Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра вычислительных методов и программирования

А. А. Навроцкий

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ В СРЕДЕ VISUAL C++

*Рекомендовано УМО по образованию
в области информатики и радиоэлектроники
в качестве учебно-методического пособия*

Минск БГУИР 2014

УДК 004.4’6(076)
ББК 32.973.26-018.2я73
Н15

Р е ц е н з е н т ы:

доцент кафедры прикладной информатики учреждения образования «Белорусский государственный аграрный технический университет», кандидат технических наук А. И. Шакирин;

доцент кафедры информационных технологий в образовании учреждения образования «Белорусский государственный педагогический университет имени Максима Танка», кандидат технических наук В. И. Новиков

Навроцкий, А. А.

Н15 Основы алгоритмизации и программирования в среде Visual C++ : учеб.-метод. пособие / А. А. Навроцкий. – Минск : БГУИР, 2014. – 160 с. : ил.

ISBN 978-985-543-006-4.

Содержит теоретические сведения о языке C++. Рассмотрены примеры написания программ в среде Microsoft Visual Studio C++. Представлены задания для лабораторных работ.

Адресовано студенам 1–2 курса университета по учебной дисциплине «Основы алгоритмизации и программирования».

УДК 004.4’6(076)
ББК 32.973.26-018.2я73

ISBN 978-985-543-006-4

СОДЕРЖАНИЕ

ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

1. Базовые элементы языка C++

Алфавит языка Си состоит из прописных и строчных букв латинского алфавита, арабских чисел, специальных символов, пробельных и разделительных символов.

Из символов алфавита формируются лексемы (элементарные конструкции языка). К лексемам относятся: идентификаторы, зарезервированные слова, знаки операций, константы, разделители.

1.1. Идентификаторы

Идентификатор – последовательность цифр и букв латинского алфавита, а также специальных символов при условии, что первой стоит буква или знак подчеркивания. Два идентификатора, для образования которых используются совпадающие строчные и прописные буквы, считаются различными.

	Например: aaa, Aaa, aAa, AaA − четыре различных идентификатора

В идентификаторе допустимо использовать любое количество символов, однако значимыми считаются только первые 32.

При выборе идентификатора необходимо:

– следить, чтобы идентификатор не совпадал с ключевыми зарезервированными словами и именами библиотечных функций;

– использовать с осторожностью символ подчеркивания в качестве первого символа идентификатора и комбинацию «_t» в конце идентификатора, т. к. такие идентификаторы были зарезервированы стандартом ANSI Си для использования разработчиками компиляторов.

При написании идентификаторов желательно придерживаться следующих общепринятых соглашений:

– имена переменных и функций пишутся строчными буквами;

– имена типов начинаются с прописной буквы;

– имена констант пишутся прописными буквами.

При написании идентификаторов обычно используют имена, отражающие внутреннюю суть объекта.

1.2. Ключевые слова

Список ключевых слов, определенных стандартом ANSI Си: auto, double, int, struct, break, long, switch, register, typedef, char, extern, return, void, case, float, unsigned, default , for, signed, union, do, if, sizeof, else, while, volatile, continue, enum, short.

1.3. Комментарии

Комментарий – текстовая или символьная информация, используемая для пояснения участков программы. Комментарии не влияют на ход выполнения программы, т. к. не являются лексемами и не включаются в содержимое исполняемого файла.

В C++ комментарии:

– начинаются последовательностью «//» и заканчиваются концом строки;

– начинаются последовательностью «/*» и заканчиваются последовательностью «*/».

1.4. Знаки операций

Знак операции – один или несколько символов, определяющих действие над операндами. Использование пробелов внутри знака операции не допускается.

1.5. Структура программы C++

Программа C++ состоит из одной или нескольких функций. Обязательным является присутствие функции main(), которой передается управление при запуске программы.

Упрощенная структура программы имеет вид:

<Директивы препроцессора>
<Описание типов пользователя>
<Прототипы функций>
<Описание глобальных переменных>
<Тела функций>

1.6. Директивы препроцессора

Препроцессор – специальная часть компилятора, обрабатывающая директивы до начала процесса компиляции программы. Директива препроцессора начинается с символа #, который должен быть первым символом строки, затем следует название директивы. В конце директивы точка с запятой не ставится. В случае необходимости переноса директивы на следующую строку применяется символ ‘\’.

Для подключения к программе заголовочных файлов используется директива **include**. Если идентификатор файла заключен в угловые скобки, то поиск файла будет вестись в стандартном каталоге, если – в дойные кавычки, то поиск проводится в следующем порядке:

– каталог, в котором содержится файл, включивший директиву;
– каталоги файлов, которые были уже включены директивой;
– текущий каталог программы;
– каталоги, указанные опцией компилятора ‘\I’;

– каталоги, заданные переменной окружения inclide.

Обработка препроцессором директивы include сводится к тому, что на место директивы помещается копия указанного в директиве файла.

Для определения символических констант используется директива define. Например, если определить в начале программы:

```
#define PI  3.14159265359
```

то во всем тексте при компиляции идентификатор PI будет заменен текстом 3.14159265359. Замена идентификатора константы не производится в комментариях и в строках. Если замещающий текст в директиве не задан, то во всем тексте соответствующий идентификатор стирается.

Описание констант с помощью директив препроцессора характерно для языка Си. В C++ рекомендуется использовать ключевое слово const, например:

```
const double pi = 3.14159265359;
```

Директива define используется также для написания макросов:

```
#define имя(параметры)реализация
```

В программе имя макроса заменяется на строку его реализации.

Например, имеется следующее макроопределение:

```
#define MAX(A,B) ((A)>(B)?(A):(B))
```

Если в программе встречается строка

```
s = MAX(a,b);
```

то перед компиляцией каждая макрокоманда заменяется макроопределением:

```
s = ((a)>(b)?(a):(b));
```

Желательно каждый параметр помещать в фигурные скобки, т. к. их отсутствие может спровоцировать ошибку.

Например создадим макрос:

```
#define SQR(A) (A*A)
```

При использовании в программе

```
s = SQR(a + b);
```

будет сформирована строка, содержащая ошибку:

```
s = a + b * a + b;
```

Надо писать следующим образом:

```
#define SQR(A) ((A) * (A))
```

тогда строка будет выглядеть следующим образом:

```
s= (a + b) * (a + b);
```

Для отмены действия директивы #define используется директива #undef. Синтаксис этой директивы следующий:

#undef идентификатор

Например: #undef MAX

Директивы могут использоваться также для условной компиляции и для изменения номеров строк и идентификатора файла.

1.7. Стандартные библиотеки C++

При создании исполняемого файла к исходной программе подключаются файлы, содержащие различные библиотеки. Такие файлы содержат уже откомпилированные функции и, как правило, имеют расширение *lib*. На этапе компиляции компоновщик извлекает из библиотечных файлов ипользуемые в программе функции. Для осуществления связи с библиотечным файлом к программе подключается заголовочный файл, который содержит информацию об именах и типах функций, расположенных в библиотеке. Подключение заголовочных файлов осуществляется с помощью директив препроцессора.

	В стандартном C++ заголовочные файлы не имеют расширения, а для файлов, унаследованных от Си, следует указывать расширение. Например, #include <math.h>.

1.8. Функции библиотеки math.h

Все аргументы в тригонометрических функциях задаются в радианах. Параметры и аргументы всех остальных функций (кроме функции abs) имеют тип **double**. Математические функции перечислены в табл. 1.1.

Таблица 1.1

Математическая функция	Функция библиотеки math.h	Содержание вычислений
1	2	3
$\|x\|$	abs(x)	Вычисление абсолютного значения целого числа. Например: s = abs(−3) → Результат $s = 3$ s = abs(3) → Результат $s = 3$ s = abs(−3.9) → Результат $s = 3$ s = abs(3.2) → Результат $s = 3$
arccos(x)	acos(x)	Вычисление значения арккосинуса числа x. Значение x может быть задано только из диапазона −1...1. В результате выполнения функции возвращается число из диапазона $-\pi/2 \ldots \pi/2$. Например: s = acos (−1) → Результат $s = 3.14159$ s = acos (0.4) → Результат $s = 1.15928$ s = acos (1.5) → Результат $s = -1.\#IND$

Продолжение табл. 1.1

1	2	3
arcsin(x)	asin(x)	Вычисление значения арксинуса числа x. Значение x может быть задано только из диапазона –1...1. В результате выполнения функции возвращается число из диапазона 0…π. Например: s = asin (−1) → Результат s = −1.5708 s = asin (0.9) → Результат s = 1.11977
arctg(x)	atan(x)	Вычисление значения арктангенса. В результате выполнения функции возвращается число из диапазона −π/2…π/2. Например: x = atan (3.5) → Результат s = 1.2925
arctg(x/y)	atan2(x,y)	Вычисление значения арктангенса двух аргументов. В результате выполнения функции возвращается число из диапазона −π…π. Если y равняется 0, то функция возвращает π/2, если $x > 0$; 0, если $x = 0$; −π/2, если $x < 0$. Например: s = atan2(4.5, 9.2) → Результат s = 0.454914 s = atan2(0, 0) → Результат s = 0 s = atan2(5.4, 0) → Результат s = 1.5708 s = atan2(−7.3, 0) → Результат s = −1.5708
Округление к большему	ceil(x)	Функция возвращает действительное значение соответствующее наименьшему целому числу, которое больше или равно x. Например: s = ceil(−3.4) → Результат s = −3 s = ceil(3.4) → Результат s = 4
cos(x)	cos(x)	Вычисление cos(x)
ch(x)	cosh(x)	Вычисление косинуса гиперболического.
e^x	exp(x)	Вычисление экспоненты числа x
$\lvert x \rvert$	fabs(x)	Вычисление абсолютного значения x
Округление к меньшему	floor(x)	Функция возвращает действительное значение, соответствующее наибольшему целому числу, которое меньше или равно x. Например: s = floor (−3.4) → Результат s = −4 s = floor (3.4) → Результат s = 3

Окончание табл. 1.1

1	2	3
Остаток от деления x на y	fmod(x,y)	Функция возвращает действительное значение, соответствующее остатку от деления x на y. Например: s = fmod (3, 4) → Результат $s = 3$ s = fmod (8, 3) → Результат $s = 2$ s = fmod (6.4, 3.1) → Результат $s = 0.2$
$\ln(x)$	log(x)	Вычисление натурального логарифма x
$\lg_{10}(x)$	log10(x)	Вычисление десятичного логарифма x
x^y	pow(x, y)	Возведение x в степень y
$\sin(x)$	sin(x)	Вычисление $\sin(x)$
$\mathrm{sh}(x)$	sinh(x)	Вычисление синуса гиперболического x
$\sqrt{x}$	sqrt(x)	Вычисление квадратного корня x
$\mathrm{tg}(x)$	tan(x)	Вычисление тангенса x
$\mathrm{tgh}(x)$	tanh(x)	Вычисление тангенса гиперболического x

1.9. Форматированный ввод/вывод данных

Функции форматированного ввода и вывода данных расположены в библиотеке *stdio.lib*.

int scanf(const char * *строка форматирования* [, *аргументы*]) – читает форматированные данные, вводимые с клавиатруы, и записывает их в место, указанное аргументом. Каждый аргумент является указателем на переменную такого же типа, что и соответствующий форматирующий символ. В случае ошибки функция возвращает значение –1 (EOF).

Строка форматирования состоит из трех видов символов:

− спецификаторы формата;

− символы, не являющиеся разделителями (за исключением символа ‘%’);

− символы-разделители (пробел ‘ ’, табуляция ‘\t’, переход на следующую строку ‘\n’).

Спецификатор формата начинается с символа ‘%’ и определяет тип считываемых аргументов. Символы формата для функции scanf приведены в табл. 1.2.

Таблица 1.2

Символ формата	Значение	Тип аргумента
1	2	3
c	Чтение одного символа	char
d	Чтение целого десятичного числа	int
i	Чтение целого числа в десятичном, восьмеричном или шестнадцатеричном формате	int

Окончание табл. 1.2

1	2	3
e, f , g	Чтение действительного числа	float
le, lf , lg	Чтение действительного числа	double
o	Чтение восьмеричного числа	int
s	Чтение строки символов	char *
x	Чтение шестнадцатеричного числа	int
u	Чтение десятичного целого числа без знака	unsigned int
p	Чтение значения указателя	void *
n	Получает число прочитанных символов	int

Целое число, расположенное между символом ‘%’ и символом формата, позволяет указать максимальное число считываемых и передаваемых в аргумент символов.

Если очередной символ в строке форматирования *не является символом-разделителем* или спецификатором формата, то функция scanf сравнивает его с текущим символом во входном потоке: в случае совпадения – пропускает, иначе прекращает свою работу.

int printf (const char * *строка форматирования* [, *аргументы*]) – выводит форматированные данные на экран.

Строка форматирования состоит из:

− символов, которые непосредственно выводятся на экран;

− управляющих символов;

− спецификаторов формата.

Управляющие символы приведены в табл. 1.3.

Таблица 1.3

Символ	Действие
\a	Сигнал
\b	Шаг назад
\f	Перевод страницы
\n	Переход на начало следующей строки
\t	Табуляция
\r	Возврат каретки
\v	Вертикальная табуляция
\\	Обратная косая черта
\’	Одиночная кавычка
\”	Двойная кавычка
\?	Знак вопроса
\0	Нулевой байт (каждый символ равен 0)

Общий вид этого спецификатора формата:

% [*флаг*] [*ширина*] [.*точность*] <*символ_формата*>

Параметр *флаг* определяет выравнивание числа при выводе. Возможные значения флага приведены в табл. 1.4.

Таблица 1.4

Флаг	Назначение
–	Выравнивает выводимое число по левому краю поля
+	Всегда будет выводиться знак числа
Пробел	Устанавливает пробел перед положительным числом и минус перед отрицательным
#	Выводит 0 перед восьмеричным числом, 0*x* – перед шестнадцатеричным

Параметр *ширина* определяет минимальное количество выводимых символов.

Параметр *точность* имеет различное назначение для различных типов выводимых данных. Для действительных чисел, выводимых с использованием спецификатора «%f» и «%e», точность определяет количество десятичных разрядов, а для спецификатора «%g» – количество значащих цифр. При выводе строк точность определяет максимальную длину поля вывода, а при выводе целых чисел − максимальное количество цифр.

Символы формата для функции scanf приведены в табл. 1.5.

Таблица 1.5

Символ формата	Значение
c	Вывод одного символа
d	Вывод целого десятичного числа
i	Вывод целого числа в десятичном, восьмеричном или шестнадцатеричном формате
e	Вывод числа с фиксированной точкой ($\pm x.xx\ e \pm xx$)
f	Вывод числа с плавающей точкой ($\pm xx.xxx$)
g	Выбирает более короткий вывод из %*e* и %*f*
o	Вывод восьмеричного числа
s	Вывод строки символов
x	Вывод шестнадцатеричного числа
u	Вывод десятичного целого числа без знака
p	Вывод значения указателя (*XXXX* : *XXXX*)
n	Выводит число прочитанных символов

	Операторы scanf и printf были определены в языке Си, и могут использоваться в C++. Однако язык C++ имеет более удобные функции для ввода/вывода данных

1.10. Потоковый ввод/вывод данных

В языке C++ ввод/вывод данных производится с использованием потоков (библиотека *iostream.h*). ***Поток*** − это логическое устройство, которое осуществляет передачу данных от источника к приемнику. В библиотеке *iostream.h* определены четыре страндартных потока:

cout − стандартный поток ввода, направлен из оперативной памяти на внешнее устройство (по умолчанию – на экран компьютера);

cin − стандартный поток вывода, направлен от внешнего устройства (по умолчанию – с клавиатуры) в оперативую память;

cerr − поток стандартной ошибки;

clog − буферизируемый поток стандартных ошибок.

Для работы с потоками применяются операции вставки в поток (<<) и извлечения из потока (>>).

Введем, например, с клавиатуры переменную *x* и выведем ее на экран:

```
cout << "Vvedie x " << endl;
  cin >> x;
cout << "x =  " << x << endl;
```

Здесь манипулятор endl переводит курсор в начало следующей строки.

	В языке Си для перехода на новую строку использовали управляющий символ ‘\n’. Однако манипулятор endl кроме переноса строки производит и сброс буферов потока вывода, что повышает надежность программы, но несколько снижает скорость ее выполнения.

Для управления вводом/выводом используются флаги форматированного ввода/вывода или манипуляторы форматирования.

Флаги устанавливают параметры ввода/вывода, которые будут действовать на все последующие операторы до тех пор, пока не будут отменены.

Для установки флага вывода используется конструкция

cout.setf(ios::*флаг*)

Для снятия флага используется конструкция

cout.unsetf(ios::*флаг*)

Если необходимо установить несколько флагов, то можно использовать операцию «или» (|), например:

cout.setf(ios:: *флаг1* | ios:: *флаг2* | ios:: *флаг3*)

В табл. 1.6 приведены некоторые флаги для форматированного ввода.

Таблица 1.6

Флаг	Описание
right	Выравнивание по правой границе
left	Выравнивание по левой границе (по умолчанию)
boolalpha	Вывод логических величин в текстовом виде
dec	Вывод величин в десятичной системе счисления (по умолчанию)
showpos	Выводит символ ‘+’ для положительных чисел
scientific	Вывод вещественных чисел в экспоненциальной форме
fixed	Фиксированная форма вывода вещественных чисел (по умолчанию)

Манипуляторы (библиотека *iomanip.lib*) помещаются в операторы ввода/вывода непосредственно перед форматируемым значением. В табл. 1.7 приведены некоторые манипуляторы форматирования.

Таблица 1.7

Манипулятор	Описание
setw(n)	Задает ширину поля вывода в *n* символов
setprecision(n)	Задает количество цифр (*n*–1) в дробной части числа
left	Выравнивание числа по левой границе (по умолчанию)
right	Выравнивание числа по правой границе
boolalpha	Вывод логических величин в текстовом виде
noboolalpha	Вывод логических величин в числовом виде
showpos	Выводит символ ‘+’ для положительных чисел
noshowpos	Не выводит символ ‘+’ для положительных чисел
scientific	Экспоненциальная форма вывода вещественных чисел
fixed	Фиксированная форма вывода вещественных чисел (по умолчанию)
setfill(ch)	Установить символ *ch* как заполнитель

Установить ширину поля вывода можно также следующим образом:
cout.width(n) − устанавливает ширину поля вывода, равную *n* позиций;
cout.presicion(m) − определяет *m* цифр в дробной части числа.

2. Базовые типы данных

2.1. Типы данных

Тип данных позволяет определить, какие значения могут принимать переменные, какая структура и какое количество ячеек используется для их размещения и какие операции допустимо над ними выполнять

Данные можно разбить на две группы: скалярные (простые) и структурированные (составные).

К ***скалярному типу*** относятся данные, представляемые одним значением (числом, символом) и размещаемые в одной ячейке из нескольких байтов.

Структурированные типы определяются пользователем как комбинация скалярных и описанных ранее структурированных типов.

Базовыми типами данных являются: целый, действительный и символьный тип.

Данные в силу особенностей их использования могут быть ***константами*** и ***переменными.*** В отличие от переменных константы не могут изменять свое значение во время выполнения программы.

2.2. Объявление переменных и констант

Объявление переменных можно сделать в любом месте программы до первого их использования, однако для повышения читабельности программы желательно это делать в начале программы.

При объявлении сначала указывается тип данных, а затем через запятую список переменных данного типа. Например:

```
int  x, y, k;
double  s, z;
```

Можно использовать две формы объявления переменных:

– объявление, не приводящее к выделению памяти;

– объявление, при котором будет выделена память в соответствии с указанным типом.

При объявлении с выделением памяти можно сразу инициализировать (присвоить определенное начальное значение) переменную. Например:

```
int k=3, m=34;
```

Для объявления констант используется ключевое слово const:

```
const double pi = 3.14159265359;
```

2.3. Целый тип данных

Целый тип данных характеризуется отсутствием в значении дробной части. В языке C++ используются следующие целые типы данных:

int – системно-зависимая переменная (имеет различную длину в различных системах). В 32-разрядных системах имеет длину от –2 147 483 648 до 2 147 483 647 и занимает 4 байта памяти;

long – системно-независимая переменная (имеет постоянную длину в различных системах). Занимает 4 байта памяти и имеет длину от –2 147 483 648 до 2 147 483 647. В 32-разрядных системах совпадает с типом **int**;

short – системно-независимая переменная (имеет постоянную длину в различных системах). Занимает 2 байта памяти и имеет длину от –32,768 до 32,767. В 16-разрядных системах совпадает с типом **int**. Данный тип использовать нежелательно, т. к. имея меньшую длину, он обрабатывается медленнее типа **int**.

Для смещения границ диапазона только в положительную область используется атрибут unsigned. Например, unsigned **int** имеет длину от 0 до 4 294 967 295.

Константы целого типа – последовательность цифр, начинающаяся со знака минус для отрицательных констант, и со знака плюс или без него для положительных констант. Для обозначения констант типа long после числа ставят букву L или l.

Константы могут быть представлены в различных системах счисления.

Десятичные константы: последовательность чисел от 0 до 9, начинающаяся не с нуля, например, 334.

Восьмеричные константы: последовательность чисел от 0 до 7, начинающаяся с нуля, например, 045.

Шестнадцатеричные константы: последовательность чисел от 0 до 9 и букв от A до F, начинающаяся с символов 0x, например 0xF5C3.

2.4. Символьный тип данных

Символьный тип предназначен для хранения одного символа, для чего достаточно выделить 1 байт памяти. Данные такого типа рассматриваются компилятором как целые, поэтому в переменных типа signed char можно хранить целые числа из диапазона –128…127. Для хранения символов используется unsigned char, который позволяет хранить 256 символов кодовой таблицы *ASCII* (*American Standard Code for Information Interchange* – Американский стандартный код для обмена информацией). Стандартный набор символов *ASCII* использует только 7 битов для каждого символа (диапазон 0…127). Добавление 8-го разряда позволило увеличить количество кодов таблицы *ASCII* до 255. Коды от 128 до 255 представляют собой расширение таблицы *ASCII* для хранения символов национальных алфавитов, а также символов псевдографики.

Значения кодовой таблицы *ASCII* с номерами 0…32 и 127 содержат непечатаемые символы, которые не имеют графического представления, но влияют на отображение текста. Символы с кодами 32…127 представлены в табл. 2.1. Символы с кодами 128…255 (кодовая таблица 866 – *MS-DOS*) представлены в табл. 2.2.

Таблица 2.1

Код	Символ	Код	Символ	Код	Символ	Код	Символ
32	пробел	56	8	80	P	104	h
33	!	57	9	81	Q	105	i
34	“	58	:	82	R	106	j
35	#	59	;	83	S	107	k
36	$	60	<	84	T	108	l
37	%	61	=	85	U	109	m
38	&	62	>	86	V	110	n
39	'	63	?	87	W	111	o
40	(	64	@	88	X	112	p
41	)	65	A	89	Y	113	q
42	*	66	B	90	Z	114	r
43	+	67	C	91	[	115	s
44	,	68	D	92	\	116	t
45	-	69	E	93	]	117	u
46	.	70	F	94	^	118	v
47	/	71	G	95	_	119	w
48	0	72	H	96	`	120	x
49	1	73	I	97	a	121	y
50	2	74	J	98	b	122	z
51	3	75	K	99	c	123	{
52	4	76	L	100	d	124	|
53	5	77	M	101	e	125	}
54	6	78	N	102	f	126	~
55	7	79	O	103	g	127	del

Таблица 2.2

Код	Символ	Код	Символ	Код	Символ	Код	Сим-вол
128	А	160	а	192	└	224	р
129	Б	161	б	193	┴	225	с
130	В	162	в	194	┬	226	т
131	Г	163	г	195	├	227	у
132	Д	164	д	196	─	228	ф
133	Е	165	е	197	┼	229	х
134	Ж	166	ж	198	╞	230	ц
135	З	167	з	199	╟	231	ч
136	И	168	и	200	╚	232	ш
137	Й	169	й	201	╔	233	щ
138	К	170	к	202	╩	234	ъ

Окончание табл. 2.2

Код	Символ	Код	Символ	Код	Символ	Код	Символ
139	Л	171	л	203	╦	235	ы
140	М	172	м	204	╠	236	ь
141	Н	173	н	205	═	237	э
142	О	174	о	206	╬	238	ю
143	П	175	п	207	╧	239	я
144	Р	176	▒	208	╨	240	Е
145	С	177	▒	209	╤	241	е
146	Т	178	▓	210	╥	242	Є
147	У	179	│	211	╙	243	є
148	Ф	180	┤	212	╘	244	Ї
149	Х	181	╡	213	╒	245	ї
150	Ц	182	╢	214	╓	246	Ў
151	Ч	183	╖	215	╫	247	ў
152	Ш	184	╕	216	╪	248	°
153	Щ	185	╣	217	┘	249	·
154	Ъ	186	║	218	┌	250	·
155	Ы	187	╗	219	█	251	√
156	Ь	188	╝	220	▄	252	№
157	Э	189	╜	221	▌	253	¤
158	Ю	190	╛	222	▐	254	■
159	Я	191	┐	223	▀	255	

Переменные символьного типа записываются в одиночных кавычках.

2.5. Вещественный тип данных

Вещественный тип данных характеризуется присутствием в числе дробной части. Число представляется в экспоненциальной форме: ***±n.mE±p***, где ***n.m*** – мантисса (*n* – целая часть, *m* – дробная часть), ***p*** – порядок.

В языке С++ используются следующие типы вещественных данных:

float – хранит числа, занимающие 4 байта памяти и находящиеся в интервале от $3.4 \cdot 10^{-38}$ до $3.4 \cdot 10^{+38}$. Данный тип позволяет хранить числа с точностью до 7 знаков после запятой;

double – хранит числа, занимающие 8 байт памяти и находящиеся в интервале от $1.7 \cdot 10^{-308}$ до $1.7 \cdot 10^{+308}$. Данный тип позволяет хранить числа с точностью до 15 знаков после запятой;

long double – характеристики зависят от типа компилятора и, как правило, совпадают с типом **double**.

Вещественное число хранится в памяти компьютера в нормализованной форме (больше единицы и меньше двойки). При нарушении нормализации мантиссу сдвигают влево до тех пор, пока старшей цифрой мантиссы не станет единица. Так как первая цифра нормализованной мантиссы всегда равна еди-

нице, то ее можно не хранить в памяти, а сэкономленный бит использовать для повышения точности представления числа. Единица присутствует в записи числа неявно и называется *неявной единицей*. Порядок числа хранится в сдвинутой форме таким образом, чтобы весь интервал значений находился в положительной области. Таким образом экономится еще один бит.

При определении вещественных констант в конце добавляется буква F для типа float, D – для типа **double** (необязательно) и L для типа long **double**.

2.6. Логический тип данных

Логический тип bool может принимать два значения: *true* (1) или *false* (0). Для хранених данных этого типа выделяется 1 байт.

2.7. Неявное преобразование типов

В большинстве случаев преобразование типов происходит автоматически с использованием приоритета типов. Типы имеют следующий порядок приоритетов:

char → short → int → long → float → double → long double

Приоритет увеличивается слева направо (в сторону увеличения занимаемой типом памяти). При проведении арифметических операций действует правило: операнд с более низким приоритетом преобразуется в операнд с более высоким приоритетом, а значения типов char и short всегда преобразуются к типу int.

При использовании операции присваивания преобразование типов не происходит. Поэтому при присваивании переменным, имеющим меньший приоритет типа, значений переменных, имеющих больший приоритет типа, возможна потеря информации.

Например, необходимо вычислить: s = a + b, где s − переменная типа double; a – символ; b − переменная типа int. Пусть a = ’d’, b = 45. Выражение будет рассчитываться следующим образом. Так как в арифметическом выражении присутствуют две переменные различных типов, то переменная с меньшим приоритетом (a) будет приведена к типу int. Для этого в памяти компьютера создается временная переменная типа int, которая будет хранить номер символа ’d’ равный 100 (см. табл. 2.1). После этого выполняется операция суммирования, результат которой будет равен 145 (100+45). Полученное значение (145) присваивается переменной s. При выполнении операции присваивания преобразование типа не происходит, но размер переменной типа double больше переменной типа int, поэтому потеря информации в этом случае не происходит.

2.8. Явное преобразование типов

Если неявное преобразование типов не приводит к требуемому результату, программист может задать преобразование типов явным образом:

static_cast <*тип*> (*переменная*)

Из языка Си сохранилась устаревшая форма приведения типов

(*тип*) *переменная*

или

тип (*переменная*)

Пример:

```
int a,b,s,f;
        a = b = 2147483647;
        s = (a*b)/a;
        f = (static_cast <double> (a)*b)/a;
```

Результат**:** $s = 0, f = 2147483647$.

При расчете переменной s вычисленное значение произведения a на b выходит за границы диапазона значений, которые могут храниться в переменной типа **int**. Создаваемая для хранения промежуточного результата временная переменная типа **int** получает ошибочную информацию, поэтому результат вычисления будет неверным.

В следующей строке переменная a явным образом приводится к типу **double**. Следовательно, результат вычисления произведения будет храниться во временной переменной наибольшего по длине типа (из **int** и **double**) − типа **double**. Полученный результат не выходит за границы диапазона значений типа **double**, поэтому ошибка не возникает.

3. Операции в языке C++

3.1. Арифметические операции

Самыми простыми арифметическими операциями являются +, –, *, /. Данные операции применимы как к целым, так и к вещественным типам данных и правила их использования аналогичны их использованию в математических вычислениях. Порядок следования операций можно изменять с помощью скобок.

Для работы только с целыми числами существует операция получения остатка от деления – %.

Например: 10 % 6 = 4, 7 % 10 =7, 10 % 5 =0.

3.2. Операция присваивания

Формат операции:

оператор_1 = *оператор_2*;

В переменную *оператор_1* заносится значение *оператора_2*. В качестве *оператора_1* можно использовать только переменную. В качестве *оператора_2* можно использовать константу, переменную, выражение или функцию.

Допустимо использовать следующее написание:

a = b = c = d; что равнозначно a = d; b = d; c =d ;

Часто в программировании используются операции такого типа:

оператор_1 = *оператор_1* ***знак_операции*** *оператор_2*;

Для сокращения записей таких операторов можно использовать сокращенную форму записи:

оператор_1 ***знак_операции*** = *оператор_2*;

Например, оператор

s = s + 2;

можно заменить оператором

s += 2;

Если *оператор_2* для операций суммирования и вычитания равен единице, то лучше использовать операции инкремента:

оператор_1++;

или декремента

оператор_1--;

Например, вместо написания i = i + 1 можно использовать i++, а вместо написания i = i – 1 можно использовать i --.

Знак инкремента или декремента может быть записан в двух формах: в префиксной (например ++i) или в постфиксной (например i++). Способ написания влияет на порядок выполнения операций в выражении. При префиксной

форме сначала выполняется инкремент или декремент, а затем арифметические операции. Если используется постфиксная форма, то сначала выполняются арифметические операции, а затем инкремент или декремент.

3.3. Операции сравнения

Операции сравнения применяются при работе с двумя операндами и возвращают *true* (1), если результат сравнения – истина, и *false* (0) – если результат сравнения – ложь. В языке Си определены следующие операции сравнения:

< *(*меньше*)*, **<=** (меньше или равно), **>** (больше),

>= (больше или равно), **!=** (не равно), **= =** (равно).

Операнды должны иметь одинаковый тип (допустимо сравнивать целый и действительный тип).

3.4. Логические операции

Логические операции работают с операндами скалярных типов и возвращают результат логического типа. Определены три логические операции «!», «&&» и «||».

Унарная логическая операция НЕ (!) возвращает *true* (1), если операнд имеет нулевое значение, и *false*(0), если операнд отличен от нуля.

Например:

```
k = 5;
a = !(k > 0);
```

Результат: 0 (*false*), т. к. операнд имеет значение 1 (*true*), которое изменяется операцией «!» на обратное – 0 (*false*).

Логическая операция И (&&) возвращает *true* (1), если операнды имеют ненулевые значения, и *false*(0) – если хотя бы один операнд имеет нулевое значение.

Например:

```
k = 5;
a = (k > 0 && k <= 10 && k != 5);
```

Результат: 0 (*false*), т. к. два первых операнда имеют значение 1 (*true*), а последний операнд имеет значение 0 (*false*).

Если ввести:

```
k = 5;
a = (k > 0 && k <= 10 && k == 5);
```

Результат: 1 (*true*), т. к. все операнды имею значение 1 (*true*).

Логическая операция ИЛИ (||) возвращает *true* (1), если хотя бы один операнд имеет ненулевое значение, и *false*(0), если все операнды имеют нулевое значение.

Например:

```
k = 5;
```

```
a = (k > 0 || k <= 10 || k != 5);
```

Результат: 1 (*true*), т. к. два первых операнда имеют значение 1 (*true*).

Если ввести:

```
k = 5;
a = (k <= 0 || k > 10 || k != 5);
```

Результат: 0 (*false*), т. к. все операнды имею значение 0 (*false*).

Допускается смешение различных логических операций в одном выражении:

```
k = 5;
a = !(k >= 0 && k < 10 || k != 5);
```

Результат: 0 (*false*).

3.5. Поразрядные логические операции

Поразрядные логические операции работают с двоичным представлением целых чисел.

Определены следующие операции:

«~» – поразрядное отрицание;

«&» – поразрядное И;

«|» – поразрядное ИЛИ;

«^» – поразрядное исключающее ИЛИ;

«<<» – поразрядный сдвиг влево;

«>>» – поразрядный сдвиг вправо.

Унарная поразрядная операция «~» инвертирует каждый бит операнда.

Таблица истинности для операций «&», «|», «^» представлена в табл. 3.1.

Таблица 3.1

Значение бит	b_1 & b_2	b_1 | b_2	b_1 ^ b_2
$b_1 = 0, b_2 = 0$	0	0	0
$b_1 = 0, b_2 = 1$	0	1	1
$b_1 = 1, b_2 = 0$	0	1	1
$b_1 = 1, b_2 = 1$	1	1	0

Операция поразрядного сдвига «>>» сдвигает биты левого операнда на число разрядов, указанное правым операндом. Правые биты теряются. Если левый операнд является беззнаковым числом, то левые освободившиеся биты заполняются нулями. Если есть знак символа, то ячейки заполняются этим символом. Сдвиг целого числа эквивалентен целочисленному делению на 2^n.

Операция поразрядного сдвига «<<» сдвигает биты левого операнда на число разрядов, указанное правым операндом. Правые биты теряются. Левые биты теряются, а правые заполняются нулями. Если есть знак символа, то ячейки заполняются этим символом. Сдвиг целого числа эквивалентен умножению на 2^n.

3.6. Приоритет операций в C++

Приоритет операций в языке C++ представлен в табл. 3.2. Приоритет уменьшается сверху вниз.

Таблица 3.2

Уровень приоритета	Тип операции	Операторы
1	Разрешение области действия	::
2	Унарные	префикс ++, префикс --, –>
	Другие	( ), [ ], . (точка)
3	Унарные	+, –, !, *, &, sizeof, new, delete, явное преобразование типа
4	Арифметические	*, /, %
5	Арифметические	+, –
6	Поразрядный сдвиг	<<, >>
7	Сравнение	>, <, >=, <=
8	Сравнение	==, !=
9	Поразрядные логические	&
10	Поразрядные логические	^
11	Поразрядные логические	|
12	Логические	&&
13	Логические	||
14	Условная	?:
15	Присваивания	=, *=, /=, %=, +=, –=, <<=, >>=, &=, ^=, |=
16	Унарные	постфикс ++, постфикс --
	Последовательность	, (запятая)

3.7. Использование блоков

Группа операторов, заключенная в фигурные скобки, называется ***блоком***. Компилятор рассматривает такую группу операторов как один составной оператор. В любой конструкции языка C++ простой оператор можно заменить блоком. Например, вместо

оператор;

можно поставить

```
{
оператор_1;
…
оператор_n;
 }
```

4. Организация разветвляющихся алгоритмов

Алгоритм называется ***разветвляющимся***, если он содержит несколько ветвей, отличающихся друг от друга содержанием вычислений. Выход вычислительного процесса на ту или иную ветвь алгоритма определяется текущими данными.

4.1. Оператор условной передачи управления if

Формат оператора выбора:

if (*логическое_выражение***)** *оператор_1*;
else *оператор_2*;

Если *логическое_выражение* истинно, то выполняется *оператор_1*, иначе – *оператор_2*.

Например:

```
if (f > 10) x = 3;
    else x = 34;
```

Оператор имеет сокращенную форму:

if (*логическое_выражение***)** *оператор_1***;**

Например: **if** (f == 0) x = 4;

Логическое_выражение всегда располагается в круглых скобках. Если *оператор_1* или *оператор_2* содержат более одного оператора, то используется блок.

В качестве *оператора_1* и *оператора_2* могут быть использованы операторы if. Такие операторы называют *вложенными*. Во вложенных операторах if ключевое слово else принадлежит ближайшему предшествующему ему if.

Например:

```
if (логическое_выражение_1) оператор_1;
if (логическое_выражение_2) оператор_2;
   else  оператор_3;
```

Оператор_3 будет выполняться, если *логическое_выражение_2* ложно. Значение *логического_выражения_1* не оказывает влияния на работу *оператора_3*.

Изменить порядок проверки можно, используя фигурные скобки:

```
if (логическое_выражение_1) {
            оператор_1;
if (логическое выражение_2) оператор_2;
                            }
   else  оператор_3;
```

Оператор_3 будет выполняться, если *логическое_выражение_1* ложно. Значение *логического_выражения_2* не оказывает влияния на работу *оператора_3*.

4.2. Условная операция

В программировании достаточно часто распространена операция сравнения двух аргументов, например:

```
if (a > b) max = a;
    else max = b;
```

В связи с этим была разработана специальная условная операция. Ее общая форма:

условие **?** *оператор_1* **:** *оператор_2*;

Если значение *условие* истинно, то результатом операции является *оператор_1*, иначе – *оператор_2*.

Например, найти наибольшее из двух чисел:

```
max = a > b ? a : b;
```

Условие может быть любым скалярным выражением, а операторы могут иметь практически любой тип.

Применение условной операции сокращает код, однако не оказывает никакого влияния на скорость выполнения программы.

4.3. Оператор множественного выбора switch

Если в программе большое число ветвлений, которые зависят от значения одной переменной, то можно вместо вложенной последовательности конструкций if .. else использовать оператор switch. Общая форма оператора следующая:

```
switch(переменная выбора) {
        case const1: операторы_1 ;   break;
                                  …
        case constN: операторы_n;   break;
        default : операторы_n+1;
                                  }
```

Работает оператор следующим образом. Сначала анализируется значение переменной выбора и проверяется, совпадает ли оно со значением одной из констант. При совпадении выполняются операторы этого case. Конструкция default **(**может отсутствовать**)** выполняется, если результат выражения не совпал ни с одной из констант. Тип переменной выбора может быть целым, символьным или перечисляемым. Тип констант сравнения должен совпадать с типом переменной выбора.

Пример 4.1. Расшифровать оценку по пятибалльной системе:

```
int otc;
```

```
cin >> otc;
switch (otc)
{
case 2: cout << "Neud" << endl; break;
case 3: cout << "Udovl" << endl; break;
case 4: cout << "Horosho" << endl; break;
case 5: cout << "Otlichno" << endl; break;
default  : cout << "Net takoy ocenki" << endl;
}
```

Если ввести 2, то на экран будет выведено «Neud».

В конце каждого набора операторов ставится оператор break, который завершает выполнение оператора switch. Если не поставить break, то после выполнения соответствующей секции управление будет передано операторам, относящимся к другим ветвям switch. Отсутствие break, как правило, приводит к ошибке в вычислениях. Однако в некоторых случаях использование секций case без break оправдано, например, если необходимо для различных значений констант сравнения выполнять одинаковую последовательность операторов.

Пример 4.2. Определить пору года по номеру месяца.

```
int mes;
cin >> mes;
switch (mes)
{
case 12:
case 1:
case 2: cout << "Zima"; break;
case 3:
case 4:
case 5: cout << "Vesna"; break;
case 6:
case 7:
case 8: cout << "Leto"; break;
case 9:
case 10:
case 11: cout << "Osen"; break;
default  : cout << "Ne paviln. vvod";
}
```

Пример 4.3. Вычислить значение выражения

$$s=\begin{cases} f(x)\cdot\sin(x), & \text{если} \quad x+y>12, \\ e^{2x}+e^{-x}, & \text{если} \quad x+y\le 5, \\ \sqrt[3]{|y\cdot f(x)|}, & \text{иначе.} \end{cases}$$

При выполнении задания предусмотреть выбор вида функции $f(x)$: tg$x$ или x^2.

```
#include <iostream.h>
#include <math.h>

int main()
{
double x, y, f, a, res;
int k;
 cout << "Vvedite x ";       cin >> x;
 cout << "Vvedite y ";       cin >> y;
    cout << "Ishodnye dannye x = " << x << " y = " << y << endl;
 cout << "Vyberite f : 1 - tg(x), 2 - x^2 ";
  cin >> k;
 switch(k)
                {
  case 1: f = tan(x);    break;
  case 2: f = pow(x,2); break;
    default : cout << "Ne vuibrana funkciya ";  return 1;
                 }
    f = x + y;
  if (a > 12) res = f * sin(x);
          else
       if (a <= 5) res = exp(2*x) + exp(-x);
             else
                   res = pow(fabs(y*f),1./3);
   cout << "Result = " << res << endl;
return 0;
}
```

5. Организация циклических алгоритмов

5.1. Оператор цикла for

Общий вид оператора:

for (*инициализирующее_выражение*; *логическое_выражение*;
инкрементирующее_выражение)
{
Тело цикла
}

Чаще всего все три выражения содержат одну переменную, которую называют счетчиком цикла.

Инициализирующее_выражение выполняется только один раз в начале выполнения цикла. Вычисленное значение инициализирует счетчик цикла.

Логическое_выражение проверяется в начале каждого цикла. Если результат равен 1 (*true*), то цикл повторяется, иначе выполняется следующий за телом цикла оператора.

Инкрементирующее_выражение предназначено для изменения значения счетчика цикла. Модификация счетчика происходит после каждого выполнения тела цикла.

Тело цикла – последовательность операторов, которая выполняется многократно, до тех пор, пока не будет выполнено условие выхода из цикла. Тело цикла может содержать внутри себя любы конструкции языка C++, в том числе – любое количество вложенных циклов.

Схема работы цикла for представлена на рис. 5.1:

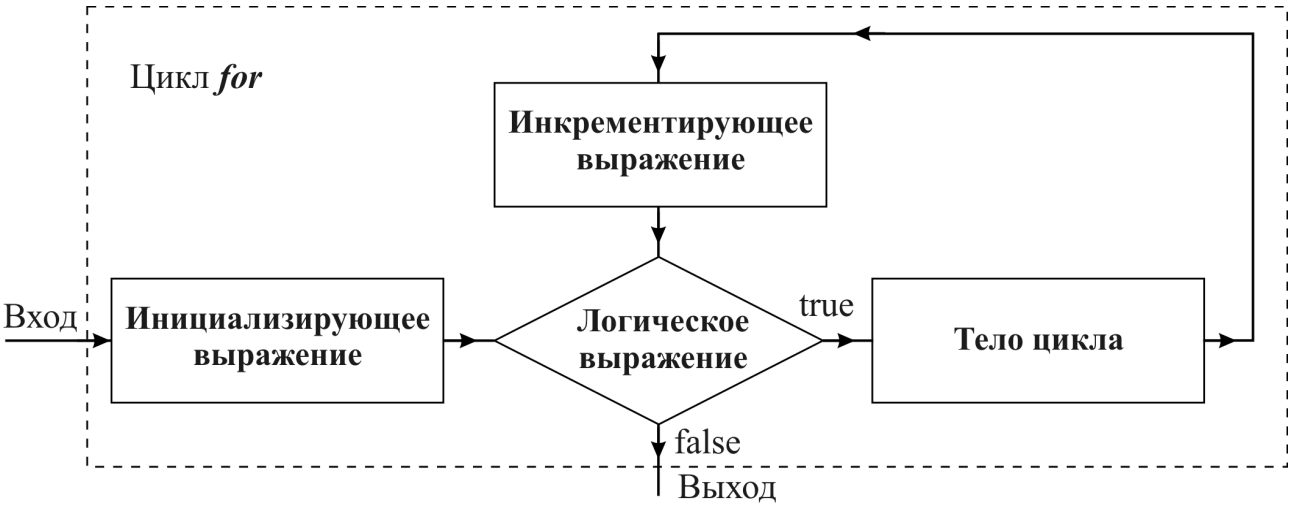


Рис. 5.1

Рассмотрим работу следующего оператора:

```
for (i = 1;   i < 10;  i++) cout << i << endl;
```

В начале цикла в переменную *i* будет занесено число 1. Затем будет проверено значение логического выражения, и т. к. оно имеет значение *true* (1 < 10), то будет выполнено тело цикла (значение *i* будет выведено на экран). После этого выполняется инкрементирующее выражение i++ и снова будет проверено значение логического выражения. Тело цикла будет выполняться до тех пор, пока логическое выражение не примет значения *false* (10 < 10). В результате на экран будут выведены цифры от 1 до 9.

Если необходимо вывести цифры от 1 до 10, то можно использовать конструкцию:

```
for (i = 1;   i <= 10;  i++)
```

Однако в C++ обычно используют конструкции со строгим неравенством:

```
for (i = 1;   i < 11;  i++)
```

Удобно совмещать выполнение инкрементирующего выражения с описанием счетчика цикла.

```
for ( int i = 1;   i < 11;  i++)
```

Такое написание удобно тем, что объявленная переменная по стандарту C++ будет существовать только внутри цикла, и далее имя этой переменной можно использовать для других целей.

В своей версии языка C++ фирма Microsoft область видимости переменной, объявленной внутри оператора for, расширила до конца блока, содержащего этот for.

Любая из секций в операторе for не является обязательной, поэтому может отсутствовать одно или несколько выражений. Допустимо такое написание бесконечного цикла:

```
for ( ; ; )
```

Для размещения в одной секции оператора for несколько операторов используется операция «запятая», которая позволяет в тех местах, где допустимо использование только одного оператора, размещать несколько операторов. Формат операции

Оператор_1, Оператор_2, …, Оператор_N

Программа для вычисления факториала числа *n* может выглядеть следующим образом:

```
for (int f=1, i=1;  i<=n;  f*=i, i++);
```

Точка с запятой в конце оператора for означает, что тело цикла отсутствует. Так как в Visual C++ область видимости переменных f и i продлена до конца следующего блока, то значение f можно вывести во внешнем по отношению к for блоке.

5.2. Оператор цикла while

Оператор цикла с предусловием

```
while (логическое_выражение)
   {
Тело_цикла
    }
```

организует повторение операторов тела цикла до тех пор, пока значение логического выражения истинно. Как только значение логического выражения становится равным 0 (*false*), циклический процесс прекращается и выполняется первый после цикла оператор. Если условие цикла сразу равно 0 (*false*), то тело цикла не выполняется ни разу.

5.3. Оператор цикла do-while

Оператор цикла с постусловием

```
do {
Тело_цикла
   }
       while (логическое_выражение);
```

организует повторение операторов тела цикла до тех пор, пока значение логического выражения истинно. Как только значение логического выражения становится равным 0 (*false*), циклический процесс прекращается и выполняется первый после цикла оператор. Вне зависимости от значения логического выражения тело цикла будет выполнено не менее одного раза.

	Оператор цикла do-while опасен тем, что тело цикла обязательно выполняется хотя бы один раз. Следовательно, необходимо проверять условие его завершения до входа в цикл. Поэтому, если это возможно, следует избегать использования этого оператора.

5.4. Операторы и функции передачи управления

Операторы и функции передачи управления позволяют изменить стандартный порядок выполнения операторов.

5.4.1. Оператор continue

Используется при организации циклических процессов. Операторы тела цикла, находящиеся после оператора continue, пропускаются, а управление передается следующему циклу. Оператор continue обычно используется вместе с

оператором if для того, чтобы при определенных значениях данных завершить текущий цикл и передать управление следующему циклу.

5.4.2. Оператор **break**

Позволяет перейти к следующему за блоком оператору. Например, в циклах он обеспечивает досрочный выход из цикла, а в операторе switch – выход из блока выбора. Следует обратить внимание на то, что оператор break выходит только из текущего блока, т. е. в случае вложенных циклов выход происходит только из одного цикла.

5.4.3. Оператор **return**

Завершает выполнение текущей функции и передает управление вызывающей функции. Управление передается следующему за вызовом текущей функции оператору.

Формат оператора:

return *выражение*;

Если значение выражения задано, то результат возвращается в вызывающую функцию в качестве значения вызываемой функции.

5.4.4. Функция **exit**

Находится в библиотеке *stdlib.lib*. Корректно прерывает выполнение программы, записывая все буферы, закрывая все потоки. Формат функции:

void exit(**int**)

Параметр является служебным сообщением системе. Как правило, 0 говорит об успешном завершении программы, ненулевые значения – об ошибке.

5.4.5. Функция **abort**

Находится в библиотеке *stdlib.lib*. Генерирует «молчаливое» исключение, не связанное с сообщением об ошибке. Корректно прерывает выполнение программы, записывая все буферы, закрывая все потоки. Формат функции:

void abort(void)

5.4.6. Оператор безусловного перехода **goto**

Передает управление оператору, помеченному меткой. Использование оператора goto существенно снижает читабельность программы и увеличивает вероятность ошибки. Поэтому использование goto в программах нежелательно.

Примером обоснованного применения оператора безусловного перехода может служить необходимость организации выхода сразу из нескольких вложенных циклов:

```
for ( i = 0; i < n; i++)
  for (j = 0; j < m; j++) {
        if (логическое_выражение) goto met;
                    }
met:  …
```

5.5. Организация циклических алгоритмов

Пример 5.1. Вывести таблицу значений функции $y(x) = \sin(x)$ на интервале от a до b с шагом h.

Вариант 1 (с использованием оператора цикла for).

```
for (double x=a; x<b+h/2; x+=h)
cout << "x = " << x << " y = " << sin(x) << endl;
```

Логическое выражение в операторе равно x < b+h/2, а не x<=b. Это вызвано следующим. Счетчик цикла на x каждом шаге увеличивает свое значение на h. Если h – дробное число, то в переменной x могут накапливаться ошибки округления, которые приводят к тому, что значение x будет, например, равно 2.00000000000000001, в то время, когда значение b = 2.0. Результат операции сравнения x <= b в этом случае будет иметь значение *false*, и, следовательно, последнее значение таблицы не будет выведено на экран. Для того чтобы гарантировать выполнение последней итерации циклического процесса, значение правой границы интервала увеличивается на величину, не превышающую h (например на h/2).

Вариант 2 (с использованием оператора цикла while).

```
x = a;
while(x<b+h/2)
{
   cout << "x = " << x << " y = " << sin(x) << endl;
  x += h;
}
```

Пример 5.2. Вычислить интеграл $s = \int_0^{2\pi} \sin x \, dx$ методом средних.

```
    h=(b-a)/100;
for (x=a+h/2, s=0; x<b; s+=sin(x)*h, x+=h) ;
```

Пример 5.3. Вычислить сумму $s(x) = \sum_{k=1}^{100} (-1)^k \frac{x^k}{k!}$.

Вначале необходимо получить рекуррентную формулу. Для получения формулы вычисляются значения слагаемых при различных значениях k: при $k = 1;\ a_1 = -1\frac{x}{1};$ при $k = 2;\ a_2 = 1\frac{x \cdot x}{1 \cdot 2};$ при $k = 3;\ a_3 = -1\frac{x \cdot x \cdot x}{1 \cdot 2 \cdot 3}$ и т. д. Видно, что на каждом шаге слагаемое дополнительно умножается на $-1\frac{x}{k}$. Исходя из этого формула рекуррентной последовательности будет иметь вид $a_k = -a_{k-1}\frac{x}{k}$.

Полученная формула позволяет избавиться от многократного вычисления факториала и возведения в степень.

```
s =0;                  // Начальное значение суммы
a=1;          // Начальное значение для вычисление очередного
              //  члена рекуррентной последовательности
 for ( int k=1; k<=100; k++)
{
 a*=-x/k;     // Вычисление очередного члена
              // рекуррентной последовательности
 s+=a;        // Суммирование всех слагаемых
}
```

Пример 5.4. Вычислить сумму $s(x)=\sum_{k=0}^{100}(-1)^k\frac{x^{2k}}{(2k)!}\sin(x)$.

В данной формуле получить рекуррентную зависимость для sin(x) сложно, поэтому функция sin(x) будет рассчитываться отдельно (как нерекуррентная часть). Для оставшейся части формулы $\sum_{k=0}^{100}(-1)^k\frac{x^{2k}}{(2k)!}$ рассчитываются значения слагаемых при различных значениях k: при $k=0$; $a_1=1\frac{1}{1}$; при $k=1$; $a_1=-1\frac{\boldsymbol{x^2}}{1\cdot\boldsymbol{2}}$; при $k=2$; $a_2=+1\frac{x^2\cdot\boldsymbol{x^2}}{1\cdot2\cdot\boldsymbol{3}\cdot\boldsymbol{4}}$; при $k=3$; $a_3=-1\frac{x^2\cdot x^2\cdot\boldsymbol{x^2}}{1\cdot2\cdot3\cdot4\cdot\boldsymbol{5}\cdot\boldsymbol{6}}$ и т. д. Формула рекуррентной последовательности будет иметь вид $a_k=-a_{k-1}\frac{x^2}{(2k-1)\cdot(2k)}$. Расчет удобно начинать не с нулевого элемента, а с первого. Поэтому значение нулевого элемента рассчитывается вручную и подставляется в начальное значение суммы.

Текст программы:

```
s = sin(x);                 // Значение суммы для нулевого элемента
  a = 1;
 for (int k=1; k<=100; k++)
{
  a *= -sqr(x)/(2*k*(2*k-1));
  s += a*sin(x);            // Здесь добавлена нерекуррентная часть
 }
```

6. Использование массивов

Массив – структура однотипных данных, каждый элемент которой хранится в отдельной ячейке, доступ к которой осуществляется по ее номеру. Массив характеризуется: именем массива, типом хранимых данных, размером (количеством элементов) и размерностью (формой представления элементов массива). Номер ячейки массива называется *индексом*. Индексы массивов должны иметь целый тип, а элементы массивов могут иметь любой тип.

6.1. Одномерные массивы

Объявление одномерного массива:

тип имя_массива [**размер**];

Пример объявления массива:

```
int c[4];
```

Размер статического массива задается константой или константным выражением целого типа.

Индексы в языке C/C++ начинаются с 0. Например, вышеобъявленный массив состоит из четырех элементов: c[0], c[1], c[2] и c[3]. Расположение элементов массива в памяти указано на рис. 6.1.

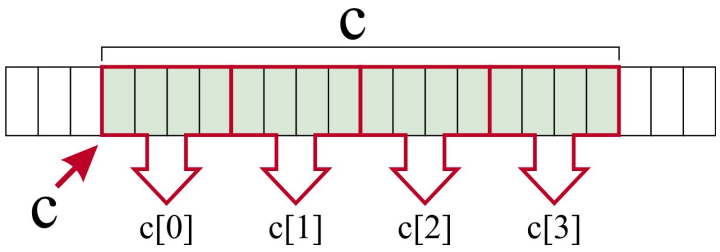


Рис. 6.1

Одновременно с объявлением можно инициализировать элементы массива:

```
double b[4]  = {1.5, 2.5, 3.75, 3.04};
     int a[4] = {1, 4};
```

Если в группе инициализации не хватает начальных значений, то оставшиеся элементы заполняются нулями, например, массив *a*: a[0] = 1, a[1] = 4, a[2] = 0 и a[3] = 0.

При объявлении со списком инициализации количество элементов можно не указывать. В этом случае размер массива будет равен количеству начальных значений. Объявление

```
char mc[ ] = {'3', 'f', w}
```

создаст массив из трех элементов.

Обращение к элементу массива происходит через указание имени массива и в квадратных скобках номера элемента массива. Например:

```
x = a[3];    a[4] = b[0] + a[2];
```

6.2. Алгоритмы работы с одномерными массивами

Пример 6.1. Ввод и вывод одномерного массива.

```
int s[10],i,j,n;
// Ввод одномерного массива
cout << "Vvedite razmer";
cin >> n;
      for (i=0; i<n; i++)
            {
      cout << "Vvedite s[" << i << "]= " ;
      cin >> s[i];
            }
// Вывод одномерного массива

      for (i=0; i<n; i++)
       {
            cout << s[i] << " " ;
       }
```

Пример 6.2. Нахождение суммы и произведения элементов одномерного массива.

```
   s = 0; p = 1;
 for (i=0; i<n; i++)
{
   s += a[i];
   p *= a[i];
}
```

Пример 6.3. Нахождение минимального и максимального элементов одномерного массива.

```
   min = max = a[0];
 for (i=1; i<n; i++)
{
if (a[i] < min) min = a[i];
```

```
if (a[i] > max) max = a[i];
}
```

Пример 6.4. Удаление из одномерного массива всех отрицательных элементов.

```
for (i=0; i<n; i++)
  if (a[i] < 0)
  {
    for (j=i+1; j<n; j++) a[j-1]=a[j];
 n--;    i--;
  }
```

6.3. Многомерные массивы

Объявление одномерного массива:

тип имя_массива [*размер_1*] [*размер_2*] … [*размер_N*];

Пример объявления двумерного массива:

```
int m[4][5];
```

Здесь объявлен двумерный массив из 4 · 5 = 20 элементов.

Можно одновременно с объявлением инициализировать элементы массива:

```
int s[2][3] = { {3, 4, 2}, {6, 3, 4} };
```

В одномерном массиве первый индекс является номером строки, а второй – номером столбца. Поэтому, например, значение элемента s[1][0] равно 6. Математически массив *s* представляет собой матрицу вида

3 4 2
6 8 5

В памяти компьютера такой массив располагается последовательно по строкам (рис. 6.2).

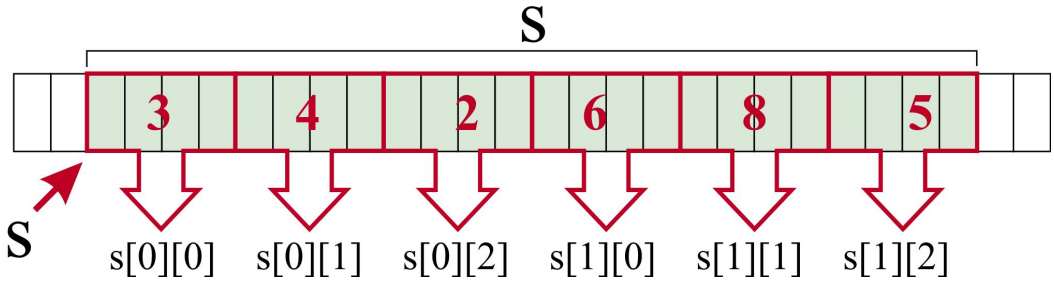


Рис. 6.2

Обращение к элементу двумерного массива происходит через указание имени массива и в квадратных скобках номеров строк и столбцов массива. Например:

```
x = s[0][2];
s[1][2] = m[3][2] + s[0][1];
```

6.4. Алгоритмы работы с двумерными массивами

Пример 6.5. Ввод и вывод двумерного массива целых чисел.

```
      // Ввод двумерного массива
cout << "Vvedite razmer n m" << endl;
cin >> n >> m;

for (i=0; i<n; i++)
 for (j=0; j<m; j++)
{
cout << "Vvedite s[" << i << "][" << j << "]: " ;
cin >> s[i][j];
}
      // Вывод двумерного массива
for (i=0; i<n; i++)
{
 for (j=0; j<m; j++)
cout << setw (9) << s[i][j] << " ";
 cout << endl;
}
```

Пример 6.6. Вывод двумерного массива из действительных чисел.

```
for (i=0; i<n; i++)
{
  for (j=0; j<m; j++)
  cout << setiosflags (ios :: fixed) <<
    setw(10) << setprecision(3)<< s[i][j] << " ";
  cout << endl;
}
```

Пример 6.7. Заполнение двумерного массива случайными действительными числами из диапазона *n*min…*n*max.

Для генерации случайных чисел используют функции библиотеки *stdlib.lib*:

rand() – генерирует псевдослучайное число из диапазона 0…RAND_MAX (32767).

srand() – устанавливает аргумент для создания новой последовательности псевдослучайных целых чисел, возвращаемых функцией **rand()**.

```
srand( time( NULL ) );
for (i=0; i<n; i++)
 for (j=0; j<m; j++)
 s[i][j] = static_cast <double> (rand()) /
                                (RAND_MAX + 1) * (nmax - nmin) + nmin;
```

Пример 6.8. Нахождение суммы элементов, лежащих на побочной диагонали.

```
s = 0;
for (i=0; i<n; i++)  s += a[i][n-i-1];
```

Пример 6.9. Перестановка строк с номерами $k1$ и $k2$.

```
for (j=0; j<m; j++)
 {
     t = a[k1][j];
  a[k1][j] = a[k2][j];
  a[k2][j] = t;
 }
```

Пример 6.10. Нахождение суммы элементов, лежащих выше главной диагонали.

```
s = 0;
for (i=0; i<n-1; i++)
 for (j=i+1;  j<m;  j++)
 s += a[i][j];
```

Пример 6.11. Упорядочение столбцов матрицы по неубыванию их максимальных элементов.

```
   for (i=0; i<n; i++)
   {
b[i] = a[i][0];
   for (j=1; j<m; j++)
    if (a[i][j] > b[i]) b[i] = a[i][j];
   }

for (i=0; i<n-1; i++)
for (j=i+1; j<m; j++)
```

```
  if (b[i] > b[j])
  {
 t = b[i];
 b[i] = b[j];
 b[j] = t;
            for (k=0; k<m; k++)
 {
t = a[i][k];
a[i][k] = a[j][k];
a[j][k] = t;
 }
  }
```

Пример 6.12. Получение из матрицы *n*-го порядка матрицы порядка *n*–1 путем удаления из исходной матрицы строк и столбцов, на пересечении которых расположен элемент с наименьшим значением

```
imin=jmin=0;
for (i=0; i<n; i++)
  for (j=0; j<m; j++)
  if (a[i][j] < a[imin][jmin]) {imin = i; jmin = j;}

for (i=0; i<n; i++)
  for (j=jmin; j<m-1; j++) a[i][j] = a[i][j+1];
   m--;
for (j=0; j<m; j++)
  for (i=imin; i<n-1; i++) a[i][j] = a[i+1][j];
  n--;
```

7. Использование указателей

7.1. Объявление указателя

Память компьютера представляет собой массив последовательно пронумерованных ячеек. При объявлении данных в памяти выделяется непрерывная область для их хранения. Например, для переменной типа **int** выделяется участок памяти размером 4 байта. Номер первого байта, выделенного под переменную участка памяти, называется адресом этой переменной.

Указатель – это переменная, значением которой является адрес участка памяти. Формат объявления указателя:

*Тип_данного *имя_указателя*;

Например:

```
int *a;    double *b, *d;    char *c;
```

На один и тот же участок памяти может ссылаться любое число указателей, в том числе и различных типов. Допустимо описывать переменные типа указатель на указатель (указатель на ячейку памяти, которая в свою очередь содержит адрес другой ячейки памяти). Например:

```
int *um1, **um2, ***um3;
```

В языке Си существует три вида указателей:

1. Указатель на объект известного типа.
2. Указатель типа void. Применяется в случаях, когда тип объекта заранее не определен.
3. Указатель на функцию. Позволяет обращаться с функциями, как с переменными.

7.2. Операции над указателями

7.2.1. Унарные операции

Над указателями можно провести две унарные операции:

1. «**&**» **(взять адрес)**. Операция позволяет получить адрес переменной.
2. «*****» **(операция разадресации)**. Позволяет получить доступ к величине, расположенной по указанному адресу.

7.2.2. Арифметические операции и операции сравнения

При выполнении арифметических операций с указателями автоматически учитывается размер данных, на которые они указывают.

Инкремент и декремент. Перемещает указатель к следующему или предыдущему элементу массива.

Например:

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

```
cout << *um << endl;   // Выводит: 1
   um++;
cout << *um << endl;   // Выводит: 2
```

Сложение. Перемещение указателя на число байт, равное произведению размера типа данного, на которое ссылается указатель, на величину добавляемой или вычитаемой константы. Например:

```
 int *um, a[5] = {1,2,3,4,5};
um = a;
cout << *um << endl;   // Выводит: 1
   um += 3;
cout << *um << endl;   // Выводит: 4
```

Вычитание. Разность двух указателей равна числу объектов соответствующего типа, размещенных в данном диапазоне адресов. Например:

```
int *um, a[5] = {1,2,3,4,5};
 um = &a[0];
 un = &a[4];
   k = un - um;
cout << k << endl;   //  Выводит: 4
```

Операции сравнения. Сравнивают адреса объектов.

7.3. Инициализация указателей

Инициализация пустым значением. Например:

```
int *a = NULL;
int *b = 0;
```

Присваивание указателю **адреса уже существующего объекта**. Например:

```
int k = 23;
int *uk = &k;   // или   int *uk(&k);
int *us = uk;
```

Присваивание указателю **адреса выделенного участка динамической памяти**:

```
int *s = new int;
int *k = (int *) malloc(sizeof(int));
```

В примере использована операция sizeof, которая определяет размер указанного параметра в байтах.

7.4. Работа с динамической памятью

Динамическая память (*heap*) – специальная область памяти, в которой во время выполнения программы можно выделять и освобождать место в соответствии с текущими потребностями. Доступ к выделяемым участкам памяти осу-

ществляется через указатели. Для работы с динамической памятью в языке Си (библиотека *malloc.lib*) определены следующие функции:

void *malloc(size) – выделяет область памяти размером *size* байт. Возвращает адрес выделенного блока памяти. Если недостаточно свободного места для выделения заданного блока памяти, то возвращает NULL.

void *calloc(n, size) – выделяет область памяти размером *n* блоков по *size* байт. Возвращает адрес выделенного блока памяти. Если недостаточно свободного места для выделения заданного блока памяти, то возвращает NULL. Вся выделенная память заполняется нулями.

void *realloc(*u) – изменяет размер ранее выделенной памяти, связанной с указателем *u* на новое число байт. Если память под указатель не выделялась, то функция ведет себя как *malloc*. Если недостаточно свободного места для выделения заданного блока памяти, то функция возвращает значение NULL.

void free(*u) – освобождает участок памяти, связанный с указателем *u*.

В языке C++ для выделения и освобождения памяти определены операции new и delete. Имеются две формы операций:

тип *указатель = new *тип* (*значение*) – выделение участка памяти в соответствии с указанным типом и занесение туда указанного значения.

delete указатель – освобождение выделенной памяти.

тип *указатель = new *тип*[n] – выделение участка памяти размером *n* блоков указанного типа.

delete [ ]*указатель* – освобождение выделенной памяти.

Операция delete не уничтожает значения, связанные с указателем, а разрешает компилятору использовать данный участок памяти.

7.5. Создание одномерного динамического массива

Для создания одномерного динамического массива необходимо знать тип элементов массива и их количество. Например, для создания одномерного динамического массива, состоящего из *n* действительных чисел, можно использовать следующие функции:

```
umas1 = static_cast <double*> (malloc(n*sizeof(double)));
```

(освобождение памяти – free(umas1))

или

```
umas1 = static_cast <double*> (calloc (n,sizeof(double)));
```

(освобождение памяти – free(umas1))

или

umas1 = **new double**(n***sizeof**(**double**));

(освобождение памяти – **delete** umas1)

или

umas1 = **new double**[n];

(освобождение памяти – **delete** []umas1)

7.6. Создание двумерного динамического массива

Двумерный динамический массив рассматривается компилятором как массив указателей на одномерные массивы (рис. 7.1).

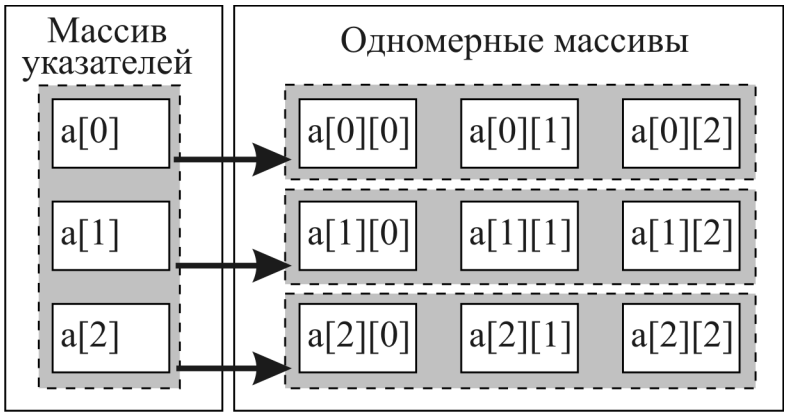


Рис. 7.1

Вначале выделяется память под одномерный массив указателей, затем каждый указатель получает адрес созданного одномерного динамического массива. Освобождение памяти осуществляется в обратном порядке.

```
double  **umas2;              // Объявление указателя на массив
umas2 = new double*[n];       // Выделение памяти
                              // под массив указателей
 for(i=0; i<n; i++)           // Выделение памяти
   umas2[i] = new double[m];// под одномерные массивы

            …       // Работа с массивом

  for(i=0; i<n; i++)       // Освобождение памяти, выделенной
   delete []umas2[i];      // под одномерные массивы
 delete []umas2;           // Освобождение памяти, выделенной
                           // под массив указателей
  umas2 = NULL;            // Очистка указателя
```

8. Функции пользователя

8.1. Понятие функции

Функция – последовательность операторов, оформленная таким образом, что ее можно вызвать по имени из любого места программы.

Функция описывается следующим образом:

тип_возвращаемого_значения имя_функции (*список_параметров*)
{
Тело_функции
}

Первая строка данного описания называется *заголовком функции*. Тип возвращаемого значения может быть любым, кроме массива или функции. Если функция не возвращает значение, то указывается тип void. В C++ по умолчанию тип возвращаемого результата − **int**.

Список параметров представляет собой набор конструкций следующей формы:

тип_параметра имя_параметра

Например:

```
int sum(int a, double b, char c)
```

Если функция не получает никаких данных, то скобки остаются пустыми:

```
int fun()
```

Широко используются *прототипы* функций (их предварительное объявление). Прототип аналогичен заголовку функции, за исключением того, что имена формальных параметров не указываются (остаются только типы), и в конце ставится точка с запятой:

```
int sum(int, double, char);
```

Широкое использование прототипов вызвано следующим:

− имеющие прототипы функции могут быть вызваны из других модулей;

− использование прототипов позволяет размещать функции в произвольном порядке (а не до первого их использования);

− размещение прототипов в одном месте делает программу более читабельной.

Правила оформления тела функции такие же, как и для любого другого участка программы. Все объявления носят локальный характер, т. е. объявленные переменные доступны только внутри функции.

Не допускается вложение функций друг в друга.

Выход из функции происходит при достижении закрывающей функцию скобки или после выполнения оператора return.

8.2. Передача параметров

При работе должно соблюдаться следующее правило: при объявлении и вызове функции параметры должны соответствовать по количеству, порядку следования и типам. Существует три основных способа передачи параметров: передача по значению, по ссылке и по указателю.

8.2.1. Передача параметров по значению

В функции создаются временные переменные, в которые передаются значения из вызывающей функции. Например:

```
int fun1(double, int, char);   // Прототип функции
            …
 int fun1(double a, int b, char c)  // Заголовок функции
    {
         Тело_функции
    }
            …
int s = fun1(d, 8, chr);  // Вызов функции
```

В момент обращения к функции в памяти создаются временные переменные с именами a, b, c. В созданные переменные копируются значения: d, 5, chr. После этого связь между передаваемыми и временными переменными разрывается.

8.2.2. Передача параметров по ссылке

В функцию передаются адреса переменных из основной программы. Ссылочный параметр – псевдоним соответствующего аргумента. Для получения адреса используется операция «взять адрес». Например:

```
void fun2(double &, int &); // Прототип функции
            …
void fun2(double &a, int& b)  // Заголовок функции
 {
   Тело_функции
    }
            …
  fun2(d, r);  // Вызов функции
```

При таком вызове передается не переменная, а ее адрес, полученный с использованием операции «взять адрес». Следовательно, при обращении к временной переменной в действительности происходит обращение к исходной переменной в вызывающей функции.

8.2.3. Передача параметров по указателю

При вызове функции в качестве аргумента передается не сама переменная, а ее адрес. Например:

```
void fun3(double *, int *);  // Прототип функции
            …
void fun3(double *a, int* b)  // Заголовок функции
{
    Тело_функции
}
            …
    fun3(&f,&k);  // Вызов функции
```

	При работе с формальными параметрами внутри функции необходимо использовать операцию разадресации, например: `s = (*a + *b) / 2;`

8.2.4. Параметры со значениями по умолчанию

При объявлении функции для некоторых аргументов можно задавать значение по умолчанию, которое передается в функцию в случае, если при вызове соответствующий параметр не задан. Так как компилятор присваивает имеющиеся значения последовательно слева направо, то аргументы, имеющие заданное по умолчанию значение, должны располагаться правее аргументов, не имеющих такого значения. Например:

```
void fun4(double,  int b = 3,  double h = 0.1);   // Прототип функции
            …
void fun4(double a,  int b,  double h)  // Заголовок функции
      {
         Тело_функции
       }
```

При вызове

```
fun4(d);    // Вызов функции
```

переменной *a* передается указанное при вызове значение *d*, а остальным, т. к. они отсутствуют в списке, присваиваются значения по умолчанию ($b = 3$, $h = 0.1$).

При вызове

```
un4(d,r)
```

переменной *a* будет передано значение *d*, переменной *b* – значение *r*, а переменной *h* значение по умолчанию $h = 0.1$.

При вызове

```
un4(d,r,f)
```

всем переменным будут переданы значения, указанные при вызове. Значения по умолчанию не используются.

Пропуск аргументов при вызове функции запрещен. Параметры по умолчанию должны быть указаны при первом упоминании функции.

8.2.5. Передача массивов в функции

При передаче массива в функцию соответствующий аргумент должен содержать тип, имя массива и квадратные скобки. При вызове указывается только имя массива. Например:

```
void funm1(int []);  // Прототип функции
            …
void funm1(int b[])  // Заголовок функции
{
Тело_функции
}
            …
funm1(a);  // Вызов функции
```

C++ передает имя массива по ссылке, т. е. при изменении элементов массива в функции меняются элементы соответствующего массива в вызывающей процедуре.

Как правило, в функцию предают не только сам массив, но и его размер.

При передаче многомерного массива скобки для первой размерности остаются пустыми, а для других размерностей должен константой указываться размер. Например, передача двумерного массива размером 3x3 организуется следующим образом:

```
void funm2(int [][3]); // Прототип функции
            …
void funm2(int b[][3])  // Заголовок функции
{
Тело_функции
}
            …
      funm2(a);  // Вызов функции
```

8.2.6. Передача переменного числа параметров

Формат объявления функции с переменным числом параметров:

тип_возвращаемого_значения имя_функции (*список_параметров*, **…**)

Список параметров содержит хотя бы один обязательный параметр. Многоточие указывает на возможность добавления любого числа параметров.

Для работы с параметрами определен тип списка va_list и три макроса:

void va_start(va_list *указатель*, *имя_послед. _обязат._аргумента*)

начинает работу со списком. Устанавливает указатель на первый необязательный аргумент.

void va_arg(va_list *указатель*, *тип_аргумента*)

возвращает значение очередного аргумента из списка. Каждый запуск макроса переводит указатель на следующий аргумент. Достижение последнего аргумента списка не контролируется.

void va_end(va_list *указатель*)

завершает работу со списком и освобождает память.

Пример: Посчитать сумму введенных аргументов. Условие прекращения ввода − значение аргумента равно –1.

```
int fun5(int, ...);  // Прототип функции
                ...
int fun5(int a, ... )  // Заголовок функции
{
       int ar, s;
 va_list argm;
   s = a;
 va_start(argm, a);
    ar = va_arg( argm, int);
   while (ar != -1)
     {
     s += ar;
      ar = va_arg( argm, int);
     }
 va_end( argm);
 return s;
}
```

Вызов функции: `int r = fun5(1, 2, 3, 4, 5, 6, -1);`

8.3. Встраиваемые функции

При использовании небольших функций часто затраты ресурсов компьютера на организацию работы с ними превышают затраты, возникающие при непосредственном внесении тела функции в код программы. В этих случаях используются встраиваемые (inline) функции:

```
inline double fun5(int a, double b)
{
Тело_функции
}
```

При компиляции в местах вызова функции будет помещаться тело функции.

8.4. Перегрузка функций

Под перегрузкой функций понимается использование различных функций с одинаковым именем. Перегруженные функции различаются компилятором по типам и числу параметров. Например, если необходимо вычислить площадь круга или прямоугольника, можно написать следующие функции:

```
double Ploch(double a, double b)  {  return a*b;  }
double Ploch(double r)                {  return 3.14*pow(r,2);  }
```

8.5. Указатель на функцию

Имя функции является константным указателем на начало функции в оперативной памяти. Разрешается использовать указатели на функции в программе.

Например, имеется функция

```
double y(double x, int n)
{
Тело_функции
}
```

Указатель на такую функцию имеет вид:

```
double (*fun)(double, int);
```

Если присвоить указателю fun адрес функции y:

```
fun = y;
```

то функцию можно вызывать

```
x = fun(t, m);
```

Пример. Вывести на экран таблицу значений функции $y(x) = \sin x$ и ее разложения в ряд $s(x) = x - \frac{x^3}{3!} + ... + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$ с точностью $\varepsilon = 0.001$. Вывести число итераций, необходимое для достижения заданной точности.

```
#include <iostream.h>
#include <math.h>
#include <iomanip.h>
   typedef double (*uf)(double, double, int &);
void tabl(double, double, double, double, uf);
double y(double, double, int &);
double s(double, double, int &);
int main()
{
cout << setw(8) << "x" << setw(15) << "y(x)" << setw(10) << "k" << endl;
tabl(0.1,0.8,0.1,0.001,y);
```

```
cout << endl;
cout << setw(8) << "x" << setw(15) << "s(x)" << setw(10) << "k" << endl;
tabl(0.1,0.8,0.1,0.001,s);
return 0;
}
void tabl(double a, double b, double h, double eps, uf fun)
{
        int k = 0;
        double sum;
         for (double x=a; x<b+h/2; x+=h)
         {
                  sum =f un(x,eps,k);
  cout << setw(8) << x << setw(15) << sum << setw(10) << k << endl;
         }
}

double y(double x, double eps, int &k)
{
return sin(x);
}

double s(double x, double eps, int &k)
{
  double a,c,sum;
        sum=a=c=x;
     k = 1;
        while (fabs(c)>eps)
        {
                 c = pow(x,2)/(2*k*(2*k+1));
        a *= -c;
        sum += a;
        k++;
        }
return sum;
}
```

9. Использование строковых переменных

В языке С++ имеется два способа работы со строковыми данными: использование массива символов типа char и использование класса *string*. В данном пособии рассматривается первый способ организации работы со строками.

9.1. Объявление строк

Объявление строки аналогично объявлению массива:

```
char имя строки [размер]
```

В отличие от массива строка должна заканчиваться нулевым символом ‘\0’ – (нуль-терминатором). Длина строки равна количеству символов плюс нулевой символ. При наборе нулевой символ помещается в конец сроки автоматически. Например, в строке

```
char st1[10] = ”123456789”;
```

символы располагаются следующим образом:

‘1’	‘2’	‘3’	‘4’	‘5’	‘6’	‘7’	‘8’	‘9’	‘\0’

Если размер строки не объявлен явно, то он будет установлен автоматически и будет равен количеству введенных символов +1.

```
char st2[ ] = ”1234”;
```

Разрешено использовать указатели на строку:

```
char *st3 = st2;
```

Доступ к отдельным символам строки осуществляется по их индексам. Например: st[2] = ’e’;

	В языке Си одиночные кавычки используются для обозначения символов, а двойные – для обозначения строк.

Массив строк объявляется как двумерный массив символов:

```
char имя[количество строк][количество символов в строке];
```

Например,

```
char str[10][5].
```

Обращение к нулевой строке массива строк: str[0].

9.2. Функции для работы со строками

Для ввода/вывода строк и символов используются функции библиотеки *stdio.lib*, для работы с нуль-терминированными строками – функции библиоте-

ки *string.h*, для преобразования типов – функции библиотеки *stdlib.lib*, а для распознавания символов – функции библиотеки *ctype.lib*.

Наиболее часто применяются следующие функции:

int puts(*строка*) – выводит на экран указанную *строку*. Переводит указатель на следующую сроку.

char *gets(*строка*) – помещает в *строку* данные введенные с клавиатуры.

char *strcpy(*строка_1*, *строка_2*) – копирует содержимое *строки_1* в *строку_2*.

Например:

```
char st1[40], st2[40] = "xyz";
  strcpy(st1, st2);
```

Результат: *st*1 = "*xyz*".

Имеется также функция strncpy (*строка_1*, *строка_2*, *количество_символов*), которая копирует в *строку_2* заданное *количество_символов* из *строки_1*.

char *strcat(*строка_1*, *строка_2*) − добавляет справа к *строке_1* содержимое *строки_2*.

```
char st1[40] = "ABCD", st2[40] = "xyz";
  strcat(st1, st2);
```

Результат: st1 = "ABCDxyz".

Имеется также функция strncat(*строка_1*, *строка_2*, *количество_символов*), которая добавляет справа к *строке_1* заданное *количество_символов* из *строки_2*.

int strcmp(*строка_1*, *строка_2*) – сравнивает содержимое *строки_1* и *строки_2*. Если *строка_1* < *строка_2*, то результат –1, если *строка_1* = *строка_2* − результат равен нулю, если *строка_1* > *строка_2* – результат равен 1.

```
char st1[40]="ABCD", st2[40]="xyz";
k=strcmp(st1, st2)
```

Результат: *k* = –1.

Имеются также следующие функции: stricmp(*строка_1*, *строка_2*), которая делает сравнение строк без учета регистра; strncmp(*строка_1*, *строка_2*, *количество_символов*), которая сравнивает в строках только заданное *количество_символов*; strnicmp(*строка_1*, *строка_2*, *количество_символов*), которая сравнивает в строках только заданное *количество_символов* без учета регистра.

char *strchr(*строка*, *символ*) – возвращает указатель на первое появление *символа* в *строке*.

Например, требуется определить позицию первого появления символа *C* в строке *ABCD*.

```
char st1[40] = "ABCD";
char *s = strchr(st1, 'C');
int k = static_cast <int> (s - st1);
```

Результат: $k = 2$.

Функция strrchr(*строка*, *символ*) возвращает указатель на последнее вхождение *символа* в *строку*.

char *strspn(*строка_1*, *строка_2*) – возвращает число позиций с начала *строки_1*, значения которых совпадают с любым из символов *строки_2*.

```
char st1[40] = "ABCD";
char st2[40] = "RAMB";
int k = strspn(st1, st2);
```

Результат: $k = 2$.

Функция strcspn(*строка_1*, *строка_2*) возвращает число позиций с начала *строки_1*, значения которых не совпадают с любым из символов *строки_2*.

char *strpbrk(*строка_1*, *строка_2*) – возвращает указатель на первое вхождение любого из символов *строки_2* в *строку_1*.

```
char st1[40] = "ABCD";
char st2[40] = "XYCB";
char *ch=strpbrk(st1, st2);
```

Результат: **ch*= ‘B’.

char *strstr(*строка_1*, *строка_2*) – возвращает указатель на первое появление *строки_2* в *строке_1*.

Например, требуется определить позицию первого вхождения строки *BC* в строку *ABCD*.

```
char st1[40] = "ABCD";
char *s = strstr(st1, "BC");
k = static_cast <int> (s - st1);
```

Результат: $k = 1$.

char *strtok(*строка_1*, *строка_2*) – возвращает указатель на лексему, находящуюся в *строке_1* (лексемой считается набор символов, отделенный от других лексем символом-разделителем, находящимся в *строке_2*).

При первом вызове функции она возвращает указатель на первый символ в *строке_1*, а после последнего символа первой лексемы устанавливает нулевой символ. При последующих вызовах функции со значением NULL в качестве первого аргумента указатель аналогичным образом переходит к следующим лексемам. После нахождения всех лексем указатель получает значение NULL.

Пример 9.1. Вывести на экран лексемы, разделенные символами ‘-’и ‘:’.

```
char st1[50];
char *wrd;
    gets(st1);
     char st2[ ] = "-:";     //
      wrd = strtok(st1,st2);
      while(wrd != NULL)
      {
              puts(wrd);
              wrd = strtok(NULL,st2);
      }
```

Если *st1="AAAA:BBBB B-C CC:-:DDDD D*", то на экран будет выведено:

```
AAAA
BBBB B
C CC
DDDD  D
```

Если для этой же строки разделитель поставить равным пробелу (char st2[] = " ";), то результат будет следующий:

```
AAAA:BBBB
B-C
CC:-:DDDD
D
```

int strlen(*строка*) – возвращает длину *строки* (нуль-терминатор ‘\0’ не учитывается).

```
char st[40] = "ABCD";
k = strlen(st);
```

Результат: $k = 4$.

char *strrev(*строка*) – изменяет порядок следования символов в *строке* на противоположный.

```
char st[40]="ABCD";
strrev (st);
```

Результат: *st* = "DCBA".

char *strdup(*строка*) – дублирует *строку*. Функция вызывает malloc для выделения памяти под новою строку (необходимо использовать free() для очистки памяти в конце работы).

```
char st1[40] = "ABCD";
  char *st2;
  st2 = strdup(st1);
  …
 free(st2);
```

Результат: *st*2 = "ABCD".

char *strlwr(*строка*) – преобразует все прописные символы (верхний регистр) *строки* в строчные символы (нижний регистр).

```
char st[40] = "aBcD";
strlwr(st);
```

Результат: *st* = “abcd”.

char *strupr(*строка*) – преобразует все строчные символы (нижний регистр) *строки* в прописные символы (верхний регистр).

```
char st[40  ]= "aBcD";
strupr(st);
```

Результат: *st* = “ABCD”.

int atoi(*строка*) – преобразует *строку* в число целого типа ( ).

```
char st1[40] = "354553";
int k = atoi(st1);
```

Результат: *k* = 354553.

double atof(*строка*) – преобразует *строку* в число действительного типа.

```
char st1[40] = "354.553";
double b = atof(st1);
```

Результат: *b* = 354.553.

char *itoa(*число*, *строка*, *основание_системы_счисления*) – преобразует символы десятичного целого *числа* в *строку* в соответствии с заданной системой счисления (от 2 до 36).

```
char st[40];
itoa(4, st, 10);
```

Результат: *st* = 4 в десятичной системе счисления.

```
char st[40];
itoa(4, st, 2);
```

Результат: *st* = 100 в двоичной системе счисления.

char *gcvt(*число*, *количество_десятичных_разрядов*, *строка*) – преобразует *число* действительного типа в *строку*. *Количество_десятичных_разрядов* должно быть не более 18.

```
double a = -254.2965;
char st[40];
gcvt(a, 7, st);   // st = “-254.2965”
gcvt(a, 5, st);   // st = “-254.3”
gcvt(a, 3, st);   // st = “-254”
gcvt(a, 2, st);   // st = “-2.5e+002”
gcvt(a, 1, st);   // st = “-3e+003”
```

Функции распознавания символов:

int isalnum(*символ*) возвращает ненулевое значение (*true*), если *символ* – буква или цифра.

int isalpha(*символ*) возвращает ненулевое значение (*true*), если *символ* – буква;

int isdigit(*символ*) возвращает ненулевое значение (*true*), если *символ* – цифра;

int ispunct(*символ*) возвращает ненулевое значение (*true*), если *символ* – знак пунктуации;

int islower(*символ*) возвращает ненулевое значение (*true*), если *символ* – буква нижнего регистра;

int isupper(*символ*) возвращает ненулевое значение (*true*), если *символ* – буква верхнего регистра;

int isspace(*символ*) возвращает ненулевое значение (*true*), если *символ* – пробел, знак табуляции, возврат каретки, символ перевода строки, вертикальной табуляции, перевода страницы.

9.3. Алгоритмы работы со строками

Пример 9.2. Проверить, присутствует ли слово “*visual”* в заданной строке.

```
char st[30];
char *ch = NULL;
 puts("Vvedite stroku");
 gets(st);
  ch = strstr(st,"visual");
  if (ch != NULL) puts("Prisutstvuet");
        else puts("Ne prisutstvuet");
```

Пример 9.3. В строке *st* удалить все символы ‘*z*’.

```
char st[100];
gets(st);
for (int i=0; i<strlen(st); i++)
   if (st[i] == 'z')
      {
for (int j=i; j<strlen(st); j++)  st[j]=st[j+1];
            i--;
      }
puts(st);
```

Пример 9.4. Выделить и вывести на печать все слова произвольной строки. Слова отделяются друг от друга одним или несколькими пробелами.

```
char st[100], sl[100];
int k=0, i;

gets(st);
strcat(st," ");
 int n = strlen(st);
     if (n < 2) return 1;
 sl[0] = '\0';

 for (i=0; i<n; i++)
      if (st[i] != ' ')
      {
            sl[k] = st[i];
         sl[k+1] = '\0';
         k++;
      }
      else
      {
            if (strlen(sl)>0) puts(sl);
      sl[0] = '\0';
            k = 0;
      }
```

Пример 9.5. Определить, является ли строка палиндромом, т. е. читается ли она слева направо так же, как и справа налево (например, «А роза упала на лапу Азора»).

```
char st[80] = "A roza   upala na  lapu Azora";
 int i = 0, j  = strlen(st)-1;
   strlwr(st);
  bool bl = true;
while (i <= j) {
   while (st[i] == ' ') i++;
   while (st[j] == ' ') j--;

     if (st[i] != st[j])
       {
          bl = false;
          break;
        }
    i++;   j--;
 }

if (bl)  cout << "Palindrom" << endl;
   else cout << "Ne palindrom" << endl;
```

Пример 9.6. В заданном предложении найти самое короткое и самое длинное слово.

```
char st1[100];
   char *wrd, *cmin, *cmax;
  gets(st1);
  char st2[] = " ";
   wrd = strtok(st1,st2);
  cmin=cmax=wrd;
   while(wrd != NULL)
   {
            if (strlen(wrd) > strlen(cmax)) cmax = wrd;
                 else
            if (strlen(wrd) < strlen(cmin)) cmin = wrd;
     wrd = strtok(NULL,st2);
   }
```

10. Типы данных, определяемых пользователем

10.1. Объявление и использование структур

Структура – составной тип данных, в котором под одним именем объединены данные различных типов. Отдельные данные структуры называются *полями*.

Объявление структуры:

```
struct им_структуры
{
        тип_элемента_1   имя_элемента_1;
        тип_элемента_2   имя_элемента_2;
            ...
        тип_элемента_n   имя_элемента_n;
};
```

Например:

```
struct struc1
{
    int m1;
    double m2, m3;
};
```

Поля структуры могут быть любого типа, в том числе массивами и структурами.

После фигурной скобки допустимо указывать переменные соответствующего структурного типа:

```
struct struc1
{
    int m1;
    double m2, m3;
} a, b ,c;
```

Объявление переменной структурного типа:

```
struc1 x;
```

К отдельным частям структуры можно *обращаться* через составное имя. Формат обращения:

имя_структуры.имя_поля

или

указатель_на_структуру **->** *имя_поля*

Например, если структура объявлена следующим образом:

```
struct struc1
{
    int m1;
    double m2, m3;
} x, *y;
```

то обратиться к полю m1 можно (после выделения памяти для *y*):

```
x.m1 = 35;
```

или

```
(&x)->m1 = 35;
```

или

```
y->m1 = 35;
```

или

```
(*y).m1 = 35;
```

Правила работы с полями структуры идентичны работе с переменными соответствующих типов. Инициализировать переменные-структуры можно путем помещения за объявлением списка начальных значений.

```
struct struc1
{
    int m1;
    double m2, m3;
} a = {5, 2.6, 34.2};
```

В качестве полей могут быть использованы другие структуры.

```
struct struc1
{
    int m1;
    double m2, m3;
      struct
      {
            int mm1;
      } m4;
} s;
```

Обращение к полю mm1 в этом случае будет следующим:

```
s.m4.mm1 = 3;
```

Если имя структуры не указывается, то такое определение называется *анонимным*.

Допустимо использовать операцию присваивания для структур одного типа. Например:

```
struc1 x, y;
    …
  x = y;
```

В этом случае все значения полей структуры y копируются в соответствующие поля структуры x.

Из структур, как правило, организуют массивы:

```
struct struc1
{     int m1;
      double m2, m3;
};
…
struc1 ms[100];  // Объявление массива структур
…
ms[99].m1 = 56;  // Обращение к полю массива структур
```

Пример. Имеется список жильцов многоквартирного дома. Каждый элемент списка содержит следующую информацию: фамилия, номер квартиры, количество комнат в квартире. Вывести в алфавитном порядке фамилии жильцов, проживающих в двухкомнатных квартирах. Память для хранения списка выделять динамически.

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

int main ()
{
  struct tzhilec
  {
       char fio[50];
       int nomer;
       int nkomnat;
  }  *spisok;

 int n, i, j;
  cout << "Vvedite kolichestvo zhilcov: " << endl;
  cin >> n;
     spisok = new tzhilec[n];
  for (i=0; i<n; i++)
  {
```

```
    cout << "Vvedite familiju: ";
            cin >> spisok[i].fio;
    cout << "Vvedite nomer kvartiry: ";
             cin >> spisok[i].nomer;
    cout << "Vvedite kolichestvo komnat: ";
            cin >> spisok[i].nkomnat;
    cout << endl;
  }

tzhilec tmp;
   for (i=0; i< n-1; i++)
       for (j=i+1; j<n; j++)
       if (spisok[i].nkomnat == 2 && spisok[j].nkomnat == 2
                    && strcmp(spisok[i].fio, spisok[j].fio )== 1)
                    {
                            tmp   = spisok[i];
                            spisok[i] = spisok[j];
                            spisok[j] =tmp;
                    }
for (i=0; i<n; i++)
       if (spisok[i].nkomnat == 2)
    cout << spisok[i].fio << "  nomer kvartiry - "
                                          << spisok[i].nomer << endl;
delete [ ]spisok;
  return 0;
}
```

10.2. Объявление и использование объединений

Объединение (*union*) – размещение под одним именем некоторой совокупности данных таким образом, чтобы размер выделяемой памяти был достаточен для размещения данного, имеющего наибольший размер. Такие структуры используются в случаях, когда отдельные поля существуют в различные моменты времени.

Объявление объединения:

```
union имя_объединения
{
набор_полей
 };
```

Например:

```
union per{
      int a;
         double b;
         char  c;
   } un;
    un.a = 567;
   cout << un.a << endl; // Значение mun.a равно 567
     un.b = 8.2;
  cout << un.a << endl;  // Ошибка: переменная un.a недоступна
  cout << un.b << endl;  // Значение mun.b равно 8.2
```

10.3. Объявление и использование перечислений

Перечисление (*enum*) задает множество значений для заданной пользователем переменной.

Объявление перечисления:

```
enum имя {набор_значений};
```

Например:

```
enum otc {NEUD, UD, HOR, OTL};
```

Каждому значению в перечислении присваивается номер. По умолчанию первое значение имеет номер 0, второе – 1, и т. д. Можно устанавливать нумерацию, отличную от заданной, по умолчанию:

```
enum otc {NEUD=2, UD, HOR, OTL} a, b = UD;
```

Объявление может совмещаться с инициализацией переменных:

```
enum otc {NEUD=2, UD, HOR, OTL} a, b = UD;
```

Перечисления могут неявно преобразовываться в целочисленные типы, но не наоборот:

```
otc a = NEUD;
int k = a;   //  Допустимая операция
    a = 2;   //  Недопустимая операция
```

11. Файлы

11.1. Понятие файла

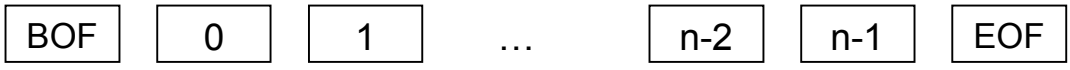
Файл − поименованная совокупность данных, расположенных на внешнем носителе. В начале работы для получения доступа к данным файл необходимо открыть. При открытии файла указатель текущей позиции помещается в начало файла. После выполнения любой операции над данными указатель сдвигается на одну позицию вперед. В конце работы файл закрывается, т. е. доступ к данным, размещенным в файле, будет запрещен. Информация о файле хранится в управляющей структуре, имеющей тип FILE.

Различают два вида файлов: текстовые и двоичные.

Текстовые файлы хранят информацию в виде последовательности символов. Вывод осуществляется аналогично выводу на экран. Текстовые файлы могут быть отредактированы в любом текстовом редакторе.

Бинарные (или двоичные) файлы предназначены для хранения последовательности байтов. Структура такого файла определяется программно.

Файлы, размещаемые на носителях информации, имеют следующую структуру:



В начале файла записана информация о файле BOF (*Begin of File*), его имя, тип, длина и т. д., в конце файла помещается признак конца файла EOF (*End of File*). Если файл пуст, то BOF и EOF совмещены.

При работе с файлами используются следующие макросы:

– NULL − определяет пустой указатель;
– EOF − значение, возвращаемое при попытке чтения после конца файла;
– FOPEN_MAX − возвращает максимальное число одновременно открытых файлов.

11.2. Функции для работы с файлами

Функции для работы с файлами размещены в библиотеках *stdio.lib* и *io.lib*. При работе с файлами используются указатели типа FILE. Формат объявления указателя на файл следующий:

```
FILE *указатель_на_файл;
```

Например:

```
FILE *fl1, *fl2;
```

Указатель содержит адрес структуры, включающей в себя различные сведения о файле, например, его имя, статус и указатель на начало файла.

Функция

FILE ***fopen**(**const char** **имя_файла*,

const char **режим_открытия*);

открывает файл и связывает его с потоком. Возвращает указатель на открытый файл.

Имя_файла − указатель на строку символов, в которой хранится имя файла и путь к нему. Например: “d:\\work\\lab2.dat”.

Режим_открытия − указатель на строку символов, в которой указывается режим открытия файла. Допустимые режимы приведены в табл. 11.1.

Таблица 11.1

Режим открытия	Действие
r (или rt)	Открывает текстовый файл для чтения. В случае отсутствия файла с указанным именем возникает ошибка
w (или wt)	Создает текстовый файл для записи. Если файл с указанным именем уже существует, то прежняя информация уничтожается
a (или at)	Открывает текстовый файл для записи. Указатель устанавливается в конец файла
rb	Открывает двоичный файл для чтения. В случае отсутствия файла с указанным именем возникает ошибка
wb	Создает двоичный файл для записи. Если файл с указанным именем уже существует, то прежняя информация уничтожается
ab	Открывает двоичный файл для записи. Указатель устанавливается в конец файла
r+ (или rt+)	Открывает текстовый файл для чтения и записи данных
w+ (или wt+)	Создает текстовый файл для чтения и записи данных
a+ (или wt+)	Открывает текстовый файл для чтения и записи данных. Указатель устанавливается в конец файла. Если файл с указанным именем отсутствует, то он будет создан
rb+ (или r+b)	Открывает бинарный файл для чтения и записи данных
wb+ (или w+b)	Создает бинарный файл для чтения и записи данных
ab+ (или a+b)	Открывает бинарный файл для чтения и записи данных. Указатель устанавливается в конец файла. Если файл с указанным именем отсутствует, то он будет создан

По умолчанию файл открывается в текстовом режиме.

Если при открытии файла произошла ошибка, функция fopen возвращает значение NULL.

Для создания файла можно записать:

```
FILE *fl;
fl = fopen("lab2.dat","w”);
```

Более грамотно будет:

```
FILE *fl;
    if ((fl = fopen("lab2.dat","w")) == NULL)
       {   cout << "Oshibka pri sozdanii"<<endl;    return 1;      }
```

Такой алгоритм позволяет обнаружить любую ошибку, возникающую при создании файла.

Для исключения ошибки, возникающей при открытии несуществующего файла, можно использовать конструкцию

```
FILE *fl;
    if ((fl = fopen("lab2.dat","r")) == NULL)
       fl = fopen("lab2.dat","w");
```

При записи обмен происходит не непосредственно с файлом, а с некоторым буфером. Информация из буфера переписывается в файл только при переполнении буфера или при закрытии файла.

Для закрытия файла используется функция

int fclose(FILE **указатель_на _файл*);

Функция закрывает поток, который был открыт с помощью вызова fopen() и записывает в файл все данные, которые еще оставались в дисковом буфере. Доступ к файлу после выполнения функции будет запрещен. Если файл был закрыт без ошибок, то функция возвращает нуль, иначе − EOF.

Для закрытия всех открытых файлов используется функция

int fcloseall(**void**);

Функция

int putc(**int** *символ*, FILE **указатель_на _файл*);

записывает ***символ*** в текущую позицию указанного открытого файла. Если функция выполнилась успешно, то она возвращает записанный символ, иначе – EOF.

Функция

int getc(FILE **указатель_на_файл*);

читает один символ из текущей позиции указанного открытого файла. После чтения указатель сдвигается на одну позицию вперед. Если достигнут конец файла, то функция возвращает значение EOF.

Функция

int feof(FILE **указатель_на_файл*);

возвращает отличное от нуля значение (*true*), если достигнут конец файла, и нуль (*false*), если конец файла не достигнут. Функция работает с файлами всех типов.

Функция

int fputs(const **char** * *строка*, FILE **указатель_на_файл*);

записывает строку символов в текущую позицию указанного открытого файла. В случае ошибки эта функция возвращает EOF. Нулевой символ в файл не записывается.

Функция

char ***fgets**(**char** **строка*, **int** *длина*, FILE **указатель_на_файл*);

читает строку символов из текущей позиции указанного открытого файла до тех пор, пока не будет прочитан символ перехода на новую строку или количество прочитанных символов не станет равным *длина–1*. В случае ошибки функция возвращает NULL.

Функция

int ***fprintf**(FILE **указатель_на_файл*,
const char **строка форматирования* [, *аргументы*]);

записывает форматированные данные в файл. Строка форматирования аналогична строке форматирования функции printf.

Функция

int ***fscanf**(FILE **указатель_на_файл*,
const char **строка_форматирования* [, *аргументы*]);

читает форматированные данные из файла. Строка форматирования аналогична строке форматирования функции scanf.

Функция

void rewind(FILE **указатель_на_файл*);

устанавливает указатель текущей позиции в начало файла.

Функция

int ferror(FILE **указатель_на_файл*);

определяет, произошла ли ошибка во время работы с файлом. Она возвращает ненулевое значение, если при последней операции с файлом произошла ошибка, иначе − возвращает 0 (*false)*.

Функция

size_t **fwrite**(**const void** **записываемое_данное*,
size_t *размер_элемента*, size_t *число_элементов*,
FILE **указатель_на_файл*);

записывает в файл заданное число данных заданного размера. Размер данных задается в байтах. Тип size_t определяется как целое без знака. Функция возвращает число записанных элементов.

Функция

size_t **fread**(**void** **переменная*,*

size_t *размер_элемента*, size_t *число_элементов*,

FILE **указатель_на_файл*);

считывает в указанную переменную заданное число данных указанного размера. Размер данных задается в байтах. Функция возвращает число прочитанных элементов.

Функция

int fileno(FILE **указатель_на_файл*);

возвращает значение дескриптора указанного файла (дескриптор – логический номер файла для заданного потока).

Функция

long filelength(**int** *дескриптор*);

возвращает длину файла с соответствующим дескриптором в байтах.

Функция

int chsize(**int** *дескриптор*, **long** *размер*);

устанавливает новый размер файла с соответствующим дескриптором. Если размер файла увеличивается, то в конец добавляются нулевые символы, если размер файла уменьшается, то все лишние данные удаляются.

Функция

long ftell(FILE **указатель_на_файл*);

возвращает значение указателя на текущую позицию файла.

Функция

int fseek(FILE * указатель_на _файл,

long **int** *число_байт*, **int** *точка_отсчета*);

устанавливает указатель в заданную позицию. Заданное количество байт отсчитывается от начала отсчета, которое задается следующими макросами: начало файла – SEEK_SET, текущая позиция – SEEK_CUR, конец файла – SEEK_END. При успешном завершении работы функция возвращает нуль, а в случае ошибки – ненулевое значение.

Пример. Написать программу для работы с бинарным файлом, содержащим список жильцов многоквартирного дома. Каждый элемент списка содержит следующую информацию: фамилия, номер квартиры, количество комнат в квартире. Найти и вывести на экран текстовый файл, содержащий информацию о жильцах, проживающих в трехкомнатных квартирах.

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>
    FILE *fl;
    struct tzhilec
   {
     char fio[50];
         int nomer;
         int nkomnat;
   }  *spisok, zhilec;
      char fname[20] = "";
      int n=0;
void fadd();                    // Ввести список
void frd();                     // Прочитать список
void rezc();                    // Вывести результат на экран
void rezf();                    // Вывести результат в файл
int menu();                             // Меню
char * fnam();                  // Ввод имени файла

int main()
{
   while (true)
   {
 switch (menu())
                 {
  case 1: fadd();    break;
  case 2: frd();   break;
  case 3: rezc(); break;
  case 4: rezf();    break;
  case 5: return 0;
    default : "Viberite pravilno!";
                 }
        puts("Press any key to continue");
        getch();        system("cls");
```

```
        }
}

int menu()
{
 cout << "Vybirite:" << endl;
 cout << "1. Vyvesti dannye v fail" << endl;
 cout << "2. Prochitat' dannye iz faila" << endl;
 cout << "3. Vyvesti rezul'tat na ekran" << endl;
 cout << "4. Vyvesti rezul'tat v fajl" << endl;
 cout << "5. Exit" << endl;
 int i;     cin >> i;
 return i;
}

char * nnf()
{
  if (strlen(fname)) return fname;
 cout << "Vvedite file name" << endl;    cin >> fname;
 return fname;
}

void fadd()
{
   if ((fl = fopen(nnf(),"ab")) == NULL)
      if ((fl = fopen(nnf(),"wb")) == NULL)
          {
       cout << "Oshibka pri sozdanii"<<endl;       return;
          }
char ch;
  do
  {
cout << "Vvedite familiju: ";                    cin >> zhilec.fio;
cout << "Vvedite nomer kvartiry: ";              cin >> zhilec.nomer;
cout << "Vvedite kolichestvo komnat: ";          cin >> zhilec.nkomnat;
      fwrite( &zhilec, sizeof(tzhilec), 1, fl );
    cout << endl<< "    Budete vvodit eshhe? (y/n) " ;
```

```
        cin >> ch;  cout << endl;
  }
  while (ch=='y');
fclose(fl);
}

void frd()
{
if ((fl = fopen(nnf(),"rb")) == NULL)
        {
      cout << "Oshibka pri otkritii"<<endl;     return;
        }
 n=filelength(fileno(fl))/sizeof(tzhilec);
   spisok = new tzhilec[n];
     fread( spisok, sizeof(tzhilec), n, fl );
cout << endl;
for (int i=0; i < n; i++)
  cout << spisok[i].fio << "  Nomer kvartiry - " << spisok[i].nomer <<
                        "  Chislo komnat- " << spisok[i].nkomnat << endl;
cout << endl;
delete []spisok;
fclose(fl);
}

void rezc()
{
if ((fl = fopen(nnf(),"rb"))==NULL)
        {
      cout << "Oshibka pri otkritii"<<endl;     return;
        }
 n=filelength(fileno(fl))/sizeof(tzhilec);
for (int i=0; i < n; i++)
{
 fread( &zhilec, sizeof(tzhilec), 1, fl );
  if (zhilec.nkomnat == 3)
        cout << zhilec.fio << "  Nomer kvartiry - "
                                        << zhilec.nomer << endl;
```

```
}
fclose(fl);
}

void rezf()
{
char fnamet[20];
FILE *ft;
 cout << "Vvedite textfile name" << endl;
 cin >> fnamet;

 if ((ft = fopen(fnamet,"w")) == NULL)
        {
      cout << "Oshibka pri sozdanii textfile"<<endl;     return;
        }
if ((fl = fopen(nnf(),"rb")) == NULL)
        {
      cout << "Oshibka pri otkritii"<<endl;      return;
        }
    n = filelength(fileno(fl))/sizeof(tzhilec);
for (int i=0; i<n; i++)
{
 fread( &zhilec, sizeof(tzhilec), 1, fl );
  if (zhilec.nkomnat == 3)
    fprintf(ft,"%s, nomer kvartiry - %d\n",zhilec.fio,zhilec.nomer);
        }
fclose(fl);
fclose(ft);
}
```

12. Область видимости и классы памяти

Область видимости определяет, в каких частях программы возможно использование данной переменной, а *класс памяти* – время, в течение которого переменная существует в памяти компьютера. Период времени между созданием и уничтожением переменной называется *временем жизни* переменной.

В языке C++ определены 4 класса памяти:

Автоматический, локальный (***auto***) класс памяти. Область видимости локальных переменных ограничена функцией или блоком, в котором она объявлена. Время жизни локальной переменной − промежуток времени между ее объявлением и завершением работы функции или блока, в которых она объявлена. Ограничение времени жизни переменной позволяет экономить оперативную память. Этот класс памяти используется по умолчанию.

Статический, локальный (***static***) класс памяти. Переменная имеет такую же область видимости, как и автоматическая. Время жизни статической локальной переменной − промежуток времени между первым обращением к функции, ее содержащей, и окончанием работы программы. Инициализация переменной происходит только при первом обращении к функции. Компилятор хранит значение переменной от одного вызова функции до другого.

Внешний, глобальный (***extern***) класс памяти. Глобальные переменные объявляются вне функций и доступны во всех функциях, находящихся ниже описания глобальной переменной. В момент создания глобальная переменная инициализируется нулем. Включение ключевого слова *extern* позволяет функции использовать внешнюю переменную, даже если она определяется позже в этом или другом файле. Память для глобальных переменных выделяется в начале программы и освобождается при завершении ее работы.

Регистровый, локальный (***register***) класс памяти. Является всего лишь «пожеланием» компилятору помещать часто используемую переменную в регистры процессора для ускорения скорости выполнения программы. Если компилятор отказался помещать переменную в регистры процессора, то переменная становится «автоматической».

13. Рекурсивные алгоритмы

13.1. Понятие рекурсии

Рекурсивным называется способ построения объекта, при котором определение объекта включает в себя аналогичный объект в виде некоторой его части. Решать задачу рекурсивно – это значит разложить ее на подзадачи, которые затем аналогичным образом (т. е. рекурсивно) разбиваются на еще меньшие подзадачи, и так до тех пор, пока на определенном уровне подзадачи не становятся настолько простыми, что могут быть решены тривиально. Путем последовательного решения всех элементарных подзадач можно получить решение всей задачи. Функция называется рекурсивной, если в теле функции содержится вызов аналогичной функции.

Например, необходимо вычислить факториал числа n ($n!$). Известно, что $n! = n \cdot (n-1)!$. Следовательно, для вычисления $n!$ необходимо вычислить $n \cdot (n-1)!$, в свою очередь для вычисления $(n-1)!$ вычисляем $(n-1) \cdot (n-2)!$, для вычисления $(n-2)!$ вычисляем $(n-2) \cdot (n-3)!$ и т. д. На каждом шаге значение вычисляемого факториала уменьшается на единицу. Задача разбивается до тех, пока значение n не станет равно 0, т. е. не будет получено тривиальное решение $0! = 1$. Текст программы вычисления факториала:

```
int fact(int n)
{
    if (n <= 0) return 1;
    else return n*fact(n-1);
}
```

Рассмотрим работу функции для расчета 4!. Процесс рекурсивных вызовов и возврата значений показан на рис. 13.1.

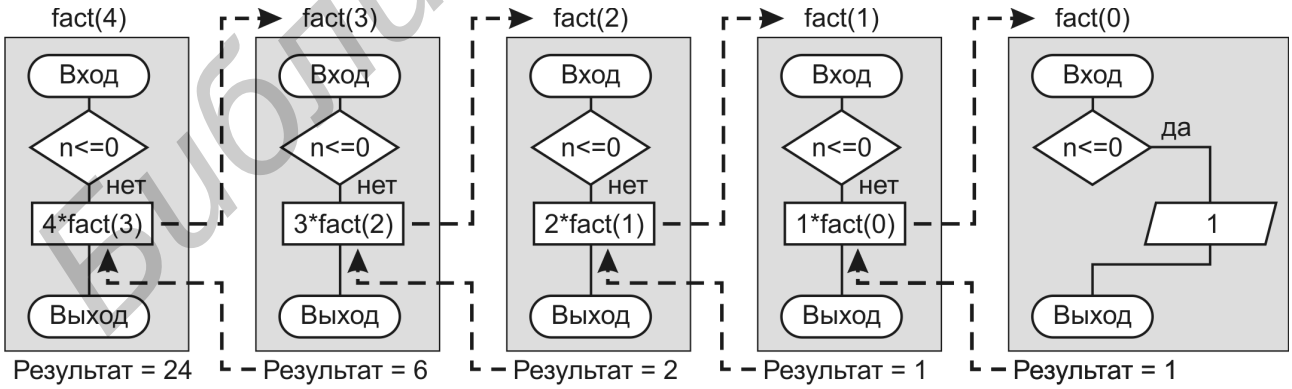


Рис. 13.1

При выполнении программы данные хранятся в специальной области памяти, называемой *стеком*. В стековой области памяти хранятся значения переменных, для которых автоматически выделяется память. Структура стека такова, что в него можно последовательно вносить данные, а затем извлекать в об-

ратном порядке (первый вошел − последний вышел). При каждом вызове рекурсивной функции локальные данные сохраняются в стеке. После достижения дна рекурсии происходит последовательная выборка данных из стека.

13.2. Условие окончания рекурсивного алгоритма

Если в рекурсивном алгоритме не предусмотрено условие окончания рекурсивных вызовов функции, то такой алгоритм будет вызывать ее бесконечно (до тех пор, пока не будет переполнен стек). Поэтому в программе обязательно должен присутствовать оператор, позволяющий при определенных значениях текущих данных прекращать рекурсивный вызов функции. Также для предотвращения переполнения программного стека необходимо делать оценку максимальной глубины рекурсии.

Бесконечная рекурсия может возникать не только при отсутствии условия прекращения рекурсивного вызова функции, но и при неполном учете всех возможных путей движения рекурсии. Например, если факториал рассчитывается следующим образом:

```
int fact(int n)
{
      if (n == 0) return 1;
      else return n*fact(n-1);
}
```

то при вводе числа, меньшего нуля, функция будет вызываться бесконечно.

13.3. Целесообразность использования рекурсии

Рекурсивные алгоритмы хорошо подходят для задач, допускающих рекурсивное разбиение на элементарные подзадачи. Однако это не означает, что для решения таких задач бесспорно использование рекурсивных программ. В большинстве случаев использование рекурсии абсолютно неэффективно.

Недостатки рекурсии.

1. Большой расход памяти и ресурсов. Это вызвано тем, что при каждом вызове подпрограммы система оставляет в памяти все локальные данные. Для обработки сложной цепочки рекурсивных вызовов требуется выделение больших ресурсов системы.

2. Часто при использовании рекурсивной программы некоторые вычисления выполняются многократно, что существенно снижает быстродействие программы. Классический пример – вычисление чисел Фибоначчи.

3. Часто, несмотря на кажущуюся простоту, программы сложны для понимания и для отладки.

Несмотря на недостатки, рекурсивные алгоритмы используются достаточно часто, т. к. существует ряд задач, решить которые без использования рекурсии достаточно сложно.

13.4. Примеры рекурсивных алгоритмов

Пример 13.1. Найти сумму $S_n = \sum_{i=1}^{n} a_i$

```
int sumr(int i)
{
  if (i < 0) return 0;
     else return a[i] + sumr(i-1);
}
```

Пример 13.2. Найти наибольший общий делитель двух чисел. Эйлер обнаружил следующее соотношение: если B делится на A нацело, то НОД(A, B) = A; иначе НОД(A, B) = НОД(B %A, A).

```
int nodr(int a, int b)
{       if( b%a == 0) return a;
                  else return nodr(b%a,a);    }
```

Пример 13.3. Найти max ($a_1 ... a_n$). Данную задачу можно разбить на следующие **элементарные подзадачи:** max (max ($a_1 ... a_{n-1}$), a_n), и далее max (max (max ($a_1 ... a_{n-2}$), a_{n-1}) a_n),…, каждая из которых решается выбором: если ($x > y$), тогда $mx=x$, иначе $mx=y$. На последнем уровне окажется **тривиальная задача** max (a_1) $\rightarrow$ $mx=a_1$, после чего находится max(mx, a_2) и т. д.

```
int maxr2(int i)
{
       if (i == 0) return a[0];
       else   {
         int mx = maxr2(i-1);
           if (a[i] > mx)  return a[i];
                  else return mx;
       }
}
```

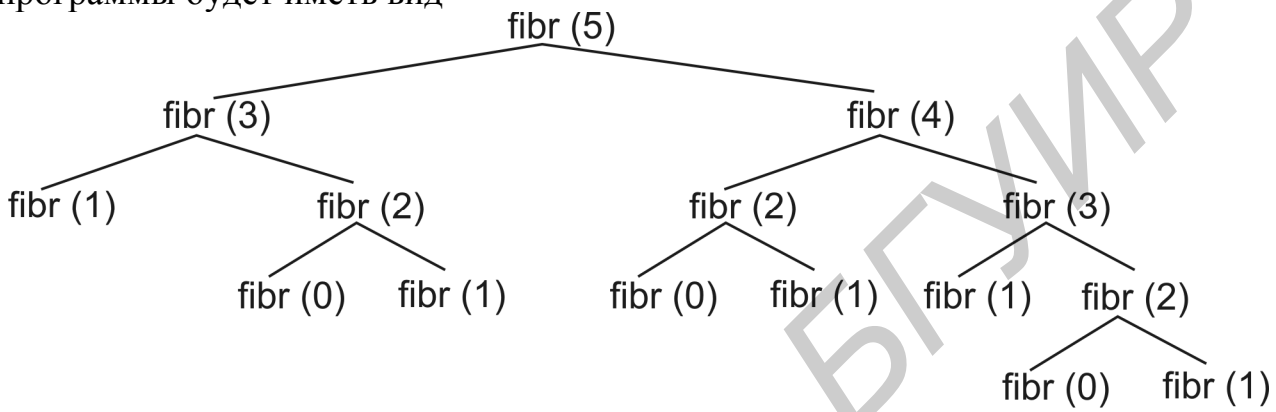
Пример 13.4. Найти сумму элементов одномерного массива. При рекурсивном разбиении массив следует делить на две половины.

```
int sumr(int a[], int i, int j)
{
       if (i == j) return a[i];
       else
        return sumr(a,i,(j+i)/2) + sumr(a,(j+i)/2+1,j);
}
```

Пример 13.5. Вычисление чисел Фибоначчи, которые определяются следующим рекурсивным соотношением: $b_0 = 0$; $b_1 = 1$, $b_n = b_{n-1} + b_{n-2}$.

```
int fibr (int n)
{   if (n <= 1) return n;
    else return fibr(n-1)+fibr(n-2);    }
```

Программа fibr имеет изящный код, однако работает неэффективно. Каждое обращение к функции приводит к вызову еще двух функций. С увеличением n число вызовов возрастает как 2^{n-1}. Например, при $n = 5$ дерево вызовов программы будет иметь вид



Видно, что при выполнении программы требуется стековая память для хранения данных для 16 (2^4) функций. Большим недостатком алгоритма является могократный вызов функции с одинаковыми параметрами.

Так как функция Фибоначчи растет достаточно быстро, то для больших значений n рекурсивный алгоритм будет работать медленно или совсем перестанет работать из-за переполнения стека. Поэтому практического интереса такая рекурсивная программа не представляет.

Рассмотренный выше пример показывает, что компактная и красивая программа не всегда эффективна. Для вычисления чисел Фибоначчи удобно использовать обычный итерационный алгоритм или функцию с одним рекурсивным вызовом:

```
int fibri (int x1, int x2, int n)
{
if (n == 1) return x2;
   else   if (n == 0) return x1;
                else    {
  x2 += x1;
  x1 = x2-x1;
      return fibri(x1, x2, n-1);
      }
}
```

Обращение к функции: fibri(0,1,n);

Пример 13.6. Задача о Ханойской башне. Даны три стержня, на один из которых нанизаны *n* колец. Кольца отличаются размером и расположены меньшее на большем. Необходимо перенести башню на другой стержень. За один раз разрешается переносить только одно кольцо, причем нельзя класть большее кольцо на меньшее. Для промежуточного хранения дисков можно использовать третий стержень.

Решение задачи для одного диска:

− переложить диск с *первого* стержня на *второй* стержень.

Решение задачи для двух дисков:

− переложить диск с *первого* стержня на *третий* стержень;

− переложить диск с *первого* стержня на *второй* стержень;

− переложить диск с *третьего* стержня на *второй* стержень;

Башня, состоящая из *n* дисков, рассматривается как башня из двух дисков: первый диск − верхний диск башни, а второй диск − все диски, располагающиеся под верхним диском. После перестановки этих двух составных дисков задача решается для $n - 1$ дисков.

```
void hanoy(int n, int sterg1, int sterg2, int sterg3)
{
      if (n > 0) {
      hanoy(n-1,sterg1,sterg3,sterg2);
          cout << "perenesty disk s "<<sterg1<<" na "<<sterg2<<endl;
      hanoy(n-1,sterg3,sterg2,sterg1);
                }
}
```

Обращение к функции: hanoy(n,1,2,3);

Пример 13.7.

Вычислить $y = x^n$ по следующему алгоритму: $y = (x^{n/2})^2$, если *n* четное; $y = x \cdot x^{n-1}$, если *n* нечетное.

```
double st(int n)
{   if (n == 0) return 1;
 else  {
      if (n%2 == 0)    {
            double p = st(n/2);
            return p * p;
                  }
      else return x * st(n-1);
       }
}
```

14. Алгоритмы сортировки

Под сортировкой понимается процесс перегруппировки элементов массива, приводящий к их упорядоченному расположению относительно заданного ключа. Ключом в зависимости от решаемой задачи может считаться любое поле структуры. Целью сортировки является облегчение поиска элементов.

Сортировку массивов принято называть *внутренней* в отличие от сортировки файлов (списков), которую называют *внешней*.

Для оценки эффективности сортировки часто используют следующие критерии:

1. *Скорость* сортировки. Определяется числом сравнений и обменов. Оценивается также скорость сортировки *в наилучшем и наихудшем случаях*, т. к. существуют алгоритмы, которые, имея хорошую среднюю скорость, медленно работают в наихудшем случае.

2. *Естественность* сортировки. Сортировка называется *естественной*, если время сортировки минимально для уже упорядоченного массива и увеличивается по мере возрастания степени неупорядоченности массива.

3. *Устойчивость* сортировки. Алгоритм сортировки является устойчивым, если в отсортированном массиве элементы с одинаковыми ключами располагаются в том же порядке, в котором они располагались в исходном массиве. Лучшими считаются алгоритмы, не переставляющие элементы с одинаковыми ключами.

Часто оценивается *сложность алгоритма* − зависимость объема работы, выполняемой некоторым алгоритмом, от размера входных данных.

Существует три основных способа сортировки:

1. Обмен. При таком способе меняются местами элементы, расположенные не по порядку. Обмен продолжается до тех пор, пока все элементы не будут упорядочены.

2. Выбор. Вначале ищется наименьший элемент и ставится на первое место, затем ищется следующий по значимости элемент и устанавливается на второе место и т. д. В результате все элементы помещаются в нужные позиции.

3. Вставка. Последовательно перебираются все элементы. Каждый элемент перемещается в ту позицию, где он должен стоять.

14.1. Простые методы сортировки

14.1.1. Метод пузырька

Самый известный, самый популярный и один из самых худших алгоритмов сортировки (сложность алгоритма − $O(n^2)$). Данная сортировка относится к классу обменных сортировок. Ее алгоритм содержит многократные сравнения соседних элементов и при необходимости их обмен. Элементы ведут себя подобно пузырькам воздуха в воде − каждый из них поднимается на свой уровень.

```
void s_puz(tmas a[], int n)
{
tmas t;
  int i, j;
  for(i=1; i < n; i++)
    for(j=n-1; j >= i; j--)
      if(a[j-1].key > a[j] .key)
  {
       t = a[j-1];
      a[j-1] = a[j];
      a[j] = t;
       }
}
```

В пузырьковой сортировке количество сравнений всегда одинаково независимо от изначальной упорядоченности массива.

14.1.2. Шейкерная сортировка

Пузырьковая сортировка имеет следующую особенность: неупорядоченные элементы на «дальнем» конце массива занимают правильные положения за один проход, а неупорядоченные элементы в начале перемещаются на свои места очень медленно. Для устранения этой особенности можно последовательно чередовать направление проверки элементов. Таким образом, сильно удаленные элементы быстро станут на свои места. Данная модификация пузырьковой сортировки называется *шейкерной сортировкой* (сложность алгоритма − $O(n^2)$).

```
void s_shaker(tmas a[], int n)
{
  tmas t;
   int i, j, k, left = 0, right = n;
 do
 {
 for(j=n-1; j>left; j--)
    if(a[j-1].key > a[j].key)
  {
      t = a[j-1];
       a[j-1] = a[j];
       a[j] = t;
  k = j;
      }
```

```
  left = k+1;
  for(i=1; i < right; i++)
if(a[i-1].key > a[i].key)
   {
       t = a[i-1];
        a[i-1] = a[i];
        a[i] = t;
 k = i;
}
right = k-1;
 } while(left < right);
}
```

Количество сравнений при такой сортировке равно количеству сравнений при пузырьковой сортировке, однако количество обменов несколько уменьшилось.

14.1.3. Сортировка выбором

В массиве выбирается элемент с наименьшим значением и меняется местами с первым элементом. Затем из оставшихся элементов находится наименьший и меняется местами со вторым элементом и т. д. Сложность алгоритма − $O(n^2)$.

```
void s_vb(tmas a[], int n)
{
   int imin, i, j;
    tmas t;
 for(i=0; i<n-1; i++)
 {
    imin = i;
  for(j=i+1; j<n; j++)
  if (a[imin].key > a[j].key) imin = j;
   if (imin != i)
   {
          t = a[imin];
          a[imin] = a[i];
          a[i] = t;
    }
    }
}
```

Количество сравнений в такой сортировке такое же, как и в пузырьковой, а количество обменов намного меньше.

14.1.4. Сортировка вставкой

Сначала сортируются два первых элемента массива. Затем алгоритм вставляет третий элемент в необходимую позицию по отношению к первым двум элементам. После этого четвертый элемент помещается в соответствующую позицию списка из трех элементов. Процесс повторяется до тех пор, пока не будут вставлены все элементы. Сложность алгоритма − $O(n^2)$.

```
void s_vst(tmas a[], int n)
{
 int i, j;
 tmas t;
 for(i=1; i<n; i++)
 {
        t = a[i];
  for(j=i-1;  j>=0 && t.key<a[j].key;  j--) a[j+1] = a[j];
        a[j+1] = t;
 } }
```

В отличие от рассмотренных ранее сортировок количество сравнений в сортировке вставками зависит от изначальной упорядоченности списка. В худшем случае сортировка вставками работает так же медленно, как и пузырьковая сортировка, а в среднем она немного быстрее. Тем не менее у сортировки вставками есть два преимущества. Во-первых, ее поведение естественно, а во-вторых, она устойчива.

14.2. Улучшенные методы сортировки

Все алгоритмы, рассмотренные выше, имеют один фатальный недостаток – они работают очень медленно. Применяемые методы оптимизации кода не дают существенного прироста производительности алгоритма. Существует правило: если используемый в программе алгоритм слишком медленный сам по себе, никакой объем ручной оптимизации не сделает программу достаточно быстрой. Решение заключается в применении лучшего алгоритма сортировки.

14.2.1. Метод Шелла

Общая идея заимствована из сортировки вставкой. Сначала сортируются элементы, расположенные на расстоянии трех позиций друг от друга. Затем сортируются элементы, расположенные на расстоянии двух позиций. Наконец, сортируются все соседние элементы. Последовательность шагов может быть и другой, однако последний шаг обязательно должен быть равнен 1. Сложность алгоритма − $O(n \log^2 n)$.

```
void s_shell(tmas a[], int n)
{
  int i, j;
  tmas t;
for(int d=3; d>0; d--)
  for(i=d; i<n; i++)
 {
    t = a[i];
 for(j=i-d;  j>=0 && t.key<a[j].key;  j-=d)  a[j+d] = a[j];
       a[j+d] = t;
 }
}
```

Количество сдвигов элементов существенно снижено по сравнению с простыми методами сортировки. Скорость сортировки Шелла зависит от выбранной последовательности шагов.

14.2.2. Сортировка слиянием

Алгоритм сортировки слиянием (сложность алгоритма − O(n logn)) следующий:

1. Сортируемый массив рекурсивно разбивается на смежные участки примерно одинакового размера до тех пор, пока в каждом участке не останется по одному элементу.

2. Смежные упорядоченные участки массива соединяются в один упорядоченный участок.

Функция слияния:

```
void slip(int left, int m, int right)
{
 int i = left, j = m+1, k = 0;
while ((I <= m) && (j <= right))
{
 if (a[i].key < a[j].key) {c[k] = a[i]; i++; k++; }
        else {c[k] = a[j]; j++; k++; }
}

while (I <= m) {c[k] = a[i]; i++; k++; }
while (j <= right) {c[k] = a[j]; j++; k++; }

for (k=0, i=left; i<=right; i++, k++) a[i] = c[k];
}
```

Функция сортировки:

```
void s_sl(int left, int right)
{
if (left < right)
{
 int m = (left+right)/2;
s_sl(left,m);
s_sl(m+1,right);
slip(left,m,right);
}
}
```

Подключение:

```
s_sl(0,n-1);
```

Сначала программа последовательно разбивает массив на смежные участки примерно одинакового размера до тех пор, пока в каждом участке не останется по одному элементу. Затем в обратном порядке происходит слияние смежных участков. Для чего последовательно извлекаются наименьшие элементы из двух массивов и помещаются в результирующий масив. Когда один из массивов заканчивается, все оставшиеся элементы этого массива перемещаются в результирующий массив.

Сортировка слиянием является одним из самых эффективных по времени, однако затратным по памяти методом, и поэтому, как правило, применяется для внешней сортировки.

14.2.3. Метод QuickSort (быстрая сортировка или Хоара)

В основе данной сортировки лежит пузырьковая сортировка. Сначала выбирается базовый элемент (средний или выбранный случайным образом). Затем элементы, большие или равные базовому, перемещаются на одну сторону, а меньшие − на другую. После этого аналогичные действия повторяются отдельно для каждой части. Процесс повторяется до тех пор, пока массив не будет отсортирован. Сложность алгоритма − O(n logn). Алгоритм по своей сути рекурсивный, поэтому его можно реализовать в виде рекурсивной функции.

```
void s_qsr(int left, int right)
{
  int i = left, j = right;
  tmas t, x;
x = a[(i+j)/2];
do {
    while (a[i].key < x.key && i < right) i++;
```

```
    while (a[j].key > x.key && j > left)   j--;
  if (l <= j) {
       t = a[i];
       a[i] = a[j];
       a[j] = t;
  i++;  j--;
   }
  } while( i<=j );
  if(left < j)  s_qsr(left, j);
  if(i < right) s_qsr(i, right);
}
```

Количество сравнений для данной сортировки намного меньше, чем у любого ранее рассмотренного метода.

Для быстрой работы алгоритма QuiskSort необходимо правильно выбрать базовый элемент. Если значение базового элемента при каждом делении будет равно наибольшему значению, то сортировка по скорости станет равной пузырьковой. Методика выбора базового элемента должна отталкиваться от природы сортируемого массива. Например, если данные расположены достаточно равномерно, то удобнее выбирать средний элемент. В других случаях можно использовать случайный выбор базового элемента.

Описанная выше рекурсивная реализация сортировки имеет красивый и понятный алгоритм, однако знание особенностей функционирования рекурсивных программ позволяет предположить, что нерекурсивная реализация будет работать лучше:

```
void s_qs(tmas a[], int n)
{

struct
{
  int l;
  int r;
} stack[20];

int i, j, left, right, s = 0;
tmas t, x;
stack[s].l = 0; stack[s].r = n-1;

while (s != -1)
{
```

```
left = stack[s].l; right = stack[s].r;
s--;
while (left < right)
{
i = left; j = right; x = a[(left+right)/2];
while (i <= j)
{
while (a[i].key < x.key) i++;
while (a[j].key > x.key) j--;
if (i <= j) {
t = a[i];  a[i] = a[j];  a[j] = t;
i++;  j--;
}
}

if ((j-left) < (right-i)) // Выбор более короткой части
{
if (i < right) {s++;  stack[s].l = i;  stack[s].r = right; }
right = j;
}
else {
if (left < j) {s++;  stack[s].l = left;  stack[s].r = j; }
left = i;
}
}
}
}
```

В данной функции в качестве базового выбирается средний элемент. Массив просматривается слева направо до тех пор, пока не будет найден элемент больший или равный базовому, и справа налево до тех пор, пока не будет найден элемент, меньший или равный базовому. Найденные элементы меняются местами. Если найденный слева элемент стоит правее элемента, найденного при поиске справа, то поиск прекращается. Массив разбивается на два новых участка. Поиск и разбиение продолжаются до тех пор, пока каждая из частей не будет состоять из одного единственного элемента.

15. Алгоритмы поиска

Цель поиска состоит в нахождении элемента, имеющего заданное значение ключевого поля.

15.1. Линейный поиск

Линейный поиск используется в случае, когда нет никакой дополнительной информации о местоположении разыскиваемых данных. Он представляет собой последовательный перебор массива до обнаружения требуемого ключа или до конца, если ключ не обнаружен:

```
int p_lin1(tmas a[], int  n, int x)
{
for(int i=0; i < n; i++)
      if (a[i].key == x) return i;
  return -1;
}
```

В данном алгоритме на каждом шаге делается две проверки: проверка на равенство ключевого поля и искомого ключа и проверка условия продолжения циклического алгоритма. Для исключения проверки условия продолжения циклического алгоритма вводится вспомогательный элемент – *барьер*, который предохраняет от выхода за пределы массива:

```
int p_lin2(tmas a[], int n, int x)
{
 a[n+1].key = x;
  int i = 0;
 while (a[i].key != x) i++;
      if (i == n+1) return -1;
            else return i;
}
```

Эффективность такого алгоритма почти в два раза выше предыдущего.

15.2. Поиск делением пополам

Поиск деления пополам используется, когда данные упорядочены, например, по неубыванию ключевого поля. Алгоритм состоит в последовательном исключении той части массива, в которой искомого элемента быть не может. Для этого берется средний элемент, и если значение ключевого поля этого элемента больше, чем значение искомого ключа, то можно исключить из рассмотрения правую половину массива, иначе исключается левая половина мас-

сива. Процесс продолжается до тех пор, пока в рассматриваемой части массива не останется один элемент.

```
int p_dv(tmas a[], int n, int x)
{
 int i=0, j=n-1, m;
while(i < j)
{
m=(i + j)/2;
if (x > a[m].key) i = m+1; else j = m;
}
if (a[i].key == x) return i;
else return -1;
}
```

15.3. Интерполяционный поиск

Для массивов с равномерным распределением элементов можно использовать формулу, позволяющую определить примерное местоположение элемента:

$$m = i + \frac{(i - j)(x - a[i].key)}{a[i].key - a[j].key},$$

где i, j − начало и конец интервала; x − искомое значение ключевого поля.

```
int p_dv(tmas a[], int n, int x)
{
 int i = 0, j = n-1, m;
while(i < j)
{
 if (a[i].key == a[j].key) // предотвращение деления на нуль
    if (a[i].key == x) return i;
                  else return -1;
      m=i + (j - i) * (x - a[i]) / (a[j] - a[i]);
if (a[m].key == x) return m;
  else
   if (x > a[m].key) i = m+1; else j = m-1;
}
 return -1;
}
```

Данный поиск быстрее двоичного в 3–4 раза, однако вблизи ключевого поля может вести себя неустойчиво. Поэтому обычно несколько шагов делают с использованием интерполяционного поиска, а затем используют двоичный поиск.

16. Динамические структуры данных

16.1. Понятие списка, стека и очереди

Объект данных считается динамической структурой, если его размер, взаимное расположение и взаимосвязи его элементов изменяются в процессе выполнения программы.

Список (list) – последовательность однотипных данных, работа с которыми ведется в оперативной памяти. В процессе работы список может изменять свой размер. Наибольшее распространение получили две формы работы со списком – очередь и стек.

Стек (stek) – список с одной точкой входа. Данные добавляются в список и удаляются из него только с одной стороны последовательности (вершины стека). Таким образом реализуется принцип «последний пришел – первым вышел».

Очередь (turn) – список с одной или двумя точками входа. Данные добавляются в конец очереди, а извлекаются из начала очереди. Таким образом реализуется принцип «первый пришел – первый вышел».

Для работы со списками предусмотрен специальный *рекурсивный тип данных,* в описании которого содержится указатель на аналогичную этому типу структуру.

Чаще всего используется следующая конструкция рекурсивного типа данных:

```
struct tinf
 {
// Набор полей структуры
};

 struct tlist
 {
   tinf s;   // Информационная часть структуры
   tlist *a;   // Адресная часть структуры
  } spis;
```

Для упрощения рассмотрения в дальнейшем будет использоваться структуру следующего типа:

```
struct tlist
 {
   int  inf;  ;    // Информационная часть структуры
   tlist *a;      // Адресная часть структуры
 } sp;
```

Однонаправленные связанные списки организуются следующим образом: память для каждого элемента выделяется отдельно (по мере надобности). В информационную часть помещаются необходимые данные, а в адресную часть – адрес предыдущей или последующей структуры. На рис. 16.1. показано размещение стека в оперативной памяти компьютера (sp − указатель на вершину стека).

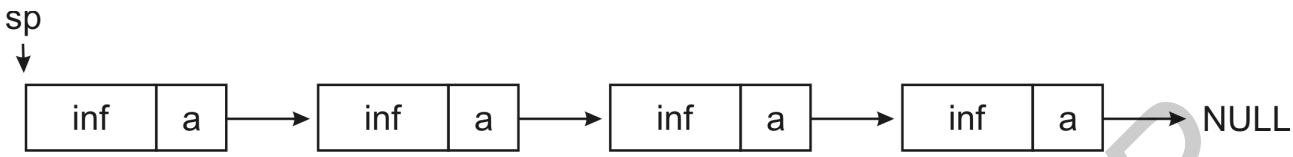


Рис. 16.1

Для перемещения по списку необходимо последовательно переходить от одной ячейки к другой. Такая организация списка называется *косвенной адресацией*. В отличие от адресации по индексу косвенная адресация менее наглядна, однако обладает большей гибкостью.

16.2. Работа со стеками

Добавление элемента в стек

```
tlist *AddStack(tlist *sp, int inf)
{   tlist *spt = new tlist;
 spt->inf = inf;
 spt->a = sp;
 return spt;  }
```

Чтение элемента с удалением

```
tlist *ReadStackD(tlist *sp, int &inf)
{   if (sp == NULL) return NULL;
 tlist *spt = sp;
   inf= sp->inf;
   sp = sp->a;
  delete spt;
 return sp;  }
```

Удаление элемента, следующего за текущим

```
void DelStackAfter(tlist *sp)
 {   if (sp->a == NULL) return ;
 tlist *spt = sp->a;
 sp->a = sp->a->a;
  delete spt;   }
```

Добавление элемента в стек после текущего

```
void AddStackAfter(tlist *sp, int inf)
{   tlist *spt = new tlist;
 spt->inf = inf;
 if (sp->a == NULL) spt->a = NULL;
          else spt->a = sp->a;
 sp->a = spt;   }
```

Удаление всего стека

```
tlist *DelStackAll(tlist *sp)
{    tlist *spt; int inf;
  while(sp != NULL)    {
    spt = sp;
     inf = sp->inf;
   cout << inf << endl;
     sp = sp->a;
   delete spt;            }
 return NULL;   }
```

Обмен элементов, следующих за текущим

```
void RevStackAfter(tlist *sp)
 {  tlist *spt = sp->a->a;
 sp->a->a = spt->a;
 spt->a = sp->a;
 sp->a = spt;  }
```

Сортировка с обменом адресами методом пузырька

```
void SortStackAfter(tlist *sp)
{  if (sp->a->a == NULL) return;
tlist *spt = NULL, *spm;
do {
for (spm=sp; spm->a->a != spt; spm=spm->a)
if (spm->a->inf > spm->a->a->inf) RevStackAfter(spm);
spt = spm->a;
} while (sp->a->a != spt);  }
```

Сортировка стека

```
tlist *SortStack(tlist *sp)
{ tlist *spt = new tlist;
```

```
spt->a = sp;
 sp = spt;
 SortStackAfter(sp);
sp = sp->a;
  delete spt;
return sp;   }
```

Поиск в стеке

```
tlist * PoiskStack(tlist *sp, int x) // Поиск
{
if (sp==NULL) return NULL;
 tlist *spt=sp;

while (spt->inf != x && spt->a != NULL) spt=spt->a;
if (spt->inf == x) return spt;
else return NULL;
}
```

16.2. Работа с однонаправленными очередями

Добавление элемента в очередь

```
void Addoch(tlist **sp,tlist **spk, int inf)
{
  tlist *spt = new tlist;
spt->inf = inf;
spt->a = NULL;
if (*spk == NULL) // Если нет элементов
*sp = *spk = spt;
   else
{
      (*spk)->a = spt;
*spk = spt;
}
return;
}
```

Подключение:

```
sp = spk = NULL;
 Addoch(&sp, &spk, информация);
```

Чтение элемента с удалением

```
tlist *ReadochD(tlist *sp, int &inf)
 {
if (sp == NULL) return NULL;
    inf = sp->inf;
   tlist *spt = sp;
   sp = sp->a;
  delete spt;
 return sp;
}
```

Удаление элемента, следующего за текущим

```
void DelOchAfter(tlist *sp)
{
if (sp->a == NULL) return;
tlist *spt = sp->a;
sp->a = sp->a->a;
  delete spt;
}
```

Удаление всей очереди

```
void DelOchAll(tlist **sp, tlist **spk) // Удаление всей очереди
{
 tlist *spt; int inf;

  while(*sp != NULL)
  {
    spt = *sp;
     inf = (*sp)->inf;
   cout << inf << endl;
     *sp = (*sp)->a;
   delete spt;
  }
 *spk = NULL;
}
```

16.3. Работа с двусвязанными списками

Двусвязанный список состоит из структур, содержащих поля для хранения адресов предыдущего и последующего элементов. Такая организация позволяет осуществлять перемещение по списку в любом направлении.

Объявление двусвязанной структуры:

```
struct tlistdbl
 {
   int inf;
   tlistdbl *left;
   tlistdbl *right;
 } *sp;
```

Чтобы избавиться от необходимости написания алгоритмов обработки крайних элементов, создается каркас двусвязанной структуры, состоящий из двух крайних, не имеющих информационной части, элементов. После этого любые добавляемые в список элементы будут являться внутренними элементами этого списка (рис. 15.6).

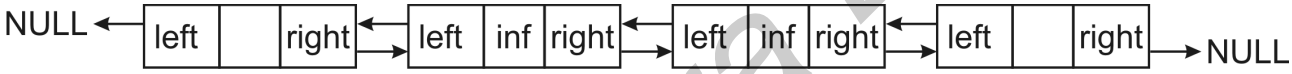


Рис. 15.6

Создание очереди.

```
void NewOchd(tlistdbl **sl, tlistdbl **sr)
{
  *sl = new tlistdbl;
  *sr = new tlistdbl;
(*sl)->left  = NULL;
 (*sl)->right = *sr;
 (*sr)->left = *sl;
(*sr)->right = NULL;
return;
}
```

Подключение:

```
tlistdbl *sl, *sr;
NewOchd(&sl, &sr);
```

Добавление элемента после заданного

```
void AddochdRight(tlistdbl *sp, int inf)
{
```

```
  tlistdbl *spt = new tlistdbl;
spt->inf = inf;
   spt->left = sp;
spt->right = sp->right;
sp->right = spt;
spt->right->left = spt;
return;
}
```

Добавление элемента перед заданным

```
void AddochdLeft(tlistdbl *sp, int inf)
{
  tlistdbl *spt = new tlistdbl;
spt->inf = inf;
   spt->left = sp->left;
spt->right = sp;
spt->left->right = spt;
sp->left = spt;
return;
}
```

Чтение и удаление элемента с адресом sp

```
int ReadochdD(tlistdbl *sp)
{
   int inf = sp->inf;
 sp->left->right = sp->right;
 sp->right->left = sp->left;
  delete sp;
 return inf;
}
```

Удаление всего списка

```
void DelOchdAll(tlistdbl **sl, tlistdbl **sr)
{
 tlistdbl *spt = (*sl)->right;
  while(spt != *sr)
  {
  cout << ReadochdD(spt) << endl;
```

```
    spt = (*sl)->right;
    }
    delete *sl;  *sl = NULL;
    delete *sr;  *sr = NULL;
   return;
  }
```

Сортировка слиянием двусвязанного списка

Разбиение списка на 2 списка

```
  void div2Ochd(tlistdbl *sl, tlistdbl *sr,tlistdbl **slL, tlistdbl **srL,tlistdbl
  **slR, tlistdbl **srR)
    {
  NewOchd(slL,srL);
  NewOchd(slR,srR);

  tlistdbl *spt = sl->right;
   while(spt != sr)
   {
      AddochdLeft(*srL, ReadochdD(spt));
      spt = sl->right;
    if (spt != sr)
    {
     AddochdLeft(*srR, ReadochdD(spt));
      spt = sl->right;
    }
    }
    delete sl;
    delete sr;
  }
```

Слияние двух отсортированных списков

```
  void slipOchd(tlistdbl **sl, tlistdbl **sr,tlistdbl *slL, tlistdbl *srL,tlistdbl
  *slR, tlistdbl *srR)
  {
  NewOchd(sl,sr);
  tlistdbl *sptL = slL->right;
  tlistdbl *sptR = slR->right;
   while ((sptL != srL) && (sptR != srR))
```

```
  {
    if (sptL->inf < sptR->inf)
  {
      AddochdLeft(*sr, ReadochdD(sptL));
      sptL = slL->right;
  }
  else
  {
      AddochdLeft(*sr, ReadochdD(sptR));
      sptR = slR->right;
  }
   }
   while (sptL != srL)
   {
      AddochdLeft(*sr, ReadochdD(sptL));
      sptL = slL->right;
   }
     delete slL; delete srL;
    while (sptR != srR)
   {
      AddochdLeft(*sr, ReadochdD(sptR));
      sptR = slR->right;
   }
      delete slR; delete srR;
  }
```

Сортировка

```
  void SotrSlipOchd(tlistdbl **sl, tlistdbl **sr)
   {
      tlistdbl *slL, *srL,*slR, *srR;
     if ((*sl)->right->right == *sr) return;
  div2Ochd(*sl, *sr, &slL, &srL, &slR, &srR);
  SotrSlipOchd(&slL, &srL);
  SotrSlipOchd(&slR, &srR);
     slipOchd(sl, sr, slL, srL, slR, srR);
  }
```

16.4. Работа с двусвязанными циклическими списками

Циклические списки – одно- или двунаправленные очереди, в которых последний элемент указывает на начало очереди. Рассмотрим циклическую двунаправленную очередь (рис. 16.3). Понятия начала и конца очереди здесь не имеют смысла, достаточно знать адрес любого элемента очереди.

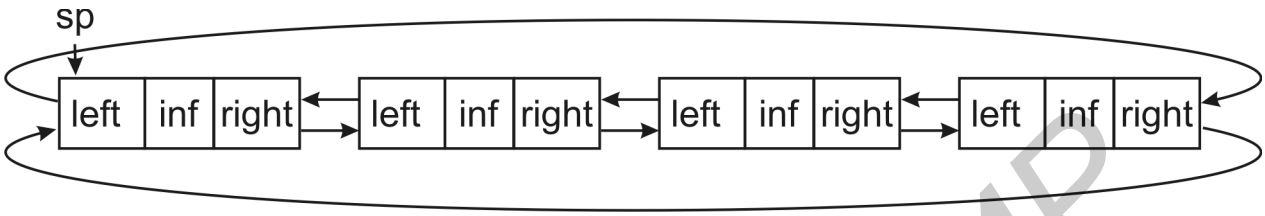


Рис. 16.3

Добавление элемента в циклический список

```
tlistdbl *AddochdC(tlistdbl *sp, int inf)  //
{
  tlistdbl *spt = new tlistdbl;
spt->inf = inf;

if (sp == NULL)
{
spt->left = spt;
spt->right = spt;
}
else
{
spt->right = sp->right;
spt->left = sp->right ->left;
sp->right ->left = spt;
sp->right = spt;
}
return spt;
}
```

Подключение

```
tlistdbl *sp = NULL;
sp = Addo chdC(sp, информация);
```

Просмотр списка:

```
while (условие)
```

```
{
cout << sp->inf << endl;
sp = sp ->right;
}
```

Удаление всего списка

```
tlistdbl *DelOchdCAll(tlistdbl *sp)
{
    tlistdbl *spt;
while(sp->right != sp)
{
 cout << sp->inf << endl;
 sp->left->right = sp->right;
 sp->right->left = sp->left;
 spt = sp;
 sp = sp->right;
  delete spt;
}

cout << sp->inf << endl;
 delete sp;
return NULL;
}
```

17. Нелинейные списки

17.1. Древовидные структуры данных

Рассмотрим древовидную структуру данных (рис. 17.1).

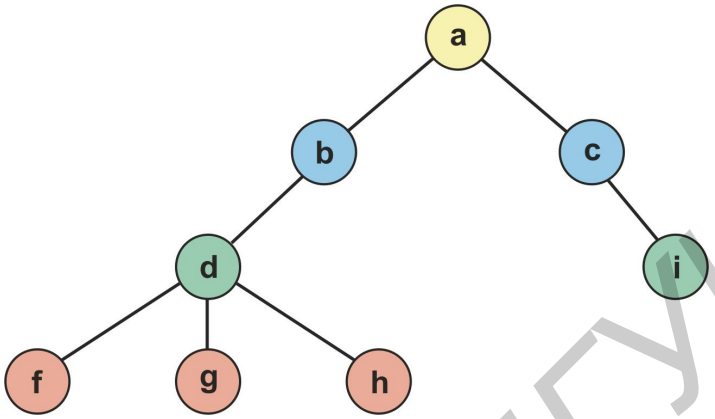


Рис. 17.1

Все данные называются ***узлами***.

Связи между узлами называется ***ветвями***.

Самый верхний узел – ***корень*** дерева (*a*).

Узлы, из которых не выходят связи, – ***листы*** дерева (*f, g, h, i*).

Узел, находящийся непосредственно над другим, называется ***родительским узлом*** *(для узла d узел b является родительским)*. Узел, находящийся непосредственно ниже, называется ***дочерним*** *(для узла b узел d является дочерним)*.

Все узлы, находящиеся выше рассматриваемого, являются его ***предками*** (*для узла d предки b и a*), а все узлы, находящиеся ниже, – ***потомками*** *(для узла b потомки – d, f, g, h)*.

Узлы, имеющие одного и того же родителя, называются ***сестринскими*** *(f, g, h)*.

Узел, не являющийся листом, называется ***внутренним*** *(b или d или c или a).*

Порядок узла (или степень узла) – количество дочерних узлов (для узла *b* порядок 1, для узла *d* порядок 3).

Степень дерева – это максимальный порядок его узлов (*рассматриваемое дерево имеет третий порядок*). Дерево второй степени называется бинарным или двоичным. Дерево степени три называется троичным деревом.

Глубина узла – число предков плюс единица (*например, для узла d глубина равна 3*).

Глубина дерева – наибольшая глубина всех узлов (*для данного дерева – 4*).

17.2. Использование древовидных структур

Для работы с древовидными структурами используется следующая конструкция рекурсивного типа:

```
struct ttree
 {
   tinf inf;
   ttree *a1;
   ttree *a2;
…
   ttree *an;
 } *proot, *p;
```

Рассмотрим размещение в памяти структуры, указанной на рис. 17.1 структуры.

```
ttree *proot, *p;
 proot = new ttree;
 proot->inf = 'a';     proot->a2 = NULL;
 p = proot;
 p->a1 = new ttree;
        p = p->a1;
        p->inf = 'b';            p->a2 = NULL;              p->a3 = NULL;
p->a1 = new ttree;
        p = p->a1;
         p->inf = 'd';
p->a1 = new ttree;
         p->a1->inf = 'f';     p->a1->a1 = NULL;         p->a1->a2 = NULL;
        p->a1->a3 = NULL;
p->a2 = new ttree;
         p->a2->inf = 'g';    p->a2->a1 = NULL;          p->a2->a2 = NULL;
        p->a2->a3 = NULL;
p->a3 = new ttree;
         p->a3->inf = 'h';    p->a3->a1 = NULL;          p->a3->a2 = NULL;
        p->a3->a3 = NULL;
         p = proot;
p->a3 = new ttree;
         p = p->a3;
         p->inf = 'c';            p->a1 = NULL;       p->a2 = NULL;
p->a3 = new ttree;
         p = p->a3;
         p->inf = 'i';    p->a1 = NULL;     p->a2 = NULL;    p->a3 = NULL;
```

Как видно из приведенного выше фрагмента программы, непосредственное заполнение даже небольшого дерева требует довольно громоздкой последовательности команд. Поэтому для работы с деревьями используют набор специфических алгоритмов.

Обходом дерева называется последовательное обращение ко всем его узлам. Следующая рекурсивная процедура осуществляет такой обход с распечаткой каждого узла:

```
void obh(ttree *p)  // Обход всего дерева
{
 if (p == NULL) return;
 // вывод при прямом обходе
  obh (p->a1);
  obh (p->a2);

  obh tree(p->an);
// вывод при обратном обходе
}
```

Прямой обход: ***a b d f g h c i***.
Обратный обход: ***f g h d b i c a***.

17.3. Двоичное дерево поиска

Если ключевые поля в дереве расположены таким образом, что для любого узла значения ключа у левого преемника меньше, чем у правого, то такое дерево называется *двоичным деревом поиска*. Предположим, что имеется набор данных, упорядоченных по ключу: *key*: 1, 5, 6, 9, 14, 21, 28, 32, 41. Для таких данных двоичное дерево поиска выглядит следующим образом (рис. 17.2):

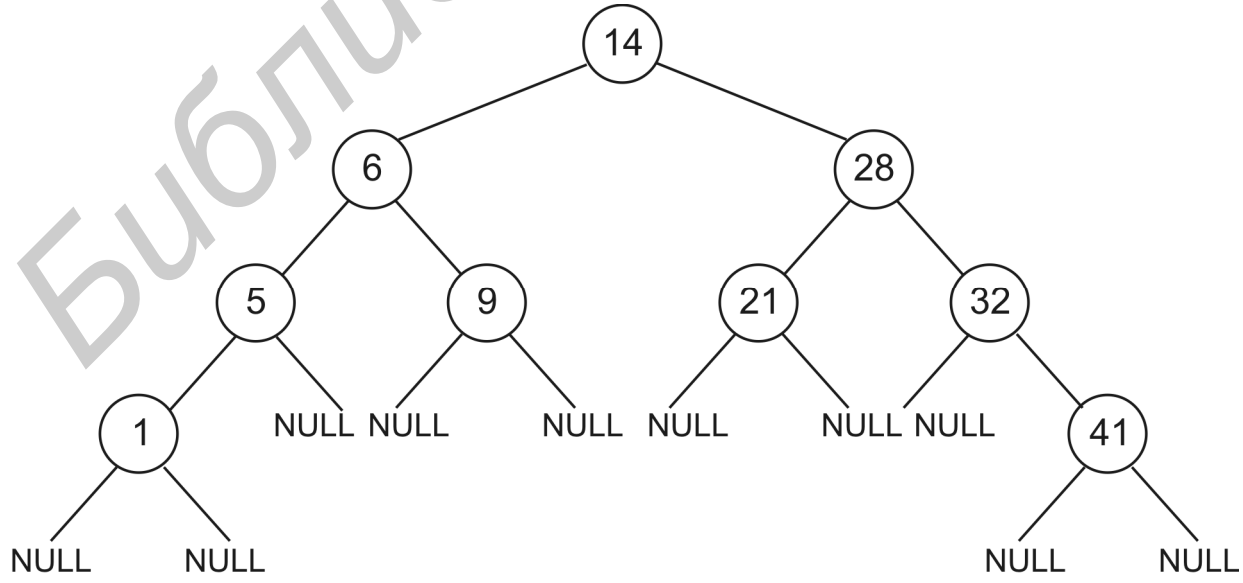


Рис. 17.2

Эффективность поиска информации в такой динамической структуре данных сравнима с эффективностью двоичного поиска в массиве.

Дерево, у которого узлы, имеющие только одну дочь, располагаются не выше двух последних уровней, называется **сбалансированным деревом.**

Для работы с двоичным деревом поиска используется следующая конструкция рекурсивного типа:

```
struct ttree
{
  int inf;
  ttree *left;
  ttree *right;
} *proot;
```

Добавление нового элемента

```
ttree *addtree(ttree *proot, int inf)
 {
ttree *nl, *pr, *ps;
bool b;

  nl = new ttree;
  nl->inf = inf;
  nl->left  = NULL;
  nl->right = NULL;
  if (proot == NULL) return nl;
      ps = proot;
    while (ps != NULL)
       {
 pr = ps;
  b = (inf < ps->inf);
  if (b) ps = ps->left;
   else ps = ps->right;
      }
   if (b) pr->left = nl;
    else pr->right = nl;
return proot;
}
```

Если вводить данные со случайным чередовании ключей, данный алгоритм формирует неплохо сбалансированное дерево. Однако если ключи в исходном множестве частично упорядочены, то получаемое дерево поиска оказывается сильно разбалансированным, и его эффективность для организации поиска оказывается сравнимой с линейным поиском в массиве.

Построение сбалансированного дерева

Для построения сбалансированного дерева можно использовать предварительно отсортированный массив. Тогда программа имеет вид

```
ttree *addBtree(int L, int R, int *mas)  {
ttree *ps;
int m;
 if (L > R) return NULL;

 m = (L + R) / 2;

  ps = new ttree;
  ps->inf = mas[m];

  ps->left = addBtree(L,m-1,mas);
  ps->right = addBtree(m+1,R,mas);
  return ps;
}
```

Симметричный обход дерева

```
void wrtree(ttree *p)
{
 if (p == NULL) return;
  wrtree(p->left);
     cout << p->inf  << "  ";
  wrtree(p->right);
}
```

Поиск элемента с заданным ключом

```
void poisktree(ttree *p, int key, bool &b, int &inf)
 {
if ((p != NULL) && (b != true))
{
```

```
        if (p->inf != key)
        {
    poisktree(p->left, key, b, inf);
    poisktree(p->right, key, b, inf);
        }
        else
        {
                b = true;
                inf = p->inf;
        }
}
    return;
}
```

Поиск элемента с максимальным ключом

```
int poiskmaxtree(ttree *p)
{
while (p->right != NULL) p = p->right;
return p->inf;
}
```

Удаление всего дерева

```
ttree *deltree(ttree *p)
{
 if (p == NULL) return NULL;
    deltree(p->left);
     deltree(p->right);

    delete(p);
   p = NULL;
return NULL;
}
```

Удаление элемента с заданным ключом

Возможны три варианта размещения удаляемого узла:

1. Если удаляется узел, не имеющий потомков (рис. 17.3).

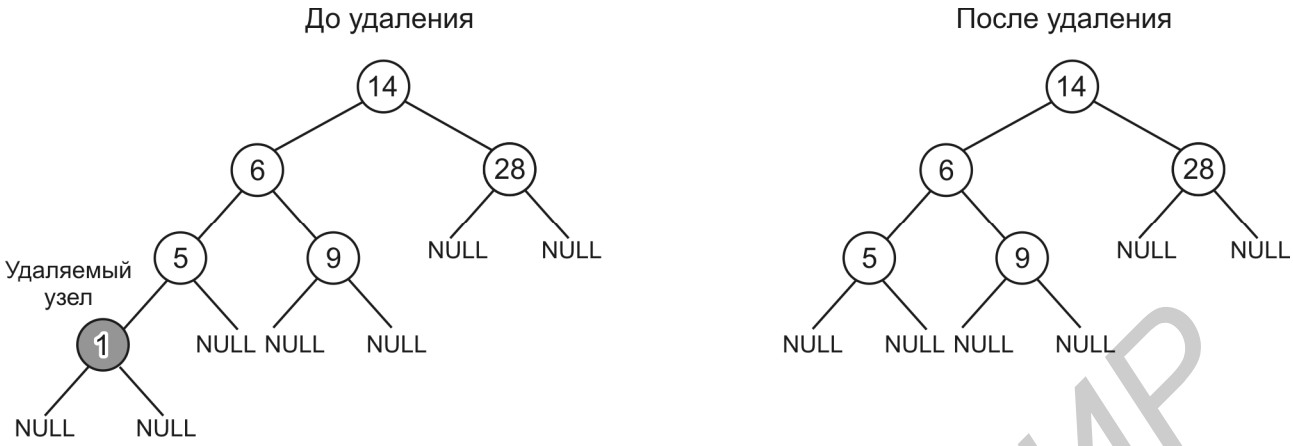


Рис. 17.3

2. Если удаляется узел, имеющий одну дочь (рис. 17.4).

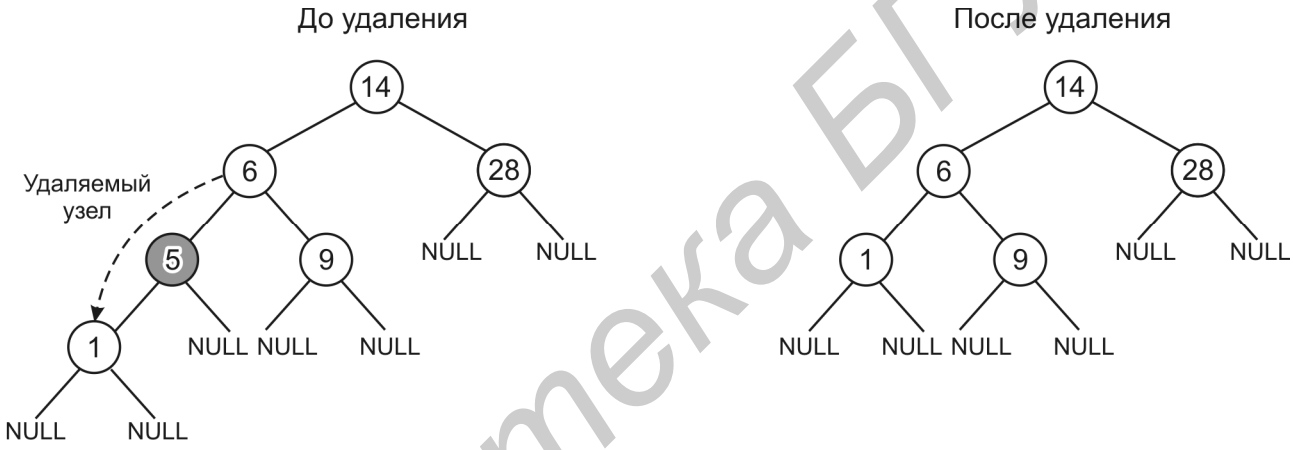


Рис. 17.4

3. Если удаляется узел, имеющий двух дочерей, то удаляемый узел заменяется узлом, имеющим наибольший ключ в левом поддереве либо наименьший ключ в правом поддереве (рис. 17.5).

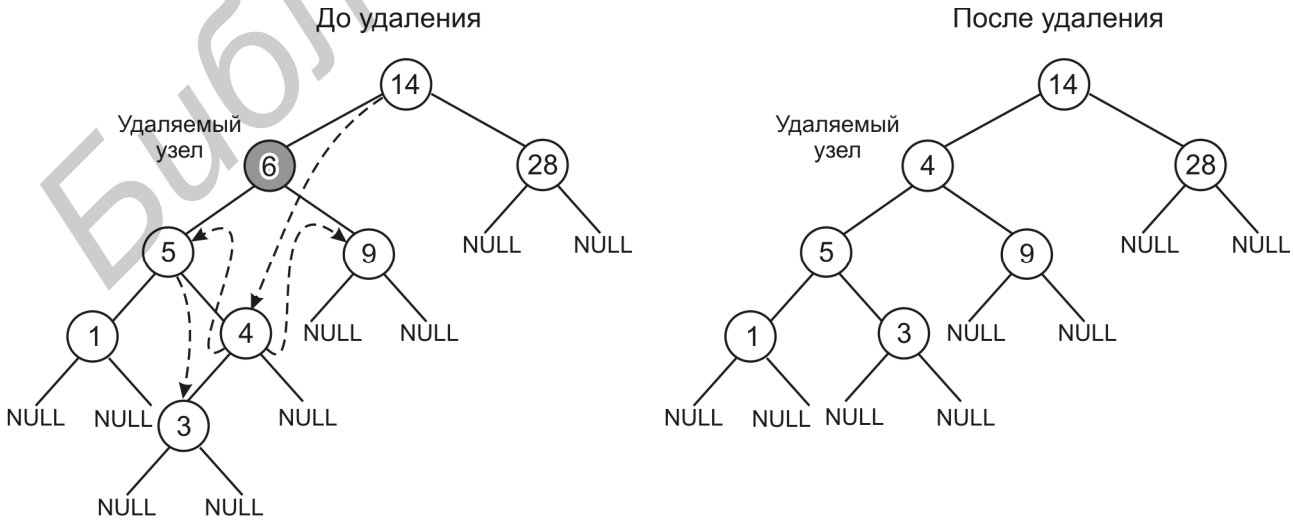


Рис. 17.5

Текст программы удаления в дереве узла, имеющего ключ, равный inf:

```
ttree *dellist(ttree *proot, int inf)
{
ttree *ps = proot, *pr = proot, *w, *v;
// Поиск удаляемого узла

while ((ps != NULL) && (ps->inf != inf))
{
pr = ps;
 if (inf < ps->inf) ps = ps->left;
            else ps = ps->right;
}

if (ps == NULL) return proot; // Если узел не найден


// Если узел не имеет дочерей
if ((ps->left == NULL) && (ps->right == NULL))
{
      if (ps == pr) // Если это был последний элемент
            {
            delete(ps);
            return NULL;
            }
      if (pr->left == ps) // Если удаляемый узел слева
             pr->left = NULL;
      else                              // Если удаляемый узел справа
              pr->right = NULL;
      delete(ps);
      return proot;
}

// Если узел имеет дочь только справа
 if (ps->left == NULL)
 {
      if (ps == pr) // Если удаляется корень
```

```
                {
            ps = ps->right;
            delete(pr);
            return ps;
                }

    if (pr->left == ps) // Если удаляемый узел слева
            pr->left = ps->right;
    else                        // Если удаляемый узел справа
            pr->right = ps->right;
    delete(ps);
  return proot;
}

// Если узел имеет дочь только слева
 if (ps->right == NULL)
 {
    if (ps == pr) // Если удаляется корень
                {
            ps = ps->left;
            delete(pr);
            return ps;
                }

    if (pr->left == ps) // Если удаляемый узел слева
            pr->left = ps->left;
    else                        // Если удаляемый узел справа
            pr->right = ps->left;
    delete(ps);
  return proot;
 }

// Если узел имеет двух дочерей
 w = ps->left;
 if (w->right == NULL) // Если максимальный следует за ps
        w->right = ps->right;
 else   // Если максимальный не следует за ps
```

```
 {
   while (w->right != NULL)
   {
    v = w;
    w = w->right;
   }
v->right = w->left;
w->left = ps->left;
w->right = ps->right;
 }

 if (ps == pr) // Если удаляется корень
                        {
                delete(ps);
                return w;
                        }

        if (pr->left == ps) // Если удаляемый узел слева
                    pr->left = w;
        else                            // Если удаляемый узел справа
                    pr->right = w;
        delete(ps);
    return proot;
}
```

18. Синтаксический анализ арифметических выражений

Одной из главных причин появления языков программирования высокого уровня явилось наличие вычислительных задач с большим объемом рутинных вычислений. Основным требованием к этим языкам является максимальное приближение формы записи арифметических выражений к естественному языку математики. Выражения в математике обычно записываются в *инфиксной* форме, например $(a + b) \cdot (k - d)$. Главным неудобством для компьютерной обработки таких выражений является наличие скобок, с помощью которых меняют стандартный порядок выполнения операций. Поэтому одной из первых задач системного программирования являлось исследование способов синтаксического анализа арифметических выражений. Среди полученных результатов наиболее удачным является использование *постфиксной* (знак операции ставится после операндов) формы представления арифметических выражений, предложенной польским математиком Я. Лукашевичем. Такая форма записи арифметических выражений получила название обратной польской записи (ОПЗ). Удобство использования ОПЗ состоит в том, что при записи выражений скобки не нужны, а полученная последовательность операндов и операций удобна для расшифровки.

18.1. Алгоритм преобразования выражения в форму ОПЗ

Эдсгер Дейкстра изобрел алгоритм для *преобразования выражений из инфиксной формы в форму ОПЗ*. Из-за сходства последовательности операций с происходящим на железнодорожной сортировочной станции алгоритм получил название «сортировочная станция».

Суть алгоритма заключается в следующем. Строка последовательно просматривается слева направо. Операнды сразу добавляются в выходную строку. Остальные символы обрабатываются следующим образом:

1. Если текущий символ − операция, а стек пуст, то операция записывается в стек.

2. Если текущий символ − открывающая скобка, то она записывается в стек.

3. Если текущий символ − закрывающая скобка, то элементы из стека извлекаются в выходную строку до тех пор, пока верхним элементом стека не станет открывающая скобка. Открывающая скобка удаляется из стека, но в выходную строку не добавляется.

4. Если текущий символ − операция, а стек не пуст, то из стека в выходную строку переносятся все операции с большим или равным приоритетом. После этого текущая операция помещается в стек.

5. После просмотра всех символов в строке операции, оставшиеся в стеке, извлекаются и помещаются в выходную строку.

Алгоритм *вычисления выражения, записанного в форме ОПЗ*, основан на использовании стека. При просмотре выражения слева направо значения операндов заносятся в стек. Если найдена операция, то из стека извлекаются два операнда, к которым применяется найденная операция. Результат заносится в стек. После выполнения всех операций в стеке остается одно значение − результат вычисления арифметического выражения.

18.2. Программа для вычисления арифметических выражений

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

struct tstk
 {
   double inf;
   tstk *a;
};

tstk *AddStack(tstk *sp, double inf)
{  tstk *spt = new tstk;
spt->inf = inf;
spt->a = sp;
 return spt;  }

tstk *ReadStack(tstk *sp, double &inf)
{
 tstk *spt = sp;
  inf = sp->inf;
  sp = sp->a;
 delete spt;
 return sp;  }

double masz[122];
char str[100], strp[100];

int priority(char ch)  // Вычисление приоритета операций
```

```
{
 switch (ch)
 {
case '(': case ')': return 0;
case '+': case '-': return 1;
case '*': case '/': return 2;
default : return -1;
 }
}

void AddPostFix(char *strin, char *strout)
{
tstk *sp=NULL;
int n=0;
char ch;
double inf;
for(unsigned int i=0; i<strlen(strin); i++)
{
ch=strin[i];
// Если это операнд
if (ch >= 'A') { strout[n++] = ch; continue; }
// Если стек пуст или найдена открывающая скобка
if (sp == NULL || ch == '(' ) { sp = AddStack(sp,ch); continue; }
// Если найдена открывающая скобка
if ( ch == ')' )
{
while (sp->inf != '(')
{
    sp = ReadStack(sp,inf);
    strout[n++] = (char)inf;
}
    sp=ReadStack(sp,inf); // Удаление открывающей скобки
 continue;
}
// Если операция
 int pr = priority(ch);
 while (sp != NULL && priority((char)sp->inf)>=pr)
```

```
 {
  sp=ReadStack(sp,inf);
  strout[n++] = (char)inf;
 }
  sp = AddStack(sp,ch);
}  // end for

 while (sp != NULL)
 {
  sp=ReadStack(sp,inf);
  strout[n++] = (char)inf;
 }
strout[n++] = '\0';
}

double rasAV(char *str, double *mz)
{
tstk *sp=NULL;
char ch;
double inf, inf1, inf2;
for (unsigned int i=0; i<strlen(str); i++)
{
ch = str[i];
// Если найден операнд
if (ch >= 'A') { sp = AddStack(sp,mz[int(ch)]); continue; }
// Если найден знак операции
sp = ReadStack(sp,inf2);
sp = ReadStack(sp,inf1);
 switch (ch)
 {
   case '+': sp = AddStack(sp, inf1 + inf2); break;
   case '-': sp = AddStack(sp, inf1 - inf2); break;
   case '*': sp = AddStack(sp, inf1 * inf2); break;
   case '/': sp = AddStack(sp, inf1 / inf2); break;
 }
}
sp = ReadStack(sp,inf);
```

```
return inf;
}

int main()
{
cout << "Vvedite a" << endl;  cin >> masz[int('a')];
cout << "Vvedite b" << endl;  cin >> masz[int('b')];
cout << "Vvedite c" << endl;  cin >> masz[int('c')];
cout << "Vvedite d" << endl;  cin >> masz[int('d')];
cout << "Vvedite f" << endl;  cin >> masz[int('f')];
cout << " Vvedite viragenie (a ,b, c, d, f) " << endl;
cin >> str;
        AddPostFix(str, strp);
cout << endl << strp;
    double s = rasAV(strp,masz);
cout << endl <<" Res = " << s << endl;
return 0;
}
```

19. Хеширование

19.1. Понятие хеширования

Для решения задачи быстрого поиска был придуман алгоритм *хеширования* (*hashing*), при котором ключи данных записываются в особую хеш-таблицу. Затем при помощи некой простой функции *i = h(key)* алгоритм хеширования определяет положение искомого элемента в таблице по значению его ключа.

Рассмотрим пример.

Имеется массив из 7 элементов, значения ключей которых находятся в диапазоне 0…15.

```
mas[0].key = 5;
mas[1].key = 15;
mas[2].key = 1;
mas[3].key = 10;
mas[4].key = 8;
mas[5].key = 3;
mas[6].key = 11;
```

Допустим, что надо найти элемент с ключом 3. Для этого метод линейного поиска сделает 6 шагов, а для использования двоичного поиска потребуется предварительная сортировка. Количество шагов зависит от способа сортировки, но затраты в этом случае будут выше, чем при линейном поиске.

Для ускорения поиска создадим новый массив (хеш-таблицу), в котором номер элемента будет равен значению ключа:

```
H[ Mas[i].key ] = Mas[i];
```

Все неиспользуемые элементы массива H имеют значение –1:

```
H[0].key = -1;          H[8].key = 8;
H[1].key = 1;           H[9].key = -1;
H[2].key = -1;          H[10].key = 10;
H[3].key = 3;           H[11].key = 11;
H[4].key = -1;          H[12].key =-1;
H[5].key = 5;           H[13].key = -1;
H[6].key = -1;          H[14].key = -1;
H[7].key = -1;          H[15].key = 15;
```

При такой организации для нахождения любого элемента достаточно сделать только один шаг. Для удаления элемента достаточно поставить значение –1 в соответствующее поле.

Для реальных задач такой подход неприемлем, т. к. размер массива должен быть достаточен для размещения элемента с максимальным ключом, что

существенно увеличивает размер хеш-таблицы. Например, для хранения телефонной базы с семизначными номерами необходим массив из 9 999 999 элементов. Для уменьшения размера хеш-таблицы используются различные схемы хеширования.

19.2. Схемы хеширования

Для уменьшения количества элементов в хеш-таблице используют различные алгоритмы сжатия ключей. Однако в этом случае высока вероятность того, что несколько различных элементов получат одинаковый номер в хеш-таблице. Для решения данной проблемы схема хеширования должна иметь алгоритм разрешения конфликтов, который определяет поведение программы в случае, если новый ключ попадает на уже занятую позицию. Методов разрешения конфликтов достаточно много, однако все они имеют одинаковую структуру:

1. С использованием значения ключа вычисляется номер позиции в хеш-таблице.

2. Если полученная позиция уже занята, то алгоритм разрешения конфликтов находит новую позицию.

3. Если новая позиция тоже занята, повторяется п. 2 до тех пор, пока не будет найдена свободная позиция.

Алгоритм размещения, использующий заданный метод разрешения конфликтов, размещает элементы в хеш-таблице.

Алгоритм поиска находит по значению ключа позицию искомого элемента в хеш-таблице. Если полное значение ключа элемента не совпадает с искомым ключом, то осуществляется дальнейший поиск, начиная с найденной позиции, в соответствии с алгоритмом разрешения конфликтов.

19.3. Хеш-таблица с линейной адресацией

Используется функция хеширования $i = Key \% M$. Алгоритм разрешения конфликтов следующий: если найденная позиция i уже занята, то ищется первая незанятая позиция.

Например, имеется следующий массив:

```
Mas[0].key = 5;
Mas[1].key = 15;
Mas[2].key = 3;
Mas[3].key = 10;
Mas[4].key = 125;
Mas[5].key = 333;
Mas[6].key = 11;
Mas[7].key = 437;
```

Данные размещаются в хеш-таблице. Функция размещения имеет вид

$$i = key \% \ 10.$$

Получается следующая хеш-таблица:

```
H[0] =  10
H[1] =  11
H[2]  =  -1
H[3]  =  3
H[4]  =  333
H[5]  =  5
H[6] =  15
H[7]  =  125
H[8]  =  437
H[9]  =  -1
```

Текст программы, реализующей заданный алгоритм:

```
# include <iostream.h>
# include <math.h>

void sv_add(int inf, int m, int *H)
{
int i = inf % m;
if (H[i] != -1)
 while (H[i] != -1)
 {
        i++;
       if (i == m) i=0;
 }
H[i] = inf;
}

int sv_seach(int inf, int m, int H[])
{
int i = abs (inf % m);
while (H[i] != -1 )
 {
  if (H[i] == inf) return i;
  i++;
      if (i >= m) i=0;
 }
```

```
 return -1;
}

void main()
{
int n = 8; // Число элементов в массиве
int mas[] = {5, 15, 3, 10, 125, 333, 11, 437};
int m = 10; // Число элементов  в хеш-таблице
int H[10];
int i;

for(i=0; i<m; i++) H[i] = -1; // Элемент не занят
for(i=0; i<n; i++) sv_add(mas[i],m,H);

int ss = 0, ii;
while (ss != -1)
{ cin >> ss;
ii= sv_seach(ss,m,H);
if (ii == -1) cout << "Net elementa" << endl;
   else cout << H[ii] << endl;
}
}
```

Достоинство: простой алгоритм вставки и поиска элементов.

Недостатки:

1. Невозможность изменения размера хеш-таблицы.

2. Сложный алгоритм удаления элемента, т. к. удаление элемента часто приводит к необходимости перестройки всей таблицы. Для преодоления данного недостатка можно использовать несколько состояний ячейки: «занята», «не занята», «удалена». Если во время поиска алгоритм попадает на ячейку со статусом «удалена», то поиск продолжается далее. При добавлении данных ячейка со статусом «удалена» считается свободной.

3. Если данные в таблице расположены неравномерно, то скорость поиска может быть очень плохой. Для преодоления данного недостатка можно использовать следующую хеш-функцию: $i = (key + M) \% 10$, где M – простое число, которое может быть сгенерировано датчиком случайных чисел. Для правильной работы датчик должен всегда устанавливаться в одинаковое начальное положение.

19.4. Хеш-таблицы с квадратичной и произвольной адресацией

Методы содержат такую же функцию хеширования, что и в методе с линейной адресацией, однако поиск свободной ячейки у них отличен от единицы. Алгоритм разрешения конфликтов в *методе с квадратичной адресацией* следующий: если найденная позиция i уже занята, то ищется первая незанятая позиция по формуле: $i = i + p^2$ (p – номер попытки).

В *методе с произвольной адресацией* незанятая позиция ищется по формуле: $i = i + r_p$ (r – заранее сгенерированный массив случайных чисел; p – номер попытки).

По сравнению с линейной адресацией данные методы дают более равномерное распределение данных в таблице, однако работают несколько медленнее.

19.5. Хеш-таблица с двойным хешированием

В отличие от метода с линейной адресацией данный метод использует две хеш-функции:

1. Нахождение позиции элемента в хеш-таблице $i = Key \% M$.

2. Если ячейка с найденным номером i свободна, то перейти к п. 6, иначе – к п. 3.

3. Вычисление значения $c = 1 + (Key \% (M - 2))$.

4. Нахождение новой позиции элемента в хеш-таблице $i = i - c$. Если $i < 0$, то $i = i + M$.

5. Если ячейка с найденным номером i свободна, то перейти к п. 6, иначе – к п. 4.

6. Вставить элемент в найденную позицию.

По сравнению с предыдущими данный метод из-за независимых друг от друга цепочек поиска свободной ячейки дает более равномерное распределение данных в хеш-таблице. Усложнение алгоритма приводит к снижению скорости его работы.

19.6. Хеш-таблица на основе связанных списков

Одним из наиболее эффективных методов разрешения конфликтов состоит в том, что элементы, попадающие на одну и ту же позицию, размещаются в связанных списках. Например, имеется следующий массив:

```
Mas[0].key = 5;
Mas[1].key = 15;
Mas[2].key = 3;
Mas[3].key = 10;
Mas[4].key = 125;
Mas[5].key = 333;
Mas[6].key = 11;
Mas[7].key = 437;
```

Данные размещаются в хеш-таблице. Функция размещения имеет вид

$$i = key \% \ 10.$$

Получается следующая хеш-таблица:

H[0] ← 10
H[1] ← 11
H[2] ← NULL
H[3] ← 3 ← 333
H[4] ← NULL
H[5] ← 5 ← 15 ← 125
H[6] ← NULL
H[7] ← 437
H[8] ← NULL
H[9] ← NULL

Текст программы:

```
# include <iostream.h>
# include <math.h>

struct tstk
 {
   int    inf;
   tstk *a;
 };

tstk ** sv_create(int m)
{
tstk **H = new tstk*[m];
  for(int i=0; i<m; i++) H[i] = NULL;
  return H;
}

void sv_add(int inf, int m, tstk **H)
{
  tstk *spt = new tstk;
spt->inf = inf;
int i = inf % m;

if (H[i] == NULL) {H[i] = spt; spt->a = NULL;}
```

```
            else { spt->a = H[i]; H[i] = spt;}
}

tstk *sv_seach(int inf, int m, tstk **H)
{
int i = abs(inf % m);
tstk *spt = H[i];
 while ( spt != NULL)
 {
    if (spt->inf == inf) return spt;
 spt=spt->a;
  }
 return NULL;
}

void sv_delete(int m, tstk **H)
{
tstk *spt,*sp;
for(int i=0; i<m; i++)
{
cout << "H["<<i<<"]= ";
sp = H[i];
while (sp != NULL)
{
cout << sp->inf << " ";
spt = sp;
sp = sp->a;
delete spt;
}
 cout << endl;
}
 delete []H;
}

void main()
{
int n = 8; // Число элементов  в массиве
```

```
int mas[] = {5, 15, 3, 10, 125, 333, 11, 437};
int m = 10; // Число элементов  в хеш-таблице
int i;

tstk **H;

 H = sv_create(m);
for(i=0; i<n; i++) sv_add(mas[i],m,H);

int ss;
tstk *p;
cin >> ss;
while (ss != -1)
{
p = sv_seach(ss,m,H);
if (p == NULL) cout << "Net elementa" << endl;
   else cout << p->inf << endl;
 cin >> ss;
}
 sv_delete(m,H);
}
```

Достоинства:

1. Достаточно простой алгоритм вставки и поиска элементов.
2. Связанная таблица не может быть переполнена.

Недостаток: плохая работа с неравномерно размещенными данными. Для преодоления этого недостатка используется методика, приведенная в предыдущем примере.

19.7. Метод блоков

Используется массив одномерных массивов одинакового размера (блоков).

Вначале обычная функция $i = Key \ \% \ M$ находит номер блока и размещает элемент в блоке. Если i-й блок переполнен, то элемент помещается в специальный блок переполнения. Этот метод хорошо зарекомендовал себя при хранении хеш-таблицы на файле, т. к. запись и чтение из файла можно осуществлять поблочно, что быстрее поэлементной работы.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

В практикуме приводятся задачи двух уровней сложности. Задачи, обозначенные символом «А», имеют низший уровень сложности, обозначенные символом «В», – более высокий уровень сложности.

1. Программирование линейных алгоритмов

А. *Ввести исходные данные и получить результат.*

А1. Написать программу пересчета веса из фунтов в килограммы (1 фунт = 0.4536 кг).

А2. Написать программу пересчета расстояния из милей в километры (1 миля = 1.609 км).

А3. Перевести дозу радиоактивного излучения из микрозивертов в миллирентгены (1 мкЗв = 0.115 мР).

А4. Перевести температуру из градусов Кельвина в градусы Цельсия (0 ºК = –273.1 К).

А5. Написать программу пересчета объема из галлонов в литры (1 галлон = 3.785 л).

А6. Написать программу пересчета расстояния из морских лиг в километры (1 морская лига = 5.556 км).

А7. Написать программу пересчета веса из унций в граммы (1 унция = = 28.35 гр.).

А8. Написать программу пересчета расстояния из кабельтов в метры (1 кабельтов = 219.5 м).

А9. Написать программу пересчета расстояния из морских миль в километры (1 морская миля = 1.852 км).

А10. Написать программу пересчета длины из ярдов в метры (1 ярд = = 0.9144 м).

А11. Написать программу пересчета объема из нефтяных баррелей в литры (1 баррель = 159 л).

А12. Написать программу пересчета скорости из морских узлов в километры в час (1 узел = 1.852 км/ч)

А13. Написать программу пересчета длины из дюймов в сантиметры (1 дюйм = 2.54 см).

А14. Написать программу пересчета скорости из миль в час в километры в час (1 mph = 1.609 км/ч).

А15. Написать программу пересчета давления из миллиметров ртутного столба в паскали (1 мм рт. ст. = 133.3 Па).

B. *Вычислить значение выражения при заданных исходных данных. Сравнить полученное значение с указанным правильным результатом.*

B1. $s=\dfrac{2\cos\left(x-\dfrac{2}{3}\right)}{\dfrac{1}{2}+\sin^2 y}\left(1+\dfrac{z^2}{3-z^2/5}\right)$

при $x = 14.26$; $y = -1\ \ 22$; $z = 3\ \ 5\cdot10^{-2}$. Ответ $s = 0.749155$.

B2. $s=\dfrac{\sqrt[3]{9+(x-y)^2}}{x^2+y^2+2}-e^{|x-y|}\mathrm{tg}^3 z$

при $x = -4.5$; $y = 0.75\cdot10^{-4}$; $z = -0.845\cdot10^{2}$. Ответ $s = -3.23765$.

B3. $s=\dfrac{1+\sin^2(x+y)}{\left|x-\dfrac{2y}{1+x^2y^2}\right|}x^{|y|}+\cos^2\left(\mathrm{arctg}\dfrac{1}{z}\right)$

при $x = 3.74\cdot10^{-2}$; $y = -0.825$; $z = 0.16\cdot10^{2}$. Ответ $s = 1.05534$.

B4. $s=\left|\cos x-\cos y\right|^{\left(1+2\sin^2 y\right)}\left(1+z+\dfrac{z^2}{2}+\dfrac{z^3}{3}+\dfrac{z^4}{4}\right)$

при $x = 0.4\cdot10^{4}$; $y = -0.875$; $z = -0.475\cdot10^{-3}$. Ответ $s = 1.98727$.

B5. $s=\ln\left(y^{-\sqrt{|x|}}\right)\left(x-\dfrac{y}{2}\right)+\sin^2(\mathrm{arctg}(z))$

при $x = -15.246$; $y = 4.642\cdot10^{-2}$; $z = 21$. Ответ $s = -182.038$.

B6. $s=\sqrt{10\left(\sqrt[3]{x}+x^{y+2}\right)}\left(\arcsin^2 z-|x-y|\right)$

при $x = 16.55\cdot10^{-3}$; $y = -2.75$; $z = 0.15$. Ответ $s = -40.6307$.

B7. $s=5\mathrm{arctg}(x)-\dfrac{1}{4}\arccos(x)\dfrac{x+3|x-y|+x^2}{|x-y|z+x^2}$

при $x = 0.1722$; $y = 6.33$; $z = 3.25\cdot10^{-4}$. Ответ $s = -205.306$.

B8. $s=\dfrac{e^{|x-y|}|x-y|^{x+y}}{\mathrm{arctg}(x)+\mathrm{arctg}(z)}+\sqrt[3]{x^6+\ln^2 y}$

при $x = -2.235\cdot10^{-2}$; $y = 2.23$; $z = 15.221$. Ответ $s = 39.3741$.

B9. $s=\left|x^{\frac{y}{x}}-\sqrt[3]{\dfrac{y}{x}}\right|+(y-x)\dfrac{\cos y-\dfrac{z}{(y-x)}}{1+(y-x)^2}$

при $x = 1.825\cdot10^{2}$; $y = 18.225$; $z = -3.298\cdot10^{-2}$. Ответ $s = 1.21308$.

B10. $s=2^{-x}\sqrt{x+\sqrt[4]{|y|}}\sqrt[3]{e^{x-1/\sin z}}$

при $x = 3.981\cdot10^{-2}$; $y = -1.625\cdot10^{3}$; $z = 0.512$. Ответ $s = 1.26185$.

B11. $s = y^{\sqrt[3]{|x|}} + \dfrac{\cos^3(y)}{e^{|x-y|} + \dfrac{x}{2}} \cdot |x-y| \left(1 + \dfrac{\sin^2 z}{\sqrt{x+y}}\right)$

при $x = 6.251$; $y = 0.827$; $z = 25.001$. Ответ $s = 0.712122$.

B12. $s = 2^{\left(y^x\right)} + \left(3^x\right)^y - \dfrac{y\left(\operatorname{arctg} z - \dfrac{1}{3}\right)}{|x| + \dfrac{1}{y^2+1}}$

при $x = 3.251$; $y = 0.325$; $z = 0.466 \cdot 10^{-4}$. Ответ $s = 4.23655$.

B13. $s = \dfrac{\sqrt[4]{y + \sqrt[3]{x-1}}}{|x-y|\left(\sin^2 z + \operatorname{tg} z\right)}$

при $x = 17.421$; $y = 10.365 \cdot 10^{-3}$; $z = 0.828 \cdot 10^5$. Ответ $s = 0.330564$.

B14. $s = \dfrac{y^{x+1}}{\sqrt[3]{|y-2|} + 3} + \dfrac{x + \dfrac{y}{2}}{2|x+y|}(x+1)^{-1/\sin z}$

при $x = 12.3 \cdot 10^{-1}$; $y = 15.4$; $z = 0.252 \cdot 10^3$. Ответ $s = 82.8256$.

B15. $s = \dfrac{x^{y+1} + e^{y-1}}{1 + x|y - \operatorname{tg} z|}\left(1 + |y-x|\right) + \dfrac{|y-x|^2}{2} - \dfrac{|y-x|^3}{3}$

при $x = 2.444$; $y = 0.869 \cdot 10^{-2}$; $z = -0.13 \cdot 10^3$. Ответ $s = -0.498707$.

Пример выполнения лабораторной работы

Условие: написать программу для вычисления линейного арифметического выражения

$$h = \frac{x^{2y} + \mathrm{e}^{y-1}}{1 + x|y - \operatorname{tg} z|} + 10 \cdot \sqrt[3]{x} - \ln(z).$$

При $x = 2.45$, $y = -0.423 \times 10^{-2}$, $z = 1.232 \times 10^3$. Ответ $h = 6.9465$.

Текст программы:

```
#include <iostream.h>
#include <math.h>
int main ()
{
  double x, y, z, a, b, c, h;
      cout << "Vvedite x: ";
         cin >> x;
      cout << "Vvedite y: ";
         cin >> y;
```

```
        cout << "Vvedite z: ";
           cin >> z;
    a = pow(x,2*y)+exp(y-1);
    b = 1+x*fabs(y-tan(z));
    c = 10*pow(x,1/3.)-log(z);
      h = a/b+c;
   cout << "Result h= " << h << endl;
  return 0;
 }
```

2. Программирование разветвляющихся алгоритмов

А. *Ввести исходные данные. Выполнить задание.*

А1. Написать программу выбора наибольшего из трех чисел.

А2. Даны три числа x, y, z. Выяснить, верно ли, что $x > y > z$. Ответ вывести в текстовой форме «верно» или «неверно».

А3. Даны три действительных числа. Перемножить отрицательные числа.

А4. Даны четыре целых числа. Найти сумму положительных чисел.

А5. Даны радиус круга и длина стороны квадрата. Выяснить, у какой фигуры площадь больше? Ответ вывести в текстовой форме «у круга» или «у квадрата».

А6. Даны три действительных числа. Найти сумму положительных чисел.

А7. Даны три действительных числа. Возвести в куб и вывести на экран отрицательные числа.

А8. Даны два целых числа. Если оба числа отрицательные, то вычислить сумму их модулей; если только одно из чисел отрицательное, то вычислить произведение чисел; если оба числа положительные, то результат равен нулю.

А9. Даны три целых числа. Вывести числа, которые делятся на три без остатка.

А10. Даны три действительных числа. Вычесть из большего числа меньшее.

А11. Даны четыре целых числа. Найти произведение отрицательных чисел.

А12. Даны три действительных числа. Перемножить четные числа.

А13. Даны четыре целых числа. От суммы положительных чисел отнять сумму модулей отрицательных чисел.

А14. Даны три целых числа. Вычесть из суммы всех чисел сумму четных чисел.

А15. Даны четыре целых числа. Выяснить, равна ли сумма двух первых чисел сумме двух последних чисел. Ответ вывести в текстовой форме «равна» или «не равна».

B. *Вычислить значение в соответствии с номером варианта. Предусмотреть возможность выбора вида функции f(x): sh(x), x^2 или e^x. Вывести на экран информацию о выполняемой ветви вычислений.*

B1. $$a=\begin{cases}\left(f(x)+y\right)^2-\sqrt[3]{\left|f(x)\right|}, & xy>0\\ \left(f(x)+y\right)^2+\sin(x), & xy<0\\ \left(f(x)+y\right)^2+y^3, & xy=0.\end{cases}$$

B2. $$b=\begin{cases}\ln(f(x))+\sqrt[4]{\left|f(x)\right|}, & x/y=0\\ \ln\left|f(x)/y\right|-y^2, & x/y<0\\ (f(x)^2+y)^3 & \textit{иначе.}\end{cases}$$

B3. $$c=\begin{cases}f(x)^2+\sqrt[3]{y}+\sin\left(y\right), & x-y=0\\ \left(f(x)-y\right)^2+\ln\left(x\right), & x-y>0\\ \left(y-f(x)\right)^2+\mathrm{tg}\left(y\right), & x-y<0.\end{cases}$$

B4. $$d=\begin{cases}\sqrt[3]{\left|f(x)+x\right|}-tg(y), & x>y\\ (y-f(x))^3+\sin(y), & x=y\\ y+x^3-\sqrt{f(x)} & \textit{иначе.}\end{cases}$$

B5. $$e=\begin{cases}\sin(f(x))/3, & xy=0\\ \ln(\left|y-f(x)\right|), & 7<xy<10\\ 2tg^2(x)-y & \textit{иначе.}\end{cases}$$

B6. $$g=\begin{cases}e^{f(x)-y}+\sqrt[3]{x}, & x/y=0\\ x^2-\ln(y^2+x), & -5<x/y<0\\ 2f(x)^2-y^3 & \textit{иначе.}\end{cases}$$

B7. $$s=\begin{cases}\sin^2(x)-f(x), & x+\left|y\right|=0\\ \sqrt[3]{\left|xy\right|}, & x+\left|y\right|<0\\ 3f^2(x) & \textit{иначе.}\end{cases}$$

B8. $$b=\begin{cases}\sqrt{x}/y, & x^2+y=0\\ \cos^3(y)-f(x), & x^2+y<0\\ \sin(\cos(2f(x)) & \textit{иначе.}\end{cases}$$

B9. $$l=\begin{cases}2\sin^2(\ln(|x|)), & y=0\\ tg(y^2-x), & -5<y<0\\ x^2+y-9 & \textit{иначе.}\end{cases}$$

B10. $$k=\begin{cases}\ln\left(|f(x)|+|y|\right), & |xy|>10\\ e^{f(x)+y}, & |xy|<10\\ \sqrt[3]{|f(x)|}+y, & |xy|=10.\end{cases}$$

B11. $$w=\begin{cases}tg^2(x)-f(x), & xy=0\\ e^{2f(x)}-y^2, & 0<xy<10\\ \ln(|y|)+2f(x) & \textit{иначе.}\end{cases}$$

B12. $$g=\begin{cases}y^2\cdot\sin^2(x), & y\cdot f(x)=0\\ tg^2(x)+f(x), & y\cdot f(x)<0\\ 2f(x)-\sin(y) & \textit{иначе.}\end{cases}$$

B13. $$q=\begin{cases}\ln(x)-f^2(x), & yf(x)=10\\ 2y-10\sin(x), & yf(x)<10\\ y^2+f^2(x) & \textit{иначе.}\end{cases}$$

B14. $$u=\begin{cases}\sin(x)+\ln(y), & x^2y=0\\ tg^2(f(x)), & 2<x^2y<7\\ f(x)^2/2+x & \textit{иначе.}\end{cases}$$

B15. $$w=\begin{cases}\sqrt[3]{f(x)}-xy, & 2x/y=0\\ \sin^2(x)-y, & 2x/y<0\\ 4y-tg(x) & \textit{иначе.}\end{cases}$$

3. Программирование циклических алгоритмов

A. *Вывести на экран таблицу значений функции y(x) для x, изменяющегося от a до b с шагом h = (b – a)/10. Задание выбрать в соответствии с номером варианта.*

A1. $$y(x)=\sum_{i=1}^{n}\left(\sin(ix)+\cos^2(i)\right).$$

A2. $y(x)=\sum_{i=1}^{n}\left(5\sin(2ix)-\cos^2(x)\right).$

A3. $y(x)=\sum_{i=1}^{n}\left(2tg(ix)*e^{2i}\right).$

A4. $y(x)=\sum_{i=1}^{n}\left(15x^2-4\cos^3(ix)\right).$

A5. $y(x)=\sum_{i=1}^{n}\left(x^{2i}*\cos^2(2ix)\right).$

A6. $y(x)=\sum_{i=1}^{n}\left(2e^{i\cdot\sin(x)}+3\sqrt{|x|}\right).$

A7. $y(x)=\sum_{i=1}^{n}\left(2\cos(ix)*ch(x)\right).$

A8. $y(x)=\sum_{i=1}^{n}\left(e^{2\cos ix}*x^{\cos(i)}\right).$

B9. $y(x)=\sum_{i=1}^{n}\left(\sin^2(i)-3e^{ix}\right).$

A10. $y(x)=\sum_{i=1}^{n}\left(2tg^2(ix)-\sqrt{|x|}\right).$

A11. $y(x)=\sum_{i=1}^{n}\left(2\ln(ix)-\sin^{2i}(x)\right).$

A12. $y(x)=\sum_{i=1}^{n}\left(4\sqrt[3]{|ix|}+\sin x\right).$

A13. $y(x)=\sum_{i=1}^{n}\left(3e^{ix}+\operatorname{ctg}(x)\right).$

A14. $y(x)=\sum_{i=1}^{n}\left(\sqrt{|\sin 2x|}+e^{-i\sin x}\right).$

A15. $y(x)=\sum_{i=1}^{n}\left(3x^{2i}+4e^{3i}\right).$

B. *Вывести на экран таблицу значений функции y(x) и ее разложения в ряд s(x) для x, изменяющегося от a до b с шагом h = (b – a)/10. Задание выбрать в соответствии с номером варианта в табл. I.*

Таблица I

№	*a*	*b*	Функция	Разложение функции в ряд Тейлора	*k*
B1	0.1	1	$y(x)=\sin(x)$	$s(x)=\sum_{n=0}^{k}(-1)^n\frac{x^{2n+1}}{(2n+1)!}$	160
B2	0.1	1	$y(x)=\mathrm{ch(x)}$	$s(x)=\sum_{n=0}^{k}\frac{x^{2n}}{(2n)!}$	100
B3	0.1	1	$y(x)=e^{x\sin(x)}$	$s(x)=\sum_{n=0}^{k}\frac{(x\cdot\sin(x))^n}{n!}$	120
B4	0.1	1	$y(x)=\cos(x)$	$s(x)=\sum_{n=0}^{k}(-1)^n\frac{x^{2n}}{(2n)!}$	80
B5	0.1	1	$y(x)=\frac{\sin x}{x}$	$s(x)=\sum_{n=0}^{k}(-1)^n\frac{x^{2n}}{(2n+1)!}$	140
B6	0.1	1	$y(x)=\mathrm{sh(x)}$	$s(x)=\sum_{n=0}^{k}\frac{x^{2n+1}}{(2n+1)!}$	80
B7	0.1	1	$y(x)=e^{-2e^x}$	$s(x)=\sum_{n=0}^{k}\frac{2^n(-e^x)^n}{n!}$	120
B8	0.1	1	$y(x)=5^x$	$s(x)=\sum_{n=0}^{k}\frac{x^n\ln^n(5)}{n!}$	100
B9	0.1	1	$s(x)=e^{2x}$	$s(x)=\sum_{n=0}^{k}\frac{(2x)^n}{n!}$	140
B10	0.1	0.5	$y(x)=x^2e^x$	$s(x)=\sum_{n=2}^{k}\frac{x^n}{(n-2)!}$	150
B11	0.1	1	$y(x)=x\sin(x)$	$s(x)=\sum_{n=0}^{k}(-1)^n\frac{x^{2n+2}}{(2n+1)!}$	100
B12	0.1	1	$y(x)=e^{\cos(x)}$	$s(x)=\sum_{n=0}^{k}\frac{\cos^n(x)}{n!}$	80
B13	–2	–0.1	$y(x)=x\cos(x)$	$s(x)=\sum_{n=0}^{k}(-1)^n\frac{x^{2n+1}}{(2n)!}$	140
B14	0.2	0.8	$y(x)=3^{x-1}$	$s(x)=\sum_{n=0}^{k}\frac{(x-1)^n\ln^n(3)}{n!}$	100
B15	0.1	0.8	$y(x)=\cos(2x)$	$s(x)=\sum_{n=0}^{k}(-4)^n\frac{x^{2n}}{(2n)!}$	180

4. Использование одномерных массивов

А. *Ввести с клавиатуры массив из 10 элементов. Выполнить задание, результат вывести на экран.*

А1. Задан массив действительных чисел. Найти сумму положительных и произведение отрицательных элементов массива.

А2. Задан массив целых чисел. Найти произведение четных и сумму отрицательных элементов массива.

А3. Задан массив действительных чисел. Найти разность между суммой положительных элементов и суммой модулей отрицательных элементов.

А4. Задан массив целых чисел. Найти сумму минимального и максимального элементов массива.

А5. Задан массив действительных чисел. Вывести на экран элементы, значение которых больше среднего значения всех элементов массива.

А6. Задан массив целых чисел. Вывести номера минимального и максимального элементов и их значения.

А7. Задан массив действительных чисел. Найти, сколько элементов находится между минимальным и максимальным элементами массива

А8. Задан массив целых чисел. Найти, сколько элементов имеют значение меньше среднего значения всех элементов массива.

А9. Задан массив действительных чисел. Найти сумму четных и произведение отрицательных элементов массива.

А10. Задан массив целых чисел. Отрицательные элементы заменить полусуммой соседних элементов. Крайние элементы не изменять.

А11. Задан массив действительных чисел. Подсчитать количество положительных и сумму отрицательных элементов.

А12. Задан массив целых чисел. Найти количество и сумму элементов, имеющих значения больше 10 и меньше 100.

А13. Задан массив действительных чисел. Найти количество и произведение отрицательных нечетных элементов.

А14. Задан массив целых чисел. Вывести на экран числа, имеющие значения меньше максимального и больше среднего значения всех элементов массива.

А15. Задан массив действительных чисел. Найти среднее значение всех элементов массива и координаты минимального и максимального элементов массива.

В. *Ввести с клавиатуры размер массива, выделить необходимый объем памяти для хранения элементов массива и ввести исходные данные. Выполнить задание, результат вывести на экран.*

В1. Задан массив целых чисел. Отсортировать элементы массива по неубыванию из модулей.

В2. Задан массив целых чисел. Преобразовать массив следующим образом: все отрицательные элементы массива перенести в начало, а все остальные – в конец, сохранив исходное взаимное расположение как среди отрицательных, так и среди положительных элементов.

B3. Задан массив целых чисел. Найти число, наиболее часто встречающееся в этом массиве.

B4. Задан массив целых чисел. Найти числа, входящие в массив не более одного раза.

B5. Задан массив действительных чисел. Сдвинуть элементы массива циклически на *n* позиций вправо (значение *n* задается с клавиатуры).

B6. Задан массив целых чисел. Удалить из массива все числа, встречающиеся в массиве более одного раза.

B7. Задан массив действительных чисел. Перенести максимальный элемент в нулевую позицию, а минимальный – в последнюю позицию массива. Взаимное расположение остальных элементов не должно изменяться.

B8. Задан массив действительных чисел. Удалить все положительные элементы, у которых справа находится отрицательный элемент.

B9. Задан массив целых чисел. Удалить из массива минимальный и максимальный элементы.

B10. Задан массив действительных чисел. Найти сумму элементов, расположенных между минимальным и максимальным элементами массива.

B11. Задан массив целых чисел. Найти произведение элементов, расположенных между последним и предпоследним положительными элементами массива.

B12. Задан массив действительных чисел. Переставить в обратном порядке элементы, расположенные между первым положительным и последним отрицательным элементами массива.

B13. Задан массив целых чисел. Удалить все элементы, стоящие до элемента с максимальным значением.

B14. Задан массив действительных чисел. Определить количество различных элементов в массиве.

B15. Задан массив целых чисел. Найти наименьший положительный элемент среди элементов с четными индексами массива.

5. Использование двумерных массивов

A. *Ввести с клавиатуры двумерный массив размером 5×5 элементов. Выполнить задание, результат вывести на экран.*

A1. Задан массив целых чисел. Подсчитать количество строк, в которых встречаются нулевые элементы.

A2. Задан массив действительных чисел. Вывести координаты минимального элемента в каждом столбце.

A3. Задан массив целых чисел. Вывести количество четных (по значению) элементов в каждой строке.

A4. Задан массив действительных чисел. Вывести число отрицательных элементов в каждом столбце.

A5. Задан массив целых чисел. Вывести среднее значение элементов каждой строки.

А6. Задан массив действительных чисел. Найти минимальное, максимальное и среднее значение всех элементов массива.

А7. Задан массив целых чисел. Найти в каждой строке элемент с минимальным значением.

А8. Задан массив действительных чисел. Вывести среднее значение элементов всех четных строк массива.

А9. Задан массив целых чисел. Вывести максимальный из элементов, расположенных в четных столбцах матрицы.

А10. Задан массив действительных чисел. Вывести среднее значение элементов каждого столбца.

А11. Задан массив целых чисел. Найти в каждом столбце элемент с максимальным значением.

А12. Задан массив действительных чисел. Вывести число положительных элементов в каждой строке.

А13. Задан массив целых чисел. Вывести количество нечетных (по значению) элементов в каждом столбце.

А14. Задан массив действительных чисел. Вывести координаты максимального элемента в каждой строке.

А15. Задан массив целых чисел. Подсчитать количество столбцов, в которых встречаются отрицательные элементы.

В. *Ввести с клавиатуры количество строк и столбцов массива, выделить необходимый объем памяти для хранения элементов массива и ввести исходные данные. Выполнить задание, результат вывести на экран.*

В1. Задана матрица размером $N \times M$. Поменять местами строку, содержащую элемент с максимальным значением, со строкой, содержащей элемент с минимальным значением.

В2. Задана матрица размером $N \times M$. Упорядочить ее столбцы по возрастанию их наименьших элементов.

В3. Задана матрица размером $N \times M$. Удалить столбец матрицы, содержащий элемент с минимальным значением.

В4. Задана матрица размером $N \times M$. Получить одномерный массив, занося в ячейку значение 0, если строка матрицы с таким же номером содержит хотя бы один нулевой элемент, и 1 в противном случае.

В5. Задана матрица размером $N \times M$. Удалить строку с максимальной суммой элементов.

В6. Задана матрица размером $N \times M$. Определить количество «особых» элементов матрицы, считая элемент «особым», если он больше суммы остальных элементов соответствующего столбца.

В7. Задана матрица размером $N \times M$. Упорядочить строки по возрастанию суммы их элементов.

В8. Задана матрица размером $N \times M$. Определить количество различных элементов матрицы (т. е. повторяющиеся элементы считать один раз).

B9. Задана матрица размером *N*×*M*. Поменять местами строку, содержащую максимальный элемент, и строку, содержащую минимальный элемент.

B10. Задана матрица размером *N*×*M*. Вывести все элементы, являющиеся максимальными в своем столбце и одновременно минимальными в своей строке.

B11. Задана матрица размером *N*×*M*. Получить одномерный массив, каждый элемент которого будет содержать значение 0, если строка матрицы с таким же номером упорядочена по возрастанию, и значение 1 в противном случае.

B12. Задана матрица размером *N*×*M*. Удалить строку матрицы, содержащую элемент с максимальным значением.

B13. Задана матрица размером *N*×*M*. Определить количество «особых» элементов матрицы, считая элемент «особым», если он меньше суммы остальных элементов соответствующей строки.

B14. Задана матрица размером *N*×*M*. Поменять местами столбец, содержащий элемент с минимальным значением, со столбцом, содержащим элемент с максимальным значением.

B15. Задана матрица размером *N*×*M*. Упорядочить ее строки по убыванию их максимальных элементов.

6. Программирование с использованием функций

A. *Вывести на экран таблицу значений функции y(x, n) для x, изменяющегося от a до b с шагом h = (b – a)/10. Расчет y(x, n) поместить в функцию. Параметры передавать указанным в табл. II способом.*

Таблица II

Номер варианта	*a*	*b*	*n*	Функция	Способ передачи параметров
1	2	3	4	5	6
A1	0.13	0.9	10	$y(x,n)=\sum_{i=1}^{n}\left(3e^{ix}+ctg(x)\right)$	По ссылке
A2	0.24	1.2	8	$y(x,n)=\sum_{i=1}^{n}\left(\sqrt{\lvert\sin 2x\rvert}+e^{-i\sin x}\right)$	По значению
A3	0.15	0.95	7	$y(x,n)=\sum_{i=1}^{n}\left(2tg(ix)\cdot e^{2i}\right)$	По указателю
A4	0.35	1.25	12	$y(x,n)=\sum_{i=1}^{n}\left(x^{2i}\cdot\cos^2(2ix)\right)$	По ссылке
A5	0.22	1.1	11	$y(x,n)=\sum_{i=1}^{n}\left(2\cos(ix)\cdot ch(x)\right)$	По значению
A6	0.36	0.9	6	$y(x,n)=\sum_{i=1}^{n}\left(\sin^2(i)-3e^{ix}\right)$	По указателю

Окончание табл. II

1	2	3	4	5	6
A7	0.34	1.1	8	$y(x,n)=\sum_{i=1}^{n}\left(2\ln(ix)-\sin^{2i}(x)\right)$	По ссылке
A8	0.23	0.9	5	$y(x,n)=\sum_{i=1}^{n}\left(3x^{2i}+4e^{3i}\right)$	По значению
A9	0.55	1.4	15	$y(x,n)=\sum_{i=1}^{n}\left(5\sin(2ix)-\cos^{2}(x)\right)$	По указателю
A10	0.32	0.8	9	$y(x,n)=\sum_{i=1}^{n}\left(15x^{2}-4\cos^{3}(ix)\right)$	По ссылке
A11	0.13	0.7	7	$y(x,n)=\sum_{i=1}^{n}\left(2e^{i\cdot\sin(x)}+3\sqrt{\lvert x\rvert}\right)$	По значению
A12	0.25	0.8	6	$y(x,n)=\sum_{i=1}^{n}\left(e^{2\cos ix}\cdot x^{\cos(i)}\right)$	По указателю
A13	0.44	1.1	9	$y(x,n)=\sum_{i=1}^{n}\left(2tg^{2}(ix)-\sqrt{\lvert x\rvert}\right)$	По ссылке
A14	0.32	1.2	11	$y(x,n)=\sum_{i=1}^{n}\left(4\sqrt[3]{\lvert ix\rvert}+\sin x\right)$	По значению
A15	0.12	1.4	18	$y(x,n)=\sum_{i=1}^{n}\left(\sin(ix)+\cos^{2}(i)\right)$	По указателю

B. *Вывести на экран таблицу значений функции и ее разложения в ряд для x, изменяющегося от a до b с шагом h = (b – a)/10. Расчет y(x) и s(x) поместить в функцию. Использовать прототипы функций. Параметры передавать указанным в табл. III способом. Расчет функции s(x) выполнить с заданной точностью* ε.

Таблица III

№	*a*	*b*	Функция	Разложение функции в ряд Тейлора	ε	Способ передачи параметров
1	2	3	4	5	6	7
1	0.8	1.8	$y(x)=\ln(x)$	$s(x)=-\sum_{n=1}^{\infty}(-1)^{n}\frac{(x-1)^{n}}{n}$	10^{-4}	По ссылке
2	0.1	0.9	$y(x)=ch^{2}(x)$	$s(x)=\sum_{n=1}^{\infty}\frac{2^{2n-1}x^{2n}}{(2n)!}$	10^{-5}	По значению

Окончание табл. III

1	2	3	4	5	6	7
3	1.9	2.9	$y(x)=\frac{1}{1+x}$	$s(x)=\sum_{n=0}^{\infty}(-1)^n x^n$	10^{-6}	По указателю
4	1	3	$y(x)=x\cdot\arctan(x)$	$s(x)=\sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+2}}{1+2n}$	10^{-4}	По ссылке
5	–0.1	1	$y(x)=2^{-x}$	$s(x)=\sum_{n=0}^{\infty}\frac{x^n(-\ln(2))^n}{n!}$	10^{-5}	По значению
6	–0.9	0.9	$y(x)=\cos(x-4)$	$s(x)=\sum_{n=0}^{\infty}(-1)^n \frac{(4-x)^{2n}}{(2n)!}$	10^{-3}	По указателю
7	–0.5	0.5	$y(x)=\cos(\sin(x))$	$s(x)=\sum_{n=0}^{\infty}(-1)^n \frac{\sin^{2n}(x)}{(2n)!}$	10^{-4}	По ссылке
8	–0.3	0.4	$y(x)=e^x+e^{-x}$	$s(x)=\sum_{n=0}^{\infty}\frac{(-x)^n+x^n}{n!}$	10^{-5}	По значению
9	–0.1	1.3	$y(x)=2^x$	$s(x)=\sum_{n=0}^{\infty}\frac{x^n\ln^n(2)}{n!}$	10^{-3}	По указателю
10	–0.5	0.5	$y(x)=e^x$	$s(x)=\sum_{n=0}^{\infty}\frac{x^{2n-1}(2n+x)}{(2n)!}$	10^{-4}	По ссылке
11	0.1	0.8	$y(x)=\ln(1+x^2)$	$s(x)=-\sum_{n=1}^{\infty}(-1)^n \frac{x^{2n}}{n}$	10^{-5}	По значению
12	1	2.5	$y(x)=\sin^2(x)$	$s(x)=-\sum_{n=1}^{\infty}(-1)^n \frac{2^{2n-1}x^{2n}}{(2n)!}$	10^{-3}	По указателю
13	–1.5	1.5	$y(x)=\cos^3(x)$	$s(x)=\frac{1}{4}\sum_{n=0}^{\infty}(-1)^n \frac{(3+9^n)x^{2n}}{(2n)!}$	10^{-4}	По ссылке
14	–0.8	0.9	$y(x)=ch(x^2)$	$s(x)=\sum_{n=0}^{\infty}(-1)^n \frac{x^{4n+2}}{(2n+1)!}$	10^{-5}	По значению
15	–2.5	1.3	$y(x)=\arctan(x)$	$s(x)=\sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{2n+1}$	10^{-3}	По указателю

7. Программирование с использованием строк

А. *Ввести строку с клавиатуры. Выполнить задание, результат вывести на экран.*

А1. Проверить баланс скобок в строке (количество открывающих скобок должно соответствовать количеству закрывающих скобок). Вывести результат проверки.

А2. Подсчитать, какое количество слов в строке начинается с символа ‘w’.

А3. Найти и вывести на экран последовательности, состоящие из трех одинаковых подряд идущих символов.

А4. Вывести на экран второе предложение строки (символы, расположенные между первой и второй точкой).

А5. Подсчитать сумму цифр, встречающихся в строке.

А6. Подсчитать количество слов в строке. Слова отделяются друг от друга одним пробелом. Перед первым словом пробела нет.

А7. Заменить в строке символ ‘-’ на символ ‘*’.

А8. Вывести на экран слова, состоящие из двух символов. Слова отделяются друг от друга одним пробелом. Первый и последний символы строки − пробелы.

А9. Ввести символ. Определить номера слов, которые начинаются с введенного символа. Слова отделяются друг от друга одним пробелом. Перед первым словом пробела нет.

А10. Подсчитать, какое количество букв ‘a’ в первом слове строки. Слова отделяются друг от друга одним пробелом. Перед первым словом пробела нет.

А11. Вывести последнее слово строки. Последний символ строки не пробел.

А12. Вывести количество слов, у которых последний символ ‘g’. Строка заканчивается символом пробела.

А13. Определить, сколько раз в строке встречается последовательность символов “wse”.

А14. Заменить в строке символ ‘-’ на символ ‘*’.

А15. Вывести на экран третье слово строки. Слова отделяются друг от друга одним пробелом. Перед первым словом пробела нет.

В. *Ввести строку с клавиатуры. Выполнить задание, результат вывести на экран.*

В1. Вывести на экран порядковый номер слова максимальной длины и номер позиции в строке, с которой оно начинается. Слова в строке разделены одним или несколькими пробелами.

В2. Удалить из строки предпоследнее слово. Слова в строке разделены одним или несколькими пробелами.

В3. Вывести слова, которые начинаются и заканчиваются одной и той же буквой. Слова в строке разделены одним или несколькими пробелами.

В4. Заменить в строке все слова “Си” на “C++”. Слова в строке разделены одним или несколькими пробелами.

В5. Дана строка, состоящая из нулей и единиц. Вывести на экран группы единиц с максимальным и минимальным количеством символов.

В6. Удалить из строки слова, содержащие символ ’r’. Слова в строке разделены одним или несколькими пробелами.

В7. Дана строка, состоящая из нулей и единиц. Подсчитать количество групп с пятью единицами.

B8. Дана строка символов, состоящая из произвольных десятичных цифр. Числа в строке отделены друг от друга одним или несколькими пробелами. Удалить из строки четные числа.

B9. Заменить в строке все группы подряд идущих пробелов на один пробел.

B10. Дана строка, состоящая из нулей и единиц. Удалить все группы, состоящие из трех нулей.

B11. Вставить слово “Visual” между вторым и третьим словом строки. Слова в строке разделены одним или несколькими пробелами.

B12. Поменять местами первое и второе слово строки. Слова в строке разделены одним или несколькими пробелами.

B13. Удалить из строки слова, содержащие четное количество символов. Слова в строке разделены одним или несколькими пробелами.

B14. Дана строка символов, состоящая из произвольных десятичных цифр. Числа в строке отделены друг от друга одним или несколькими пробелами. Вывести на экран числа этой строки в порядке возрастания их значений.

B15. Дана строка, состоящая из нулей и единиц. Вывести группу с максимальным количеством одинаковых символов.

8. Программирование с использованием структур

A. *Объявить структуру с заданными полями. Ввести необходимый список. Память для хранения списка выделять динамически. Выполнить задание, результат вывести на экран.*

A1. Имеется список студентов. Каждый элемент списка содержит следующую информацию: фамилия, год и место рождения, три экзаменационных оценки за последнюю сессию. Вывести информацию о студентах, имеющих средний балл больше 7.

A2. Имеется список сотрудников предприятия. Каждый элемент списка содержит следующую информацию: фамилию, год рождения и год поступления на работу. Вывести информацию о сотрудниках фирмы, родившихся до 1980 года.

A3. Имеется телефонная база данных. Каждый элемент базы содержит следующую информацию: номер телефона, фамилия и адрес абонента. Вывести на экран фамилии абонентов, номера телефонов которых начинаются на цифру 5.

A4. Имеется список автомобилей. Каждый элемент списка содержит следующую информацию: марка, год выпуска, объем двигателя и максимальная скорость. Вывести информацию об автомобилях, выпущенных после 2000 года и имеющих максимальную скорость больше 180 км/ч.

A5. Имеется список стран мира. Каждый элемент списка содержит следующую информацию: название страны и ее столицы, название части света, в которой находится страна, площадь страны. Вывести информацию о странах, находящихся в Африке.

A6. Имеется расписание движения междугородных автобусов. Каждый элемент расписания содержит следующую информацию: номер рейса, время

отправления, пункт назначения, время прибытия в пункт назначения. Вывести информацию о всех рейсах до города Могилева.

А7. Имеется список книг. Каждый элемент списка содержит следующую информацию: название, фамилия автора, год издания, количество страниц. Вывести все книги, название которых начинается на букву ‘А’.

А8. Имеется список участников спортивных соревнований. Каждый элемент списка содержит следующую информацию: название команды, фамилия спортсмена, возраст, рост и вес. Вывести информацию о спортсменах, рост которых выше 190 см.

А9. У администратора железнодорожных касс хранится информация о свободных местах в поездах. Каждый элемент списка содержит следующую информацию: время отправления, пункт назначения, число свободных мест. Вывести информацию о поездах, следующих в Москву, на которые имеются свободные места.

А10. Имеется список товаров, хранящихся на складе. Каждый элемент списка содержит следующую информацию: наименование, количество, цена. Вывести информацию о товарах, количество которых меньше 10 шт.

А11. В аэропорту имеется список пассажиров, зарегистрировавшихся на рейс. Каждый элемент списка содержит следующую информацию: фамилия, номер билета, вес багажа. Вывести список пассажиров, вес багажа которых превышает 20 кг.

А12. Имеется список участников олимпиады. Каждый элемент списка содержит следующую информацию: название учебного заведения, фамилия, количество набранных очков. Вывести участников, набравших больше 10 баллов.

А13. Имеется список семян овощных культур. Каждый элемент списка содержит следующую информацию: название культуры, номера месяцев посева, высадки рассады и уборки урожая. Вывести информацию о растениях, время посева которых − месяц март.

А14. Имеется список студентов. Каждый элемент списка содержит следующую информацию: фамилия, год и место рождения, три экзаменационных оценки за последнюю сессию. Вывести информацию о студентах, родившихся после 1995 года.

А15. Имеется список автомобилей. Каждый элемент списка содержит следующую информацию: марка, год выпуска, объем двигателя и расход топлива. Вывести информацию об автомобилях с объемом двигателя более 3 литров и расходом топлива менее 10 литров на 100 км.

В. *Объявить структуру с заданными полями. Динамически выделить память для хранения списка. Ввести данные. Выполнить задание, результат вывести на экран.*

В1. Имеется список студентов. Каждый элемент списка содержит следующую информацию: фамилия, год и место рождения, три экзаменационных

оценки за последнюю сессию. Вывести информацию о студентах, проживающих в Минске в порядке убывания среднего балла.

В2. Имеется список сотрудников предприятия. Каждый элемент списка содержит следующую информацию: фамилия, год рождения и год поступления на работу. Вывести информацию о сотрудниках фирмы, родившихся после 1985 года в порядке убывания стажа работы.

В3. Имеется телефонная база данных. Каждый элемент базы содержит следующую информацию: номер телефона, фамилия и адрес абонента. Вывести на экран в алфавитном порядке фамилии абонентов, номера телефонов которых начинаются на цифру 3.

В4. Имеется список автомобилей. Каждый элемент списка содержит следующую информацию: марка, год выпуска, объем двигателя и максимальная скорость. Вывести информацию об автомобилях, выпущенных после 2005 года в порядке убывания их максимальной скорости.

В5. Имеется список стран мира. Каждый элемент списка содержит следующую информацию: название страны, год образования государства, название части света, в которой находится страна и площадь страны. Вывести информацию о странах, находящихся в Европе, в порядке возрастания их площади.

В6. Имеется расписание движения междугородных автобусов. Каждый элемент расписания содержит следующую информацию: номер рейса, время отправления, пункт назначения, время прибытия в пункт назначения. Вывести информацию о рейсах до города Гродно в порядке возрастания времени их отправления.

В7. Имеется список книг. Каждый элемент списка содержит следующую информацию: название, фамилия автора, год издания, количество страниц. Вывести в алфавитном порядке названия книг, изданных до 1990 года.

В8. Имеется список участников спортивных соревнований. Каждый элемент списка содержит следующую информацию: название команды, фамилия спортсмена, его возраст, рост и вес. Вывести в алфавитном порядке фамилии спортсменов, возраст которых младше 18 лет.

В9. У администратора железнодорожных касс хранится информация о свободных местах в поездах. Каждый элемент списка содержит следующую информацию: время отправления, пункт назначения, число свободных мест. Вывести информацию о поездах до Бреста в порядке убывания количества свободных мест.

В10. Имеется список товаров, хранящихся на складе. Каждый элемент списка содержит следующую информацию: наименование, количество, цена. Вывести в алфавитном порядке информацию о товарах, количество которых на складе больше 10 и меньше 100 шт.

В11. В аэропорту имеется список пассажиров, зарегистрировавшихся на рейс. Каждый элемент списка содержит следующую информацию: фамилия, номер билета, вес багажа. Вывести в алфавитном порядке фамилии пассажиров, вес багажа которых не превышает 15 кг.

B12. Имеется список участников олимпиады. Каждый элемент списка содержит следующую информацию: название учебного заведения, фамилия участника, количество набранных очков. Вывести в порядке убывания количество набранных очков фамилии участников из БГУИР.

B13. Имеется список семян овощных культур. Каждый элемент списка содержит следующую информацию: название культуры, номера месяцев посева, высадки рассады и уборки урожая. Вывести в алфавитном порядке названия культур, урожай которых убирается в августе.

B14. Имеется список студентов. Каждый элемент списка содержит следующую информацию: фамилия, год и место рождения, три экзаменационных оценки за последнюю сессию. Вывести в алфавитном порядке фамилии студентов, которые сдали экзамены без двоек.

B15. Имеется список автомобилей. Каждый элемент списка содержит следующую информацию: марка, год выпуска, объем двигателя и расход топлива. Вывести в порядке возрастания расхода топлива информацию об автомобилях, выпущенных после 2004 года.

9. Программирование с использованием файлов

A. *Создать бинарный файл, записать в него десять действительных чисел и закрыть файл. Открыть файл для чтения, прочитать записанные данные и выполнить задание. Результат вывести на экран и в текстовый файл. Закрыть все открытые файлы.*

A1. Найти сумму четных и количество отрицательных чисел.

A2. Найти количество нечетных чисел, стоящих перед положительными числами.

A3. Вывести положительные числа, кратные трем, и все отрицательные.

A4. Выяснить какое из чисел (минимальное или максимальное) находится ближе к началу файла.

A5. Выяснить, расположены ли числа в файле по возрастанию их значений.

A6. Выяснить, каких чисел больше, отрицательных или положительных.

A7. Найти количество чисел, значение которых больше среднего значения всех чисел.

A8. Найти разность между суммой модулей положительных и суммой модулей отрицательных чисел.

A9. Найти количество чисел, находящихся между минимальным и максимальным числами.

A10. Выяснить, имеются ли отрицательные числа, значение которых по модулю больше среднего значения всех чисел.

A11. Вывести четные числа, стоящие после числа с максимальным значением.

A12. Подсчитать сумму чисел, стоящих между максимальным и минимальным числами.

A13. Вывести отрицательные числа, стоящие перед числом с минимальным значением.

A14. Найти среднее значение положительных и среднее значение отрицательных чисел.

A15. Найти число, значение которого наиболее близко к среднему значению всех чисел.

B. *Написать набор функций для выполнения следующих задач: создание бинарного файла; запись данных в файл; открытие файла и чтение из него данных; вывод результата на экран; вывод результата в текстовый файл. Для вызова необходимых функций использовать меню. Задание взять из соответствующего варианта лабораторной работы №8.*

10. Написание рекурсивных программ

A. *Ввести с клавиатуры одномерный массив. Решить задачу путем рекурсивного разбиения массива на две части. Для контроля решить задачу с использованием циклического алгоритма.*

A1. Найти количество отрицательных элементов массива. При рекурсивном разбиении массив делить на две половины.

A2. Найти сумму положительных элементов массива. При рекурсивном разбиении массив делить на первую треть и остальную часть (2/3) массива.

A3. Найти количество четных элементов массива. При рекурсивном разбиении массив делить на две половины.

A4. Найти количество элементов массива, значения которых больше 10 и меньше 20. При рекурсивном разбиении массив делить на первую треть и остальную часть (2/3) массива.

A5. Найти значение минимального элемента массива. При рекурсивном разбиении массив делить на две половины.

A6. Определить, встречаются ли отрицательные элементы в массиве. При рекурсивном разбиении массив делить на первые 2/3 и остальную треть массива.

A7. Определить количество элементов массива, для которых выполняется условие $\sin(a[i]) > 0$. При рекурсивном разбиении массив делить на две половины.

A8. Определить количество элементов массива, для которых выполняется условие $0 < \cos(a[i]) < 0.5$. При рекурсивном разбиении массив делить на первые 2/3 и остальную треть массива.

A9. Найти произведение отрицательных элементов массива. При рекурсивном разбиении массив делить на две половины.

A10. Найти сумму элементов массива, для которых выполняется условие $a[i]^2 > 10$. При рекурсивном разбиении массив делить на первые 2/3 и остальную треть массива.

A11. Определить, встречаются ли четные элементы в массиве. При рекурсивном разбиении массив делить на две половины.

А12. Найти произведение положительных элементов массива. При рекурсивном разбиении массив делить на первую треть и остальную часть (2/3) массива.

А13. Найти номер максимального элемента массива. При рекурсивном разбиении массив делить на первую треть и остальную часть (2/3) массива.

А14. Найти сумму элементов массива, значения которых больше 3 и меньше 10. При рекурсивном разбиении массив делить на первые 2/3 и остальную треть массива.

А15. Найти произведение нечетных элементов массива. При рекурсивном разбиении массив делить на первую треть и остальную часть (2/3) массива.

В. *Решить задачу двумя способами – с применением рекурсии и без нее.*

В1. Написать функцию умножения двух чисел, используя только операцию сложения.

В2. В упорядоченном массиве целых чисел a_i, $i = 0...n–1$ найти номер элемента x методом бинарного поиска: если $x \le a_{n/2}$, тогда $x \in [a_1 \dots a_{n/2}]$, иначе $x \in [a_{n/2+1} \dots a_n]$. Если элемент x отсутствует в массиве, то вывести соответствующее сообщение.

В3. Написать функцию сложения двух чисел, используя только операцию добавления единицы.

В4. Вычислить произведение двух целых положительных чисел $P = a \cdot b$ по следующему алгоритму: если b четное, то $P = 2 \cdot (a \cdot b / 2)$, иначе – $P = a + (a \cdot (b - 1))$. Если $b = 0$, то $P = 0$.

В5. Подсчитать сумму цифр в десятичной записи заданного числа.

В6. Найти значение функции Аккермана *A(m, n)*, которая определяется для всех неотрицательных целых аргументов m и n следующим образом: $A(0, n) = n + 1$, если $m = 0$; $A(m,0) = A(m - 1, 1)$, если $n = 0$; $A(m, n) = A(m - 1, A(m, n - 1))$, если и $m > 0$ и $n > 0$.

В7. Вычислить произведение четного значения ($n \ge 2$) сомножителей $y(n) = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \dots \cdot \frac{n}{n-1} \cdot \frac{n}{n+1}$.

В8. Проверить, является ли заданная строка палиндромом.

В9. Вычислить число сочетаний $C_n^k = \frac{n!}{k!(n-k)!}$ по формуле: $C_n^0 = C_n^n = 1$, $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$ при $n > 1$, $0 < k < n$.

В10. Вычислить $y(n) = \sqrt{1 + \sqrt{2 + \dots + \sqrt{(n-1) + \sqrt{n}}}}$.

В11. Вычислить значение $x = \sqrt{a}$, используя формулу $x_n = \frac{1}{2}(x_{n-1} + a / x_{n-1})$, в качестве начального приближения использовать значение $x_0 = (1 + a)/2$.

B12. Вычислить $y(n,k)=1^k+2^k+...+n^k$.

B13. Вычислить $y(n)=\cfrac{1}{n+\cfrac{1}{(n-1)+\cfrac{1}{(n-2)+\cfrac{1}{.. ..}}}}$

$$....+\cfrac{1}{1+\cfrac{1}{2}}.$$

B14. Подсчитать количество цифр в заданном числе.

B15. Вычислить $y(n)=\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{3+\cfrac{1}{.. ..}}}}$

$$....+\cfrac{1}{(n-1)+\cfrac{1}{n}}.$$

11. Сортировка массивов

A. *Ввести массив из n целых чисел. Отсортировать числа по неубыванию указанным методом. Результат вывести на экран.*

A1. Метод пузырька.
A2. Шейкерная сортировка.
A3. Сортировка выбором.
A4. Сортировка вставкой.
A5. Метод Шелла.
A6. Метод пузырька.
A7. Шейкерная сортировка.
A8. Сортировка выбором.
A9. Сортировка вставкой.
A10. Метод Шелла.
A11. Метод пузырька.
A12. Шейкерная сортировка.
A13. Сортировка выбором.
A14. Сортировка вставкой.
A15. Метод Шелла.

B. *Дополнить программу, написанную при выполнении лабораторной работы №9, функциями упорядочения массива структур по неубыванию заданного ключа. Результат вывести на экран.*

B1. Ключ: год рождения студента. Методы сортировки: QuickSort и сортировка выбором.

B2. Ключ: год поступления на работу сотрудника. Методы сортировки: QuickSort и сортировка вставкой.

B3. Ключ: номер телефона абонента. Методы сортировки: QuickSort и метод Шелла.

B4. Ключ: год выпуска автомобиля. Методы сортировки: QuickSort и сортировка выбором.

B5. Ключ: год образования государства. Методы сортировки: QuickSort и сортировка вставкой.

B6. Ключ: номер рейса автобуса. Методы сортировки: QuickSort и метод Шелла.

B7. Ключ: количество страниц в книге. Методы сортировки: QuickSort и сортировка выбором.

B8. Ключ: рост спортсмена. Методы сортировки: QuickSort и сортировка вставкой.

B9. Ключ: время отправления поезда. Методы сортировки: QuickSort и метод Шелла.

B10. Ключ: цена товара. Методы сортировки: QuickSort и сортировка выбором.

B11. Ключ: вес багажа пассажира. Методы сортировки: QuickSort и сортировка вставкой.

B12. Ключ: количество набранных очков участником олимпиады. Методы сортировки: QuickSort и метод Шелла.

B13. Ключ: номер месяца уборки урожая. Методы сортировки: QuickSort и сортировка выбором.

B14. Ключ: год рождения студента. Методы сортировки: QuickSort и сортировка вставкой.

B15. Ключ: объем двигателя автомобиля. Методы сортировки: QuickSort и метод Шелла.

12. Поиск по ключу в одномерном массиве

A. *Задан отсортированный по неубыванию массив целых чисел. Вывести на экран номер элемента с заданным ключом или информацию о том, что такого элемента в массиве нет. Поиск вести указанным методом.*

A1. Метод поиска: линейный. Ключ: 70.

A2. Метод поиска: двоичный. Ключ: 17.

A3. Метод поиска: линейный с барьером. Ключ: 2.

A4. Метод поиска: двоичный. Ключ: 84.

A5. Метод поиска: линейный. Ключ: 12.

A6. Метод поиска: двоичный. Ключ: 25.

A7. Метод поиска: линейный с барьером. Ключ: 44.

A8. Метод поиска: двоичный. Ключ: 74.

A9. Метод поиска: линейный. Ключ: 41.

A10. Метод поиска: двоичный. Ключ: 7.
A11. Метод поиска: линейный с барьером. Ключ: 28.
A12. Метод поиска: двоичный. Ключ: 82.
A13. Метод поиска: линейный. Ключ: 93.
A14. Метод поиска: двоичный. Ключ: 27.
A15. Метод поиска: линейный с барьером. Ключ: 31.

B. *Дополнить программу, написанную при выполнении лабораторной работы №10 функциями поиска элементов по ключу в массиве структур. Найти элемент с заданным ключом указанным методом поиска (для упрощения предполагается, что в массиве присутствует не более одного такого элемента). Если элемент не найден, то вывести соответствующее сообщение.*

B1. Вывести на экран фамилию студента, родившегося в 1980 году. Методы поиска: линейный с барьером и двоичный.

B2. Вывести на экран фамилию сотрудника, который был принят на работу в 1999 году. Метод поиска: интерполяционный.

B3. Вывести на экран фамилию абонента, на которого зарегистрирован номер телефона 7972474. Методы поиска: линейный и двоичный.

B4. Вывести на экран максимальную скорость автомобиля, выпущенного в 1996 году. Методы поиска: линейный с барьером и двоичный.

B5. Вывести на экран название государства, образованного в 1927 году. Метод поиска: интерполяционный.

B6. Вывести на экран пункт назначения автобуса с номером рейса 295. Методы поиска: линейный с барьером и двоичный.

B7. Вывести на экран название книги, в которой 1575 страниц. Методы поиска: линейный и двоичный.

B8. Вывести на экран фамилию спортсмена, у которого рост равен 197 см. Метод поиска: интерполяционный.

B9. Вывести на экран пункт назначения поезда, который отправляется 11 часов. Методы поиска: линейный с барьером и двоичный.

B10. Вывести на экран наименование товара с ценой, равной 265 000 руб. Методы поиска: линейный и двоичный.

B11. Вывести на экран фамилию пассажира, у которого багаж весит 58 кг. Метод поиска: интерполяционный.

B12. Вывести на экран фамилию участника олимпиады, который набрал 212 очков. Методы поиска: линейный с барьером и двоичный.

B13. Вывести на экран название культуры, которую убирают в июне (шестом месяце года). Методы поиска: линейный и двоичный.

B14. Вывести на экран средний балл, набранный на экзамене студентом, родившимся в 1991 году. Методы поиска: линейный с барьером и двоичный.

B15. Вывести на экран марку автомобиля с объемом двигателя 1998 $см^3$. Метод поиска: интерполяционный.

13. Работа со стеками

А. *Создать стек, состоящий из n целых чисел. Выполнить задание, результат вывести на экран. В конце работы освободить всю динамически выделенную память.*

А1. Найти минимальный элемент стека.

А2. Выяснить, имеются ли в стеке отрицательные числа.

А3. Найти разность суммы четных и суммы нечетных элементов стека.

А4. Найти произведение нечетных элементов стека.

А5. Найти номер второго (от вершины) нечетного элемента стека.

А6. Найти среднее значение всех элементов стека.

А7. Найти произведение трех первых положительных элементов стека.

А8. Найти разность первого и последнего элементов стека.

А9. Найти сумму трех последних элементов стека.

А10. Найти количество отрицательных элементов стека.

А11. Найти сумму трех первых и произведение остальных элементов стека.

А12. Выяснить, имеются ли в стеке числа, большие 250.

А13. Выяснить, каких элементов в стеке больше отрицательных или положительных.

А14. Найти максимальный элемент стека.

А15. Найти сумму положительных элементов стека.

В. *Создать стек, состоящий из n целых чисел. Выполнить задание. Информационную часть в оперативной памяти не перемещать. Результат вывести на экран. В конце работы освободить всю динамически выделенную память.*

В1. Удалить из стека все нечетные числа.

В2. Поменять местами минимальный и максимальный элементы стека.

В3. Преобразовать стек таким образом, чтобы порядок следования элементов был изменен на обратный.

В4. Поменять местами второй и предпоследний элементы стека.

В5. Добавить элемент со значением 88 перед каждым отрицательным элементом.

В6. Добавить элемент со значением 77 перед предпоследним элементом стека.

В7. Удалить каждый третий (по порядку) элемент стека.

В8. Найти среднее значение всех элементов стека. Удалить из стека все элементы, значение которых меньше среднего значения.

В9. Удалить каждый третий элемент стека.

В10. Удалить из стека все отрицательные числа.

В11. Удалить все элементы стека, расположенные перед минимальным элементом стека.

В12. Удалить все элементы, расположенные между первым и последним отрицательными элементами стека.

В13. Добавить элемент со значением 33 после максимального элемента стека.

В14. Поменять местами первый положительный и предпоследний отрицательный элементы стека.

В15. Удалить из стека все элементы, значения которых находятся в диапазоне от 0 до 10.

14. Работа с двусвязанными списками

А. *Создать двусвязанный список, состоящий из n целых чисел. Выполнить задание. Информационную часть в оперативной памяти не перемещать. Результат вывести на экран. В конце работы освободить всю динамически выделенную память.*

А1. Удалить минимальный элемент очереди.

А2. Добавить между двумя подряд идущими отрицательными элементами очереди элемент со значением 99.

А3. Добавить элемент со значением 55 после каждого отрицательного элемента очереди.

А4. Поменять местами первый и последний элементы очереди.

А5. Удалить все элементы, стоящие перед первым отрицательным элементом.

А6. Удалить отрицательные элементы очереди.

А7. Поменять местами последний и максимальный элементы очереди.

А8. Добавить после каждого нечетного элемента очереди элемент со значением 0.

А9. Удалить второй и предпоследний элементы очереди.

А10. Удалить все четные элементы очереди.

А11. Добавить элемент со значением 77 после первого и перед последним элементами очереди.

А12. Удалить четные элементы очереди.

А13. Поменять местами первый и минимальный элементы очереди.

А14. Удалить все элементы, стоящие после минимального элемента очереди.

А15. Удалить максимальный элемент очереди.

В. *Выполнить задание в соответствии с вариантом. Информационную часть в оперативной памяти не перемещать. Результат вывести на экран. В конце работы освободить всю динамически выделенную память.*

В1. Создать двусвязанный список, состоящий из *n* целых чисел. Извлечь из первого списка и переместить во второй список все отрицательные числа.

В2. Создать двусвязанный список, состоящий из *n* целых чисел. Удалить из списка все элементы, находящиеся между его максимальным и минимальным элементами.

В3. Создать двусвязанный список, состоящий из *n* целых чисел. Переместить во второй список элементы, повторяющиеся в первом списке более одного раза.

В4. Создать двусвязанный список, состоящий из *n* целых чисел. Преобразовать его в два списка: первый список должен содержать только четные числа, второй − нечетные.

В5. Создать двусвязанный список, состоящий из *n* действительных чисел. Расположить элементы списка в обратном порядке.

В6. Создать два двусвязанных списка, состоящих из *n* целых чисел, упорядоченных по неубыванию. Переместить все данные в третий список, удаляя повторяющиеся значения.

В7. Создать двусвязанный список, состоящий из *n* целых чисел. Переместить во второй список элементы, находящиеся между минимальным и максимальным элементами первого списка.

В8. Создать два двусвязанных списка, состоящих из *n* целых чисел упорядоченных по неубыванию. Переместить в третий список элементы со значениями, которые встречаются и в первом и во втором списках.

В9. Создать двусвязанный список, состоящий из *n* целых чисел. Отрицательные элементы удалить, а четные перенести во второй список.

В10. Создать двусвязанный список, состоящий из *n* символов латинского алфавита и символов арифметических операций. Переместить символы арифметических операций во второй список.

В11. Создать два двусвязанных списка, состоящих из *n* символов латинского алфавита. Переместить все данные в третий список таким образом, чтобы строчные символы находились в левой половине списка, а прописные − в правой.

В12. Создать двусвязанный список, состоящий из *n* целых чисел. Переместить во второй список элементы, значения которых больше среднего значения элементов первого списка.

В13. Создать двусвязанный список, состоящий из *n* символов латинского алфавита. Удалить из списка элементы с повторяющимися более одного раза значениями.

В14. Создать два двусвязанных списка, состоящих из *n* целых чисел, упорядоченных по неубыванию. Преобразовать их в третий список, который будет упорядочен по невозрастанию.

В15. Создать двусвязанный список, состоящий из *n* символов латинского алфавита. Преобразовать его в два списка: первый список должен содержать прописные символы, второй − строчные.

15. Работа с древовидными структурами данных

А. *Создать сбалансированное дерево поиска, состоящее из целых чисел. Вывести информацию на экран, используя прямой, обратный и симметричный обход дерева. Выполнить задание, результат вывести на экран. В конце работы освободить всю динамически выделенную память.*

А1. Найти количество листьев дерева.

А2. Найти количество узлов, имеющих только одного потомка слева.

А3. Вывести значения узлов, являющихся листами дерева.

А4. Найти количество листов в правом поддереве.

А5. Определить степень дерева.

А6. Определить количество узлов дерева, имеющих глубину, равную 3.

А7. Найти количество узлов, имеющих одного потомка.

А8. Найти сумму значений внутренних узлов дерева.

А9. Вывести значения узлов, имеющих только одного потомка справа.

А10. Найти максимальный (по значению) элемент в левом поддереве.

А11. Найти количество узлов в левом поддереве.

А12. Найти минимальный (по значению) элемент в правом поддереве.

А13. Найти сумму значений узлов, степень которых больше 2.

А14. Найти количество внутренних узлов дерева.

А15. Определить глубину дерева.

В. Создать сбалансированное дерево поиска, состоящее из целых чисел. Вывести информацию на экран, используя прямой, обратный и симметричный обход дерева. Выполнить задание, результат вывести на экран. В конце работы освободить всю динамически выделенную память.

В1. Поменять местами узлы с минимальным и максимальным ключами в левом поддереве.

В2. Поменять местами узел с максимальным ключом и узел, являющийся корнем дерева.

В3. Найти количество листьев на каждом уровне дерева.

В4. Удалить в дереве все узлы, имеющие отрицательные ключи.

В5. Поменять местами узлы с минимальным и максимальным ключами.

В6. Удалить в дереве все узлы, имеющие только одного потомка слева.

В7. Удалить все узлы дерева, имеющие значение ключа больше 5.

В8. Удалить из левой ветви дерева узел с максимальным значением ключа и всех его потомков.

В9. Удалить все узлы дерева, имеющие значение ключа, равное 25.

В10. Найти узел, имеющий значение, ближайшее к среднему значению всех ключей дерева.

В11. Удалить из правой ветви дерева узел с минимальным значением ключа и всех его потомков.

В12. Удалить в дереве все узлы, имеющие только одного потомка справа.

В13. Удалить в дереве все узлы, имеющие четные ключи.

В14. Удалить из дерева ветвь, с вершиной, имеющей заданный ключ.

В15. Удалить из дерева узел с заданным ключом.

16. Вычисление алгебраических выражений

А. *Ввести заданное арифметическое выражение и необходимые данные. Преобразовать запись арифметического выражения в форму обратной польской записи. Вычислить арифметическое выражение. Результат вывести на экран. Задание выбрать в соответствии с номером варианта.*

A1. $a \cdot b \cdot (c - d) + f$.

A2. $a - b / c \cdot d / f$.

A3. $a + b \cdot (c - d) / f$.

A4. $a \cdot b - c \cdot (d + f)$.

A5. $(a - b) \cdot (c - d) \cdot f$.

A6. $(a + b) / (c - d) \cdot f$.

A7. $(a + b + c) \cdot d - f$.

A8. $a \cdot (b / c - d / f)$.

A9. $a \cdot (b + c - d) / f$.

A10. $a - b \cdot (c - d + f)$.

A11. $a / (b \cdot c - d) + f$.

A12. $(a - b) / (c - d) \cdot f$.

A13. $a / b + c / d - f$.

A14. $(a + b - c) \cdot d / f$.

A15. $a \cdot b / (c - d + f$.

B. *Ввести заданное арифметическое выражение и необходимые данные. Преобразовать запись арифметического выражения в форму обратной польской записи (для обозначения операции возведения в степень использовать знак ^). Вычислить арифметическое выражение. Результат вывести на экран. Задание выбрать в соответствии с номером варианта.*

B1. $(x - y)^w \cdot \dfrac{c + k}{f - k}$.

B2. $\dfrac{f + s}{f - y} + \dfrac{y + s^w}{x - s}$.

B3. $a \cdot x^w - \dfrac{b + x}{y}$.

B4. $x \cdot \dfrac{y + a}{y + b^w} - c$.

B5. $\dfrac{x^w}{x - y} \cdot s + b^2$.

B6. $\dfrac{c + k \cdot s}{f - k \cdot s^w} + a$.

B7. $b - s \cdot \dfrac{x}{x^w + y^w}$.

B8. $\dfrac{c^w \cdot d^w}{k^w \cdot (k+c)}$.

B9. $x^w - y^w + \dfrac{a+y}{a-x}$.

B10. $x - y \cdot \dfrac{(x+y)^w}{x+k}$.

B11. $(a-b)^w \cdot \dfrac{x^w}{x^w + y}$.

B12. $\dfrac{x-c}{c+y^w} \cdot x - y$.

B13. $\dfrac{x}{f-k^w} + xy - c$.

B14. $ax^w + (cy - a) \cdot y$.

B15. $a^w \cdot \dfrac{x+k^w}{y-k} + s$.

17. Программирование с использованием хеширования

A. *Ввести массив из n целых чисел из заданного диапазона. Создать хеш-таблицу из M элементов. Осуществить поиск элемента в хеш-таблице. Вывести на экран исходный массив, хеш-таблицу и результат поиска. Задание выбрать в соответствии с номером варианта в табл. IV.*

Таблица IV

Номер варианта	*n*	Диапазон значений	*M*	Схема хеширования
A1	12	23000−45000	15	С линейной адресацией
A2	8	53000−78000	10	На основе связанных списков
A3	15	12000−34000	20	С линейной адресацией
A4	9	11000−53000	10	На основе связанных списков
A5	16	45000−76000	20	С линейной адресацией
A6	12	24000−54000	10	На основе связанных списков
A7	8	32000−68000	10	С линейной адресацией
A8	14	26000−77000	10	На основе связанных списков
A9	9	38000−58000	15	С линейной адресацией
A10	11	24000−79000	10	На основе связанных списков
A11	12	27000−58000	15	С линейной адресацией
A12	7	47000−89000	10	На основе связанных списков
A13	11	44000−73000	15	С линейной адресацией
A14	9	39000−76000	10	На основе связанных списков
A15	12	23000−58000	15	С линейной адресацией

В. *Объявить и ввести массив структур из n элементов. Создать хеш-таблицу из M элементов. Осуществить поиск элемента по ключу в хеш-таблице. Вывести на экран исходный массив, хеш-таблицу и все поля найденной структуры. Задание выбрать в соответствии с номером варианта в табл. V.*

Таблица V

Номер варианта	*n*	Поля структуры	Ключевое поле	*M*	Схема хеширования
B1	6	фамилия, номер группы, оценка	Оценка	15	С квадратичной адресацией
B2	8	марка автомобиля, максимальная скорость, год выпуска	Год выпуска	10	С произвольной адресацией
B3	7	фамилия, номер группы, год рождения	Год рождения	20	С двойным хешированием
A4	9	фамилия, номер телефона, адрес	Номер телефона	10	На основе связанных списков
B5	9	название книги, количество страниц, год издания	Количество страниц	20	С квадратичной адресацией
B6	7	наименование товара, цена, день выпуска	Цена	10	С произвольной адресацией
B7	8	пункт назначения, номер рейса, время отправления	Номер рейса	10	С двойным хешированием
B8	6	фамилия, вес, рост	Вес	10	На основе связанных списков
B9	9	фамилия, количество очков, занятое место	Количество очков	15	С квадратичной адресацией
B10	8	фамилия, вес, рост	Рост	10	С произвольной адресацией
B11	7	фамилия, количество очков, занятое место	Занятое место	15	С двойным хешированием
B12	7	пункт назначения, номер рейса, время отправления	Время отправления	10	На основе связанных списков
B13	8	наименование товара, цена, день выпуска	День выпуска	15	С квадратичной адресацией
B14	9	название книги, количество страниц, год издания	Год издания	10	С произвольной адресацией
B15	8	марка автомобиля, максимальная скорость, год выпуска	Максимальная скорость	15	С двойным хешированием

ПРИЛОЖЕНИЯ

1. Консольный режим работы среды Visual C++ 6.0

Программа, создаваемая в среде Visual C++, всегда оформляется в виде отдельного проекта. Проект (*project*) – набор взаимосвязанных исходных файлов, предназначенных для решения определенной задачи, компиляция и компоновка которых позволяет получить выполняемую программу. В проект входят как файлы, непосредственно создаваемые программистом, так и файлы, которые автоматически создает и редактирует среда программирования.

Для **создания нового проекта** необходимо:

– выбрать **File – New**;

– в открывшемся окне на закладке **Projects** выбрать тип проекта **Win32 Console Application**;

– в поле **Project Name** ввести имя проекта, например *maylab1*;

– в поле **Location** ввести имя каталога, в котором будет размещен проект и полный путь к нему, например *D:\WORK\mylab1*. Каталог также можно выбрать, используя диалоговое окно **Choose Directory**, для чего надо щелкнуть мышью по кнопке **…** ;

– указать тип создаваемого проекта – **Win32 Console Application**;

– щелкнуть мышью по кнопке **ОК;**

– в открывшемся окне мастера приложений **Win32 Console Application – Step 1 of 1** выбрать **An empty project** (пустой проект) и щелкнуть по кнопке **Finish;**

– в открывшемся окне **New Project Information** (информация о новом проекте) щелкнуть мышью по кнопке **ОК.**

Для работы с консольным приложением необходимо создать новый или добавить существующий файл с текстом программы.

Для **создания нового файла** необходимо:

– выбрать **File – New**;

– в открывшимся окне на закладке **Files** выбрать тип файла **C++ Source File**;

– в поле **File name:** ввести имя файла. Для удобства желательно ввести имя, совпадающее с именем проекта, например *maylab1*;

– щелкнуть мышью по кнопке **ОК.**

Для **добавления в проект уже существующего файла** с текстом программы необходимо:

– скопировать имеющийся файл (расширение **cpp**) в рабочую папку проекта;

– в окне **Workspace**, закладка **FileView**, щелкнуть правой кнопкой мыши по папке **Source Files**;

– в открывшемся диалоговом окне **Insert Files**... выбрать добавляемый файл и щелкнуть мышью по кнопке **ОК.**

В папке проекта, как правило, размещено пять файлов и одна вложенная папка. Файлы имеют следующее назначение.

Файл с расширением **dsw** (например *mylab1.dsw*) – файл проекта, который объединяет все входящие в проект файлы.

Файл с расширением **dsp** (например *mylab1.dsp*) предназначен для построения отдельного проекта или подпроекта.

Файл с расширением **opt** (например *mylab1.opt*) содержит все настройки данного проекта.

Файл с расширением **ncb** (например *mylab1.ncb*) – служебный файл.

Файл с расширением **cpp** (например *mylab1.cpp*) – файл текста программы.

2. Выполнение программы

Для компиляции, компоновки и запуска программы на выполнение используются следующие пункты подменю Build:

Compile (Ctrl+F7) – компиляция выбранного файла. Результаты компиляции выводятся в окно Output.

Build (F7) – компоновка проекта. Компилируются все файлы, в которых произошли изменения с момента последней компоновки. Если компоновка прошла без ошибок, то среда программирования создаст исполняемый файл с расширением **exe**, который можно будет запустить на выполнение.

Rebuild All – перекомпоновка проекта. Компилируются все файлы проекта независимо от того, были ли в них произведены изменения или нет.

Execute (**Ctrl+F5**) – выполнение исполняемого файла, созданного в результате компоновки проекта. Для файлов, в которые были внесены изменения, выполняется перекомпилирование и перекомпоновка.

Если в процессе компиляции были обнаружены синтаксические ошибки, то выводится соответствующее сообщение. В этом случае необходимо последовательно исправлять ошибки и компилировать проект снова.

После окончания работы проект можно закрыть, выбрав **File** – **Close Workspace**, или закрыть приложение MVC++.

Для открытия сохраненного ранее проекта необходимо выбрать **File** – **Open Workspace**... В открывшемся диалоговом окне выбрать папку проекта и открыть в ней файл с расширением **dsw**.

3. Отладка программы

Если синтаксических ошибок в программе нет, но результат выполнения программы неверный, необходимо искать логические ошибки. Для поиска логических ошибок используется встроенный отладчик.

Для пошагового выполнения программы необходимо нажимать клавишу **F10**. При каждом нажатии выполняется текущая строка. Если необходимо пошагово проверить текст вызываемой функции, то следует нажать **F11**. Для досрочного выхода из функции нажать **Shift+F11**. Если необходимо начать отлад-

ку с определенного места программы, то надо установить курсор в соответствующую строку программы и нажать **Ctrl+F10**.

Другим способом отладки является установка *точек прерывания* программы. Для этого надо поместить курсор в нужную строку и нажать **F9**. Точка прерывания обозначается красным кружком на специальном поле, расположенном слева от окна текста программы. Для удаления точки прерывания следует в необходимой строке повторно нажать **F9.** Количество точек прерывания в программе может быть любым.

Для выполнения программы до точки прерывания необходимо нажать **F5**. Для продолжения отладки нажимается клавиша **F5** (для выполнения программы до следующей точки прерывания) или используются клавиши для пошаговой отладки.

Желтая стрелка на поле слева от окна текста программы указывает на строку, которая будет выполнена на следующем шаге отладки.

Для контроля за значениями переменных удобно использовать следующий способ: подвести указатель мыши к интересующей переменной и задержать его на несколько секунд. На экране рядом с именем переменной появится окно, содержащее текущее значение этой переменной. Кроме этого, значения переменных будут отображаться в окнах, расположенных снизу. В левом нижнем окне отображаются значения последних использованных программой переменных. В правом нижнем окне (Watch) можно задать имена переменных, значения которых необходимо контролировать.

Литература

1. Основы алгоритмизации и программирования. Язык Си : учеб. пособие / М. П. Батура [и др.]. – Минск : БГУИР, 2007. – 240 с.

2. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт. – СПб. : Невский диалект, 2005. − 352 с.

3. Кнут, Д. Э. Искусство программирования. В 3 т. Т 3: Сортировка и поиск / Д. Э. Кнут. – М . : Вильямс, 2011. − 824 с.

4. Лафоре, Р. Объектно-ориентированное программирование в C++ / Р. Лафоре. − СПб. : Питер, 2007. − 928 с.

5. Керниган, Б. Язык программирования Си / Б. Керниган, Д. Ритчи. – СПб. : Невский диалект, 2003. − 335 с.

6. Страуструп, Б. Язык программирования C++ / Б. Страуструп. – СПб. : БИНОМ, 2008. − 1104 с.

7. Хопкрофт, Дж. Структуры данных и алгоритмы / Дж. Хопкрофт, Дж. Ульман, А. Ахо. – М. : Вильямс, 2003. − 382 с.

Св. план 2013, поз. 26

Учебное издание

Навроцкий Анатолий Александрович

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ В СРЕДЕ VISUAL C++

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редакторы *И. В. Ничипор, М. А. Зайцева*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *А. А. Лысеня*

Подписано в печать 10.06.2014. Формат 68x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 9,42. Уч.-изд. л. 8,0. Тираж 150 экз. Заказ 199.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
2200013, Минск, П. Бровки, 6