Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение
высшего профессионального образования
«Магнитогорский государственный технический
университет им. Г.И. Носова»

В.Е. Торчинский
А.Н. Калитаев
В.Д. Тутарова

Практикум по программированию

Учебное пособие

Магнитогорск
2013

УДК 684.438(075)
ББК 32.973-01я7
Т 619

Рецензенты

Заведующий кафедрой прикладной информатики
и управляющих систем автоматики,
кандидат технических наук, доцент
Новотроицкий филиал ФГАОУ ВПО «НИТУ МИСиС»
С.Н. Басков

Доцент кафедры математические методы в экономике
кандидат экономических наук, доцент
ФГБОУ ВПО «МГТУ»
Т.А. Иванова

В.Е. Торчинский, А.Н. Калитаев, В.Д. Тутарова
Практикум по программированию [Электронный ресурс]: Учебное пособие / В.Е. Торчинский, А.Н. Калитаев, В.Д. Тутарова; ФГБОУ ВПО «МГТУ». – Электрон. текстовые дан. (1,87 Мб). – Магнитогорск: ФГБОУ ВПО «МГТУ», 2013. – 1 электрон. опт. диск (CD-R). – Систем. требования: IBM PC, любой, более 1 GHz; 512 Мб RAM; 10 Мб HDD; MS Windows XP и выше; Adobe Reader; CD/DVD-ROM дисковод; мышь. – Загл. с контейнера.

Пособие позволит приобрести представление об алгоритмах и конечном наборе базовых управляющих структур языка С++, представление о структуре программных средств, знания основ кодирования на языке высокого уровня С++.

В учебном пособии помимо «стандартных» тем, таких как объявление переменных, функции и др., подробно рассматривается работа с векторами и строками, динамические многомерные массивы, указатели, линейные односвязные списки. Примеры и задания для самостоятельной работы, содержащиеся в каждой главе, помогут читателю закрепить изученный теоретический материал.

Учебное пособие рассчитано на студентов направления 230100 – «Информатика и вычислительная техника» и начинающих программистов, которые хотят изучить тонкости программирования на языке С++.

УДК 684.438(075)
ББК 32.973-01я7

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В середине 50-х годов, когда вычислительная техника прочно укоренилась в университетах и научно-исследовательских лабораториях США и Европы, наступило время стремительного прогресса в области программирования. С появлением языков высокого уровня программисты получили возможность больше времени уделять решению конкретной проблемы, не отвлекаясь особенно на весьма тонкие вопросы организации самого процесса выполнения задания на машине. Кроме того, появление этих языков ознаменовало первый шаг на пути создания программ, которые вышли за пределы научно-исследовательских лабораторий и финансовых отделов.

Языки программирования – это тщательно и изобретательно составленные последовательности слов, букв, чисел и мнемонических сокращений, которыми люди пользуются для общения с компьютерами с целью выполнения задач. Каждый язык имеет свою грамматику и синтаксис. Больше чем за шестьдесят лет в мире программирования насчитывается несколько сотен таких языков. Языки программирования служат разнообразным целям – от решения сложных математических задач до создания музыкальной партитуры и машинной графики. Выбор языка программирования обычно определяется одним или, возможно, несколькими из трех факторов: язык должен быть удобен для программиста, пригоден для данного компьютера и для решения данной задачи.

Практикум по программированию на языке С++ содержит лабораторные работы, сгруппированные по темам для быстрого освоения данного языка, в которых содержится большое количество подробно разобранных задач и сопровождается заданиями для самостоятельной работы.

Кроме того, он может быть полезен для студентов, аспирантов и преподавателей.

ЛАБОРАТОРНАЯ РАБОТА № 1.
ЗНАКОМСТВО С КОМПИЛЯТОРОМ C++. СОЗДАНИЕ КОНСОЛЬНЫХ ПРОЕКТОВ И МЕТОДОВ ОТЛАДКИ ПРОГРАММ

Вызов компилятора C++ осуществляется или через меню Пуск→Программы→Borland Developer Studio 2006→Turbo C++, или щелчком мышью по пиктограмме с соответствующим именем, если она размещена на рабочем столе. После запуска Turbo C++ появляется главное окно программы, представленное на рис. 1.1. Рабочий стол Borland Developer Studio 2006→Turbo C++ включает в себя четыре окна:

Окно *Project Workspace 1* (окно рабочей области) предназначено для оказания помощи при написании и сопровождении больших многофайловых программ. Одна из вкладок этого окна содержит список файлов проекта.

Окно *Editor 2* (окно редактора) используется для ввода и проверки исходного кода.

Окно *Output 3* (окно вывода) служит для вывода сообщений о ходе компиляции, сборки и выполнения программы. В частности, сообщения о возникающих ошибках появляются именно в этом окне.

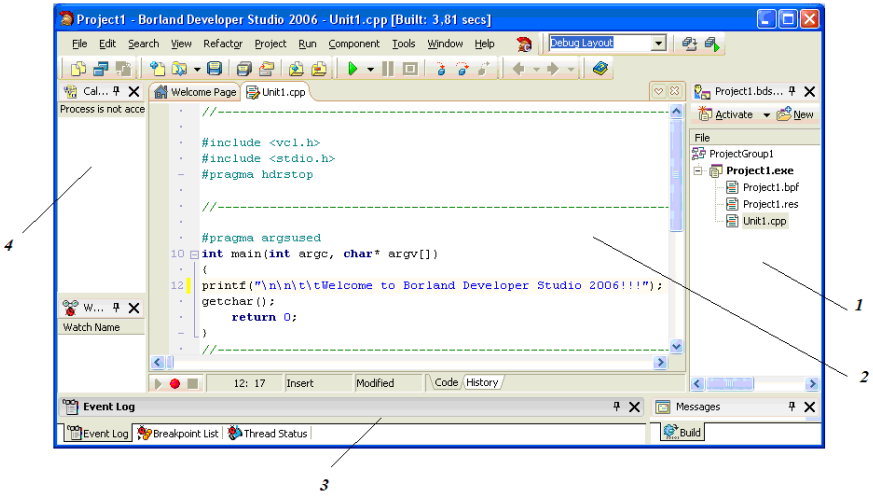


Рис. 1.1. Главное окно Borland Developer Studio 2006→Turbo C++

Под заголовком главного окна, как и во всех Windows-приложениях, находится строка меню. Для кнопок панелей инструментов предусмотрена удобная контекстная помощь (всплывает

подсказка с назначением кнопки). Borland Developer Studio 2006 позволяет строить проекты разных типов, ориентированных на различные сферы применения. Так как этот пакет спроектирован на Windows-платформе, то почти все типы проектов являются оконными Windows-приложениями с соответствующим графическим интерфейсом. В то же время разработчики предусмотрели работу и с так называемыми консольными приложениями.

Для создания нового проекта в режиме консоли выполняем команду File->New->Other… (рис. 1.2).

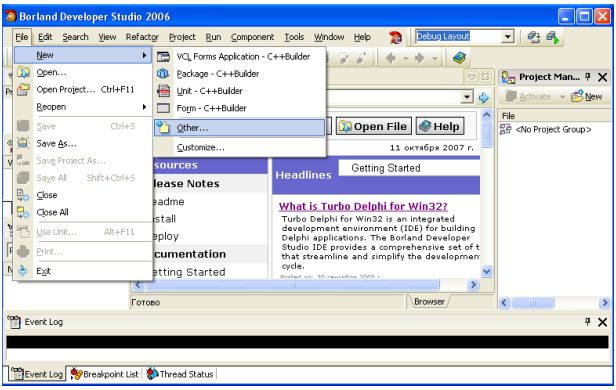


Рис. 1.2

В открывшемся диалоговом окне New выберите тип Console Application (рис. 1.3), а затем в окне New Console Application отредактируйте параметры работы с консольным приложением, как рекомендуется на рис. 1.4.

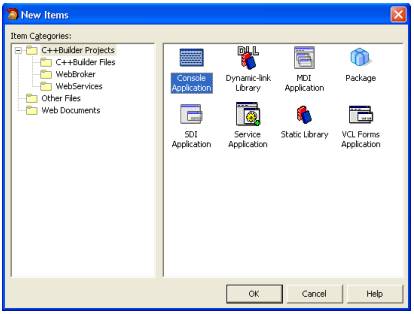


Рис. 1.3

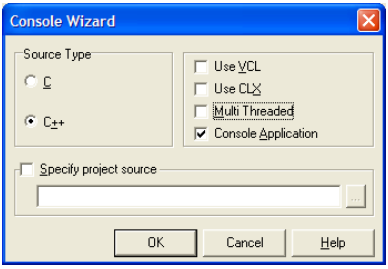


Рис. 1.4

При запуске консольного приложения операционная система создает консольное окно, через которое идет весь ввод-вывод программы. Внешне это напоминает работу в операционной среде

MS-DOS в режиме командной строки. Этот тип приложений удобен при изучении языка C/C++.

До начала написания программного кода с целью дальнейшего предотвращения потерь данных проекту необходимо присвоить имя, для сохранения проекта выполняем команду File->Save All (рис. 1.5).

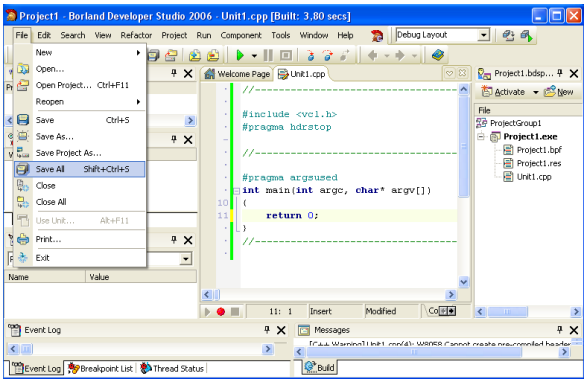


Рис. 1.5. Сохранение проекта (этап 1)

Затем выбираем папку, предварительно созданную для хранения разработанных программ, и в строке Имя файла последовательно вводим имя модуля и имя проекта (рис. 1.6), заканчивая каждую операцию нажатием кнопки «Сохранить».

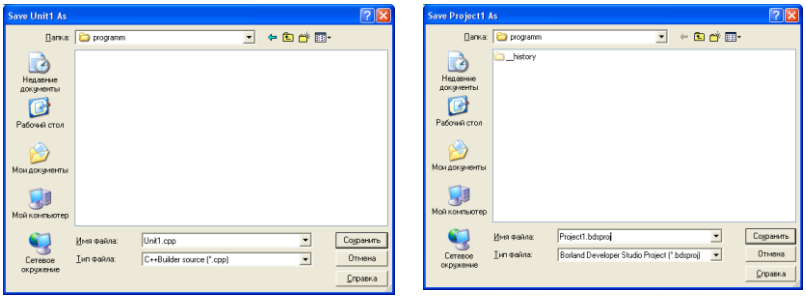


Рис. 1.6. Сохранение проекта (этап 2)

Операция открытия проекта производится выполнением команды Open Project (рис. 1.7).

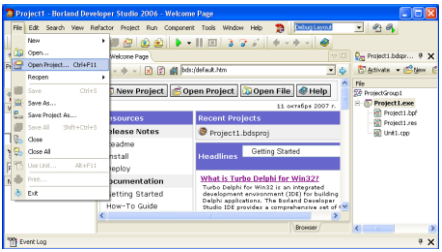


Рис. 1.7. Открытие проекта (этап 1)

Затем необходимо выбрать папку, в которой хранится проект и сам файл проекта (рис. 1.8) и нажать кнопку «Открыть».

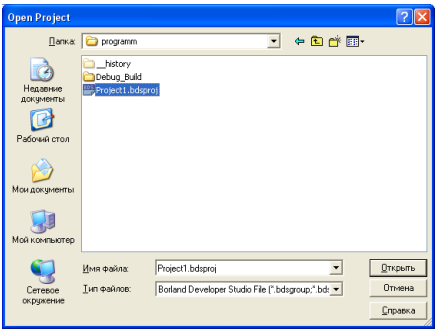


Рис. 1.8. Открытие проекта (этап 2)

В результате работы над программой часто возникает необходимость просмотреть промежуточные результаты вычисления. Это достигается с помощью окна отладки *Watch*. Рассмотрим работу программы на примере нахождения суммы и разности двух чисел, код которой приведен ниже.

```
int main()
{
int a=5, b=2, c, d;
c=a+b;
d=a-b;
return 0;
}
```

Для вызова окна *Watch* выполним команду *Run->Add Watch*. В верхней строке *Expression* вводим имена переменных, работу которых необходимо проконтролировать (рис. 1.9).

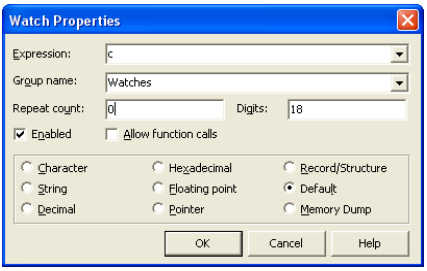


Рис. 1.9

В этом случае помимо экранной подсказки, переменные отображается в окне *Watch List*, расположенном в левой части окна (рис. 1.10). В этом окне приведены значения переменных, при этом выбор переменных для просмотра определяет сам создатель программы.

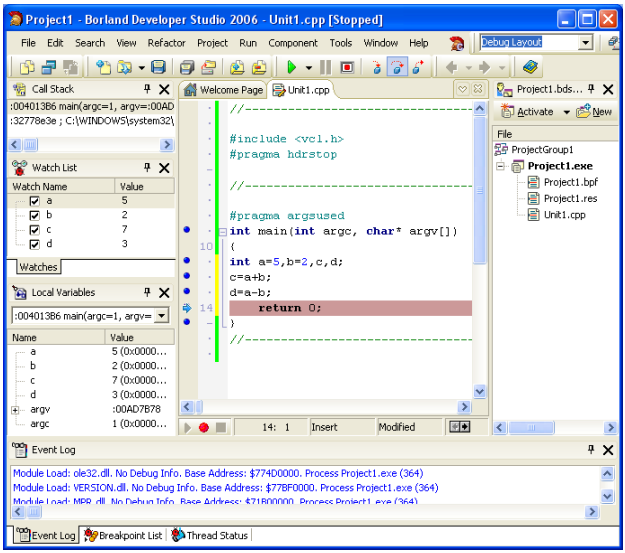


Рис. 1.10.

Точка прерывания позволяет остановить выполнение программы перед любой выполняемой инструкцией (оператором) с тем, чтобы продолжать выполнение программы либо в пошаговом режиме, либо в непрерывном режиме до следующей точки прерывания. Точка прерывания задается щелчком мыши перед некоторым оператором и обозначается в виде красного кружочка на левом поле окна редактирования. Повторный щелчок на указанной метке снимает точку прерывания. В программе может быть не-

сколько точек прерывания. Результаты работы программы с использованием окна Watch приведены на рис. 1.11.

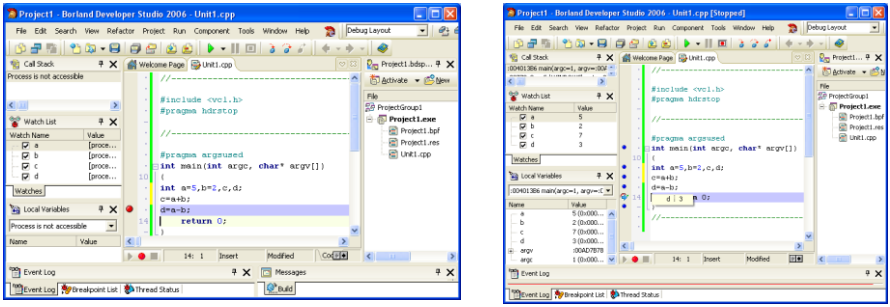


Рис. 1.11

На рис. 1.12 представлен исходный код программы, выполняющей вывод текстовой информации *(«Welcome to Borland Developer Studio 2006!!!»)* на экран консольного приложения, при этом используем функцию вывода *printf*.

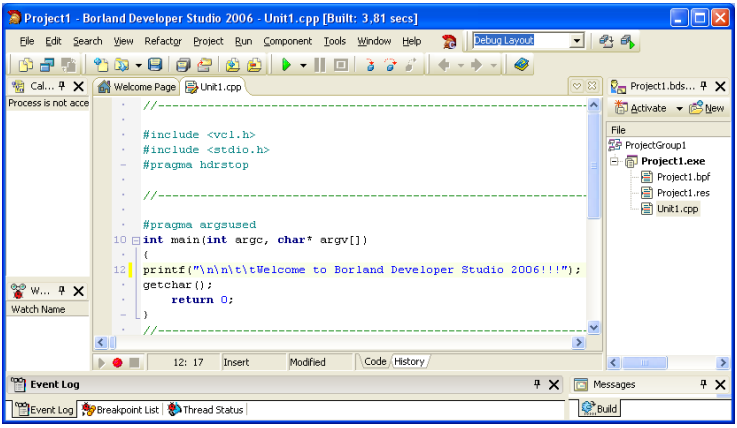


Рис. 1.12

Для компиляции, компоновки и запуска программы необходимо выполнить команду *Run->Run* или нажать клавишу *F9*.

Результат работы программы приведен на рис. 1.13.

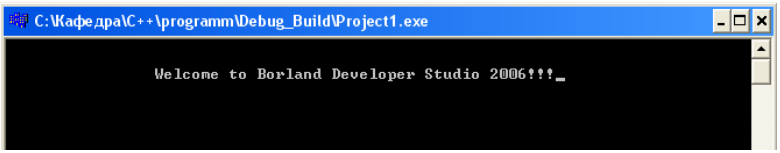


Рис. 1.13

ЛАБОРАТОРНАЯ РАБОТА № 2.
ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ

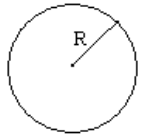
Если в программе все операторы выполняются последовательно, один за другим такая программа называется линейной. Рассмотрим ряд задач с использованием линейных алгоритмов.

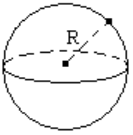
Задача 2.1

1. Формулировка задачи

Требуется написать программу, которая по заданному радиусу окружности вычисляет: длину окружности, площадь круг, площадь сферы и объем шара.

2. Математическая постановка задачи





Для расчета перечисленных характеристик воспользуемся формулами:

длина окружности – $C = 2\pi R$;

площадь круга – $S = \pi R^2$;

площадь сферы – $S = 4\pi R^2$;

объем шара – $V = \frac{4}{3}\pi R^3$.

3. Выбор переменных программы

Из приведенного выше решения определяем следующие переменные:

- исходные данные – радиус окружности (*R*);
- справочные данные – число π (*Pi*);
- результат – длина окружности (*C*), площадь круга ($S_{кр}$), площадь сферы ($S_{сф}$) и объем шара (*V*).

Так как радиус окружности может принимать любые значения (2; 2,5; 3,75), все переменные определяем как действительные числа.

4. Схема алгоритма

Составим блок схему этой задачи (рис. 2.1).

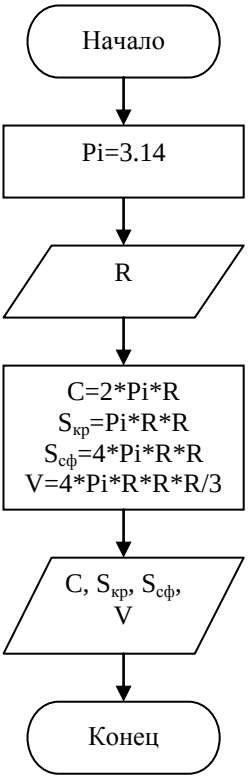


Рис. 2.1. Блок – схема к задаче 2.1

5. Программа

Программа на *C++* состоит из функций. Функция – это поименованная последовательность операторов. Функция состоит из заголовка и тела. При создании модуля автоматически создается заголовок главной функции программы *int main()*. Имя *main* указывает на то, что именно с этой функции начинается выполнение программы. Обычно главная функция возвращает целочисленное значение, за это отвечает оператор *return 0.*

В фигурных скобках ({…}) записывается тело функции.

```
#include <stdio.h>
#include <math.h>
int main()
{   float R,C,S1,S2,V;
   printf("Input R->"); scanf("%f",&R);
   C=2*M_PI*R; S1=M_PI*pow(R,2);
   S2=4*M_PI* pow(R,2); V=4*M_PI* pow(R,3)/3.0;
```

```
    printf("\nC=%6.2f",C);
    printf("\nS1=%6.2f",S1);
    printf("\nS2=%6.2f",S2);
    printf("\nV=%6.2f",V);
    getchar(); getchar();
    return 0;
}
```

Внесем пояснения по тексту программы. Так, при выборе переменных все они (*R,C,S1,S2,V*) были определены как действительные числа, то им присвоен вещественный тип *float*. В общем случае тип переменных выбирается исходя из возможного диапазона значений и требуемой точности представления данных. В табл. 2.1 приведено описание для переменных, представляющих число целого типа.

Таблица 2.1

Встроенные типы данных

Тип данных	Размер (байт)	Диапазон
char	1	-128 – 127
signed char	1	-128 – 127
unsigned char	1	0 – 255
wchar_t	2	0 – 65535
bool	1	false or true
short	2	-32768 – 32767
unsigned short	2	0 – 65535
int	Соответствует разрядности ОС	
unsigned int	Соответствует разрядности ОС	
long	4	-2147483648 – 2147483647
unsigned long	4	0 – 4294967295
long long	8	-9223372036854775808 – 9223372036854775807
float	4	$3.4*10^{-38}$ – $3.4*10^{38}$ (7 цифр)
double	8	$1.7*10^{-308}$ – $1.7*10^{308}$ (15 цифр)
long double	10	

Синтаксис оператора объявления можно описать примерно так:

тип имя_переменной [= инициализирующее_значение][, ...];

Как и любой другой оператор *C*, он оканчивается точкой с запятой.

В данном варианте программы использован ввод-вывод с помощью функций библиотеки <*stdio.h*> (в стиле *C*). Функция *printf("Input R->")* выполняет вывод переданного ей в качестве параметра строкового литерала *Input R->,* то есть последовательности символов в кавычках, на стандартное устройство вывода (дисплей). Символ '\n' называются управляющей последовательностью и задает переход на новую строку.

Для ввода исходных данных используется функция *scanf("%f",&R)*. Параметры любой функции перечисляются через запятую. В первом параметре функции *scanf* в виде строкового литерала *%f* задается спецификация формата вводимой величины, соответствующая типу вводимой переменной. Спецификация *%f* соответствует типу *float*. В качестве второго параметра функции передается адрес переменной, по которому будет помещено вводимое значение. Операция взятия адреса обозначается *&.*

Для вывода результата используем оператор *printf("\nC=%6.2f",C).* Все символы литерала, за исключением управляющей последовательности *\n* и спецификации формата *%f* выводятся на дисплей без изменения. При выводе форматная спецификация будет заменена конкретным значением переменной *C*.

Формат вывода чисел можно уточнить при помощи модификаторов формата – чисел, которые записаны перед спецификацией. Первое число «*6*» задает минимальное количество позиций, отводимых под выводимую величину, второе «*2*» – сколько из этих позиций отводится под дробную часть величины. Необходимо учитывать, что точка тоже занимает одну позицию. Если заданного количества окажется недостаточно, то компилятор автоматически выделит поле необходимой длины.

Функции ввода-вывода данного типа *printf, scanf u getchar()* без подключения библиотеки <*stdio.h*> работать не будут.

Существует другой способ организации ввода-вывода с помощью классов (в стиле C++). Если требуется ввести с клавиатуры данные, применяется так называемое приглашение к вводу *cout<<"Input R->"<<endl*. На экран выводится указанная в операторе строка символов *Input R->*, и курсор переводится на новую строку с помощью манипулятора *endl.*

В операторе *cin>>R* выполняется ввод с клавиатуры одного числа в переменную *R*. Для этого используется стандартный объект *cin* и операция извлечения (чтения) >>.

Для вывода результата используется объект *cout<<"C="<<C<<endl*, при этом все символы, находящиеся внутри кавычек, включая пробелы, выводятся без изменения. При выводе значений переменных выполняется преобразование из внутреннего представления числа в строку символов, представляющих это число. Под значение отводится ровно столько позиций, сколько необходимо для вывода всех его значащих цифр. В этом случае подключается библиотека <*iosteam.h*>.

```
#include<iostream.h>
#include <math.h>
int main()
{
   float R,C,S1,S2,V;
   cout << "Input R->" << endl;
   cin>>R;
   C=2*M_PI*R;
   S1=M_PI*pow(R,2);
   S2=4*M_PI* pow(R,2);
   V=4*M_PI* pow(R,3)/3.0;
   cout<<"C="<<C<<endl;
   cout<<"S1="<<S1<<endl;
   cout<<"S2="<<S2<<endl;
   cout<<"V="<<V<<endl;
   getchar(); getchar();
  return 0;
}
```

Так как программы содержат константу M_PI, то необходимо подключить библиотеку <*math.h>.* Список математических функций библиотеки <*math.h*> приведен в табл. 2.2.

Таблица 2.2

Список математических функций библиотеки <*math.h*>

Функция	Описание
abs	Возвращает модуль целого числа
acos	Возвращает арккосинус аргумента
asin	Возвращает арксинус аргумента
atan	Возвращает арктангенс аргумента
atan2	Возвращает арктангенс отношения аргументов
ceil	Округляет вверх
cos	Вычисляет косинус
cosh	Вычисляет гиперболический косинус

«Продолжение табл. 2.2»

Функция	Описание
exp	Возвращает степень числа e
fabs	Возвращает модуль вещественного числа
floor	Округляет вниз
fmod	Возвращает остаток от деления x на y
frexp	Выделяет из числа мантиссу и экспоненциальную часть
ldexp	Преобразует мантиссу и показатель степени в число
log	Вычисляет натуральный логарифм
log10	Вычисляет логарифм по основанию 10
modf	Разбивает число на целую и дробную части
pow	Возводит число в степень
sin	Вычисляет синус
sinh	Вычисляет гиперболический синус
sqrt	Вычисляет квадратный корень
tan	Вычисляет тангенс аргумента
tanh	Вычисляет гиперболический тангенс аргумента

Таблица 2.3

Операции языка *C*

Операция	Описание	Приоритет	Ассоциация
Первичные и постфиксные операции			
[]	вычисление индексного выражения	16	слева направо
()	вызов функции	16	слева направо
.	элемент структуры	16	слева направо
->	элемент указателя	16	слева направо
++	постфиксный инкремент	15	слева направо
--	постфиксный декремент	15	слева направо
Одноместные операции			
++	префиксный инкремент	14	справа налево

«Продолжение табл. 2.3»

Операция	Описание	Приоритет	Ассоциация
--	префиксный декремент	14	справа налево
sizeof	размер в байтах	14	справа налево
(тип)	приведение типа	14	справа налево
~	поразрядное NOT	14	справа налево
!	логическое NOT	14	справа налево
-	унарный минус	14	справа налево
&	взятие адреса	14	справа налево
*	разыменование указателя	14	справа налево
Двухместные и трехместные операции			
Мультипликативные			
*	умножение	13	слева направо
/	деление	13	слева направо
%	взятие по модулю	13	слева направо
Аддитивные			
+	сложение	12	слева направо
-	вычитание	12	слева направо
Поразрядного сдвига			
<<	сдвиг влево	11	слева направо
>>	сдвиг вправо	11	слева направо
Отношения			
<	меньше	10	слева направо

«Продолжение табл. 2.3»

Операция	Описание	Приоритет	Ассоциация
<=	меньше или равно	10	слева направо
>	больше	10	слева направо
>=	больше или равно	10	слева направо
==	равно	9	слева направо
! =	не равно	9	слева направо
Поразрядные			
&	поразрядное AND	8	слева направо
^	поразрядное XOR	7	слева направо
|	поразрядное OR	6	слева направо
Логические			
&&	логическое AND	5	слева направо
||	логическое OR	4	слева направо
Условные			
? :	условная операция	3	справа налево
Присваивания			
=	присваивание	2	справа налево
*=	присвоение произведения	2	справа налево
/=	присвоение частного	2	справа налево
%=	присвоение модуля	2	справа налево

«Продолжение табл. 2.3»

Операция	Описание	Приоритет	Ассоциация
+=	присвоение суммы	2	справа налево
-=	присвоение разности	2	справа налево
<<=	присвоение левого сдвига	2	справа налево
>>=	присвоение правого сдвига	2	справа налево
&=	присвоение AND	2	справа налево
^=	присвоение XOR	2	справа налево
|=	присвоение OR	2	справа налево
,	последовательного вычисления	1	слева направо

6. Тесты

Для испытания программы можно использовать тесты, представленные в табл. 2.4.

Таблица 2.4

Тесты для задачи 2.1

R	C	$S_{кр}$	$S_{сф}$	V
0	0	0	0	0
2	12,57	12,57	50,27	33,51
2,25	14,14	15,9	63,62	47,71

7. Результаты тестирования

Результат работы программы представлен на рис. 2.2.

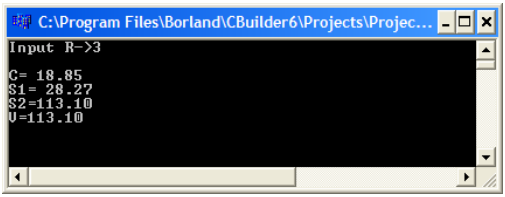


Рис. 2.2. Результат работы программы задачи 2.1

Задача 2.2

Составить программу для вычисления значения функции для любого заданного *x*:

$$y = \frac{1+\sin^3\left(x^2+2x\right)}{2+\left|x-\dfrac{2x}{\sin x+2}\right|}+e^{2x+1}.$$

Листинг программы

```
#include <stdio.h>
#include <math.h>
int main()
{
   float x,x1,x2,y;
   printf("Input x->");
   scanf("%f",&x);
   x1=pow(sin(pow(x,2)+2*x),3)+1;
   x2=fabs(x-2*x/(sin(x)+2))+2;
   y=x1/x2+exp(2*x+1);
   printf("\ny=%6.2f",y);
   getchar(); getchar();
 return 0;
}
```

Результаты тестирования

Результат работы программы представлен на рис. 2.3.

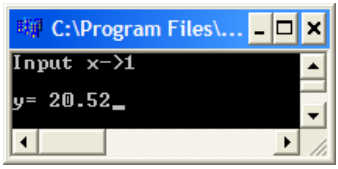
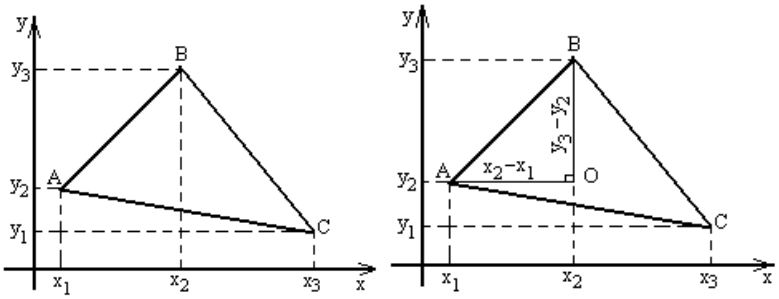


Рис. 2.3. Результат работы программы задачи 2.2

Задача 2.3

Треугольник задается координатами своих вершин на плоскости: *A(x1,y1)*, *B(x2,y2)*, *C(x3,y3).* Требуется написать программу, которая вычисляет площадь треугольника *ABC*.



Для решения задачи можно использовать формулу Герона: $S = \sqrt{(P(P-A)(P-B)(P-C))}$, где *P* – полупериметр.

Задача 2.4

Требуется написать программу, которая по введенному количеству секунд определяет количество суток, часов, минут и секунд. Например,

4000 с = 0 сут. 1 ч. 6 мин. 40 с.

Задача 2.5

Требуется написать программу, которая по введенному времени определяет меньший угол между часовой и минутной стрелкой. Например,

12 ч. 00 мин – угол = 0.

2 ч. 30 мин – угол = 105.

Индивидуальные задания
по теме «Программирование линейных алгоритмов»

Задание И2.1. Составить программу для вычисления значения функции для любого заданного *x*. При выводе исходных данных результат округлять до второго знака после запятой:

№	Значение функции	№	Значение функции
1	$y = x\ln(x^4+5) + \frac{x}{\cos x + 2} + 2\sin\sqrt{x^2+1}$	2	$y = e^{\sin^2 x+1} - \frac{x}{\ln\lvert\cos^3 x + 5\rvert} + (1+x)^x$
3	$y = \frac{\sin x + \cos x}{\sqrt[3]{\cos x - \sin x + 3}} + (e^x + 1)^2$	4	$y = 3^x + 4x - \frac{\ln\left\lvert\frac{\cos^2 x + 1}{x^2+1}\right\rvert}{2\sin x + 25}$
5	$y = \frac{\cos x}{\pi + 2x^2} + x\ln\left(x^4 + \sqrt{\lvert x\rvert}\right)$	6	$y = \frac{1+\sin^3(x^2+2x)}{2+\left\lvert x - \frac{2x}{\sin x + 2}\right\rvert} + e^{2x+1}$

№	Значение функции	№	Значение функции
7	$y=\frac{1+\sin\sqrt{\lvert x+1\rvert}}{\cos^2\left(12x^2+4\right)+1}+e^{\frac{\sqrt{\lvert x\rvert}}{\sin^2 x+2}}$	8	$y=\sqrt{\lvert 2\sin^3 3x\rvert}-\frac{\ln\left(\cos^2 x+1\right)}{\ln\lvert x^4+2x^2+10\rvert}-e^{\frac{-2x}{5}}$
9	$y=x+\frac{x^3}{3}-\frac{x^5}{5}+\frac{xe^{2x}}{\sin^2 x+1}$	10	$y=2\ln\left\lvert\frac{\sin x+2}{\cos^2 x+1}\right\rvert+e^{\sin^2 x+1}$
11	$y=2^{-x}-\cos x+\frac{\sin(2x)}{\ln\left(\cos\frac{e^{-x}}{x^2+1}+2\right)}$	12	$y=\frac{1+\sin^2\left(x^2+2x\right)}{2+\left\lvert x-\frac{2x}{\left(1+x^2\right)\left(\sin x+2\right)}\right\rvert}+\ln\lvert\sin^2 x-2\rvert$
13	$y=\lvert x^2-x^3\rvert-\frac{7x}{\left(x^3+15x\right)^2+1}+e^{\cos^2 x+1}$	14	$y=x\ln\left(x^2+1\right)+\frac{x}{\cos^2 x+1}+\sqrt{\lvert x+1\rvert}$
15	$y=\frac{1+\sin\sqrt{x^2+1}}{\cos^2 12x+1}+\ln\left(1+x^2\right)$	16	$y=\frac{\left(\sin^2 x+\cos x\right)^{3x}}{\ln\left(\frac{\cos x+2}{e^{\sin^2 x+1}}\right)}-\left(5+\cos^2 x\right)^x$
17	$y=x-10^{\sin x}-\frac{\ln\lvert\cos x+2\rvert}{e^{\frac{\cos x}{\sin^2 x+1}}}$	18	$y=3^x+4x-\frac{\ln\left\lvert\frac{\cos^2 x+1}{x^2+1}\right\rvert}{2\sin x+25}$
19	$y=x-\frac{10\sin x}{\ln\left(\frac{e^{-2x}}{2\sin^2 x+1}\right)}+\lvert x^4-x^5\rvert$	20	$y=\cos^2\sin\frac{1}{x^2+1}-\frac{\ln\left\lvert\frac{\cos x+2}{\sin^2 x+1{,}2}\right\rvert}{e^{-2x}+1}$

Задание И2.2. Составить программу для решения задач, согласно варианту.

1. Лошадиный барышник на ярмарке купил лошадь за m рублей, а продал за n. Через некоторое время он купил ту же лошадь за h рублей, а продал за q. Каков его барыш?
2. Скорый поезд вышел из Москвы в Санкт-Петербург и шел без остановок со скоростью v $\frac{км}{ч}$. Другой поезд вышел ему навстречу из Санкт-Петербурга и тоже шел без остановок, но со скоростью g $\frac{км}{ч}$. На каком расстоянии друг от друга будут поезда за t часов до встречи?
3. У n хозяев по n кошек, каждая кошка съедает по n мышей, каждая мышь съедает по n колосьев ячменя, из каждого колоса может вырасти по *n* мер зерна. Сколько мер зерна сохраняется благодаря этим кошкам?

4. Между городами *A* и *B* S километров. Их них выехали два велосипедиста и со скоростью *v* $\frac{км}{ч}$ каждый помчались навстречу друг другу. Вместе с первым велосипедистом из города *A* стартовала муха со скоростью *g* $\frac{км}{ч}$. Встретившись с велосипедистом из города *B*, муха развернулась и полетела к первому, встретившись с ним, опять полетела ко второму Когда велосипедисты съехались и остановились, муха угомонилась и села одному из них на голову. Сколько километров пролетела муха?
5. Пасли ребята коней. Если пересчитать ноги коней и детей, получится *n*, а если головы, то *m*. Сколько было ребят и сколько коней?
6. В жаркий день *n* косцов выпили бочонок кваса за *m* часов. Сколько косцов выпьют такой же бочонок кваса за *t* часов?
7. Вол съел копну за *t*, конь – за *m*, коза – за *n* часов. За сколько времени вол, конь и коза – съедят ту копну вместе?
8. Веселый француз пришел в трактир с неизвестною суммой своего богатства, занял у хозяина столько денег, сколько у себя имел; из сей суммы издержал n рублей. С остатком пришел в другой трактир, где опять, занявши столько, сколько имел, издержал в оном также *n* рублей; то же учинил в третьем и четвертом трактирах. По выходе из четвертого не имел уже ничего. С какой суммой пришел он в первый трактир?
9. Коммерсант, имея стартовый капитал *k* рублей, занялся торговлей, которая ежемесячно увеличивает капитал на *p*%. Через сколько лет он накопит сумму *s*, достаточную для покупки собственного магазина?
10. Селекционер вывел новый сорт зерновой культуры и снял с опытной делянки *k* кг семян. Посеяв 1 кг семян, можно за сезон собрать *p* кг семян. Через сколько лет селекционер сможет засеять новой культурой поле площадью *s* га, если норма высева *n* кг/га?
11. За первый год производительность труда на предприятии возросла на *p1*%, за второй и третий – соответственно на *p2* и *p3*%. Найти среднегодовой прирост производительности (в процентах).
12. Заданы уравнения двух пересекающихся прямых на плоскости: *y = k1*x+ b1* и *y = k2*x+ b2*. Найти (в градусах и минутах) угол между ними.

13. Из круга радиуса *r* вырезан прямоугольник, большая сторона которого равна *a*. Найти максимальный радиус круга, который можно вырезать из полученного прямоугольника?
14. Владелец автомобиля приобрел новый карбюратор, который экономит *50%* топлива, новую систему зажигания, которая экономит *30%* топлива, и поршневые кольца, экономящие *20%* топлива. Верно ли, что его автомобиль теперь сможет обходиться совсем без топлива? Найти фактическую экономию для произвольно заданных сэкономленных процентов.
15. В пространстве заданы два вектора своими координатами. Найти угол (в градусах) между векторами *A* и *B*.
16. На тело действуют две силы, заданные векторами *A* и *B*. Найти величину и направление (углы с координатными осями) их равнодействующей.
17. Животновод в начале каждой зимы повышает отпускную цену на молоко на *p*%, а каждым летом – снижает на столько же процентов. Изменится ли цена на молоко и если да, то в какую сторону и на сколько через *N* лет?
18. Пловцу надо под прямым углом к фарватеру преодолеть реку Урал шириной *b* м. Его скорость в стоячей воде *v1 м/с*; скорость течения реки – *v2 м/с*. Под каким углом к фарватеру он должен плыть, чтобы его «не снесло»? Сколько времени займет переправа? Как изменится решение, если посредине реки пловец устал, и его скорость с *v1 м/с* упала до *v3 м/с*?
19. Студент съедает торт за пять минут, преподаватель – за полчаса, а заведующий кафедрой – за час. За какое время они съедят торт вместе? (4 минуты)
20. Правительство гарантирует, что процент инфляции в новом году составит *p*% в месяц. Какого роста цен за год можно ожидать?
21. В бассейн поступает холодная и горячая вода. Холодной водой бассейн заполняется за 6 минут 40 секунд, а горячей водой – за 8 минут. Вся вода вытекает из полного бассейна за 13 минут 20 секунд. Сколько времени понадобится, чтобы наполнить бассейн полностью, при условии, что открыты оба крана, но слив открыт? (за 5 минут)
22. Студенты варили варенье, но забыли закрыть окно. С улицы в открытое окно залетали пчелы. В первый час на варенье прилетела 1 пчела, во второй час – 2 пчелы, а в третий – 3 пчелы и т.д. Сколько пчел залетело на варенье за сутки, если окно не закрывали? (300 пчел)
23. Старинная задача. Летела стая гусей, а навстречу им летит один гусь и говорит: Здравствуйте, сто гусей!» А передний ста-

рый гусь ему и отвечает: «Нет, нас не сто гусей! Вот если б нас было еще столько, да еще полстолька, да еще четверть столько, да ты, гусь, то было бы сто гусей, а теперь … Вот и рассчитай-ка, сколько нас?» (36 гусей)

24. Старинная народная задача. Крестьянка пришла на базар продавать яйца. Первая покупательница купила у нее половину всех яиц и еще пол-яйца. Вторая покупательница приобрела половин оставшихся яиц и еще пол-яйца. Третья купила всего одно яйцо. После этого у крестьянки не осталось ничего. Сколько яиц она принесла на базар? (7)
25. Муж сообщил своей жене, что отныне каждый месяц им следует экономить вдвое больше, чем в предыдущем месяце. Они сэкономят 1 доллар в первый месяц, 2 доллара во второй месяц, 4 доллара в третий месяц и так далее. Сколько они сэкономят за год? (4095)
26. В 5 мисках – 100 орехов. В первой и второй мисках вместе 52 ореха. Во второй и в третьей – 43, в третьей и четвертой – 34, в четвертой и пятой – 30. Сколько орехов в каждой миске? (27, 25, 18, 16 и 14)

ЛАБОРАТОРНАЯ РАБОТА № 3.
ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ВЕТВЛЕНИЯ

В линейной программе все операторы выполняются последовательно, один за другим. Для того чтобы в зависимости от исходных данных обеспечить выполнение разных последовательностей операторов, применяются операторы ветвления *if* и *switch*. Оператор *if* обеспечивает передачу управления на одну из двух ветвей вычислений, оператор *switch* – на одну из произвольного числа ветвей.

Условный оператор if

Используется для разветвления процесса вычислений на два направления. Структурная схема оператор приведена на рис. 3.1.

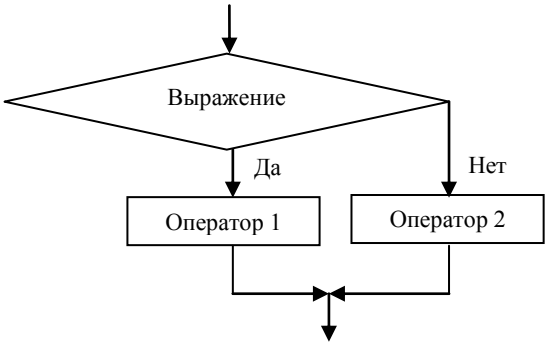


Рис. 3.1. Структурная схема условного оператора

Формат оператора:

if (выражение) оператор_1; [*else* оператор_2;]

Сначала вычисляется выражение, которое может иметь арифметический тип или тип указателя. Если оно принимает значение не равное нулю, выполняется первый оператор, иначе второй. После этого управление передается на оператор, следующий за условным.

Одна из ветвей может отсутствовать (опускается ключевое слово *else*). Если требуется выполнить несколько операторов, их необходимо заключить в блок *({})*, иначе компилятор не сможет понять, где заканчивается ветвление. Блок может содержать любые операторы, в том числе и описания и другие условные операторы (но не может состоять из одних описаний). Необходимо учитывать, что переменная, описанная в блоке, вне блока не существует.

Задача 3.1. Расчет значений кусочных функций

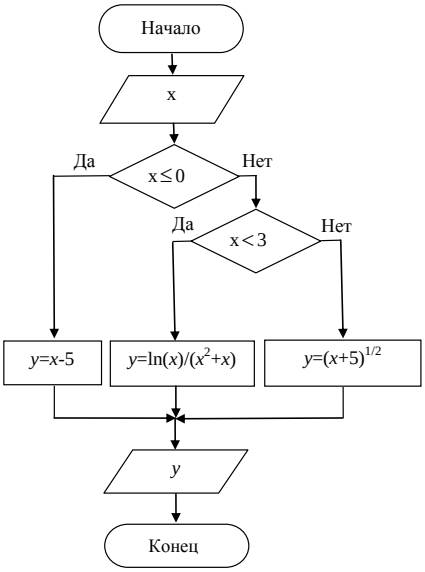
Определить значение кусочной функции для любого заданного *x*.

$$y = \begin{cases} x - 5, & x \le 0, \\ \dfrac{\ln x}{x^2 + x}, & 0 < x < 3, \\ \sqrt{x+5}, & x \ge 3. \end{cases}$$

Выбор переменных программы: аргумент и значение функции могут принимать любые значения (2; 2,5; 3,75), все переменные определяем как действительные числа.

Схема алгоритма

Составим блок схему этой задачи (рис. 3.2).



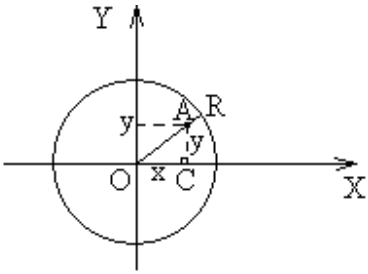
Листинг программы

```
#include <stdio.h>
#include <math.h>
int main()
{
float x,y;  // Аргумент и значение функции,
            // переменные вещественного типа
 printf("Enter x ");
 scanf("%f",&x);  // Ввод переменой x
 if (x<=0)
```

```
 y=x-5;
 else if (x<3)
  y=log(pow(x,2)+x);
 else
  y=pow(x+5,1/2.);
printf("y(%2.1f)=%2.1f",x,y);  // Вывод результата в виде y(x)=y
getchar();getchar();
return 0;
}
```

Задача 3.2. Написать программу, которая определяет по введенным координатам, попадает ли заданная точка в окружность, с центром в точке O (0,0) и заданным радиусом R.

Математическая постановка задачи



Заданная точка имеет координаты – x, y. Рассмотрим треугольник AOC. <OCA=90º, OC=x, AC=y, следовательно по теореме Пифагора AO^2 (гипотенуза)= OC^2+AC^2.

$AO^2=x^2+y^2$. Поэтому условие принадлежности точки окружности можно записать в виде:

$x^2+y^2 \le R^2$.

Выбор переменных программы

Из приведенного выше решения определяем следующие переменные:

- исходные данные – радиус окружности (*R*) и координаты точки (*x* и *y*);
- результат – сообщение – «находится точка в окружности или нет».

Так как радиус окружности и координаты точки могут принимать любые значения (2; 2,5; 3,75), все переменные определяем как действительные числа.

Схема алгоритма

Составим блок-схему этой задачи (рис. 3.2).

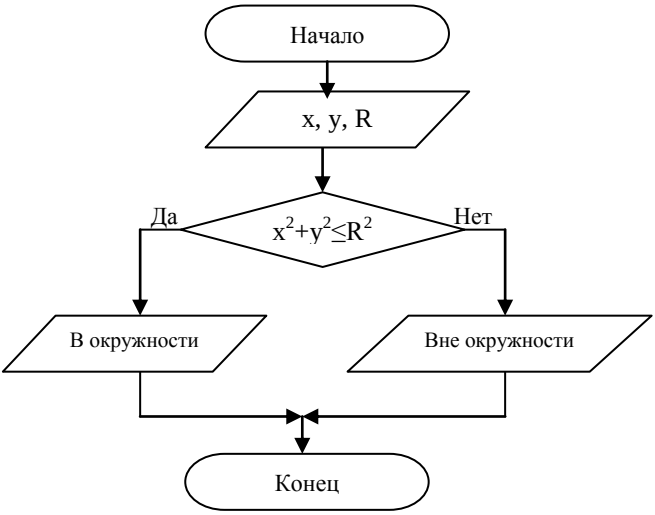


Рис. 3.2. Блок схема к задаче 3.2

Листинг программы

```
#include <stdio.h>
int main()
{
float x,y,R;  // Координаты точки и радиус,
              // переменные вещественного типа
 printf("Введите X, Y, R ->");
 scanf("%f%f%f",&x,&y, &R);//Ввод переменных x,y,R
 if (x*x+y*y<=R*R)
 printf("A(%2.1f,%2.1f) принадлежит окружности (R=%2.1f)", x,y,R);
 else
 printf("A(%2.1f,%2.1f) не принадлежит окружности (R=%2.1f)", x,y,R);
getchar();getchar();
return 0;
}
```

Тесты

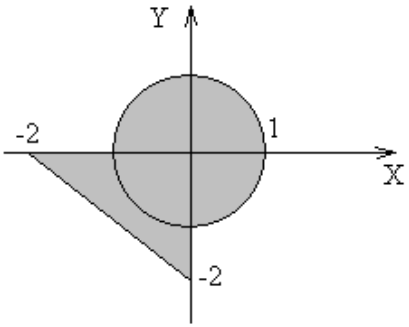
Для испытания программы можно взять тесты, представленные в табл. 3.1.

Таблица 3.1

Тесты для задачи 3.2

X	y	R	Результат
0	0	0	Принадлежит
2	2	1	Не принадлежит
1	1	2	Принадлежит

Задача 3.3. Написать программу, которая по введенным координатам точки определяет, попадает ли эта точка в заштрихованную область.



Математическая постановка задачи

Запишем условия попадания точки в область в виде формул. Область можно описать как круг, пересекающийся с треугольником. Точка может попадать либо в круг, либо в треугольник, либо в их общую часть:

$$\left\{x^2 + y^2 \le 1\right\} \textit{ или } \begin{Bmatrix} x \le 0 \\ y \le 0 \\ y \ge -x - 2 \end{Bmatrix}.$$

Первое условие задает попадание точки в круг, второе – в треугольник.

Выбор переменных программы

Из приведенного выше решения определяем следующие переменные:

- исходные данные – координаты точки (*x* и *y*);
- результат – сообщение – «находится точка в заштрихованной области».

Так как координаты точки могут принимать любые значения (2; 2,5; 3,75), все переменные определяем как действительные числа.

Схема алгоритма

Составим блок схему этой задачи (рис. 3.3).

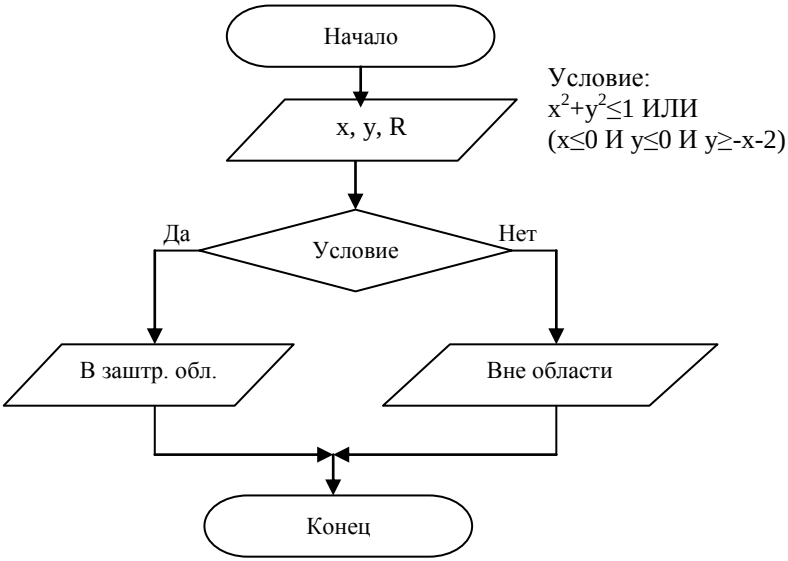


Рис. 3.3. Блок-схема к задаче 3.2

Листинг программы

```
#include <stdio.h>
main()
{
float x,y;  //Координаты точки
 printf("Введите X, Y ->");
 scanf("%f%f",&x,&y);
 if (x*x+y*y<=1 || x<=0 && y<=0 && y>=-x-2)
 printf("A(%2.1f,%2.1f) принадлежит ", x,y);
 else
 printf("A(%2.1f,%2.1f) не принадлежит ", x,y);
getchar();getchar();
return 0;
}
```

Тесты

Для испытания программы можно взять тесты, представленные в табл.3.2.

Таблица 3.2

Тесты для задачи 3.3

x	y	Результат
0	0	Принадлежит
1	1	Не принадлежит
-1	-1	Принадлежит

Оператор выбора switch

Оператор *switch* (переключатель) предназначен для разветвления процесса вычислений на несколько направлений. Формат оператора:

```
switch (выражение)
{
case    константное_выражение_1: [список_операторов_1]
case    константное_выражение_2: [список_операторов_2]
…
case    константное_выражение_n: [список_операторов_n]
[default:          операторы]
 }
```

Выполнение оператора начинается с вычисления выражения (оно должно быть целочисленным), а затем управление передается первому оператору из списка, помеченного константным выражением, значение которого совпало с вычисленным. После этого, если выход из переключателя явно не указан, последовательно выполняются все остальные ветви.

Выход из переключателя обычно выполняется с помощью оператора *break*.

Все константные выражения должны иметь различные значения, но быть одного и того же целочисленного типа.

Задача 3.4. Составить программу, реализующую действия простейшего калькулятора.

Выбор переменных программы

Определяем следующие переменные:

– исходные данные – операнды (*a* и *b*) и знак операции (*op*);
– результат (*res*).

Операнды определяем как действительные числа, а знак операций как символ.

Листинг программы

```
#include <stdio.h>
main()
{
float a,b; //Операнды
float res; //Результат
char op;  //Знак операции
bool f=true;
 printf("Input op ->");
 scanf("%c",&op);
```

```
 printf("Input a, b ->");
 scanf("%f%f",&a,&b);
 switch (op)
 {
  case '+':
    res=a+b;
    break;
  case '-':
    res=a-b;
    break;
  case '*':
    res=a*b;
    break;
  case '/':
    res=a/b;
    break;
  deault :
    printf("No %c operation", op);
    f=false;
 }
  if (f) printf("%2.1f %C %2.1f = %2.1f", a,op,b,res);
getchar();getchar();
return 0;
}
```

Индивидуальные задания
по теме «Программирование алгоритмов условного перехода»

Задание И3.1. Расчет значений кусочных функций

Определить значение кусочной функции для любого заданного x.

1.
$$y=\begin{cases} x^2,\ x\le 0,\\ \cos x,\ 0<x<5,\\ \sqrt{x},\ x\ge 5. \end{cases}$$

2.
$$y=\begin{cases} x^2-4,\ x\le 0,\\ \cos x+5,\ 0<x<5,\\ \sqrt{x}+x^2,\ x\ge 5. \end{cases}$$

3.
$$y=\begin{cases} e^x+x^2,\ x\le 0,\\ \cos x,\ 0<x<5,\\ 3x-\sqrt{x},\ x\ge 5. \end{cases}$$

4.
$$y=\begin{cases} \ln|x|,\ x<0,\\ \cos x,\ \ 0\le x<3,\\ x^{-5/6},\ x\ge 3. \end{cases}$$

5.
$$y=\begin{cases} e^{-x},\ x\le 0,\\ \sin 2x,\ 0<x<5,\\ \sqrt{x},\ x\ge 5. \end{cases}$$

6.
$$y=\begin{cases} \sin x+x^2,\ x\le 0,\\ \cos x,\ 0<x<5,\\ e^x+\sqrt{x},\ \ x\ge 5. \end{cases}$$

7.
$$y=\begin{cases}\sqrt[3]{x^2}, & x<0,\\ \cos x, & 0\le x<5,\\ x+3, & x\ge 5.\end{cases}$$

8.
$$y=\begin{cases}e^{-2x}, & x\le 0,\\ \cos(x-\dfrac{\pi}{6}), & 0<x<5,\\ \sqrt{x}, & x\ge 5.\end{cases}$$

9.
$$y=\begin{cases}x^4, & x\le \pi,\\ \cos x, & \pi<x<5,\\ \sqrt{x}, & x\ge 5.\end{cases}$$

10.
$$y=\begin{cases}e^{x}, & x\le -5,\\ 0, & -5<x<5,\\ e^{-x+1}, & x\ge 5.\end{cases}$$

11.
$$y=\begin{cases}x^2, & x\le -\dfrac{3}{2},\\ \cos(x), & -\dfrac{3}{2}<x<5,\\ \sqrt{|\sin x|}, & x\ge 5.\end{cases}$$

12.
$$y=\begin{cases}x^2, & x\le 6,\\ e^{-x}, & 6<x<25,\\ 4, & x\ge 25.\end{cases}$$

13.
$$y=\begin{cases}x-5, & x\le 0,\\ 2\cos x, & 0<x<10,\\ 5+\sqrt{x}, & x\ge 10.\end{cases}$$

14.
$$y=\begin{cases}\sin 2x, & x\le -2\pi,\\ e^{x}+e^{-x}, & -2\pi<x<5,\\ \cos 2x, & x\ge 5.\end{cases}$$

15.
$$y=\begin{cases}x^2, & x\le 0,\\ \ln x, & 0<x<5,\\ \sqrt{x}-5x, & x\ge 5.\end{cases}$$

16.
$$y=\begin{cases}x+5, & x<0,\\ e^{x}, & 0\le x\le 1,\\ \sin x, & x>1.\end{cases}$$

17.
$$y=\begin{cases}3x^2, & x\le -1,\\ 5x, & -1<x\le 3,\\ x-4, & x>3.\end{cases}$$

18.
$$y=\begin{cases}\sin x, & x<0,\\ 10, & x=0,\\ \cos x, & x>0.\end{cases}$$

19.
$$y=\begin{cases}x^2+22, & x<-5,\\ e^{x}, & -5\le x\le 5,\\ \ln x, & x>5.\end{cases}$$

20.
$$y=\begin{cases}x^2-5, & x\le -2,\\ \cos 2x, & -2<x<1,\\ 15+\sqrt{x}, & x\ge 1.\end{cases}$$

21.
$$y=\begin{cases}\sin(x-5), & x\le 0,\\ \dfrac{\cos 2x}{x^2+1}, & 0<x<3,\\ \sin(5+\sqrt{x}), & x\ge 3.\end{cases}$$

22.
$$y=\begin{cases}x^2-3x+9, & x\le 3,\\ \dfrac{1}{x^3+6}, & 3<x<6,\\ \sqrt{x}, & x\ge 6.\end{cases}$$

23.
$$y=\begin{cases}-x^2-3x+9, & x\ge 3,\\ \dfrac{1}{x^3-6}, & 3>x\ge 2,\\ \sqrt{|x|}, & x<2.\end{cases}$$

24.
$$y=\begin{cases}x-3, & x\le 3,\\ \ln x+2, & 3<x<6,\\ \sqrt[3]{x}, & x\ge 6.\end{cases}$$

25.

$$y=\begin{cases} x-5,\ x\le 0,\\ \dfrac{\ln x}{x^2+x},\ 0<x<3,\\ \sqrt{x+5},\ x\ge 3. \end{cases}$$

26.

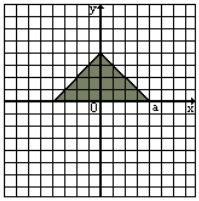
$$y=\begin{cases} \sqrt{|x-5|},\ x\le 0,\\ \sqrt{|2\cos x|},\ 0<x<1,\\ \sqrt{x},\ x\ge 1. \end{cases}$$

27.

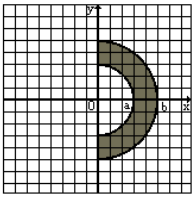
$$y=\begin{cases} \dfrac{e^x}{x^2+1},\ x\le -3,\\ (x+3)^2,\ -3<x<5,\\ \dfrac{5+\ln\left(\sqrt{x}\right)}{\sqrt{x^4+x+5}},\ x\ge 5. \end{cases}$$

Задание ИЗ.2. Программирование геометрического места точек. Составить программу для определения принадлежности точки заданной области.

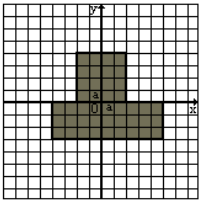
1.



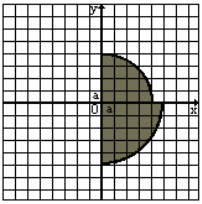
2.



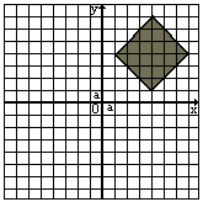
3.



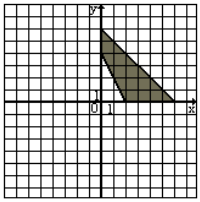
4.



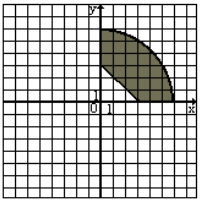
5.



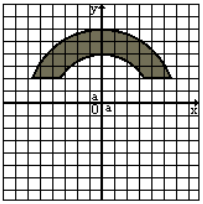
6.



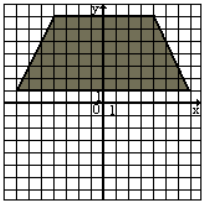
7.



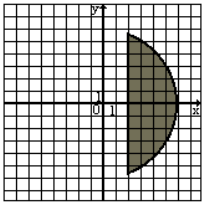
8.



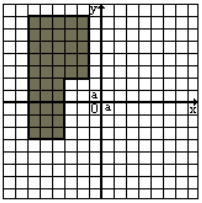
9.



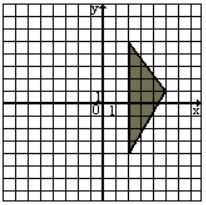
10.



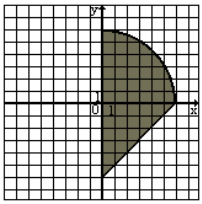
11.



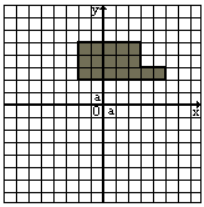
12.



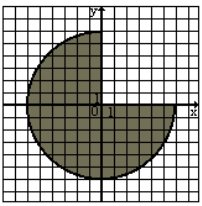
13.



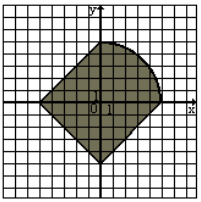
14.



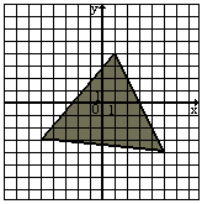
15.



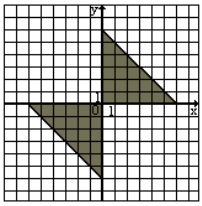
16.



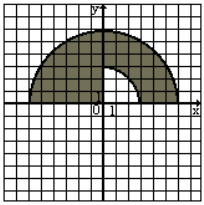
17.



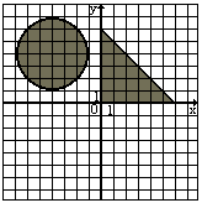
18.



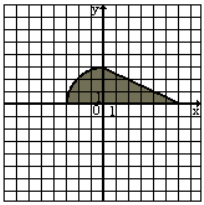
19.



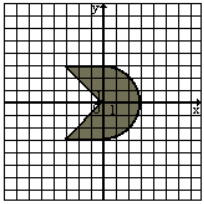
20.



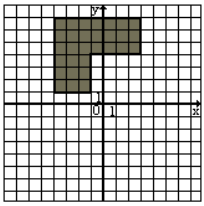
21.



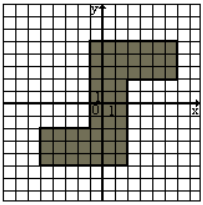
22.



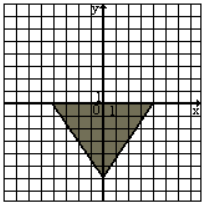
23.



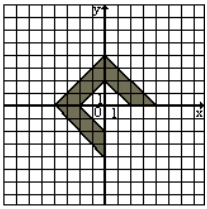
24.



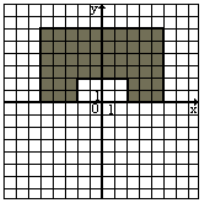
25.



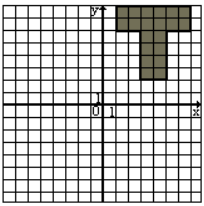
26.



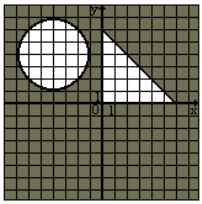
27.



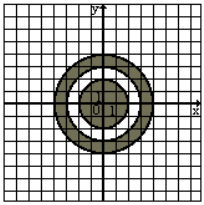
28.



29.



30.



Задание И3.3.

1. Задан четырехугольник со сторонами *A*, *B*, *C*, *D*. Определить, является ли он параллелограммом или ромбом.
2. На плоскости задан отрезок координатами своих концов. Определить, имеет ли он общие точки с осями координат.
3. На плоскости задан отрезок координатами своих концов. Определить, принадлежит ли отрезок полностью первой четверти координатной плоскости.
4. На плоскости задан отрезок координатами своих концов. Определить, параллелен ли он осям координат.
5. На плоскости задана окружность с центром в точке (*X1*, *Y1*), радиусом *R*. Определить, принадлежит ли точка с координатами *X*, *Y* заданному кругу, лежит на ее границе или вне ее.
6. На плоскости задана точка с координатами *X*, *Y*. Определить, какой четверти принадлежит точка.
7. Задан координатами своих концов отрезок на плоскости. Определить, имеет ли он длину больше *L1* и меньше *L2* (*L1<L2*).
8. На плоскости заданы две концентрические окружности радиусами *R1*, *R2*, с центром в точке (*X*, *Y*). Определить, принадлежит ли точка *X1*, *Y1* кольцу между двумя окружностями.
9. Даны три отрезка *A*, *B*, *C*. Определить, существует ли треугольник со сторонами, равными данным отрезкам.
10. Даны три отрезка *A*, *B*, *C*. Определить, является ли треугольник, со сторонами, равным отрезкам, остроугольным, прямоугольным или тупоугольным.
11. Задан треугольник координатами своих вершин. Определить, является он остроугольным, прямоугольным или тупоугольным.
12. Задан треугольник координатами своих вершин. Определить, является ли он тупоугольным с периметром больше заданного числа *M*.
13. Задана точка на плоскости и два прямоугольника со сторонами, параллельными осям координат. Определить, принадлежит точка только первому прямоугольнику, только второму или обоим вместе.
14. На плоскости заданы точка, круг, радиусом *R* с центром в начале координат и вписанный в него квадрат со сторонами, параллельными осям координат. Определить, находится точка одновременно и в круге, и в квадрате, вне их или только в круге.
15. На плоскости заданы точка, круг радиусом *R* с центром в начале координат и описанный вокруг него квадрат со сторона-

ми, параллельными осям координат. Определить, находится точка одновременно и в круге, и в квадрате, вне их или только в круге.

16. На плоскости задана точка, не находящаяся на осях координат. Если точка лежит в круге с центром в начале координат и радиусом *R*, определить, какой четверти координатной плоскости принадлежит точка.
17. Даны две точки *A(x1,y1)* и *B(x2,y2)*. Составить программу, определяющую, которая из точек находится ближе к началу координат.
18. Даны меры двух углов треугольника (в градусах). Определить, существует ли треугольник с такими углами, и если да, то будет ли он прямоугольным.
19. С клавиатуры вводятся длины отрезков *a*, *b*, *c* и *d*. Оценить их на возможность построения треугольников. В качестве теста взять числа 3, 5, 9, 10.
20. Даны три точки *A(x1,y1), B(x2,y2)* и *C(x3,y3)*. Определить будут ли они расположены на одной прямой. Если нет, то вычислить $\angle ABC$.
21. На плоскости расположены три точки *a, b, c*. Определить, какая из точек *b* или *c* расположены ближе к *a*.
22. Даны три положительных числа *a, b, c*. Проверить, могут ли они быть длинами сторон треугольника. Если да, то вычислить площадь этого треугольника.
23. Дана окружность радиусом *R*. Определить, впишется ли правильный треугольник со стороной a в эту окружность.
24. Заданы размеры *A*, *B* прямоугольного отверстия и размеры *x*, *y*, *z* кирпича. Определить, пройдет ли кирпич через это отверстие, параллельно сторонам отверстия.
25. Заданы отрезки *a*, *b*, *c*, *d*. Проверить могут ли они служить сторонами прямоугольника. Если да, то вычислить площадь этого прямоугольника.
26. Дана точка *A(x,y)*. Определить, принадлежит ли она треугольнику с вершинами в точках *(x1,y1)*, *(x2,y2)*, *(x3,y3)*.
27. Два прямоугольника, заданы координатами своих левого верхнего и правого нижнего углов (стороны параллельны осям). Для первого прямоугольника это точки *(x1,y1)* и *(x2,0)*, для второго – *(x3,y3)* и *(x4,0)*. Составить программу, определяющую, пересекаются ли данные прямоугольники, и вычисляющую площадь общей части, если она существует.

28. На плоскости задана своими координатами точка *A*. Укажите, где она расположена (на какой оси или в каком координатном угле).
29. Заданы три стороны треугольника *a*, *b* и *c*. Определить является ли этот треугольник прямоугольным, и, если является, какая сторона служит гипотенузой.
30. Найти координаты точек пересечения прямой *y=kx+b* и окружности радиуса *R* с центром в начале координат. В каких координатных четвертях находятся точки пересечения? Если точек пересечения нет, или прямая касается окружности, выдать соответствующее сообщение.
31. На плоскости задана окружность, радиусом *R*, с центром в начале координат, и дана точка с координатами *X, Y*. Определить, принадлежит ли точка кругу, лежит на границе или вне ее.

Задание И3.4. Задачи по теме «Оператор выбора»

1. Составьте программу вычисления функции, если *x* – натуральное число; *a, b, c* – действительные числа:

$$y=\begin{cases} a+bx+cx^2, & x=1; \\ a\cdot\sin\left(xb^2\right), & x=2; \\ \sqrt{\left|a+bx\right|}, & x=3; \\ a\cdot\ln\left|bx+cx^2\right|, & x=4; \\ e^{-ax}+\sin bx+cx, & x=5. \end{cases}$$

   В противном случае (если *x* не входит в заданный диапазон) вывести сообщение.
2. Написать программу, которая бы по введенному номеру времени года (1 – зима, 2 – весна, 3 – лето, 4 – осень) выдавала соответствующие этому времени года месяцы, количество дней в каждом из месяцев.
3. В старояпонском календаре был принят 12-летний цикл. Годы внутри цикла носили названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, курицы, собаки и свиньи. Написать программу, которая вводит номер некоторого года и печатает его название по старояпонскому календарю.
   (Справка: 1996 г. – год Крысы – начало очередного цикла.)
4. Для целого числа *k* от 1 до 99 напечатать фразу «Мне *k* лет», учитывая при этом, что при некоторых значениях *k* слово

«лет» надо заменить на слово «год» или «года». Например, 11 лет, 22 года, 51 год.

5. Написать программу, которая бы по введенному номеру единицы измерения (1 – дециметр, 2 – километр, 3 – метр, 4 – миллиметр, 5 – сантиметр) и длине отрезка *L* выдавала бы соответствующее значение длины отрезка в метрах.
6. Написать программу, которая по введенному числу от 1 до 12 (номеру месяца) выдает все приходящиеся на этот месяц праздничные выходные дни (например, если введено число 1, то должно получиться 1 января – Новый год, 7 января – Рождество).
7. Составить программу, которая на ввод знака препинания выдает ни экран дисплея его название. Например, на ввод точки выдает текст: «Точка».
8. Даны два действительных положительных числа *x* и *y*. Арифметические действия над числами пронумерованы (1 – сложение, 2 – вычитание, 3 – умножение, 4 – деление). Составить программу, которая по введенному номеру выполняет то или иное действие над числами.
9. Написать программу, которая бы по введенному номеру единицы измерения (1 – килограмм, 2 – миллиграмм. 3 – грамм, 4 – тонна, 5 – центнер) и массе M выдавала бы соответствующее значение массы в килограммах.
10. Пусть элементами равностороннего треугольника являются:
    - сторона *a*;
    - площадь *S*;
    - высота *h*;
    - радиус вписанной окружности *r*;
    - радиус описанной окружности *R*.

    Составить программу, которая по заданному номеру и значению соответствующего элемента вычисляла бы значение всех остальных элементов треугольника.
11. Составить программу для определения подходящего возраста кандидатуры для вступления в брак, используя следующее соображение: возраст девушки равен половине возраста мужчины плюс 7, возраст мужчины определяется соответственно как удвоенный возраст девушки минус 14.
12. Напишите программу, которая по заданной дате определяет время года. Программа должна проверять корректность введенной даты.

13. Напишите программу, которая читает натуральное число в десятичном представлении, а на выходе выдает это же число в десятичном представлении и на естественном языке.
    Например, 7 семь; 204 двести четыре; 52 пятьдесят два.
14. Вычислить порядковый номер дня в невисокосном году по заданным числу и месяцу.
15. Даны два комплексных числа *x* и *y*. Арифметические действия над числами пронумерованы (1 – сложение, 2 – вычитание, 3 – умножение, 4 – деление). Составить программу, которая по введенному номеру выполняет то или иное действие над числами.
16. Написать программу, которая требует ввода времени дня и, в зависимости от введенного значения, желает доброго утра, доброго дня, доброго вечера или спокойной ночи.
17. Дано натуральное число. Определить остаток от деления на 4 и вывести его в текстовом виде (ноль, один, два, три).
18. Дано неотрицательное число *k*, не превышающее десяти тысяч. Напечатать фразу «*k* ворон» русскими словами. (Пример: если *k*=23, то должно быть напечатано «двадцать три вороны»; если *k*=3651, то «три тысячи шестьсот пятьдесят одна ворона»).
19. Пусть элементами круга являются радиус (первый элемент), диаметр (второй элемент) и длина окружности (третий элемент). Составить программу, которая по номеру элемента запрашивала бы его соответствующее значение и вычисляла бы площадь круга.

ЛАБОРАТОРНАЯ РАБОТА № 4. ЦИКЛЫ

Операторы цикла используются для организации многократно повторяющихся вычислений. Любой цикл состоит из тела цикла, т.е. тех операторов, которые выполняются несколько раз, начальных установок, модификации параметра цикла и проверки условия выполнения цикла.

Один проход цикла называется итерацией. Проверка условия выполняется на каждой итерации либо до тела цикла (цикл с предусловием), либо после тела цикла (цикл с постусловием). Тело цикла с постусловием всегда выполняется хотя бы один раз, после чего проверяется необходимость следующего его выполнения. Проверка необходимости выполнения цикла с предусловием делается до тела цикла, поэтому, возможно, что он не выполнится ни разу.

Переменные, изменяющиеся в теле цикла и используемые при проверке условия выполнения, называются параметрами цикла. Целочисленные параметры цикла, изменяющиеся с постоянным шагом при каждой итерации, называются счетчиками цикла.

Начальные установки могут явно не присутствовать в программе, их смысл состоит в том, чтобы до входа в цикл задать значения переменным, которые в нем используются.

Цикл с предусловием (while)

Синтаксис оператора *while* выглядит следующим образом:

while (выражение) оператор;

Выражение определяет условие повторения тела цикла, представленного простым или составным оператором. Выполнение оператора начинается с вычисления выражения. Если оно истинно (не равно 0), выполняется оператор, после чего управление возвращается заголовку цикла, и все повторяется снова. Когда условие оказывается ложным (равно 0), управление передается следующему после цикла оператору.

Распространенный прием программирования – организация бесконечного цикла с заголовком *while* (*true*) либо *while* (1) и принудительным выходом из тела цикла по выполнению какого-либо условия.

В круглых скобках после ключевого слова *while* можно вводить описание переменной. Областью ее действия является цикл:

```
while (int x = 0)
{
... /* область действия x */
}
```

Задача 4.1. Написать программу, которая выводит на экран таблицу умножения на введенное число.

```
int main()
{
int n;
printf("Input n->"); scanf("%d",&n);
int i=1;
while (i<=10)
{
  printf("\n%2d*%2d=%2d",i,n,i*n);
  i++;
}
getchar();getchar();
  return 0;
}
```

Цикл с постусловием (do while)

Цикл с постусловием имеет вид:

```
do
оператор
while выражение;
```

Сначала выполняется простой или составной оператор, составляющий тело цикла, а затем вычисляется выражение. Если оно истинно (не равно 0), тело цикла выполняется еще раз. Цикл завершается, когда выражение станет равным 0 или в теле цикла будет выполнен какой-либо оператор передачи управления. Тип выражения должен быть целым или приводимым к нему.

Задача 4.2. Написать программу для решения задачи № 4.1, используя цикл *do…while*.

```
int main()
{
int n;
printf("Input n->"); scanf("%d",&n);
int i=1;
do
{
  printf("\n%2d*%2d=%2d",i,n,i*n);
  i++;
}
```

```
while (i<=10);
getchar();getchar();
  return 0;
}
```

Наиболее общий способ организации цикла (for)

Цикл *for* имеет следующий формат:

for (выражение-1; выражение-2; выражение-3) тело;

Выражение-1 обычно используется для установления начального значения переменных, управляющих циклом. Выражение-2 – это выражение, определяющее условие, при котором тело цикла будет выполняться. Выражение-3 определяет изменение переменных, управляющих циклом после каждого выполнения тела цикла.

Схема выполнения оператора *for*:

1. Вычисляется выражение-1.

2. Вычисляется выражение-2.

3. Если значения выражения-2 отлично от нуля (истина), выполняется тело цикла, вычисляется выражение-3 и осуществляется переход к пункту 2, если выражение-2 равно нулю (ложь), то управление передается на оператор, следующий за оператором *for*.

Существенно то, что проверка условия всегда выполняется в начале цикла. Это значит, что тело цикла может ни разу не выполниться, если условие выполнения сразу будет ложным. Модификации выполняются после каждой итерации цикла и служат обычно для изменения параметров цикла. Простой или составной оператор представляет собой тело цикла. Любая из частей оператора *for* может быть опущена (но точки с запятой надо оставить на своих местах!).

Оператор *for* может быть заменен оператором *while* следующим образом:

```
выражение-1;
while (выражение-2)
{  тело
   выражение-3;
}
```

Цикл *for* удобен при организации циклов с параметром (переменной, которая при каждом входе в цикл принимает новое значение).

Задача 4.3. Написать программу для решения задачи № 4.1, используя цикл с параметром.

```
int main()
{
int n;
printf("Input n->"); scanf("%d",&n);
for (int i=1; i<=10;i++)
  printf("\n%2d*%2d=%2d",i,n,i*n);
getchar();getchar();
  return 0;
}
```

Часто встречающиеся ошибки при программировании циклов – использование в теле цикла неинициализированных переменных и неверная запись условия выхода из цикла. Чтобы избежать ошибок, рекомендуется:

- проверить, всем ли переменным, встречающимся в правой части операторов присваивания в теле цикла, присвоены до этого начальные значения (а также возможно ли выполнение других операторов);
- проверить, изменяется ли в цикле хотя бы одна переменная, входящая в условие выхода из цикла;
- предусмотреть аварийный выход из цикла по достижению некоторого количества итераций;
- если в теле цикла требуется выполнить более одного оператора, необходимо заключать их в фигурные скобки.

Индивидуальные задания по теме «Циклы»

Задание И4.1. Цикл с параметром

1. Дано натуральное число *N*. Вычислить $N!=1\cdot 2\cdot 3\cdot \ldots \cdot N$.
2. Дано натуральное число *N*. Вычислить:
   $\left(1+\frac{1}{1^2}\right)\left(1+\frac{1}{2^2}\right)\ldots\left(1+\frac{1}{N^2}\right)$.
3. Дано натуральное число *N*. Вычислить:
   $\frac{1}{2}+\frac{3}{4}+\frac{5}{6}+\ldots+\frac{2n-1}{2n}$.
4. Даны действительное число *a*, натуральное число *N*. Вычислить: *a(a+1)(a+2)(a+3)...(a+N-1).*
5. Даны действительное число *a*, натуральное число *N*.

Вычислить: $\frac{1}{a}+\frac{1}{a(a+1)}+\ldots+\frac{1}{a(a+1)\ldots(a+n)}$.

6. Даны действительное число *a*, натуральное число *N*.
   Вычислить: $\frac{1}{a}+\frac{1}{a^2}+\frac{1}{a^4}+\ldots+\frac{1}{a^{2n}}$.
7. Даны действительное число *a*, натуральное число *N*. Вычислить: $a(a-N)(a-2N)\ldots(a-N^2)$.
8. Для натурального *N* найти: $\sin x+\sin^{2!}x+\ldots+\sin^{N!}x$, где *x* – любое число.
9. Для натурального *N* найти: $1+\frac{1}{2!}+\frac{1}{3!}+\ldots+\frac{1}{N!}$.
10. Для натурального *N* найти: $\sin\frac{1}{N}+\sin\frac{1}{2N}+\ldots+\sin\frac{1}{N^2}$.
11. Для натурального *N* найти: $\frac{1!}{\cos N}+\frac{2!}{\cos N}+\ldots+\frac{N!}{\cos N}$.
12. Для натурального *N* найти:
    $\left(x^2+1\right)\cos x+\left(x^2+1\right)\cos 2x+\ldots+\left(x^2+1\right)\cos Nx$, где *x* – любое число.
13. Для натурального *N* найти: $\frac{a}{(1+1)!}+\frac{a}{(2+1)!}+\ldots+\frac{a}{(N+1)!}$, где *a* – любое число.
14. Вычислить $P=\left(1-\frac{1}{2^2}\right)\left(1-\frac{1}{3^2}\right)\ldots\left(1-\frac{1}{H^2}\right)$, *H>2*.
15. Вычислить *y=cos(1 + cos(2 + ... + cos(39 + cos(40))...)).*
16. Вычислить: $y=\cos x+\cos x^2+\cos x^3+\ldots+\cos x^n$.
17. Вычислить *y*, для *n>1*. *y=1! + 2! + 3! + ... + n!.*
18. Вычислить *y=sin1! + sin2! + sin3! + ... + sin10!.*
19. Для натурального *N* найти:
    $\frac{1}{\sin 1}+\frac{1}{\sin(1+2)}+\frac{1}{\sin(1+2+3)}+\ldots+\frac{1}{\sin(1+2+3+\ldots+N)}$.
20. Для натурального *N* вычислить:
    $\frac{1}{\sin 1}+\frac{1}{\sin 1+\sin 2}+\ldots+\frac{1}{\sin 1+\sin 2+\ldots+\sin N}$.

21.– 30. Составить программу для вычисления суммы ряда:

21. $\sum_{n=1}^{10} \frac{\sqrt[4]{n}}{n+1}$.

22. $\sum_{n=1}^{40} \frac{1}{\sqrt[3]{n}+n}$.

23. $\sum_{n=1}^{10} arctg\,\frac{1}{n}$.

24. $\sum_{n=1}^{15} n + e^{-2n}$.

25. $\sum_{n=1}^{40} \frac{\sqrt[3]{n+1}}{n}$.

26. $\sum_{n=1}^{25} \cos^2 \frac{\pi}{4 \cdot n}$.

27. $\sum_{n=1}^{10} \frac{1}{\sqrt[3]{n} \cdot (n+1) \cdot (n+2)^3}$.

28. $\sum_{n=1}^{40} \frac{1}{(2 \cdot n - 1) \cdot 2^{2 \cdot n - 1}}$.

29. $\sum_{n=1}^{100} \frac{1}{n \cdot (n+1)}$.

30. $\sum_{n=1}^{20} \frac{e^{-n}}{(2 \cdot n - 1)}$

Задание И4.2. Цикл с предусловием и с постусловием

1. Вычислить *y* – первое из чисел *sin(x)*, *sin(sin(x))*, *sin(sin(sin(x)))*, …, меньшее по модулю 10^{-4}.
2. Найти первый отрицательный элемент последовательности *cos(ctg(h))*, где *h = 1, 2, 3, …*
3. Дано действительное число *E (E>0)*. Необходимо вычислить следующую сумму: $\sum_{n=1}^{\infty} \frac{1}{3^n} \cos 3^{n-1}$. Следует учесть только те

слагаемые, в которых множитель $\frac{1}{3^n}$ имеет величину не меньшую, чем *E*.

4-10. Не используя стандартные функции, вычислить с точностью $E=10^{-4}$:

4. $y = sh(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots + \frac{x^{2n+1}}{(2n+1)!}$;

5. $y = e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!}$;

6. $y = \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} + \ldots + \frac{(-1)^{n-1} x^n}{n}$;

7. $y = arctg(x) = x - \frac{x^3}{3} + \frac{x^5}{5} + \ldots + \frac{(-1)^n x^{2n+1}}{2n+1}$;

8. $y = \sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots + \frac{(-1)^n x^{2n+1}}{(2n+1)!}$;

9. $y = arctg(-\frac{1}{x}) = -\frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + \ldots$, для *x=1, 2, 3, …*

10. $y = \cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots = \sum_{n=0}^{\infty} t_n(x)$,

11. Задано положительное число *E* ($E<10^{-2}$). Последовательность *a1, a2, …, an* образуется по следующему закону:

$$a_i = \left(1 - \frac{1}{2}\right) \cdot \left(1 - \frac{1}{3}\right) \cdot \ldots \cdot \left(1 - \frac{1}{i+1}\right).$$

Найти первый элемент последовательности, для которого выполнено условие $|a_n - a_{n-1}| < e$. Элементы последовательности вывести в 1 столбец. Задачу решить без применения массива.

12. Составить программу вычисления функции *ch(x)* по приближенной формуле $ch(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots + \frac{x^{2n+1}}{(2n+1)!}$ с точностью $E \geq \left| \frac{x^{2n+1}}{(2n+1)!} \right|$.
13. Составить программу для вычисления количества положительных значений функции $y = \sum_{i=1}^{10} \cos(5i^2 + 8x)$. При *x*, изменяющемся от 13 до 33 с шагом 1.5, используя цикл *do while*.
14. Дана последовательность, состоящая из дробей: $\frac{1}{1}, \frac{4}{2}, \frac{9}{4}, \frac{16}{8} \ldots$ Какое минимальное количество элементов последовательности нужно сложить, чтобы сумма превысила заданное число *S* (*S*>1)?
15. Найти первый отрицательный элемент последовательности $\sin(ctg(x_i))$, если x_1 – запрашивается, а $x_{i+1} = x_i + 0.3$.
16. Числа Фибоначчи определяются по формулам: $f_0 = 0,\ f_1 = 1,\ f_k = f_{k-2} + f_{k-1}$. Найти первое число Фибоначчи, большее *m* (*m* >0).
17. Запрашивается число *n*. Логической переменной *t* присвоить значение true, если в последовательности $\sin(x^k)$ (*k*=1, 2, 3, … *n*) есть хотя бы одно отрицательное число, иначе *false*.
18. Числа Фибоначчи определяются по формулам: $f_0 = 0,\ f_1 = 1,\ f_k = f_{k-2} + f_{k-1}$. Вычислить сумму всех чисел Фибоначчи, которые не превосходят *M* (*M*>0).
19. Дана последовательность целых чисел (не менее 100). Элементы последовательности генерируются случайным образом в диапазоне [-30; 70]. Вычислить сумму элементов последовательности, предшествующих «особому» элементу, то есть элементу, который больше своего соседа справа. Задача решается без применения массива.
20. Даны положительные действительные числа *A*, *X*, *E*. В последовательности *y1*, *y2*, …, образованной по закону *y0*=*A*;

$y_i = \dfrac{y_{i-1} + \dfrac{X}{y_{i-1}}}{2}$, *i*=1, 2, 3, …, найти первый элемент y_n, для которого выполнено неравенство $\left| {y_n}^2 - {y_{n-1}}^2 \right| < E$.

21. Число π вычисляется по формуле Грегори следующим образом: $\pi = 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots\right)$, причем, чем больше слагаемых в скобках, тем выше точность вычисления числа π. Определить минимальное количество слагаемых для вычисления π с точностью 0.001.
22. Дана последовательность, состоящая из дробей: $\frac{1}{1}, \frac{4}{2}, \frac{7}{3}, \frac{10}{4} \ldots$ Какое минимальное количество элементов последовательности нужно сложить, чтобы сумма превысила заданное число *S* (*S*>1)?
23. Дана последовательность, состоящая из дробей: $\frac{1}{1}, \frac{3}{2}, \frac{5}{3}, \frac{7}{4} \ldots$ Какое минимальное количество элементов последовательности нужно сложить, чтобы сумма превысила заданное число *S* (*S*>1)?
24. Дана последовательность, состоящая из дробей: $\frac{1}{2}, \frac{3}{4}, \frac{5}{6}, \frac{7}{8} \ldots$ Какое минимальное количество элементов последовательности нужно сложить, чтобы сумма превысила заданное число *S* (*S*>1)?
25. Найти первый отрицательный элемент последовательности $sin(x^k)$, где *к*=1, 2, 3, ….

Задание И4.3. Дополнительные задачи

1. Дано натуральное число *n*. Найти сумму первой и последней цифры этого числа.
2. Дано натуральное число *n*. Переставив местами первую и последнюю цифры этого числа, получить новое число.
3. Даны два натуральных числа *m* и *n* ($m \le 9999, n \le 9999$). Проверить, есть ли в записи числа *m* цифры, совпадающие с цифрами в записи числа *n*.

4. Даны натуральные числа *n, k*. Проверить, есть ли в записи числа n^k (не более 10^9) цифра *m*.
5. Среди всех *n*-значных чисел указать те, сумма цифр которых равна данному числу *k*.
6. Найти наибольшую и наименьшую цифры в записи данного натурального числа.
7. Найти на отрезке *[n, m]* минимальное натуральное число, имеющее наибольшее количество делителей.
8. Дано целое число в диапазоне от 0 до 1000000. Проверить присутствуют ли одновременно в записи числа цифры 1 и 5.
9. Дано целое число в диапазоне от 0 до 1000000. Проверить отсутствуют ли одновременно в записи числа цифры 2 и 6.
10. Дано целое число в диапазоне от 0 до 1000000. Проверить присутствуют ли в записи числа хотя бы одна цифра кратная трем.
11. Дано целое число в диапазоне от 0 до 1000000. Проверить отсутствуют ли в записи числа четные цифры.
12. Найти произведение цифр заданного *k*-значного числа.
13. Дано произвольное целое положительное число *K* ($K \le 10^9$). Вывести цифры этого числа в порядке неубывания (например, 546085 => 045568).
14. Дано произвольное целое положительное число *K* ($K \le 10^9$). Вывести цифры этого числа в порядке невозрастания (например, 546085 => 865540).
15. Дано произвольное целое положительное число *K* ($K \le 10^9$). Вывести цифры этого числа в обратном порядке (например, 5485 => 5845).
16. Дано произвольное целое положительное число *K* ($K \le 10^9$). Вывести новое число, полученное из K вычеркиванием всех четных цифр (например, 234583 => 353).
17. Дано произвольное целое положительное число *K* ($K \le 10^9$). Вывести это число без первой и последней цифры (например, 234583 => 3458).
18. Дано произвольное целое положительное число *K* ($K \le 10^9$). Найти сумму всех четных цифр этого числа.
19. Дано произвольное целое положительное число *K* ($K \le 10^9$). Найти сумму всех нечетных цифр этого числа.
20. Дано произвольное целое положительное число *K* ($K \le 10^9$). Найти произведение всех цифр этого числа, которые кратны трем.

21. Дано произвольное целое положительное число *K* ($K \le 10^9$). Найти сумму всех цифр этого числа, больших заданного *T* ($0 \le T \le 9$).
22. Дано произвольное целое положительное число *K* ($K \le 10^9$). Найти количество нулей в цифровой записи числа.
23. Дано произвольное целое положительное число *K* ($K \le 10^9$). Найти количество цифр в цифровой записи данного числа, которые имеют наименьшее значение (например, 956569 – количество цифр с наименьшим значением равно 2 – две цифры 5).
24. Дано произвольное целое положительное число *K* ($K \le 10^9$). Найти количество цифр в *K* равных *T* ($0 \le T \le 9$).
25. Дано произвольное целое положительное число *K* ($K \le 10^9$). Вывести новое число, полученное из *K* путем замены последней цифры на значение наибольшей цифры (например, 2854353 => 2354358).
26. Дано произвольное целое положительное число *K* ($K \le 10^9$). Вывести новое число, полученное из K вычеркиванием всех единиц (например, 2134513 => 23453).

Задание И4.4. Сложные задачи

1. Дана лента билетов, содержащих 6-разрядные номера от *N* до *M*. Определить минимальное количество билетов, расположенных между 2-я «счастливыми» билетами. («Счастливым» билетом считать билет, сумма первых 3-х цифр которого равна сумме 3-х последних цифр.)
2. Дана лента билетов, содержащих 6-разрядные номера от *N* до *M*. Определить максимальное количество билетов, расположенных между 2-я «счастливыми» билетами. («Счастливым» билетом считать билет, сумма первых 3-х цифр которого равна сумме 3-х последних цифр.)
3. Дана лента билетов, содержащих 6-разрядные номера от *N* до *M*. Определить количество билетов, расположенных между первым и последним «счастливыми» билетами. («Счастливым» билетом считать билет, сумма первых 3-х цифр которого равна сумме 3-х последних цифр.)
4. Дана лента билетов, содержащих 6-разрядные номера от *N* до *M*. Определить количество «счастливых» билетов. («Счастливым» билетом считать билет, сумма первых 3-х цифр которого равна сумме 3-х последних цифр.)
5. Дана лента билетов, содержащих 6-разрядные номера от *N* до *M*. Определить минимальное количество билетов, располо-

женных между 2-я билетами с номерами – палиндромами. (Палиндромом является симметричное число, например: 234432 или 254452.)

6. Дана лента билетов, содержащих 6-разрядные номера от *N* до *M*. Определить максимальное количество билетов, расположенных между 2-я билетами с номерами – палиндромами. (Палиндромом является симметричное число, например: 234432 или 254452.)

Задание И4.5. Задачи на моделирование циклических процессов

1. На заводе собирают прибор из трех блоков. Известно, что среди блоков первого типа встречаются 2% со скрытыми дефектами, среди блоков второго и третьего типа – соответственно 3% и 5% дефектных. С использованием генератора случайных чисел промоделировать сборку 1000 деталей и определить, сколько будет собрано приборов без брака.
2. Шахматист *A* в среднем на каждые 100 партий выигрывает у шахматиста *B* на 6 партий больше, чем проигрывает, а доля ничьих равна 80%. С использованием генератора случайных чисел промоделировать матч из 24 партий. С каким результатом он закончится?
3. Составить программу, позволяющую промоделировать опрос 100 человек и на его основании выяснить: переименовать село Папинск в село Маминск или оставить за ним прежнее название. Примечание: Программа должна генерировать ответы самостоятельно с использованием генератора случайных чисел.
4. Имеется две спичечные коробки, в каждой из которых находится по 10 спичек. Случайным образом выбирается коробка и из нее достается одна спичка. Процесс продолжается до тех пор, пока одна из коробок не опустеет. С использованием генератора случайных чисел промоделировать этот процесс, и ответить на вопрос: сколько спичек будет всего сожжено?
5. В детском саду имеется группа детей из 20 человек. Каждому ребенку на утреннике Дед Мороз случайным образом дарит одну из следующих домашних игрушек: зайца, мяч или куклу. С использованием генератора случайных чисел промоделировать этот процесс, и ответить на вопрос: сколько игрушек каждого вида было подарено?
6. Робот находится в центре окружности радиусом 3,5 метра и в каждый момент времени делает шаг (длиной 1 м) в случайном направлении: на север, на юг, на восток или на запад. С ис-

пользованием генератора случайных чисел промоделировать этот процесс, и ответить на вопрос: хватит ли 12 шагов, чтобы выйти за пределы окружности?

7. Известно, что в среднем из 100 выстрелов солдат А поражает мишень 75 раз, а солдат В – 80 раз. С использованием генератора случайных чисел промоделировать соревнование между ними, в котором каждому нужно попасть в цель по 10 раз. Кто быстрее поразит все мишени?
8. В среднем из 128 компьютеров в течении месяца на одном выходит из строя дисплей. За тот же период на одной из 67 ЭВМ происходит поломка дисковода и на двух из 53 машин происходит крах системы из-за заражения вирусом. С использованием генератора случайных чисел смоделировать работу дисплейного класса из 13 компьютеров за один месяц и ответить на вопрос: каково общее количество поломок за этот период?
9. Гусеница ползет по куску резины длиной 7 см, стремясь достичь противоположного конца. Ползет она со скоростью 1 см/мин. Кусок резины может растягиваться до любой длины. Через минуты вы вытягиваете резину так, чтобы она удлинилась вдвое (т.е. стала 14 см в длину). Гусеница прочно держится на поверхности и продолжает двигаться, когда вы тянете резину. Она ползет все с той же скоростью. Еще через минуту вы снова вытягиваете резину так, чтобы ее первоначальная дина утраивается (т.е. она становится равной 21 см). Гусеница продолжает ползти, а вы каждую минуту продолжаете тянуть резину, в четвертый раз уже удлинив ее в четыре раза. Доберется ли гусеница когда-нибудь до противоположного конца? Если да, то когда?
10. Некий мужчина отправляется на службу, которая находится на расстоянии 1 км от дома. Дойдя до места работы, он вдруг вспоминает, что забыл поцеловать жену, и поворачивает назад. Пройдя полпути, он меняет решение, посчитав, что правильнее вернуться на работу. Пройдя 1/3 км по направлению к конторе, он вдруг осознает, что будет настоящем подлецом, если так и не поцелует жену. На этот раз, прежде чем снова изменить мнение он проходит 1/4 км. Так он продолжает метаться, и после N-го этапа, пройдя 1/N км, снова меняет решение. Это продолжается пока расстояние 1/N превышает длину шага, после чего человек останавливается. Определить координату точки остановки.

11. Кролики питаются дарами земли, а лисы поедают кроликов. Изменение в численности популяции кроликов и лис описывается дифференциальным уравнением:

$$\frac{\partial R}{\partial t} = \mathrm{A}\cdot\mathrm{R} - \mathrm{B}\cdot\mathrm{R}^2 - \mathrm{C}\cdot\mathrm{R}\cdot\mathrm{F};$$

$$\frac{\partial F}{\partial t} = -D\cdot F + \mathrm{E}\cdot\mathrm{R}\cdot\mathrm{F},$$

где *R* – количество кроликов;
F – количество лис;
AR – определяет рождаемость кроликов;
BR^2 – смертность кроликов естественным путем;
CRF – смертность кроликов за счет съедания лисами;
DF – естественная смертность лис;
ERF – темп рождаемости лис.
Рассмотреть контрольный пример при A=0,04, B=0,00005, C=0,002, D=0,03 и E=0,0002. Отобразить изменение численности кроликов и лис по месяцам в течение года, выбрав начальное количество кроликов 340 и лис 35.

12. Составить программу для построения траектории мяча, который начинает двигаться из левого верхнего угла экрана. Уравнение движения имеет вид: $\frac{\partial \mathrm{V}}{\partial \mathrm{t}} = \mathrm{G} - \mathrm{k}\cdot\mathrm{V}$,
где *V* – функция изменения скорости;
k – коэффициент сопротивления воздуха;
G – ускорение свободного падения.
Построить траекторию движения мяча. Проследить влияние *k* и начальной скорости движения.

13. В магазине стоит очередь из *N* человек. Время обслуживания *i*-го покупателя t_i – случайная величина, распределенная по закону равномерной плотности в интервале [2,5; 10,4].
Получить $t_1, t_2, \ldots, t_N$ – времена пребывания в очереди каждого покупателя. Указать номер того человека, для обслуживания которого потребовалось минимальное время.

14. Мощность радиоактивного излучения пропорциональна концентрации радиоактивного вещества. Период полураспада одного из изотопов углерода составляет 8 дней. В начальный момент времени мощность излучения составляет 2 рентгена/час. Определить, через сколько дней мощность излучения снизится до величины 0,15 рентгена/час.
Изменение концентрации и, следовательно, мощности излучения описывается формулой:

$Q = Q_0 e^{-\lambda t}$,

где Q_0 – начальная мощность;

$$\lambda = \frac{\ln 2}{P};$$

P – период полураспада.

Для решения задачи необходимо изменять *t* с заданным шагом, вычислять Q и сравнивать его с допустимой величиной.

15. Известно, что в 1 г живой клетчатки (например, дерева) содержится $7.5 \cdot 10^{10}$ ядер радиоактивного углерода. После гибели организма (дерева) их концентрация уменьшается по закону $N = N_0 e^{-\lambda t}$, где N_0 – начальная концентрация;

$$\lambda = \frac{\ln 2}{t};$$

T – период полураспада (для радиоактивного углерода равен 5570 годам).

Построить таблицу зависимости концентрации радиоактивных ядер от времени для интервала времени от 0 до 6000 лет с шагом 500 лет, считая за 0 момент гибели организма (дерева).

16. Плотность воздуха убывает с высотой по закону $\rho = \rho_0 e^{-hz}$. Считая, что $\rho_0 = 1.29 кг/м^3$, $z = 1.25 \cdot 10^{-4} 1/м$, напечатать таблицу зависимости плотности от высоты значений от 0 до 1000 м с шагом 100 м. Определить, на какой высоте плотность будет меньше 1 кг/м^3.
17. Концентрация хлорной извести в бассейне объемом *V* м^3 составляет 10 г/л. Через одну трубу в бассейн вливают чистую воду с объемной скоростью Q м^3/час, через другую трубу с такой же скоростью вода выливается. При условии идеального перемешивания концентрация хлорной извести изменяется по закону

$$C = C_0 e^{-Qt/V},$$

где *t* – время;

C_0 – начальная концентрация.

Определить, через какое время концентрация хлорной извести достигнет безопасной для человека величины 0,1 г/л. Задачу решить при Q=150 м^3/час, *V*=10000 л, C_0=10 г/л.

18. Два спортсмена одновременно начинают движение из одной точки. Первый спортсмен начинает движение со скоростью 10 км/час и в начале каждого следующего часа увеличивает скорость на 1 км/час. Второй начинает движение со скоростью 9 км/час и в начале каждого следующего часа увеличивает скорость на 1,6 км/час. Выяснить какой спортсмен преодолеет больший путь через 1 час; через 4 часа. Определить, когда второй спортсмен догонит первого.

ЛАБОРАТОРНАЯ РАБОТА №5.
РАБОТА С ОДНОМЕРНЫМИ МАССИВАМИ

Создание и заполнение массивов

Массивы – это группа элементов одинакового типа (double, float, int и т.п.). Из объявления массива компилятор должен получить информацию о типе элементов массива и их количестве. Объявление массива имеет два формата:

спецификатор-типа описатель [константное_выражение];
спецификатор-типа описатель [ ];

Описатель – это идентификатор массива.

Спецификатор-типа задает тип элементов объявляемого массива. Элементами массива не могут быть функции и элементы типа void.

Константное выражение в квадратных скобках задает количество элементов массива. Константное выражение при объявлении массива может быть опущено в следующих случаях:

- при объявлении массив инициализируется,
- массив объявлен как формальный параметр функции,
- массив объявлен как ссылка на массив, явно определенный в другом файле проекта.

В языке СИ определены только одномерные или линейные массивы, т.е. за каждым элементом массива закреплен только один его порядковый номер.

Описывать массив *int A[10]* – это значит выделить 10 последовательных элементов в памяти (каждый размером *sizeof*(*int*)) для массива с именем *A*, элементы которого имеют тип *int*. Каждый элемент массива в общем виде описывается как *A[i],*
где *A* – имя массива,

i – номер или индекс массива (*0<=i< n*)

A[i] – значение элемента массива.

Отметим, что в языке СИ первый элемент массива имеет индекс равный 0.

Пример описания массива:
double b[10]; / вектор из 10 элементов, имеющих тип double */*

После объявления массива каждый его элемент можно обработать, указав идентификатор (имя) массива и индекс элемента в квадратных скобках. Например, запись *Mas[2], VectorZ[10]* позволяет обратиться ко второму элементу массива *Mas* и десятому элементу массива *VectorZ*.

Задача 5.1. Заполнить массив 20 целыми числами. Элементы массива вывести на экран монитора.

Листинг программы

```
#include <stdio.h>
#include <mem.h>
int main()
{
 //Заполнение одномерного массива
 int a[20];
 //Input
 for(int i=0;i<20;i++)
 {
  printf("Input a[%d] ", i);
  scanf("%d", &a[i]);
 }
 //Вывод элементов массива в линию
 for(int i=0;i<20;i++)
  printf("%d ", a[i]);
 printf("\n");
//Вывод элементов массива в столбец
 for(int i=0;i<20;i++)
  printf("%d\n", a[i]);
  getchar();getchar();
 return 0;
}
```

Вариант работы программы с массивом из 3 элементов целого типа представлен на рис. 5.1.

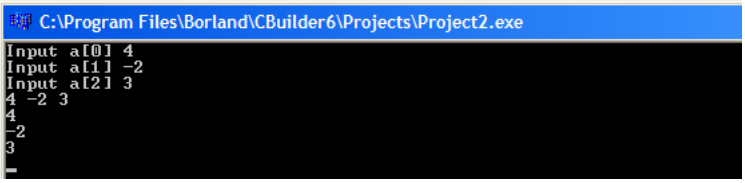


Рис. 5.1.

Заполнение одномерного массива с помощью генератора случайных чисел

Функция *random*(100) генерирует случайные числа от 0 до 99 целого типа.

Примеры:

a[i]=random(100) //Генерируются числа от 0 до 99 (100 не входит в диапазон) целого типа

a[i]=random(100)-50 //Генерируются целые числа в диапазоне от -50 до 49

Для генерации целых в диапазоне *[A, B]* можно использовать следующее выражение: *random(B-A+1)+A,* а для генерации вещественных –

(double)rand()/RAND_MAX(B-A)+A.*

Для того чтобы при каждом запуске программа генерировала новые значения, необходимо добавить вызов функции *randomize ()*.

Задача 5.2. Заполнить массив N целыми числами от –15 до 38. Элементы массива вывести в строку.

Листинг программы

```
#include <stdio.h>
int main()
{ int a[100];
 randomize();
 int k;
 printf("Vvedite razmer massiva=");
 scanf("%d",&k);
 //Заполнение массива
 for(int i=0;i<k;i++)
  a[i]=random(38-(-15)+1)-15;
//Вывод элементов массива в строку
 for(int i=0;i<k;i++)
  printf("a[%d]=%d ", i,a[i]);
 printf("\n");
  getchar();getchar();
   return 0;
}
```

Нахождение суммы (или произведения) элементов

Задача 5.3. Заполнить массив 15 целыми случайными числами от –10 до 15. Найти сумму четных чисел.

Листинг программы

```
#include <stdio.h>
int main()
{ int a[15];
 randomize();
 int s;
   s=0;
 for(int i=0;i<15;i++)
 {
   a[i]=random(15-(-10)+1)-10;
   printf("a[%d]=%d ", i,a[i]);
   if (a[i]%2==0) s=s+a[i];
 }
```

```
 printf("\n");
 printf("s=%d", s);
  getchar();getchar();
   return 0;
}
```

Результаты работы программы приведены на рис. 5.2.

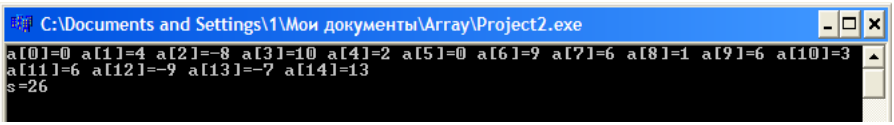


Рис. 5.2.

Задача 5.4. Дан массив целых чисел. Найти минимальный элемент и его номер. Вывести элементы массива в порядке убывания.

Листинг программы

```
#include <stdio.h>
int main()
{
 const n=15;
 int a[n]={3,5,2,8,1,-5,4,-2,-13,-8,2,3,1,-3,4};
 printf("First massiv\n");
 // Вывод исходного массива
  for (int i=0; i<n; i++)
  printf("a[%d]=%4d  ", i,a[i]);
  printf("\n");
 // Поиск минимального числа
 int MinValue=a[0];
 for (int i=1; i<n; i++)
  if (a[i]<MinValue) MinValue=a[i];
 printf("Min=%d\n", MinValue);
 // Поиск индекса минимального числа
 char MinIndex=0;
 for (int i=1; i<n; i++)
  if (a[i]<a[MinIndex]) MinIndex=i;
 printf("Min=%d\n", MinIndex);
 // Сортировка
 int tmp;
 for (int i=0; i<n-1; i++)
  for (int j=i+1; j<n; j++)
 if (a[i]>a[j])
 {
  tmp=a[i]; a[i]=a[j]; a[j]=tmp;
 }
   printf("Massiv after sortirovka\n");
 for (int i=0; i<n; i++)
```

```
 printf("%4d", a[i]);
 getchar();
 return 0;
}
```

Результаты работы программы приведены на рис. 5.3.

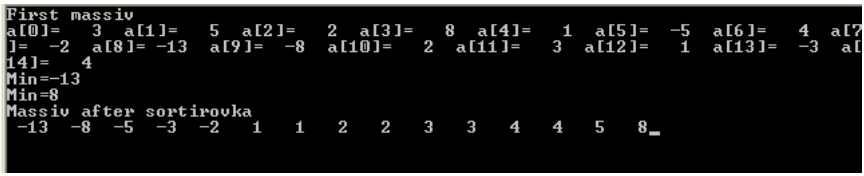


Рис. 5.3

Задание для индивидуального выполнения по теме «Работа с одномерными массивами»

Задание И5.1. При решении всех задач использовать структурированный тип – массив.

1. Сгенерировать элементы массива случайным образом в заданном диапазоне *[a; b]* согласно варианту.

1.	a = -10	b = 7	*2.*	a = -15	b = 88
3.	a = -15	b = 37	*4.*	a = -18	b = 23
5.	a = 25	b = 57	*6.*	a = -37	b = 56
7.	a = -65	b = 2	*8.*	a = -22	b = 132
9.	a = -100	b = 107	*10.*	a = -8	b = 45
11.	a = -19	b = 37	*12.*	a = -105	b = 66
13.	a = -48	b = 32	*14.*	a = -25	b = 87
15.	a = -13	b = 10	*16.*	a = -78	b = 43
17.	a = -16	b = 21	*18.*	a = -9	b = 60
19.	a = -45	b = 52	*20.*	a = -3	b = 29
21.	a = -1	b = 48	*22.*	a = -74	b = 39
23.	a = -23	b = 20	*24.*	a = -10	b = 29
25.	a = -12	b = 4	*26.*	a = 20	b = 44
27.	a = -66	b = 33	*28.*	a = -11	b = 31
29.	a = -5	b = 5	*30.*	a = -23	b = 35
31.	a = -17	b = 16			

2. Дана последовательность целых чисел $a_0, ..., a_{50}$. Получить сумму тех чисел данной последовательности, которые:
   - кратны 5;
   - нечетны и отрицательны;
   - удовлетворяют условию $a_i < i^2$.
3. Даны натуральное число n и последовательность целых чисел $a_0, ..., a_n$. Найти количество и сумму тех элементов данной последовательности, которые делятся на 5 и не делятся на 7.
4. Даны натуральные числа n, p и последовательность целых чисел $a_0, ..., a_n$. Получить произведение элементов последовательности $a_0, ..., a_n$, кратных p.
5. Даны натуральные p, q и последовательность целых чисел $a_0, ..., a_{67}$ ($p > q \geq 0$). В последовательности $a_0, ..., a_{67}$ заменить на нуль все элементы, модуль которых при делении на p дает в остатке q.
6. Даны натуральное число n, последовательность действительных чисел $a_0, ..., a_n$. Получить удвоенную сумму всех положительных элементов последовательности $a_0, ..., a_n$.
7. Даны натуральное число n, последовательность натуральных чисел $a_0, ..., a_n$. Вычислить обратную величину произведения тех элементов ai последовательности $a_0, ..., a_n$, для которых выполнено $i+1<a_i<i!$.
8. Даны натуральное число n, последовательность действительных чисел $a_0, ..., a_n$. В последовательности $a_0, ..., a_n$ все отрицательные элементы увеличить на 0,5, а все неотрицательные заменить на 0,1.
9. Даны натуральное число n, последовательность действительных чисел $a_0, ..., a_n$. В последовательности $a_0, ..., a_n$ все элементы меньше 2, заменить нулями. Получить сумму элементов, принадлежащих отрезку [3; 7], а также число таких элементов.
10. Даны натуральное число n, последовательность действительных чисел $a_0, ..., a_n$. В последовательности $a_0, ..., a_n$ все неотрицательные элементы, не принадлежащие отрезку [1; 2], заменить на единицу. Получить число отрицательных элементов и число элементов, принадлежащих отрезку [1;2].
11. Даны натуральное число n, целые числа $a_0, ..., a_n$. Получить сумму положительных и число отрицательных четных элементов последовательности $a_0, ..., a_n$.
12. Даны натуральное число n, последовательность целых чисел $a_0, ..., a_n$. Заменить все большие семи элементы последова-

тельности $a_0, ..., a_n$ числом 7. Вычислить количество таких элементов, которые имеют четный индекс.

13. Дана последовательность целых чисел $a_0, ..., a_{45}$. Получить число отрицательных элементов последовательности $a_0, ..., a_{35}$ и число нулевых элементов всей последовательности.
14. Даны натуральное число n, целое a и последовательность целых чисел $x_0, ..., x_n$. Если в последовательности $x_0, ..., x_n$ есть хотя бы один элемент, равный a, то получить сумму всех элементов, следующих за первым таким элементом, в противном случае ответом должно быть число – 10.
15. Даны целые числа $a_0, ..., a_{50}$. Получить последовательность $b_0, ..., b_{50}$, которая отличается от исходной тем, что все нечетные элементы удвоены.
16. Даны натуральное число n, действительное числа r и последовательность действительных чисел $a_0, ..., a_n$ ($n \geq 2$). Сколько среди точек $(a_0, a_n), (a_2, a_{n-1}), ..., (a_n, a_0)$ таких, которые принадлежат кругу радиуса r с центром в начале координат.
17. Дан одномерный массив. Переставить в обратном порядке элементы массива, расположенные между первым минимальным и первым максимальным элементами.
18. Дан массив целых чисел, состоящих из 20 элементов. Найти:
    - сумму элементов, имеющих нечетное значение.
    - вывести индексы тех элементов, значения которых больше заданного числа A.
    - определить, есть ли в заданном массиве положительные элементы кратные K (K вводится с клавиатуры). Если таковых нет, то вывести сообщение.
19. Дан массив целых чисел. Найти, сколько в нем пар одинаковых соседних элементов.
20. Дан массив целых чисел, состоящий из 25 элементов. Найти:
    - сумму элементов, имеющих нечетные индексы.
    - подсчитать количество элементов массива, значения которых больше заданного числа A и кратны 5.
    - найти номер первого отрицательного элемента, делящегося на 5 с остатком.
21. Дан массив чисел (не менее 100 элементов). Найти сумму 5 наименьших элементов.
22. Заданы N целых чисел. Определить, сколько среди них четных и притом делящихся на 7 без остатка. Вывести эти числа.
23. Написать программу, которая перемещает последний элемент массива, вставляя его, после первого отрицательного элемента этого же массива.

24. Вставить число 13 после всех элементов массива, кратных 13.
25. Дан массив целых чисел. Найти сумму 7 наибольших элементов.
26. Заданы действительное число *X* и массив *A[n]*. Определить элемент, который по значению ближайший к числу *X*. Напечатать элемент и его номер.

Задание И5.2

1. Дан массив из 25 вещественных чисел. Записать в этот же массив сначала все положительные числа и нули, а затем все отрицательные, сохраняя порядок их следования.
2. Дана последовательность действительных чисел $A_0, \ldots, A_{50}$. Получить последовательность $B=\{A_2, A_3, \ldots, A_{50}, A_0\}$.
3. Дан массив действительных чисел *A[n]*. Получить массив *B[n]*, где $B=\{A_1, A_1+A_2, A_1+A_2+A_3, \ldots, A_1+A_2+\ldots+A_n\}$.
4. Дана последовательность целых чисел $A_0, \ldots, A_{50}$. Получить последовательность $B_0, \ldots, B_{50}$, которая отличается от исходной тем, что все нечетные элементы удвоены.
5. Даны натуральное число *n*, действительные числа $X_0, \ldots, X_n$. В последовательности $X_0, \ldots, X_n$ все элементы, которые меньше двух заменить нулями. Подсчитать их количество.
6. Даны натуральное число *n*, действительные числа $A_0, \ldots, A_n$. В последовательности $A_0, \ldots, A_n$ все отрицательные элементы увеличить на 0,5, а все неотрицательные заменить на 0,1.
7. Даны целые числа $A, N, X_0, \ldots, X_n$. Определить, каким по счету идёт в последовательности $X_0, \ldots, X_n$ элемент, равный *A*. Если такого элемента нет, то ответом должно быть число 0.
8. Дан массив действительных чисел *A[30]*. Получить новый массив $D[15]=\{A_1 \cdot A_{16}, A_2 \cdot A_{17}, A_3 \cdot A_{18}, \ldots, A_{15} \cdot A_{29}\}$.
9. Задан целочисленный массив *B[20]*. Определить максимальный и минимальный элементы массива и их порядковые номера.
10. Задан массив *B[15]*. Найти максимальный и минимальный элементы массива и поменять их местами.
11. Определить сумму элементов массива *A[25]*, значения которых кратны трем и подсчитать их количество.
12. Для заданной последовательности *B* из *n* элементов найти сумму $C=\sum_{k=1}^{n}\frac{(B_k+1)}{K}$.

13. Для заданной последовательности *X* из *n* элементов определить отношение A/B, где $A=\prod_{k=1}^{n} X_k,\ B=\sum_{k=1}^{n} X_k$.
14. Найти сумму 1-го, 4-го, 9-го, 16-го, …, *k*-го элементов одномерного массива *X* из *n* элементов (*k* – наибольшее целое число, не превышающее $\sqrt{n}$).
15. При заданных элементах $X_1, X_2, \ldots, X_n$ и четном *n* найти значения сумм $C=\sum_{i=1}^{\frac{n}{2}} X_i,\ D=\sum_{i=\frac{n}{2}+1}^{n} X_i$.
16. Найти произведение, сомножители которого представлены каждым третьим элементом: X_1, X_4, X_7, и т. д. – заданной последовательности $X_1, X_2, \ldots, X_n$.
17. При заданных элементах $X_1, X_2, \ldots, X_n$ и четном *n* найти разность сумм $C1=\sum_{i=1}^{\frac{n}{2}} X_{(2\cdot i-1)},\ C2=\sum_{i=1}^{\frac{n}{2}} X_{2\cdot i}$.
18. При заданных координатах $C_1, C_2, \ldots, C_n$ одной и координатах $B_1, B_2, \ldots, B_n$ другой точки *n*-мерного пространства найти расстояние между ними по формуле: $R=\sqrt{(C_1-B_1)^2+\ldots+(C_n-B_n)^2}$.
19. Дано натуральное число *n*. Получить последовательность $B_1,\ldots, B_n$, где значение $B_i = i!$.
20. При заданных коэффициентах $A_1, A_2, \ldots, A_n$ и заданном значении *X* вычислить значение многочлена $A_1X+A_2X^2+\ldots+A_nX^n$.
21. При заданных элементах массива $X: X_0, X_1, \ldots, X_{n-1}, X_n$ найти по формуле $Y_k=X_0X_{k+1}$ последовательность элементов массива *Y*: $Y_0, Y_1, \ldots, Y_{n-1}$.
22. Из последовательности *X* из *n* элементов сформировать последовательность *T* по правилу: $T_1=X_1;\ T_2=\sqrt{|X_2|};\ T_3=\sqrt[3]{|X_3|};\ \ldots;\ T_n=\sqrt[n]{|X_n|}$.
23. Дана последовательность чисел среди которых имеется один нуль. Вывести на печать все числа, включительно, до нуля.

24. Дан целочисленный массив с количеством элементов *N*. Напечатать те элементы, индексы которых являются степенями двойки.
25. Даны координаты *N* точек на плоскости. Найти номера пары точек, расстояние, между которыми наибольшее.
26. Дан одномерный массив *A[n].*
    Найти:$S=max(A_0, A_2, \ldots, A_{2k}, \ldots) + min(A_1, A_3, \ldots, A_{2k+1}, \ldots)$.

ЛАБОРАТОРНАЯ РАБОТА № 6. МНОГОМЕРНЫЕ МАССИВЫ

Поскольку элементом массива может быть массив, можно определить и многомерные массивы. Они формализуются списком константных-выражений, следующих за идентификатором массива, причем каждое константное-выражение заключается в свои квадратные скобки.

Каждое константное-выражение в квадратных скобках определяет число элементов по данному измерению массива, так что объявление двухмерного массива содержит два константных-выражения, трехмерного – три и т.д.

Примеры:

```
 int a[2][3]; /* представлено в виде матрицы
    a[0][0] a[0][1] a[0][2]
    a[1][0] a[1][1] a[1][2] */
int w[3][3] = { { 2, 3, 4 },
                { 3, 4, 8 },
                { 1, 0, 9 } };
```

В последнем примере объявлен массив *w[3][3].* Списки, выделенные в фигурные скобки, соответствуют строкам массива, в случае отсутствия скобок инициализация будет выполнена неправильно.

В языке СИ можно использовать сечения массива, однако на использование сечений накладывается ряд ограничений. Сечения формируются вследствие опускания одной или нескольких пар квадратных скобок. Пары квадратных скобок можно отбрасывать только справа налево и строго последовательно. Сечения массивов используются при организации вычислительного процесса в функциях языка СИ, разрабатываемых пользователем.

Примеры:

int s[2][3];

Если при обращении к некоторой функции написать *s[0],* то будет передаваться нулевая строка массива *s*.

int b[2][3][4];

При обращении к массиву *b* можно написать, например, *b[1][2]* и будет передаваться вектор из четырех элементов, а обращение *b[1]* даст двухмерный массив размером 3 на 4. Нельзя написать *b[2][4],* подразумевая, что передаваться будет вектор, потому что это не соответствует ограничению, наложенному на использование сечений массива.

Пример объявления символьного массива.

char str[] = "объявление символьного массива";

Следует учитывать, что в символьном литерале находится на один элемент больше, так как последний из элементов является управляющей последовательностью *'\0'*.

Индексное выражение задает элемент массива и имеет вид:

выражение-1 [выражение-2]

Тип индексного выражения является типом элементов массива, а значение представляет величину, адрес которой вычисляется с помощью значений *выражение-1* и *выражение-2*.

Обычно *выражение-1* – это указатель, например, идентификатор массива, а *выражение-2* – это целая величина. Однако требуется только, чтобы одно из выражений было указателем, а второе целочисленной величиной. Поэтому *выражение-1* может быть целочисленной величиной, а выражение-2 указателем. В любом случае *выражение-2* должно быть заключено в квадратные скобки. Хотя индексное выражение обычно используется для ссылок на элементы массива, тем не менее, индекс может появляться с любым указателем.

Индексные выражения для ссылки на элементы одномерного массива вычисляются путем сложения целой величины со значениями указателя с последующим применением к результату операции разадресации (*).

Так как одно из выражений, указанных в индексном выражении, является указателем, то при сложении используются правила адресной арифметики, согласно которым целая величина преобразуется к адресному представлению, путем умножения ее на размер типа, адресуемого указателем. Пусть, например, идентификатор *arr* объявлен как массив элементов типа *double*.

double arr[10];

Таким образом, чтобы получить доступ к *i*-тому элементу массива *arr* можно написать *arr[i]*, что, в силу сказанного выше, эквивалентно *i[arr].* При этом величина *i* умножается на размер типа *double*, а затем это значение складывается со значением указателя *arr*, что в свою очередь дает адрес *i*-го элемента массива. К полученному адресу применяется операция разадресации, т.е. осуществляется выборка элемента массива *arr* по сформированному адресу.

Таким образом, результатом индексного выражения *arr[i]* (или *i[arr]*) является значение *i*-го элемента массива.

Для ссылки на элемент многомерного массива индексное выражение должно иметь несколько индексов заключенных к квадратные скобки:

выражение-1 [выражение-2][выражение-3] ...

Такое индексное выражение интерпретируется слева направо, т.е. вначале рассматривается первое индексное выражение:

выражение-1 [выражение-2]

Результат этого выражения есть адресное выражение, с которым складывается *выражение-3* и т.д. Операция разадресации осуществляется после вычисления последнего индексного выражения. Отметим, что операция разадресации не применяется, если значение последнего указателя адресует величину типа массива.

Пример:

int mass [2][5][3];

Рассмотрим процесс вычисления индексного выражения *mass[1][2][2].*

1. Вычисляется выражения *mass[1].* Ссылка индекс 1 умножается на размер элемента этого массива, элементом же этого массива является указатель на двумерный массив, содержащий 5x3 элементов, имеющих тип *int*. Получаемое значение складывается со значением указателя *mass*. Результатом является указатель на второй двухмерный массив размером (5x3) в трехмерном массиве *mass*.

2. Второй индекс 2 указывает на массив из трех элементов типа *int*, и складывается с адресом, соответствующим *mass [1].*

3. Так как каждый элемент трехмерного массива – это величина типа *int*, то индекс 2 увеличивается на размер типа *int* перед сложением с адресом *mass [1][2].*

4. Наконец, выполняется разадресация полученного указателя. Результирующим выражением будет элемент типа *int*.

Если было бы указано *mass [1][2]*, то результатом был бы указатель на массив из трех элементов типа *int*. Соответственно значением индексного выражения *mass [1]* является указатель на двухмерный массив.

Задача 6.1. Ввод-вывод, обнуление

```
#include <stdio.h>
#include <mem.h>
int main()
{
//Two-dimensional array
 int b[2][2];
 //Input
 for(int i=0;i<2;i++)
  for(int j=0;j<2;j++)
  {
 printf("Input b[%d][%d] ", i, j);
 scanf("%d", &b[i][j]);
  }
```

```
//Output as matrix
for(int i=0;i<2;i++)
{
 for(int j=0;j<2;j++)
printf("%4d", b[i][j]);
 printf("\n");
}
//Zeroing
memset(b, 0, sizeof(b));
getchar();getchar();
return 0;
}
```

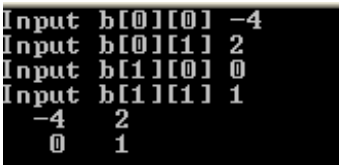


Рис. 6.1.

Задача 6.2. Олимпиада по информатике проводится в *m* туров.

Вывести победителей каждого тура и победителей в общем зачете. Максимальное число баллов в туре - 50 .

```
#include <stdio.h>
int main()
{
 const n=3;                    // число участников
 const m=2;                    // число туров
 int rez[n][m+1]; // последний столбец на сумму
 int max[m+1];   // лучшие результаты
 char name[n][20];             // фамилии
 int i, j;
 // Ввод данных
 for (i=0; i<n; i++)
 {
  printf("Input a surname of %d-st participant ", i+1);
  scanf("%s", name[i]);
  for (j=0; j<m; j++)
  {
 printf("Input its result in %d-st round ", j+1);
 scanf("%d", &rez[i][j]);
  }
 }
 // Подсчет суммарного результата
 for (i=0; i<n; i++)
 {
```

```
 rez[i][m]=0;
 for (j=0; j<m; j++)
rez[i][m]+=rez[i][j];
}
// Поиск максимальных результатов
for (j=0; j<m+1; j++)
{
 max[j]=rez[0][j];
 for (i=1; i<n; i++)
if (rez[i][j]>max[j]) max[j]=rez[i][j];
}
// Вывод победителей в турах
int t;
for (j=0; j<m; j++)
{
 printf("Winners in %d-st round with result %d\n", j+1, max[j]);
 t=0;
 for (i=0; i<n; i++)
if (rez[i][j]==max[j])
{
 t++;
 printf("%d. %-s\n", t, name[i]);
}
}
// Вывод победителя в общем зачете
printf("Winners in the general offset with result %d\n", max[m]);
t=0;
for (i=0; i<n; i++)
 if (rez[i][m]==max[m])
 {
t++;
printf("%d. %-s\n", t, name[i]);
 }
getchar();getchar();
return 0;
}
```

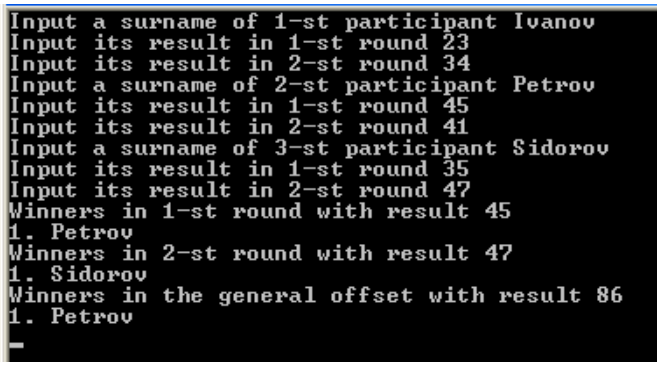


Рис. 6.2.

Задание для индивидуального выполнения
по теме «Работа с многомерными массивами»

Задание И6.1. Работа с двумерными массивами (матрицами)

1. Задана квадратная матрица размером *k*×*k*, составить программу вычисления суммы и произведения элементов матрицы, находящихся на главной диагонали.
2. Составить программу, которая позволяет для матрицы размерностью *m*×*n* вычислить произведение положительных, сумму отрицательных и количество нулевых элементов.
3. Двумерный массив *C* содержит 5 строк и столбцов, одномерный массив *A* содержит 5 элементов. Вычислить произведение матрицы на вектор.
4. Дана матрица *B*. Для каждого столбца с чётным номером вычислить и напечатать сумму квадратов элементов этого столбца, а для каждого столбца с нечетным номером вычислить произведение элементов.
5. Найти минимальный элемент *j*-го столбца матрицы *A* размером *n*×*n*, для которого сумма абсолютных значений элементов максимальна (если таких столбцов несколько, взять первый из них).
6. Дан двумерный массив *A*. Каждый элемент массива, стоящий выше главной диагонали, заменить его квадратом, а ниже диагонали – кубом его значения. Элементы главной диагонали оставить без изменения.
7. Задан двумерный массив *B*. Найти в нём максимальный элемент и разделить на него каждый элемент массива *B*.

8. Заданы матрица $n \times n$ и число $k<n$. Строку с максимальным по модулю элементом в k-м столбце переставить с k-й строкой.
9. Дана матрица B. Для каждого столбца матрицы вычислить и напечатать разность между квадратом суммы и суммой квадратов элементов этого столбца.
10. Заданы двумерный массив 5×5 и число K. Разделить элементы K-ой строки на диагональный элемент, расположенный в данной строке.
11. Определить и напечатать среднее значение элементов двухмерного массива размером $n \times m$. Найти индексы элемента, наиболее близкого по значению к среднему.
12. Составить программу нахождения минимального положительного элемента матрицы размерностью $m \times n$, а также значение индексов этого элемента.
13. Составить программу для определения количества отрицательных, положительных и равных нулю элементов матрицы $R(m \times n)$.
14. Составить программу вычисления суммы и произведения элементов квадратной матрицы A размерностью $n \times n$, расположенных ниже главной диагонали.
15. Составить программу для преобразования квадратной матрицы, состоящего в симметричной относительно главной диагонали перестановки её элементов.
16. Дана действительная матрица размера $m \times n$. Найти значение наибольшего по модулю элемента матрицы, а также индексы всех элементов с найденными значениями модуля.
17. Дана действительная матрица размера $m \times n$. Найти максимальные элементы по строкам и их сумму.
18. В данной действительной квадратной матрице порядка n найти сумму элементов строки, в которой расположен элемент с наименьшим значением. Предполагается что такой элемент единственный.
19. В данной действительной матрице размера 6x9 поменять местами строку, содержащую элемент с наибольшим значением, со строкой, содержащий элемент с наименьшим значением. Предполагается, что эти элементы единственны.
20. Дана действительная матрица размера $m \times n$, все элементы которой различны. В каждой строке выбирается элемент с наименьшим значением, затем среди этих чисел выбирается наибольшее. Указать индексы элемента с найденным значением.
21. Дана целочисленная квадратная матрица порядка 8. Найти наименьшее значение из элементов столбца, который обла-

дает наибольшей суммой модулей элементов. Если таких столбцов несколько, то взять первый.

22. Даны натуральные число n, целочисленная квадратная матрица порядка *n*. Получить $b_0, b_1, \ldots, b_{n-1}$, где b_i – это наименьшее значение элементов, находящихся в начале *i*-й строки матрицы до элемента, принадлежащего главной диагонали включительно.
23. Дана целочисленная квадратная матрица порядка n. Получить $b_0, b_1, \ldots, b_{n-1}$, где b_i – это значение первого по порядку положительного элемента *i*-й строки (если таких элементов нет, то принять b_1 =1).
24. Дана целочисленная квадратная матрица порядка *n*. Получить $b_0, b_1, \ldots, b_{n-1}$, где b_i – это сумма элементов, расположенных за первым отрицательным элементом в *i*-й строке (если все элементы неотрицательны, то принять b_i=100).
25. Дана целочисленная квадратная матрица порядка *n*. Получить $b_0, b_1, \ldots, b_{n-1}$, где b_i – это сумма элементов, предшествующих последнему отрицательному элементу *i*-й строки (если все элементы строки неотрицательны, то принять b_i=-1).
26. Дана действительная квадратная матрица порядка *n*. Отсортировать в порядке убывания те строки, которые начинаются с отрицательного элемента.
27. Дана матрица *A[m, m]* и массив *X[m].* Нечетные строки матрицы заменить на *X[m],* четные столбцы матрицы *A* заменить на *X[m].*
28. Дана матрица *A[m, m].* В матрице *A* поменять местами 1-ю и 2-ю строки, 3-ю и 4-ю, ..., 19-ю и 20-ю.
29. Дана матрица *A[m, m]* и натуральное число *K*. Получить из массива *A* массив *B*, заменив все элементы *K*-го столбца на нулевые.

Задание И6.2. Вложенные циклы в матричных задачах. В решении задачи можно использовать произвольный способ ввода и вывода как матрицы, так и результатов.

1. Для группы учащихся известны годовые оценки по следующим предметам: математика, физика, химия, информатика. Отобрать кандидатов на олимпиады (с отличными оценками) по каждому из предметов.
2. Для группы учащихся известны годовые оценки по следующим предметам: математика, физика, химия, информатика. Найти среднюю в группе оценку по каждому из предметов.

3. Спортсмены на соревнованиях совершают 6 попыток в прыжках в длину. Определить лучший результат для каждого участника.
4. Для группы учащихся известны годовые оценки по следующим предметам: математика, физика, химия, информатика. Найти в группе среднюю оценку для каждого учащегося.
5. Для группы фирм известен курс их акций за каждый из месяцев календарного года. Составить список тех фирм, курс акций которых все время повышался (т.е. курс за каждый последующий месяц больше, чем за предыдущий).
6. Выступления *N* спортсменов оцениваются *M* судьями по одной и той же числовой шкале. Нужно узнать конечный результат каждого спортсмена, если он вычисляется так: из всех *M* оценок удаляются максимальная и минимальная (если таких оценок несколько, то удаляется только одна), затем из оставшихся (*M-2*) находится их среднее арифметическое.
7. Известна среднемесячная температура воздуха на следующих островах Карибского моря: Куба, Тринидад, Ямайка, Гаити. Определить, на каком из островов среднегодовая температура максимальна.
8. Известна среднемесячная температура воздуха на следующих островах Карибского моря: Куба, Тринидад, Ямайка, Гаити. Определить для каждого из месяцев, на каком из островов максимальная температура.
9. Дан двумерный массив из 5 строк и 6 столбцов. Определить для каждого четного столбца максимальный элемент. Найти произведение этих элементов.
10. Дан двумерный массив из 5 строк и 6 столбцов. Определить, какая строка массива имеет максимальную сумму элементов (считать, что строк с одинаковой суммой нет).
11. Дан двумерный массив из 5 строк и 6 столбцов. Определить, для каждой нечетной строки минимальный элемент. Найти произведение этих элементов.
12. Дан двумерный массив из 5 строк и 6 столбцов. Определить для каждой строки минимальный элемент. Среди этих элементов найти максимальный.
13. Дан двумерный массив из 5 строк и 6 столбцов. Определить для каждого столбца максимальный элемент. Среди этих элементов найти минимальный.
14. Дан двумерный массив из 5 строк и 6 столбцов. Определить для каждого столбца массива сумму минимального и максимального элементов. Найти произведение этих сумм.

15. Дан двумерный массив из 5 строк и 6 столбцов. Определить для каждой строки массива произведение минимального и максимального элементов. Найти сумму этих произведений.
16. Дан двумерный массив из 5 строк и 6 столбцов. Определить для каждой строки массива произведение элементов. Найти, в какой строке это произведение максимально.
17. Дан двумерный массив из 5 строк и 6 столбцов. Определить, какой столбец массива имеет минимальную сумму элементов (считать, что столбцов с одинаковой суммой нет).
18. Известны места 8 лыжников на каждом из 7 стартов Кубка мира. Определить победителя в общем зачете (с минимальной суммой мест). Если таких несколько, то победителем считается тот, кто лучше других претендентов на первое место выступил на последнем этапе.
19. По результатам метеорологических наблюдений за 10 последних лет известно количество солнечных дней в году для пяти морских курортов. Кроме этого известны расстояния до них. Определить курорт с наиболее благоприятным климатом (с максимальным суммарным количеством солнечных дней за время наблюдений). Если таких несколько, то вывести ближайший из них.
20. В автопарке находится 10 автомобилей. Известен их пробег в течение каждого из 5 рабочих дней. Определить, какой из автомобилей за рабочую неделю преодолел максимальное расстояние. Если таких несколько, то вывести хотя бы одного из них.
21. Бригада рабочих состоит из 8 человек. Известно сколько деталей выпустил каждый из них в течение каждого из 5 рабочих дней. Определить, какой рабочий произвел максимальное количество деталей. Если таких несколько, то вывести хотя бы одного из них.
22. На каждом этаже трехэтажного дома жилых 6 комнат, каждая из которых имеет форму прямоугольника. Длина и ширина каждой комнаты известны. Определить, какой из этажей дома имеет минимальную жилую площадь. Если таких несколько, то вывести хотя бы одного из них.
23. Известны итоги сдачи экзаменов группой из 20 студентов по четырем предметам. Определить, по результатам каких экзаменов в группе максимальный средний балл.
24. В соревнованиях по десятиборью участвуют 8 спортсменов. Известны результаты в баллах по каждому из видов. Вывести фамилии спортсменов и их результаты в сумме в порядке занятых мест.

25. Известен курс акций 5 фирм за каждый из месяцев календарного года. За второе полугодие для каждой фирмы определить месяц максимального прироста курса. Если таких несколько, то вывести последний из них.

Задание И6.3. Задачи на сортировку и упорядочивания элементов массива

1. В массиве из 10 целых чисел найти наименьший элемент. Выполнить сортировку по невозрастанию между минимальным и последним элементом.
2. В массиве из 15 вещественных чисел найти наибольший элемент. Выполнить сортировку по неубыванию между максимальным и последним элементом.
3. В массиве из 25 вещественных чисел найти наименьший элемент. Выполнить сортировку по убыванию между минимальным и первым элементом.
4. Упорядочить по неубыванию массив, содержащий 20 целых чисел.
5. Упорядочить по невозрастанию массив, содержащий 10 целых чисел.
6. Упорядочить по неубыванию массив, содержащий 15 вещественных чисел.
7. Упорядочить по невозрастанию массив, содержащий 25 вещественных чисел.
8. Дан массив целых чисел, содержащий 20 элементов, записать в этот же массив сначала все отрицательные числа и нули, затем все положительные, сохраняя порядок их следования.
9. Дан двумерный массив, содержащий 5 строк и 3 столбца. Элементами массива являются целые числа. Переставить строки таким образом, чтобы элементы первого столбца были упорядочены по неубыванию.
10. Дан двумерный массив, содержащий 4 строки и 5 столбцов. Элементами массива являются целые числа. Переставить строки таким образом, чтобы элементы пятого столбца были упорядочены по невозрастанию.
11. Дан двумерный массив, содержащий 3 строки и 4 столбца. Элементами массива являются вещественные числа. Переставить строки таким образом, чтобы элементы первого столбца были упорядочены по невозрастанию.
12. Дан двумерный массив, содержащий 4 строки и 4 столбца. Элементами массива являются вещественные числа. Переставить строки таким образом, чтобы элементы второго столбца были упорядочены по невозрастанию.

13. Дан двумерный массив, содержащий 2 строки и 4 столбца. Элементами массива являются целые числа. Переставить столбцы таким образом, чтобы элементы первой строки были упорядочены по невозрастанию.
14. Дан двумерный массив, содержащий 3 строки и 4 столбца. Элементами массива являются целые числа. Переставить столбцы таким образом, чтобы элементы второй строки были упорядочены по невозрастанию.
15. Дан двумерный массив, содержащий 3 строки и 2 столбца. Элементами массива являются целые числа. Переставить столбцы таким образом, чтобы элементы третьей строки были упорядочены по невозрастанию.
16. Дан одномерный массив A[n]. Упорядочить его по неубыванию до первого максимального числа, а от него до конца массива по невозрастанию.
17. Определите номер наибольшего элемента массива A и наибольшего значения среди модулей элементов массива A. Упорядочить массив по убыванию абсолютных величин.

ЛАБОРАТОРНАЯ РАБОТА № 7. СТРУКТУРЫ

Массивы позволяют обращаться с набором логически связанных однотипных элементов как с единым целым. Если же требуется хранить набор разнородных, но логически связанных данных, описывающих, например, состояние некоторого объекта реального мира, используются *структуры.* Структуры могут быть простые и связанные указателями в цепь, дерево или сеть.

Структура состоит из элементов различных типов, причем каждый элемент структуры имеет собственное имя. Описание структуры имеет вид

```
struct [имя_типа]
{
  [список описаний]
} [список_переменных];
```

Список описаний содержит типы и имена элементов структуры. Например, структура stud

```
struct stud
{
char name[20];              /*Фамилия и инициалы*/
struct
{
int day;                    /*День*/
char month[9];              /*Месяц*/
int year;           /*Год рождения*/
} date;
char address[40];   /*Адрес*/
} student;
```

состоит из трех элементов (одномерного символьного массива *name*[20], структуры *date* и еще одного символьного массива *address*[40]). Второй элемент, в свою очередь, структура *date* состоит также из трех элементов (переменной целого типа *day*, одномерного символьного массива *month*[9] и переменной целого типа *year*). Таким образом, структура *stud* содержит фамилию, дату рождения и адрес местожительства студента, т.е. структура группирует различные данные, относящиеся к конкретному человеку.

Для доступа к отдельным элементам структуры имеются две операции: точка и стрелка, за которыми следует имя элемента. Какую из них следует применять, зависит от того, имеете ли вы дело с самой переменной-структурой или у вас есть только указатель на нее. С именем переменной применяется точка, с указателем – стрелка. Доступ к отдельному элементу структуры осуществляется по составному имени, образованному из имени самой

структуры и имени элемента, разделенных точкой. Например, *student.name* – фамилия студента, *student.date.year* – его год рождения. Имея в виду предыдущее определение, можно было бы написать:

```
stud *p,s;
p->name="Ivanov A.A.";
```

Или

```
s.name="Ivanov A.A.";
```

Задача 7.1. Найти сумму, разность и произведение двух комплексных чисел.

```
#include <stdio.h>
#include <iostream.h>
// Описание структуры
struct
{
 int re,  // Действительная часть комплексного числа
     im;  // Коэффициент при мнимой части числа
}a,b,c,d,e;

int main()
{
 cout<<"Enter complex a (re and im): ";
 cin>>a.re>>a.im;
 cout<<"Enter complex b (re and im): ";
 cin>>b.re>>b.im;
 c.re=a.re+b.re;
 c.im=a.im+b.im;
 d.re=a.re-b.re;
 d.im=a.im-b.im;
 e.re=a.re*b.re+(-1)*(a.im*b.im);
 e.im=a.re*b.im+a.im*b.re;
 cout<<"a="<<a.re<<"+("<<a.im<<"i)"<<endl;
 cout<<"b="<<b.re<<"+("<<b.im<<"i)"<<endl;
 cout<<"Result (c=a+b,d=a-b,e=a*b):"<<endl;
 cout<<"c="<<c.re<<"+("<<c.im<<"i)"<<endl;
 cout<<"d="<<d.re<<"+("<<d.im<<"i)"<<endl;
 cout<<"e="<<e.re<<"+("<<e.im<<"i)"<<endl;
 getchar();getchar();
 return 0;
}
```

Результат работы программы приведен на рис. 7.1.

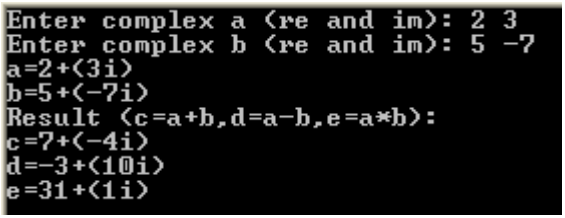


Рис. 7.1

Задача 7.2. Вывести данные по студентам, у которых есть хотя бы одна оценка «удовлетворительно».

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

const int n=25,    // Максимальное кол-во студентов в группе
          m=20;    // Максимальное кол-во символов в
                   // имени и фамилии студентов
// Описание структурированного типа данных - student
struct student
{
 char       surname [m], //Фамилия
            name[m];   //Имя
 unsigned int day,              //День
            month,              //Месяц
            year,               //Год
            ball[3];      //Оценки по трем предметам
};

int main()
{
 student stud[n];                        // Массив данных типа student
 cout<<"Enter count student: ";
 int k,kol=0;                            // k-кол-во студентов в группе
 cin>>k;
// Ввод информации о k-студентах группы
for(int i=0;i<k;i++)
 {
   cout<<"Enter information about "<<i+1<<" - student"<<endl;
   cout<<"Surname: ";
   cin>>stud[i].surname;
   cout<<"Name: ";
   cin>>stud[i].name;
   cout<<"Birthday (day, month, year) ";
```

```
            cin>>stud[i].day>>stud[i].month>>stud[i].year;
            cout<<"Ball (fisica, mathematica, information) ";
            cin>>stud[i].ball[0]>>stud[i].ball[1]>>stud[i].ball[2];
          }
          cout<<endl;
         // Вывод заголовка таблицы
cout<<setw(20)<<"Surname"<<setw(20)<<"Name"<<setw(18)<<"Birthday"<<se
tw(20)<<"Ball(fis,math,inf)"<<endl;
         // Вывод информации о всех студентах группы
           for(int i=0;i<k;i++)
            {
         // Подсчет кол-ва студентов, имеющих 3 хотя бы по одному
         // предмету
           if (stud[i].ball[0]==3||stud[i].ball[1]==3||
         stud[i].ball[2]==3) kol++;
            cout<<setw(20)<<stud[i].surname<<setw(20)<<stud[i].name<<
setw(8)<<""<<setw(2)<<setfill('0')<<stud[i].day<<'/'
             <<setw(2)<<setfill('0')<<stud[i].month<<'/'<<stud[i].year<<
setfill(' ')<<setw(16)<<stud[i].ball[0]<<' '<<stud[i].ball[1]<<'
'<<stud[i].ball[2]<<endl;
            }
          cout<<endl<<endl;
          cout<<"Result:"<<endl;
         // Если в группе есть такие студенты, то выводим
         // информацию о них
          if (kol>0)
          {
cout<<setw(20)<<"Surname"<<setw(20)<<"Name"<<setw(18)<<"Birthday"<<se
tw(20)<<"Ball(fis,math,inf)"<<endl;
           for(int i=0;i<k;i++)
            if (stud[i].ball[0]==3||stud[i].ball[1]==3||stud[i].ball[2]==3)
            cout<<setw(20)<<stud[i].surname<<setw(20)<<stud[i].name<<
setw(8)<<""<<setw(2)<<setfill('0')<<stud[i].day<<'/'
            <<setw(2)<<setfill('0')<<stud[i].month<<'/'<<stud[i].year<<setfill(' ')<<
            setw(16)<<stud[i].ball[0]<<' '<<stud[i].ball[1]<<' '<<stud[i].ball[2]<<endl;
          }
         // Иначе выводим информацию, что таких студентов нет
          else cout<<"No student";
           getchar();getchar();
           return 0;
         }
```

Результат работы программы приведен на рис. 7.2.

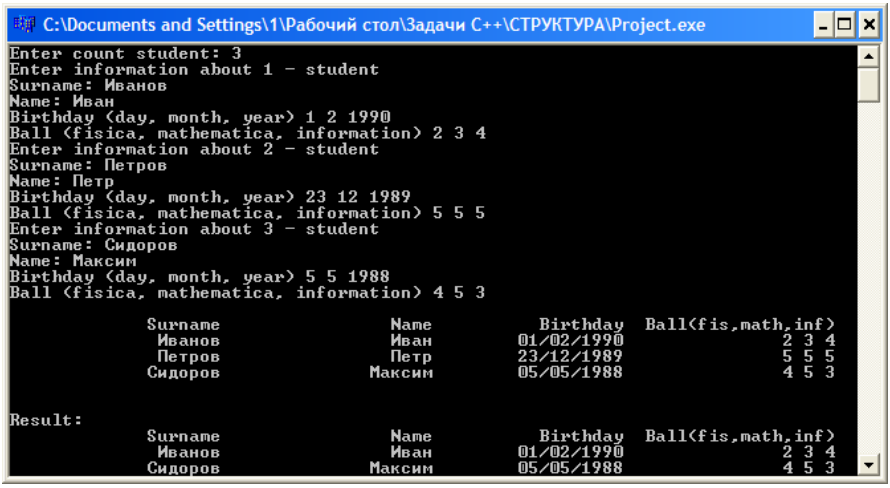


Рис. 7.2

Присваивание будет осуществляться только в том случае, когда две структуры однотипны.

Сравнивать непосредственно две структуры нельзя. Сравнивать можно только члены этих структур.

Члены структур хранятся в памяти подряд, друг за другом. В объединении все члены перекрывают друг друга, располагаясь по одному и тому же адресу. Это означает, что одна и та же оперативная память используется для хранения значений различных переменных одинаковых или разных типов. Это целесообразно использовать в том случае, когда значения переменных объединения не используются в программе одновременно, а поочередно. Объединения используют также для преобразования типов. При этом одно и то же значение рассматривается то, как значение одного типа, то, как значение другого типа. Форма объявления объединения аналогична форме объявления структур, но вместо ключевого слова *struct* используется ключевое слово *union*, Например:

```
union sm {
int   ival;
float   fvai,
char   cval; } uval;
```

где *sm* – имя типа объединения; *uval* – имя переменной типа объединение; *ivaf*, *fval*, *cval* – переменные различных типов, занимаю-

щие одну и ту же оперативную память, выделенную переменной *uval*.

Форма обращения к элементам объединения идентична форме обращения к элементам структур. Например, в программе могут быть операторы присваивания:

```
uval.ival = 5;
uval.fval = 3.5;
uval.cval = 'a';
```

Объединения могут быть в составе структур. В составе объединения могут быть массивы. Комбинация массивов и структур позволяет создавать мощные структуры данных, которые могут помочь организовать информацию почти без всяких ограничений.

Задание для индивидуального выполнения по теме «Работа со структурами» [1]

Задание И6. Составить список учебной группы, включающий 10 человек. Для каждого студента указать: фамилию и имя, дату рождения (год, месяц и число) оценки за сессию (от 3 до 5 экзаменов). Информацию о каждом студенте оформить в виде структуры, а совокупность структур объединить в массив. Составить программу, которая обеспечивает ввод полученной информации, ее просмотр в виде таблицы, а также вывод информации на экран монитора согласно конкретному варианту. В случае если в группе нет студентов с требуемыми данными, выдать соответствующее сообщение.

Варианты:

1. Вывести анкетные данные студентов-отличников.
2. Вывести анкетные данные всех студентов, у которых по всем предметам «удовлетворительно».
3. Вывести анкетные данные всех студентов успевающих на «хорошо» и «отлично».
4. Вывести анкетные данные студентов, получивших по предмету физика оценку «неудовлетворительно».
5. Вывести анкетные данные студентов, получивших по предмету физика оценку «отлично».
6. Вывести анкетные данные студентов, получивших по одному из предметов оценку «хорошо», а по всем другим – «отлично».
7. Вывести список студентов, фамилии которых начинаются с буквы "А" и их оценки по всем предметам.
8. Вывести список студентов, фамилии которых начинаются с буквы "В" и их даты рождения.

9. Вывести оценки по всем предметам студентов, фамилии которых начинаются с букв "B" и "D".
10. Вывести фамилии и даты рождения студентов, не получивших ни одной оценки «удовлетворительно» по всем предметам.
11. Упорядочить список студентов по среднему баллу и вывести весь список.
12. Упорядочить список студентов по предмету физика и вывести весь список.
13. Вычислить средний балл группы и вывести список студентов, имеющих средний балл выше, чем в целом по группе.
14. Вычислить средний балл группы и вывести список студентов, имеющих средний балл ниже, чем в целом по группе.
15. Вычислить средний балл группы и вывести список студентов, имеющих средний балл, близкий к среднему баллу группы (отличающийся не более чем на K процентов).
16. Упорядочить список студентов по дате рождения и вывести весь список.
17. Вывести список студентов, упорядоченный по алфавиту.
18. Вывести список студентов, упорядоченный по месяцу рождения.
19. Вывести список студентов, упорядоченный по именам.
20. Вывести анкетные данные студентов, получивших по предметам физика и химия оценку «неудовлетворительно».
21. Вывести анкетные данные студентов, получивших по предмету информатика оценку «отлично».
22. Вывести анкетные данные студентов, имеющих по предметам физика и химия оценку «хорошо» или «отлично».
23. Вывести оценки студентов, фамилии которых начинаются на буквы "B", "D", "E".
24. Вывести оценки студентов, фамилии которых начинаются на буквы "B", "D", "E" и имеющих средний балл по всем предметам выше, чем средний балл в целом по группе.

ЛАБОРАТОРНАЯ РАБОТА № 8. РАБОТА С ФУНКЦИЯМИ

Определение и вызов

Мощность языка СИ во многом определяется легкостью и гибкостью в определении и использовании функций в СИ-программах. В отличие от других языков программирования высокого уровня в языке СИ нет деления подпрограмм на процедуры и функции, здесь вся программа строится только из функций.

Функция – это совокупность объявлений и операторов, обычно предназначенная для решения определенной задачи. Каждая функция должна иметь имя, которое используется для ее объявления, определения и вызова. В любой программе на СИ должна быть функция с именем main (главная функция), именно с этой функции, в каком бы месте программы она не находилась, начинается выполнение программы.

При вызове функции ей при помощи аргументов (формальных параметров) могут быть переданы некоторые значения (фактические параметры), используемые во время выполнения функции. Функция может возвращать некоторое (одно!) значение. Это возвращаемое значение и есть результат выполнения функции, который при выполнении программы подставляется в точку вызова функции, где бы этот вызов ни встретился. Допускается также использовать функции, не имеющие аргументов, и функции, не возвращающие никаких значений. Действие таких функций может состоять, например, в изменении значений некоторых переменных, выводе на печать некоторых текстов и т.п.

С использованием функций в языке СИ связаны три понятия – определение функции (описание действий, выполняемых функцией), объявление функции (задание формы обращения к функции) и вызов функции.

Определение функции задает тип возвращаемого значения, имя функции, типы и число формальных параметров, а также объявления переменных и операторы, называемые телом функции, и определяющие действие функции. В определении функции также может быть задан класс памяти.

Пример:

```
int rus (unsigned char r)
 {  if (r>='A')
      return 1;
    else
      return 0;
 }
```

В данном примере определена функция с именем *rus*, имеющая один параметр с именем *r* и типом *unsigned char*. Функция возвращает целое значение, равное *1*, если параметр функции является буквой русского алфавита, или *0* в противном случае.

В языке СИ нет требования, чтобы определение функции обязательно предшествовало ее вызову. Определения используемых функций могут следовать за определением функции *main*, перед ним, или находится в другом файле.

Однако для того, чтобы компилятор мог осуществить проверку соответствия типов передаваемых фактических параметров типам формальных параметров, до вызова функции нужно поместить объявление (прототип) функции.

Объявление функции имеет такой же вид, что и определение функции, с той лишь разницей, что тело функции отсутствует, и имена формальных параметров тоже могут быть опущены. Для функции, определенной в последнем примере, прототип может иметь вид

int rus (unsigned char r); или rus (unsigned char);

В программах на языке СИ широко используются, так называемые, библиотечные функции, т.е. функции, предварительно разработанные и записанные в библиотеки. Прототипы библиотечных функций находятся в специальных заголовочных файлах, поставляемых вместе с библиотеками в составе систем программирования, и включаются в программу с помощью директивы *#include*.

Если объявление функции не задано, то по умолчанию строится прототип функции на основе анализа первой ссылки на функцию, будь то вызов функции или определение. Однако такой прототип не всегда согласуется с последующим определением или вызовом функции. Рекомендуется всегда явно задавать прототип функции. Это позволит компилятору либо выдавать диагностические сообщения, при неправильном использовании функции, либо корректным образом регулировать несоответствие аргументов, устанавливаемое при выполнении программы.

Объявление параметров функции при ее определении может быть выполнено в так называемом «старом стиле», при котором в скобках после имени функции следуют только имена параметров, а после скобок объявления типов параметров. Например, функция rus из предыдущего примера может быть определена следующим образом:

```
int rus (r)
unsigned char r;
```

{ ... / тело функции */ ... }*

В соответствии с синтаксисом языка СИ определение функции имеет следующую форму:

[спецификатор-класса-памяти] [спецификатор-типа] имя-функции
([список-формальных-параметров])
{ тело-функции }

Необязательный спецификатор-класса-памяти задает класс памяти функции, который может быть **static** или **extern**. Подробно классы памяти будут рассмотрены в следующем разделе.

Спецификатор-типа функции задает тип возвращаемого значения и может задавать любой тип. Если спецификатор-типа не задан, то предполагается, что функция возвращает значение типа *int*.

Функция не может возвращать массив или функцию, но может возвращать указатель на любой тип, в том числе и на массив и на функцию. Тип возвращаемого значения, задаваемый в определении функции, должен соответствовать типу в объявлении этой функции.

Функция возвращает значение, если ее выполнение заканчивается оператором *return*, содержащим некоторое выражение. Указанное выражение вычисляется, преобразуется, если необходимо, к типу возвращаемого значения и возвращается в точку вызова функции в качестве результата. Если оператор *return* не содержит выражения или выполнение функции завершается после выполнения последнего ее оператора (без выполнения оператора *return*), то возвращаемое значение не определено. Для функций, не использующих возвращаемое значение, должен быть использован тип *void*, указывающий на отсутствие возвращаемого значения. Если функция определена как функция, возвращающая некоторое значение, а в операторе return при выходе из нее отсутствует выражение, то поведение вызывающей функции после передачи ей управления может быть непредсказуемым.

Список-формальных-параметров – это последовательность объявлений формальных параметров, разделенная запятыми. Формальные параметры – это переменные, используемые внутри тела функции и получающие значение при вызове функции путем копирования в них значений соответствующих фактических параметров. Список-формальных-параметров может заканчиваться запятой (,) или запятой с многоточием (,...), это означает, что число аргументов функции переменно. Однако предполагается, что функция имеет, по крайней мере, столько обязательных аргумен-

тов, сколько формальных параметров задано перед последней запятой в списке параметров. Такой функции может быть передано большее число аргументов, но над дополнительными аргументами не проводится контроль типов.

Если функция не использует параметров, то наличие круглых скобок обязательно, а вместо списка параметров рекомендуется указать слово *void*.

Порядок и типы формальных параметров должны быть одинаковыми в определении функции и во всех ее объявлениях. Типы фактических параметров при вызове функции должны быть совместимы с типами соответствующих формальных параметров. Тип формального параметра может быть любым основным типом, структурой, объединением, перечислением, указателем или массивом. Если тип формального параметра не указан, то этому параметру присваивается тип *int*.

Для формального параметра можно задавать класс памяти *register*, при этом для величин типа *int* спецификатор типа можно опустить.

Идентификаторы формальных параметров используются в теле функции в качестве ссылок на переданные значения. Эти идентификаторы не могут быть переопределены в блоке, образующем тело функции, но могут быть переопределены во внутреннем блоке внутри тела функции.

При передаче параметров в функцию, если необходимо, выполняются обычные арифметические преобразования для каждого формального параметра и каждого фактического параметра независимо. После преобразования формальный параметр не может быть короче, чем int, т.е. объявление формального параметра с типом char равносильно его объявлению с типом *int*. А параметры, представляющие собой действительные числа, имеют тип *double*.

Преобразованный тип каждого формального параметра определяет, как интерпретируются аргументы, помещаемые при вызове функции в стек. Несоответствие типов фактических аргументов и формальных параметров может быть причиной неверной интерпретации.

C++ допускает сосуществование двух или более функций, имеющих одно и то же имя, но разные списки параметров. Такие функции называются перегруженными. О списке параметров в этом случае говорят как о сигнатуре функции, поскольку именно он используется различения разных версий одноименных функций. Имя и сигнатура однозначно идентифицируют версию.

Тело функции – это составной оператор, содержащий операторы, определяющие действие функции.

Все переменные, объявленные в теле функции без указания класса памяти, имеют класс памяти *auto*, т.е. они являются локальными. При вызове функции локальным переменным отводится память в стеке и производится их инициализация. Управление передается первому оператору тела функции и начинается выполнение функции, которое продолжается до тех пор, пока не встретится оператор *return* или последний оператор тела функции. Управление при этом возвращается в точку, следующую за точкой вызова, а локальные переменные становятся недоступными. При новом вызове функции для локальных переменных память распределяется вновь, и поэтому старые значения локальных переменных теряются.

Функции используют память из стека программы. Некоторая область стека отводится функции и остается связанной с ней до окончания ее работы, по завершении которой отведенная ей память освобождается и может быть занята другой функцией. Иногда эту часть стека называют областью активации. Каждому параметру функции отводится место в данной области, причем его размер определяется типом параметра. При вызове функции память инициализируется значениями фактических аргументов. Стандартным способом передачи аргументов является копирование их значений, т.е. передача по значению. При этом способе функция не получает доступа к реальным объектам, являющимся ее аргументами. Вместо этого она получает в стеке локальные копии этих объектов. Изменение значений копий никак не отражается на значениях самих объектов. Локальные копии теряются при выходе из функции.

Значения аргументов при передаче по значению не меняются. Следовательно, программист не должен заботиться о сохранении и восстановлении их значений при вызове функции. Без этого механизма любой вызов мог бы привести к нежелательному изменению аргументов, не объявленных константными явно. Передача по значению освобождает человека от лишних забот в наиболее типичной ситуации.

Однако, если в качестве параметра передать указатель на некоторую переменную, то, используя операцию разадресации, можно изменить значение этой переменной.

Пример:

```
/*   Неправильное использование параметров   */
void change (int x, int y)
{  int k=x;
   x=y;
   y=k;
}
```

В данной функции значения переменных *x* и *y*, являющихся формальными параметрами, меняются местами, но поскольку эти переменные существуют только внутри функции *change*, значения фактических параметров, используемых при вызове функции, останутся неизменными. Для того чтобы менялись местами значения фактических аргументов можно использовать функцию, приведенную в следующем примере.

Пример:

```
/*    Правильное использование параметров   */
  void change (int *x, int *y)
  {  int k=*x;
     *x=*y;
     *y=k;}
```

При вызове такой функции в качестве фактических параметров должны быть использованы не значения переменных, а их адреса

change (&a, &b);

Ссылки

В C++ имеется модифицированная форма указателей, называемая ссылкой (reference). Ссылка – это указатель, который автоматически разыменовывается при любом обращении к нему. Поэтому ссылке, как таковой, нельзя присвоить значение, как это можно сделать с обычным указателем. Ссылку можно только инициализировать.

Ссылки как псевдонимы переменных

Переменная-ссылка объявляется со знаком “&”, предшествующим ее имени; инициализирующим выражением должно быть имя переменной. Рассмотрите такой пример:

```
// Alias.cpp: Ссылка как псевдоним.
//
#include <stdio.h>
int main()
{
int iVar = 7777;
// Инициализация целой переменной.
int *iPtr = &iVar;
// Инициализация указателя адресом iVar.
int &iRef = iVar;
// Инициализация ссылки как псевдонима iVar.
int *irPtr = &iRef;
// Инициализация указателя "адресом" ссылки.
printf("iVar = %d\n", iVar) ;
```

```
printf("*iPtr = %d\n", *iPtr);
printf("iRef = %d\n", iRef);
printf("*irPtr = %d\n", *irPtr) ;
printf("iPtr = %p, irPtr = %p\n", iPtr, irPtr);
return 0;
}
```

Первые четыре оператора *printf* выводят одно и то же число 7777. последний оператор печатает значения двух указателей, первый из которых инициализирован адресом переменной, а второй адресом ссылки. Эти адреса также оказываются одинаковыми *(iPtr=0012FF88, irPtr=0012FF88).* Как видите, после инициализации ссылки ее имя используется в точности как имя ассоциированной с ней переменной, т. е. как псевдоним.

Еще раз подчеркнем, что ссылку после инициализации нельзя изменить; все обращения к ссылке будут относиться на самом деле к переменной, именем которой она была инициализирована.

Ссылки в качестве параметров функции

Все сказанное в предыдущем параграфе не имеет существенного практического значения; к чему, в самом деле, вводить ссылку на переменную, если с тем же успехом можно обращаться к ней самой? По-иному дело обстоит в случае, когда ссылкой объявляется параметр функции.

Параметры функции мало чем отличаются от ее локальных автоматических переменных. Разница только в том, что они располагаются на стеке выше адреса в регистре *EBP*, а локальные переменные – ниже. Передача параметров при вызове функции эквивалентна созданию локальных переменных и инициализации их значениями соответствующих аргументов.

Это означает, что если объявить формальный параметр как ссылку, то при вызове он будет инициализирован в качестве псевдонима переменной-аргумента. Если говорить о конкретной реализации этого механизма, функции передается указатель на переменную, автоматически разыменуемый при обращении к нему в теле функции.

Тем самым параметры-ссылки делают возможной настоящую «передачу по ссылке» без использования явных указателей и адресов. Вот сравнение различных видов передачи параметров:

```
// RefPar.cpp: Передача параметров по ссылке.
#include <stdio.h>
void change (int &x, int &y)
{
int k;
```

```
k = x;
x = y;
y = k;
}
```

Вызов:

```
change (a, b);
```

Помимо случаев, когда функция должна возвращать значения в параметрах, ссылки могут быть полезны при передаче больших структур, поскольку функции тогда передается не сама структура, а ее адрес.

Примечание

Передавая параметры по ссылке, нужно всегда помнить о возможных побочных эффектах. Если только функция не должна явно изменять значение своего аргумента, лучше объявить параметр как ссылку на константу (например, *const int SiVal*). Тогда при попытке присвоить параметру значение в теле функции компилятор выдаст сообщение об ошибке.

Ссылка в качестве возвращаемого значения

Возвращаемое функцией значение также может быть объявлено ссылкой. Это позволяет использовать функцию в левой части присваивания. Рассмотрите такой пример:

```
#include <stdio.h>
#include <assert.h>
const int arrSize = 8;
int &refItem(int indx)  // Возвращает ссылку на элемент iArray[indx]
{
  static int iArray[arrSize];
  // Проверка диапазона:
  // Tests a condition and possibly aborts. assert is a macro that expands
  // to an if statement; if test evaluates to zero, the assert macro calls
  // the _assert  function and aborts the program
  assert(indx >= 0 && indx< arrSize);
  return iArray[indx];
}
int main()
{
  for (int i =0; i<arrSize; i++)
    refItem(i) = 1 << i;
  // Присваивает значение элементу iArray[i].
  for (int i=0; i<arrSize; i++)
    printf("iArray[%02d] = %4d\n", i, refItem(i));
  return 0;
}
```

Вывод:

```
iArray[00] =   1
iArray[01] =   2
iArray[02] =   4
iArray[03] =   8
iArray[04] =  16
iArray[05] =  32
iArray[06] =  64
iArray[07] = 128
```

В первом из операторов for функция ***refItem()*** вызывается в левой части присваивания. Во втором for она возвращает значение, которое передается функции *printf ()*. Обратите внимание, что, во-первых, массив *iArray[]* объявлен как статический локальный в *refItem()*, благодаря чему непосредственное обращение к нему вне этой функции невозможно. Во-вторых, *refItem()* попутно проверяет допустимость переданного ей индекса.

Пример 2. Функция удвоения:

```
#include <stdio.h>
void mul2(int a, int &b);
int main(int argc, char* argv[])
{
  int a=2, b=3;
  printf("1. a=%d b=%d\n", a, b);              //2 3
  mul2(a, b);
  printf("3. a=%d b=%d\n", a, b);              //2 6
  getchar();
  return 0;
}
void mul2(int a, int &b)
{
  a*=2;
  b*=2;
  printf("2. a=%d b=%d\n", a, b);              //4 6
}
```

Передача массивов

Т.к. имя переменной-массива определяет его адрес, то при передаче в функцию массива операцию взятия адреса применять не следует. Массив всегда передается по адресу, поэтому его изменение в функции приведет к изменению фактического параметра.

Размер массива неважен при объявлении параметра. Все три приведенные записи

```
void f(int*);
```

void f(int[]);
void f(int[10]);

эквивалентны. Контроль за тем, не является ли аргумент массивом, не производится. При передаче массива в качестве параметра часто необходимо указывать его размер. Обычно для такого указания используют дополнительный параметр функции.

void f(int[10], int size);

Другой способ сообщить функции размер массива-параметра – объявить параметр как ссылку. В этом случае размер становится частью типа, и компилятор может проверить аргумент в полной мере.

void f(int (&arr)[10]);

Для передачи массива по значению можно его поместить в структуру, которая содержит единственное поле – сам массив.

```
#include <stdio.h>
typedef struct
            {
              int a[1];
            } S;
void f(int a[])
{
  a[0]=2;
}
void f2(S x)
{
  x.a[0]=2;
}
int main(int argc, char* argv[])
{
   int a[1]={1};
   printf("%d\n", a[0]);
   f(a);
   printf("%d\n", a[0]);
   S t;
   t.a[0]=1;
   printf("%d\n", t.a[0]);
   f2(t);
   printf("%d\n", t.a[0]);
   getchar();
   return 0;
}
```

Прототип

Если требуется вызвать функцию до ее определения в рассматриваемом файле, или определение функции находится в другом исходном файле, то вызов функции следует предварять объявлением этой функции. Объявление (прототип) функции имеет следующий формат:

[спецификатор-класса-памяти] [спецификатор-типа] имя-функции ([список-формальных-параметров]) [,список-имен-функций];

В отличие от определения функции, в прототипе за заголовком сразу же следует точка с запятой, а тело функции отсутствует. Если несколько разных функций возвращают значения одинакового типа и имеют одинаковые списки формальных параметров, то эти функции можно объявить в одном прототипе, указав имя одной из функций в качестве имени-функции, а все другие поместить в список-имен-функций, причем каждая функция должна сопровождаться списком формальных параметров. Правила использования остальных элементов формата такие же, как при определении функции. Имена формальных параметров при объявлении функции можно не указывать, а если они указаны, то их область действия распространяется только до конца объявления.

Прототип – это явное объявление функции, которое предшествует определению функции. Тип возвращаемого значения при объявлении функции должен соответствовать типу возвращаемого значения в определении функции.

Если прототип функции не задан, а встретился вызов функции, то строится неявный прототип из анализа формы вызова функции. Тип возвращаемого значения создаваемого прототипа int, а список типов и числа параметров функции формируется на основании типов и числа фактических параметров, используемых при данном вызове.

Таким образом, прототип функции необходимо задавать в следующих случаях:

1. Функция возвращает значение типа, отличного от *int*.

2. Требуется проинициализировать некоторый указатель на функцию до того, как эта функция будет определена.

Наличие в прототипе полного списка типов аргументов параметров позволяет выполнить проверку соответствия типов фактических параметров при вызове функции типам формальных параметров, и, если необходимо, выполнить соответствующие преобразования.

В прототипе можно указать, что число параметров функции переменно, или что функция не имеет параметров.

Если прототип задан с классом памяти *static*, то и определение функции должно иметь класс памяти *static*. Если спецификатор класса памяти не указан, то подразумевается класс памяти *extern*.

Вызов функции имеет следующий формат:

адресное-выражение ([список-выражений])

Поскольку синтаксически имя функции является адресом начала тела функции, в качестве обращения к функции может быть использовано адресное-выражение, имеющее значение адреса функции.

Список-выражений представляет собой список фактических параметров, передаваемых в функцию. Этот список может быть и пустым, но наличие круглых скобок обязательно.

Выполнение вызова функции происходит следующим образом:

1. Вычисляются выражения в списке выражений и подвергаются обычным арифметическим преобразованиям. Затем, если известен прототип функции, тип полученного фактического аргумента сравнивается с типом соответствующего формального параметра. Если они не совпадают, то либо производится преобразование типов, либо формируется сообщение об ошибке. Число выражений в списке выражений должно совпадать с числом формальных параметров, если только функция не имеет переменного числа параметров. В последнем случае проверке подлежат только обязательные параметры. Если в прототипе функции указано, что ей не требуются параметры, а при вызове они указаны, формируется сообщение об ошибке.

2. Происходит присваивание значений фактических параметров соответствующим формальным параметрам.

3. Управление передается на первый оператор функции.

4. Выполнение оператора *return* в теле функции возвращает управление и возможно, значение в вызывающую функцию. При отсутствии оператора return управление возвращается после выполнения последнего оператора тела функции, а возвращаемое значение не определено.

Задача 8.1. Сколько нужно купить карточек спортлото, чтобы гарантированно угадать все номера?

$$C = \frac{n!}{(n-m)!m!}$$

```
#include <stdio.h>
double fakt(int);
int main(int argc, char* argv[])
{
  int n=49, m=6;
  printf("Count tickets = %-10.0lf", fakt(n)/fakt(n-m)/fakt(m));
  getchar();
  return 0;
}
double fakt(int x)
{
  double p=1;
  for (int i=1; i<=x; i++)
    p*=i;
  return p;
}
```

Перед определением функции может находиться директива inline. Например:

```
inline int abs( int obj )
{
   // возвращает абсолютное значение obj
   return( obj < 0 ? -obj : obj );
}
```

В этом случае компилятор подставляет в точку вызова тело функции.

Указатели на функцию

Адресное выражение, стоящее перед скобками определяет адрес вызываемой функции. Это значит, что функция может быть вызвана через указатель на функцию.

Пример:

*int (*fun)(int x, int *y);*

Здесь объявлена переменная *fun* как указатель на функцию с двумя параметрами: типа *int* и указателем на *int*. Сама функция должна возвращать значение типа *int*. Круглые скобки, содержащие имя указателя fun и признак указателя *, обязательны, иначе запись

*int *fun (int x, int *y);*

будет интерпретироваться как объявление функции fun возвращающей указатель на *int*.

Вызов функции возможен только после инициализации значения указателя fun и имеет вид:

*(*fun)(i, &j);*

В этом выражении для получения адреса функции, на которую ссылается указатель *fun*, используется операция разадресации *****.

Указатель на функцию может быть передан в качестве параметра функции. При этом разадресация происходит во время вызова функции, на которую ссылается указатель на функцию. Присвоить значение указателю на функцию можно в операторе присваивания, употребив имя функции без списка параметров.

Пример:

```
double (*fun1)(int x, int y);
double fun2(int k, int l);
fun1=fun2;        /* инициализация указателя на функцию */
(*fun1)(2,7); /* обращение к функции */
```

В рассмотренном примере указатель на функцию *fun1* описан как указатель на функцию с двумя параметрами, возвращающую значение типа *double*, и также описана функция *fun2*. В противном случае, т.е. когда указателю на функцию присваивается функция, описанная иначе, чем указатель, произойдет ошибка.

Рассмотрим пример использования указателя на функцию в качестве параметра функции вычисляющей производную от функции *cos(x)*.

Пример:

```
#include <stdio.h>
#include <math.h>
  double proiz(double x, double dx, double (*f)(double x) );
  double fun(double z);
  int main()
  {
    double x;      /*  точка вычисления производной */
    double dx;     /*  приращение                   */
    double z;      /*  значение производной         */
    scanf("%lf%lf", &x, &dx);/*ввод значений x и dx  */
    z=proiz(x, dx, fun);      /*  вызов функции     */
    printf("%f", z); /* печать значения производной  */
    return 0;
  }
  double proiz(double x, double dx, double (*f)(double z) )
  { /* функция, вычисляющая производную  */
     double xk,xk1,pr;
     xk=(*f)(x);
     xk1=(*f)(x+dx);

     pr=(xk1/xk-1.)*xk/dx;
```

$$\frac{f(x+dx)-f(x)}{dx}=\left(\frac{f(x+dx)}{f(x)}-1\right)*\frac{f(x)}{dx}$$

```
  return pr;
}
double fun( double z)
{    /* функция от которой вычисляется производная  */
  return (cos(z));
}
```

Для вычисления производной от какой-либо другой функции можно изменить тело функции *fun* или использовать при вызове функции *proiz* имя другой функции. В частности, для вычисления производной от функции *cos(x)* можно вызвать функцию *proiz* в форме

z=proiz(x,dx,cos);

а для вычисления производной от функции *sin(x)* в форме

z=proiz(x,dx,sin);

Рекурсия

Любая функция в программе на языке СИ может быть вызвана рекурсивно, т.е. она может вызывать саму себя. Компилятор допускает любое число рекурсивных вызовов. При каждом вызове для формальных параметров и переменных с классом памяти *auto* и *register* выделяется новая область памяти, так что их значения из предыдущих вызовов не теряются, но в каждый момент времени доступны только значения текущего вызова.

Переменные, объявленные с классом памяти *static*, не требуют выделения новой области памяти при каждом рекурсивном вызове функции и их значения доступны в течение всего времени выполнения программы.

Классический пример рекурсии – это математическое определение факториала *n!*:

n! = 1 при n=0;
n(n-1)! при   n>1 .*

Функция, вычисляющая факториал, будет иметь следующий вид:

```
long fakt(int n)
{
return ( (n==1) ? 1 : n*fakt(n-1)  );
}
```

Хотя компилятор языка СИ не ограничивает число рекурсивных вызовов функций, это число ограничивается ресурсом памяти компьютера и при слишком большом числе рекурсивных вызовов может произойти переполнение стека.

Предварительная инициализация параметров функции

Список параметров в определении и прототипе функции, кроме согласования типов параметров, имеет еще одно назначение.

Объявление параметра может содержать инициализатор, то есть выражение, которое должно обеспечить параметру присвоение начального значения. Инициализатор параметра не является константным выражением. Начальная инициализация параметров происходит не на стадии компиляции (как, например, выделение памяти под массивы), а непосредственно в ходе выполнения программы.

Следующие строки демонстрируют пример объявления функции с инициализацией параметров. Для инициализации параметра ww используется функция *XX*.

```
int BigVal;
int XX(int);
int ZZ(int tt, int ww = XX(BigVal));
```

Второй параметр можно проинициализировать и таким способом, вовсе не указывая его имени. Синтаксис объявления позволяет сделать и такое!

```
int ZZ(int tt, int = XX(BigVal));
```

Единственное условие подобной инициализации — соответствие типа параметра и типа выражения, значение которого используется при начальной инициализации.

Прототипы функции могут располагаться в различных областях видимости. Его можно даже разместить в теле определяемой функции. Каждое объявление функции может содержать собственные варианты объявления и инициализации параметров. Но во множестве объявлений одной и той же функции в пределах одной области видимости не допускается повторная инициализация параметров. Всему должен быть положен разумный предел.

Кроме того, в C++ действует еще одно ограничение, связанное с порядком инициализации параметров в пределах области видимости. Инициализация проводится непременно с самого последнего (самого правого) параметра в списке объявлений параметров. Инициализация параметров не допускает пропусков: инициализированные параметры не могут чередоваться с параметрами неинициализированными.

```
int MyF1 (int par1, int par2, int par3, int par4 = 10);
……..
int MyF1 (int par1, int par2 = 20, int par3 = 20, int par4);
```

```
……..
int MyF1 (int par1 = 100, int, int, int);
```

Пример:

```
#include <stdio.h>
int f(int, int=4);
int main(int argc, char* argv[])
{
  printf("%d\n", f(2));                    //8
  printf("%d\n", f(2,3));          //6
  getchar();
  return 0;
}
int f(int a, int b)
{
 return a*b;
}
```

Функции с переменным числом параметров

При вызове функции с переменным числом параметров в вызове этой функции задается любое требуемое число аргументов. В объявлении и определении такой функции переменное число аргументов задается многоточием в конце списка формальных параметров или списка типов аргументов.

Все аргументы, заданные в вызове функции, размещаются в стеке. Количество формальных параметров, объявленных для функции, определяется числом аргументов, которые берутся из стека и присваиваются формальным параметрам. Программист отвечает за правильность выбора дополнительных аргументов из стека и определение числа аргументов, находящихся в стеке.

Примерами функций с переменным числом параметров являются функции из библиотеки функций языка СИ, осуществляющие операции ввода-вывода информации (*printf, scanf* и т.п.). Функция *printf()* в библиотеке объявлена следующим образом:

int printf(const char, ...);*

Это гарантирует, что при любом вызове *printf()* ей будет передан первый аргумент типа *const char*.* Содержание такой строки, называемой форматной, определяет, необходимы ли дополнительные аргументы при вызове. При наличии в строке формата метасимволов, начинающихся с символа %, функция ждет присутствия этих аргументов.

Программист может разрабатывать свои функции с переменным числом параметров. Для обеспечения удобного способа доступа к аргументам функции с переменным числом параметров

имеются три макроопределения (макросы) *va_start, va_arg, va_end*, находящиеся в заголовочном файле *stdarg.h*. Эти макросы указывают на то, что функция, разработанная пользователем, имеет некоторое число обязательных аргументов, за которыми следует переменное число необязательных аргументов. Обязательные аргументы доступны через свои имена как при вызове обычной функции. Для извлечения необязательных аргументов используются макросы *va_start, va_arg, va_end* в следующем порядке.

Макрос *va_start* предназначен для установки аргумента *arg_ptr* на начало списка необязательных параметров и имеет вид функции с двумя параметрами:

void va_start(arg_ptr, prav_param);

Параметр prav_param должен быть последним обязательным параметром вызываемой функции, а указатель *arg_prt* должен быть объявлен с предопределением в списке переменных типа *va_list* в виде:

va_list arg_ptr;

Макрос *va_start* должен быть использован до первого использования макроса *va_arg*.

Макрокоманда *va_arg* обеспечивает доступ к текущему параметру вызываемой функции и тоже имеет вид функции с двумя параметрами

type_arg va_arg(arg_ptr,type);

Эта макрокоманда извлекает значение типа *type* по адресу, заданному указателем *arg_ptr*, увеличивает значение указателя *arg_ptr* на длину использованного параметра (длина *type*) и, таким образом, параметр *arg_ptr* будет указывать на следующий параметр вызываемой функции. Макрокоманда *va_arg* используется столько раз, сколько необходимо для извлечения всех параметров вызываемой функции.

Макрос *va_end* используется по окончании обработки всех параметров функции и устанавливает указатель списка необязательных параметров на ноль (*NULL*).

Пример 1:

```
#include <stdarg.h>
#include <stdio.h>
int main()
{ int n;
  int sred_znach(int,...);
n=sred_znach(2,3,4,-1);/* вызов с четырьмя параметрами */
  printf("n=%d\n",n);
```

```
  n=sred_znach(5,6,7,8,9,-1);/* вызов с шестью параметрами   */
  printf("n=%d\n",n);
  getchar();
  return (0);
}
int sred_znach(int x,...)
{
   int i=0,  j=0,  sum=0;
   va_list uk_arg;
va_start(uk_arg,x);/*  установка указателя uk_arg на  */
                /*  первый необязятельный параметр */
 if (x!=-1) sum=x; /*  проверка на пустоту списка     */
   else return (0);
   j++;
   while ( (i=va_arg(uk_arg,int))!=-1)
                        /* выборка очередного   */
   {                    /* параметра и проверка */
     sum+=i;                 /* на конец списка      */
     j++;
   }
   va_end(uk_arg);    /* закрытие списка параметров   */
   return (sum/j);
}
```

Пример 2: Возможен другой вариант – первый параметр функции отвечает за количество суммируемых элементов. Кроме того, можно отказаться от применения макросов и разобрать стек в коде функции.

```
#include <stdarg.h>
#include <stdio.h>
int main()
{int n;
  int sred_znach2(int,...);
  n=sred_znach2(3,2,3,4);
  printf("n=%d\n",n);
  getchar();
  return (0);
}
int sred_znach2(int n,...)
{
  int *pPointer = &n;
  int Sum = 0;
  for (int i=n ; i; i--) Sum += *(++pPointer);
  return Sum/n;
}
```

Объявления функций

Функции всегда определяются глобально. Они могут быть объявлены с классом памяти *static* или *extern*. Объявления функций на локальном и глобальном уровнях имеют одинаковый смысл.

Правила определения области видимости для функций отличаются от правил видимости для переменных и состоят в следующем.

1. Функция, объявленная как *static*, видима в пределах того файла, в котором она определена. Каждая функция может вызвать другую функцию с классом памяти *static* из своего исходного файла, но не может вызвать функцию, определенную с классом *static* в другом исходном файле. Разные функции с классом памяти *static* имеющие одинаковые имена могут быть определены в разных исходных файлах, и это не ведет к конфликту.

2. Функция, объявленная с классом памяти *extern*, видима в пределах всех исходных файлов программы. Любая функция может вызывать функции с классом памяти *extern*.

3. Если в объявлении функции отсутствует спецификатор класса памяти, то по умолчанию принимается класс *extern*.

Все объекты с классом памяти *extern* компилятор помещает в объектном файле в специальную таблицу внешних ссылок, которая используется редактором связей для разрешения внешних ссылок. Часть внешних ссылок порождается компилятором при обращениях к библиотечным функциям СИ, поэтому для разрешения этих ссылок редактору связей должны быть доступны соответствующие библиотеки функций.

Время жизни и область видимости программных объектов

Время жизни переменной (глобальной или локальной) определяется по следующим правилам.

1. Переменная, объявленная глобально, т.е. вне всех блоков (блок – часть кода, которая сгруппирована и воспринимается как единое целое, записанная в операторных скобках *{…}*), существует на протяжении всего времени выполнения программы.

2. Локальные переменные (т.е. объявленные внутри блока) с классом памяти *register* или *auto*, имеют время жизни только на период выполнения того блока, в котором они объявлены. Если локальная переменная объявлена с классом памяти *static* или *extern*, то она имеет время жизни на период выполнения всей программы.

Видимость переменных и функций в программе определяется следующими правилами.

1. Переменная, объявленная или определенная глобально, видима от точки объявления или определения до конца исходного файла. Можно сделать переменную видимой и в других исходных файлах, для чего в этих файлах следует ее объявить с классом памяти extern.

2. Переменная, объявленная или определенная локально, видима от точки объявления или определения до конца текущего блока. Такая переменная называется локальной.

3. Переменные из объемлющих блоков, включая переменные объявленные на глобальном уровне, видимы во внутренних блоках. Эту видимость называют вложенной. Если переменная, объявленная внутри блока, имеет то же имя, что и переменная, объявленная в объемлющем блоке, то это разные переменные, и переменная из объемлющего блока во внутреннем блоке будет невидимой.

4. Функции с классом памяти **static** видимы только в исходном файле, в котором они определены. Всякие другие функции видимы во всей программе.

Метки в функциях видимы на протяжении всей функции.

Имена формальных параметров, объявленные в списке параметров прототипа функции, видимы только от точки объявления параметра до конца объявления функции.

Для некоторых инструкций языка C++ разрешено объявлять переменные внутри управляющей части. Например, в цикле for переменную можно определить внутри инструкции инициализации. Подобные переменные видны только в локальной области самого цикла **for** и вложенных в него (это верно для стандарта C++, в предыдущих версиях языка поведение было иным). Аналогично переменная может быть объявлена внутри условия инструкций *if* и *switch*, а также внутри условия циклов *while* и *for*.

```
if ( int *pi = getValue() )
{  // pi != 0 -- *pi можно использовать здесь
}
else
{  // здесь pi тоже видна
}
```

Инициализация глобальных и локальных переменных

При инициализации необходимо придерживаться следующих правил:

1. Объявления, содержащие спецификатор класса памяти **extern**, не могут содержать инициаторов.

2. Глобальные переменные всегда инициализируются, и если это не сделано явно, то они инициализируются нулевым значением.

3. Переменная с классом памяти *static* может быть инициализирована константным выражением. Инициализация для них выполняется один раз. Если явная инициализация отсутствует, то переменная инициализируется нулевым значением.

4. Инициализация переменных с классом памяти *auto* или *register* выполняется всякий раз при входе в блок, в котором они объявлены. Если инициализация переменных в объявлении отсутствует, то их начальное значение не определено.

5. Начальными значениями для глобальных переменных и для переменных с классом памяти *static* должны быть константные выражения. Адреса таких переменных являются константами и эти константы можно использовать для инициализации объявленных глобально указателей. Адреса переменных с классом памяти *auto* или *register* не являются константами и их нельзя использовать в инициаторах.

Пример:

```
int global_var;
int func(void)
{ int local_var;                /* по умолчанию auto  */
static int *local_ptr=&local_var;   /*  так неправильно   */
static int *global_ptr=&global_var; /*  а так правильно   */
register int *reg_ptr=&local_var;   /*  и так правильно   */
}
```

В приведенном примере глобальная переменная *global_var* имеет глобальное время жизни и постоянный адрес в памяти, и этот адрес можно использовать для инициализации статического указателя *global_ptr*. Локальная переменная *local_var*, имеющая класс памяти **auto** размещается в памяти только на время работы функции *func*, адрес этой переменной не является константой и не может быть использован для инициализации статической переменной *local_ptr*. Для инициализации локальной регистровой переменной *reg_ptr* можно использовать неконстантные выражения, и, в частности, адрес переменной *local_ptr.*

Задача 8.2. N спортсменов на соревнованиях совершают по M попыток в тройных прыжках в длину. Определить средний результат по всем попыткам для каждого спортсмена и лучший результат среди всех спортсменов и каким спортсменом он был показан? Нахождение результатов оформить в подпрограмме.

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

void init(float**,int,int);
int poisk(float**,int,int,float&);
void print(float**,int,int);

int main()
{
 randomize();
 int n, m, num;
 cout<<"Enter count sportsman: ";
 cin>>n;
 float bestrez=0,
      **rez=(float**)calloc(n, sizeof(float*));
 cout<<"Enter count jump: ";
 cin>>m;
 for(int i=0;i<n;i++)
   rez[i]=(float*)calloc(m+1,sizeof(float));
 init(rez,n,m);
 num=poisk(rez,n,m,bestrez);
 print(rez,n,m);
 cout<<"The best jump "<<num<<" - sportsman " <<
setprecision(4)<<bestrez<<" m.!";
 for(int i=0;i<n;i++)
  free(rez[i]);
 free(rez);
 getchar();getchar();
   return 0;
}

void init(float **a,int k,int l)
{
 for(int i=0;i<k;i++)
  for(int j=0;j<l;j++)
    a[i][j]=(float)rand()/RAND_MAX*4+13;
}

int poisk(float **a, int k, int l, float &bestrez)
{
 int nn=0;
 for(int i=0;i<k;i++)
   {
    a[i][l]=0;
    for(int j=0;j<l;j++)
    {
     a[i][l]+=a[i][j];
```

```
            if (a[i][j]>bestrez)
             {
              bestrez=a[i][j];
              nn=i+1;
             }
            }
           a[i][l]/=l;
          }
          return nn;
        }

        void print(float **a,int k,int l)
        {
         cout<<"Result: "<<endl;
         for(int i=0;i<k;i++)
          {
           cout<<"Sportsman "<<setw(4)<<(i+1)<<": ";
           for(int j=0;j<l+1;j++)
            j<l?(cout<<setw(7)<<setprecision(4)<<a[i][j]):(cout<<setw(7) <<"
sr="<<setprecision(6)<<a[i][l]);
           cout<<endl;
          }
        }
```

Результаты работы программы приведены на рис. 8.1.

```
Enter count sportsman: 4
Enter count jump: 5
Result:
Sportsman    1:   16.36  13.93  14.11  15.71  13.43    sr=14.7106
Sportsman    2:   14.57  13.65  15.03  16.07  13.29    sr=14.5224
Sportsman    3:   14.06  14.45  13.22  14.99  14.41    sr=14.2242
Sportsman    4:   15.45  13.02  16.26  16.64  14.73    sr=15.2192
The best jump 4 - sportsman 16.64 m.!
```

Рис. 8.1. Результат работы программы (задача 8.2)

Задание для индивидуального выполнения по теме «Работа с функциями».

Задание И.8.1.

1. Дано *N* десятков целых чисел. Определить, сколько из них могут составлять геометрическую прогрессию. Проверку оформить в виде функции.
2. Дано N десятков целых чисел. Определить, сколько из них могут составлять арифметическую прогрессию. Проверку оформить в виде функции.
3. Дано *N* десятков целых чисел. Определить, сколько из них могут составлять ряд Фибоначчи. Первое число Фибоначчи равно

0, второе – 1. Каждое последующее равно сумме двух предыдущих. Проверку оформить в виде функции.

4. Дано *N* пар чисел, представляющих собой координаты точек на плоскости. Найти *R* – радиус наименьшей окружности с центром в начале координат, в которую попадают все точки. Определение расстояния от точки до начала координат оформить в виде функции.
5. Известны оценки группы студентов за сессию. В группе 20 студентов, в сессии 4 экзамена. Определить суммарную стипендию. Считать, что стипендия в размере R рублей начисляется студентам, сдавшим сессию без троек, а отличники получают стипендию, повышенную на 25% . Подсчет стипендии студента оформить в виде функции.
6. Известен расход электроэнергии по всем квартирам 24-х квартирного дома. Определить суммарную плату за электричество. При расходе до 100 кВт*ч на человека берется тариф R копеек за 1 кВт*ч, в случае превышения нормы тариф возрастает на 20%. Подсчет платы для квартиры оформить в виде функции.
7. Известна ежемесячная заработная плата персонала предприятия в течение календарного года. Вывести фамилии тех сотрудников, у которых годовая заработная плата выше средней. Считать, что штат предприятия составляет 7 человек. Подсчет годовой зарплаты работника оформить в виде функции.
8. Известна ежемесячная заработная плата персонала предприятия в течение календарного года. Вывести фамилии сотрудников с минимальной и максимальной годовой заработной платой. Считать, что штат предприятия составляет 8 человек. Подсчет годовой зарплаты работника оформить в виде функции.
9. Дан одномерный массив из 100 случайных целых чисел в диапазоне от 5 до 25 включительно. Вывести все числа, которые максимально часто встречаются в массиве и количество их повторений. Подсчет количества повторений для числа оформить в виде функции.
10. Дан одномерный массив из 150 случайных целых чисел в диапазоне от 14 до 37 включительно. Вывести те числа, которые наиболее редко встречаются в массиве и количество их повторений. Подсчет количества повторений для числа оформить в виде функции.
11. Дан одномерный массив из 50 случайных целых чисел в диапазоне от 10 до 85 включительно. Вывести в порядке возрастания те числа из данного диапазона, которые ни разу не

встречаются в массиве. Создать функцию для поиска элемента в массиве.

12. Дан одномерный массив из 40 случайных целых чисел в диапазоне от 16 до 89 включительно. Вывести минимальное и максимальное числа из данного диапазона, которые ни разу не встречаются в массиве. Создать функцию для поиска элемента в массиве.

ЛАБОРАТОРНАЯ РАБОТА № 9. ДИНАМИЧЕСКИЕ ОБЪЕКТЫ

Динамические объекты отличаются от статических тем, что память на них выделяется не во время компиляции программы, а во время ее выполнения. Выделение и освобождение памяти можно осуществлять двумя способами.

1. Выделение памяти в соответствие с типом указателя

Оператор *new* состоит их ключевого слова **new**, за которым следует спецификатор типа. Этот спецификатор может относиться к встроенным типам или к типам классов. Например:

new int;

размещает в памяти один объект типа *int*.

Одним из аспектов работы с памятью является то, что размещаемые в ней объекты не имеют имени. Оператор *new* возвращает не сам объект, а указатель на него. Все манипуляции с этим объектом производятся косвенно через указатели:

*int *pi = new int;*

Здесь оператор *new* создает один объект типа *int*, на который ссылается указатель **pi**. Выделение памяти во время выполнения программы называется динамическим выделением. Мы говорим, что память, адресуемая указателем *pi*, выделена динамически.

Второй аспект, состоит в том, что эта память не инициализируется. Она содержит «мусор», оставшийся после предыдущей работы. Проверка условия:

*if (*pi==0)*

вероятно, даст *false*, поскольку объект, на который указывает *pi*, содержит случайную последовательность битов.

Следовательно, объекты, создаваемые с помощью оператора new, рекомендуется инициализировать.

Программист может инициализировать объект типа *int* из предыдущего примера следующим образом:

*int *pi = new int(0);*

Константа в скобках задает начальное значение для создаваемого объекта; теперь *pi* ссылается на объект типа *int*, имеющий значение *0*. Выражение в скобках называется инициализатором. Это может быть любое выражение (не обязательно константа), возвращающее значение, приводимое к типу *int*.

Оператор *new* выполняет следующую последовательность действий: выделяет память для объекта, затем инициализирует его значением, стоящим в скобках. Для выделения памяти вызывается библиотечная функция **new().** Предыдущий оператор при-

близительно эквивалентен следующей последовательности инструкций:

```
int ival = 0;    // создаем объект типа int
      // и инициализируем его 0
int *pi = &ival; // указатель ссылается на этот объект
```

не считая, конечно, того, что объект, адресуемый *pi*, создается библиотечной функцией *new().*

Описанные операторы *new* могут вызывать одну проблему: хип, к сожалению, является конечным ресурсом, и в некоторой точке выполнения программы мы можем исчерпать его. Если функция *new()* не может выделить затребованного количества памяти, она возбуждает исключение *bad_alloc***.**

Время жизни объекта, на который указывает *pi*, заканчивается при освобождении памяти, где этот объект размещен. Это происходит, когда *pi* передается оператору *delete*.

Например,

```
delete pi;
```

освобождает память, на которую ссылается *pi*, завершая время жизни объекта типа *int*. Программист управляет окончанием жизни объекта, используя оператор *delete* в нужном месте программы. Этот оператор вызывает библиотечную функцию *delete(),* которая возвращает выделенную память в хип. Поскольку хип конечен, очень важно возвращать ее своевременно.

Оператор *delete* может использоваться только по отношению к указателю, который содержит адрес области памяти, выделенной в результате выполнения оператора *new*. Попытка применить *delete* к указателю, не ссылающемуся на такую память, приведет к непредсказуемому поведению программы.

Ниже приведены примеры опасных и безопасных операторов *delete*:

```
void f() {
  int i;
  char str[11] = "dwarves";
  int *pi = &i;
  short *ps = 0;
  double *pd = new doub1e(33);
  delete str; // плохо: str не является динамическим объектом
  delete pi; // плохо: pi ссылается на локальный объект
  delete ps; // безопасно
  delete pd; // безопасно
}
```

Вот три основные ошибки, связанные с динамическим выделением памяти:

– не освободить выделенную память. В таком случае память не возвращается в хип. Эта ошибка получила название утечки памяти;

– дважды применить оператор *delete* к одной и той же области памяти. Такое бывает, когда два указателя получают адрес одного и того же динамически размещенного объекта. В результате подобной ошибки мы вполне можем удалить нужный объект. Действительно, память, освобожденная с помощью одного из адресующих ее указателей, возвращается в хип и затем выделяется под другой объект. Затем оператор *delete* применяется ко второму указателю, адресовавшему старый объект, а удаляется при этом новый;

– изменять объект после его удаления. Такое часто случается, поскольку указатель, к которому применяется оператор *delete*, не обнуляется.

2. Выделение памяти под нетипизированный указатель

Оператор *new* выделяет область памяти размером соответствующим заданному типу. Программист имеет возможность выделить область памяти заданного размера. Для этого используются функции семейства *alloc* (табл. 9.1).

Таблица 9.1

Функции для выделения и освобождения памяти – файлы *sdlib.h, alloc.h*

Функция	Прототип и краткое описание действий
malloc	*void *malloc(unsigned s);* Возвращает указатель на блок динамически распределяемой памяти длиной *s* байт. При неудачном завершении возвращает значение *NULL*. Если размер аргумента == 0, возвращает *NULL*.
calloc	*void *calloc(unsigned n, unsigned m);* Возвращает указатель на начало области динамически распределенной памяти (блок размером *n*m*) для размещения *n* элементов по *m* байт каждый. При неудачном завершении возвращает значение *NULL*. Блок инициализируется нулями.
free	*void free(void *bl);* Освобождает блок динамически распределяемой памяти с адресом первого байта *bl* выделенный функциями *calloc(), malloc(),* или *realloc().*

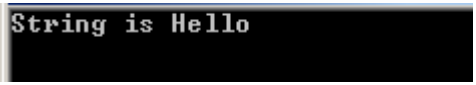
«Продолжение табл. 9.1»

Функция	Прототип и краткое описание действий
realloc	*void *realloc(void *bl, unsigned ns);* Изменяет размер ранее выделенной динамической памяти с адресом начала *bl* до размера *ns* байт. Если *ns* – ноль, блок памяти освобожден, и возвращается *NULL*. Аргумент *bl* указывает на память, предварительно полученный, вызовами malloc, calloc, или realloc. Если *bl* равен *NULL*, то функция выполняется как *malloc().* Копирует содержимое старого блока памяти в новый. Возвращает адрес нового блока памяти, который может отличаться от старого.

Функция *malloc()*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
   char *str;
   // выделение динамической памяти
   if ((str = (char *) malloc(10)) == NULL)
   {
     printf("Not enough memory to allocate buffer\n");
     exit(1); /* память не выделена – завершить работу
    программы */
   }
   strcpy(str, "Hello"); // Копируем строку "Hello" в str
   printf("String is %s\n", str);
   free(str);          /* Освобождение блока динамически
     распределяемой памяти */
   getchar();
   return 0;
}
```

Результаты выполнения программы:

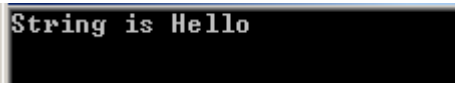


Функция *calloc()*

```
#include <stdio.h>
 #include <stdlib.h>
 #include <string.h>
 int main()
```

```
{
  char *str = NULL;
  // выделение динамической памяти
  str = (char *) calloc(10, sizeof(char));
  strcpy(str, "Hello");  // Копируем строку "Hello" в str
  printf("String is %s\n", str);
  free(str);          /* Освобождение блока динамически
      распределяемой памяти */
  getchar();
  return 0;
}
```

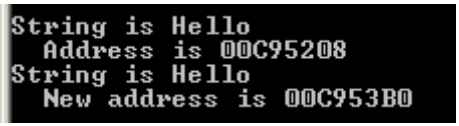
Результаты выполнения программы:



Функция *realloc()*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
  char *str;
  str = (char *) malloc(10); /* выделение блока динамически
распределяемой памяти длиной 10 байт */
  strcpy(str, "Hello");  // Копируем строку "Hello" в str
  printf("String is %s\n  Address is %p\n", str, str);
  str = (char *) realloc(str, 20); /* перевыделение блока
динамически распределяемой памяти длиной 20 байт */
  printf("String is %s\n  New address is %p\n", str, str);
  free(str);          /* Освобождение блока динамически
    распределяемой памяти */
  getchar();
  return 0;
}
```

Результаты выполнения программы:



Динамическое создание и уничтожение массивов

Создание динамического массива при помощи оператора new.

Оператор *new* может выделить из хипа память для размещения массива. В этом случае после спецификатора типа в квадратных скобках указывается размер массива. Он может быть задан сколь угодно сложным выражением. new возвращает указатель на первый элемент массива. Например:

```
// создание единственного объекта типа int
// с начальным значением 1024
int *pi = new int( 1024 );
// создание массива из 1024 элементов
// элементы не инициализируются
int *pia = new int[ 1024 ];
// создание двумерного массива из 4x1024 элементов
int (*pia2)[ 1024 ] = new int[ 4 ][ 1024 ];
```

pi содержит адрес единственного элемента типа *int*, инициализированного значением 1024; *pia* — адрес первого элемента массива из 1024 элементов; *pia2* — адрес начала массива, содержащего четыре массива по 1024 элемента, т.е. *pia2* адресует 4096 элементов.

В общем случае массив, размещаемый в хипе, не может быть инициализирован. Задавать инициализатор при выделении оператором *new* памяти под массив не разрешается.

Основное преимущество динамического массива состоит в том, что количество элементов в его первом измерении не обязано быть константой, т.е. может не быть известным во время компиляции.

```
#include<stdio.h>
#include<string.h>
#include<alloc.h>
const char *string="Welcome to Borland Developer Studio 2006!!!";
int main()
{
  char *str;
  str=(char *)calloc(strlen(string)+1,sizeof(char));
  strcpy(str,string);
  printf("String is: %s\n", str);
  free(str);
  getchar();
  return 0;
}
```

Результаты выполнения программы:

Единица, прибавляемая к значению, которое возвращает *strlen(),* необходима для учета завершающего нулевого символа в С-строке. Отсутствие этой единицы – весьма распространенная ошибка, которую достаточно трудно обнаружить, поскольку она проявляет себя косвенно: происходит затирание какой-либо другой области программы. Большинство функций, которые обрабатывают массивы, представляющие собой С-строки символов, пробегают по элементам, пока не встретят завершающий нуль. Если в конце строки нуля нет, то возможно чтение или запись в случайную область памяти.

Отметим, что только первое измерение массива, создаваемого с помощью оператора new, может быть задано значением, вычисляемым во время выполнения. Остальные измерения должны задаваться константами, известными во время компиляции.

Пример:

```
int getDim();
// создание двумерного массива
int (*pia3)[ 1024 ] = new int[ getDim() ][ 1024 ]; // правильно
// ошибка: второе измерение задано не константой
int **pia4 = new int[ 4 ][ getDim() ];
// создание трехмерного массива
long (*lp)[2][4];        // Определили указатель
lp = new long [3][2][4];  // правильно
lp = new long [];  // ошибка: размер неизвестен
lp = new long [][2][4];  // ошибка: размер неизвестен
lp = new long [3][][4];  // ошибка: размер неизвестен
```

Оператор *delete* для уничтожения массива имеет следующую форму:

```
delete[] str1;
```

Пустые квадратные скобки необходимы. Они говорят компилятору, что указатель адресует массив, а не единичный элемент. Поскольку тип *str1* – указатель на *char*, без этих скобок компилятор не поймет, что удалять следует целый массив.

Отсутствие скобок не является синтаксической ошибкой, но правильность выполнения программы не гарантируется.

Создание динамического массива при помощи функции calloc()

При динамическом распределении памяти для массивов можно описать соответствующий указатель и присваивать ему значение при помощи функции calloc. Одномерный массив *a[10]* из элементов типа *float* можно создать следующим образом

```
float *a;
a=(float*)(calloc(10, sizeof(float));
```

Для создания двумерного массива вначале нужно распределить память для массива указателей на одномерные массивы, а затем распределять память для одномерных массивов. Пусть, например, требуется создать массив *a[m][n],* это можно сделать при помощи следующего фрагмента программы:

```
main ()
{  double **a;
   int n,m,i;
   scanf("%d %d",&n,&m);
   a=(double **)calloc(m, sizeof(double *));
   for (i=0; i<m; i++)
     a[i]=(double *)calloc(n, sizeof(double));
}
```

Аналогичным образом можно распределить память и для трехмерного массива размером *m, n, l*. Следует только помнить, что ненужную для дальнейшего выполнения программы память следует освобождать при помощи функции *free().*

```
main ()
{  long ***a;
   int n, m, l, i, j;
   scanf("%d %d %d", &m, &n, &l);
   /*    распределение памяти  */
   a=(long ***)calloc(m, sizeof(long **) );
   for (i=0; i<m; i++)
   {          a[i]=(long **)calloc(n, sizeof(long *) );
              for (j=0; j<n; j++)
                        a[i][j]=(long *)calloc(l, sizeof(long) );
   }
   . . . . . .
   /*    освобождение памяти      */
   for (i=0; i<m; i++)
   {          for (j=0; j<n; j++)
                        free (a[i][j]);
              free (a[i]);
   }
   free (a);
}
```

Отметим следующее обстоятельство, что указатель на массив не обязательно должен показывать на начальный элемент некоторого массива. Он может быть сдвинут так, что начальный элемент будет иметь индекс отличный от нуля, причем он может быть как положительным так и отрицательным.

Пример:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{  float *q, **b;
   int i, j, k, n, m;
   scanf("%d %d", &n, &m);
   q=(float *)calloc(m, sizeof(float));
/* сейчас указатель q показывает на начало массива */
   q[0]=22.3;
   q-=5;
/* теперь начальный элемент массива имеет индекс 5,*/
/* а конечный элемент индекс n-5                   */
   q[5]=1.5;
/* сдвиг индекса не приводит к перераспределению   */
/* массива в памяти и изменится начальный элемент  */
   q[6]=2.5;   /*  -  это второй элемент            */
   q[7]=3.5;   /*  -  это третий элемент            */
   q+=5;
/* теперь начальный элемент вновь имеет индекс 0,  */
/* а значения элементов q[0], q[1], q[2] равны     */
/* соответственно 1.5, 2.5, 3.5                    */
   q+=2;
/* теперь начальный элемент имеет индекс -2,       */
/* следующий -1, затем 0 и т.д. по порядку         */
   q[-2]=8.2;
   q[-1]=4.5;
   q-=2;
/* возвращаем начальную индексацию, три первых     */
/* элемента массива q[0], q[1], q[2], имеют        */
/* значения 8.2, 4.5, 3.5                          */
   q--;
/* вновь изменим индексацию.                       */
/* Для освобождения области памяти, в которой размещен */
/* массив q используется функция free(q), но поскольку */
/* значение указателя q смещено, то выполнение     */
/* функции free(q) приведет к непредсказуемым последствиям.*/
/* Для правильного выполнения этой функции         */
/* указатель q должен быть возвращен в первоначальное */
/* положение                                       */
   free(++q);
```

```
/* Рассмотрим возможность изменения индексации и      */
/* освобождения памяти для двумерного массива         */
   b=(float **)calloc(m, sizeof(float *));
   for (i=0; i < m; i++)
            b[i]=(float *)calloc(n, sizeof(float));
/* После распределения памяти начальным элементом     */
/* массива будет элемент b[0][0]                      */
/* Выполним сдвиг индексов так, чтобы начальным       */
/* элементом стал элемент b[1][1]                     */
   for (i=0; i < m ; i++) --b[i];
   b--;
/* Теперь присвоим каждому элементу массива сумму его */
/* индексов                                           */
   for (i=1; i<=m; i++)
            for (j=1; j<=n; j++)
                     b[i][j]=(float)(i+j);
/* Обратите внимание на начальные значения счетчиков  */
/* циклов i и j, он начинаются с 1 а не с 0           */
/* Возвратимся к прежней индексации                   */
   for (i=1; i<=m; i++) ++b[i];
      b++;
/* Выполним освобождение памяти                       */
   for (i=0; i < m; i++)  free(b[i]);
            free(b);
return 0;
}
```

Задание для индивидуального выполнения по теме «Работа с динамическими массивами»

Задание И.9.2. Предусмотреть выделение памяти для динамических массивов с использованием оператора new, функции calloc() и освобождение выделенной памяти.

1. Для группы учащихся известны годовые оценки по следующим предметам: математика, физика, химия, информатика. Отобрать кандидатов на олимпиады (с отличными оценками) по каждому из предметов. Отбор кандидатов по предмету оформить в подпрограмме.
2. Для группы учащихся известны годовые оценки по следующим предметам: математика, физика, химия, информатика. Найти среднюю в группе оценку по каждому из предметов. Суммирование оценок по предмету оформить в подпрограмме.
3. Спортсмены на соревнованиях совершают 6 попыток в прыжках в длину. Определить лучший результат для каждого участника. Нахождение лучшего результата для спортсмена оформить в подпрограмме.

4. Для группы учащихся известны годовые оценки по следующим предметам: математика, физика, химия, информатика. Найти в группе среднюю оценку для каждого учащегося. Суммирование оценок учащихся оформить в подпрограмме.
5. Для группы фирм известен курс их акций за каждый из месяцев календарного года. Составить список тех фирм, курс акций которых все время повышался (т.е. курс за каждый последующий месяц больше, чем за предыдущий). Проверку роста курса осуществить в подпрограмме.
6. Выступления *N* спортсменов оцениваются *M* судьями по одной и той же числовой шкале. Нужно узнать конечный результат каждого спортсмена, если он вычисляется так: из всех M оценок удаляются максимальная и минимальная (если таких оценок несколько, то удаляется только одна), затем из оставшихся (*M-2)* находится их среднее арифметическое. Вычисление конечного результата спортсмена оформить в подпрограмме.
7. Известна среднемесячная температура воздуха на *N* островах Карибского моря. Определить, на каком из островов среднегодовая температура максимальна. Вычисление среднегодовой температуры оформить в подпрограмме.
8. Известна среднемесячная температура воздуха на *N* островах Карибского моря. Определить для каждого из месяцев, на каком из островов максимальная температура. Нахождение максимальной температуры месяца оформить в подпрограмме.
9. Дан двумерный массив из *N* строк и *M* столбцов. Определить для каждого четного столбца максимальный элемент. Найти произведение этих элементов.
10. Дан двумерный массив из *N* строк и *M* столбцов. Определить, какая строка массива имеет максимальную сумму элементов (считать, что строк с одинаковой суммой нет).
11. Дан двумерный массив из *N* строк и *M* столбцов. Определить, для каждой нечетной строки минимальный элемент. Найти произведение этих элементов.
12. Дан двумерный массив из *N* строк и *M* столбцов. Определить для каждой строки минимальный элемент. Среди этих элементов найти максимальный.
13. Дан двумерный массив из *N* строк и *M* столбцов. Определить для каждого столбца максимальный элемент. Среди этих элементов найти минимальный.
14. Дан двумерный массив из *N* строк и *M* столбцов. Определить для каждого столбца массива сумму минимального и максимального элементов. Найти произведение этих сумм.

15. Дан двумерный массив из *N* строк и *M* столбцов. Определить для каждой строки массива произведение минимального и максимального элементов. Найти сумму этих произведений.
16. Дан двумерный массив из *N* строк и *M* столбцов. Определить для каждой строки массива произведение элементов. Найти, в какой строке это произведение максимально.
17. Дан двумерный массив из *N* строк и *M* столбцов. Определить, какой столбец массива имеет минимальную сумму элементов (считать, что столбцов с одинаковой суммой нет).
18. Известны места *N* лыжников на каждом из *M* стартов Кубка мира. Определить победителя в общем зачете (с минимальной суммой мест). Если таких несколько, то победителем считается тот, кто лучше других претендентов на первое место выступил на последнем этапе.
19. По результатам метеорологических наблюдений за *N* последних лет известно количество солнечных дней в году для *M* морских курортов. Кроме этого известны расстояния до них. Определить курорт с наиболее благоприятным климатом (с максимальным суммарным количеством солнечных дней за время наблюдений). Если таких несколько, то вывести ближайший из них.
20. В автопарке находится *N* автомобилей. Известен их пробег в течение каждого из *M* рабочих дней. Определить, какой из автомобилей за рабочую неделю преодолел максимальное расстояние. Если таких несколько, то вывести хотя бы одного из них.
21. Бригада рабочих состоит из *N* человек. Известно сколько деталей выпустил каждый из них в течение каждого из *M* рабочих дней. Определить, какой рабочий произвел максимальное количество деталей. Если таких несколько, то вывести хотя бы одного из них.
22. На каждом этаже *M*-этажного дома жилых *N* комнат, каждая из которых имеет форму прямоугольника. Длина и ширина каждой комнаты известны. Определить, какой из этажей дома имеет минимальную жилую площадь. Если таких несколько, то вывести хотя бы одного из них.
23. Известны итоги сдачи экзаменов группой из *N* студентов по *M* предметам. Определить, по результатам каких экзаменов в группе максимальный средний балл.
24. В соревнованиях по многоборью участвуют *N* спортсменов. Известны результаты в баллах по каждому из видов. Вывести фамилии спортсменов и их результаты в сумме в порядке занятых мест.

25. Известен курс акций *N* фирм за каждый из *M* месяцев. За второе полугодие для каждой фирмы определить месяц максимального прироста курса. Если таких несколько, то вывести последний из них.

ЛАБОРАТОРНАЯ РАБОТА №10. РАБОТА СО СТРОКАМИ

Массивы для хранения строк

В этой главе описывается способ представления строк символов, унаследованный языком C++ от языка C. Строки C по-прежнему широко используются и являются неотъемлемой частью языка C++.

Одним из способов представления строк является использование массива базового типа *char*. Например, строку "*Hello*" удобно представить как массив из шести индексированных переменных: пяти букв слова *"Hello"* и одного нулевого символа *'\0'*, служащего маркером конца строки. Символ *'\0'* называется *нуль-символом* или *нулевым символом,* а когда он используется в качестве маркера конца строки – *нуль-терминатором.* При использовании таких маркеров программа может считывать массивы посимвольно и знать, когда следует остановиться. Строка, хранящаяся в описанном формате, называется строкой *C*. В программе C++ нуль-символ записывается как *'\0'*, то есть в виде двух символов, но на самом деле, подобно символу новой строки *'\n'*, он является одним символом. Как и любое другое символьное значение, он может храниться в переменной типа char или элементе массива с базовым типом char.

Строковая переменная C представляет собой просто массив символов. Так, следующее объявление массива:

char s[10];

создает строковую переменную *C*, в которой может храниться строковое значение *C*, состоящее из десяти или менее символов.

Массив длиной десять символов вмещает строку из девяти символов и нуль-символ *'\0'*, отмечающий ее конец.

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]
H	e	l	l	o	\0	?	?	?	?

Объявление строковой переменной C

Строковая переменная *C* — это обычный массив символов, используемый особым образом. Она объявляется так же, как любой другой массив символов.

Синтаксис

char имя_массива[максимальный_размер_строки_C + 1];

Пример

char my_string[ll];

Единица прибавляется для того, чтобы массив вмещал нуль-символ *'\0'*, отмечающий конец хранящейся в массиве строки. В частности, строковая переменная *my_string* в приведенном выше примере вмещает строку *C* длиной в десять или менее символов.

Инициализируя строковую переменную *C*, можно опустить размер массива. C++ автоматически присвоит ей размер, который будет на единицу больше, чем длина заключенной в кавычки строки (один дополнительный элемент массива для символа '\0').

Например, объявление

```
char short_string[] = "abc";
```

эквивалентно объявлению

```
char short_string[4] = "abc"
```

Однако не путайте следующие две инициализации:

```
char short_string[] = "abc";
```

и

```
char short_string[] = {'a', 'b', 'c'};
```

Они не равнозначны. Первая помещает после символа *'c*' символ '*\0*', а вторая этого не делает (она вообще не помещает в массив символ *'\0*' – ни после *'c'*, ни в каком-либо другом месте).

Поскольку строковая переменная *C* является массивом, она состоит из набора элементов, с которыми можно работать по отдельности.

Для примера рассмотрим следующий фрагмент программы:

```
int index = 0;
while (my_string[index] != '\0')
  {
    my_string[index] = 'x';
    index++;
  }
```

Данный код изменяет строковое значение, хранящееся в переменной *my_string,* помещая в нее строку *C*, состоящую только из символов *'x*'.

Инициализация строковой переменной C

Строковую переменную *C* можно инициализировать при объявлении, как в следующем примере:

```
char my_string[] = "Hello, world!";
```

Эта инициализация автоматически помещает в конец строки *C* символ *'\0'*. Если в квадратных скобках не задано число, создается массив, длина которого на единицу больше длины помещаемой в него строки. Так, приведенный оператор объявляет массив *my_string* из четырнадцати индексированных переменных (трина-

дцать для символов строки *"Hello, world!"* и один для нуль-символа *'\0'*).

Строковые значения и переменные *C* отличаются от значений и переменных других типов данных, и многие операции языка C++ к ним неприменимы. Так, нельзя использовать строковую переменную *C* в операторе присваивания. Если же попытаться сравнить две строки *C* посредством оператора ==, результат будет не таким, как ожидалось. Это объясняется тем, что строки *C* являются массивами. Присваивание значения строковой переменной *C* выполняется не так просто, как другим переменным C++. Скажем, следующий оператор присваивания:

```
char my_string[10];
my_string = "Hello";
```

недопустим.

Хотя в объявлении такой переменой можно пользоваться знаком равенства для ее инициализации, больше нигде в программе это не разрешается. Знак равенства в объявлении переменной

```
char my_str[7] = "Hello";
```

означает именно инициализацию, а не присваивание, которое выполняется иначе. Существуют разные способы присваивания значения строковой переменной *C*, простейший из них заключается в вызове стандартной функции *strcpy*:

```
strcpy(my_str, "Hello");
```

Этот вызов присваивает переменной *my_str* строку "*Hello*". Данная версия функции strcpy не проверяет, превышает ли размер строки размер строковой переменной.

Во многих реализациях C++ имеется более безопасная версия данной функции, называемая strncpy (с буквой *n*). У нее есть третий аргумент, в котором задается максимальное количество копируемых символов.

Например:

```
char a_str[10];
strncpy(a_str, a_str_v, 9);
```

Вызов копирует максимум девять символов из строковой переменной *a_str_v* (независимо от ее длины) в строковую переменную *a_str*.

Проверку эквивалентности двух строковых переменных *C* нельзя выполнять обычным способом (то есть с помощью оператора ==). Поэтому если применить его для сравнения двух строк, результат окажется неверным, и при этом даже не будет выведено сообщение об ошибке. Сравнение двух строк *C* на эквивалент-

ность выполняется с помощью стандартной функции *strcmp*. Вот так:

```
if (strcmp(c_str1, c_str2))
cout « "The strings are NOT the same.";
else
cout « "The strings are the same.";
```

Функции *strcpy* и *strcmp* располагаются в библиотеке с заголовочным файлом <*string.h*>. Для их использования нужно добавить в начало программы следующую директиву:

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

Функции из библиотеки string

Описания некоторых наиболее популярных функций из библиотек с заголовочным файлом *string.h* и *sdlib.h* приведены в табл. 10.1. Для их использования в начало файла программы нужно включить директиву

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

или

```
#include < sdlib.h >
```

Таблица 10.1

Функции для работы со строками – файлы *string.h, sdlib.h*

Функция	Прототип и краткое описание действий
atof	*double atof(char *str);* Преобразует строку *str* в вещественное число типа *double*.
atoi	*int atoi(char *str);* Преобразует строку *str* в десятичное целое число.
atol	*long atol(char *str);* Преобразует строку *str* в длинное десятичное целое число.
itoa	*char *itoa(int v, char *str, int baz);* Преобразует целое *v* в строку *str*. При изображении числа используется основание *baz*. Для отрицательного числа и *baz=10* первый символ – «минус» (-).
ltoa	*char *ltoa (long v, char *str, int baz);* Преобразует длинное целое *v* в строку *str*. При изображении числа используется основание *baz*

«Продолжение табл. 10.1»

Функция	Прототип и краткое описание действий
strcat	*char *strcat (char *sp, char *si);* Приписывает строку *si* к строке *sp* (конкатенация строк).
strchr	*char *strchr (char *str, int c);* Ищет в строке *str* первое вхождение символа *c*.
strcmp	*int strcmp (char *str1, char *str2);* Сравнивает строки *str1* и *str2*. Результат отрицателен, если *str1 < str2;* равен нулю, если *str1 == str2* и положителен, если *str1 > str2* (сравнение беззнаковое).
strcpy	*char *strcpy (char *sp, char *si);* Копирует байты строки *si* в строку *sp*.
strspn	*char *strspn (char *str1, char *str2);* Определяет длину первого сегмента строки *str1*, содержащего символы, не входящие во множество символов строки *str2*.
strdup	*char *strdup (const char *str);* Выделяет память и переносит в нее копию строки *str*.
strlen	*unsigned strlen (char *str);* Вычисляет длину строки *str*.
strlwr	*char *strlwr (char *str);* Преобразует буквы верхнего регистра в строке в соответствующие буквы нижнего регистра.
strncat	*char *strncat (char *sp, char *si, int kol);* Приписывает *kol* символов строки *si* к строке *sp* (конкатенация).
strncmp	*int strncmp (char *str1, char *str2, int kol);* Сравнивает части строки *str1* и *str2*, причем рассматриваются первые *kol* символов. Результат отрицателен, если *str1 < str2*; равен нулю, если *str1 == str2* и положителен, если *str1 > str2* (сравнение беззнаковое).
strncpy	*char *strncpy (char *sp, char *si, int kol);* Копирует *kol* символов строки *si* в строку *sp* («хвост» отбрасывается или дополняется пробелами).

«Продолжение табл. 10.1»

Функция	Прототип и краткое описание действий
strnicmp	*char *strnicmp(char *str1, char *str2, int kol);* Сравнивает не более *kol* символов строки *str1* и строки *str2*, не делая различия регистров (см. функцию *strncmp*()).
strnset	*char *strnset(char *str, int c, int kol);* Заменяет первые *kol* символов строки *str* символом *c*.
strpbrk	*char *strpbrk(char *str1, char *str2);* Ищет в строке *str1* первое появление любого из множества символов, входящих в строку *str2*.
strrchr	*char *strrchr(char *str, int c);* Ищет в строке *str* последнее вхождение символа *c*.
strset	*int strset(char *str, int c);* Запоняет строку *str* заданным символом *c*.
strspn	*int strspn(char *str1, char str2);* Определяет длину первого сегмента строки *str1*, содержащего только символы, из множества символов строки *str2*.
strstr	*char *strstr(const char *str1, const char *str2);* Ищет в строке *str1* подстроки *str2*. Возвращает указатель на тот элемент в строке *str1*, с которого начинается подстрока *str2*.
strtod	*double *strtod(const char *str, char **endptr);* Преобразует символьную строку *str* в число двойной точности. Если *endptr* не равен *NULL*, то **endptr* возвращается как указатель на символ, при достижении которого прекращено чтение строки *str*.
strtok	*char *strtok(char *str1, const char *str2);* Ищет в строке *str1* лексемы, выделенные символами из второй строки *str2*.
strtol	*long *strtol(const char *str, char **endptr, int baz);* Преобразует символьную строку *str* к значению «длинное число» с основанием *baz*. Если *endptr* не равен *NULL*, то **endptr* возвращается как указатель на символ, при достижении которого прекращено чтение строки *str*.

«Продолжение табл. 10.1»

Функция	Прототип и краткое описание действий
strupr	*char *strupr(char *str);* Преобразует буквы нижнего регистра в строке *str* в буквы верхнего регистра.
ultoa	*char *ultoa(unsigned long v, char *str, int baz);* Преобразует беззнаковое длинное целое *v* в строку *str*.

Задача 10.1. Дан текст, состоящий из *N (2≤N≤10)* строк с максимальной длиной 80 символов. Необходимо вывести слова в алфавитном порядке, в которых больше *K* символов. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует.

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

const len = 80;
int get_word(char **&txt, int cnt_str, char **&wrd)
{
  int cnt_wrd=0;
  char *p;
  for (int i = 0; i < cnt_str; i++)
   {
    p=strtok(txt[i]," .,!?:;");
    while(p)
     {
              cnt_wrd++;
              if (cnt_wrd > 1)
               {
                wrd=(char**)realloc(wrd,cnt_wrd*sizeof(char*));
                wrd[cnt_wrd-1]=(char*)malloc((len+1)*sizeof(char));
               }
              strcpy(wrd[cnt_wrd-1],p);
              p=strtok(NULL," .,!?:;");
     }
   }
  return cnt_wrd++;
}

int main(int argc, char* argv[])
{
 char *text[len]={"Welcome to Borland Developer Studio 2006,",
 "an integrated development environment for building Delphi,",
```

```
        "Delphi for NET, C#, and C++ applications.",
        "Use this Help system to find conceptual, procedural, ",
        "and reference information about Borland Developer Studio 2006. ",
        "Where appropriate, this Help system provides links ",
        "to the Microsoft NET Framework SDK Help.",
        "The following Web pages offer additional assistance, information, and
resources:",
        "Borland Home Page; Borland Product Support;",
        "Borland Newsgroups;Borland Developer Network"};
        char **word, *tmp;
        int count_string, count_word, K,fl=0;
        cout<<"Enter count string: ";
        cin>>count_string;
        cout<<endl<<"***TEXT***"<<endl<<endl;
        for(int i=0; i < count_string; i++)
         cout<<text[i]<<endl;
        cout<<endl<<"Enter count char in word: ";
        cin>>K;
        word=(char**)malloc(sizeof(char*));
        word[0]=(char*)malloc((len+1)*sizeof(char));
        count_word=get_word(text,count_string,word);
         for(int i=0; i < count_word-1; i++)
          for(int j=i+1; j < count_word; j++)
            if(strcmpi(word[i],word[j])>0)
             {
                        tmp=word[i];
                        word[i]=word[j];
                        word[j]=tmp;
             }
        cout<<endl<<"***WORDS***"<<endl<<endl;
        for(int i=0; i < count_word; i++)
         if (strlen(word[i])>K)
          {
          cout<<word[i]<<endl;
          fl=1;
          }
        if (!fl)
          cout<<"Not word length > "<<K<<" chars!";
         cin.get();  cin.get();
          return 0;
       }
```

Результаты работы программы приведены на рис. 10.1.

Рис. 10.1

Задание для индивидуального выполнения
по теме «Работа со строками»:

Задание И10.1.

1. Дан текст, состоящий из *N* (*2≤N≤10*) строк с максимальной длиной 80 символов. Необходимо вывести в алфавитном порядке последние слова всех предложений. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки создать пользовательскую функцию.
2. Дан текст, состоящий из *N (2≤N≤10)* строк с максимальной длиной 80 символов. Необходимо вывести в обратном алфавитном порядке первые слова всех предложений. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки создать пользовательскую функцию. Для выделения слов из строки создать пользовательскую функцию.
3. Дан текст, состоящий из 2 строк с максимальной длиной 80 символов. Необходимо вывести в алфавитном порядке слова, присутствующие в обеих строках одновременно. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только

из букв, перенос слов по слогам отсутствует. Для выделения слов из строки создать пользовательскую функцию.

4. Дан текст, состоящий из 2 строк с максимальной длиной 80 символов. Необходимо вывести в обратном алфавитном порядке те слова, которые не присутствуют в обеих строках одновременно. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки создать пользовательскую функцию.
5. Дан текст, состоящий из 3 строк с максимальной длиной 80 символов. Необходимо вывести все предложения данного текста в порядке неубывания их длины. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения предложений из текста создать пользовательскую функцию.
6. Дан текст, состоящий из N $(2 \le N \le 10)$ строк с максимальной длиной 80 символов. Необходимо вывести в обратном алфавитном порядке те слова, где количество гласных превышает количество согласных. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки создать пользовательскую функцию.
7. Дан текст, состоящий из N $(2 \le N \le 10)$ строк с максимальной длиной 80 символов. Необходимо вывести в алфавитном порядке те слова, где количество согласных превышает количество гласных. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки создать пользовательскую функцию.
8. Дан текст, состоящий из N $(2 \le N \le 10)$ строк с максимальной длиной 80 символов. Необходимо вывести в алфавитном порядке те слова, длина которых превышает K символов. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки создать пользовательскую функцию.

9. Дан текст, состоящий из *N (2≤N≤10)* строк с максимальной длиной 80 символов. Необходимо вывести в обратном алфавитном порядке те слова, длина которых не превышает *K* символов. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки создать пользовательскую функцию.
10. Дан текст, состоящий из *N (2≤N≤10)* строк с максимальной длиной 80 символов. Необходимо вывести в обратном алфавитном порядке те слова, последняя буква которых является гласной. Считать, текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки создать пользовательскую функцию.
11. Дан текст, состоящий из *N (2≤N≤10)* строк с максимальной длиной 80 символов. Необходимо вывести в алфавитном порядке те слова, последняя буква которых является согласной. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки создать пользовательскую функцию.
12. Дан текст, состоящий из *N (2≤N≤10)* строк с максимальной длиной 80 символов. Необходимо вывести в алфавитном порядке те слова, первая буква которых является согласной, а последняя гласной. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки создать пользовательскую функцию.
13. Дан текст, состоящий из *N (2≤N≤10)* строк с максимальной длиной 80 символов. Необходимо вывести в алфавитном порядке предпоследние слова всех предложений. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует, минимальная длина предложений — два слова. Для выделения слов из строки создать пользовательскую функцию.
14. Дан текст, состоящий из *N (2≤N≤10)* строк с максимальной длиной 80 символов. Необходимо вывести в алфавитном по-

рядке вторые слова всех предложений. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует, минимальная длина предложений — два слова. Для выделения слов из строки создать пользовательскую функцию.

15. Дан текст, состоящий из 3-х строк с максимальной длиной 80 символов. Необходимо вывести в обратном алфавитном порядке те слова, которые присутствуют в 3-й строке и не присутствуют в первых двух. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки создать пользовательскую функцию.
16. Дан текст, состоящий из N $(2 \le N \le 10)$ строк с максимальной длиной 80 символов. Необходимо вывести в алфавитном порядке слова, начинающиеся с прописных букв. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки создать пользовательскую функцию.
17. Дан текст, состоящий из N $(2 \le N \le 10)$ строк с максимальной длиной 80 символов. Необходимо вывести в обратном алфавитном порядке слова, являющиеся палиндромами. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки и определения является ли они палиндромами, создать пользовательские функции.
18. Дан текст, состоящий из N $(2 \le N \le 10)$ строк с максимальной длиной 80 символов. Необходимо вывести все предложения данного текста, являющиеся палиндромами, в порядке невозрастания их длины. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения предложений из строки и определения является ли они палиндромами, создать пользовательские функции.
19. Дан текст, состоящий из N $(2 \le N \le 10)$ строк с максимальной длиной 80 символов. Необходимо вывести этот текст, зашифрованный кодом Цезаря. (Код Цезаря заменяет одну букву другой, отстоящей от нее на заданное количество позиций в

алфавите. Например, при сдвиге, равном 1, буква А заменяется на Б, Б — на В, …, Я — на А.) Размер сдвига символов принять равным номеру буквы в слове. Например, слово ДОМ — шифруется как ЕРП. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки и зашифровки слова создать пользовательские функции.

20. Составить программу для расшифровки текста, зашифрованного в пункте 19. Для выделения слов из строки и расшифровки слова создать пользовательские функции.
21. Дан текст, состоящий из N $(2 \le N \le 10)$ строк с максимальной длиной 80 символов. Необходимо вывести этот текст, зашифрованный кодом Цезаря. (Код Цезаря заменяет одну букву другой, отстоящей от нее на заданное количество позиций в алфавите. Например, при сдвиге, равном 1, буква А заменяется на Б, Б — на В, …, Я — на А.) Размер сдвига символов принять равным номеру буквы в предложении. Например, если предложение начинается со слова ДОМ, то оно шифруется как ЕРП. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения предложений из текста и зашифровки предложения создать пользовательские функции.
22. Составить программу для расшифровки текста, зашифрованного в пункте 21. Для выделения предложений из текста и расшифровки предложения создать пользовательские функции.
23. Дан текст, состоящий из N $(2 \le N \le 10)$ строк с максимальной длиной 80 символов. Необходимо вывести этот текст, зашифрованный кодом Цезаря. (Код Цезаря заменяет одну букву другой, отстоящей от нее на заданное количество позиций в алфавите. Например, при сдвиге, равном 1, буква А заменяется на Б, Б — на В, …, Я — на А.) Размер сдвига символов принять равным остатку от деления длины слова на номер буквы в слове плюс единица. Например, слово ДОМ — шифруется как ЕРН. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения слов из строки и зашифровки слова создать пользовательские функции.

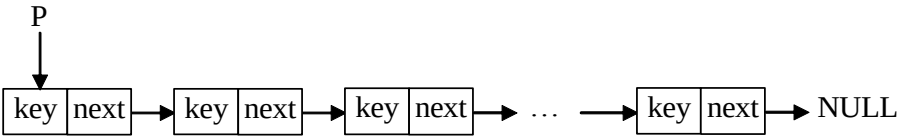
24. Составить программу для расшифровки текста, зашифрованного в пункте 23. Для выделения слов из строки и расшифровки слова создать пользовательские функции.
25. Дан текст, состоящий из *N (2≤N≤10)* строк с максимальной длиной 80 символов. Необходимо вывести этот текст, зашифрованный кодом Цезаря. (Код Цезаря заменяет одну букву другой, отстоящей от нее на заданное количество позиций в алфавите. Например, при сдвиге, равном 1, буква А заменяется на Б, Б — на В, …, Я — на А.) Размер сдвига символов принять равным остатку от деления количества слов в предложении на номер буквы в слове плюс единица. Например, если в предложении 8 слов, слово ДОМ — шифруется как ЕПР. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует. Для выделения предложений из текста и зашифровки предложения создать пользовательские функции.
26. Составить программу для расшифровки текста, зашифрованного в пункте 25. Для выделения предложений из текста и расшифровки предложения создать пользовательские функции.
27. Дан текст, состоящий из *N (2≤N≤10)* строк с максимальной длиной 80 символов. Необходимо вывести этот текст, зашифрованный кодом Цезаря. (Код Цезаря заменяет одну букву другой, отстоящей от нее на заданное количество позиций в алфавите. Например, при сдвиге, равном 1, буква А заменяется на Б, Б — на В, …, Я — на А.) Размер сдвига символов принять равным сумме сдвигов двух предыдущих букв. Для первой буквы принять сдвиг равным 0, а для второй — 1. Например, слово КОШКА — шифруется как КПЩМГ. Считать, что текст написан синтаксически грамотно, в качестве знаков препинания используются точка и запятая, слова состоят только из букв, перенос слов по слогам отсутствует.
28. Составить программу для расшифровки текста, зашифрованного в пункте 27. Для выделения слов из строки и расшифровки слова создать пользовательские функции.

ЛАБОРАТОРНАЯ РАБОТА №11.
ЛИНЕЙНЫЙ ОДНОСВЯЗНЫЙ СПИСОК

Если тип определен следующим образом

```
typedef struct List_Item *List;
struct List_Item
      { int  key;
        List next;
      };
```

то получается следующая структура данных



Такая структура называется линейным односвязным списком.

Определим некоторые операции с линейными списками.

Файл *list.h*

```
#ifndef __LIST_H
#define __LIST_H
typedef struct List_Item *List;
struct List_Item
      { int  key;
        List next;
      };
void PrintList(List);        //Печать списка
List InsertBegin(List, int);  //Вставка элемента в начало списка
List FindItem(List, int);     //Поиск элемента по ключу
void InsertAfter(List, int);  //Вставка элемента после заданного
void DeleteAfter(List);       //Удаление элемента после заданного
void FreeList(List);         //Освобождение памяти
#endif
```

Реализация (*list.cpp*)

```
#include "list.h"
#include "stdio.h"
#include "stdlib.h"
void PrintList(List p)
{
  while (p)
  {
    printf("%d ", p->key);
    p=p->next;
  }
```

```
  printf("\n");
};
List InsertBegin(List p, int x)
{
  //List q=(List)malloc(sizeof(List_Item));
  List q=new List_Item;
  q->key=x;
  q->next=p;
  return q;
};
List FindItem(List p, int x)
{
  while (p && p->key!=x)
    p=p->next;
  return p;
};
void InsertAfter(List p, int x)
{
  //List q=(List)malloc(sizeof(List_Item));
  List q=new List_Item;
  q->key=x;
  q->next=p->next;
  p->next=q;
};
void DeleteAfter(List p)
{
  List q=p->next;
  p->next=q->next;
  delete q; //free(q);
  };
void FreeList(List p)
{
  List q;
  while (p)
  {
    q=p;
    p=p->next;
    delete q; //free(q);
  }
};
```

Тестирование (*main.cpp*)

```
#include "list.h"
#include <stddef.h>
#include <stdio.h>
int main(int argc, char* argv[])
{
 List p=NULL;
```

```
  for (int i=1;i<=5;i++)
    p=InsertBegin(p,i);
  PrintList(p);
  List q=FindItem(p, 3);
  InsertAfter(q,10);
  PrintList(p);
  q=FindItem(p, 2);
  DeleteAfter(q);
  PrintList(p);
  FreeList(p);
  getchar();
  return 0;
}
```

Результат работы программы (рис. 11.1):

```
5  4  3  2  1
5  4  3  10  2  1
5  4  3  10  2

Enter key of delete element: 10

5  4  3  2
```

Рис. 11.1

Задание для индивидуального выполнения
по теме «Линейный односвязный список»

1. Набрать и отладить выше приведенную программную реализацию линейного односвязного списка.
2. Написать функцию удаления элемента по заданному ключу (предусмореть вариант существования в списке нескольких элементов с заданным ключом, включая первый элемент в списке).

ЛАБОРАТОРНАЯ РАБОТА №12. КЛАССЫ

Объектно-ориентированный подход к программированию. Предназначение понятия класса состоит в том, чтобы предоставить программисту инструмент для создания новых типов, столь же удобных в обращении сколь и встроенные типы. В идеале тип, определяемый пользователем, способом использования не должен отличаться от встроенных типов, только способом создания.

Рассмотрим реализацию понятия даты с использованием *struct* для того, чтобы определить представление даты *date* и множества функций для работы с переменными этого типа:

```
struct date { int month, day, year; };
// дата:    месяц, день, год  }
date today;
void set_date(date*, int, int, int);
void next_date(date*);
void print_date(date*);
```

Никакой явной связи между функциями и типом данных нет. Такую связь можно установить, описав функции как члены структуры:

```
struct date {
int month, day, year;
void set(int, int, int);
void get(int*, int*, int*);
void next();
void print();
};
```

Функции, описанные таким образом, называются функциями членами (методами) и могут вызываться только для специальной переменной соответствующего типа с использованием стандартного синтаксиса для доступа к членам структуры. Например:

```
date today;        // сегодня
date my_burthday;  // мой день рождения
my_burthday.set(30,12,1950);
today.set(18,1,1985);
my_burthday.print();
today.next();
```

Поскольку разные структуры могут иметь функции члены с одинаковыми именами, при определении функции члена необходимо указывать имя структуры:

```
void date::next()
{
  if ( day > 27 ) {
  // делает сложную часть работы
```

```
    }
    else day++;
}
```

В функции члене имена членов могут использоваться без явной ссылки на объект. В этом случае имя относится к члену того объекта, для которого функция была вызвана.

Классы Описание date в предыдущем подразделе дает множество функций для работы с date, но не указывает, что эти функции должны быть единственными для доступа к объектам типа *date*. Это ограничение можно наложить, используя вместо *struct class*:

```
class date {
    int month, day, year;
public:
    void set(int, int, int);
    void get(int*, int*, int*);
    void next();
    void print();
};
```

Метка *public* делит тело класса на две части. Имена в первой, закрытой части, могут использоваться только функциями членами. Вторая, открытая часть, составляет интерфейс к объекту класса. *struct* — это просто *class*, у которого все члены общие, поэтому функции члены определяются и используются точно так же, как в предыдущем случае. Например:

```
void date::print()
{
    printf("%d.%d.%d.", day, month, year);
}
```

Ссылки на себя. В функции члене на члены (поля) объекта, для которого она была вызвана, можно ссылаться непосредственно. Например:

```
class x {
    int m;
public:
    int readm() { return m; }
};
x aa;
x bb;
void f()
{
    int a = aa.readm();
    int b = bb.readm();
    // ...
```

```
}
```

В первом вызове *m* относится к *aa.m*, а во втором — к *bb.m*. Указатель на объект, для которого вызвана функция-член, является скрытым параметром функции. На этот неявный параметр можно ссылаться явно как на *this*. В каждой функции класса x указатель this неявно описан как

```
x* this;
```

и инициализирован так, что он указывает на объект, для которого была вызвана функция член. *this* не может быть описан явно, так как это ключевое слово. Класс *x* можно эквивалентным образом описать так:

```
class x {
   int m;
public:
   int readm() { return this->m; }
};
```

При ссылке на члены (поля) использование *this* излишне. Главным образом *this* используется при написании функций членов, которые манипулируют непосредственно указателями

Инициализация. Использование для обеспечения инициализации объекта класса функций вроде *set_date()* (установить дату) неэлегантно и чревато ошибками. Поскольку нигде не утверждается, что объект должен быть инициализирован, то программист может забыть это сделать, или (что приводит, как правило, к столь же разрушительным последствиям) сделать это дважды. Есть более хороший подход: дать возможность программисту описать функцию, явно предназначенную для инициализации объектов. Поскольку такая функция конструирует значения данного типа, она называется конструктором. Конструктор распознается по тому, что имеет то же имя, что и сам класс. Например:

```
class date {
   // ...
   date(int, int, int);
};
```

Когда класс имеет конструктор, все объекты этого класса будут инициализироваться. Если для конструктора нужны параметры, они должны даваться:

```
date today = date(23,6,1983);
date xmas(25,12,0);// (xmas - рождество)
date my_burthday;  // недопустимо, опущена инициализация
```

Часто бывает хорошо обеспечить несколько способов инициализации объекта класса. Это можно сделать, задав несколько конструкторов. Например:

```
class date {
   int month, day, year;
public:
   // ...
   date(int, int, int);    // день месяц год
   date(char*);            // дата в строковом представлении
   date(int);              // день, месяц и год сегодняшние
   date();                 // конструктор по умолчанию: сегодня
};
```

Если конструкторы существенно различаются по типам своих параметров, то компилятор при каждом использовании может выбрать правильный:

```
date today(4);
date july4("Июль 4, 1983");
date guy("5 Ноя");
date now;           // инициализируется по умолчанию
```

Размножение конструкторов в примере с *date* типично. При разработке класса всегда есть соблазн обеспечить «все», поскольку кажется проще обеспечить какое-нибудь средство просто на случай, что оно кому-то понадобится или потому, что оно изящно выглядит, чем решить, что же нужно на самом деле. Последнее требует больших размышлений, но обычно приводит к программам, которые меньше по размеру и более понятны. Один из способов сократить число родственных функций – использовать параметры по умолчанию. В случае date для каждого параметра можно задать значение по умолчанию, интерпретируемое как «по умолчанию принимать: *today*» (сегодня).

```
class date {
   int month, day, year;
public:
   // ...
   date(int d =0, int m =0, int y =0);
   date(char*);          // дата в строковом представлении
};
date::date(int d, int m, int y)
{
   day = d ? d : today.day;
   month = m ? m : today.month;
   year = y ? y : today.year;
   // проверка, что дата допустимая
   // ...
```

}

Когда используется значение параметра, указывающее «брать по умолчанию», выбранное значение должно лежать вне множества возможных значений параметра. Для дня *day* и месяца *mounth* ясно, что это так, но для года *year* выбор нуля неочевиден. В европейском календаре нет нулевого года. Сразу после 1 г. до н.э. (year==-1) идет 1 г. н.э. (year==1), но для реальной программы это может оказаться слишком тонко.

Объект класса без конструкторов можно инициализировать путем присваивания ему другого объекта этого класса. Это можно делать и тогда, когда конструкторы описаны. Например:

date d = today;// инициализация посредством присваивания

По существу, имеется конструктор по умолчанию, определенный как побитовая копия объекта того же класса. Если для класса X такой конструктор по умолчанию нежелателен, его можно переопределить конструктором с именем *X(X&)*

Копирующий конструктор. Инициализация объекта другим объектом того же класса называется почленной инициализацией по умолчанию. Копирование одного объекта в другой выполняется путем последовательного копирования каждого нестатического члена и осуществляется конструктором копирования. Вместе с конструктором умолчания, конструктор копирования входит в обязательный набор конструкторов для любого класса. Реализация механизма копирования значений для транслятора не является неразрешимой задачей. Конструктор копирования всего лишь создает копии объектов. Этот процесс реализуется при помощи стандартного программного кода. И построить такой код транслятор способен самостоятельно.

```
class x {
  //……
};
x a;
x b=a;
x c(a);
x d=x(a);
```

Проектировщик класса может изменить такое поведение, предоставив специальный копирующий конструктор. Если он определен, то вызывается всякий раз, когда один объект инициализируется другим объектом того же класса.

Часто почленная инициализация не обеспечивает корректного поведения класса. Поэтому мы явно определяем копирующий конструктор. Копирующий конструктор принимает в качестве фор-

мального параметра ссылку на объект класса (традиционно объявляемую со спецификатором *const*).

```
class x {
  //……
public:
  x(const x &t);
  //……
};
```

Очистка. Определяемый пользователем тип чаще имеет, чем не имеет, конструктор, который обеспечивает надлежащую инициализацию. Для многих типов также требуется обратное действие, деструктор, чтобы обеспечить соответствующую очистку объектов этого типа. Имя деструктора для класса *X* есть *~X()* («дополнение конструктора»). В частности, многие типы используют некоторый объем динамической памяти, который выделяется конструктором и освобождается деструктором. Вот, например, традиционный стековый тип, из которого для краткости полностью выброшена обработка ошибок:

```
class char_stack {
   int size;
   char* top;
   char* s;
public:
   char_stack(int sz) { top=s=new char[size=sz]; }
   ~char_stack()      { delete []s; }    // деструктор
   void push(char c)  { *top++ = c; }
   char pop()         { return *--top;}
}
```

Когда *char_stack* выходит из области видимости, вызывается деструктор:

```
void f()
{
   char_stack s1(100);
   char_stack s2(200);
   s1.push('a');
   s2.push(s1.pop());
   char ch = s2.pop();
   printf("%c\n", ch;
}
```

Когда вызывается *f(),* конструктор *char_stack* вызывается для *s1*, чтобы выделить вектор из 100 символов, и для *s2*, чтобы выделить вектор из 200 символов. При возврате из *f()* эти два вектора будут освобождены.

Законченный класс. Программирование без скрытия данных (с применением структур) требует меньшей продуманности, чем программирование со скрытием данных (с использованием классов). Структуру можно определить, не слишком задумываясь о том, как ее предполагается использовать. А когда определяется класс, все внимание сосредотачивается на обеспечении нового типа полным множеством операций; это важное смещение акцента. Время, потраченное на разработку нового типа, обычно многократно окупается при разработке и тестировании программы.

Вот пример законченного типа *PascalSet,* который реализует понятие «битовое множество целых однобайтовых беззнаковых»:

Файл *pascalset.h*

```
#ifndef _PS_H
#define _PS_H
#include <mem.h>
class PasSet {
  unsigned char data[32];
  unsigned char GetMask(unsigned char x)
    {return 1<<x;}
  unsigned char GetNumByte(unsigned char x)
    {return x/8;}
  unsigned char GetNumBit(unsigned char x)
    {return x%8;}
public:
  //Конструктор
  PasSet()
    {memset(data, 0, sizeof(data));}
  //Проверка вхождения
  int InSet(unsigned char x);
  //Включение элемента
  void include(unsigned char x);
  //Исключение элемента
  void exclude(unsigned char x);
  //Вывод
  void print();
};
#endif
```

Файл *pascalset.cpp*

```
#include <stdio.h>
#include "pascalset.h"
int PasSet::InSet(unsigned char x)
{
  unsigned char nbyte=GetNumByte(x);
  unsigned char nbit=GetNumBit(x);
  unsigned char mask=GetMask(nbit);
```

```
  return data[nbyte] & mask;
}
void PasSet::include(unsigned char x)
{
  unsigned char nbyte=GetNumByte(x);
  unsigned char nbit=GetNumBit(x);
  unsigned char mask=GetMask(nbit);
  data[nbyte]|=mask;
}
void PasSet::exclude(unsigned char x)
{
  unsigned char nbyte=GetNumByte(x);
  unsigned char nbit=GetNumBit(x);
  unsigned char mask=GetMask(nbit);
  if (InSet(x)) data[nbyte]^=mask;
}
void PasSet::print()
{
  for (int i=0; i<256; i++)
    if (InSet((unsigned char)i)) printf("%d ", i);
  printf("\n");
}
```

Файл mainset.cpp

```
#include <stdio.h>
#include "pascalset.h"
int main(int argc, char* argv[])
{
  PasSet s;
  s.include(8);s.include(3);s.include(5);
  s.print();
  s.exclude(3);
  s.print();
  if (s.InSet(5)) printf("5 is in set\n");
  else printf("5 is not in set\n");
  getchar();
  return 0;
}
```

Результат работы программы

```
3  5  8
5  8
5 is in set
```

Задание для индивидуального выполнения
по теме «Классы»:

Задание И12.1.

1. Создать класс для хранения календарных дат. Обеспечить возможность работы с датами в различных форматах, изменения даты на заданное количество дней. Перегрузить операцию *«–»* для нахождения разности дат и операции сравнения. Для класса определить оператор <<. Стандартные функции и типы С для работы с датами не использовать.
2. Создать класс для хранения одномерных целочисленных массивов. Обеспечить возможность задания количества элементов и базовой индексации. Запрограммировать методы поиска элементов и сортировки. Перегрузить операции для сложения и вычитания векторов. Перегрузить операцию индексирования (т.к. оператор взятия индекса может появляться как слева, так и справа от оператора присваивания, то функция должна возвращать *int&*) с проверкой допустимости индекса.
3. Создать класс для хранения строк. Запрограммировать методы поиска подстроки, копирования, замены и удаления заданной подстроки, определения длины строки. Перегрузить операцию *«+»* для конкатенации строк, операцию присваивания и операцию индексирования (т.к. оператор взятия индекса может появляться как слева, так и справа от оператора присваивания, то функция должна возвращать *char&*) с проверкой допустимости индекса.
4. Создать класс для хранения обыкновенных дробей. Запрограммировать метод сокращения дроби. Перегрузить арифметические операции. Для класса определить оператор <<. Предусмотреть возбуждение исключительных ситуаций (при делении на ноль, переполнении).
5. Создать класс, для хранения n-разрядных (n<100) целых чисел (длинная арифметика). Перегрузить арифметические операции над числами (сложение, вычитание, умножение, деление, целочисленное деление). Для класса определить оператор <<.
6. Создать класс, содержащий информацию о почтовом адресе организации. Предусмотреть возможность раздельного изменения составных частей адреса, создания и уничтожения объектов этого класса.
7. Создать класс для представления комплексных чисел. Обеспечить выполнение операций сложения, вычитания и умножения комплексных чисел.

8. Создать класс для объектов-векторов, задаваемых координатами концов в трехмерном пространстве. Обеспечить операции сложения и вычитания векторов с получением нового вектора (суммы или разности), вычисления скалярного произведения двух векторов, длины вектора, косинуса угла между векторами.
9. Создать класс прямоугольников со сторонами, параллельными осям координат. Предусмотреть возможность перемещения прямоугольников на плоскости, изменение размеров, построение наименьшего прямоугольника, содержащего два заданных прямоугольника, являющегося общей частью (пересечением) двух прямоугольников.
10. Создать класс многочленов от одной переменной, задаваемых степенью многочлена и массивом коэффициентов. Предусмотреть методы для вычисления значения многочлена для заданного аргумента, операции сложения, вычитания и умножения многочленов с получением нового объекта-многочлена, вывод на экран описания многочлена.
11. Создать класс, обеспечивающий представление матрицы произвольного размера с возможностью изменения числа строк и столбцов, вывода на экран подматрицы любого размера и всей матрицы.

ЛАБОРАТОРНАЯ РАБОТА №13.
ПЕРЕОПРЕДЕЛЕНИЕ ВВОДА-ВЫВОДА НА ЯЗЫКЕ С++

С++ предоставляет группу классов и функций для ввода-вывода, определяемые в библиотеке iostreams. Чтобы получить к ним доступ, программа должна иметь директиву *#include <iostreams.h>.*

Стандартный ввод/вывод. С++ предоставляет четыре предварительно определенных потоковых объекта:

- *cin* стандартный ввод;
- *cout* стандартный вывод;
- *cerr* стандартный вывод ошибок;
- *clog* полностью буферизованная версия cerr;

Есть возможность перенаправить эти стандартные потоки из/на другие устройства и файлы.

Оператор поразрядного сдвига влево <<, применительно к операциям с потоками, называется оператором вставки или оператором «поместить в», а оператор поразрядного сдвига вправо >> – называется оператором извлечения или оператором «прочитать из».

Класс *istream* включает перекрываемые определения для оператора >>, используемого со стандартными типами *[int, long, double, float, char*(строка)]*. Таким образом, предложение cin >> x; вызывает соответствующую функцию оператора >> для istream cin, определенного в *iostream.h* и использует ее для направления этого входного потока в позицию памяти, представляемую переменной *x*. Аналогично, класс *ostream* имеет перекрываемые определения для оператора <<, который разрешает с помощью предложения *cout << x*; посылать значение x в *ostream cout* для вывода.

Данная программа просто копирует *cin* в *cout*. При отсутствии изменения направления она копирует ввод с клавиатуры на экран:

```
// Выводит на экран ввод с клавиатуры
#include <iostream.h>
int main()
{
  char ch;
  while (cin >> ch)
    cout << ch;
}
```

Примечание. Эта программа просто сохраняет каждый вводимый символ в переменной *ch*, а затем выводит значение *ch* на экран.

Форматируемый вывод. Простой ввод/вывод в C++ эффективен потому, что выполняет только минимальные преобразования в соответствии с используемыми типами данных. Для целых чисел преобразование такое же, как определенное по умолчанию преобразование для *printf*. Предложения

```
int i=5; cout << i;
```

и

```
int i=5; printf("%d", i);
```

дают одинаковый результат.

Форматирование позволяет определить для каждого активного потока систему исчисления для преобразования (десятичная, восьмеричная и шестнадцатеричная), заполнение слева или справа, формат с плавающей запятой (научный или фиксированный) и будут ли пропускаться при вводе пробелы. Другие параметры, которые можно изменять, включают ширину поля (для вывода) и символ, используемый для заполнения. Эти флаги можно проверять, устанавливать, очищать с помощью различных функций элементов.

```
int i = 87;
cout.width(7); // устанавливает ширину поля, равной 7
cout.fill('*'); // устанавливает символ * для заполнения
cout << i << '\n'; // выводит на экран *****87
//после << ширина очищается и становится равной 0,
cout << i << '\n'; // выводит на экран 87
double d=3.14159;
cout.precision(4);
cout.fill('*'); // устанавливает символ * для заполнения
cout.width(8);
cout << d << '\n'; // выводит на экран ***3.142
```

Установление *width*, равной *0* (по умолчанию) означает, что дисплей будет использовать столько позиций экрана, сколько необходимо. Если заданная ширина недостаточна для правильного представления, то считается, что ширина равна *0* (т.е. усечения нет). По умолчанию заполнение обеспечивает выравнивание справа (заполнение слева) для всех типов.

Манипуляторы. Наиболее изящный способ установки флагов формата (и выполнения других функций для потока) использует специальный механизм, известный как манипуляторы. Подобно операторам << и >> манипуляторы могут включаться в цепочку операторов потока:

```
cout << setw(7) << dec << i << setw(6) << oct << j;
cout << setw(7) << setprecision(4) << endl; // вывод d из
                                            // пред. примера
```

Без манипуляторов понадобилось бы шесть отдельных предложений.

Параметризованный манипулятор *setw* принимает один аргумент типа *int*, чтобы установить ширину поля.

Непараметризованные манипуляторы, например, *dec*, *oct* и *hex* устанавливают систему исчисления для преобразования: десятичную, восьмеричную и шестнадцатеричную. В вышеприведенном примере *int i* будет выводится на экран в десятичном виде в поле шириной 7; *int j* будет выводится на экран в восьмеричном виде в поле шириной 6.

К другим простым параметризованным манипуляторам относится *setbase*, *setfill*, *setprecision*, *setiosflags* и *resetiosflags*. Чтобы использовать любые из параметризованных манипуляторов, программа должна включить оба заголовочных файла: *iomanip.h u iostream.h*. Непараметризованные манипуляторы не требуют *iomanip.h.*

Полезные непараметризованные манипуляторы включают:

– *ws* (извлечение пробелов): *istream >> ws*; будет устранять любые пробелы в *istream*.

– *endl* (конец строки и очистка буфера): *ostream << endl*; будет вставлять новую строку в ostream, а затем очищает буфер *ostream*.

– *ends* (конец строки с нулем): *ostream << ends*; будет добавлять 0 в *ostream*.

– flush (очистка буфера): ostream << flush.

Ввод/вывод с диска. Библиотека *istream* включает много классов, порожденных из *streambuf*, *ostream* и *istream*, предоставляя, таким образом, широкий выбор методов ввода/вывода файлов. Класс *filebuf* например, поддерживает ввод/вывод через дескрипторы файлов с помощью функций элементов для открытия, закрытия и поиска.

Наиболее часто используемыми классами являются ifstream (порожденный из *istream*), *ofstream* (порожденный из ostream) и *fstream* (порожденный из *iostream*). Все они поддерживают форматированный файл ввода/вывода, используя объекты *filebuf*. Ниже приведен простой пример программы для ввода/вывода с диска массива структур.

```
#include <iostream.h>
#include <fstream.h>
int main(int argc, char* argv[])
{
  struct emp
  {
    char* name;
```

```
  int dept;
};
emp e[2];
e[0].name="aaaa";
e[0].dept=1;
e[1].name="bbbbbbb";
e[1].dept=2;
ifstream source;
ofstream dest;
char filename[20]="s.dat";
dest.open(filename);
for(int i=0;i<2;i++)
   dest<<e[i].name<<" "<<e[i].dept<<endl;
dest.close();
source.open(filename,ios::nocreate); // файл должен быть создан
for(int i=0;i<2;i++)
{
   source>>e[i].name>>e[i].dept;
   cout<<e[i].name<<" "<<e[i].dept<<endl;
}
source.close();
cin.get();
return 0;
}
```

При этом файл *s.dat* содержит следующие данные:

aaaa 1
bbbbbbb 2

Если поле *name* может содержать пробелы, то такой способ ввода/вывода непригоден. Нужно использовать разделитель полей, отличный от пробела, например, @.

```
for(int i=0;i<2;i++)
    dest<<e[i].name<<"@"<<e[i].dept<<endl;
```

Теперь чтение данных будет выглядеть следующим образом:

```
char buf[22];
for(int i=0;i<2;i++)
{
   source.get(e[i].name, 100, '@');
   source.get();                  //отбросить @
   source.get(buf, 100, '\n');
   source.get();                  //отбросить \n
   e[i].dept=atoi(buf);
   cout<<e[i].name<<" "<<e[i].dept<<endl;
}
```

Ввод/вывод для типов данных, определенных пользователем

Реальное преимущество потоков C++ заключается в легкости, с которой можно совместить >> и << для управления вводом/выводом пользовательских типов данных. Рассмотрим простую структуру данных, которую можно объявить:

```
struct emp
{
  char *name;
  int dept;
  long sales;
};
```

Чтобы совместить << для вывода объектов типа *emp*, необходимо следующее определение:

```
ostream& operator << (ostream& str, emp& e)
{
  str << setw(25) << e.name << ": Отдел " << setw(6) << e.dept << tab
<<
      "З.П. " << e.sales << " р." << '\n';
  return str;
}
```

Функция-оператор << должна возвращать *ostream&,* ссылку на ostream, для того, чтобы можно было продолжать цепочку вывода. Теперь можно вывести объекты типа emp следующим образом:

```
#include <iostream.h>
#include <iomanip.h>
...
emp jones = {"Иванов", 25, 1000};
cout << jones;
получив вывод на дисплей
Иванов: Отдел 25 З.П. 1000 р.
Манипулятор tab в определении << определен пользователем:
ostream& tab(ostream& str)
{
  return str << '\t';
}
```

Пример: Переопределить операции << и >> для ввода-вывода объектов класса time.

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

```
class TMyTime
{
int hour, minute, second;
   public:
     TMyTime(int, int, int);
     friend ostream& operator << (ostream&, TMyTime&);
     friend istream& operator >> (istream&, TMyTime&);
};
// Конструктор
TMyTime(int h = 0,  int m = 0, int s = 0)
     {
       hour = h;
       minute = m;
       second = s;
     }
// Переопределение операции ввод >>
istream& operator >> (istream& s, TMyTime& t)
{
  int h_n = 0, m_n = 0, s_n = 0;
  char ch=0;
  s >> h_n >> ch >> m_n >> ch >> s_n;
  if (s)
   {
     if (h_n<0||h_n>23) { cout<<"Error hour (default hour = 0)"<<endl;
t.hour=0;}
     else  t.hour = h_n;
     if (m_n<0||m_n>59) { cout<<"Error minute (default minute =
0)"<<endl; t.minute=0;}
     else  t.minute = m_n;
     if (s_n<0||s_n>59) { cout<<"Error second (default second = 0)"<<endl;
t.second=0;}
     else  t.second = s_n;
   }
 return s;
}
// Переопределение операции вывод <<
ostream& operator << (ostream& s, TMyTime& t)
{
 s<<endl<<"The time enter user: "<<endl;
 s<<"\t1.       (hh(24):mm:ss):       "<<setw(2)<<t.hour<<":"<<setw(2)<<
t.minute<<":"<<setw(2)<<t.second<<endl;
 s<<"\t2. (hh(24) hour mm min ss sec): "<<setw(2)<<t.hour<<" hour
"<<setw(2)<<t.minute<<" min "<<setw(2)<<t.second<<" sec "<<endl;
t.hour%=12;
 s<<"\t3. (hh(12):mm:ss): "<<setw(2)<<t.hour<<":"<<setw(2)<<
t.minute<<":"<<setw(2)<<t.second<<endl;
 s<<"\t4. (hh:mm:ss am/pm): ";
```

```
 if (t.hour<12)
s<<setw(2)<<t.hour<<":"<<setw(2)<<t.minute<<":"<<setw(2)<<t.second<
<" pm"<<endl;
 else
  if (t.hour==12&&!t.minute&&!t.second)
s<<setw(2)<<t.hour<<":"<<setw(2)<<t.minute<<":"<<setw(2)<<t.second<
<" am"<<endl;
  else
   s<<setw(2)<<t.hour-12<<":"<<setw(2)<<t.minute<<":"<<
setw(2)<<t.second<<" am"<<endl;
 return s;
}
unit_TMyTime.cpp
#include <iostream.h>
#include"unit_TMyTime.h"
int main(int argc, char* argv[])
{
  TMyTime mytime;
  cout<<"Enter time (hh,mm,ss): ";
  cin >> mytime;
  cout.fill('0');
  cout<< mytime;
  getchar();getchar();
  return 0;
}
```

Результаты работы программы:

```
Enter time (hh,mm,ss): 21,35,59

The time enter user:
        1. (hh(24):mm:ss): 21:35:59
        2. (hh(24) hour mm min ss sec): 21 hour 35 min 59 sec
        3. (hh(12):mm:ss): 09:35:59
        4. (hh:mm:ss am/pm): 09:35:59 pm
```

Задание для индивидуального выполнения
по теме «Переопределение ввода-вывода на языке С++»:

1. Переопределить операции << и >> для ввода-вывода матриц размерностью *m*n*, где *m* и *n* константы.

2. Переопределить операции << и >> для ввода-вывода объектов класса *complex*.

3. Переопределить операции << и >> для ввода-вывода объектов класса *data*.

4. Переопределить операции << и >> для ввода-вывода объектов класса *time*.

5. Структура «абонент телефонной сети» содержит следующие поля: 1) фамилия (строка), 2) улица, 3) дом, 4) квартира, 5)

тел. номер. Переопределить операции << и >> для файлового ввода-вывода такого типа данных.

6. Структура «карточка библиотечного каталога» содержит следующие поля: 1) массив фамилий авторов (до 3 эл-в), 2) название произведения, 3) УДК, 4) год издания. Переопределить операции << и >> для файлового ввода-вывода такого типа данных.

7. Переопределив операции << и >>, разработать механизм сохранения в файле структур, где первое поле — размер массива, второе — указатель на динамический массив.

8. Объект «колода карт» представляет собой массив структур, где первое поле — масть, второе — значимость. Используя операции << и >>, разработать механизм сохранения состояния колоды в файле и восстановления из файла.

9. Переопределить операции << и >> для файлового ввода-вывода объектов типа «классный журнал по заданному предмету». В объекте хранятся оценки учеников и темы занятий с учетом даты.

10. Переопределить операции << и >> для файлового ввода-вывода объектов типа «склад», где хранятся объекты типа «товар», для которых известно:

1) артикул, 2) количество, 3) цена.

11. Переопределить операции << и >> для файлового ввода-вывода объектов типа «тест», где хранятся вопросы и результаты ответов (типа да-нет) группы респондентов.

12. Переопределить операции << и >> для файлового ввода-вывода объектов типа «курс валют», которые содержат данные о курсах с начала календарного года.

13. Переопределить операции << и >> для файлового ввода-вывода объектов типа «курс акций», где хранятся данные о курсах с начала календарного года.

14. Переопределить операции << и >> для файлового ввода-вывода объектов типа «расписание вылетов самолетов за неделю».

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Время жизни и область видимости программных объектов. Классы памяти. Инициализация глобальных и локальных переменных
2. Динамические объекты. Способы выделения и освобождения памяти. Линейный односвязный список.
3. Динамические массивы. Особенности выделения и освобождения памяти для многомерных массивов.
4. Директивы препроцессора. Макроопределения.
5. Объектно-ориентированный подход к программированию. Классы.
6. Объектно-ориентированный подход к программированию. Инициализация и разрушение объектов. Конструкторы и деструкторы.
7. Объектно-ориентированный подход к программированию. Ограничения доступа к членам класса. Друзья класса.
8. Объектно-ориентированный подход к программированию. Наследование.
9. Перегрузка операций.
10. Организация ввода-вывода на языке C++. Потоки ввода-вывода.
11. Шаблоны функций.
12. Шаблоны классов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Серебренникова И.Г., Воронина О.Б. Структурный тип данных: Методические указания и варианты индивидуальных заданий по дисциплине «Программирование на языке высокого уровня» для студентов всех форм обучения специальности 220400. Магнитогорск: МГТУ, 2003. 21 с.
2. Экономическая информатика. Учебник для Вузов. под. ред. д. э. н., проф. В.В. Евдокимова. С. – П., 1997. 592 с.
3. Гричишин Я.Т. Алгоритмы и программы на Бейсике. – М.: Просвещение, 1988.
4. Чернов Б.И. Программирование на алгоритмических языках Бейсик, Фортран, Паскаль. – М.: Просвещение, 1991.
5. Поддубная Л.М. Мне нравится Паскаль. – М.: Радио и связь, 1992.
6. Бабушкина В.Е. и др. Практикум по программирования на Turbo Pascal. –М.: Бином, 1998.
7. Логунова О.С., Ильина Е.А., Кирпичева Н.Т., Вяльцина Т.Н., Тутарова В.Д. Программирование на языке VISUAL BASIC FOR APPLICATION: Учебное пособие. – Магнитогорск: МГТУ, 2000.
8. Юркин А. Задачник по программированию. – СПб.: Питер, 2002. – 192 с.
9. Перельман Я.И. Живая математика. – М.: Наука, 1978. – 160 с.
10. Павловская Т.А., Щупак Ю.А. С/С++. Структурное программирование: Практикум. – СПб.: Питер, 2005. – 239 с.
11. Есипов А.С., Паньгина Н.Н., Громада М.И. Информатика. Сборник задач и решений для общеобразовательных учебных заведений. – СПб.: Наука и Техника, 2001. – 368 с.
12. Филиппова Н.В. и др. Практикум по программированию: учебное пособие. – Магнитогорск: МГТУ им. Г.И.Носова, 2000. 84 с.
13. Алексеев Е.Р., Чеснокова О.В. Турбо Паскаль 7.0 – М.: НТ Пресс, 2006. – 320 с.

Разные задачи

1. По известному радиусу определить длину окружности, площадь круга, площадь поверхности сферы и объем шара.

2. Дано натуральное K — количество секунд. Определить сколько это составляет часов, минут и секунд. Например, 4000 секунд — это 1 час, 6 минут и 40 секунд.

3. Определить сколько лет понадобится России, чтобы собрать урожай зерна, требуемый изобретателем шахмат. Считать, что среднегодовой сбор составляет 70 млн. тонн, а на 1 грамм приходится 10 зерен.

4. Вычислить значение выражения:

$$y = \sqrt{\left| \frac{a^{\sin^2 b + \cos b^3} + \sqrt[3]{b^2}}{\sqrt[4]{\left| \frac{a \cdot tgb}{1 - e^{\sqrt{a}}} \right|}} \right|}$$

Контрольный пример: при *a=b=0.5 y=1.214*

5. Дано время — два целых числа количество часов и минут. Необходимо определить меньший угол между часовой и минутной стрелками на циферблате часов.

Тесты для проверки

02:43	176.5
11:40	110.0
14:30	105.0
23:40	110.0
00:00	0.0
12:00	0.0
23:28	176.0
10:02	71.0
19:03	166.5
16:40	100.0

12:18	99.0
05:47	108.5
18:42	51.0
15:29	69.5
16:44	122.0
21:10	145.0
15:48	174.0
22:59	24.5
11:34	143.0
06:52	106.0
13:30	135.0

6. Волшебный мост

Крестьянин, возвращаясь с ярмарки, увидел на мосту странную картину. Какой-то человек сначала считал деньги в кошельке, затем бросал в реку несколько монеток, бежал на другой конец моста, снова считал деньги в кошельке, и опять бросал несколько монеток и шел на другой конец моста. Наконец, пересчитав свои деньги, он явно обрадовался и отправился в дальнейший путь.

– Что ты делал? Зачем ты бросал деньги в воду? – спросил крестьянин, догнав странного человека.

Видя, что свой секрет скрыть не удастся, человек рассказал, что мост волшебный, что, если бросить с моста ровно 29 копеек, то, как только перейдешь мост, количество рублей в оставшейся сумме денег превращаются в новой сумме в количество копеек, а копейки – в рубли, что, перейдя мост несколько раз, можно получить сумму, намного большую первоначальной.

– Самое важное – вовремя остановиться, – сказал человек и ушёл.

Крестьянин задумался, достал кошелек и пересчитал свои деньги. У него было 46 рублей 47 копеек. «29 копеек – не деньги, дай-ка попробую». После первого прохода у него получилось 18р.46к., после второго прохода – 17р.18к., а после третьего – 89р.16к. «Ух-ты! А еще больше можно получить?» – обрадовался крестьянин. После четвертого прохода у него стало 87р.88к., после пятого – 59р.87к., после шестого – 58р.59к., после седьмого – 30р.58к., после восьмого – 29р.30к., после девятого – 1р.29к., а после десятого осталась 1 копейка.

«Эх, дурачина, надо было после третьего раза остановиться!» – расстроился крестьянин.

Напишите программу, которая по начальной сумме денег у крестьянина определит оптимальное число проходов по мосту для получения наибольшей конечной суммы.

Контрольные примеры:

7699–9904–81	1136–9902–63	3563–8916–10	6287–6287–0
7069–9998–196	3599–9964–197	7038–9967–196	

7. Разложение

Альберт хочет представить некоторое целое положительное число N в виде сумме квадратов двух целых положительных чисел P и Q (0<P<=Q). Это не всегда возможно. Если точного разложения не существует, Альберту нужно подобрать такие P и Q, чтобы значение выражения $|N–P^2–Q^2|$ было минимальным. Если существует несколько вариантов разложения, минимизирующих значение указанного выражения, то вывести вариант с меньшим Q.

Напишите программу, которая вводит с клавиатуры целое число N (1<=N<=106) и выводит на экран целые значения P и Q.

Тесты для проверки

N	25	2581	9	9999	888888
P·Q	3 4	30 41	2 2	60 80	534 777