объединяет две группы, уже отсортированные 2i-сортировкой).
Расстояния в группах можно уменьшать по-разному, но последнее должно быть единичным. В худшем случае вся сортировка приходится на последний проход

Программный текст.

```
#define _CRT_SECURE_NO_WARNINGS // для корректной работы scanf()
#include <stdio.h>
// Функция сортировки Шелла
void shellSort(int *num, int size)
{
  int increment = 3;    // начальное приращение сортировки
  while (increment > 0)  // пока существует приращение
  {
    for (int i = 0; i < size; i++)  // для всех элементов массива
    {
      int j = i;          // сохраняем индекс и элемент
      int temp = num[i];
      // просматриваем остальные элементы массива, отстоящие от j-ого
      // на величину приращения
```

```
      while ((j >= increment) && (num[j - increment] > temp))
      {  // пока отстоящий элемент больше текущего
        num[j] = num[j - increment]; // перемещаем его на текущую позицию
        j = j - increment;       // переходим к следующему отстоящему элементу
      }
      num[j] = temp; // на выявленное место помещаем сохранённый элемент
    }
    if (increment > 1)      // делим приращение на 2
      increment = increment / 2;
    else if (increment == 1)   // последний проход завершён,
      break;  // выходим из цикла
  }
}
int main()
{
  int m[10];
  // Вводим элементы массива
  for (int i = 0; i<10; i++)
  {
    printf("m[%d]=", i);
    scanf("%d", &m[i]);
  }
  shellSort(m, 10); // вызываем функцию сортировки
  // Выводим отсортированные элементы массива
  for (int i = 0; i<10; i++)
    printf("%.2d ", m[i]);
  getchar(); getchar();
  return 0;
}
```

Результат выполнения

Анализ алгоритма.

Приводимая программа не ориентирована на некую определенную последовательность расстояний. Все t расстояний обозначаются соответственно h_1, h_2, ..., h_t, для них выполняются условия $h_t=1$; $h_{i+1}<h_i$.

Каждая h-сортировка программируется как сортировка с помощью прямого включения. Дональд Кнут рекомендует такую последовательность: 1, 3, 7, 15, 31, ..., то есть $h_{k-1}=2h_k+1$, $h_t=1$ и $t = [\log_2 n] - 1$.

Сортировка с помощью дерева

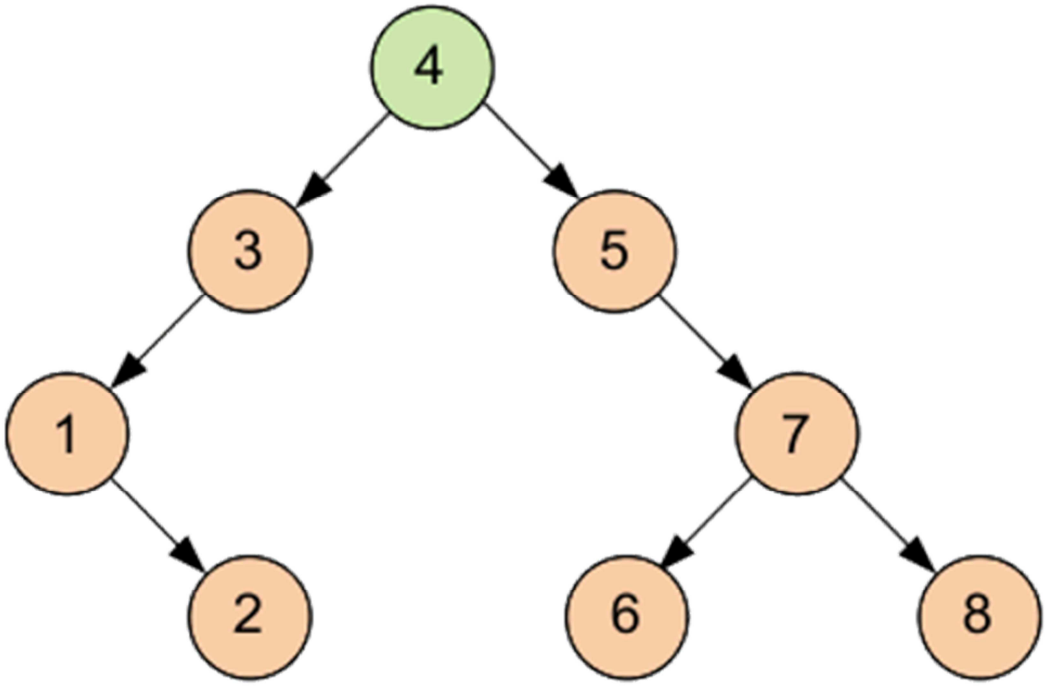
Источник: https://prog-cpp.ru/sort-tree/

Сортировка с помощью дерева осуществляется на основе бинарного дерева поиска. Бинарное (двоичное) дерево поиска – это бинарное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

- оба поддерева – левое и правое, являются двоичными деревьями поиска;
- у всех узлов левого поддерева произвольного узла X значения ключей данных меньше, чем значение ключа данных самого узла X;
- у всех узлов правого поддерева произвольного узла X значения ключей данных не меньше, чем значение ключа данных узла X.

Данные в каждом узле должны обладать ключами, на которых определена операция сравнения меньше. Для сортировки с помощью дерева исходная сортируемая последовательность представляется в виде структуры данных «дерево».

Например, исходная последовательность имеет вид: **4, 3, 5, 1, 7, 8, 6, 2.** Корнем дерева будет начальный элемент последовательности. Далее все элементы, меньшие корневого, располагаются в левом поддереве, все элементы, большие корневого, располагаются в правом поддереве. Причем это правило должно соблюдаться на каждом уровне. После того, как все элементы размещены в структуре «дерево», необходимо вывести их, используя инфиксную форму обхода.



Программный текст.

```
#include <iostream>
using namespace std;
// Структура - узел дерева
struct tnode
{
  int field;           // поле данных
  struct tnode *left;  // левый потомок
  struct tnode *right; // правый потомок
};
// Вывод узлов дерева (обход в инфиксной форме)
void treeprint(tnode *tree)
{
  if (tree != NULL) {      //Пока не встретится пустой узел
    treeprint(tree->left);  //Рекурсивная функция вывода левого поддерева
treeprint(tree->right); //Рекурсивная функция вывода правого поддерева
  }
}
// Добавление узлов в дерево
struct tnode * addnode(int x, tnode *tree) {
  if (tree == NULL)     // Если дерева нет, то формируем корень
  {
    tree = new tnode; //память под узел
    tree->field = x;   //поле данных
    tree->left = NULL;
    tree->right = NULL; //ветви инициализируем пустотой
  }
  else     // иначе
    if (x < tree->field)   //Если элемент x меньше корневого, уходим влево
      tree->left = addnode(x, tree->left); //Рекурсивно добавляем элемент
    else  //иначе уходим вправо
      tree->right = addnode(x, tree->right); //Рекурсивно добавляем элемент
    return(tree);
}
//Освобождение памяти дерева
void freemem(tnode *tree)
{
  if (tree != NULL)    // если дерево не пустое
  {
    freemem(tree->left);   // рекурсивно удаляем левую ветку
    freemem(tree->right);  // рекурсивно удаляем правую ветку
    delete tree;           // удаляем корень
  }
}
// Тестирование работы
```

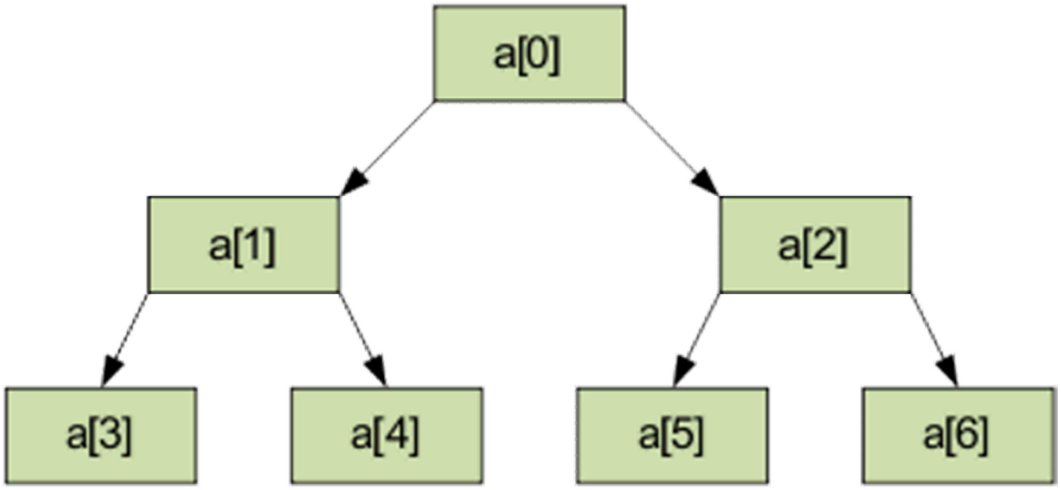
```
int main()
{
  struct tnode *root = 0;    // Объявляем структуру дерева
  system("chcp 1251");    // переходим на русский язык в консоли
  system("cls");
  int a;           // текущее значение узла
  // В цикле вводим 8 узлов дерева
  for (int i = 0; i< 8; i++)
  {
    cout << "Введите узел " << i + 1 << ": ";
    cin >> a;
    root = addnode(a, root); // размещаем введенный узел на дереве
  }
  treeprint(root);    // выводим элементы дерева, получаем отсортированный
массив
  freemem(root);      // удаляем выделенную память
  cin.get();  cin.get();
  return 0;
}
```

Результат выполнения

Пирамидальная сортировка

Источник: https://prog-cpp.ru/sort-pyramid/

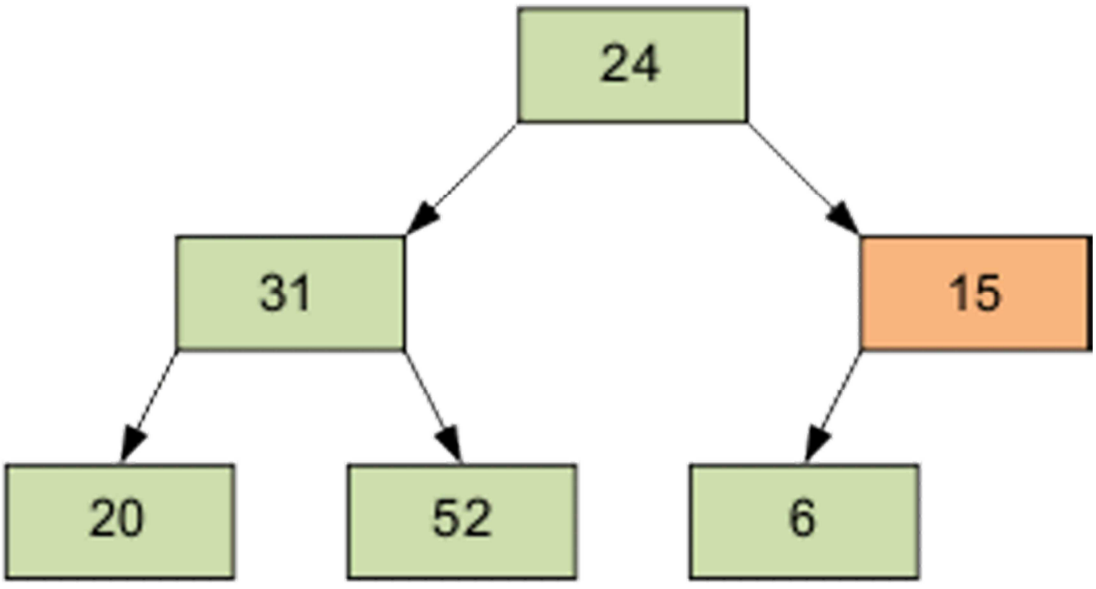
Пирамидой (кучей) называется двоичное дерево такое, что a[i]≤a[2i+1], a[i] ≤ a[2i+2].



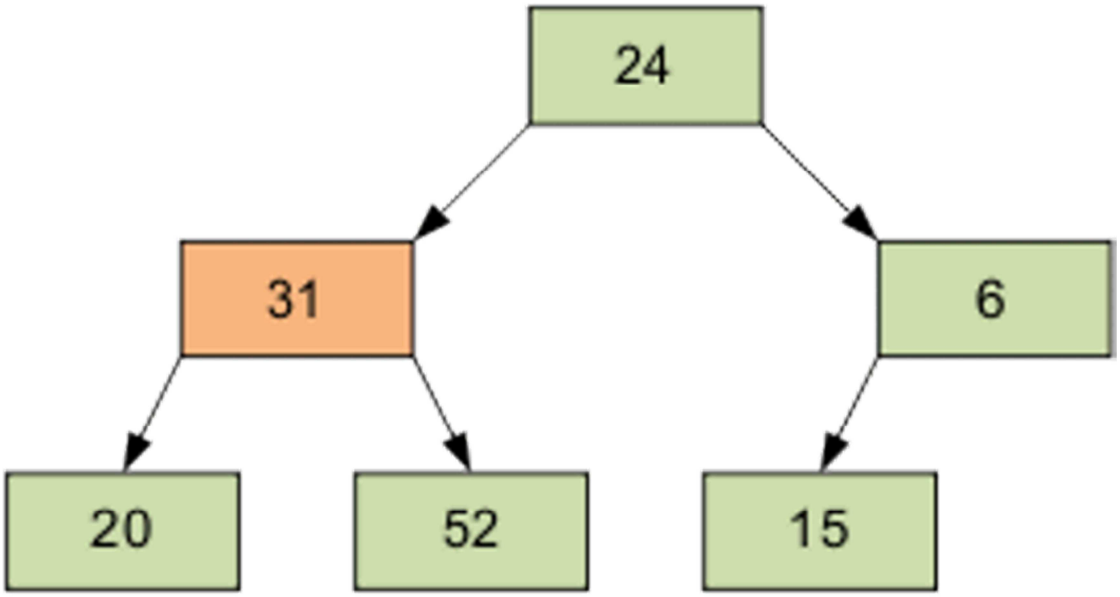
a[0] – минимальный элемент пирамиды. Общая идея пирамидальной сортировки заключается в том, что сначала строится пирамида из элементов исходного массива, а затем осуществляется сортировка элементов. Выполнение алгоритма разбивается на два этапа.

1. Построение пирамиды. Определяется правая часть дерева, начиная с n/2-1 (нижний уровень дерева). Берется элемент левее этой части массива и просеивается сквозь пирамиду по пути, где находятся меньшие его элементы, которые одновременно перемещаются вверх; из двух возможных путей выбирается путь через меньший элемент. Например, пусть массив для сортировки имеет вид: 24, 31, 15, 20, 52, 6.

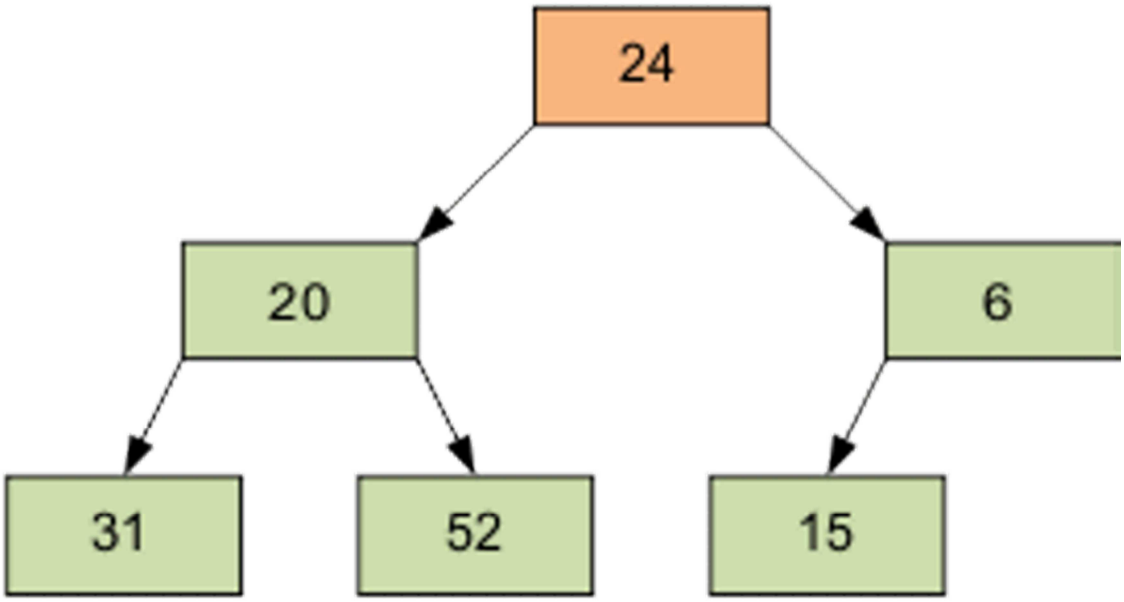
Расположим элементы в виде исходной пирамиды; номер элемента правой части (6/2-1)=2 – это элемент 15.



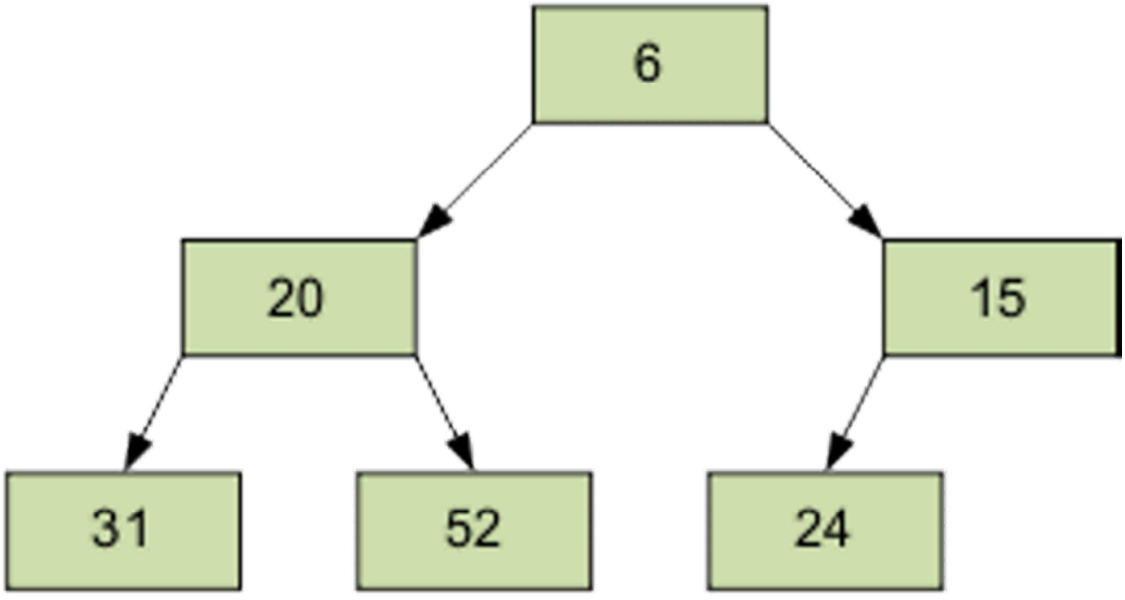
Результат просеивания элемента 15 через пирамиду.



Следующий просеиваемый элемент – 1, равный 31.

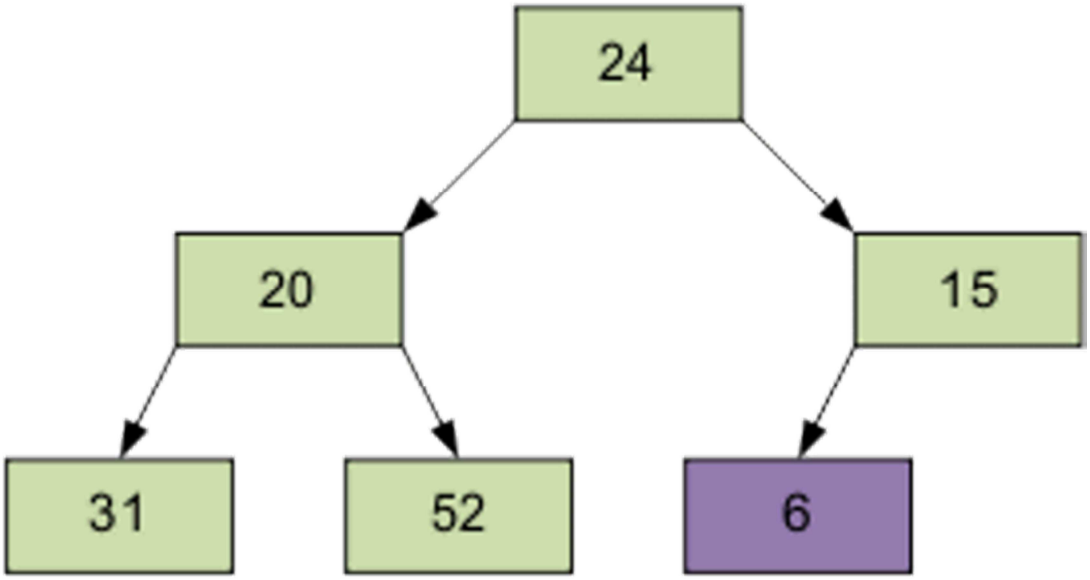


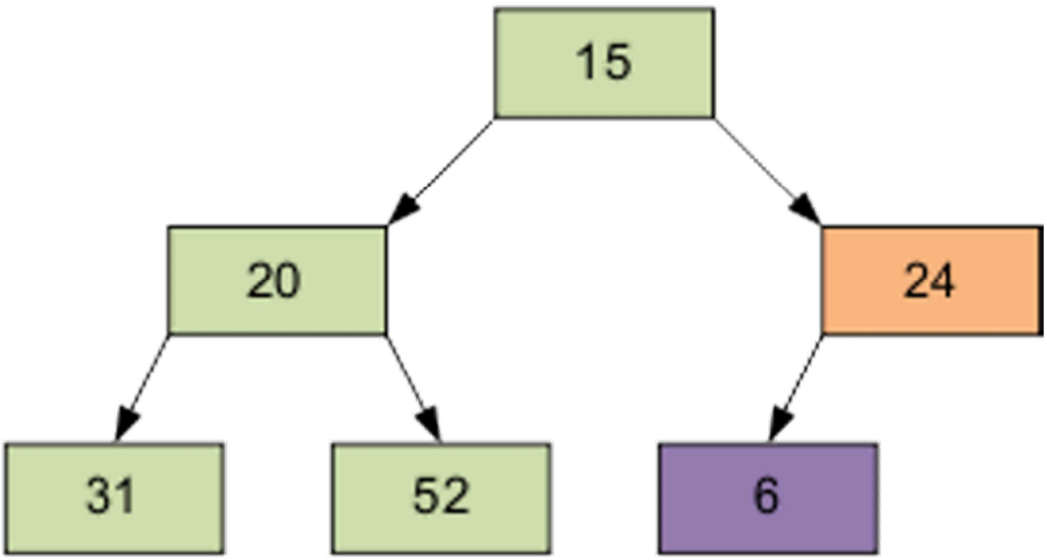
Затем – элемент 0, равный 24.



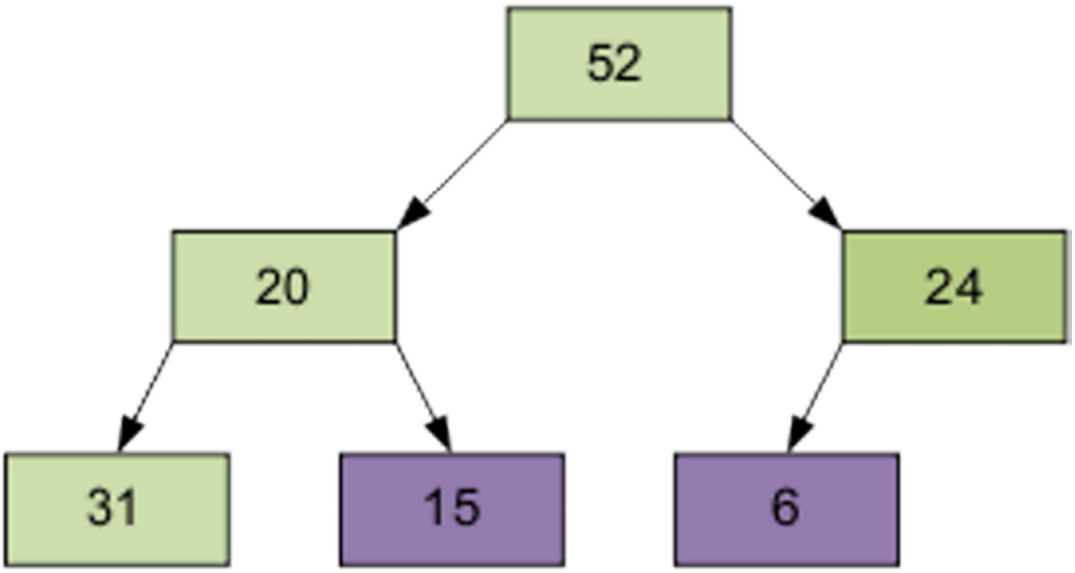
В итоге просеивания наименьший элемент оказывается на вершине пирамиды, но полученный массив еще не упорядочен.

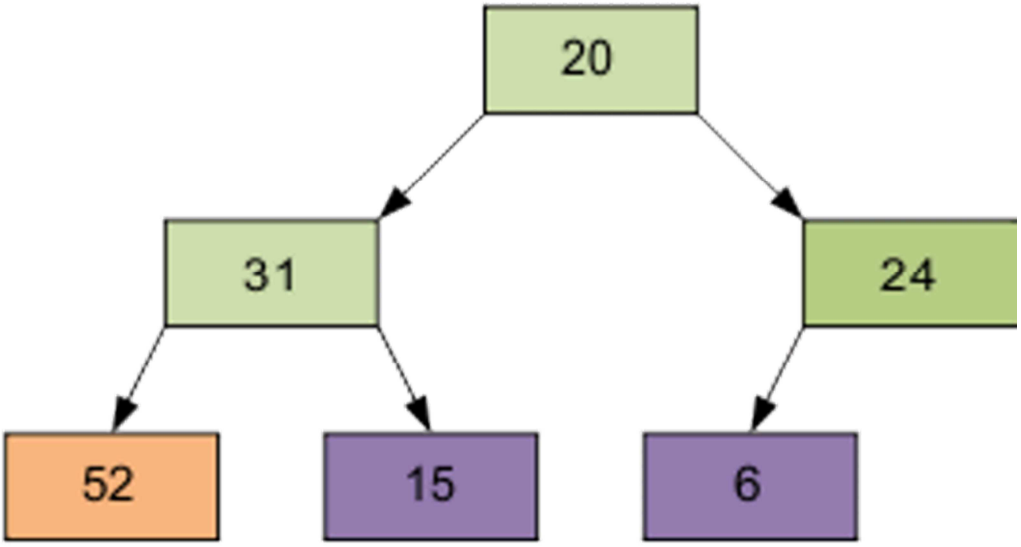
2. Сортировка на построенной пирамиде. Берется последний элемент массива в качестве текущего. Верхний (наименьший) элемент массива и текущий меняются местами. Текущий элемент (он теперь верхний) просеивается сквозь n-1 элементную пирамиду. Затем берется предпоследний элемент и т.д.

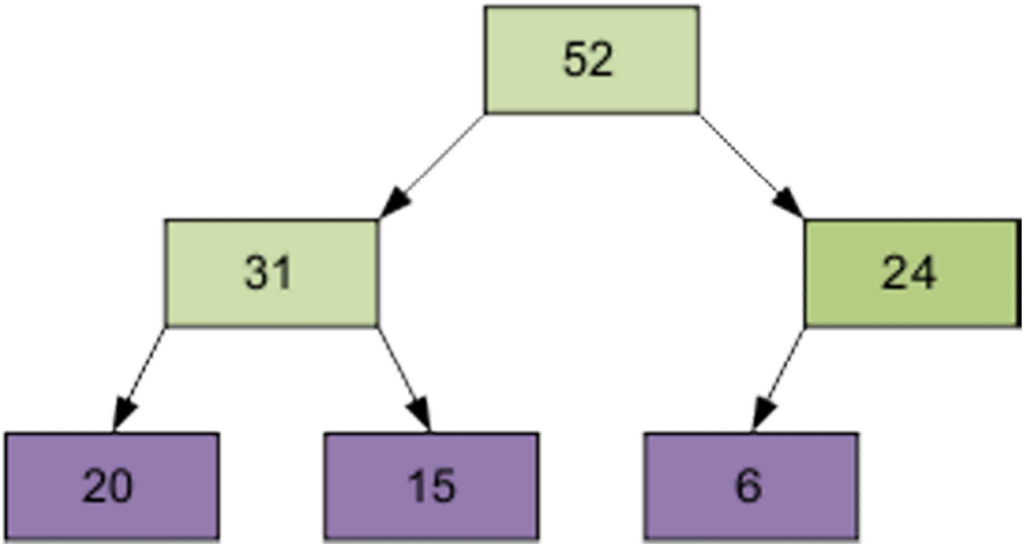


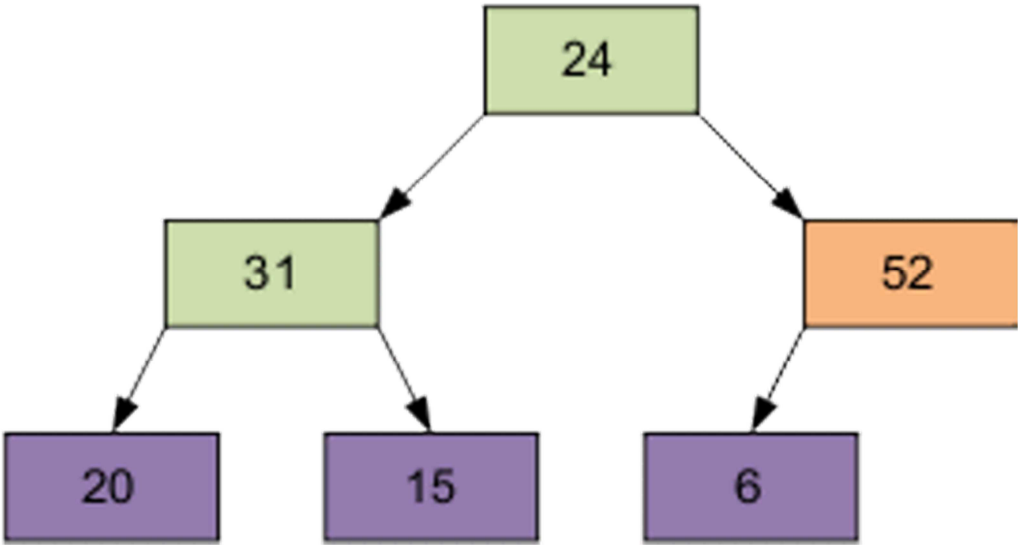


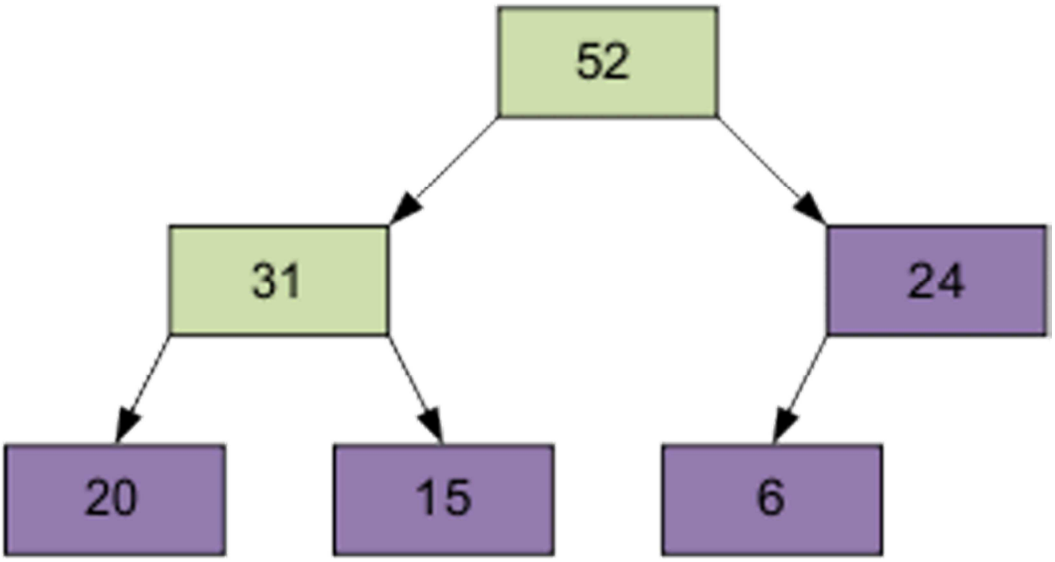
В итоге массив будет отсортирован по убыванию.

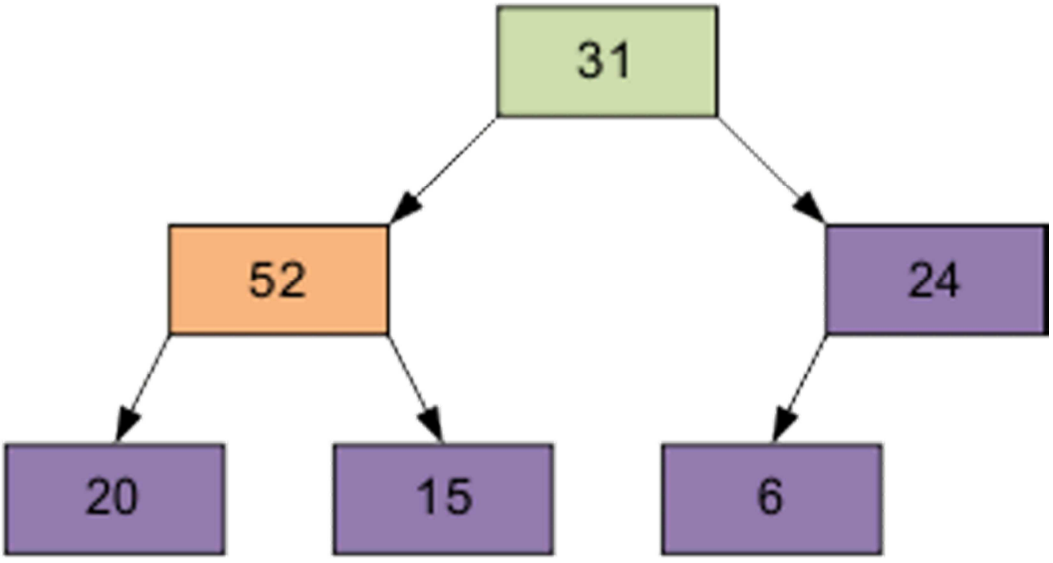

52
20
24
31
15
6

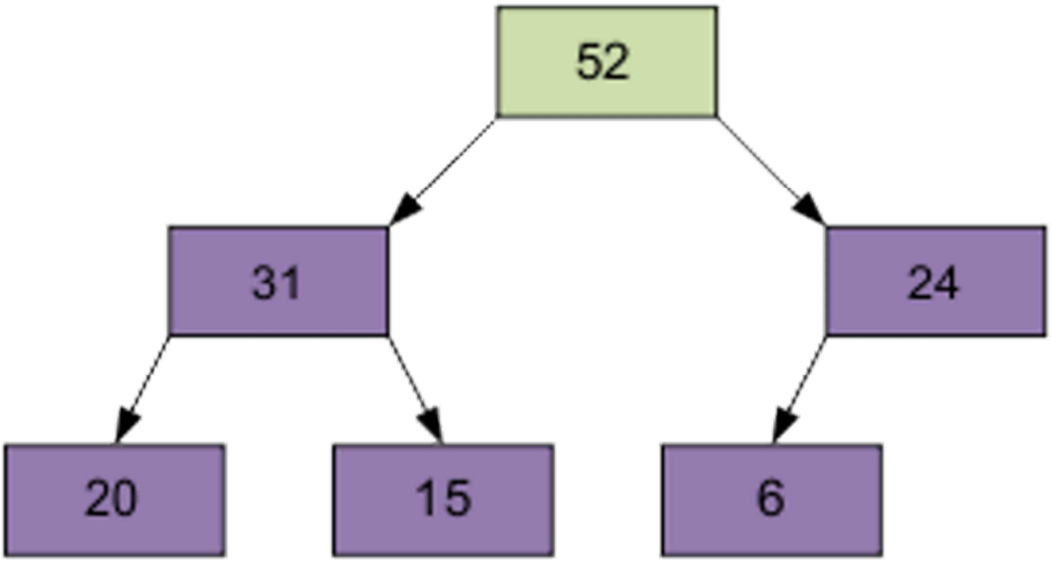

20
31
24
52
15
6

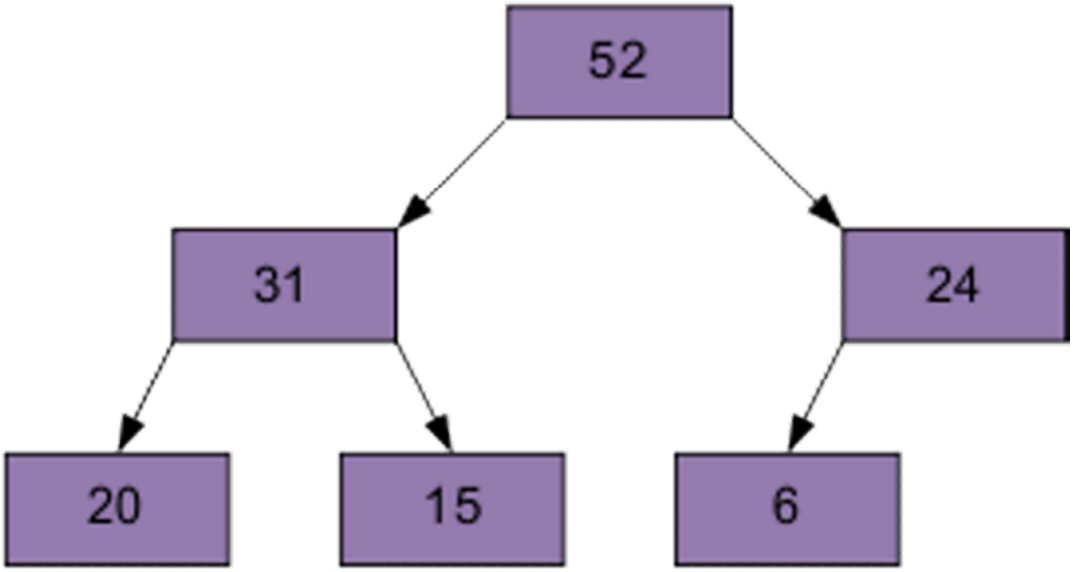

52
31
24
20
15
6


24
31
52
20
15
6


52
31
24
20
15
6


31
52
24
20
15
6





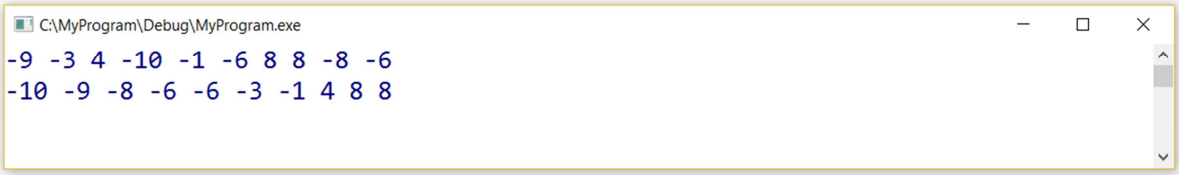
Программный текст.

```
#include <stdio.h>
#include <stdlib.h>
// Функция "просеивания" через кучу – формирование кучи
void siftDown(int *numbers, int root, int bottom)
{
  int maxChild; // индекс максимального потомка
  int done = 0; // флаг того, что куча сформирована
  // Пока не дошли до последнего ряда
  while ((root * 2 <= bottom) && (!done))
  {
    if (root * 2 == bottom)     // если мы в последнем ряду,
      maxChild = root * 2;    // запоминаем левый потомок
    // иначе запоминаем больший потомок из двух
    else if (numbers[root * 2] > numbers[root * 2 + 1])
      maxChild = root * 2;
```

```
    else
      maxChild = root * 2 + 1;
    // если элемент вершины меньше максимального потомка
    if (numbers[root] < numbers[maxChild])
    {
      int temp = numbers[root]; // меняем их местами
      numbers[root] = numbers[maxChild];
      numbers[maxChild] = temp;
      root = maxChild;
    }
    else // иначе
      done = 1; // пирамида сформирована
  }
}
// Функция сортировки на куче
void heapSort(int *numbers, int array_size)
{
  // Формируем нижний ряд пирамиды
  for (int i = (array_size / 2); i >= 0; i--)
    siftDown(numbers, i, array_size - 1);
  // Просеиваем через пирамиду остальные элементы
  for (int i = array_size - 1; i >= 1; i--)
  {
    int temp = numbers[0];
    numbers[0] = numbers[i];
    numbers[i] = temp;
    siftDown(numbers, 0, i - 1);
  }
}
int main()
{
  int a[10];
  // Заполнение массива случайными числами
  for (int i = 0; i<10; i++)
    a[i] = rand() % 20 - 10;
  // Вывод элементов массива до сортировки
  for (int i = 0; i<10; i++)
    printf("%d ", a[i]);
  printf("\n");
  heapSort(a, 10); // вызов функции сортировки
  // Вывод элементов массива после сортировки
  for (int i = 0; i<10; i++)
    printf("%d ", a[i]);
  printf("\n");
  getchar();
```

```
  return 0;
}
```

Результат выполнения



Краткая характеристика алгоритма. Алгоритм сортировки эффективен для больших n. В худшем случае требуется $n \cdot \log_2 n$ шагов, сдвигающих элементы. Среднее число перемещений примерно равно $(n/2) \cdot \log_2 n$.

Литература

1. Шилдт Г. C++ для начинающих. Серия «Шаг за шагом» / пер. с англ. – М.: ЭКОМ Паблишера, 2013. – 640 с.: ил.
2. Кушниренко А.Г., Лебедев Г.В. Программирование для математиков: Учеб. пособие для вузов – М.: Наука. Гл. ред. физ.-мат. лит. 1988. – 384 с.
3. Подбельский В.В., Фомин С.С. Программирование на языке Си: Учебное пособие. – 2-е доп. изд. – М.: Финансы и статистика, 2003. 600с.: ил.
4. Березин Б.И., Березин С.Б. Начальный курс С и С++. – М.: Диалог-МИФИ, 2005. – 288 с.

Электронное учебное издание

Александр Юрьевич **Игумнов**

ОСНОВЫ ПРОГРАММИРОВАНИЯ (ЯЗЫК С++)

Учебное пособие

Электронное издание сетевого распространения

Редактор Матвеева Н.И.

Темплан 2021 г. Поз. № 5.
Подписано к использованию 18.06.2021. Формат 60x84 1/16.
Гарнитура Times. Усл. печ. л. 4,2.

Волгоградский государственный технический университет.
400005, г. Волгоград, пр. Ленина, 28, корп. 1.

ВПИ (филиал) ВолгГТУ.
404121, г. Волжский, ул. Энгельса, 42а.