ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение высшего профессионального образования

УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Т. Е. Родионова

КУРС ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ СИ

учебное пособие
для студентов специальности 23040165

Ульяновск
2006

УДК 681.3.06 (075)
ББК 22.18я7
Р60

Рецензенты: Заведующий кафедрой «Механика и теория управления» УлГУ, доктор физико-математических наук, профессор А. С. Андреев
доктор технических наук, профессор кафедры «Информационные технологии» УлГУ И. В. Семушин

Утверждено редакционно-издательским Советом университета в качестве учебного пособия

Родионова, Т. Е.
Р60 Курс программирования на языке Си: учебное пособие для студентов специальности 23040165 / Т. Е .Родионова. — Ульяновск: УлГТУ, 2006. — 118 с.

ISBN 5-89146-900-0
ISBN 978-5-89146-900-0

Пособие предназначено для подготовки студентов специальности 23040165, по дисциплине «Алгоритмические языки и программирование». Составлено в соответствии с учебным планом специальности. Разработано на кафедре прикладной математики и информатики. В пособии рассматриваются принципы создания программ на языке Си. Изложены принципы процедурного и объектно-ориентированного программирования. Приводятся основные операторы С и С++, описание основных и структурных типов данных, примеры программ. Достоинством является наличие большого количества примеров для иллюстрации излагаемого материала и наличие заданий для выполнения лабораторных работ по основным разделам изучаемого курса.

Предназначено для студентов вузов дневной формы обучения.

УДК 681.3.06 (075)
ББК 22.18я7

ISBN 5-89146-900-0

ISBN 978-5-89146-900-0

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Рассмотрим структурный подход в программировании, который уже давно широко применяется программистами и который является важной частью объектно-ориентированного программирования. Примеры понятий, действий и структур будем рассматривать на языке программирования Си.

Структуру программ можно представить в виде различных схем (например, блок-схемы). Сложные программы имеют сложную и запутанную структуру, в которой сам автор может долго «блуждать». Поэтому стали развиваться технологии программирования, которые позволяют улучшить читаемость исходных текстов программы.

На заре вычислительной техники, когда в распоряжении пользователей были ограниченные ресурсы ЭВМ, а разработчик программ был и главным ее пользователем, главное внимание обращалось на получение эффективных программ в смысле оптимального использования ресурсов ЭВМ.

Теперь сфера применения ЭВМ чрезвычайно расширилась, разработка и эксплуатация осуществляется разными людьми. Наряду с эффективностью на первый план выдвигаются такие характеристики программ, как: понятность, хорошая документированность, надежность, гибкость, удобство сопровождения и т. д. С этим связана трудоемкость процесса программирования и быстрый рост стоимости программного обеспечения.

Появилась необходимость придерживаться определенных принципов или дисциплины программирования. Появление новой технологии программирования, основанной на структурном подходе, связано с именем известного голландского ученого Э. Дейкстры. «Если отладка – процесс удаления ошибок, то программирование должно быть процессом их внесения». В своих работах он высказал предположение, что квалификация программиста обратно пропорциональна числу операторов безусловного перехода в его программах. Структурный подход имеет целью снижение трудоемкости процесса создания программного обеспечения. Язык Си изначально предполагает использование структурного программирования.

1. ОСНОВНЫЕ КОНСТРУКЦИИ ЯЗЫКА ПРОГРАММИРОВАНИЯ СИ

Язык программирования Си был разработан и реализован в 1972 году Денисом Ритчи. Популярность языка объясняется тем, что это лаконичный язык, сочетающий возможность использования машинно-ориентированных средств с одной стороны и создавать мобильные программы с другой стороны. Язык Си является гибким и эффективным языком программирования.

В настоящее время существует достаточно много реализаций языка Си в различных средах программирования.

1.1 Основные элементы языка

Алфавит языка Си это практически все символы, имеющиеся на стандартной клавиатуре компьютера. Некоторые операции обозначаются

комбинациями символов; значение символов операций в ряде случаев зависит от контекста, в котором они употреблены.

Лексема – это единица текста программы, которая имеет определенный смысл для компилятора и которая не может быть разбита в дальнейшем. Разбор на лексемы происходит в порядке следования символов в программе. За очередную лексему принимается максимальный ряд знаков, который могут образовывать лексемы.

Лексемами являются:

1. знаки пунктуации [ ] { } < > , ; ( );
2. идентификаторы;
3. знаки операций;
4. константы;
5. ключевые слова.

Чтобы избежать неоднозначности и обеспечить правильное вычисление выражений, рекомендуется использовать пробелы и круглые скобки для разделения лексем.

Например, разбирая выражение a - - - b; компилятор трактует его как (a--)-b; а не как a-(--b);

Комментарии – это последовательность символов, которые компилятор воспринимает как отдельный пробельный символ и игнорируется. Комментарии на языке Си могут выделяться следующими способами:

1) // это комментарий от указанных наклонных символов до конца текущей строки;

2) /*… это комментарий на несколько строк программы, все, что заключено между начальным и конечным символами комментария игнорируется транслятором

… */

Комментарий можно вставить в любое место, где можно вставить пробел.

Пример: int /* это счетчик */ count;

Идентификаторы (имена переменных, функций, типов и меток, используемых в программе) – это последовательность букв и цифр, начинающаяся с буквы. Возможно использование знака подчеркивания. Компилятор языка Си различает строчные и прописные буквы (например, Temp, temp, Temp).

Ключевые слова – это предопределенные идентификаторы, которые имеют специальное значение для компилятора языка. Их нельзя использовать для имен переменных, функций и т. д.

Константы – это явное представление значения. В языке Си определены 4 типа констант:

1) целые;
2) с плавающей точкой;
3) символьные;
4) строки.

Целая константа – это десятичное, восьмеричное или шестнадцатеричное число, может быть положительным или отрицательным. Десятичная константа – это последовательность цифр от 0 до 9, и она не должна начинаться с 0. Восьмеричная константа – это последовательность цифр от 0 до 7, начинающаяся с 0. Шестнадцатеричная константа – это последовательность цифр от 0 до 9 и латинских букв a–f или A–F, и она должна начинаться с символов 0x, 0X.

Пример: 10сс 8сс 16сс (сс – система счисления)

10	012	0x1a
16	020	0x10
25	031	0x19

По количеству значащих цифр целые десятичные константы делятся на:

- int – целое;
- unsigned int – беззнаковое целое;
- long int – длинное целое;
- unsigned long – беззнаковое длинное целое.

Чтобы задать для любой целой константы тип long, достаточно в конец записи константы приписать букву L или l; буква U или u в конце записи числа указывает на тип unsigned int.

Например:

5L – целая константа длинного типа;

321 – целая константа;

2746U – беззнаковое целое.

Константа с плавающей точкой – это десятичное число, которое состоит из целой, дробной части и экспоненты (например, 111.75, 2.5-E).

Символьная константа – это символ, заключенный в апострофы, либо целая константа, которой предшествует обратная косая черта.

Например, 'A', '\33', '\042'.

Все символьные константы имеют целый тип. К символьным константам относятся также управляющие последовательности (например, '\n' – переход на новую строку, '\0' – знак кода ноль, '\t' – горизонтальная табуляция, '\\' – обратная косая черта, '\'' – апостроф, '\"' – кавычки).

Строка – это последовательность символов, заключенная в кавычки («Это строка»). В конце каждой строки компилятор добавляет нулевой символ ('\0'). Строка описывается как массив символов. Размер массива задается на единицу больше, чтобы предусмотреть место под код нуля. Строка может располагаться в программе на нескольких строках экрана. Для этого используется обратная дробная черта.

Пример: «Это очень длин \
ная строка»

Обратная дробная черта игнорируется компилятором, и следующая строка считается продолжением.

1.2 Типы данных

Классификацию типов данных языка Си можно представить в виде следующей схемы:

Основные (базовые) типы данных:

char – символьный;

int – целый;

short – короткое целое;

long – длинное целое;

float – число с плавающей точкой с одинарной точностью;

double – вещественное число с двойной точностью.

При определении основного типа можно поставить модификаторы: signed (знаковый), unsigned (беззнаковый). При отсутствии модификатора по умолчанию принимается знаковый модификатор. В таблице приведен диапазон значений и размер в байтах базовых типов данных.

Таблица 1

Диапазоны значений типов данных

Тип данных	Диапазон значений	Размер в байтах
char (signed)	-128 . . . 127	1
char (unsigned)	0 . . . 255	1
short, int (signed)	-32 768 . . . 32 767	2
short, int (unsigned)	0 . . . 65535	2
long (signed)	-2 147 483 648 . . . 2 147 483 347	4
long (unsigned)	0 . . . 4 294 967 295	4
float	3.4e-38 . . . 3.4e38	4
double	1.7e-308 . . . 1.7e308	8
long double	3.4e-4932 . . . 3.4e4932	10

Все переменные должны быть объявлены до их применения. Описание переменных начинается с указания типа данных, затем следует список идентификаторов переменных через запятую. Описание завершается знаком «;». Примеры описания переменных:

int i, j, k;

double max, min;

При описании переменных, их можно сразу проинициализировать.

Например:

```
int i=0, j=1, k=3;
char s='S', v='\n';
```

В языке Си предусмотрены именованные константы, которые описываются при помощи модификатора const (неизменяемый). Константе нельзя присвоить иное значение, чем было установлено при объявлении. Инициализация при объявлении именованной константы является обязательной.

```
//описание именованной константы
const float pi=3.14;
```

1.3 Выражения

Поскольку язык Си является типизированным языком, в нем определены явные и неявные преобразования типов данных. Неявные преобразования выполняются при бинарных арифметических операциях и при операции присваивания.

В выражении желательно использовать переменные и данные одного типа. Если операция выполняется над разными типами данных, то происходит «повышение» типа.

Последовательность от высшего к низшему типу следующая:

- double
- float
- long
- int
- char

Применение модификатора unsigned повышает уровень типа. При выполнении операции присваивания может произойти как повышение, так и понижение типа. Преобразование значения с плавающей точкой к целочисленному типу сводится к отбрасыванию дробной части.

Явное преобразование типов задается следующим образом: перед преобразуемым выражением ставится имя типа в круглых скобках.

Например,

```
int a,b;
float c;
c=(float)(a/b);
```

Результат операции a/b будет целое число, так как оба операнда – целые; указав явное преобразование результата операции к вещественному типу, мы получаем нужное значение.

Арифметические операции:

1. Унарные операции (т. е. с одним аргументом):

-a – смена знака;

++a – инкремент (увеличение на единицу) соответствует выражению a=a+1;

--a – декремент (уменьшение на единицу) соответствует выражению a=a−1.

Операции могут быть префиксные (знак стоит перед аргументом) и постфиксные (после аргумента). Пример: a++; a--;

Префиксные операции вычисляются перед вычислением значения всего выражения. При постфиксной операции сначала вычисляется значение всего выражения, и лишь потом идет изменение аргумента.

Например, рассмотрим следующий фрагмент программы:

```
int n=5, x, y;
x=n++; // после выполнения выражения x=5, а n=6
y=++n; // после выполнения выражения y=7, а n=7
```

2. Бинарные операции:

+ – сложение;

− – вычитание;

* – умножение;

/ – деление;

% – вычисление остатка от деления (только над целыми числами).

Пример, 10%3 результат операции равен 1.

Операции сравнения:

== – равно

!= – не равно

> – больше

>= – больше или равно

< – меньше

<= – меньше или равно

Результатом сравнения является целое число (тип int). Оно равно 0 при невыполнении условия (ложь), в остальных случаях истина.

Логические операции:

! – НЕ (инверсия)

&& – логическое И. Значение правого операнда вычисляется, только если значение левого операнда – true.

|| – логическое ИЛИ. Значение правого операнда вычисляется, только если значение левого операнда – false.

Результат операции целое число (0 – если ложь).

Примеры логических выражений: (a>0)&&(b>0), (a<=0)&&(a>=-10).

1.4 Структура программы на Си

Любая программа имеет начало, т.е. оператор с которого начинается ее выполнение. Заканчиваться программа может в нескольких местах.

Программа на Си состоит из одной или нескольких функций. Одна из этих функций обязательно должна иметь имя main ( ). Это основная функция, с которой начинается выполнение программы. Она может иметь любое место в программе. Тело функции должно быть заключено в фигурные скобки. Перед именем функции должен стоять тип возвращаемого значения. Пока будем

считать, что наша функция не возвращает никаких значений, поэтому ее тип – void (пустой). Пример описания функции:

```
 void main ( )
 {тело функции
 }
```

Структура файла в программе: вначале идут директивы препроцессора, потом описание внешних переменных, а затем описание функций.

Пример простой программы на языке Си:

```
#include <stdio.h> //- директива препроцессора
void main ( ) // начало описания основной функции
{int a,b; // описание переменных
a=17;
b=-123;
int c;
c=a+b;
printf("Сумма чисел %d и %d равна %d \n",a,b,c); // функция вывода
/* %d – формат для вывода целого числа */
} // конец основной функции и программы.
```

1.5 Препроцессор

Препроцессор – это часть компилятора. Директивы препроцессора не превращаются в код, они управляют действиями компилятора. За ними не ставится «;». Каждая директива начинается с новой строки. Директивы производят замену лексем в исходном тексте, вставку внешних файлов и т. д. Директива обязательно начинается со значка «#». Дальше идет ключевое слово. Директива препроцессора может находиться в любом месте файла, но ее действие распространяется только на остаток.

Язык Си гибкий и расширяемый. На основе базовых команд создано множество более емких команд. Естественно, нужно использовать эти созданные расширения, это упрощает разработку программы. Многие «добавленные» команды уже вошли в стандарт языка, но, прежде чем их использовать в программе, нужно подключить ту библиотеку, в которой реализована конкретная команда.

Директива включения внешнего файла:

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

У нее есть две модификации:

1) #include <Name>

2) #include “Name”

Name – имя внешнего файла. Если имя файла стоит в угловых скобках, то этот файл следует искать в стандартной библиотеке языка Си. Файлы стандартной библиотеки имеют расширение «h», и в них хранятся все стандартные функции (<math.h> - математические функции). Если имя файла указано в кавычках, то его следует искать в текущем каталоге. В качестве Name

можно указывать не только имя файла, но и весь его путь, начиная с диска (например, #include "a:\lab\data.cpp").

Директивы препроцессора

1) Определяющая директива

define идентификатор строка подстановки

define MAX 20

Она может стоять в любом месте программы, но ее действие распространяется на оставшуюся часть программы. В качестве строки подстановки может быть константное выражение, но все переменные должны быть определены выше. В качестве строки подстановки могут быть заданы операторы.

2) # undef идентификатор

Отменяет препроцессорное определение идентификатора.

3) # ifdef идентификатор

Проверяет, определен ли на данный момент идентификатор.

4) # ifndef идентификатор

Проверяет, не определен ли идентификатор на данный момент.

5) директива условной компиляции

if Exp text p

elif Exp1 Text1

elif Exp2 Text2

…

else Text f

endif

Позволяет отменить компиляцию отдельных частей программы.

Exp – это константное выражение, которое проверяется и не должно содержать sizeof, приведение типа и элементов перечисления.

Ветвь else может отсутствовать, ветвь elif может повторяться произвольное число раз.

6) директива ошибки

error сообщение

При срабатывании директивы компиляция прекращается, в сообщении выводится стандартный поток вывода об ошибках.

Пример:

```
 # indef _ _ COMPACT _ _
 # error NEED COMPACT MODEL
 # endif
```

Прагмы

Прагма позволяет управлять специфическими возможностями компилятора. Прагмы вставляются в текст программы и управляют работой компилятора на отдельных частях программы.

Каждая прагма начинается с новой строки обязательно со значка «#». Дальше идет ключевое слово. Например:

pragma message сообщение

посылает сообщение в stdout.

pragma argsused.

Эта прагма, поставленная перед функцией, подавляет сообщение компилятора о неиспользуемых аргументах.

1.6 Операторы

Оператор – единица выполнения программы. Оператор должен заканчиваться символом «точка с запятой» – «;». Любой оператор может быть помечен меткой. Метка состоит из имени и символа двоеточия – «:». Метки описывать не надо. Несколько операторов можно объединить в блоки. После блока «;» не ставится.

Операторы можно разделить на 3 основные группы:

1. Операторы выражения;
2. Пустые операторы и блоки;
3. Операторы, начинающиеся с ключевого слова:
   1) условные операторы;
   2) операторы циклов;
   3) операторы переходов.

Оператор выражения

Любое выражение, которое заканчивается точкой с запятой, является оператором (++i;). Оператор присваивания может быть простым и составным. Составное присваивание имеет вид: выражение1 @= выражение2;. Знак @ – это один из знаков операций: * / % + - << >> & ^.

Например,

a=13; – простое присваивание;

a+=2; – составное присваивание означающее a=a+2;

b−=3; – составное присваивание означающее b=b−3;

При этом переменной присваивается значение, скорректированное с помощью знака операции и правого аргумента.

В одном выражении может быть несколько присваиваний, они выполняются справа налево (i=j=k=l=0;).

Пустые операторы и блоки

Блок – это составной оператор, состоит из фигурных скобок, помечающих начало блока операторов и конец данного блока. Все, что объединено в блок – синтаксически считается одним оператором.

Пустой оператор состоит только из «;» и ничего не выполняет. Применяется для того, чтобы пометить меткой блок. Например,

metka1 : ; {…}

Используется также в операторах цикла, когда действий нет, но по синтаксису оператор там должен быть.

Условный оператор if

Существует две разновидности этого оператора:

1) if (условие) оператор1;

2) if (условие) оператор1;
else оператор2;

Если условие истинно, то выполняется оператор1. Если условие ложно, то в первом случае управление передается на следующий оператор после условия, а во втором – выполняется оператор2.

На месте оператора может стоять блок, если необходимо выполнить несколько действий. Оператор if может быть вложенным, причем else будет относиться к ближайшему по иерархии if.

Пример:

```
if (условие 1)
    {if (условие 2) оператор1;}
    else оператор2;
```

Условная операция

Условная операция является краткой записью оператора if. Формат операции:

```
Выражение1? выражение2: выражение3;
|                  |                |
if                 then             else
```

Выражение 1 – проверяемое условие; может быть целого, плавающего типа или указатель.

Если выражение 1 истинно, то выполняется выражение 2, если выражение 1 ложно, то выполняется выражение 3.

Пример: найти максимальное из двух значений и записать его в переменную max.

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

Оператор цикла for

Формат оператора:

for(выражение1; выражение2; выражение3) оператор;

выражение1 – задает начальное значение для переменных; выполняется один раз вначале всего цикла.

выражение2 – условие выхода из цикла. Проверяется перед каждым возможным выполнением цикла. Если условие ложно, то цикл заканчивается.

выражение3 – обычно это модификация переменных, выполняется в конце каждой итерации.

Пример: for(i=1; i<=10; i++) оператор;

Возможности оператора for:

1. Можно считать как в порядке убывания, так и возрастания переменной.
2. Шаг параметра может быть любым. for(i=0; i<=2; i+=0,23) оператор;
3. Можно вести подсчет с помощью символов.

```
for(char ch='a'; ch<='z'; ch++)
printf("Величина кода ASCII для символа %c равна %d \n",ch,ch);
```

4. В качестве выражения 3 можно использовать любое правильное выражение. Оно будет выполняться в конце каждой операции.

for(x=1; y<=75; y=x++*2) printf(“x=%f y=%f \n”,x,y);

5. Можно пропускать одно или более выражений, но при этом сохранять символ «;». Пример:

for(a=3; a<=25;) a=a*3; for(; ;) {тело цикла} – это бесконечный цикл, пустое условие считается истинным.

6. Выражение 1 не обязательно должно инициализировать параметры цикла. Главное, что оно выполняется один раз при начале цикла. Пример:

for (printf (“Введите числа \n”); num<10;) scanf (“%d”,& num);

7. Параметр цикла может изменяться в теле цикла.

8. Выражений 1 и выражений 3 в цикле может быть несколько. Они соединяются с помощью операции «,». Операция «,» объединяет несколько выражений в одно и гарантирует, что самое левое будет выполняться первым.

Пример: for(i=0,j=0; i<j; i++; j−−;) оператор;

Оператор цикла while

Формат оператора:

while (выражение) оператор;

Тело цикла выполняется пока выражение истинно. Если тело цикла состоит из нескольких операторов, то используется блок – {…}.

Пример бесконечного цикла:

while(1) { printf(“Это никогда не кончится\n”); }

Пример вывода квадратов чисел от 1 до 10:

i=1;

while(i<=10) { printf(“%2d%4d\n”,i,i*i); }

Оператор цикла do

Формат оператора:

do оператор; while (выражение);

Применяется, когда необходимо, чтобы тело цикла выполнилось хотя бы один раз. Если выражение истинно, то тело цикла опять выполняется, а если ложно, то осуществляется выход из цикла.

Пример вывода квадратов чисел от 1 до 10:

i=1;

do

printf(“%2d%4d\n”,i,i*i);

while(i<=10);

Операторы переходов

1. Переход по метке – **goto** метка. Пример:

goto m1;

m1:; {… или m1: оператор;

Переход по метке возможен только внутри блока, функции.

2. Оператор продолжения – **continue**. Оператор вызывает остановку текущей итерации цикла и вызывает новую итерацию.

Пример (вывод четных чисел от 0 до 100):

for (i=0; i<100; i++)

```
{if (i%2) continue;
printf ("%d\n", i) ;}
```

3. Оператор разрыва – **break**. Используется в операторах цикла и в операторе переключателе. Приводит к выходу из конструкции. Если используются вложенные циклы, то происходит выход из самой внутренней.

Оператор «переключатель»

Этот оператор предназначен для организации выбора одного из множества вариантов. Формат оператора:

```
switch (выражение)
{ case метка 1: оператор;
     . . .
  case метка N: оператор;
  default: оператор;
}
```

Выражение может быть одним из основных типов (лучше целые или символьные константы). Каждая ветвь помечается словом «case», и рядом с ним стоит метка. Значение выражения сравнивается с меткой, и если они совпали, выполняется оператор. Если совпадений с ветками не произошло, то выполняется ветвь default.

Одна ветвь может быть помечена сразу несколькими метками:

```
case м1:
case м2:
case м3: оператор;
```

Операторов в ветви может быть несколько, и их не нужно брать в блок за исключением default.

```
Пример: case м10:оператор1;
                 оператор2;
                 оператор3;
        case м11:
```

Если произошло совпадение по метке, то далее будут выполнены все операторы, начиная с указанного оператора и до default. Поэтому в конце каждой ветви ставится оператор «break».

```
switch (выражение)
{ case M1: оператор; break;
  . . .
  case MN: оператор; break;
  default: оператор;
}
```

Пример: Фрагмент программы выполнения операции над числами по заданному знаку. Считаем, что знак операции хранится в переменной «sign», а числа в x и y.

```
switch (sign)
{case '-': z=x-y; break;
 case '+': z=x+y; break;
```

```
  case '*': z=x*y; break;
  case '/': z=x/y; break;
  default: printf("Неизвестная операция !\n");
}
```

Форматизированный ввод данных

При использовании функций форматного ввода и вывода данных программа должна содержать включение файла <stdio.h>. Формат оператора:

scanf (управляющая строка, аргументы);

Управляющая строка берется в кавычки. Она содержит спецификации преобразования, которые указывают, как нужно интерпретировать входной символ.

Аргументы – это ссылки, которые указывают, где нужно хранить входную информацию.

Пример: int m;

scanf ("%d", &m);

Необходимо запомнить 2 правила:

1) Если нужно ввести значение одного из основных типов, то перед именем переменной обязательно нужно написать «&» (амперсант).

2) Если вводится строковая переменная, то «&» перед именем не нужен.

Управляющая строка имеет вид:

% [*] [длина] тип

В квадратных скобках стоят необязательные параметры.

* – входное поле читается, но не сохраняется;

длина – положительное десятичное число, которое задает максимальное количество читаемых символов, если раньше не встретятся разделители (пробел, табуляция, переход на новую строку);

тип – задает тип читаемых данных.

Таблица 2

Символы типа для ввода

Символ типа	Ожидаемый тип ввода
d	десятичное целое
o	восьмеричное целое
x	шестнадцатеричное целое
n	короткое целое
u	десятичное целое без знака
ld	длинное десятичное целое
lo	длинное восьмеричное целое
lx	длинное шестнадцатеричное целое
c	одиночный символ, включая пробел, перевод строки
s	строка символов
F, e, g	величины типа float
le, lf	величины типа double

Форматизованный вывод данных

Существует два основных типа экранных режимов – текстовый и графический. Рассмотрим вывод текста на экран в текстовом режиме. Этот режим устанавливается при выполнение программы по умолчанию.

В стандартном режиме экран разбивается на 25 строк и 80 столбцов. Любая позиция на экране определяется номером строки и номером столбца. Нумерация строк и столбцов начинается с левого верхнего угла.

Формат оператора:

printf (управляющая строка, аргументы);

Управляющая строка показывает, как должны быть напечатаны аргументы, она заключается в кавычки. Она может содержать обычный текст и спецификации формата.

Спецификации формата имеют вид:

% [флаг] [длина] [точность] тип

Таблица 3

Символы типа для вывода

Символ типа	Тип вывода
d	int
i	int
u	unsigned
o	восьмеричное беззнаковое
x	16-ричное a-f
X	16-ричное A-F
f	double
e	double e
E	double E
g	double
c	вывод одиночного символа
s	вывод строки

Символ флажка управляет выравниванием вывода, печатью знака числа и пробелов.

Таблица 4

Символы флажка

Символ флажка	Значение	Значение по умолчанию
–	выравнивание влево внутри поля	вправо
+	присоединение знака “+” или “–”	печатается только “–”
Пробел	если число положительное, то к нему присоединяется пробел на месте знака	пробела нет

Окончание таблицы 4

Символ флажка	Значение	Значение по умолчанию
#	если печатаются числа по формату (o, x, X), то к выводимым числам присоединяются (0, 0x, 0X)	не печатается

Поле длины определяет минимальное число выводимых символов. Это не приводит к усечению выводимого значения.

Поле точности определяет количество цифр после запятой. Каждой спецификации в управляющей строке должен соответствовать свой аргумент.

Функцию printf можно использовать для преобразования данных. В качестве аргументов функции может быть константа, выражение, имя переменной или имя функции, которая возвращает значение основного типа.

Пример: 1) printf(“%d\n”,10); // будет выведено 10
printf(“%o\n”,10); // будет выведено 12
2) printf(“%c %d\n”, ’A’, ’A’); // будет выведено A65

2. ИСПОЛЬЗОВАНИЕ СОСТАВНЫХ ТИПОВ ДАННЫХ

2.1 Массивы

Массив – это совокупность однотипных элементов. Формат описания:

тип данных имя [размер];

«Тип_данных» задает тип элементов массива. «Размер» — количество элементов в нем Элементы массива нумеруются с нуля. Пример описания массива из 10 элементов с индексами от 0 до 9:

int a[10];

Элементы массива:

a_0, a_1, … , a_9 (a_{10} - не существует!)

Описание двумерного массива:

float d[10][10]; (каждый размер в своих скобках)

Обращение к элементу массива:

к одномерному – a[i]

к двумерному – d[i][j].

При описании массив можно проинициализировать, т. е. задать все его значения или часть значений.

Пример: int s[2][3]={{4,5,6},{7,8,9}}; (значения для инициализации заключаются в скобки).

При инициализации многомерных массивов начальные значения для каждой новой строки заключаются в фигурные скобки. Если отдельных фигурных скобок нет, то инициализация производится по мере возрастания индексов.

int s [2][3]={4,5,6,7,8,9};

int s[1][2]={{4,5,6},{7,8,9}};

int f[1][2]={10,11,12,13,14};

Проинициализируем часть массива:

int s [2][3]={{4,5}, {7}}; (инициализируем от начала каждой строки).

char p[2][2]={{'n'},{'y'}};

Массив s инициализируется полностью заданными значениями.

В массиве f из его шести значений (размер массива f – 2 строки и 3 столбца) инициализируется только первые 5 элементов (это элементы с индексами 0,0 0,1 0,2 1,0 1,1). В массиве p инициализируются только 2 элемента: p[0][0]='n' и p[1][0]='y'.

Если не проинициализировать элементы массива перед началом работы с ним, то внешние и статические массивы инициализируются нулем, а автоматические и регистровые будут содержать «мусор», оставшийся в этом участке памяти.

Если задан размер массива, то значения, не заданные явно, определяются в зависимости от класса памяти.

Для одномерного массива можно не указывать его размер, если он проинициализирован.

Пример: int p[ ]={1,2,3,4,5};

Пример: Вывод на печать двумерного массива в виде матрицы (a[N][M])

```
printf(“Исходный массив\n”);
for (i=0; i<N; i++)
{ for (j=0; j<M; j++)
  printf(“%4d”,a[i][j]);
  printf(“\n”);
}
```

2.2 Указатели.

В языке Си существует сильная взаимосвязь между указателями и массивами, настолько сильная, что указатели и массивы фактически надо рассматривать одновременно.

Указатель – это переменная, значением которой является ссылка на другой объект или адрес памяти, выделенной под объект.

Описание указателя:

int b; – описание переменной;

int *pb; – описание указателя (pb – указатель на переменную целого типа)

Прежде чем использовать указатель в программе, его нужно проинициализировать, т. е. занести в него необходимый адрес объекта.

pb=&b; - записываем в переменную pb адрес переменной b.

Операция & – это операция получения адреса объекта. Для обращения к значению переменной через указатель используется операция “*” (косвенная адресация).

*pb – мы обратились к значению переменной b.

*pb=*pb+1; // в значение b заносится 11 (10+1)

Операция получения адреса применяется к переменным и элементам массива.

Любое действие, которое достигается индексированием массива, может быть выполнено и с помощью указателя. Вариант с указателем в общем-то будет быстрее, но он, по крайней мере для начинающих, несколько тяжеловат для понимания.

Описание int a[10] определяет массив a размером в 10 элементов, т. е. это блок из 10 последовательных объектов, именуемых a[0], a[1],..., a[9]. Запись a[i] обозначает элемент в i-й позиции от начала.

Если pa — это указатель на целое значение, описанный как

int *pa;

то присваивание: pa=&a[0] устанавливает в pa ссылку на нулевой элемент массива a, т. е. pa содержит адрес a[0].Теперь присваивание x=*pa копирует содержимое a[0] в x.

Если содержимое pa указывает на отдельный элемент массива a, то по определению pa+1 указывает на следующий элемент, и, вообще, pa - i указывает на i-й элемент перед pa, а pa+1 — на i-й элемент после. Таким образом, если pa указывает на a[0], то *(pa+1) относится к содержимому a[1], pa+i есть адрес a[i], а *(pa+i) есть содержимое a[i].

Эти замечания справедливы вне зависимости от типа переменных в массиве a. Определение операции «добавление 1 к ссылке» и другой ссылочной арифметики подразумевает масштабирование, связанное с размером памяти для объекта, на который указывает ссылка. Таким образом, в pa+i значение i, прежде чем будет добавлено к pa, будет умножено на размер объекта, на который указывает pa. Очевидно, что между индексированием и ссылочной арифметикой связь очень тесная. Фактически любое упоминание массива приводится транслятором к ссылке на начало этого массива, т. е. имя массива есть ссылочное выражение. Это приводит к небольшому числу полезных следствий. Так как имя массива есть синоним для местоположения нулевого элемента, то присваивание pa=&a[0] можно записать и в таком виде: pa=a. Не удивительно теперь, по крайней мере на первый взгляд, что значение a[i] можно записать как *(a+i). Вычисляя a[i], транслятор сразу же переводит его в *(a+i); эти две формы полностью эквивалентны. Применяя операцию & к обеим частям этого равенства, получаем, что &a[i] и a+i также идентичны: a+i — адрес i-го элемента относительно a. С другой стороны, если pa — ссылка, то ее можно использовать с индексом: pa[i] идентично *(pa+i). Короче, любой массив и индексное выражение можно записать как ссылку и смещение и, наоборот, причем это можно делать даже в одном операторе.

Однако между именем массива и указателем есть одно различие, о котором следует всегда помнить. Указатель есть переменная, так что pa=a и pa++ суть осмысленные операции. Имя же массива — константа, а не переменная, поэтому конструкции вроде a=pa или a++, или p=&a недопустимы.

Пример. Программа распечатки содержимого одномерного массива с использованием указателя. Массив предварительно проинициализирован.

```
#include<stdio.h>
void main()
{
int p[5]={1,2,3,4,5};
int *ref;
ref=p;
printf("\n");
for(int i=0;i<5;i++)
printf("%d\t",*(ref+i));
}
```

Пример. Фрагмент программы, реализующий заполнение двумерного массива случайными элементами с использованием указателя.

```
int a[5][6],pa;
. . . . . .
randomize();
pa=&a[0][0];
for(i=0;i<5*6;i++)
    *(pa+i)=random(2);
```

Рассмотрим теперь, как получить доступ к элементу многомерного массива, используя указатель. Допустим, в программе описан трехмерный массив и указатель на него:

```
int arr[L][M][K], *ptr;
ptr=&arr[0][0][0];
```

Массив arr состоит из L элементов, каждый из которых — двумерный массив M на N. Каждый массив M на N в памяти располагается по строкам.

Необходимо получить доступ к элементу arr[i][j[k]. Последовательно это вычисляется так:

ptr — адрес 0-го массива M на N
ptr+i*(M*N) — адрес i-го массива M на N
ptr+i*(M*N)+j*N — адрес j-й строки i-го массива M на N
ptr+i*(M*N)+i*N+k — адрес элемента arr[i][j][k]
(ptr+i(M*N)+i*N+k) — значение элемента arr[i][j][k]

2.3 Строки

Строки — это последовательность символов, заключенная в кавычки. В конце каждой строки компилятор добавляет нулевой символ, представляемый управляющей последовательностью ‘\0’.

Строка описывается как массив символов. Число элементов массива равно числу элементов в строке плюс символ конца строки (\0). Символьная строка в программе может располагаться на нескольких строках. Для переноса используется обратная дробная черта с последующим нажатием клавиши ввод.

Обратная дробная черта игнорируется компилятором, и следующая строка считается продолжением предыдущей.

Для работы со строками очень удобно использовать указатели.

Пример. Записать введенную строку символов в обратном порядке.

```
#include<stdio.h>
void main()
{
int top,bot;
char string[10],temp;  /*описание строки как массива  символов*/
scanf("%s",string);
/* при  вводе  строк  символ & не используется,  так как имя массива является указателем на его начало */
for(top=0,bot=10;top<bot:top++,bot--)
    {
     temp=string[top];
    string[top]=string[bot];
    string[bot]=temp;
    }
    printf("%s\n",string);
}
```

Для ввода одиночного символа из входного потока используется функция getchar(). Для вывода одиночного символа используется функция putchar(ch), где ch — выводимый символ. Аргументом функции вывода может быть одиночный символ (включая знаки, представляемые управляющими последовательностями), переменная или функция, значением которой является одиночный символ.

Пример. Программа вводит из входного потока один символ, а затем выводит его на экран.

```
#include<stdio.h>
void main()
{
char ch;
ch=getchar();
putchar(ch);
}
```

Пример. Программа, реализующая чтение и печать символов до ввода знака ∗.

```
#include<stdio.h>
#define STOP ∗
void main()
{
char ch;
while((ch=getchar())!=STOP)     putchar(ch);
}
```

В условии, стоящем после ключевого слова while, реализовано сразу три действия: ввод символа с помощью функции getchar(); занесение символа в переменную ch; проверка на признак конца ввода. Для реализации проверки на конец ввода последовательности символов используется идентификатор STOP, определенный директивой препроцессора.

Для работы со строками есть набор функций. Для ввода со стандартного устройства ввода (клавиатуры) чаще всего используются библиотечные функции из модуля стандартного ввода-вывода: scanf и gets.

Для ввода строки с помощью функции scanf используют формат «%s», причем обратите внимание на то, что перед идентификатором строки не используется знак адреса «&», так как одномерный массив уже представлен указателем на его начало:

```
char *s;
scanf("%s", s);
```

Функция gets() считывает символы до тех пор, пока не достигнет символа перехода на новую строку. Функция принимает все символы вплоть до символа перевода строки, но не включает его. К концу строки добавляется завершающий ноль ('\0').

Для вывода строк можно использовать две функции printf и puts. В функции printf в качестве формата передается "%s", удобство при использовании этой функции заключается в том, что помимо строки можно сразу выводит данные других типов. Особенность функции puts заключается в том, что после вывода строки автоматически происходит переход на следующую строку.

Операции со строками

Описание функции работы со строками содержится в файле <string.h> библиотеки стандартных функций. Рассмотрим некоторые из них:

1. Соединение последовательностей символов.

```
char *strcat(char *s1,char *s2)
```

Функция возвращает указатель на полученную строку.

2. Поиск первого вхождения символа в строку.

```
char *strchr(char *s,int c)
```

Функция просматривает строку (обращение к строке с помощью указателя s) и ищет символ с кодом c. Возвращает указатель на найденный символ или пустое значение.

3. Сравнение строк.

```
int strcmp(char *s1,char *s2)
```

Сравниваются две указанные строки. Результат — переменная типа int:

```
s1<s2    отрицательное значение
s1==s2   значение 0
s1>s2    положительное значение
```

4. Копирование символов.

```
char *strcpy(char *s1,char *s2)
```

Копируется последовательность символов, указанная параметром s1 по адресу s2.

5. Определение длины строки (без завершающего нуля).

int strlen(char *s)

6. Поиск вхождения одной последовательности символов в другую.

char *strstr(const chat *s1, const char *s2) – возвращает указатель на положение первого появления последовательности символов из s2 в строке s1 (исключая завершающие пробелы); возвращает NULL, если совпадений не найдено.

7. Переформирование строки в отдельные знаки.

chat *strtok(chat *s1,const chat *s2) – эта функция переформирует строку s1 в отдельные знаки; строка s2 содержит символы, которые используются в качестве разделителей. Функция вызывается последовательно. Для первого вызова s1 должен указывать на строку, которую необходимо разбить на знаки. Функция находит разделитель, который следует за символом, не являющимся разделителем, и заменяет его пробелом. Она возвращает указатель на строку, содержащую первый знак. Если ни одного знака не найдено, она возвращает NULL. Чтобы найти следующий знак в строке, необходимо вызвать функцию опять, но первым аргументом поставить NULL. Каждый последовательный вызов возвращает указатель на следующий знак или на NULL, если больше знаков не найдено.

Пример разделения предложения на отдельные слова:

```
char s[80]; // Исходное предложение
char slova[20][20]; // Массив слов в предложении
char *p;
int sl; //Счетчик слов в предложении
char *r="!;:,.?- ";        // Возможные разделители между словами
gets(s); //Ввод строки
p=strtok (s, r);
  while (p)
    {
    sl++;
    strcpy(slova[sl],p);
    p=strtok (NULL, r);
    }
```

Проверка символов

Для проверки символов используются функции, возвращающие значения «истина» или «ложь». Прототипы этих функций описаны в файле <ctype.h> библиотеки стандартных функций. Рассмотрим некоторые из них:

Функция	«Истина», если
isalpha(c)	c — символ алфавита
isupper(c)	c — символ верхнего регистра

islower(c)	c — символ нижнего регистра
isdigit(c)	c — цифра от 0 до 9
isxdigit(c)	c — шестнадцатеричная цифра
isalnum(c)	c — буква или цифра
isspace(c)	c — пробел, табуляция, перевод строки

Пример. Программа проверяет введенную строку и затем выводит на экран ту ее часть, которая начинается с символа 'y'.

```
#include<stdio.h>
#include<string.h>
void main()
{
char string[10];
char *ptr;
printf("Введите строку\n");
gets(string);
ptr=strchr(string,'y');
printf("это строка %s\n",ptr);
}
```

2.4 Структуры

Записи (структуры) являются одними из основных структур данных в языках программирования высокого уровня. Понятие записи используется при машинной обработке различных документов, таблиц, баз данных.

Запись — это структура, состоящая из фиксированного числа компонент, называемых полями. В одном поле данные имеют один и тот же тип, а в разных полях могут иметь разные типы, за исключением функций.

```
struct {
        type1 id11,id12,...,id1n;
        type2 id21,id22,...,id1m;
        .....................
        typei idk1,idk2,...,idkp;
       } описатель [описатель];
```

Здесь idij — идентификаторы полей; typei — типы полей; описатель — имя переменной с заданной структурой.

Пример. Описание переменных date1 и date2. Каждая переменная содержит два поля.

```
struct {
        int year;
        short day;
       } date1, date2;
```

Для описания структуры удобно использовать шаблоны. Описание шаблона идет без последующего списка переменных.

Формат шаблона следующий

```
struct имя_типа_структуры
        {
         список описаний;
        };
```

Описание шаблона является описанием нового типа данных. Далее можно описывать переменные, используя имя шаблона.

Пример. Описание шаблона для даты (день, месяц, год).

```
struct data{
            int day;
            char month[10];
            int year;
           };
   struct data d1,d2,d3; /*описание переменных*/
```

При описании возможна инициализация переменной.

```
struct data d1={4,"Сентябрь",1998};
```

Можно задавать имя типа структуры с помощью описателя типа typedef.

Описатель типа

Этот описатель позволяет создать свое собственное имя типа.

Формат описателя типа:

typedef спецификатор_типа описатели;

Спецификатор типа — это основной или производный тип данных или тип, который ранее определен программистом. Описатель — это новое имя созданного нами типа.

Пример. Описание нового типа данных «дата», состоящего их трех полей.

```
typedef struct{
            int day;
            char month[10];
            int year;
           }data;
   data d1,d2,d3;
```

Структура не может содержать в качестве элемента структуру такого же типа, но может включать указатель на структуру этого типа при условии, что в объявлении структуры указано имя типа. Это позволяет создавать связанные списки структур.

Пример. Описание бинарного дерева.

```
struct tree{
            int number;
            struct tree *left;
            struct tree *right;
           };
```

Доступ к элементу структуры осуществляется с помощью символа «.», обозначающего операцию получения элемента структуры.

Примеры обращения к элементам структур, описанных выше:

d1.day

d2.year

Доступ к элементам структур может осуществляться и с помощью указателей.

Пример. Описание указателя на структуру и обращение к ее элементам.

```
data * ptr;
ptr=&d2;
ptr->day=4;
ptr->month="Декабрь";
ptr->year=1998;
```

Обращение к элементам структуры можно записать еще следующим образом:

```
(*ptr).day=4;
(*ptr).month="Декабрь";
(*ptr).year=1998;
```

Пример. Описание массива, элементами которого являются структуры типа data.

```
data d[5];
```

Пример. Программа, реализующая ввод списка студентов и вывод информации о студенте по его номеру в списке. Ввод информации о студенте реализован в виде отдельной функции.

```
#include<stdio.h>
const int n=3;
typedef struct{
                int num;
                char fam[20];
                char name[15];
                }student;
student mas[n];
student *ptr=&mas[0];
void vvod(student *p)
    {
    scanf("%d",&p->num);
    scanf("%s",p->fam);
    scanf("%s",p->name);
    }
void main()
{
int i,d;
for(i=0;i<n;i++)
```

```
vvod(ptr+i);
printf("Введите номер студента в списке\n");
scanf("%d",&d);
for(i=0;i<n;i++)
if (mas[i].num==d)    printf(" Студент %s  %s\n",mas[i].fam,mas[i].name);
}
```

Процедуре vvod передается значение указателя на элемент массива mas, который является структурой. В приведенной программе использованы оба способа обращения к элементу структуры (с использованием указателя и без него).

2.5 Перечисление

Если переменная может принимать значения лишь из определенного ряда значений, то ее можно описать как перечисление.

Формат описания:

enum имя типа {список определений} описатель[ли];

Например: enum Color {Red, Green, Blue} h1;

Описана переменная h1, которая относится к типу Color, она может принимать только 3 значения: Red, Green, Blue. Пример ее использования:

```
h1=Green;
h1++; //  Blue.
```

Каждый идентификатор связан с целым числом (по умолчаниюнумерация начинается с 0 и далее увеличивается на единицу). Идентификатору можно присвоить значение любого константного выражения. Оно должно быть целое, но может быть и отрицательное. Следующему элементу присвоится значение на единицу большее, если оно не задано явно. Например:

enum Color {Red (=0), Green=3 (=3), Blue (=4)} h2;

Значения идентификаторов в этом случае будут следующие: Red=0, Green=3, Blue =4.

Правила использования перечислений:

1. Перечисления могут содержать повторяющиеся значения;

2. Идентификаторы в списке определений должны отличаться от других идентификаторов в программе в области видимости;

3. Имя типа перечислений должно быть уникальным;

4. Если тип уже объявлен, то переменные можно описать следующим образом:

enum Color m1, m2, m3;

Пример: программа, которая по заданному дню определяет какой день был вчера и какой будет завтра.

```
# define NUMDAYS 7
enum DAYS {sun, mon, the, wen, thi, fri, sat};
void main ( )
{enum DAYS day1, day2, day3;
```

```
  enum DAYS daybefore (enum DAYS);   // Описаны
  enum DAYS dayafter (enum DAYS);    // 3 прототипа
  void printday (enum DAYS);         // функций
// Значения перечислений с клавиатуры вводить нельзя и из файла тоже.
// Поэтому зададим значения присваиванием.
day1=the;
day2=daybefore (day1);
day3=dayafter (day1);
printf (“Если сегодня  “);
printday (day1);
printf (“, то завтра будет “);
printdat (day3);
printf (“, а вчера был[а] “);
printday (day2);
printf (“\n”);
}
// Описание функции определения вчерашнего дня.
enum DAYS daybefore (enum DAYS day)
{ int prev;
  prev=(day-1) % NUMDAYS;
  return (enum DAYS) (prev<0) ? (NUMDAYS-1): prev;
}
// Описание функции определения завтрашнего дня.
enum DAYS dayafter (enum DAYS day)
{ return (enum DAYS) ((day+1) % NUMDAYS)
}
void printday (enum DAYS day)
{ static char *days [ ] = {“воскресенье”, “понедельник”, “вторник”,
“среда”,  “четверг”, “пятница”, “суббота”};
int day_i=day;
if (day_i<0 || day_i>6)   //проверка выхода за границу дней
printf (“Ошибка\n”);
else printf (“%s”, days[day_i]);
}
```

2.6 Объединения

Объединение – это переменная, которая может хранить в одной и той же области памяти значения разных типов (не одновременно), т. е. объединение аналогично структуре, все элементы которой разного типа, а размер структуры зависит от размера наибольшего элемента. Все элементы хранятся, начиная с одного и того же адреса.

Описание объединения:

union имя объединения {список объявлений} описатель[ли];

Пример: union num {int a;
float b;
double c;
} n1, n2;

Размер памяти будет выделен double (так как он самый большой). Обращение к элементу объединения такое же, как к элементу структуры. Например: n1.a=10; n2.b=2.54;

2.7 Поля битов

Поля битов должны быть описаны как элемент структуры. В структуре описание битового поля следующее:

Тип_данных идентификатор: константное выражение;

Константное выражение задает количество бит в поле, оно должно быть целым и неотрицательным.

Тип выражения может быть знаковый или беззнаковый, целый (char, int, long). Знаковый тип поддерживается компилятором только синтаксически, на самом деле он преобразуется в беззнаковый.

Пример описания структуры с битовыми полями:

```
struct my_struct
{…
unsigned flag1:1;
nsigned flag2:1;
 …
} s1;
```

Идентификатор – необязательный элемент описания, его может и не быть. Неименованные поля используются для выравнивания.

Размер битового поля может быть нулевым, тогда следующее битовое поле будет начинаться на границе типа int. Обращение к полям бита точно такое же, как к полям структуры:

s1.flag1;

Побитовые операции

Аргументы и результат побитовых операций только целого типа.

1) Операции сдвига:

>> – сдвиг вправо;

<< – сдвиг влево.

Пример записи: a<<b;

Это бинарные операции – левый аргумент (a) подлежит сдвигу, правый – (b) определяет число бит, на которое произойдет сдвиг.

Если мы правый аргумент возьмем отрицательным, или его размер больше a, то результат будет не определен.

При сдвиге влево происходит дополнение нулями справа. При сдвиге вправо метод заполнения освобождающихся левых битов зависит от типа левого аргумента. Если левый аргумент был объявлен как беззнаковый, то заполнение нулями, если знаковый, то копиями знакового типа.

Пример:

```
unsigned a=10; //00001010
int res=a>>2;   //00000010
// Переведем числа в двоичную систему счисления для иллюстрации
int res2=a<<2; //00101000
```

2) поразрядное (побитовое) логическое «и». Обозначается «&». Используется для выделения некоторой группы разрядов. Например:

```
c=x&0177;
```

3) поразрядное логическое «или». Обозначается «¦». Используется для включения битов. Например:c=x¦017;

4) операция обращения или инверсия. Обозначается «~». Превращает 1 в 0, 0 в 1. Например:

```
c=x&~077;
```

3. ФУНКЦИИ

3.1 Описание функции

Функция является самостоятельной единицей программы. При вызове функции ей могут быть переданы параметры. Функция может возвращать значение в место ее вызова. Каждая функция имеет вид:

```
тип функции   имя функции (список аргументов)
{тело функции
}
```

Тип функции – это тип возвращаемого значения (по умолчанию считается int). Если функция не должна возвращать значений, то опишем ее как void.

Имя функции не должно совпадать с именами стандартных функций из библиотеки.

Пример: Функция, проверяющая является ли заданный символ русской буквой.

```
int rus (char c)
{ if ((c>=’А’) && (c<=’Я’))
return 1;
else
return 0;
}
```

Оператор return возвращает значение в место вызова.

Общий вид return:

```
return (выражение);
```

Тип возвращаемого выражения определяется типом функции. Выражение не может быть массивом, структурой, объединением или функцией. Можно возвращать указатели.

Существует два способа описания аргументов функции:

1. int rus (char c)
   {. . .
    }
2. int rus (c)   // В списке аргументов можно использовать только их
   char c; // имена, а описание их типов сделать ниже.
   {. . .
    }

Аргументы в списке разделяются запятыми. Если описание функции располагается после ее вызова, то необходимо использовать прототип функции, чтобы компилятор мог проверить соответствие описаний. Прототип функции имеет такой же формат, что и определение функции, только заканчивается «;». Он не имеет тела функции, в списке аргументов можно указывать только их типы. Таким образом, прототип задает тип функции, имя функции, количество и тип аргументов.

Пример: программа, которая определяет, являются ли вводимые символы с клавиатуры русскими буквами.

```
#include <stdio.h>
char slovo [20];
void main( )
{ int rus (char) c;    // прототип функции
char ch;
int i=a;
while ((ch=getchar( ))!='\n')
{if (rus (ch))
slovo[i+1]=ch;
else printf ("Не русская буква");
}
printf ("Это слово %s из %d букв \n", slovo);
}
int rus (char c)
{ if ((c>='А') && (c<='Я'))
return 1;
else
return 0;
}
```

3.2 Рекурсивные функции

Если функция может вызвать саму себя, то этот вызов называется рекурсивным. Рекурсивные вызовы функции замедляют выполнение программы.

Пример: функция вычисления факториала.

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

3.3 Использование указателей для связи между функциями

Связь параметра функции с аргументом (фактическим параметром) осуществляется только по значению. Для связи через переменную необходимо передавать параметры в виде указателей.

Пример программы, иллюстрирующей разные способы передачи параметров:

```
void main ( )
{ char alfa=’c’, beta=’b’;
void fun (char*, char); // прототип функции
fun (&alfa, beta); // вызов функции
printf (“%c %c”, alfa, beta);
}
void fun (ref, val) // описание функции
char *ref, val;
{ *ref=’j’;
   val=’*’;
}
```

В вызванную функцию передается указатель на переменную alfa и значение переменной beta. Функция своими действиями преобразует значение первой переменной, но не трогает значение второй, так как присваивание символа * осуществляется своей внутренней переменной, не связанной с beta. В результате работы программы будет напечатано «jb».

Если переменная участвует в работе нескольких функций, то необходимо передавать указатель на нее или описать ее глобальной переменной.

3.4 Параметры функции main

Параметры функции main передаются обычно из командной строки. У функции main два параметра: argc – это целое число, которое определяет число передаваемых параметров; argv – это массив указателей на передаваемые параметры.

Argv[0] содержит указатель на имя вашей программы.

Argv[1] содержит указатель на первый передаваемый параметр.

Таким образом, argc получается на единицу больше передаваемых параметров. Параметр argc определяет размер символьного массива argv.

Пример: программа распечатывает передаваемые ей аргументы.

```
# include <stdio.h>
void main (int argc, char *argv[ ])
{ printf (“Передаваемые параметры: \n”);
while (*argv)
printf (“%s\n”, argv++);
}
```

3.5 Установки по умолчанию

Для аргументов функции можно задать значения по умолчанию. Эти параметры следует расположить в конце списка передаваемых аргументов. При вызове функции значения этих параметров следует указывать лишь тогда, когда они отличаются от умалчиваемых значений. Например:

```
void function (int a, int b, int c=1) // значение переменной «c» задано
                                      // по умолчанию
{ тело функции
}
```

«int c=1» – установка по умолчанию. Она будет действительна до тех пор, пока при вызове явно не изменится.

Пример вызова данной функции:

```
function (5, 10, 0);
function (3, 9).
```

3.6 Хранение информации и вызов функции

Информация может храниться в оперативной памяти (ОП), в регистрах центрального процессора (ЦП) и в КЭШ памяти процессора. При запуске программы фрагмент памяти из ОП загружается в КЭШ память.

Если происходит вызов функции, то процессор должен:

- сохранить состояние своих регистров;
- поместить в стэк аргументы;
- загрузить в КЭШ память новый фрагмент, где начинается функция.

После окончания выполнения функции процессор должен вернуться к своему исходному состоянию, т. е. опять повторить все эти три действия.

Для того чтобы исключить перезагрузку КЭШ памяти функцию можно сделать подставляемой. В подставляемую функцию нельзя включать циклы и текст на ассемблере.

Описание подставляемой функции осуществляется с помощью ключевого слова inline. Например:

```
inline long fact (int n) {. . .}
inline int abs(int a){return a>0?a:-a;}
```

В этом случае в место вызова функции компилятор целиком вставит ее фрагмент кода. Это увеличит размер программы, но уменьшит время ее выполнения.

3.7 Перегрузка имен функций

Если функции выполняют приблизительно одинаковые действия над разными типами данных, то их можно описать с одинаковыми именами, но по количеству и типу аргументов они должны обязательно отличаться. Описание перегружаемых функций осуществляется с помощью ключевого слова overload.

Пример:

```
overload pow;
int pow (int, int);
```

```
double pow (double, double);
c=pow (2, 3);
d=pow (1.5, 2.1);
```

4. КЛАССЫ ПАМЯТИ

При объявлении переменной ей назначается определенный класс памяти. Он характеризует время существования объема памяти, место хранения переменных и область видимости.

Область видимости – это та часть программы, в которой данная переменная может быть использована.

Место хранения – ОП или регистры центрального процессора.

Время существования: глобальное или локальное.

Существует 4 класса памяти объектов:

1) автоматический. Назначается ключевым словом auto.

2) регистровый. Назначается ключевым словом register.

3) статический. Назначается ключевым словом static.

4) внешний. Назначается ключевым словом extern.

Этот спецификатор указывается перед типом данных.

По умолчанию внешними считаются объекты, описанные вне функций и автоматическими – в пределах функций.

Автоматические переменные

Формат описания: auto int i;

Автоматическая переменная описана внутри функции, является локальной, т. е. память под нее выделяется при входе в функцию и освобождается при выходе из нее. Переменная данного класса автоматически не инициализируется. Она хранится в ОП и может быть использована только внутри той функции или блока, где описана.

Внешние переменные

Внешние переменные описаны вне функции и доступны из всех модулей программы, даже если программа хранится в нескольких файлах. Описание внешней переменной идет без слова extern. Слово extern ставится внутри функции. Внутри функции можно явно указать, что вы работаете с внешней переменной. Для этого повторяется ее описание с ключевым словом extern. Инициализировать переменную можно только в главном описании, а в объявлениях, которые начинаются с extern, инициализация невозможна. Внешняя переменная хранится в ОП, является глобальной, т. е. существует, пока работает программа. Рассмотрим примеры описаний переменных:

1.
```
int h;
main ( )
{extern int h;
 . . .
}
void magic ( )
```

```
{ extern int h;
 . . .
}
```

В этом примере описана одна внешняя переменная h, которую мы будем использовать в функциях main и magic, и явно это показываем строчками extern int h.

```
2. int h;
   main ( )
   { extern int h;
    . . .
    }
    void magic ( )
    { . . .
     }
```

Описана одна внешняя переменная h, которая доступна и main и magic. В magic нет явного описания h, но использовать ее можно.

```
3. int h;
  main ( )
  { int h; // auto
    . . .
   }
   void magic ( )
   { auto int h;
     . . .
    }
```

Здесь описаны 3 разные переменные с именем h. Одна внешняя переменная, локальная переменная в main и локальная переменная в magic. В этом случае к внешней переменной h из этих функций обратиться просто по имени нельзя.

Статические переменные

Спецификатор static используется для объявления переменных на внешнем уровне. Статическая переменная хранится в ОП и существует в течение всего времени выполнения программы, но доступна только в пределах того модуля, функции, блока, в котором описана. Если переменная описана вне функции, то она доступна всем функциям файла. Если переменная описана внутри функции или блока, то она может изменяться только данной функцией, хотя компилятор запоминает ее значение от одного вызова до другого. Статические объекты можно инициализировать. Инициализация происходит только один раз при первом вызове функции или блока, в котором она описана. Например:

```
int func1 ( )
{ static int a=0;
  . . .
}
```

Регистровые переменные

Регистровые переменные хранятся в регистрах центрального процессора. Доступ и работа с ними идут быстрее. В остальном, это те же самые автоматические переменные. Регистровыми можно описать переменные типа int и char, а также указатели размера int. К регистровым переменным нельзя применить операцию взятия адреса. Если нет свободного регистра, то компилятор их сам переименовывает в автоматические.

Таблица 5

Классы памяти переменных

Класс памяти	Ключевое слово	Продолжительность действия	Область действия
автоматический	auto	временная	локальная
регистровый	register	временная	локальная
внешний	extern	постоянная	глобальная (все файлы)
статический	static	постоянная	локальная
внешний статический	static	постоянная	глобальная (1 файл)

Функции имеют классы памяти static и extern, так как они описаны на внешнем уровне. По умолчанию функции считаются extern. Если функцию описать как static, то она будет доступна только в текущем файле программы. Если массивы описаны как static или extern, то они автоматически инициализируются нулями.

5. ФАЙЛЫ

Для использования файла в программе необходимо описать файловую переменную. Описание файловой переменной:

FILE * имя переменной;

Например:

FILE *fp;

FILE – это ключевое слово, описание которого хранится в библиотечном файле stdio.h.

Для открытия файла обычно используется следующая функция:

fopen(строка, режим).

Аргумент *строка* задает имя открываемого файла, например, “a:\lab\lab6.dat”.

Таблица 6

Режим доступа к файлу

Символ	Тип доступа
“r”	файл открыт для чтения
“w”	файл открыт для записи. Если файла не существует, то он создается, если файл уже существует, то его содержимое теряется
“a”	Файл открывается для записи в конец файла. Если файл не существует, то он создается
“r+”	Поток открывается для чтения и записи

Окончание таблицы 6

Символ	Тип доступа
“w+”	Открывается пустой файл для чтения и записи, если файл существует, то его содержимое теряется
“a+”	Файл открывается для чтения и записи в конец файла. Если файл не существует, то он создается

Пример:

```
FILE *fi;
if ((fi=fopen(“a:\lab.dat”, “r”)) = = NULL)
{printf (“Ошибка открытия файла”);    exit(1);}
else printf (“Файл открыт”);
```

Функция fopen возвращает указатель на файл или NULL при ошибке (если файл не открылся).

С началом выполнения программы открываются три стандартных файла: стандартный ввод (stdin), вывод (stdout) и стандартный вывод сообщения об ошибках (stderr). Эти имена можно использовать в функциях ввода, вывода вместо имени файла.

Функция freopen (“file.dat”, “w”, stdout) переназначает указатель одного потока на другой.

Таблица 7

Функции чтения и записи в файл

Объект операции	Чтение		Запись	
	из файла stdin	из любого файла	в файле stdout	в любом файле
Последовательность байтов	—	fread	—	fwrite
Отдельный символ	getc, getchar, getch, getche	fgetc, fgetchar	putc, putchar, ungetc	fputc, fputchar
Работа с целыми числами (int)	—	getw	—	putw
Строки	gets	fgets	puts	fputs
Для форматизованных данных	scanf	fscanf	printf	fprintf

Работа с указателями потока

При выполнении операций чтения и записи указатель потока можно изменять (передвигать, перемещать). Начальная установка указателя зависит от режима открытия файла (rw – на начало, a – на конец).

Существует пять функций для установки указателя потока:

1. ftell (указ. файла) – получает текущую позицию указателя потока.

Пример:

```
FILE *fi;
printf (“%ld”, ftell(fi));
```

2. fgetpos (указ. файла, переменная) – записывает в переменную текущую позицию файла.

Пример: long pos;

fgetpos (fi, &pos);

3. fsetpos (указ. файла, переменная) – устанавливает указатель файла в позицию, задаваемую переменной.

Пример: fgetpos (fi, *pos);

4. fseek (указ. файла, кол-во байт, позиция) – устанавливает указатель файла в позицию, отстоящую на количество байт от места, определяемого параметром позиции. В качестве позиций можно задать одно из следующих значений:

SEEK_SET – начальная позиция;

SEEK_CUR – текущая позиция;

SEEK_END – конец потока.

Количество байт можно указывать с минусом:

fseek (fi, -n, seek_cur)

long n;

5. rewind (указ. файла) – устанавливает указ. файла на начало потока.

Для закрытия файла используется функция fclose (указатель файла);

Например:

fclose (fi);

Функция fcloseall ( ) – закрывает все открытые потоки.

6. УПРАВЛЕНИЕ ОПЕРАТИВНОЙ ПАМЯТЬЮ (ОП)

6.1 Функции для работы с ОП

Управление оперативной памятью реализуется посредством библиотечных функций языка Си, содержащихся в файле <alloc.h>: malloc, calloc, free. Функции malloc и calloc осуществляют выделение памяти, а функция free — ее освобождение.

Функция malloc(unsigned size) возвращает в качестве своего значения указатель на область памяти. Размер этой области в байтах определяется с помощью аргумента size. Если выделение объема памяти невозможно, то результатом функции будет пустой указатель.

Функция calloc(unsigned count, size) выделяет обнуленную область памяти, в которой можно разместить count объектов размером size каждый. Результат функции — указатель на выделенную область памяти.

Функция free(ptr) освобождает область памяти, выделенную ранее, определяемую с помощью аргумента ptr.

В языке C++ определена также функция new для динамического выделения памяти. Пример ее использования:

int *ptr;

ptr= new int; //выделение памяти размером int

Для освобождения памяти используется функция delete.

```
delete ptr; // очистка выделенной памяти
```

Пример:

```
typedef struct fio
{. . .
}
fio *ptr;
ptr=(fio*) malloc(sizeof(fio));
if (ptr= =NULL) printf (“Нет памяти!”);
```

Функция sizeof определяет размер памяти в байтах для аргумента fio.

(fio*) – операция приведения к типу.

6.2 Модели памяти

Модель памяти указывает, как программа использует память. Вся память разбита на сегменты. Каждый сегмент – 64К.

Существует 6 моделей памяти:

1. Tiny. Выбирается, когда код программы и данные вместе не превышают одного сегмента.

2. Small. Под код программы выделяется один сегмент и под данные – один сегмент.

3. Medium (средняя). Под данные выделяется один сегмент, а код программы может быть до одного Мб.

4. Compact (компактная). Один сегмент для кода программы, данные до одного Мб.

5. Large (большая). И данные, и код программы могут быть более одного сегмента, но размер отдельного объекта должен быть меньше одного сегмента.

6. Huge (огромная). Аналогична large, но ограничение на размер отдельного элемента снято.

Вид модели памяти устанавливается в Option / Compiler / Model.

При описании указателей можно использовать модификаторы.

Near, far, huge – 3 модификатора. Эти же модификаторы можно использовать при описании функций. По принципу умолчания считается, что модификатор near стоит в моделях 1, 2, 4 – для функций, а для данных в 1, 2, 3. Модификатор far стоит в моделях 3, 5, 6 – для функций, а для данных в 4, 5, 6. Выделение памяти происходит за пределами выделяемого сегмента.

6.3 Динамические списки

Статические переменные – это переменные, размер которых известен заранее. Они описываются в программе, и компилятор выделяет под них память.

Динамические переменные – это переменные, структура которых заранее известна, т.е. их состав. Память под них выделяется в процессе выполнения программы. Примером динамических данных являются списки.

Список — это множество элементов, каждый из которых состоит, как минимум, из двух полей. Одно поле содержит либо саму информацию, либо ссылку на нее. Другое поле содержит ссылку на следующий элемент списка. Элемент списка называют «звено» списка. Таким образом, список — это цепочка связанных звеньев от первого до последнего. Последнее звено не ссылается на следующий элемент, поэтому поле ссылки имеет значение «пустой указатель». По однонаправленному списку можно двигаться только в одном направлении — от заглавного (первого) звена к последнему.

Двунаправленный (двусвязный) список — это множество элементов, каждый из которых имеет два поля с указателями; одно поле содержит ссылку на следующий элемент, другое поле — ссылку на предыдущий элемент и информационное поле.

Наличие ссылок как на следующее звено, так и на предыдущее позволяет от каждого звена двигаться по списку в любом направлении.

Список, первое звено которого имеет ссылку на последнее, а последнее на первое, называется кольцевой. Кольцевой список имеет заглавное звено. Заглавное звено, как и в случае однонаправленного списка, позволяет обрабатывать первое и последнее звенья в общем цикле. Однако в таком кольцевом списке надо каждый раз проверять, не является ли очередное звено заглавным.

Характерной особенностью динамических данных является невозможность установить заранее фиксированный объем памяти. Выделение памяти под отдельное звено списка происходит в тот момент, когда она появляется во время выполнения программы, а не во время трансляции.

Пример**:** программа, создающая линейный однонаправленный список из фамилий студентов.

```
#include<alloc.h>
#include<stdio.h>
void main()
{ typedef  struct man { char name[20];
                man *next; } man;
man *first,   *cur;
int  n;
printf("введите  кол-во  имен  ");
scanf("%d",&n);
first=(man   *)malloc(sizeof(man));
cur=first;
for(int  i=0;i<n;i++)
{ if (i) { (*cur).next=(man *)malloc(sizeof(man));
         cur=(*cur).next; }
  printf("введите имя        ");
  scanf("%s",(*cur).name);
  (*cur).next=NULL;
 }
```

```
/* просмотр и вывод */
cur=first;
while (cur!=NULL)
 {printf("Это %s\n",(*cur).name);
  cur=(*cur).next; }
}
```

Данная программа вначале запрашивает количество имен в списке, а затем формирует его, заполняя информационное поле.

Наиболее рациональной является программа, где формирование списка происходит в зависимости от ответа на запрос о прекращении ввода информационных полей.

Пример двусвязного списка — список делегатов от городов. Количество городов и делегатов от каждого города будет произвольным.

```
typedef struct deputat
{char *name;
  deputat *next;
};
typedef struct town
{char *name town;
town *next town;
deputat *spisok;
};
```

7. ОБРАБОТКА МНОГОМОДУЛЬНЫХ ПРОГРАММ

Если текст программы содержится в нескольких модулях (файлах), то для их совместной обработки можно определить проект. Для этого создается файл, содержащий описание проекта. В главном меню есть опция Project; команда Open Project (открыть проект) создает файл с расширением prj. Для включения файлов в проект используется команда Add Item, а для удаления файла из проекта – Delete Item. В конце работы проект надо закрыть командой Close Project.

Например, текст программы хранится в двух файлах: first.cpp и second.cpp. Создаем проект с именем program.prj. С помощью команды Add Item включаем в проект имена необходимых файлов. Для запуска программы будет использоваться файл program.exe.

8. ОТЛАДКА И ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ

Выполнение программы будет осуществляться только в случае удачной компиляции. В процессе компиляции системой могут быть сделаны предупреждения (Warning) и сообщения об ошибках (Error). При просмотре данных сообщений курсор устанавливается в строку с ошибкой. В процессе отладки следует обращать внимание на предупреждения, стремясь устранить

причину их появления. Игнорируя их, можно снизить надежность и эффективность программы.

После устранения синтаксических ошибок программа тестируется на наличие логических ошибок. Для этого можно использовать следующие инструменты отладчика:

− Пошаговое выполнение программы с помощью клавиши F7 (осуществляет вход в вызываемые функции);

− Пошаговое выполнение программы с помощью клавиши F8 (без входа в вызываемые функции);

− Использование окна наблюдений значений переменных (Watch);

− Использование окна оценки и модификации значений переменной (evaluate/modify). Вызов этого окна возможен комбинацией клавиш ctrl+F4;

− Выполнение программы до того места, в котором находится курсор (F4);

− Расстановка в программе точек прерывания (ctrl+F8).

Даже после того, как программа отлажена и работает, она может столкнуться с определенными проблемами. Это исключительные ситуации. Существует много причин их появления: не хватает памяти для размещения переменных, не открывается входной или выходной файл и т. д. Достаточно часто программа при этом «вылетает», или возникают ошибки общей защиты.

Поэтому для написания надежных программ рекомендуется выполнять в программе следующие действия:

− Проверять удачное завершение операций выделения памяти;

− Проверять допустимый диапазон вводимых значений;

− Проверять удачное открытие файлов.

9. РАБОТА С ВИДЕОПАМЯТЬЮ

9.1 Оперативная память. Структура адресного пространства.

Одним из основных элементов компьютера, позволяющим ему нормально функционировать, является память. Внутренняя память компьютера – это место хранения информации, с которой он работает. Внутренняя память компьютера является временным рабочим пространством; в отличие от нее внешняя память, такая как файл на дискете, предназначена для долговременного хранения информации. Информация во внутренней памяти не сохраняется при выключении питания.

Память компьютера организована в виде множества ячеек, в которых могут храниться значения; каждая ячейка обозначается адресом. Размеры этих ячеек и, собственно, типы значений, которые могут в них храниться, отличаются у разных компьютеров.

Большинство современных компьютеров, и, в том числе, все персональные компьютеры, используют размер ячейки состоящей из 8 бит или

«байта». Байт позволяет хранить код одной буквы алфавита или одного символа.

Так как IBM/PC использует ячейки памяти длиной восемь бит или один байт, в памяти могут храниться значения, которые можно выразить восемью битами. Это значения до двух в восьмой степени или 256.

Для удобства манипулирования символьными данными компьютеру необходимо, чтобы коды символов преобразовывались в байтовые величины. Большинство компьютеров, включая IBM/PC, используют код ASCII, американский стандартный код для обмена информации.

В коде ASCII числовые значения присваиваются всем обычно используемым символам, таким как буквы алфавита, строчные и заглавные, цифры, знаки пунктуации. Несколько кодов зарезервированы для управления, например, чтобы указать конец строки символов.

Таблицы стандартных кодов ASCII и расширенных кодов ASCII для IBM/PC можно найти во многих справочниках.

Каждая ячейка памяти имеет адрес, который используется для ее нахождения. Адреса – это числа, начиная с нуля для первой ячейки, увеличивающиеся по направлению к последней ячейке памяти. Поскольку адреса – это те же числа, компьютер может использовать арифметические операции для вычисления адресов памяти.

Архитектура каждого компьютера накладывает собственные ограничения на величину адресов. Наибольший возможный адрес определяет объем адресного пространства компьютера или то, какой объем памяти он может использовать. Адрес всегда хранится в двух двухбайтовых словах, называемых адресом сегмента и смещения.

Сегмент – это участок памяти, имеющий длину 64 кБ и начинающийся с физических адресов (0, 16 ,32 ,48, . .). Смещение указывает, сколько байт от начала сегмента надо пропустить, чтобы добраться до нужного адреса. Фрагмент памяти в 16 байт называется параграфом. Таким образом, сегменты могут пересекаться.

Вычисление абсолютного адреса происходит на основе адреса сегмента и адреса смещения.

Память компьютера используется для различных целей – часть ее занимает программа, другая часть используется для хранения данных, с которыми в данный момент работает программа. Помимо памяти, для временного хранения данных микропроцессор использует еще и регистры, что существенно ускоряет работу.

Микропроцессор имеет четыре шестнадцатиразрядных регистра общего назначения, называемых AX, BX, CX и DX. Каждый из них может быть разделен на два восьмиразрядных регистра, указанием старшей (H-high) или младшей (L-low) части полного (X) регистра. Таким образом, восьмиразрядные регистры называются AH, AL,BH, BL, CH, CL, DH и DL.

Некоторые из ячеек памяти, находящиеся в области памяти с адресами от 400 до 500, содержат т.н. глобальные переменные DOS. Информацию из этих ячеек можно получить, используя непосредственное обращение к ним.

Адрес видеопамяти в текстовом режиме для цветного графического дисплея равен B800:0000. Размер видеопамяти:80*25*2=4000 байт. Одна строка дисплея описывается 2*80=160 байтами видеопамяти.

Каждый символ экрана занимает 2 байта видеопамяти: первый байт хранит значение символа, второй – его атрибут (цвет фона, на котором изображен символ и цвет самого символа). Значение байта-атрибута удобно задавать шестнадцатеричным числом.

Формирование байта-атрибута происходит по следующему правилу:

Мерцание	Цвет фона			Цвет символа			
7	6	5	4	3	2	1	0

Например, для вывода синих символов на белом фоне без мерцания можно сформировать следующее значение байта-атрибута 0x72h (0x – признак шестнадцатеричного числа).

9.2 Программирование прямого обращения к ОП

Пример: сформировать в переменной vid_mem начальный адрес области видеопамяти.

```
char far *vid_mem;
vid_mem = (char far *) 0xB8000000;
```

Пример: организовать перемещение на экране курсора, используя глобальные ячейки DOS (адреса ячеек хранящих вертикальную и горизонтальную координаты курсора, даны в примере).

```
#include <conio.h>
char far *p_x=(char far *) 0x00000450;
char far *p_y=(char far *) 0x00000451;
int  i;

void main(void)
{
 clrscr();
 *p_x=5;
 *p_y=5;
 for( i=1;i<=10;i++)
 {
   cputs("привет");
   *p_y=*p_y+1;
   *p_x=i+5;
 }
 getch();
}
```

Пример: используя прямое обращение к видеопамяти, вывести на экран произвольную горизонтальную строку.

```
#include <conio.h>
char far *vid_mem=(char far*)0xB8000000;
int i;
int x=10,y=10,attr=33;
char *str="строка1\0";
char far*v;

void main()
{
  clrscr();
  v=vid_mem;
  v+=(x*160)+y*2;
  for(i=0;*(str+i)!='\0';i++)
   {
     *v++=*(str+i);
     *v++=attr;
   }
}
```

10. ПРЕРЫВАНИЯ

10.1 Понятие прерывания. Типы прерываний

Компьютер должен обладать способностью реагировать на события, происходящие вне его микропроцессора, например, воспринимать информацию, вводимую с клавиатуры. Существует два способа организации такой реакции. Один способ состоит в постоянном ожидании события. Такой способ называется «сканированием» или «опросом», и такой опрос может занимать большую часть времени компьютера. Другой способ позволяет компьютеру спокойно выполнять свою работу, пока не произойдет событие, требующее его внимания. Такой подход называется использованием «прерываний». Использование прерываний позволяет наиболее эффективно организовать работу компьютера, поскольку время центрального процессора не расходуется вхолостую на ожидание.

Прерывание – это кратковременное приостановка текущей процедуры программы, позволяющая выполнить другую процедуру. После завершения прерывания прерванная программа продолжает выполняться так, как будто бы ничего не происходило. Эти две процедуры могут быть несвязанными – и прерывание не окажет никакого воздействия на прерванную процедуру. Они могут быть взаимозависимы – прерванная программа может быть

модифицирована процедурой обработки прерывания. Прерывание может быть вызвано внешним по отношению к выполняемой программе событием или в результате действий самой программы. Прерывание может быть вызвано аппаратно или командой из программы.

Механизм прерывания работает следующим образом: каждому из основных типов прерываний присвоен свой номер. Например, прерывание таймера имеет номер 8, гибкие диски, используют номер 14. В самом начале оперативной памяти IBM/PC хранится таблица с адресами программ, которые должны вызываться при возникновении различных прерываний. Эти адреса иногда называются векторами прерываний. Прерывание с номером 0 имеет вектор, хранящийся в ячейке с нулевым адресом, прерывание 1 имеет свой вектор в ячейке 4 и так далее. Когда происходит прерывание номер «X», вектор, хранящийся по адресу 4*X, загружается в регистры адреса программы, т. е., регистры CS и IP, и компьютер начинает выполнять программу обслуживания прерывания, которая размещается по этому адресу.

Когда обработка прерывания заканчивается, программа обработки возвращает управление программе, которая выполнялась в момент возникновения прерывания, с помощью специальной команды IRET или «возврат из прерывания». Чтобы такой возврат мог быть выполнен, необходимо сохранить в стеке текущие адреса программы до загрузки в регистры CS и IP вектора прерывания.

В компьютере PC имеется 256 различных прерываний, с номерами от 0 до 0xff. Для хранения их адресов зарезервирована память с адресами от 0 до 0x400.

Некоторые из прерываний определены для использования процессором. Например, прерывание 0 возникает при делении на 0. Другие определены для вызова функций BIOS, третьи – для использования ДОС.

Иногда бывает необходимо, чтобы работа процессора не прерывалась, например, при выполнении какой-либо критической операции. Для этого у микропроцессора имеется специальная команда, которая позволяет отложить обслуживание прерываний, запоминая их, и парная ей команда, восстанавливающая нормальный режим обслуживания прерываний. Когда прерывания запрещаются, запрос прерываний не теряется, он запоминается и будет обслуживаться, как только будут разрешены прерывания.

Обычно прерывания не запрещаются на сколько-нибудь продолжительное время. Прерывания допустимо запрещать лишь на очень короткие промежутки времени, необходимые для выполнения некоторых внутренних операций процессора, состоящих из небольшого числа команд. Типичным примером таких операций, которые не могут быть прерваны на полпути, может служить загрузка нового набора значений в регистры сегментов. Поскольку эти регистры необходимы для правильной работы любой программы, нарушение согласованности загрузки в них значений может привести к полной неразберихе, поэтому необходимо запретить прерывания на время загрузки в них новых адресов.

Существуют три типа прерываний, которые получили названия аппаратных, логических и программных. Между ними нет принципиальной разницы, но использование разделяет их на три отдельных категории.

Аппаратные прерывания вырабатываются устройствами, требующими внимания процессора. В IBM/PC таких прерываний несколько. Во-первых, имеется так называемое немаскируемое прерывание, используемое для сообщения об отказе питания, оно имеет номер 2. Далее прерывание 8 используется таймером, номер 9 – клавиатурой и 14 – контролером гибких дисков.

Имеется также семь зарезервированных номеров прерываний, 6, 7, с 10 по 13 и 15, которые могут быть использованы в дальнейшем, если возникнет необходимость в дополнительных аппаратных прерываниях. Два из этих семи прерываний уже нашли свое назначение, прерывание 12 зарезервировано для адаптера связи, а прерывание 15 – для интерфейса устройства печати.

Логические прерывания формируются самим процессором, когда он встречает какое-либо необычное условие. Таких прерываний предусмотрено четыре. Прерывание 0 возникает при попытке деления на ноль. Прерывание 1 используется для управления пошаговым режимом работы микропроцессора, при котором команды выполняются по одной. Это прерывание выставляется отладчиками для пошагового выполнения программ. Прерывание 3 вырабатывается командой установки «контрольных точек», которая также используется при отладке. Прерывание 4 формируется при возникновении условия переполнения, например, если результат арифметической операции не помещается в регистр. Таким образом, четыре логических прерывания распадаются на две пары: одна для арифметических операций (деление на ноль и переполнение) и вторая для отладки программ (шаговый режим и контрольные точки).

Программы прерывания вызываются как процедуры другими программами. Для вызова процедуры программа должна знать ее адрес, а вызываемая процедура может не знать адреса вызывающей программы, поскольку механизм вызова автоматически генерирует адрес возврата, который будет использован вызываемой программой после завершения ее выполнения.

Программные прерывания обеспечивают такую возможность путем выработки прерывания самой программой. Например, если программе необходимо вычислить время дня, ей совершенно не требуется знать адрес программы подсчета времени – достаточно знать только, что программа подсчета времени дня запускается программным прерыванием 26.

Программные прерывания используются для вызова всех служебных функций, представляемых обычным пользователям. Эти функции включают все процедуры системы BIOS и ПЗУ и служебные процедуры ДОС.

10.2 Прерывания системы ROM-BIOS.

BIOS (Basic Input/Output System - базовая система ввода/вывода) расположена в ROM (read-only memory – постоянное запоминающее устройство

– ПЗУ) и частично в файле, который загружается при загрузке компьютера (загружаемый BIOS). Доступными для пользователя является область памяти ROM_BIOS.

Существует всего 12 прерываний ROM-BIOS, распадающихся на 5 групп: шесть из двенадцати прерываний обслуживают определенные периферийные устройства, два дают отчет об оборудовании компьютера, одно работает с часами и календарем, одно выполняет операции по печати экрана и два прерывания переводят компьютер в совершенно иное состояние, запуская ROM-BIOS и системную программу начального запуска. Как мы в дальнейшем увидим, большинство прерываний относятся к группе служебных подфункций, которые выполняют большую часть работы. Например, прерывание 16 (16-ричное 10), связанное с выдачей изображения, имеет 16 подфункций, выполняющих все от установки режима изображения до изменения размеров курсора. Подфункция вызывается с помощью обращения к прерыванию, управляющему ею, и задания в регистре AH номер подфункции.

Как правило, если подпрограмма-прерывание возвращает какой-либо простой результат, то этот результат остается в регистре AX; это применимо как к BIOS, так и к языкам программирования.

Обработка изображений реализована прерыванием с номером 16 или 10h. Это многофункциональное прерывание. Каждая функция этого прерывания имеет свой номер, по которому она может быть вызвана. Номер функции при вызове прерывания заносится в регистр AH.

Таблица 8

Служебные функции выдачи изображения

Номер функции		Описание
10 с.с.	16 с.с.	
0	0	Установить режим выдачи изображения
1	1	Установить размер курсора
2	2	Установить позицию курсора
3	3	Считать позицию курсора
4	4	Считать позицию светового пера
5	5	Установить активную страницу дисплея
6	6	Прокрутить окно вверх
7	7	Прокрутить окно вниз
8	8	Считать символ и атрибут
9	9	Записать символ и атрибут
10	A	Записать символ
11	B	Установить цветовую палитру
12	C	Записать точку пикселя
13	D	Прочитать точку пикселя
14	E	Записать символ в режиме телетайпа
15	F	Получить текущий режим выдачи изображения
19	13	Записать строку символов

Таблица 9

Регистры, используемые для установки позиции курсора с помощью служебной функции 2

Номер функции	Параметры
AH = 2	DH = номер строки
	DL = номер колонки
	BH = номер страницы (для графических
	режимов устанавливается в 0)

Таблица 10

Регистры, используемые для чтения положения курсора с помощью служебной функции 3

Номер функции	Параметры
AH = 3	DH = номер строки
	DL = номер колонки
	BH = номер страницы (для графических
	режимов устанавливается в 0)
	CH = начальная строка растра для курсора
	CL = конечная строка растра для курсора

Таблица 11

Регистры, используемые для записи символа и атрибута текста с помощью служебной функции 9

Номер функции	Параметры
AH = 9	AL = записываемый на экран символ ASCII
	BL = записываемый на экран атрибут символа
	BH = номер активной страницы дисплея
	(в графических режимах не нужен)
	CX = число выводов символа с атрибутом

Таблица 12

Регистры, используемые для записи символа с помощью служебной функции 10

Номер функции	Параметры
AH = 10	AL = записываемый на экран символ ASCII
	BL = атрибут цвета для графических режимов
	BH = номер активной страницы дисплея
	(в графических режимах не нужен)
	CX = число записей символа

10.3 Использование прерываний BIOS для работы с клавиатурой

Функции для работы с клавиатурой вызываются с помощью прерывания 22(16). Этих функций всего три; они имеют номера от 0 до 2. Как и для всех других функций ROM-BIOS, при вызове функции номер для работы с клавиатурой задается в регистре AH.

Процедура 0 возвращает очередной набранный на клавиатуре символ. Если символ уже находится в буфере ROM-BIOS, то он возвращается немедленно. В противном случае процедура ожидает, пока он не будет набран. Каждый символ клавиатуры возвращается в виде пары байтов, называемых главным и вспомогательным байтами. Главный байт, возвращаемый в AL, равен либо 0 для специальных символов, соответствующих, например, функциональным клавишам, либо коду ASCII для обычных ASCII-символов. Вспомогательный байт, возвращаемый в AH, представляет собой либо идентификатор специального символа, либо скэн-код стандартной клавиатуры PC для ASCII-символов.

Таблица 13

Три служебных функции ROM-BIOS доступа к клавиатуре

Функция	Описание
0	Прочитать с клавиатуры следующий символ
1	Получить ответ о готовности символа
2	Получить состояние клавиши переключения регистров

Если при вызове процедуры 0 в буфере клавиатуры нет ни одного символа, то процедура ждет, пока он не появится, что, естественно, приостанавливает программу. Следующая рассматриваемая нами процедура, позволяет программе проверять ввод с клавиатуры, не приостанавливая своего исполнения. В противоположность тому, что излагается в некоторых версиях технического руководства IBM, процедуры 0 и 1 можно использовать для ввода как обычных ASCII-символов, так и специальных символов, отвечающих, например, функциональным клавишам.

10.4 Программные средства для обращения к прерываниям

Для описания регистров, используемых при обращении к прерываниям, в пользовательской TC-программе в библиотеке DOS.H создан специальный тип REGS.

```
struct WORDREGS
{ unsigned int  ax,bx,cx,dx,si,di,cslag,flags;
 }
struct BYTEREGS
 { unsingred char al,ah,bl,bh,cl,ch,dl,dh;
 }
union REGS{
```

```
struct WORDREGS x;
struct BYTEREGS h;
}
```

В этом случае в пользовательских программах для применения прерываний нужно описать переменную с указанным типом данных.

Описатель UNION означает наложение памяти при хранении переменных, перечисленных в различных списках шаблона. Это наложение можно изобразить следующей схемой:

0	AX		BX	
1	AL	AH	BL	BH

Таким образом, каждый байт двухбайтного регистра, например AX, имеет свое имя и доступен для самостоятельного обращения. Пример обращения к полям регистровой переменной r:

```
r.x.ax=5;
r.h.ah=2;
```

Замечание. Язык Си позволяет обращение к регистрам без предварительного описания соответствующих переменных. Имена таких регистров необходимо записывать большими буквами и предварять символом подчеркивания.

Например:

```
_AX=5;
_Ah=2;
```

Для обращения к прерыванию в TC (библиотека Dos.h) существует несколько функций. Рассмотрим их синтаксис.

1) void geninterrupt(int intr_num)

Макрокоманда geninterrupt вызывает программное прерывание, номер которого задан параметром intr_num. Состояние всех регистров после вызова прерывания зависит от конкретного прерывания.

Пример: Вывод символа * в позицию 80,25.

```
#include <conio.h>
#include <dos.h>

void writechar(char ch);  /* прототип функции */

int main(void)
{
clrscr();
gotoxy(80,25);
```

```
writechar('*');
getch();
return 0;
}

/*  выводится символ в текущую позицию курсора          */
/*  с использованием видео BIOS для того, чтобы избежать*/
/*  прокрутки экрана при выводе в позицию (80,25)       */

void writechar(char ch)
{
struct text_info ti; /* шаблон text_info описан в conio.h*/
gettextinfo(&ti); /* захватить текущие установки текста */
_AH = 9;          /* прерывание 0x10, подфункция 9      */
_AL = ch;         /* выводимый символ                   */
_BH = 0;          /* видео-страница                     */
_BL = ti.attribute;/* видео-атрибут                     */
_CX = 1;          /* коэффициент повторения             */
geninterrupt(0x10);/* вывод символа                     */
}
```

2) int int86(int intno,union REGS *inregs,
union REGS *outregs);

Функция int86 вызывает программное прерывание процессора 8086, номер прерывания указан в аргументе intno. Перед выполнением программного прерывания функция копирует содержимое регистров из inregs в сами регистры. После возврата из прерывания функция копирует текущие значения регистров в outregs. Если установлен флаг переноса, то это значит, что произошла ошибка. Отметим, что inregs может указывать на ту же структуру, что и outregs. Функция int86 возвращает значение регистра AX после завершения программного прерывания. Если флаг переноса установлен (outregs -> x.cflag != 0), то есть произошла ошибка, то данная функция присваивает глобальной переменной _doserrno код ошибки.

Пример: Вывод слова «Привет».

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#define VIDEO 0x10

void movetoxy(int x, int y)
{
union REGS regs;
```

```
regs.h.ah = 2;  /* устанавливает позицию курсора */
regs.h.dh = y;
regs.h.dl = x;
regs.h.bh = 0;  /* видео страница 0 */
int86(VIDEO, &regs, &regs);
}

int main(void)
{
clrscr();
movetoxy(35, 10);
printf("Привет\n");
return 0;
}
```

Рассмотрим еще несколько примеров вызова прерываний.
Пример: Установить курсор в положение (14,1) на экране дисплея.

```
#include <dos.h>
void main(void)
{
int x,y;
union REGS r;
r.h.ah=2;
r.h.dl=1;
r.h.dh=14;
r.h.bh=0;
int86 (0x10,&r,&r);
}
```

Пример: Прочитать символ с экрана, расположенный под курсором. Алгоритм реализован с помощью пользовательской функции.

```
void readchar(x,attr)
char *x,*attr;
{ union REGS r;
r.h.ah=8;
r.h.bh=0;
int86 (0x10,&r,&r);
*attr=r.h.ah;
*x=r.h.al;
}
```

Пример: Вывести символ 'a' на экран на место, указанное курсором. Алгоритм реализован с помощью пользовательской функции.

```
writechar(x,attr)
int x, attr;
{ union REGS r;
r.h.ah=9;
r.h.bh=0;
r.h.dl=x;
r.h.bl=attr;
r.x.cx=1;
int86 (0x10,&r,&r);
}
```

Пример: Очистить экран.

```
#include <dos.h>
void main(void)
{
r.h.ah=6;
r.h.al=0;
r.h.bh=0x7;
r.h.cx=0;
r.h.cl=0;
r.h.dh=24;
r.h.dl=79;
int86 (ox10,&r,&r);
}
```

Пример: Составить функцию вывода строки в центре экрана красными буквами на зеленом фоне с мерцанием, вертикально.

```
main()
{ char buffer[]="строка1\0";
int x,y,attr;
char *buf=buffer;
attr=4+16*2+127;
x=10;y=10;
while (*buf!='\0')
    { goto_xy(x,y);
     writechar(*buf,attr);
     buf++;
     y++;
     }
}
```

Пример: Ввести символ с клавиатуры.

Фрагмент программы с использованием функции getch() из библиотеки Conio.h.

```
char c;
c=getch();
if(c==0) getch();
.....
```

Фрагмент этой же программы с использованием прерывания 16h

```
char c;
union REGS r;
r.h.ah:=1;
intr(0x16,&r,&r);
if (r.h.al==0)
  c=r.h.ah;
else c=r.h.al;
```

11. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ (ООП)

В стиле процедурного программирования данные и функции существуют раздельно. В объектно-ориентированном программировании (ООП) данные и функции для работы с ними объединены в один блок.

Три основных принципа ООП:

1. Инкапсуляция – это объединение данных и функций для работы с ними в единый объект. Это позволит в максимальной степени изолировать объект от внешнего окружения

2. Наследование – это добавление к уже существующему объекту новых возможностей и тем самым создание нового объекта.

3. Полиморфизм – это свойство родственных объектов (объектов, классы которых являются производными от одного) вести себя по–разному, в зависимости от ситуации, возникающей в момент выполнения программы.

Преимущества ООП:

1) можно повторно использовать ранее созданный код;

2) программы хорошо структурированы;

3) программы легко тестируются;

4) программы при необходимости легко расширяются.

11.1 Определение класса

Класс – это тип, определенный пользователем, содержит данные и функции для работы с ними. Память выделяется только тогда, когда класс используется для создания объекта.

Посмотрим, как можно представить в языке понятие даты, используя для этого тип структуры и набор функций, работающих с переменной этого типа.

struct date (int month, day, year;);// описание структуры

```
date today; // описание переменной
void set_date (date* int, int, int); //описание функций для работы с датой
void next_date (date*);
void print_date (date*);
//…
```

Никакой явной связи между функциями и структурой date нет. Ее можно установить если описать функции как члены структуры:

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

Таким образом, получено описание нового типа данных — класса. Класс также может быть описан с помощью ключевого слова class. Описание класса должно быть до его использования.

Пример описания класса:

```
class date   // это описание нового типа данных, его имя date
{int day, month, year; //  это данные – члены класса
public: void set(int, int, int);  //  функции – члены
void print( );                    //        класса
void get (*int, *int, *int;);
void next ();
};
```

К переменным day, month, year могут обратиться только функции-члены класса. Обращение извне к ним невозможно.

Опишем переменную соответствующего типа (объект):

```
date today;
date my_birthday;
```

Описанные функции-члены класса можно вызвать только через имя объекта, используя стандартную запись обращения к члену структуры:

```
my_birthday.set (30, 12, 1970);
today.set (1,09,2006);
my_birthday.print();
today.next;
```

Можно описать указатель на данный тип и использовать его для обращения к членам класса:

```
data *ptr_d;
ptr_d → print ( );
```

При описании класса могут использоваться метки:

1) private — это закрытые данные и функции, они могут использоваться только функциями-членами данного класса;

2) public — это открытая часть класса, она обеспечивает связь объекта с внешней программой;

3) protected — защищенная часть класса**,** аналогична private, но в отличие от нее может наследоваться производным классом.

К закрытым данным и к закрытым функциям можно обращаться только из функций-членов данного класса. Открытые части класса используются для связи объектов класса с программой, в которой они существуют. По принципу умолчания, при описании класса словом class, его члены по умолчанию считаются закрытыми, если не использованы другие метки. Эти метки можно использовать в описании класса несколько раз и в любом порядке. Также класс можно описать через ключевые слова: struct (структура), union (объединение). Все члены класса, описанного через struct, считаются открытыми, но их можно закрыть, используя метку private. Члены класса, описанного через union, могут быть только открытыми.

Описание функций-членов класса может содержаться в описании класса или вне его. В этом случае при определении функции нужно указать имя класса:

```
void date::next ( ) //описание тела функции вне класса
{
      if (++date>28) {
       // описание действий
      }
}
```

date – имя класса, к которому относится функция;

:: - оператор расширения области действия;

В теле функции имена данных-членов класса можно использовать без указания имени объекта. В таком случае имя относится к тому члену объекта, для которого была вызвана функция.

```
void date::print( ) // печать даты в принятом в России виде
{
      cout <<day<<.<<month<<.<<year;
}
```

cout – оператор вывода в Си++. Выводит список в стандартный поток вывода. Выводит данные или переменные основных типов.

От функций не членов класса date наши данные ограждены:

```
void backdate( )
{
      today.day-- // ошибка, так как такой функции не было в
}                 // описании класса date
```

Есть ряд преимуществ в том, что доступ к структуре данных ограничен явно указанным списком функций. Любая ошибка в дате (например. December, 36, 1985) могла быть внесена только функцией-членом, поэтому первая стадия отладки — локализация ошибки — происходит даже до первого пуска программы.

Это только частный случай общего правила: любое изменение в поведении типа date может и должно вызываться изменениями в его членах. Другое преимущество в том, что потенциальному пользователю класса для работы с ним достаточно знать только определения функций-членов. Защита частых данных основывается только на ограничении использования имен членов класса.

В функции-члене можно непосредственно использовать имена членов того объекта, для которого она была вызвана:

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

При первом вызове readm( )обозначает aa.m, а при втором – bb.m.

Пример программы, которая работает со строками:

```
#include<stdio.h>
#include<string.h>
class WiseString
{const char *s;
  int len;
public:
void assign(const char *string)
{s=string;
len=strlen(s);
}
void print( ) const;
}
void main ( )
{WiseString str1, str2;
str1.assign (“ПМИд-21”);
str2.assign (“4-ый семестр”);
str1.print( );
str2.print( );
}
void WiseString :: print( ) const
{const char *ft=”Это строка\n Ее содержание:\n %s\n Длина %d
символов”;
printf(ft, s, len);
}
```

Инициализация и удаление объектов

Инициализация объектов класса с помощью таких функций, как set_date() – неэлегантное и чревато ошибками решения. Поскольку явно не было указано, что объект требует инициализации, программист может либо забыть это сделать, либо сделать дважды, что может привести к столь же катастрофическим последствиям. Лучше дать программисту возможность описать функцию, явно предназначенную для инициализации объектов. Поскольку такая функция конструирует значение данного типа, она называется конструктором. Эту функцию легко распознать – она имеет то же имя, что и ее класс:

```
Class date {
//…
date (int, int, int);
};
```

Если в классе есть конструктор, все объекты этого класса будут проинициализированы. Если конструктору требуются параметры, их надо указывать:

```
date today = date (23, 6, 1983);
date xmas (25, 12, 0); // краткая форма
date my_birthday; // неправильно, нужен параметр
```

Часто бывает удобно указать несколько способов инициализации объекта. Для этого нужно описать несколько конструкторов:

```
class date {
      int month, day, year;
public:
//…
date (int, int, int); // день, месяц, год
date (int, int); // день, месяц и текущий год
date (int); // день и текущие месяц, год
date(); // стандартное значение: текущие день, месяц и год
date (const char*); // дата в строковом представлении
};
```

Следовательно, вне описания класса необходимо дать определение каждого конструктора date.

Параметры конструкторов подчиняются тем же правилам о типах параметров, что и все остальные функции. Пока конструкторы достаточно различаются по типам своих параметров. Транслятор способен правильно выбрать конструктор:

```
date today (4);
date july (“July 4. 1983”);
date guy (“5Nov”);
date now; // инициализация стандартным значениям
```

Объект класса без конструктора может инициализироваться присваиванием ему другого объекта этого класса. Это не запрещается и в том случае, когда конструкторы описаны:

date d = today; // инициализация присваиванием

Пользовательские типы чаще имеют, чем не имеют конструкторы, которые проводят инициализацию. Для многих типов требуется и обратная операция – деструктор, гарантирующая правильное удаление объектов этого типа. Деструктор класса X обозначается ~X («дополнение конструктора»). В частности, для многих классов используется свободная или динамическая память, выделяемая конструктором и освобождаемая деструктором.

В классе может быть только один деструктор. Он должен располагаться в открытой части класса и не иметь параметров. Деструктор вызывается автоматически при выходе из модуля, который содержит описание класса.

Основная память чаще всего освобождается автоматически при выходе программы из блока, в котором определена переменная типа класса.

Пример программы, работающей со стеком:

```
class stack
{int size;     //     стек состоит из  символов
char* top;    //     указатель стека
public:
stack (int sz)    // конструктор
{top=s=new char [sz];  // выделение памяти и занесение ее в s и top
}
~stack ( ) {delete s;}   // деструктор  удаляет память
};
void f ( )
{stack f (100);   // вызов конструктора для 100 объектов
        …
 }
```

В момент закрытия объекта автоматически вызывается деструктор, в него можно заключить все действия, связанные с завершением работы над объектом (например, очистить память, закрыть все файлы, с которыми работала программа, выдать какое-нибудь сообщение).

Уточнение имени переменной

Использование оператора расширения области видимости (::) для уточнения имени объекта может быть необходимо в следующих случаях:

1. Используется для обращения к глобальной переменной, закрытой локальной переменной.

```
int i=0;  // –  глобальная переменная
int f ( )
{…
int i=0; // –  локальная переменная с тем же именем
i++; // –  увеличиваем на 1 локальную переменную
:: i++; // –  увеличение на 1 глобальной переменной
```

```
   …
}
```

2. Используется для указания явного различия между именем членов класса и прочими именами.

Пример1:

```
class x
{int m;
public:…
void set (int m)
{x:: m=m; // x:: m это уточненное имя,   m это имя аргумента функции
}
};
```

Пример 2:

```
class my_file
{
public:
int open (char*, char*) //определена своя функция open, которая
};                              //    «закрывает» стандартную
int my_file :: open (char*name, char*spec) // при описании своей
{…                              //       функции open для обращения к
if(::open(name, flag))   //       стандартной open используем оператор ::
{…
}
}
```

11.2 Дружественные функции класса

Для связи между функциями вместо значения самой переменной можно передать ее адрес (указатель) или ссылку. Если у нас есть переменная t, то t& означает ссылку на t.

```
int  t;  // описание переменной
int& p t = t;  //описание ссылки
pt =10; //в переменную t записано значение 10
```

Ссылка – это второе имя объекта, то есть когда мы передаем функции ссылку на объект, она получает адрес объекта и тем самым может его модифицировать. Для ссылки не требуется дополнительного пространства в памяти, она является псевдонимом переменной.

Пример передачи данных между функциями (в этих функциях x и y меняются местами):

```
void s1 (int*x, int*y)  //  передача аргументов через указатель
{int z=*y; *y=*x;
*x=z;    }
void s2 (int &x, int &y)  // передача аргументов через ссылки
{int z=y; y=x; x=z; }
```

Воздействуя на формальные параметры внутри функций, мы изменяем значения аргументов, с которыми функция была вызвана.

Предположим, вы определили два класса, vector и matrix (вектор и матрица). Каждый скрывает свое представление и предоставляет полный набор действий для манипуляции объектами его типа. Теперь определим функцию, умножающую матрицу на вектор. Для простоты допустим, что в векторе четыре элемента, которые индексируются 0...3, и что матрица состоит из четырех векторов, индексированных 0...3. Допустим также, что доступ к элементам вектора осуществляется через функцию elem(), которая осуществляет проверку индекса, и что в matrix имеется аналогичная функция. Один подход состоит в определении глобальной функции multiply() (перемножить) примерно следующим образом:

```
vector multiply(matrix& m, vector& v);
{
vector r;
for (int i = 0; i<3; i++) { // реализуется формула r[i] = m[i] * v;
r.elem(i) = 0;
for (int j = 0; j<3; j++)
r.elem(i) += m.elem(i,j) * v.elem(j);
}
return r;
}
```

Это своего рода «естественный» способ, но он очень неэффективен. При каждом обращении к multiply() elem() будет вызываться 4*(1+4*3) раза. Теперь, если мы сделаем multiply() членом класса vector, мы сможем обойтись без проверки индексов при обращении к элементу вектора, а если мы сделаем multiply() членом класса matrix, то мы сможем обойтись без проверки индексов при обращении к элементу матрицы. Однако членом двух классов функция быть не может. Нам нужно средство языка, предоставляющее функции право доступа к закрытой части класса. Функция не член, получившая право доступа к закрытой части класса, называется другом класса (friend). Функция становится другом класса после описания как friend. Например:

```
class matrix;
class vector {
float v[4];
// ...
friend vector multiply(matrix&, vector&);
};
class matrix {
vector v[4];
// ...
friend vector multiply(matrix&, vector&);
};
```

Функция друг не имеет никаких особенностей, помимо права доступа к закрытой части класса. Описание вводит имя дружественной функции в самой внешней области видимости программы и сопоставляется с другими описаниями этого имени. Описание друга может располагаться или в закрытой, или в открытой части описания класса; где именно, значения не имеет. Теперь можно написать функцию умножения, которая использует элементы векторов и матрицы непосредственно:

```
vector multiply(matrix& m, vector& v);
{
vector r;
for (int i = 0; i<3; i++) { // r[i] = m[i] * v;
r.v[i] = 0;
for (int j = 0; j<3; j++)
r.v[i] += m.v[i][j] * v.v[j];
}
return r;
}
```

Функция член одного класса может быть другом другого. Например:

```
class x {
// ...
void f();
};
class y {
// ...
friend void x::f();
};
```

Нет ничего необычного в том, что все функции члены одного класса являются друзьями другого. Для этого есть даже более краткая запись:

```
class x {
friend class y;
// ...
};
```

Такое описание friend делает все функции члены класса y друзьями x.

11.3 Переопределение операторов

Переопределения операторов, заложенных в C++, позволяет программисту в дополнение к арифметическим и логическим операторам и операторам отношения переопределить операторы вызова ( ) и индексации [ ], а также переопределить операторы присваивания и инициализации.

Часто программы работают с объектами, которые являются конкретными представлениями абстрактных понятий. Например, тип данных int в C++ вместе с операциями +, -, *, / и т. д. предоставляет реализацию (ограниченную) математического понятия целых чисел. Такие понятия обычно включают в себя множество операций, которые кратко, удобно и привычно

представляют основные действия над объектами. К сожалению, язык программирования может непосредственно поддерживать лишь очень малое число таких понятий. Например, такие понятия, как комплексная арифметика, матричная алгебра, логические сигналы и строки не получили прямой поддержки в C++. Классы дают средство спецификации в C++ представления неэлементарных объектов вместе с множеством действий, которые могут над этими объектами выполняться. Иногда определение того, как действуют операции на объекты классов, позволяет программисту обеспечить более общепринятую и удобную запись для манипуляции объектами классов, чем та, которую можно достичь используя лишь основную функциональную запись. Например:

```
class complex {
    double re, im;
public:
    complex(double r, double i) { re=r; im=i; }
    friend complex operator+(complex, complex);
    friend complex operator*(complex, complex);
};
```

Этот класс определяет простую реализацию понятия комплексного числа, в которой число представляется парой чисел с плавающей точкой двойной точности, работа с которыми осуществляется посредством операций + и * (и только). Программист задает смысл операций + и * с помощью определения функций с именами operator+ и operator*. Если, например, даны b и c типа complex, то b+c означает (по определению) operator+(b,c). Теперь есть возможность приблизить общепринятую интерпретацию комплексных выражений. Например:

```
void f()
{
complex a = complex(1, 3.1);
complex b = complex(1.2, 2);
complex c = b;
a = b+c;
b = b+c*a;
c = a*b+complex(1,2);
}
```

При этом выполняются обычные правила приоритетов, поэтому второй оператор означает b=b+(c*a), а не b=(b+c)*a.

Можно описывать функции, определяющие значения следующих операций:

```
+ - * / % ^ & | ~ !
= < > += -= *= /= %= ^= &=
|= << >> >>= <<= == != <= >= &&
|| ++ -- [] () new delete
```

Последние четыре — это индексирование, вызов функции, выделение свободной памяти и освобождение свободной памяти. Изменить приоритеты перечисленных операций невозможно, как невозможно изменить и синтаксис выражений. Нельзя, например, определить унарную операцию % или бинарную !. Невозможно определить новые лексические символы операций, но в тех случаях, когда множество операций недостаточно, вы можете использовать запись вызова функции. Используйте, например, не **, а pow(). Эти ограничения могут показаться драконовскими, но более гибкие правила могут очень легко привести к неоднозначностям. Например, на первый взгляд определение операции **, означающей возведение в степень, может показаться очевидной и простой задачей, но подумайте еще раз. Должна ли ** связываться влево (как в Фортране) или вправо (как в Алголе)? Выражение a**p должно интерпретироваться как a*(*p) или как (a)**(p)?
Имя функции операции есть ключевое слово operator (то есть, операция), за которым следует сама операция, например, operator<<. Функция операция описывается и может вызываться так же, как любая другая функция. Использование операции - это лишь сокращенная запись явного вызова функции операции. Например:

```
void f(complex a, complex b)
{
complex c = a + b;          // сокращенная запись
complex d = operator+(a,b); // явный вызов
}
```

При наличии предыдущего описания complex оба инициализатора являются синонимами.

Бинарная операция может быть определена или как функция член, получающая один параметр, или как функция друг, получающая два параметра. Таким образом, для любой бинарной операции @ запись aa@bb может интерпретироваться или как aa.operator@(bb), или как operator@(aa,bb). Унарная операция, префиксная или постфиксная, может быть определена или как функция член, не получающая параметров, или как функция друг, получающая один параметр. Таким образом, для любой унарной операции @ запись aa@ или @aa может интерпретироваться или как aa.operator@(), или как operator@(aa). Рассмотрим следующие примеры:

```
class X {
// друзья
friend X operator-(X);      // унарный минус
friend X operator-(X,X);    // бинарный минус
friend X operator-();       // ошибка: нет операндов
friend X operator-(X,X,X);  // ошибка: несоответствие числа
                                    // аргументов
X* operator&(); // унарное & (взятие адреса)
X operator&(X); // бинарное & (операция И)
X operator&(X,X);   // ошибка: несоответствие числа аргументов
```

```
};
```

Когда операции ++ и -- перегружены, префиксное использование и постфиксное различить невозможно.

Функция операция должна или быть членом, или получать в качестве параметра по меньшей мере один объект класса (функциям, которые переопределяют операции new и delete, это делать необязательно). Это правило гарантирует, что пользователь не может изменить смысл никакого выражения, не включающего в себя определенного пользователем типа. В частности, невозможно определить функцию, которая действует исключительно на указатели.

Функция операция, первым параметром которой предполагается основной тип, не может быть функцией членом. Рассмотрим, например, сложение комплексной переменной aa с целым 2: aa+2, при подходящим образом описанной функции члене, может быть проинтерпретировано как aa.operator+(2), но с 2+aa это не может быть сделано, потому что нет такого класса int, для которого можно было бы определить + так, чтобы это означало 2.operator+(aa). Даже если бы такой тип был, то для того, чтобы обработать и 2+aa и aa+2, понадобилось бы две различных функции члена. Так как компилятор не знает смысла +, определенного пользователем, то не может предполагать, что он коммутативен, и интерпретировать 2+aa как aa+2. С этим примером могут легко справиться функции друзья.
Все функции операции по определению перегружены. Функция операция задает новый смысл операции в дополнение к встроенному определению и может существовать несколько функций операций с одним и тем же именем, если в типах их параметров имеются отличия, различимые для компилятора, чтобы он мог различать их при обращении.

Рассмотрим пример описания объекта, состоящего из текстовой строки, ее длины и двух переопределенных операций: сложения и вывода (<<).

```
class string {public: char s [80];
                      int str_len;
string operator + (string, string);
friend ostream & operator << (ostream &, string &);
 };
string string :: operator + (string s1, string s2)
{ string temp;
  strcpy (temp, s1.s);
  strcut (temp, s2.s);
  temp.str_len=s1.str_len+s2.str_len;
  return temp;
}
ostream & operator << (ostream & st, string & x)
{return (st << "Это строка:" << x.s << "Ее длина=" << x.str_len << '\n');
 }
void main ( )
```

```
{ string st1, st2, st3;
  strcpy (st1.s, «Это 1 строка»);
  strcpy (st2.s, «Это 2 строка»);
  st1.str_len=strlen (s1.s);
  st2.str_len=strlen (s2.s);
  st3=st1+st2;
  cout<<st3;
}
```

11.4 Наследование

Наследование – это повторное использование уже работающих классов с внесенными необходимыми дополнениями.

Наследование заключается в приеме некоторым производным классом компонентов базового класса. Формат описания производного класса следующий:

Class Tag: public Base {…};

Tag — имя производного класса;

Base — имя базового класса;

public – указывает право наследования;

операция «:» означает базируется на.

Компонентами производного класса является все компоненты базового класса, за исключением конструктора, деструктора и операции «=». К ним добавляются те компоненты, которые описаны для производного класса.

Таблица: соотношение атрибутов доступа в базовом и производном классе.

Наследование с правом доступа	Доступ в базовом классе	Доступ в производном классе
public	public protected private	public protected недоступно
protected	public protected private	protected protected недоступно
private	public protected private	private private недоступно

Исходя из вышесказанного, можно сделать следующие выводы:

1) открытые элементы базового класса (public) полностью употребляются в производном классе;

2) закрытые элементы базового класса (private) в производном классе недоступны;

3) защищенные элементы базового класса (protected) могут полностью использоваться в производном классе.

Если в производном классе определен компонент с тем же именем, что и в базовом, то к нему можно обратиться, используя «::» – оператор разрешения области видимости.

Пример описания класса Person, строящегося на основе базовых классов Job и Name:

```
struct Name {char*first name;
             char*second name;      //Ф.И.О.
             char*surname;
                    …
            }
struct Job {char*Company; // организация
            char*Position;  // должность
                   …
           }
struct Person : Name, Job {int age;  // возраст
                           char*sex;  // пол
                                  …
                          }
```

При создании объекта производного класса сначала вызываются конструкторы базового класса, а затем конструкторы производного класса.

При разрушении объекта сначала вызываются деструкторы производного класса, а затем деструкторы базового класса.

Чтобы передать конструкторам базовых классов параметры, их необходимо указать в определении конструктора производного класса после «:».

```
struct Name { char*first name;
              char*second name;
              char*surname;
Name (char*FN, char*SN, char*Sur N)  // конструктор
{first name=FN; second name=SN;
surname=Sur N;}
~Name ( ) {…}
}
struct Job {char*Company;
            char*Position;
       Job (char*C, char*P)  // конструктор
{Company=C, Position=P}
 };
struct Person: Name, Job
{int age;
 char*sex;
```

Person (char*I first name, char*I second name, char*I surname, char*I Company, char*I Position, int I age, char*I sex):

Name (I first name, I second name, I surname),

```
Job (I Company, I Position)  // конструктор производного класса
{age=I age;
  sex=I sex;
}
void main ( )
{
Person  Pl  (″Иван″,  ″Иванович″,  ″Иванов″,  ″УлГТУ″,  ″Инженер″,  50,
″муж″);
}
```

Если существует указатель на базовый класс, то его можно использовать для обращения к производному классу. Это связано с тем, что все наследуемое идет в производный класс первым. Использовать указатели на производный класс для базового класса нельзя.

Считаем, что у нас есть базовый класс Base 1, и есть производный класс Deriv. Рассмотрим примеры описания и использования указателей на базовый и производный классы:

```
1) Base 1 *p; // p — указатель на базовый класс
   p=new Base 1; // проинициализировали указатель
   p=new Deriv;
2) Deriv d;
   Base 1 * b_ptr=&d; // неявное преобразование указателей
3) Deriv *d_ptr;
   d_ptr=(Deriv*) b_ptr; // явное преобразование указателей
```

Объекты класса конструируются снизу вверх: сначала базовый, потом члены, а потом сам производный класс. Уничтожаются они в обратном порядке: сначала сам производный класс, потом члены, а потом базовый.

11.5 Использование свободной памяти при работе с классами

Рассмотрим особенности использования свободной памяти для размещения компонентов объекта класса, размещения объектов, особенности обработки таких объектов и удаление их из памяти.

Рассмотрим очень простой класс строк string:

```
struct string {
char* p;
int size;   // размер вектора, на который указывает p
string(int sz) { p = new char[size=sz]; }
~string() { delete p; }
};
```

Строка — это структура данных, состоящая из вектора символов и длины этого вектора. Вектор создается конструктором и уничтожается деструктором. Оператор new используется для выделении в памяти место под строку. Деструктор вызывается при выходе из блока, в котором определяется объект.

Рассмотрим пример:

```
main ( )
{
string* p = new string (100);
string* q = new string (200);
//…
```

Для доступа к компонентам класса в этом случае используется оператор ->. Например:

```
p->size
q->size
```

Компоненты данных класса могут быть указатели на классовый тип или ссылки на классовый тип. Для указателей и ссылок на классовый тип не нужно, чтобы такой класс был определен, нужно только, чтобы имя класса было объявлено. Например:

```
class Node;
Node *nh;
```

Используя указатель или ссылку на классовый тип как элемент класса, можно строить рекурсивные классовые структуры. Например, вершина бинарного дерева может содержать указатели на правого и левого потомка.

```
class Node {
Node *left, *right;
//…
};
```

12. ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ

12.1 Построение программ разветвленной структуры

Цель работы: приобретение практических навыков записи арифметических выражений и использования в программе оператора условия, условной операции и оператора переключателя.

Часть 1. Условный оператор

Вариант 1. Заданы вершины треугольника A(x1,y1), B(x2,y2), C(x3,y3). Вычислить длину медианы, проведенной из A.

Вариант 2. Известно, что из четырех чисел x1,x2,x3,x4 три равны между собой, а одно отлично от них. Присвоить переменной NF номер этого числа, а переменной F значение этого числа.

Вариант 3. Составить программу, которая бы с помощью оператора переключателя реализовала бы все возможные операции над двумя целыми числами.

Вариант 4. Составить программу, которая бы с помощью оператора переключателя реализовала бы все возможные операции над двумя вещественными числами.

Вариант 5. Для целого числа K от 1 до 9 вывести фразу «мне K лет», учитывая при этом, что при некоторых значениях K слово «лет» надо заменить на слово «год» или «года».

Вариант 6. Для натурального числа K напечатать фразу «мы нашли K грибов в лесу», согласовав окончание слова «гриб» с целым числом K.

Вариант 7. Составить программу, которая бы реализовала следующий алгоритм: по введенным названиям двух нот (до, ре, ми, фа, соль, ля, си) определить интервал, образованный нотами. Секунда — это интервал из двух соседних нот (по кругу), терция — интервал через ноту и т. д. (кварта, квинта, секста, септима).

Вариант 8. Введенные значения переменных a, b, c поменять местами так, чтобы оказалось a>=b>=c.

Вариант 9. Дано число x. Напечатать в порядке возрастания числа cos(x), 1+|x|, (1+ x·x) · (1+ x·x).

Вариант 10. Даны числа a, b, c, d, e, f. Найти координаты точки пересечения прямых, описываемых уравнениями $a \cdot x + b \cdot y = c$ и $d \cdot x + e \cdot y = f$, если она существует.

Вариант 11. Даны числа a, b, c. Если нельзя построить треугольник с такими длинами сторон, то вывести 0, если треугольник равносторонний — 1, если равнобедренный — 2, если прямоугольный — 3.

Вариант 12. Составить программу согласно условию. Присвоить переменной F значение 1, если ни одно из чисел x, y, z не является положительным и целым, и 0 в противоположном случае.

Вариант 13. Составить программу согласно условию. Присвоить переменной F значение 1, если только два числа из трех чисел x, y, z являются положительными и целыми, и 0 в противоположном случае.

Вариант 14. Составить программу согласно следующему условию. Присвоить переменной f значение 1, если цифра 3 входит в запись заданного трехзначного числа x, и 0 в противоположном случае.

Вариант 15. Заданы координаты вершин треугольника. Выяснить является ли заданный треугольник тупоугольным или нет.

Вариант 16. Если среди трех чисел A, B, C имеется хотя бы одно четное, вычислить максимальное из них, иначе – минимальное.

Вариант 17. Ввести положительное число A>=1. Найти наибольшее из выражений вида 1/A и sin(A).

Вариант 18. Ввести два числа. Меньшее число заменить их полусуммой, а большее — удвоенным произведением.

Вариант 19. Ввести три числа A, B, C. Удвоить каждое из них, если A>=B>=C, иначе поменять значения A и B.

Вариант 20. Определить, является ли точка с координатами X, Y точкой пересечения диагоналей квадрата со стороной R, одна вершина которого расположена в начале координат.

Вариант 21. Определить, лежит ли точка с координатами (X, Y) вне круга радиуса R с центром в точке (A, B) или внутри него.

Вариант 22. Определить корни системы уравнений

a·x + b·y = c

n·x + m·y = d.

Вариант 23. Вычислить y = sin X , если X<0 и y = tg X , если X>=0.

Вариант 24. Вычислить

$$y = \begin{cases} 1 + \text{sqrt}(\cos(x)), & x < -0.5 \\ x + 1, & -0.5 <= x <= 1 \\ 1 - x^2, & x > 1 \end{cases}$$

Вариант 25. Вычислить

$$y = \begin{cases} sgrt(tg(x^2 - 1)), & x > 1 \\ -2 * x,, & 0 <= x <= 1 \\ e^{\cos(x)}, & x < 0 \end{cases}$$

Часть 2. Оператор переключатель

Вариант 1. Составить программу, которая по номеру квартиры выдает фамилию ее владельца.

Вариант 2. Описать список времен года: лето, осень, зима, весна. По введенному значению времени года программа должна перечислить все месяцы этого сезона.

Вариант 3. Составить программу, которая по названию месяца выдавала бы количество дней в месяце.

Вариант 4. Составить программу, которая по названию месяца выдавала бы время года, к которому он принадлежит.

Вариант 5. Составить программу, которая по порядковому номеру месяца выдавала бы его название.

Вариант 6. Составить программу, которая бы по порядковому номеру месяца выдавала, к какому времени года он принадлежит.

Вариант 7. Составить программу, которая по введенному времени года выдавала бы название месяцев, относящихся к нему.

Вариант 8. Составить программу, которая по названию месяца выдавала бы его порядковый номер и название времени года.

Вариант 9. Дан список дисциплин, изучаемых в УлГТУ, и отчетность по ним. Составить программу, которая по названию дисциплины выдавала бы отчетность по нему.

История (экзамен, зачет);

Культурология (зачет);

Философия (экзамен, зачет);

Иностранный язык (экзамен, зачет);

Экономика (экзамен).

Вариант 10. Дан список дисциплин, изучаемых в УлГТУ, и номер семестра, когда они изучаются. Составить программу, которая по номеру семестра выдавала бы список изучаемых дисциплин.

История — 2,1;

Культурология — 3,4;

Философия — 4,3;

Иностранный язык — 4,1,2,3.

Вариант 11. По списку дисциплин, приведенных в 10 варианте заданий, составить программу, которая выдавала бы список дисциплин, читаемых на определенном курсе. Учитывать, что 1 курс — это 1 и 2 семестр, 2 курс — 3,4 семестр и т. д.

Вариант 12. Составить программу, которая бы с помощью оператора переключателя реализовала бы все возможные операции над двумя целыми числами.

Вариант 13. Составить программу, которая бы с помощью оператора переключателя реализовала бы все возможные операции над вещественными числами.

Вариант 14. Составить программу, которая бы присваивала переменной T значение true, если дата d1,m1 предшествует (в рамках года) дате d2,m2 и значение false в противоположном случае (d1 и d2 — дата, m1 и m2 — месяц).

Вариант 15. Составить программу, которая бы выдавала название месяца, следующего за введенным месяцем (с учетом того, что за декабрем идет январь).

Вариант 16. Составить программу, которая бы выдавала по названию страны название столицы этой страны (использовать не менее 6 – 7 названий).

Вариант 17. Составить программу, которая бы по русскому названию языка программирования выводила английское название этого языка.

Вариант 18. Составить программу, которая бы по введенному числу (до 10) выдавала бы название этой цифры.

Вариант 19. Составить программу, которая бы по введенному названию страны выдавала название ее континента.

Вариант 20. Составить программу, которая бы по значению переменной X, означающему некоторую длину в следующих единицах измерения: дециметр, километр, метр, миллиметр, сантиметр; выдавала бы эту длину в метрах.

Вариант 21. Составить программу, которая реализовала бы следующие действия: по введенному числу K (до 10) выдавала бы соответствующую ей римскую цифру.

Вариант 22. Для целого числа K от 1 до 9 напечатать фразу «мне K лет», учитывая при этом, что при некоторых значениях K слово «лет» надо заменить на слово «год» или «года».

Вариант 23. Для натурального числа K напечатать фразу «мы нашли K грибов в лесу», согласовав окончание слова «гриб» с числом K.

Вариант 24. Составить программу, которая бы реализовала следующий алгоритм: переменной T присвоить значение true, если сочетание день, месяц образует правильную дату, и значение false — иначе (учитывая количество дней в месяце и название месяца).

Вариант 25. Составить программу, которая бы реализовала следующий алгоритм: по порядковому номеру дня года определить дату, т. е. число и месяц.

12.2 Программирование циклов

Цель работы: изучение операторов цикла и особенностей их применения.

Вариант 1. Вычислить S = 1 + 2/2 + (2·4)/(2+4) + ... +(2·4·6·...(2·N))/(2+4+6+..+(2··N)) для заданного N.

Вариант 2. Вычислить S = sqrt(3 + sqrt(6 + sqrt(9 + ... + sqrt(96 + +sqrt(99)...).

Вариант 3. Вычислить S = 1/sqrt(1·) + 1/sqrt(3·5) + ... + 1/sqrt(199·201).

Вариант 4. Числа Фибоначчи определяются формулами F(0)=F(1)=1, F(i)=F(i-1)+F(i-2), i=2,3,... Найти 35-е число Фибоначчи.

Вариант 5. Найти первое число Фибоначчи, большее N, где N — заданное натуральное число, большее 1.

Вариант 6. Найти сумму чисел Фибоначчи, больших M и меньших N, где M и N — заданные натуральные числа, 1<M<N.

Вариант 7. Вычислить сумму всех чисел Фибоначчи, которые не превосходят 100.

Вариант 8. Подсчитать количество чисел Фибоначчи, которые не превосходят заданного целого числа. Напечатать их.

Вариант 9. Вычислить S = (1 + 1/3) · (1/5 + 1/7) · (1/9 +1/11 + 1/13) · ... · (1/33 + 1/35 + 1/37 + 1/39 + 1/41 + 1/43).

Вариант 10. Вычислить S = 4/2 + (4·7)/(2·6) +(4·7·10)/(2·6·10) + ... + (4·7·10·...·301)/(2·6·10·...·398).

Вариант 11. Вычислить S = cos(1 +cos(2 + ... + cos(39+ cos(40)...).

Вариант 12. Вычислить S = sh(x) = x + x^3/3! + x^5/5! + ... + x^(2·N+1)/(2·N+1)! для заданного N.

Вариант 13. Вычислить S = cos(x) = 1 - x^2/2! + x^4/4! +..+ (-1)^N· ·x^(2·N)/(2·N)! для заданного N.

Вариант 14. Вычислить S = Ln(1+x) = x - x^2/2 + x^3/3 +..+ (-1)^(N-1) · ·x^N/N для заданного N и |x|<1.

Вариант 15. Вычислить S = arctg(x) = x - x^3/3 + x^5/5 +...+ (-1)^N· ·x^(2·N+1)/(2·N+1) для заданного N и |x|<1.

Вариант 16. В последовательности целых положительных чисел определить максимальное четное число и его порядковый номер.

Вариант 17. В последовательности вещественных чисел определить наименьшее отрицательное число и его порядковый номер.

Вариант 18. В последовательности целых чисел определить третье положительное число и подсчитать количество цифр в нем.

Вариант 19. В последовательности символов выдать на печать TRUE, если значение последнего символа равно Ф.

Вариант 20. В последовательности чисел выдать на печать TRUE, если значение максимального числа больше числа 10.

Вариант 21. В последовательности вещественных чисел подсчитать произведение чисел, кратных 3.

Вариант 22. В последовательности чисел сравнить, что больше, сумма положительных или произведение отрицательных.

Вариант 23. В последовательности символов подсчитать количество букв и количество цифр.

Вариант 24. В последовательности чисел определить предпоследнее отрицательное число.

Вариант 25. Вычислить сумму ряда, общий член которого задан формулой $An=(x^n)/n!$.

12.3 Обработка массивов данных

Цель работы: ознакомиться с данными типа массив и основными приемами программирования задач обработки массивов.

Часть 1. Обработка одномерного массива

Вариант 1. Дан одномерный массив A, состоящий из N элементов. Сколько значений элементов в массиве A встречается более одного раза?

Вариант 2. Дан одномерный массив A, состоящий из N элементов. Сколько значений элементов встречается в массиве по 3 раза?

Вариант 3. Дан одномерный массив A, состоящий из N элементов. Переписать в одномерный массив B все элементы, заключенные между максимальным и минимальным значениями.

Вариант 4. Дан одномерный массив A, состоящий из N элементов. Определить количество чисел, входящих в массив по одному разу.

Вариант 5. Дан одномерный массив A, состоящий из N элементов. Подсчитать максимальное количество подряд идущих нулей.

Вариант 6. Дан одномерный массив A, состоящий из N элементов. Перенести в начало массива все положительные элементы, а в конец массива — все отрицательные.

Вариант 7. Дан одномерный массив A, состоящий из N элементов. Перенести в начало массива все четные элементы, а в конец массива — все нечетные.

Вариант 8. Дан одномерный массив A, состоящий из N элементов. Исключить из массива первый положительный элемент, следующий за максимальным.

Вариант 9. Дан одномерный массив A, состоящий из N элементов. Исключить из массива все нулевые элементы, расположенные между максимальным и минимальным элементами.

Вариант 10. Дан одномерный массив A, состоящий из N элементов. Исключить из массива первый, предшествующий максимуму, положительный элемент.

Вариант 11. Дан одномерный массив A, состоящий из N элементов. Подсчитать максимальное количество подряд идущих отрицательных элементов.

Вариант 12. Дан одномерный массив A, состоящий из N элементов. Найти первый и последний положительные элементы массива и подсчитать количество элементов, заключенных между ними.

Вариант 13. Дан одномерный массив A, состоящий из N элементов. Подсчитать максимальное количество положительных элементов, заключенных между нулями.

Вариант 14. Дан одномерный массив A, состоящий из N элементов. Считаем, что отрицательные элементы разбивают его на группы. Найти группу положительных элементов массива с максимальной суммой.

Вариант 15. Дан одномерный массив A, состоящий из N элементов. Считаем, что отрицательные элементы разбивают его на группы. Найти количество полученных групп, содержащих нули.

Вариант 16. Дан массив чисел $a_1,...,a_N$. Выяснить, имеются ли в данном массиве два идущих подряд положительных элемента. Подсчитать количество таких пар.

Вариант 17. Дан массив целых чисел $a_1,...,a_N$. Подсчитать количество пар элементов, удовлетворяющих условию $a_i<a_{i+1}$.

Вариант 18. Дан массив целых чисел $a_1,...,a_N$. Найти в данной последовательности все пары a_i, a_{i+1},такие, что $a_i=0$ и a_{i+1} кратно 10.

Вариант 19. Дан массив чисел $a_1,...,a_N$. Выяснить, имеются ли в данном массиве два идущих подряд отрицательных элемента. Подсчитать количество таких пар.

Вариант 20. Дан массив целых чисел $a_1,...,a_N$. Подсчитать количество пар элементов, удовлетворяющих условию $a_i>a_{i+1}$.

Вариант 21. Дан одномерный массив A, состоящий из N элементов. Подсчитать максимальное количество подряд идущих положительных чисел.

Вариант 22. Дан одномерный массив A, состоящий из N элементов. Подсчитать максимальное количество подряд идущих отрицательных чисел.

Вариант 23. Дан одномерный массив A, состоящий из N элементов. Исключить из массива все положительные элементы, расположенные между максимальным и минимальным элементами.

Вариант 24. Дан одномерный массив A, состоящий из N элементов. Исключить из массива все отрицательные элементы, расположенные между максимальным и минимальным элементами.

Вариант 25. Дан одномерный массив A, состоящий из N элементов. Исключить из массива первый отрицательный элемент, следующий за максимальным.

Часть 2. Обработка двумерного массива

Вариант 1. Дана матрица целых чисел. В строках, все элементы которых четные, расположить элементы в обратном порядке.

Вариант 2. Дана матрица символов. Подсчитать количество строк, в которых букв больше, чем цифр.

Вариант 3. Дана матрица целых чисел. Собрать все положительные элементы массива выше главной диагонали (заполнение осуществлять по строкам).

Вариант 4. Дана матрица целых чисел. Собрать все нулевые элементы выше главной диагонали (заполнение осуществлять параллельно главной диагонали).

Вариант 5. Дана матрица символов. Написать программу обращения к каждому элементу этой матрицы, если считать, что имена строк — буквы алфавита (по возрастанию), а имена столбцов — целые числа (по возрастанию).

Вариант 6. Дана матрица вещественных чисел. Найти максимальный элемент и ближайший к нему (по значению) элемент матрицы. Поиск осуществлять в квадратном контуре, центром которого является максимум, а длина стороны — пять элементов массива.

Вариант 7. Дана матрица вещественных чисел. Найти минимальный элемент и ближайший к нему (по значению) элемент матрицы. Поиск осуществлять в направлении параллельном главной диагонали.

Вариант 8. Дана матрица вещественных чисел. Найти число с максимальной дробной частью, переставить строки и столбцы так, чтобы это число стояло в левом верхнем углу.

Вариант 9. Дана матрица целых чисел. Подсчитать количество элементов, предшествующих максимуму, и количество элементов, следующих за минимумом.

Вариант 10. Дана матрица символов. Определить строку, в которой максимальное количество букв.

Вариант 11. Дана матрица целых чисел. Собрать все отрицательные элементы выше побочной диагонали (заполнение осуществлять по строкам).

Вариант 12. Дана матрица вещественных чисел. Найти максимальный элемент и наиболее удаленный от него (по значению) элемент матрицы. Поиск осуществлять в квадратном контуре, центром которого является максимум, а длина стороны — три элемента массива.

Вариант 13. Дана матрица вещественных чисел. Найти минимальный элемент и наиболее удаленный от него (по значению) элемент матрицы. Поиск осуществлять в направлении параллельном главной диагонали.

Вариант 14. Дана матрица целых чисел. Собрать все положительные элементы ниже побочной диагонали (заполнение осуществлять параллельно побочной диагонали).

Вариант 15. Дана матрица вещественных чисел. Найти число с максимальной дробной частью, переставить строки и столбцы так, чтобы это число стояло в левом верхнем углу.

Вариант 16. Дана вещественная матрица A(N,M). Составить программу нахождения минимального отрицательного элемента матрицы и нахождения его местоположения.

Вариант 17. Дана вещественная матрица A(N,M). Составить программу нахождения максимального положительного элемента матрицы и нахождения его местоположения.

Вариант 18. Дана матрица вещественных чисел. Найти минимальный элемент и наиболее удаленный от него (по значению) элемент матрицы. Поиск осуществлять в направлении параллельном побочной диагонали.

Вариант 19. Дана матрица целых чисел. Собрать все отрицательные элементы выше побочной диагонали (заполнение осуществлять по столбцам).

Вариант 20. Дана матрица вещественных чисел. Найти минимальный элемент и ближайший к нему (по значению) элемент матрицы. Поиск осуществлять в квадратном контуре, центром которого является минимум, а длина стороны — пять элементов массива.

Вариант 21. Дана матрица вещественных чисел. Найти минимальный элемент матрицы. Поиск осуществлять в направлении параллельном побочной диагонали.

Вариант 22. Дана матрица вещественных чисел. Найти максимальный элемент матрицы. Поиск осуществлять в направлении, параллельном главной диагонали.

Вариант 23. Дана матрица вещественных чисел. Найти максимальный элемент и ближайший к нему (по значению) элемент матрицы. Поиск осуществлять в направлении параллельном побочной диагонали.

Вариант 24. Дана матрица вещественных чисел. Найти число с минимальной дробной частью, переставить строки и столбцы так, чтобы это число стояло в правом верхнем углу.

Вариант 25. Дана матрица вещественных чисел. Найти максимальное и минимальное положительные числа и количество чисел между ними.

Часть 3. Сортировка массива

Вариант 1. Дана последовательность, расположить ее положительные элементы, стоящие на нечетных местах по возрастанию.

Вариант 2. Дана последовательность, расположить ее ненулевые элементы по убыванию.

Вариант 3. Переставить строки исходной матрицы так, чтобы убывало количество нулей в строках.

Вариант 4. Упорядочить все строки матрицы по числу элементов, кратных 3, т. е. на первое место поставить строку с наименьшим числом таких элементов и т. д., на последнее место – с наибольшим числом таких элементов.

Вариант 5. Расположить в порядке возрастания положительные элементы правого верхнего треугольника матрицы.

Вариант 6. Расположить в порядке убывания положительные элементы левого нижнего треугольника матрицы.

Вариант 7. Дана последовательность, элементы которой есть целые двузначные числа. Упорядочить последовательность по возрастанию сумм цифр соответствующих элементов.

Вариант 8. Дана последовательность, расположить ее отрицательные элементы по убыванию.

Вариант 9. Дана последовательность, элементы которой есть целые двузначные числа. Упорядочить последовательность по убыванию произведений цифр соответствующих элементов.

Вариант 10. Дана последовательность, расположить ее элементы, попадающие в интервал от A до B в порядке возрастания.

Вариант 11. Дана последовательность, расположить ее четные (по значению) элементы по убыванию.

Вариант 12. Дана последовательность, расположить ее элементы, кратные 3, по убыванию.

Вариант 13. Дана матрица. Упорядочить ее строки, содержащие нули, в порядке возрастания их количества.

Вариант 14. Дана матрица. Упорядочить по убыванию положительные элементы ее правой половины.

Вариант 15. Дана матрица. Упорядочить по возрастанию ненулевые элементы ее нижней половины.

Вариант 16. Дана последовательность чисел, расположить ее отрицательные элементы, стоящие на нечетных местах, по возрастанию.

Вариант 17. Дана последовательность чисел, расположить ее ненулевые элементы по возрастанию.

Вариант 18. Переставить строки исходной матрицы так, чтобы возрастало количество нулей в строках.

Вариант 19. Упорядочить все строки матрицы по числу элементов, кратных 2, т. е. на первое место поставить строку с наименьшим числом таких элементов и т. д., на последнее место — с наибольшим числом таких элементов.

Вариант 20. Расположить в порядке возрастания ненулевые элементы правого верхнего треугольника матрицы.

Вариант 21. Расположить в порядке убывания ненулевые элементы левого нижнего треугольника матрицы.

Вариант 22. Дана последовательность, элементы которой есть целые трехзначные числа. Упорядочить последовательность по возрастанию сумм цифр соответствующих элементов.

Вариант 23. Дана последовательность чисел, расположить ее положительные элементы по убыванию.

Вариант 24. Дана последовательность чисел, элементы которой есть целые трехзначные числа. Упорядочить последовательность по убыванию произведений цифр соответствующих элементов.

Вариант 25. Дана последовательность, расположить ее элементы, не попадающие в интервал от A до B, в порядке возрастания.

12.4 Обработка строк

Цель работы: знакомство с программными средствами описания и обработки строковых данных в языке Си.

Вариант 1. Дано предложение, слова в нем разделены пробелом, подсчитать, сколько букв «а» в каждом слове.

Вариант 2. Дано предложение, слова в нем разделены пробелом, подсчитать, сколько букв и цифр в последнем слове.

Вариант 3. Дано предложение, слова в нем разделены пробелом, поменять местами первое и последнее слова.

Вариант 4. Дано предложение, слова в нем разделены пробелом, поменять местами четные и нечетные по порядку следования слова.

Вариант 5. Даны N предложений. Найти в каждом первое слово и напечатать их в строку через пробел.

Вариант 6. Даны N предложений. Найти в каждом последнее слово и напечатать их в строку через пробел.

Вариант 7. Даны N предложений. Подсчитать количество слов в каждом предложении и вывести на печать.

Вариант 8. Дано предложение, слова в нем разделены пробелом. Подсчитать количество слов, которые начинаются с той буквы, которой заканчивается предыдущее слово.

Вариант 9. Дано предложение, слова в нем разделены пробелом. Подсчитать количество слов, которые начинаются с той же буквы, что и последующее слово.

Вариант 10. Дано предложение, слова в нем разделены пробелом. Упорядочить слова в порядке возрастания их длины.

Вариант 11. Дано предложение, слова в нем разделены пробелом. Упорядочить слова по алфавиту (только по первой букве).

Вариант 12. Дано N предложений, слова в которых разделены пробелами. Вывести их на печать в порядке возрастания количества слов в предложении.

Вариант 13. Дано N предложений, слова в которых разделены пробелами. Вывести их на печать в порядке возрастания общей длины слов в предложении (без учета количества разделяющих пробелов).

Вариант 14. Дано предложение, слова в нем разделены пробелом. Составить из него два предложения по правилу: в одно переписать все четные по порядку следования слова, а в другое – нечетные.

Вариант 15. Дано N предложений, слова в которых разделены пробелом. Составить новый текст по следующему правилу: исключить из текста все слова на букву 'a'.

Вариант 16. Дано предложение. Подсчитать, сколько раз встречается в каждом слове заданный символ.

Вариант 17. Дано предложение. Распечатать все буквы B, перед которыми непосредственно находится буква C в одном слове.

Вариант 18. Дано предложение. Напечатать true, если в заданном слове буква **a** встречается чаще, чем буква **b**, и напечатать false в противоположном случае.

Вариант 19. Дано N предложений, слова в которых разделены пробелами. Вывести их на печать в порядке убывания количества слов в предложении.

Вариант 20. Дано предложение, слова в нем разделены пробелом. Составить из него два предложения по правилу: в первое переписать все нечетные по порядку следования слова, а во второе – четные.

Вариант 21. В предложении найти все однокоренные слова. Корень задается с клавиатуры.

Вариант 22. Удалить в предложении все повторные вхождения слов и распечатать получившееся предложение.

Вариант 23. Выделить из введенного предложения слова, содержащие повторяющиеся буквы.

Вариант 24. Все буквы каждого слова в предложении записать в обратном порядке и распечатать получившееся предложение.

Вариант 25. Проверьте на совпадение два предложения. Количеством пробелов между словами пренебрегать. Знаки препинания учитывать.

12.5 Обработка структур данных

Цель работы: знакомство с описанием структур данных на языке Си, получение практических навыков обработки структур с использованием указателей.

Составить модель следующего объекта:

Вариант 1. Состав студентов факультета с разбивкой на группы. Количество специальностей на факультете, групп каждой специальности и студентов в каждой группе задать самостоятельно. Составить модуль поиска адреса (ссылки) элемента списка по его информационным полям.

Вариант 2. Состав специальностей вуза с разбивкой на факультеты. Количество факультетов и специальностей каждого факультета задать самостоятельно. Составить модуль поиска элементов с диапазоном шифров специальностей от ШИФР1 до ШИФР2.

Вариант 3. Список внутренних телефонов организации с разбивкой по отделам. Количество отделов и телефонов внутри отдела задать самостоятельно. Составить модуль поиска всех телефонов с заданными двумя первыми цифрами.

Вариант 4. Список участков предприятия с разбивкой по цехам. Количество цехов и участков каждого цеха задать самостоятельно. Составить модуль поиска участка с максимальным номером.

Вариант 5. Список работников цеха с разбивкой по профессиям. Количество профессий и работников каждой профессии цеха задать самостоятельно. Составить модуль сортировки фамилий по алфавиту.

Вариант 6. Состав парка ЭВМ вычислительного центра с разбивкой ЭВМ по сериям. Количество серий и ЭВМ разных серий задать самостоятельно. Составить модуль сортировки ЭВМ по возрастанию объема оперативной памяти.

Вариант 7. Список номеров и маршрутов рейсов автобусов с разбивкой рейсов по районам следования. Количество районов и рейсов в каждый район задать самостоятельно. Составить модуль сортировки рейсов по убыванию номеров.

Вариант 8. Список ПО на лазерных дисках в вычислительном центре с разбивкой по операционным системам. Количество ОС и дисков для каждой системы задать самостоятельно. Составить модуль поиска элементов списка по его информационным полям.

Вариант 9. Состав спортивного соревнования с разбивкой по видам спорта. Количество видов спорта и участников каждого вида спорта задать самостоятельно. Составить модуль формирования нового списка спортсменов, содержащего фамилии каждого второго спортсмена.

Вариант 10. Программа работы конференции с разбивкой докладов по секциям. Количество секций и докладов в каждой секции задать самостоятельно. Составить модуль формирования нового списка, содержащего доклады с несколькими авторами.

Вариант 11. Конструкция технического устройства, состоящего из нескольких блоков. Причем каждый блок содержит определенное количество модулей. Составить модуль формирования нового списка, из которого удаляется модуль с заданным наименованием.

Вариант 12. Конструкция модуля, состоящая из микросхем. Количество микросхем и вводов-выводов каждой микросхемы задать самостоятельно. Составить модуль формирования нового списка вводов-выводов, из которого удаляются элементы с номерами 7 и 14.

Вариант 13. Конструкция вычислительной сети, состоящей из узлов разных типов. Количество узлов и типов узлов задать самостоятельно. Составить модуль формирования нового списка узлов, в котором добавлен новый заданный узел.

Вариант 14. Книга состоит из глав и параграфов. Количество глав и параграфов каждой главы задать самостоятельно. Составить модуль формирования нового списка параграфов, в котором первый и последний элементы поменялись местами.

Вариант 15. Задачники состоят из списка задач, разбитых на темы. Количество тем и задач задать самостоятельно. Составить модуль формирования нового списка задач, в котором после задачи с заданным номером записан элемент с новой задачей.

Вариант 16. Набор материалов, имеющихся на складе, с разбивкой по виду товара. Количество видов товаров и штук каждого товара задать самостоятельно. Составить модуль формирования двух новых списков из исходного списка товаров. В первый список попадают номера товаров меньше некоторого заданного, во второй – больше.

Вариант 17. Список школьников, занимающихся в кружках, с разбивкой по кружкам. Количество кружков и школьников в каждом кружке задать самостоятельно. Составить модуль формирования двух новых списков из списка школьников по следующему принципу: в первый список попадают первые десять школьников, во второй — остальные.

Вариант 18. Список покупок, сделанных кем-то в течение месяца, с разбивкой по статьям расходов. Количество статей расхода и покупок по каждой статье задать самостоятельно. Составить модуль формирования нового списка покупок с ценой больше заданной ЦЕНЫ.

Вариант 19. Список кварталов города с разбивкой по районам. Количество районов и кварталов в каждом районе задать самостоятельно. Составить модуль формирования нового списка кварталов, сводного для двух указанных районов.

Вариант 20. Список учебных дисциплин, которые должен изучить студент за время обучения в вузе с разбивкой на циклы (аппаратный, общенаучный, гуманитарный и т. д.). Количество циклов и дисциплин в циклах для разных специальностей задать самостоятельно. Составить модуль формирования нового списка, полученного слиянием трех списков дисциплин для заданных циклов.

Вариант 21. Список деталей механического устройства с разбивкой деталей по агрегатам. Количество агрегатов и деталей в каждом агрегате задать самостоятельно. Составить модуль формирования нового списка деталей, общих для двух заданных агрегатов.

Вариант 22. Список номеров автомобилей, паркующихся на платной стоянке, с разбивкой по маркам. Количество марок и автомобилей каждой марки задать самостоятельно. Составить модуль формирования нового списка, содержащего номера с заданной частью номера.

Вариант 23. Список предприятий, расположенных в городе, с разбивкой по министерствам. Количество министерств и предприятий по каждому министерству задать самостоятельно. Составить модуль формирования нового списка предприятий, являющихся заводами.

Вариант 24. Список вузов страны с разбивкой по профилям. Количество профилей и вузов каждого профиля задать самостоятельно. Составить модуль формирования нового списка вузов заданного профиля, являющихся университетами.

Вариант 25. Список сотрудников предприятия с разбивкой по должностям. Количество должностей и количество сотрудников, имеющих одинаковую должность, задать самостоятельно. Составить модуль формирования списка однофамильцев, работающих на одной должности.

12.6 Обработка списков

Цель работы: приобрести практические навыки работы с динамическими структурами данных на языке Си.

Часть 1. Составить динамическую модель следующего объекта

Вариант 1. Состав студентов факультета с разбивкой на группы. Количество специальностей на факультете, групп каждой специальности и студентов в каждой группе переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль поиска адреса (ссылки) элемента списка по его информационным полям.

Вариант 2. Состав специальностей вуза с разбивкой на факультеты. Количество факультетов и специальностей каждого факультета переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль поиска элементов с диапазоном шифров специальностей от ШИФР1 до ШИФР2.

Вариант 3. Список внутренних телефонов организации с разбивкой по отделам. Количество отделов и телефонов внутри отдела переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль поиска всех телефонов с заданными двумя первыми цифрами.

Вариант 4. Список участков предприятия с разбивкой по цехам. Количество цехов и участков каждого цеха переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль поиска участка с максимальным номером.

Вариант 5. Список работников цеха с разбивкой по профессиям. Количество профессий и работников каждой профессии цеха переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль сортировки фамилий по алфавиту.

Вариант 6. Состав парка ЭВМ вычислительного центра с разбивкой ЭВМ по сериям. Количество серий и ЭВМ разных серий переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль сортировки ЭВМ по возрастанию объема оперативной памяти.

Вариант 7. Список номеров и маршрутов рейсов автобусов с разбивкой рейсов по районам следования. Количество районов и рейсов в каждый район переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль сортировки рейсов по убыванию номеров.

Вариант 8. Список ПО на лазерных дисках в вычислительном центре с разбивкой по операционным системам. Количество ОС и дисков для каждой системы переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль поиска элементов списка по его информационным полям.

Вариант 9. Состав спортивного соревнования с разбивкой по видам спорта. Количество видов спорта и участников каждого вида спорта переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка спортсменов, содержащего фамилии каждого второго спортсмена.

Вариант 10. Программа работы конференции с разбивкой докладов по секциям. Количество секций и докладов в каждой секции переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка, содержащего доклады с несколькими авторами.

Вариант 11. Конструкция технического устройства, состоящего из нескольких блоков. Причем каждый блок содержит произвольное количество модулей. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка, из которого удаляется модуль с заданным наименованием.

Вариант 12. Конструкция модуля, состоящая из микросхем. Количество микросхем и вводов-выводов каждой микросхемы переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка вводов-выводов, из которого удаляются элементы с номерами 7 и 14.

Вариант 13. Конструкция вычислительной сети, состоящей из узлов разных типов. Количество узлов и типов узлов переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка узлов, в котором добавлен новый заданный узел.

Вариант 14. Книга состоит из глав и параграфов. Количество глав и параграфов каждой главы переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка параграфов, в котором первый и последний элементы поменялись местами.

Вариант 15. Задачники состоят из списка задач, разбитых на темы. Количество тем и задач переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка задач, в котором после задачи с заданным номером записан элемент с новой задачей.

Вариант 16. Набор материалов, имеющихся на складе, с разбивкой по виду товара. Количество видов товаров и штук каждого товара переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования двух новых списков из исходного списка товаров. В первый список попадают номера товаров меньше некоторого заданного, во второй – больше.

Вариант 17. Список школьников, занимающихся в кружках, с разбивкой по кружкам. Количество кружков и школьников в каждом кружке переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования двух новых списков из списка школьников по следующему принципу: в первый список попадают первые десять школьников, во второй — остальные.

Вариант 18. Список покупок, сделанных кем-то в течение месяца, с разбивкой по статьям расходов. Количество статей расхода и покупок по каждой статье переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка покупок с ценой больше заданной ЦЕНЫ.

Вариант 19. Список кварталов города с разбивкой по районам. Количество районов и кварталов в каждом районе переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка кварталов, сводного для двух указанных районов.

Вариант 20. Список учебных дисциплин, которые должен изучить студент за время обучения в вузе, с разбивкой на циклы (аппаратный, общенаучный, гуманитарный и т. д.). Количество циклов и дисциплин в циклах для разных специальностей переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка, полученного слиянием трех списков дисциплин для заданных циклов.

Вариант 21. Список деталей механического устройства с разбивкой деталей по агрегатам. Количество агрегатов и деталей в каждом агрегате переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка деталей, общих для двух заданных агрегатов.

Вариант 22. Список номеров автомобилей, паркующихся на платной стоянке, с разбивкой по маркам. Количество марок и автомобилей каждой марки переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка, содержащего номера с заданной частью номера.

Вариант 23. Список предприятий, расположенных в городе, с разбивкой по министерствам. Количество министерств и предприятий по каждому министерству переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка предприятий, являющихся заводами.

Вариант 24. Список вузов страны с разбивкой по профилям. Количество профилей и вузов каждого профиля переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования нового списка вузов заданного профиля, являющихся университетами.

Вариант 25. Список сотрудников предприятия с разбивкой по должностям. Количество должностей и количество сотрудников, имеющих одинаковую должность, переменно. Для фрагмента модели, являющегося одномерным списком, составить модуль формирования списка однофамильцев, работающих на одной должности.

Часть 2. Создание и обработка кольцевого списка

Вариант 1. Дана последовательность символов, оканчивающихся точкой. Найти всех «соседей» заданного символа. Первый и последний символ считать «соседями».

Вариант 2. Дана последовательность символов, оканчивающихся точкой. Подсчитать количество символов, у которых одинаковые «соседи». Первый и последний символ считать «соседями».

Вариант 3. Дана последовательность символов, оканчивающихся точкой. Удалить все символы, у которых одинаковые «соседи». Первый и последний символ считать «соседями».

Вариант 4. Дана последовательность символов, оканчивающихся точкой. Переставить в обратном порядке все символы между первым и последним вхождениями заданного символа (если символ входит в последовательность не менее двух раз).

Вариант 5. Дана последовательность символов. В конец данной последовательности добавить все ее символы, располагая их в обратном порядке. Например, из последовательности символов 1 2 3 получить кольцевой список 1 2 3 2 1.

Вариант 6. Дана последовательность латинских букв, оканчивающихся точкой. Пусть символ **k** означает отмену предыдущей буквы; **n** символов подряд отменяют (стирают) **n** предыдущих букв, если они есть. Преобразовать последовательность с учетом вхождения в нее символа **k.**

Вариант 7. Дана запись многочлена от переменой **x** произвольной степени с целыми коэффициентами, причем его одночлены могут быть и не упорядочены по степеням **x,** а одночлены одной и той же степени могут повторяться. Например, $8 \cdot x^4 - 15 \cdot x + 5 \cdot x^4 - x^2 + 5 - x$. Привести подобные члены в этом многочлене.

Вариант 8. Дана запись многочлена от переменой **x** произвольной степени с целыми коэффициентами, причем его одночлены могут быть и не упорядочены по степеням **x,** а одночлены одной и той же степени могут повторяться. Например, $8 \cdot x^4 - 15 \cdot x + 5 \cdot x^4 - x^2 + 5 - x$. Расположить одночлены по убыванию степеней **x.**

Вариант 9. Дано натуральное число **n** и действительные числа $x_1, x_2, \ldots x_n$. С помощью кольцевого списка вычислить $x_1 \cdot x_n + x_2 \cdot x_{n-1} + \ldots + x_n \cdot x_1$.

Вариант 10. Дано натуральное число **n** и действительные числа $x_1, x_2, \ldots x_n$. С помощью кольцевого списка вычислить $(x_1 + x_n) \cdot (x_2 + x_{n-1}) \cdot \ldots \cdot (x_n + x_1)$.

Вариант 11. Дано натуральное число **n** и действительные числа $x_1, x_2, \ldots x_n$. С помощью кольцевого списка вычислить $(x_1 + x_2 + 2 \cdot x_n) \cdot (x_2 + x_3 + 2 \cdot x_{n-1}) \ldots \cdot (x_{n-1} + x_n + 2 \cdot x_1)$.

Вариант 12. Дано натуральное число **n** и действительные числа $x_1, x_2, \ldots x_n$. Получить кольцевой список вида $x_1, x_2, \ldots x_n, x_1, x_2, \ldots x_n$.

Вариант 13. Дано натуральное число **n** и действительные числа $x_1, x_2, \ldots x_n$. Получить кольцевой список вида $x_1, x_2, \ldots x_n, x_n, x_{n-1}, \ldots x_1$.

Вариант 14. Дано натуральное число **n** и действительные числа $x_1, x_2, \ldots x_n$. Получить кольцевой список вида $x_n, x_{n-1}, \ldots x_1, x_1, x_2, \ldots x_n$.

Вариант 15. Дана последовательность латинских букв, оканчивающихся точкой. Пусть символ **k** означает отмену последующей буквы; **n** символов подряд отменяют (стирают) **n** последующих букв, если они есть. Преобразовать последовательность с учетом вхождения в нее символа **k.**

12.7 Использование видеопамяти

В заданиях 1– 24 вывод на экран осуществлять с помощью прямого обращения к видеопамяти. Библиотечные функции, реализующие вывод на экран (conio.h), к программному файлу не подключать. Содержательную часть задачи реализовать с помощью функции пользователя. Программа должна вызвать данную функцию несколько раз с различным набором аргументов.

Вариант 1. Разработать функцию, реализующую горизонтальное меню в верхней строке экрана:

вход: массив строк;

выход: номер выбранной строки.

Замечание 1. Число строк неограниченно. Если строки меню не размещаются на строке экрана, то формируется меню с меньшим числом строк. Остальная часть строк остается за кадром и доступ к ним осуществляется с помощью навигационных клавиш (т. е. осуществляется скроллинг строки).

Замечание 2. Разместить строки меню так, чтобы они занимали полную строку экрана.

Функция должна реагировать на клавиши: ←, →, Home, end, Enter.

Вариант 2. Разработать функцию, реализующую вертикальное меню в центре экрана.

вход: массив строк;

выход: номер выбранной строки.

Замечание. Число строк неограниченно. Если строки меню не размещаются экране, то формируется меню с меньшим числом строк. Остальная часть строк остается за кадром и доступ к ним осуществляется с помощью навигационных клавиш.

Функция должна реагировать на клавиши: ↑, ↓, Enter, PdDn, PgUp.

Вариант 3. Разработать функцию, реализующую запрос одной из альтернатив "ДА - НЕТ" и возвращающую номер выбранной альтернативы. ("ДА" - 1,"НЕТ" - 2"). Запрос организовать в форме вертикального меню в центре экрана с выбором с помощью навигационной клавиатуры.

Вариант 4. Разработать функцию, реализующую запрос одной из альтернатив "ДА - НЕТ" и возвращающую номер выбранной альтернативы. ("ДА" - 1,"НЕТ" - 2"). Запрос организовать в форме горизонтального меню в центре экрана с выбором с помощью ввода первой буквы (либо "Д/D/д/d", либо "Н/N/н/n").

Вариант 5. Разработать функцию, входными данными для которой является массив из N строк. Функция должна вывести на экран строки в виде вертикального меню (координаты верхнего левого угла меню фиксированы). Выбор нужной строки осуществить с помощью навигационной клавиатуры. Функция должна возвращать номер выбранной строки.

Замечание. Строки меню полностью умещаются на одном экране.

Функция должна реагировать на клавиши: ↑, ↓, Enter, home (первая строка), end (последняя строка).

Вариант 6. Разработать функцию, организующую вывод в центр экрана «всплывающего» окна до заданных пределов. Процесс «всплытия» должен происходить с замедлением по времени, обеспечивающим наблюдение за ним. После завершения формирования окна вывести в него произвольную строку. Входными данными для процедуры являются предельные размеры окна и строка, выводимая в него.

Функция должна (после всплытия окна) реагировать на клавиши: ←, →, ↑, ↓ и обеспечивать передвижение окна по экрану.

Вариант 7. Разработать функцию, организующую вывод в центр экрана «выплывающего» из левой границы экрана окна до центра экрана. Процесс «всплытия» должен происходить с замедлением по времени, обеспечивающим наблюдение за ним. После завершения формирования окна вывести в него произвольную строку. Входными данными для функции являются размеры окна и строка, выводимая в него. Функция должна (после всплытия окна) реагировать на клавиши: ←, →, ↑, ↓ и обеспечивать изменение размера окна, enter – «центрирование» окна после изменения его размеров.

Вариант 8. Разработать функцию, организующую вывод в заданное место экрана окна заданных размеров. Окно должно «проявляться» из отдельных частей (последовательность «проявления» частей – случайная, размеры частей – произвольны, но процесс «проявления» должен занимать не менее 10 этапов) с замедлением по времени. После нажатия <enter> окно должно также постепенно исчезать.

Вариант 9. Разработать функцию, организующую вывод в заданное место экрана окна заданных размеров. Окно должно «проявляться» из отдельных горизонтальных «слоев» («слои» проявляются в случайном порядке) с замедлением по времени. После нажатия <enter> окно должно также постепенно исчезать в порядке, обратном проявлению.

Функция должна (после проявления окна) реагировать на клавиши: ←, →, ↑, ↓ и обеспечивать изменение размера окна.

Вариант 10. Разработать функцию, организующую вывод в заданное место экрана окна заданных размеров. Окно должно «проявляться» из отдельных вертикальных «слоев» («слои» проявляются в случайном порядке) с замедлением по времени. После нажатия <enter> окно должно также постепенно исчезать в порядке, обратном проявлению.

Функция должна (после проявления окна) реагировать на клавиши: ←, →, ↑, ↓ и обеспечивать передвижение окна по экрану.

Вариант 11. Разработать функцию, организующую вывод в центр экрана окна, «растущего» постепенно от минимального (с размерами 1*1) до заданных пределов. Процесс «роста» должен происходить с замедлением по времени, обеспечивающим наблюдение за ним. После завершения

формирования окна вывести в него произвольную строку. Входными данными для функции являются предельные размеры окна и строка, выводимая в него.

Функция должна (после формирования окна) реагировать на клавиши: ←, →, ↑, ↓ и обеспечивать передвижение окна по экрану.

Вариант 12. Разработать функцию, организующую вывод в заданное место экрана окна заданных размеров. Окно должно «проявляться» из отдельных вертикальных «слоев» («слои» проявляются в случайном порядке) с замедлением по времени. После нажатия <enter> окно должно также постепенно исчезать в порядке, обратном проявлению.

Функция должна (после проявления окна) реагировать на клавиши ↑, ↓ и обеспечивать: изменение цвета окна (<стрелка_вверх>) и изменение цвета экрана за пределами окна (<стрелка_вниз>).

Вариант 13. Разработать функцию, организующую вывод в центр экрана «выплывающего» из левой границы экрана окна до центра экрана. Процесс «всплытия» должен происходить с замедлением по времени, обеспечивающим наблюдение за ним. После завершения формирования окна вывести в него произвольную строку. Входными данными для функции являются размеры окна и строка, выводимая в него.

Функция должна (после всплытия окна) реагировать на клавиши ←, → и обеспечивать изменение цвета: цвета окна (→) и изменение цвета строки (←).

Вариант 14. Разработать функцию, входными данными для которой является массив из N строк. Функция должна вывести на экран строки в виде вертикального меню (координаты верхнего левого угла меню фиксированы). Первая буква строк меню должна быть выделена другим цветом. Выбор нужной строки осуществить как с помощью навигационной клавиатуры, так и нажатием первой буквы строки меню. Функция должна возвращать номер выбранной строки.

Замечание. Строки меню полностью умещаются на одном экране.

Функция должна реагировать на клавиши: ↑, ↓ , Enter, home (первая строка), end (последняя строка).

Вариант 15. Разработать функцию, входными данными для которой является массив из N строк. Функция должна вывести на экран строки в виде горизонтального меню (координаты верхнего левого угла меню фиксированы). Первая буква строк меню должна быть выделена другим цветом. Выбор нужной строки осуществить как с помощью навигационной клавиатуры, так и нажатием первой буквы строки меню. Функция должна возвращать номер выбранной строки.

Замечание. Строки меню полностью умещаются на одном экране.

Функция должна реагировать на клавиши: ←, →, Enter, home (первая строка), end (последняя строка).

Вариант 16. Разработать функцию, реализующую запрос одной из альтернатив "ДА - НЕТ" и возвращающую номер выбранной альтернативы.

("ДА" - 1,"НЕТ" - 2"). Запрос организовать в форме горизонтального меню в центре экрана с выбором с помощью ввода первой буквы (либо "Д/д/D/d", либо "Н/н/N/n"). Программа–заглушка должна обратиться к функции и вывести номер выбранной альтернативы.

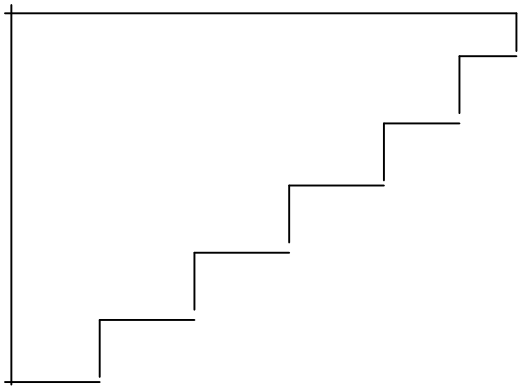
Вариант 17. Разработать функцию, входными данными для которой является массив из 5 строк вида: "F1 - XXX","F2 - XXX",...,"F5 - XXX", где XXX – произвольная комбинация символов. Процедура должна вывести на экран строки в виде горизонтального меню (координаты верхнего левого угла меню фиксированы). Выбор нужной строки осуществить с помощью выбора нужной функциональной клавиши. Изображение функциональной клавиши выполнить цветом, отличным от цвета остальных символов строки.

Функция должна возвращать номер выбранной строки.

Вариант 18. Разработать функцию, входными данными для которой является массив из 5 строк вида: "F1 - XXX","F2 - XXX",...,"F5 - XXX", где XXX – произвольная комбинация символов. Функция должна вывести на экран строки в виде вертикального меню (координаты верхнего левого угла меню фиксированы). Выбор нужной строки осуществить с помощью выбора нужной функциональной клавиши. Изображение функциональной клавиши выполнить цветом, отличным от цвета остальных символов строки.

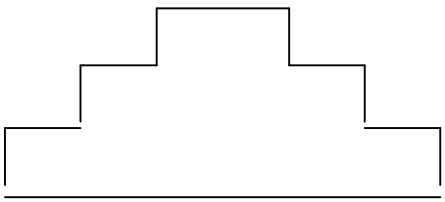
Функция должна возвращать номер выбранной строки.

Вариант 19. Разработать функцию, выводящую в текстовом режиме в центре экрана фигуру в форме лестницы вниз:



Число ступеней и размер лестницы являются входными данными функции. После формирования фигуры программа должна реагировать на клавиши - $\leftarrow$, $\rightarrow$ и обеспечивать изменение цвета экрана внутри фигуры ($\rightarrow$) и изменение цвета экрана за пределами фигуры ($\leftarrow$).

Вариант 20. Разработать функцию, выводящую в текстовом режиме в центре экрана фигуру в форме лестницы



Число ступеней и размер лестницы являются входными данными функции. После формирования фигуры программа должна реагировать на клавиши HOME и END и обеспечивать изменение цвета экрана внутри фигуры (HOME) и изменение цвета экрана за пределами фигуры (END).

Вариант 21. Разработать функцию, выводящую в текстовом режиме в центре экрана фигуру в форме лестницы вниз (см. вариант 19). Число ступеней и размер лестницы являются входными данными функции. После формирования фигуры программа должна реагировать на клавиши ←, → и обеспечивать передвижение по экрану фигуры.

Вариант 22. Разработать подпрограмму-функцию, выводящую в текстовом режиме в центре экрана фигуру в форме лестницы (см. вариант 20). Число ступеней и размер лестницы являются входными данными функции. После формирования фигуры программа должна реагировать на клавиши HOME и END и обеспечивать передвижение фигуры по экрану: HOME – вверх, END – вниз.

Вариант 23. Разработать функцию, выводящую на экран в текстовом режиме в центре экрана фигуру (см. вариант 20). Число ступеней и размер лестницы являются входными данными функции. После формирования фигуры программа должна реагировать на клавиши PgUp и PgDn и обеспечивать передвижение фигуры по экрану: PgUp – вверх, PgDn – вниз.

Вариант 24. Разработать функцию, выводящую на экран в текстовом режиме в центре экрана замкнутую фигуру в форме лестницы вверх. Число ступеней и размер фигуры являются входными данными функции. После формирования фигуры программа должна реагировать на клавиши ←, → и обеспечивать передвижение фигуры по экрану.

12.8 Обработка прерываний

Решить задачи, приведенные в задании 12.7 с использованием прерываний ROM-BIOS. Библиотеки Crt и Conio.h к программному файлу не присоединять.

12.9 Описание класса

Вариант 1. Определите класс формирования разбора упорядоченного массива целых чисел. Общий интерфейс класса должен выглядеть примерно так:

```
class mas
{
//…
public:
      mas (char*);
int razbor ( );
      void print ( );
};
```

конструктор mas::mas ( ) имеет параметр-строку, защищающую массив, и проверяет данные объекта на упорядочность. Функция mas::razbor ( ) возвращает число элементов в массиве, а mas::print( ) выдает элементы массива в cout. Использовать эти функции можно так:

mas (“1, 3, 4 ,4.3”); // элементы массива вводятся в строчку через запятую

cout<<”x =” <<x.razbor( )<< “\n”;

x.print();

программа должна выполнить данные с тремя различными объектами.

Вариант 2. Определите класс для формировании, сортировки и вывода массива вещ чисел. Общий интерфейс класса должен выглядеть примерно так:

```
class mas_real
{
//…
public:
      mas_real (char*);
      void sort( );
void print( );
};
```

Использовать эти функции можно так:

mas (“1, 3.4, 4 ,4.3”); // элементы массива вводятся в строчку через запятую

x.sort( );

x.print ( ); // вывод результата в cout

программа должна выполнить данные с тремя различными объектами.

Вариант 3. Определите класс для формирования, разбора и вывода матрицы целых чисел, первый столбец которых упорядочен по возрастанию. Общий интерфейс класса должен выглядеть примерно так:

```
class matr
{
//…
public:
matr (char*);
int razbor1( );
int razbor2 ( );
void print ( );
};
```

конструктор matr::matr( ) имеет параметр-строку, задающую матрицу и проверяет данные объекта на упорядоченность. Функция matr::razbor1( ) возвращает число строк, а функция matr::razbor1( ) возвращает число столбцов в матрице. matr::print() выдает элементы матрицы в cout.

Использовать эти функции можно так:

mas (“(“1, 3, 4), (4, 3, 12)”); // элементы массива вводятся в строчку через запятую, строки заключаются в скобки

cout<<”x =” <<x.razbor1( )<< “\n”;

cout<<”x =” <<x.razbor2( )<< “\n”;

x.print ( );

программа должна выполнить данные с тремя различными объектами.

Вариант 4. Определите класс для формирования, разбора и вывода простых арифметических выражений, состоящих из целых констант и операций +, –, /. Общий интерфейс класса должен выглядеть примерно так:

```
class expr
{
//…
public:
    expr (char*);
    int razbor ( );
    void print( );
};
```

конструктор expr::exper( ) имеет параметр-строку, задающую выражение. Функция expr::razbor ( ) возвращает число операндов в выражение, expr::print( ) выдает представление cout.

Использовать эти функции можно так:

```
Expr (“123/4+123*4-3””);
cout<<”x =” <<x.razbor( )<< “\n”;
x.print( );
```

программа должна выполнить данные с тремя различными объектами.

Вариант 5. Определите класс для формирования, вывода на дисплей и стирания графических точек. В состав функций-членов должны войти функции, возвращающие координаты точки. Общий интерфейс класса должен выглядеть примерно так:

```
class point
{
//…
public
    point (int x, int y);
    int getx( );
    int gety( );
void Hide( );
void show( );
};
```

point – конструктор класса;

getx ( ), gety ( ) – функции, возвращающие соответственно значения координат точек по X и по Y;

Hide – «стирает» нарисованную на экране точку;

show – «рисует» точку нужным цветом.

Использовать эти функции можно так:

```
point (100,100);
cout <<"x = "<< x.getx( ) << " y = " << x.gety( )<< "\n";
x.show( );
x.Hide( );
```

программа должна выполнить данные с тремя различными объектами.

Вариант 6. Определите класс для формирования, вывода на дисплей и стирания графических линей. В состав функций-членов должны войти функции, возвращающие координаты линии. Общий интерфейс класса должен выглядеть примерно так:

```
class line
{
//…
public
      line (int, int, int, int);
      int getx1( );
      int gety1( );
int getx2( );
      int gety2( );
void Hide( );
void show( );
};
```

point – конструктор класса;

getx1 ( ), gety1 ( ), getx2 ( ), gety2 ( ) – функции, возвращающие соответственно значения координат крайних точек линии по X и по Y;

Hide – «стирает» нарисованную на экране точку;

show – «рисует» точку нужным цветом.

Использовать эти функции можно так:

```
point (100,100, 200, 200);
x.show( );
x.Hide( );
```

программа должна выполнить данные с тремя различными объектами.

Вариант 7. Определите класс для формирования, подсчета количества четных цифр и вывода на дисплей строки символов. В процессе формирования конструктор должен «отфильтровать» (т. е. удалять из строки) символы, отличные от заглавных латинских букв или цифр. Общий интерфейс класса должен выглядеть примерно так:

```
class string
{
//…
public:
      string (char*);
      int count( );
      void print ( );
```

};
string – конструктор класса;
count – функция, возвращающая количество четных цифр;
print – выводит строку в cout.
Использовать эти функции можно примерно так:
string(“DJVHKS^*()*05#!$EFCW97$%^*”
);
cout << “x=” << x.count() << “\n”;
x.print( );
программа должна выполнить данные с тремя различными объектами.

Вариант 8. Определить класс для формирования и, подсчета кол-ва «длинных» слов(т. е. длинна которых превышает 15 символов) и вывода на дисплей массива слов. В процессе формирования конструктор должен «отфильтровать» (т. е. удалить из массива) слова, содержащие символы отличные от заглавных латинских букв и цифр. Общий интерфейс класса должен выглядеть примерно так:

```
class mas_string
{
//…
public:
mas_string (char*);
int count( );
void( );
};
```

mas_string – конструктор класса;
count – функция. Возвращающая кол-во длинных слов;
print – выводит массив слов в cout.
Использовать эти функции можно примерно так:

```
mas_string( “DJV, HKS ^*()* 05#!$E”
);
// разделители слов – пробел или запятая
cout << “x=” << x.count( ) << “\n”;
x.print( );
```

программа должна выполнить данные с тремя различными объектами.

Вариант 9. Определите класс для формирования и вывода на дисплей массива случайных целых чисел в заданном кол-ве и с заданным распределением. В процессе формирования конструктор должен «отфильтровать» (т. е. удалять из массива) все нечетные числа. Общий интерфейс класса должен выглядеть примерно так:

```
class mas_random
{
//…
public:
      mas_random (int, int);
```

```
        int count( );
    void print( );
    };
```

mas_random – конструктор класса;

count – функция, возвращающая количество чисел, сумма разрядов у которых больше 10;

print – выводит массив в cout.

Использовать эти функции можно примерно так:

```
mas_random( 20, 100
);
cout << “x=” << x.count( ) << “\n”;
x.print( );
```

программа должна выполнить данные с тремя различными объектами.

Вариант 10. Определите класс для формирования и вывода на дисплей матрицы случайных целых чисел заданной размерности и с заданным распределением. В процессе формирования конструктор долежен «отфильтровывать» (т. е. заменять на ноль) все нечетные числа. Общий интерфейс класса должен выглядеть примерно так:

```
class маек_random
{
//…
public:
    маек_random (int, int, int);
    int count( );
void print( );
```

matr_random – конструктор класса;

count – функция, возвращающая количество нулевых элементов матрицы print – выводит матрицу.

Использовать эти функции можно примерно так:

```
matr_random( 5, 5, 100);
cout << “x=” << x.count( ) << “\n”;
x.print( );
```

программа должна выполнить данные с тремя различными объектами.

Вариант 11. Определите класс для формирования, кодирования и вывода на дисплей случайного двоичного вектора заданной длины. Кодирование осуществлять в линейном коде с проверкой на четность. (К вектору добавляется один контрольный разряд – такой, чтобы общее число единичных разрядов в коде было четным). Общий интерфейс класса должен выглядеть примерно так:

```
class kod_chet
{
//…
public:
    kod_chet (int); // конструктор класса
```

```
        void kod( ); // кодирование вектора
        void print ( ); // вывод вектора в cout
};
```

использовать эти функции можно примерно так:

```
kod_chet (20);
x.kod( );
x.print( );
```

программа должна выполнить данные с тремя различными объектами.

Вариант 12. Определите класс для формирования, декодирования и вывода двоичного вектора. Декодирование осуществлять в линейном коде с проверкой не четность. Общий интерфейс класса должен выглядеть примерно так:

```
class dekod_chet
{
//…
public
        dekod_chet (char*); // конструктор класса
        int dekod( ); // декодироване вектора
                        // 1-есть ошибка, 0-нет
        void print( );
};
```

Использовать эти функции можно примерно так:

```
dekod_chet (“1000100000000111111”);
k=k.dekod( );
x.print( );
```

Организовать вывод результата декодирования.

Программа должна выполнить данные с тремя различными объектами.

Вариант 13. Определите класс для формирования, декодирования и вывода на дисплей группы случайных двоичных векторов заданной длинны. Количество векторов в группе задается. Кодирование осуществляется в линейном коде с проверкой на четность. (К вектору добавляется один контрольный разряд – такой, чтобы общее число единичных разрядов в коде было четным). Общий интерфейс класса должен выглядеть примерно так:

```
class kod_chet
{
//…
public
        kod_chet (int, int ; // конструктор класса
        int kod( ); // декодироване вектора
        void print( );
};
```

Использовать эти функции можно примерно так:

```
kod_chet (12, 20);
x.kod( );
```

x.print( );

Программа должна выполнить данные с тремя различными объектами.

Вариант 14. Определите класс для формирования, декодирования и вывода группы двоичных векторов. Декодирование осуществлять в линейном коде с проверкой на четность. Общий интерфейс класса должен выглядеть примерно так:

```
class dekods-chet
// ...
public:
dekod_chet(char*); // конструктор класса:
int* dekod( ) // декодирование векторов:
// (1 - есть омвка.О-нет):
void print( ); // вывод закодированных векторов в cout
};
```

Использовать эти функции можно так:

```
dekod_chet (“1000001000001111111, 1111110000111111001, 1110011000001111111”);
x.print( );
```

Организовать вывод результата декодирования.

Программа должна выполнить данные с тремя различными объектами.

Вариант 15. Определите класс для формирования полного множества двоичных векторов, ортогональных заданному и вывода их в cout. (Вектора X = (x1,...,xn) и Y = (y1,…,yn) ортогональны если xi*yi+…+xn*yn=0). Общий интерфейс класса должен выглядеть примерно так:

```
clasc ort_vec
// ...
public:
ort_vec(char*); // конструктор класса;
void kods( ); // формирует множество ортогональных векторов
void print( ): // вывод закодированных векторов в cout
};
```

Использовать эти функции можно примерно так:

```
ort_vec (“1100”);
x.kods( );
x.print( );
```

Программа должна выполнить данные с тремя различными объектами.

Вариант 16. Определите класс для формирования, кодирования и вывода на дисплей двоичного вектора. Кодирование осуществлять в линейном коде с проверкой на нечетность. (К вектору добавляется один контрольный разряд – такой, чтобы общее число единичных разрядов в коде было нечетным). Общий интерфейс класса должен выглядеть примерно так:

```
class kod_nechet
{
// ...
```

```
public:
kode_nechet(char*); // конструктор масса:
void kod( ); // кодирование вектора:
void print( ); // вывод кода вектора в cout.
};
```

Использовать эти функции можно так:

```
kod_nechet (“1000001000001111111”);
x.kod( );
x.print( );
```

Программа должна выполнить данные с тремя различными объектами.

Вариант 17. Определите класс для формирования, сортировки по алфавиту и вывода на дисплей массива слов. В процессе формирования конструктор должен «отфильтровывать» (т. е. удалять из массива) слова, содержащие символы, отличные от заглавных латинских букв. Общий интерфейс класса должен выглядеть примерно так:

```
class mas_string
{
// ...
public:
mas_stririg (char**);
int count( );
void sort( );
void print( );
};
```

mas_string – конструктор класса;

count – функция, возвращающая количество длинных слов (длиной >10 символов);

sort( ) – сортирует массив слов по алфавиту;

print – выводит пассив слов в cout.

Использовать эти функции можно так:

```
mas_strlng (“HJX”, “SDVSL;DCL”, “SFV”);
cout << “x=” << x.count( ) << “\n”;
x.sort( );
x.print( );
```

Программа должна выполнить данные с тремя различными объектами.

Вариант 18. Определите класс для формирования, сортировки и вывода на дисплей массива вещественных чисел. Общий интерфейс класса должен выглядеть примерно так:

```
class mas_real
{
//…
public:
      mas_real(char*);
      void sort( );
```

```
    void print( );
};
```
Ипользовать эти функции можно примерно так:
```
mas_real (“1.2, 32.5, 4.3”)
x.sort( );
x.print( );
```
Программа должна выполнить данные с тремя различными объектами.

Вариант 19. Определите класс для формирования, сортировки и вывода на дисплей случайных целых чисел в заданном количестве и с заданным распределением. В процессе формирования конструктор должен «отфильтровывать» (т. е. заменить на нуль) все нечетные числа. Общий интерфейс класса должен выглядеть примерно так:
```
class mas_rendom
{
//…
public:
    mas_rendom(int, int);
    void sort( );
    void print( );
};
```
Ипользовать эти функции можно примерно так:
```
mas_rendom (20, 100)
x.sort( );
x.print( );
```
Программа должна выполнить данные с тремя различными объектами.

Вариант 20. Определите класс для формирования, перевода в десятичную форму и вывода на дисплей целых шестнадцатеричных чисел. В процессе формирования конструктор должен «отфильтровывать» (т. е. заменить на нуль) все символы, отличные от представления шестнадцатеричного разряда (т. е. символы отличные от 0-9 и A-F). Общий интерфейс класса должен выглядеть примерно так:
```
class num_16
{
//…
public
    num_16 (char*);
    void to_10( );
    void print( );
};
```
num_16-конструктор класса

to_10 – функция перевода строки с 16-ми символами в строку с десятичными символами

Ипользовать эти функции можно примерно так:
```
num_16 (“956a56b44f”);
```

x.to_10( );
x.print( );
Программа должна выполнить данные с тремя различными объектами.

12.10 Описание функций-членов класса

Вариант 1. Определите класс new_mas для обработки множества целых чисел, используя разработанный класс mas (см. лаб. раб. № 9 вариант 1). Новый класс дополнительно должен реализовать следующие операции:

(search) поиск нового элемента;

(insert) включение нового элемента в множество;

(delete) удаление элемента из множества;

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 2. Определите класс new_mas_real для обработки массива вещественных чисел, используя разработанный класс mas_real (см. лаб. раб. № 9 вариант 2). Новый класс дополнительно должен реализовать следующие операции:

(search) поиск нового элемента;

(insert) включение нового элемента в массив;

(delete) удаление элемента из массива;

(add) слияние двух упорядоченных массивов в один упорядоченный массив.

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 3. Определите класс new_matr для обработки матрицы целых чисел, используя разработанный класс matr (см. лаб. раб. № 9 вариант 3). Новый класс дополнительно должен реализовать следующие операции:

(search) поиск нового элемента;

(insert) включение новой строки;

(delete) удаление элемента из массива;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 4. Определите класс new_expr для обработки арифметических выражений, используя разработанный класс matr (см. лаб. раб. № 9 вариант 4). Новый класс дополнительно должен реализовать следующие операции:

(count) подсчитывает значение выражения;

(add) находит сумму значений двух выражений;

(mult) находит произведение значений двух выражений;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 5. Определите класс new_point для формирования, вывода на дисплей и стирания графических изображений :

-точек;

-окружностей;

используя разработанный класс point (см. лаб. раб. № 9 вариант 5).

Новый класс дополнительно должен реализовать следующие операции:

(move_p) передвигает точку в заданное место;

(draw) рисует окружность;

(move_c) передвигает окружность в заданное место;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 6. Определите класс new_line для формирования, вывода на дисплей и стирания графических изображений :

-точек;

-треугольников;

используя разработанный класс line (см. лаб. раб. № 9 вариант 6).

Новый класс дополнительно должен реализовать следующие операции:

(move_i) передвигает линию в заданное место;

(draw) рисует треугольник;

(hide) стирает треугольник;

(move_c) передвигает треугольник в заданное место;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 7. Определите класс new_string для обработки строк, используя разработанный класс string (см. лаб. раб. № 9 вариант 7).

Новый класс дополнительно должен реализовать следующие операции:

(add) склеивание двух строк;

(minus) удаление из первой строки всех символов второй строки;

(mult) формирование строки состоящей из символов, присутствующих одновременно как в первой строке, так и во второй;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 8. Определите класс new_mas_string для обработки массива строк, используя разработанный класс mas_string (см. лаб. раб. № 9 вариант 8).

Новый класс дополнительно должен реализовать следующие операции:

(add) склеивание двух массивов строк;

(minus) удаление из первой массива всех слов второго массива;

(mult) формирование массива состоящей из слов, присутствующих одновременно как в первой строке, так и во второй;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 9. Определите класс new_mas_random для обработки массива строк, используя разработанный класс mas_random (см. лаб. раб. № 9 вариант 9).

Новый класс дополнительно должен реализовать следующие операции:

(form) формировать из заданного массива массив требуемой длины.

(add) сложение двух векторов (по правилу zi=xi+yi);

(minus) вычитание двух векторов (по правилу zi=xi-yi);

(mult) умножение скаляра на вектор (по правилу zi=v+yi);

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 10. Определите класс new_matr_random для обработки матрицы, используя разработанный класс matr_random (см. лаб. раб. № 9 вариант 10).

Новый класс дополнительно должен реализовать следующие операции:

(add) сложение двух матриц;

(minus) вычитание двух матриц;

(mult) умножение скаляра на матрицу;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 11. Определите класс new_kod_chet для обработки двоичных векторов, используя разработанный класс kod_chet (см. лаб. раб. № 9 вариант 11).

Новый класс дополнительно должен реализовать следующие операции:

(add) сложение двух векторов;

(rang) вычисление ранга вектора (c=a[1]+a[2]+…+a[n]);

(mult) умножение векторов(c[i]=a[i]*b[i]);

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 12. Определите класс new_dekod_chet для обработки двоичных векторов, используя разработанный класс dekod_chet (см. лаб. раб. № 9 вариант 12).

Новый класс дополнительно должен реализовать следующие операции:

(del) деление двух векторов с получением вектора частного (ch) и вектора остатка (ost);

(rang) вычисление ранга вектора (c=a[1]+a[2]+…+a[n]);

(mult) скалярное умножение векторов(c=sum(a[i]*b[i]));

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 13. Определите класс new_kodes_chet для обработки группы двоичных векторов, используя разработанный класс kodes_chet (см. лаб. раб. № 9 вариант 13).

Новый класс дополнительно должен реализовать следующие операции:

(drt) проверка взаимной ортогональности группы векторов;

(tran) транспонирование двоичной матрицы;

(sum) вычисление суммы двух матриц;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 14. Определите класс new_dekodes_chet для обработки группы двоичных векторов, используя разработанный класс dekodes_chet (см. лаб. раб. № 9 вариант 14).

Новый класс дополнительно должен реализовать следующие операции:

(ort) проверка взаимной ортогональности группы векторов;

(mult_1) умножение двоичной матрицы на скаляр (значение скаляра 0 или 1);

(mult_2) умножение двоичной матрицы на двоичный вектор;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 15. Определите класс new_ort_vec для обработки группы двоичных векторов, используя разработанный класс ort_vec (см. лаб. раб. № 9 вариант 15).

Новый класс дополнительно должен реализовать следующие операции:

(tran) транспонирование двоичной матрицы;

(mult) перемножение двоичных матриц;

(mult_2) умножение двоичной матрицы на двоичный вектор;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 16. Определите класс new_kod_nechet для обработки вектора, используя разработанный класс kod_nechet (см. лаб. раб. № 9 вариант 16).

Новый класс дополнительно должен реализовать следующие операции:

(sogl) проверка на согласованость размерности матрицы и вектора для выполнения перемножения;

(mult_1) перемножение двоичного вектора на скаляр;

(mult_2) перемножение двоичного вектора на двоичную матрицу;

(mult_3) умножение двоичной матрицы на двоичный вектор;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 17. Определите класс new_mas_string для обработки массива строк, используя разработанный класс mas_string (см. лаб. раб. № 9 вариант 17).

Новый класс дополнительно должен реализовать следующие операции:

(add) склеивание двух массивов строк;

(rshift) циклический сдвиг вправо на заданное растояние массива;

(lshift) циклический сдвиг влево на заданное растояние массива;

(mult_3) умножение двоичной матрицы на двоичный вектор;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 18. Определите класс new_mas_real для обработки массива вещественных чисел, используя разработанный класс mas_real (см. лаб. раб. № 9 вариант 18).

Новый класс дополнительно должен реализовать следующие операции:

(add) слияние двух упорядоченных массивов в один упорядоченный;

(delete) удаление элемента из массива;

(mult) перемножение двух массивов по правилу скалярного произведения;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 19. Определите класс new_mas_random для обработки массива целых чисел, используя разработанный класс mas_random (см. лаб. раб. № 9 вариант 19).

Новый класс дополнительно должен реализовать следующие операции:

(minus) вычитание двух массивов по правилу вычитания двух вукторов;

(delete) удаление элемента из массива;

(mult) перемножение двух массивов по правилу скалярного произведения;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 20. Определите класс new_num_16 для обработки 16-х чисел, используя разработанный класс num_16 (см. лаб. раб. № 9 вариант 20).

Новый класс дополнительно должен реализовать следующие операции:

(minus) вычитание двух 16-х чисел;

(add) сложение двух 16-х чисел;

Объявите какие можно функции дружественными.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

12.11 Дружественные функции

Вариант 1. Перепишите следующие функции класса new_mas (см. лаб. раб. № 10 вариант 1).

(insert) замените на оператор + (включение нового элемента в множество)

(delete) замените на оператор – (удаление элемента из множества);

Дополнительно переопределите следующий оператор:

Сложение двух массивов (оператор++)

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 2. Перепишите следующие функции класса new_mas_real (см. лаб. раб. № 10 вариант 2).

(insert) замените на оператор >> (включение нового элемента в массив);

(delete) замените на оператор – (удаление элемента из массива);

(add) замените на оператор + (слияние двух упорядоченных массивов в один упорядоченный массив).

Дополнительно переделайте следующий оператор:

Нахождение суммы положительных элементов в массиве – унарный оператор +.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 3. Перепишите следующие функции класса new_matr (см. лаб. раб. № 10 вариант 3).

(insert) замените на оператор >> (включение новой строки);

(delete) замените на оператор << (удаление элемента из массива);

Дополнительно переопределите следующие операторы:

Сложение матриц – оператор +

вычитание матриц – оператор –

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 4. Перепишите следующие функции класса new_expr (см. лаб. раб. № 10 вариант 4).

(add) замените на оператор + (находит сумму значений двух выражений);

(mult) замените на оператор * (находит произведение значений двух выражений);

Дополнительно переопределите следующие операторы:

Вычитание значений двух выражений – оператор – ;

Деление значений двух выражений – оператор /;

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 5. Перепишите следующие функции класса point (см. лаб. раб. № 10 вариант 5).

(move_p) замените на оператор > (передвигает точку в заданное место);

(move_c) замените на оператор >> (передвигает окружность в заданное место);

Дополнительно переопределите следующие операторы:

Вычерчивание линии между двумя заданными точками – оператор +.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 6. Перепишите следующие функции класса new_line (см. лаб. раб. № 10 вариант 6).

(move_i) замените на оператор > (передвигает линию в заданное место);

(move_c) замените на оператор >> (передвигает треугольник в заданное место);

Дополнительно переопределите следующие операторы:

Вычерчивание двух параллельных линии – оператор +.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 7. Перепишите следующие функции класса new_string (см. лаб. раб. № 10 вариант 7).

(add) замените на оператор + (склеивание двух строк);

(minus) замените на оператор – (удаление из первой строки всех символов второй строки);

(mult) замените на оператор * (формирование строки состоящей из символов присутствующих одновременно как в первой строке так и во второй);

Дополнительно переопределите следующие операторы:

Сравнивание двух строк – оператор ==.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 8. Перепишите следующие функции класса new_mas_string (см. лаб. раб. № 10 вариант 8).

Новый класс дополнительно должен реализовать следующие операции:

(add) замените на оператор + (склеивание двух массивов строк);

(minus) замените на оператор – (удаление из первой массива всех слов второго массива);

(mult) замените на оператор * (формирование массива состоящей из слов присутствующих одновременно как в первом строке так и во втором);

Дополнительно переопределите следующие операторы:

Сравнивание двух массивов строк – оператор ==.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 9. Перепишите следующие функции класса new_mas_random (см. лаб. раб. № 10 вариант 9).

(add) замените на оператор + (сложение двух векторов (по правилу zi=xi+yi));

(minus) замените на оператор – (вычитание двух векторов (по правилу zi=xi-yi));

(mult) замените на оператор * (умножение скаляра на вектор (по правилу zi=v+yi));

Дополнительно переопределите следующий оператор:

Возведение вектора в квадрат – оператор ^.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 10. Перепишите следующие функции класса new_matr_random (см. лаб. раб. № 10 вариант 10).

Новый класс дополнительно должен реализовать следующие операции:

(add) замените на оператор + (сложение двух матриц);

(minus) замените на оператор – (вычитание двух матриц);

(mult) замените на оператор * (умножение скаляра на матрицу);

Дополнительно переопределите следующий оператор:

Возведение матрицы в заданную степень – оператор ^.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 11. Перепишите следующие функции класса new_kod_chet (см. лаб. раб. № 10 вариант 11).

(add) замените на оператор + (сложение двух векторов);

(rang) замените на унарный оператор + (вычисление ранга вектора (c=a[1]+a[2]+…+a[n]));

(mult) замените на оператор * (умножение векторов(c[i]=a[i]*b[i]));

Дополнительно переопределите следующий оператор:

Получение инверсного кода (по правилу c[i]=1 – a[i]) – унарный оператор ^.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 12. Перепишите следующие функции класса new_dekod_ (см. лаб. раб. № 10 вариант 12).

(del) замените на оператор / (деление двух векторов с получением вектора частного (ch) и вектора остатка (ost));

(rang) замените на унарный оператор + (вычисление ранга вектора (c=a[1]+a[2]+…+a[n]));

(mult) замените на оператор * (скалярное умножение векторов(c=sum(a[i]*b[i])));

Дополнительно переопределите следующий оператор:

Сравнение двух векторов – оператор ==.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 13. Перепишите следующие функции класса new_kodes_chet (см. лаб. раб. № 10 вариант 13).

(drt) замените на оператор | (проверка взаимной ортогональности группы векторов);

(tran) замените на унарный оператор ! (транспонирование двоичной матрицы);

(sum) замените на бинарный оператор / (вычисление суммы двух матриц);

Дополнительно переопределите следующий оператор:

Удаление из одной группы векторов всех общих с другой группой векторов – бинарный оператор –.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 14. Перепишите следующие функции класса new_dekodes_chet (см. лаб. раб. № 10 вариант 14).

(ort) замените на оператор ^ (проверка взаимной ортогональности группы векторов);

(mult_1) замените на оператор * (умножение двоичной матрицы на скаляр (значение скаляра 0 или 1));

(mult_2) замените на оператор ~ (умножение двоичной матрицы на двоичный вектор);

Дополнительно переопределите следующий оператор:

Получении группы инверсных кодов (по правилу c[i]=i-a[i])- унарный оператор ^.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 15. Перепишите следующие функции класса new_ort_vec (см. лаб. раб. № 10 вариант 15).

(tran) замените на унарный оператор! (транспонирование двоичной матрицы);

(mult) замените на оператор * (перемножение двоичных матриц);

(mult_2) замените на оператор ^ (умножение двоичной матрицы на двоичный вектор);

Дополнительно переопределите следующий оператор:

Сравнение двух групп векторов – оператор ==.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 16. Перепишите следующие функции класса new_kod_nechet (см. лаб. раб. № 10 вариант 16).

(sogl) замените на оператор ~ (проверка на согласованость размерности матрицы и вектора для выполнения перемножения);

(mult_1) замените на оператор * (перемножение двоичного вектора на скаляр);

(mult_2) замените на оператор ^ (перемножение двоичного вектора на двоичную матрицу);

(mult_3) замените на оператор && (умножение двоичной матрицы на двоичный вектор);

Дополнительно переопределите следующий оператор:

Перестановка разрядов вектора в обратном порядке (по правилу c[i]=a[n-i+1]).

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 17. Перепишите следующие функции класса new_mas_ (см. лаб. раб. № 10 вариант 17).

(add) замените на оператор + (склеивание двух массивов строк);

(rshift) замените на оператор >> (циклический сдвиг вправо на заданное растояние массива);

(lshift) замените на оператор << (циклический сдвиг влево на заданное растояние массива);

Дополнительно переопределите следующий оператор:

Сравнение двух строк – оператор ==.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 18. Перепишите следующие функции класса new_mas_real (см. лаб. раб. № 10 вариант 18).

(add) замените на оператор + (слияние двух упорядоченных массивов в один упорядоченный);

(delete) замените на оператор / (удаление элемента из массива);

(mult) замените на оператор * (перемножение двух массивов по правилу скалярного произведения);

Дополнительно переопределите следующий оператор:

Вычисление суммы отрицательных элементов массива – оператор –.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 19. Перепишите следующие функции класса new_mas_random (см. лаб. раб. № 10 вариант 19).

(minus) замените на оператор – (вычитание двух массивов по правилу вычитания двух вукторов);

(delete) замените на оператор / (удаление элемента из массива);

(mult) замените на оператор * (перемножение двух массивов по правилу скалярного произведения);

Дополнительно переопределите следующий оператор:

Сравнение двух массивов – оператор ==.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

Вариант 20. Перепишите следующие функции класса new_num_16 (см. лаб. раб. № 10 вариант 20).

(minus) замените на оператор – (вычитание двух 16-х чисел);

(add) замените на оператор + (сложение двух 16-х чисел);

Дополнительно переопределите следующий оператор:

Перемножение двух шестнадцатеричных чисел – оператор *.

Программа должна выполнить данные операции с несколькими объектами. Результаты всех операций вывести в cout.

12.12 Создание списка объектов

Необходимо организовать линейный связанный список из структуры, указанной в задании. Со списком необходимо выполнить следующие действия:

- Чтение из файла и размещение в свободной памяти;
- Просмотр списка;
- Включение новой компоненты в заданное место списка. Варианты указания места включения:

a. В начало списка;

b. В конец списка;

c. Место указывается номером, который должна иметь компонента после включения.

- Удаление заданной компоненты. Варианты указания удаляемой компоненты;

a. Первая компонента;

b. Вторая компонента

c. Задается номер удаляемой компоненты;

- Удаление списка.

Выполнение операции над списком должно осуществляться в интерактивном режиме.

Замечание 1. В скобках задаются соответственно вариант включения и вариант исключения компоненты.

Замечание 2. В четных вариантах заданий необходимо описать операцию и удаления соответственно с помощью функции insert и delete, а в нечетных с помощью операторов + и -.

Вариант 1. Личность (имя, фамилия, адрес) (аа)

Вариант 2. Работник (номер бригады, фамилия, разряд) (аб)

Вариант 3. Студент (номер группы, фамилия, размер стипендии) (ав)

Вариант 4. Совхоз (название совхоза, засеваемая площадь, количество колхозников) (ба)

Вариант 5. Книга (название, автор) (бб)

Вариант 6. Преподаватель (фамилия, квалификация, стаж) (бв)

Вариант 7. Файл (название файла, количество байт) (ва)

Вариант 8. Программист (фамилия, отдел, язык программирования) (вб)

Вариант 9. Фирма (название, назначение, уставной капитал) (вв)

Вариант 10. Трамвай (номер трамвая, номер маршрута) (аа)

Вариант 11. Музыкант (муз инструмент, муз. жанр, муз. образование) (аб)
Вариант 12. Спорт (вид спорта, массовость) (ав)
Вариант 13. Конфликт (повод, причина, результат) (ба)
Вариант 14. Статья (автор, название, журнал, номер журнала, год выпуска) (бб)
Вариант 15. Оружие (калибр, тип, фирма изготовитель) (бв)
Вариант 16. Компьютер (марка, параметры, название) (ва)
Вариант 17. Животное (род, место обитания) (вб)
Вариант 18. Болезнь (инкубационный период, основной симптом) (вв)
Вариант 19. Наука (предмет исследования, методы исследования) (аа)
Вариант 20. Авиакомпания (название авиакомпании, капитал) (аб)

12.13 Производные классы

Необходимо организовать линейный связанный список из двух структур, указанных в задании. Со списком необходимо выполнить следующие действия:

- Чтение из файла и размещение в свободной памяти;
- Просмотр списка;
- Включение новой компоненты в заданное место списка. Варианты указания места включения:

a. В начало списка;
b. В конец списка;
c. Место указывается номером, который должна иметь компонента после включения.

- Удаление заданной компоненты. Варианты указания удаляемой компоненты;

a. Первая компонента;
b. Вторая компонента;
c. Задается номер удаляемой компоненты

- Удаление списка.

Выполнение операции над списком должно осуществляться в интерактивном режиме.

Замечание 1. В скобках задаются соответственно вариант включения и вариант исключения компоненты

Замечание 2. В четных вариантах заданий необходимо описать операцию и удаления соответственно с помощью функции insert и delete, а в нечетных с помощью операторов + и -.

Вариант 1. Личность (имя, фамилия, адрес) (аа)
Лидер (личность, название увлечения)
Вариант 2. Работник (номер бригады, фамилия, разряд) (аб)
Бригадир (работник, количество подчиненных)

Вариант 3. Студент (номер группы, фамилия, размер стипендии) (ав)
Староста (студент, число студентов в группе)

Вариант 4. Совхоз (название совхоза, засеваемая площадь, кол-во колхозников) (ба) Село (совхоз, ФИО председателя)

Вариант 5. Книга (название, автор) (бб)
Монография (книга, издательство)

Вариант 6. Преподаватель (фамилия, квалификация, стаж) (бв)
Зав. кафедрой (преподаватель, количество сотрудников)

Вариант 7. Файл (название файла, количество байт) (ва)
Индексный файл (файл, имя индекса)

Вариант 8. Программист (фамилия, отдел, язык программирования) (вб)
Ведущий программист (программист, число подчиненных)

Вариант 9. Фирма (название, назначение, уставной капитал) (вв)
Учредитель (фирма, число дочерних фирм)

Вариант 10. Трамвай (номер трамвая, номер маршрута) (аа)
Снегоочиститель (трамвай, число маршрутов)

Вариант 11. Музыкант (муз. инструмент, муз. жанр, муз. образование) (аб)
Гитарист (музыкант, число струн в гитаре)

Вариант 12. Спорт (вид спорта, массовость) (ав)
Футбол (спорт, тренер, судья)

Вариант 13. Конфликт (повод, причина, результат) (ба)
Война (конфликт, число жертв)

Вариант 14. Статья (автор, название, журнал, номер журнала, год выпуска) (бб)
Очерк (статья, герой очерка)

Вариант 15. Оружие (калибр, тип, фирма изготовитель) (бв)
Пулемет (оружие, скорострельность)

Вариант 16. Компьютер (марка, параметры, название) (ва)
Персональный компьютер (компьютер, цена)

Вариант 17. Животное (род, место обитания) (вб)
Кошка (животное, имя)

Вариант 18. Болезнь (инкубационный период, основной симптом) (вв)
Грипп (болезнь, осложнение)

Вариант 19. Наука (предмет исследования, методы исследования) (аа)
Химия (наука, ФИО ученого)

Вариант 20. Авиакомпания (название авиакомпании, капитал) (аб)
Страна (авиакомпания, название страны)

ПРИЛОЖЕНИЕ

Таблица 1

Приоритеты операций

№ группы	Обозначение операции	Название операции
1.	( )	вызов функции
	[ ]	выделение элементов массива
	.	выделение элемента структуры или объединения
	→	выделение элемента структуры или объединения, адресуемого указателем
2.	!	логическое отрицание (не)
	~	побитовое отрицание
	−	изменение знака
	++	инкремент (увеличение на единицу)
	−−	декремент (уменьшение на единицу)
	&	определение адреса
	*	обращение по адресу
	(тип)	преобразование типа
	sizeof	определение размера в байтах
3.	*	умножение
	/	деление
	%	взятие остатка (процент)
4.	+	сложение
	−	вычитание
5.	<<	сдвиг влево
	>>	сдвиг вправо
6.	<	меньше
	<=	меньше или равно
	>	больше
	>=	больше или равно
7.	==	равно в логических выражениях (if a==b)
	!=	не равно
8.	&	побитовая операция «и»
9.	^	побитовая операция «исключающее или»
10.	¦	побитовая операция «или»
11.	&&	логическая операция «и»
12.	¦¦	логическая операция «или»
13.	?:	условная операция (if then else)
14.		Операции присваивания
	=	
	*=	
	/=	

Окончание таблицы 1

1	2	3
	+=	
	−=	
	%=	
	<<=	
	>>=	
	&=	
	^=	
	¦=	

Операции с одинаковым приоритетом выполняются слева направо, кроме операций присваивания, они выполняются справа налево.

Таблица 2

Некоторые математические функции

Обозначение	Тип аргумента	Название
abs(x)	int	модуль
fabs(x)	double	модуль
labs(x)	long	модуль
cos(x)	double	cosx
sin(x)	double	sinx
tan(x)	double	tgx
sqrt(x)	double	$\sqrt{x}$
pow(x,y)	x,y-double	x^y
ceil(x)	double	округление в сторону увеличения
floor(x)	double	округление в сторону уменьшения
fmod(x,y)	x,y-double	определение остатка от деления x на y
exp(x)	double	e^x
log(x)	double	ln(x)
log10(x)	double	lg(x)
modf(x,y)	double	разделение числа на целую и дробную части

БИБЛИОГРАФИЧЕСКИЙ СПИСОК:

1. Керниган Б., Язык программирования Си. Задачи по языку Си. / Б. Керниган, Д. Ритчи. – М. : ФиС, 1985. – 280 с.
2. Рассохин, Д. От Си к Си++. / Д. Рассохин. – М.: Эдэль, 1993 – 128 с.
3. Страуструп, Б. Язык программирования Си++. – М.: Радио и связь, 1991. – 352 с.
4. Белецкий, Я. Энциклопедия языка Си. / Я. Белецкий. – М.: Мир, 1992. – 687 с.
5. Белецкий, Я. Турбо Си++: Новая разработка: учебное пособие для студентов высших учебных заведений. / Я. Белецкий. – М.: Машиностроение, 1994. – 400 с.
6. Пильщиков, В.Н. Сборник упражнений по языку Паскаль: учебное пособие для вузов. / В. Н. Пильщиков. . – М.: Высш. Шк., 1990. – 223 с.

Учебное издание
РОДИОНОВА Татьяна Евгеньевна
КУРС ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ СИ
Учебное пособие
Редактор Виничук О. В.
Подписано в печать 28.12.2006. Формат 60×Р4 /16.
Бумага офсетная. Печать трафаретная. Усл. печ. л. 6,64.
Тираж 100 экз. Заказ .
Ульяновский государственный технический университет
432027, г. Ульяновск, ул. Сев. Венец, д. 32.
Типография УлГТУ, 432027, г. Ульяновск, ул. Сев. Венец, д. 32.