Лабораторная работа № 1

Изучение интегрированной среды C
(2 часа)

***Цель работы*:** научиться использовать интегрированную среду **C.**

Теоретические сведения

Вид интегрированной среды и ее возможности зависят от типа и версии компилятора.

Пример

```
/*    ЗАНЯТИЕ N 1
   Разработал Петров Ю.В.  */

/* <-Это начало многострочного комментария
   Это окончание многострочного комментария ->   */

// <-Это однострочный комментарий

#include <stdio.h> //Директива препроцессора (#include) включает
#include <conio.h> //Заголовочные файлы с расширением (.h)

void main(void)    //Главная функция
 {          //Начало составного оператора (блока, тела функции)
  clrscr();   //Функция очистки экрана
  printf("\n\t Здравствуй,\n");  //Функция вывода на экран
  printf("\n          мир!");  // '\n' '\t' -управляющие
                               //          последовательности
 }          //Конец составного оператора (блока, тела функции)

  /* Результат выполнения программы

    Здравствуй,
      мир!                    */
```

Ход работы

1. Выполнить загрузку интегрированной среды разработки **C (IDE)** для **Turbo C,** расположенной в каталоге **N:\APL\TC\BIN\TC.EXE**, из активной директории.
2. Изучить особенности **IDE** (структуру меню и подменю), а также повторить возможности текстового редактора (типа **Brief**), изучить «горячие» клавиши.
3. Выполнить загрузку программ-примеров **(N:\APL\TC\LAB*.CPP),** их корректировку, сохранение и компиляцию.

4 Изучить непонятные синтаксические конструкции с использованием встроенной помощи. Скопировать пример из помощи в активное окно и изучить его работу.
5 Повторить выполнение пунктов 1-4 для интегрированной среды разработки **Borland C (N:\APL\BC\BIN\BC.EXE).**
6 Сравнить возможности **IDE Turbo C** и **Borland C.**
7 Написать отчет и сделать выводы по работе, изучив контрольные вопросы по теме.

Требования к отчетам

Отчет по лабораторным работам оформляются в соответствии с методическими указаниями “Структура и правила оформления текстовых документов” на основе ДСТУ 3008.95 “Документация, отчёты в сфере науки и техники. Структура и правила оформления”. Отчеты оформляются на отдельных листах формата А4. В конце семестра отчеты сшиваются с титульным листом и сдаются на кафедру после защиты всех работ.

Отчёт должен содержать

1 Фамилию, имя, отчество и группу студента
2 Номер и название работы
3 Цель работы
4 Индивидуальное задание
5 Текст программы
6 Результаты работы программы
7 Выводы

Вопросы для контроля и самостоятельной работы

1 Как осуществляется запуск и выход из **IDE?**
2 Как осуществляется настройка путей для подключения внешних файлов?
3 Перечислите режимы компиляции. Что они обозначают?
4 Как производится подключение библиотеки графических функций?
5 Как осуществляется открытие и закрытие окон, переход между окнами, копирование текста из окна в окно?
6 Как осуществляется модификация имени файла?
7 Что такое проект? Как проиизводится создание, дополнение и выполнение проекта?
8 Почему загрузка IDE осуществляется из активной директории?
9 Назовите «горячие » клавиши и их назначение.
10 Как выделить комментарии в **C?**
11 Как подключиться заголовочные файлы, что они содержат?
12 Что обозначается словом **main()?**

Лабораторная работа №2

Функции ввода/вывода printf(), scanf(). Линейные вычислительные процессы (2 часа)

Цель работы: Изучить форматы объявлений и работу основных функций ввода/вывода информации. Научиться составлять простые программы с линейным вычислительным процессом.

Теоретические сведения

Одной из основных задач при программировании является ввод и вывод данных. В **C** для этого применяют ряд функций **– printf(), scanf(), cprintf(), ecvt(), fprintf(), fread(), fscanf(), hutc(), puts(), putw(), sprintf(), vprintf().** Функция **printf()** осуществляет форматированный вывод в поток **stdout.** Объявление функции находится в заголовочном файле **<stdio.h>.**

Синтаксис объявления функции

printf() #includ <stdio.h> printf (const char* format [,argument,…]);

В скобках **[ ]** указаны необязательные элементы спецификации.
Спецификация формата, определяющая вывод аргументов, имеет вид:

%[flags] [width] [.precision] [F/N/h/l/L] type

Функция **scanf()** - Осуществляет форматированный ввод из потока **stdin**
Синтаксис **#include <stdio.h>**

int scanf(const char *format[,adress, ...]);

Неотображаемыми символами являются пробел **( )**, символ табуляции **(\t)**, символ перехода на новую строку **(\n**) и другие управляющие последовательности. Если функция **scanf()** встречает неотображаемый символ в форматной строке, она будет считывать, но не сохранять всю последовательность символов вплоть до следующего отображаемого символа во входном потоке.

Отображаемыми символами являются все другие символы кода **ASCII**, за исключением символа процента **(%).** Если функция **scanf()** встречает в строке формата отображаемый символ, то она прочитает, и сохранит соответствующий ему символ.

Спецификации формата предписывают функциям **scanf()** осуществить чтение и преобразование символов из входного поля в значения определенного типа, затем запомнить их в память по адресу, указанному соответствующим адресным аргументом. Завершающий (последний) неотображаемый символ не читается (включая символ перехода на новую строку), если только он не описан явно в форматной строке.

Спецификация формата функции …**scanf()** имеет следующий вид

% [*] [widht] [F|N] [h|l|L] <type>

Спецификация формата начинается с символа процента **(%).** После этого символа следуют символы спецификации. Ниже представлено общее описание строки формата **scanf()**, управляющей формированием потока данных.

Символ или спецификатор	Чем управляет или что определяет
***-** подавление назначения **width** – ширина поля **Size** N, F – модификаторы размера указателя h, l, l, L – модификаторы типа аргумента	Отменяет присваивание следующего поля ввода Максимальное число считываемых символов Изменяет размер по умолчанию адресного аргумента Указатель типа **near,** Указатель типа **far;** Тип **short int**; Тип **long int** (если символ типа указывает на преобразование к целому типу); Тип **double** (если символ типа указывает на преобразование к типу с плавающей запятой); Тип **long double** (допустим только при преобразованиях к типу с плавающей запятой)

Символы типа

Символ	Ожидается на входе	Тип аргумента
d	Десятичное целое	Указатель на **int (int *arg)**
D	Десятичное целое	Указатель на **long (long *arg)**
o	Восьмеричное целое	Указатель на **int (int *arg)**
O	Восьмеричное целое	Указатель на **long (long *arg)**
i	Десятичное целое	Указатель на **int (int *arg)**
I	Десятичное целое	Указатель на **long (long *arg)**
u	Десятичное целое без знака	Указатель на **unsigned int (unsigned int *arg)**
U	Десятичное целое без знака	Указатель на **unsigned long (unsigned long *arg)**
x	Шестнадцатеричное целое	Указатель на **int (int *arg)**
X	Шестнадцатеричное целое	Указатель на **long (long *arg)**
e, E	Число с плавающей запятой	Указатель на **float (float *arg)**
f	Число с плавающей запятой	Указатель на **float (float *arg)**
g, G	Число с плавающей запятой	Указатель на **float (float *arg)**
s	Строка символов	Указатель на массив символов **(char arg[ ])**
c	Символ	Указатель на символ **(char *arg)**

Пример

```
/*    ЗАНЯТИЕ N 2
   Разработал Сидоров К.В.  */

#include <stdio.h> //Директива препроцессора (#include) включает
#include <conio.h> //Заголовочные файлы с расширением (.h)
#include <string.h>
   int a;       //Объявление глобальной переменной типа int
int main(void)  //Главная функция
  {//Начало составного оператора (блока, тела функции)
   char c,buf[20];       //Объявление локальных переменных
   char *pst="\"slovo\"";//Объявление локальной переменной с
   float f=26.6;         //инициализацией
   double d;
   clrscr();             //Функция очистки экрана
   printf(" Ввод переменной типа char:   "); //Функция вывода на экран
   fflush(stdin);        //Функция очистки буфера клавиатуры
```

```
  scanf("%c",&c);       //Функция ввода данных,
                        // & - операция взятия адреса
  printf(" Вывод переменной типа char:  ");
  printf("%c\n",c);
  printf("\n Ввод переменной типа int:  ");
  scanf("%d",&a);
  printf(" \t Вывод переменной типа int\n");
  printf(" Формат вывода (int): +6d     #6o     #8x\n");
  printf("\t\t     |%+6d|%#6o|%#8x|\n ",a,a,a);
  printf("\n Ввод переменной типа int:  ");
  scanf("%d",&a);
  printf("\t Вывод переменной типа int\n");
  printf(" Формат вывода (int): -6d     +6d     #8d\n");
  printf("\t\t     |%-6d|%+6d|%#6d|\n",a,a,a);
  printf("\n Вывод исходной строки:  %s\n\n",pst);
  printf(" Ввод строки в массив:    ");
  scanf("%s",buf);
  printf(" Вывод строки из массива: %s\n\n",buf);
  printf(" Ввод переменных типа float and double (через пробел):\n");
  printf("\t\t      ");
  scanf("%f %lf",&f,&d);
  printf("\t Вывод переменных типа float and double\n");
  printf(" Формат вывода (float):  10.6f       10.6e       +10.6g\n");
  printf("\t\t     |%10.6f|%10.6e|%+10.6g|\n",f,f,f);
  printf(" Формат вывода (double): 10.8lf      10.8e       10.8g\n");
  printf("\t\t     |%10.8lf|%10.8e|%+10.8g|\n ",d,d,d);
  getche(); //Функция ввода символа с клавиатуры
  return 0; //Оператор возврата значения из функции (0)
 }          //Конец составного оператора (блока, тела функции)

/*  Результат выполнения программы

 Ввод переменной типа char:   u
 Вывод переменной типа char:  u
 Ввод переменной типа int:    78
          Вывод переменной типа int
 Формат вывода (int):      +6d     #6o     #8x
                           |   +78|  0116|    0x4e|
 Ввод переменной типа int:    90
         Вывод переменной типа int
 Формат вывода (int):      -6d     +6d    #8d
                           |90    |   +90|    90|
 Вывод исходной строки:  "slovo"
 Ввод строки в массив:    символы 45!"#:$.;?%;?
 Вывод строки из массива: символы 45!"#:$.;?%;?
 Ввод переменных типа float and double (через пробел):
```

```
                    78.89  6778.0
          Вывод переменных типа float and double
Формат вывода (float):   10.6f       10.6e        +10.6g
                     | 78.889999|7.889000e+01|    +78.89|
Формат вывода (double):  10.8lf      10.8e         10.8g
                     |6778.00000000|6.77800000e+03|     +6778|  */
```

Ход работы

1 Изучить теоретические сведения.
2 Выполнить загрузку интегрированной среды разработки **C (IDE)** для **Borland C,** расположенной в каталоге **N:\APL\BC\BIN\BC.EXE**, из активной директории.
3 Ознакомиться с форматом функций **printf()** и **scanf().**
4 Скопировать примеры для функций **printf, scanf** из встроенной помощи в активное окно и изучить их работу. Изучить синтаксические конструкции, приведенные во встроенной помощи.
5 Выполнить загрузку программ-примеров **(N:\APL\TC\LAB*.CPP),** их корректировку с использованием различных возможностей функций **printf (), scanf (),** сохранение файлов и компиляцию.
6 Написать отчет и сделать выводы по работе.
7 Подготовиться к защите лабораторной работы, изучив вопросы по данной теме, изучив контрольные вопросы по теме.

Индивидуальное задание к лабораторной работе №2

Составить программу для форматированного ввода и вывода данных заданного типа согласно индивидуальному заданию приведенному в таблице 2.1.

Таблица 2.1 - Индивидуальное задание

вариант	первый тип	второй тип	третий тип	четвертый тип	выравнивание по краю	Точность вещественных типов
1	unsigned int	long int	float	double	левый	14.5
2	signed int	long double	char	short int	правый	12.8
3	unsigned int	unsigned long int	short int	float	правый	10.3
4	long int	char	double	float	левый	11.2
5	unsigned long int	float	int	long double	правый	7.3
6	signed long int	long double	unsigned int	float	правый	16.5
7	short int	long double	float	unsigned int	левый	11.5
8	unsigned long int	float	long int	char	правый	20.9

9	float	signed int	long double	char	левый	13.6
10	long int	float	double	char	левый	14.3
11	char	signed long int	long double	float	правый	9.6
12	float	int	long double	unsigned long int	левый	8.2
13	char	unsigned long int	float	long int	левый	12.4
14	float	signed long int	long double	signed long int	правый	15.7
15	long double	short int	float	unsigned long int	правый	17.5
16	long double	unsigned long int	int	char	левый	14.2
17	float	short int	char	long double	правый	10.5
18	unsigned long int	short int	long int	char	левый	11.7
19	char	double	long double	unsigned int	левый	16.12
20	float	int	double	char	левый	10.3
21	long double	unsigned int	double	float	правый	9.5
22	long double	float	long double	unsigned long int	левый	13.6
23	float	long int	long double	unsigned long int	левый	12.4
24	float	long int	char	unsigned long int	правый	13.8
25	signed int	long double	float	unsigned short long	правый	10.6
26	float	double	char	long int	левый	12.5
27	signed long int	long double	float	char	правый	11.4
28	int	long double	unsigned long int	float	левый	10.3
29	unsigned long int	float	long int	char	левый	15.6
30	signed char	long double	unsigned char	short int	левый	18.10

Требования к содержанию отчёта приведены в лабораторной работе №1.

Вопросы для контроля и самостоятельной работы

1 Как осуществляется вывод и ввод информации в языке? Существуют ли встроенные операторы ввода/вывода?
2 Для чего используется форматированный ввод/вывод?
3 С какого символа начинается форматная строка?
4 Для чего устанавливаются флаги?
5 Какие символы преобразования используются и в каких случаях?
6 Какие элементы формата являются обязательными элементами?
7 В каком заголовочном файле приведены объявления (прототипы) этих функций?
8 Отличаются ли символы типа **<type>,** применяемые для функций **prinf()** и **scanf()**?
9 Отличаются ли типы аргументов функций? Если отличаются, то в чём различие и чем оно вызвано?
10 О чём свидетельствует модификатор **const**?
11 Назовите функции для ввода/вывода информации.

Лабораторная работа №3

Разработка программ со скалярными типами данных (2 часа)

Цель работы: Рассмотреть и изучить скалярные типы данных **C** (**int, char, float** и др.) и их использование.

Теоретические сведения

В **C** переменные должны быть объявлены, т.е. их тип специфицирован до того, как эти переменные будут использованы. Объявления переменных могут быть сделаны в любом месте программы. При объявлении переменных применяется префиксная запись, при которой вначале указывается тип, а затем - имя переменной.

Объявления переменных могут быть глобальными с классами памяти **extern** и **static** (вне функции **main(),** по умолчанию **extern**), и локальными с классами памяти **auto, register** (внутри блока, функции, например, внутри функции **main(),** по умолчанию **auto**).

Глобальные переменные инициализируются нулевым значением по умолчанию.

По умолчанию предполагается, что переменные являются знаковыми. Беззнаковые переменные описываются явно при помощи спецификатора **insigned**

Например:

```
int n1, n2, n3, n4;      // множественное объявление переменных
n1=15; float  weight =23.56;  // объявление переменных и их инициализация
unsignid int exam;  char ch ='+', c='A';
char slash='/',Slash='\';      // регистр у переменных различается slash не Slash
char str[ ]=''строка символов'';  // инициализация строковой константой
char *pstr=''строка символов'';
```

В первом объявлении имеется список переменных, содержащих несколько имен (**n1,n2,**...). Все они имеют тип (**int**) (целое). Переменная **exam** целого типа, беззнаковая и объявлена отдельно. Если используются спецификаторы **unsigned, long, short** то спецификатор **int** можно опускать, так как переменные имеют тип **int** по умолчанию.

В **C** имеется множество предопределенных типов данных, включая несколько видов целых, указателей, переменных с плавающей точкой, массивов, объединений, структур и тип **void** (пустой). Слово **void** означает, например, что указатель может указывать на любой тип, функция не получает параметров или не возвращает значение. Скалярные типы включают символьные типы, целые, плавающие типы, указатели, ссылки и перечисления. Целые типы включают несколько разновидностей целых и символьных данных. Арифметические типы объединяют целые и вещественные (плавающие). Символьные типы: отдельные символы, литеральные строковые константы (символьные строки, символические константы). Строковые константы всегда заканчиваются нулем. В строковых константах могут быть управляющие последовательности (**\n, \t,** \\ и др.), которые допускаются везде, где могут быть печатные символы.

’’c:\\bc\\test\\f.c’’

Составные типы - включают в себя массивы, структуры и объединения.

Каждый объект заданного типа занимает определенное число единиц памяти. За единицу принимается один байт. Число единиц памяти, требуемое для размещения элемента данного типа, может быть вычислено с использованием операции **sizeof(<тип>)** или **sizeof(<объект_типа>)**. Ниже приведена таблица, показывающая основные типы данных, их размер и диапазон значений.

Таблица 3.1 - Размер и диапазон значений переменных различных типов

Тип	**Размер в байтах**	**Диапазон значений**
signed char	**1**	**128...127**
unsigned char	**1**	**0...255**
signed int	**2**	**-32768...32767**
unsigned int	**2**	**0...65535**
signed short		аналогично **int**
unsigned short		аналогично **unsigned int**
signed long	**4**	**2147483648…2147483647**
unsigned long	**4**	**0...4294967295**
float	**4**	**-3.4e38...3.4e38**
double	**8**	**-1.7e308...1.7e308**
long double	**10**	**-1.7e308...1.7e308**

Пользователь имеет возможность связать объявление типа данных с более простым и наглядным новым именем для этого типа данных (назначить псевдоним), используя средство #**typedef** с форматом **typedef <объявление типа> <псевдоним>.**

Например, определим новое имя для типа **unsigned int**

typedef unsigned int NATUR; // unsigned int связывается с именем –
// псевдонимом **NATUR**
NATUR i; //объявляем переменную **i** типа **insigned int**

Для запрета изменения значений переменных используется ключевое слово-модификатор **const.**

const int value16 = 0xf9ac; const unsignid int value8 = 0234; // переменные в // шестнадцатеричном **(∅Xddd***)* и восьмеричном **(∅ddd**) виде.

Для определения именованных констант и макроопределений (макросов) используют директиву препроцессора:

#define <ИМЯ> (argument 1[, argument 2…]) <подставляемый текст>

Директива создает временные имена (именованные константы), которые препроцессор заменяет на подставляемый текст, поэтому имена не являются объявлениями переменных.

#define STRING ’’\a\’’Hello, Wold ! \’’\n’’

#define HEX 0x9b
#define DECIMAL 155

Глобальные переменные **(extern, static)** инициализируются нулевым значением по умолчанию.

Локальные переменные **(auto, register)** не инициализируются по умолчанию.

Пример

```
/*   ЗАНЯТИЕ N 3
  Разработал .....
Выполнить объявления типов, переменных с различными классами
памяти, констант. Ввести исходные значения переменных,
вычислить значения переменных и вывести результаты расчетов.
Использовать макроопределение для выбора наибольшего значения*/
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>
enum en {NACH=0, N=10, MIN=1000};//Вместо следующих директив:
                //(#define NACH 0) (#define N 10) (#define MIN 1000)
typedef int norm;     //Объявление типа norm -синонима int
float x;              //Объявление глобальной переменной (extern)
void main(void)
{clrscr();
norm i=NACH,j=NACH;   //i,j-переменные типа int (norm)
static float min=MIN; //Объявление глобальной переменной (static)
auto float y=N, k=0;   //Множественное объявление переменных (auto)
float a,b,c,d;         //Множественное объявление переменных (auto)
printf("Значения переменных после инициализации:\
\n\t x\t y\t min\t k\t i\t j\n");
printf("%8.5f %8.5f %8.5e %8.5f %3d %3i\n",x,y,min,k,i,j);
printf("Введите значения переменных типа float:\n\ta, b, c, d:\n\t");
scanf("%f %f %f %f",&a,&b,&c,&d);
printf("Значения переменных после ввода с клавиатуры,\n\t a\t \
b\t c\t d\n");
printf("%8.5f %8.5f %8.5e %8.5f\n",a,b,c,d);
x=NACH+N;
y=a*x*x*x+b*x*x+c*x+d;
min=y+i*pow(y,2);          //функция Y^2
k=x*sin(j);
x+= 0.1;                    //x=x+0.1;
printf("\n ЗНАЧЕНИЕ Y=%f",y);
printf("\n ЗНАЧЕНИЕ min=%10.5e",min);
printf("\n ЗНАЧЕНИЕ x=%10.5f",x);
printf("\n ЗНАЧЕНИЕ k=%3.1f\n ",k);
```

```
//---------------------------------------
#define MAX(x,y) ((x)<(y)?(y):(x)); //Макроопределение MAX
//если (x)<(y), то значение результата -y, если (x)>(y), то -x
 float af,bf,rez;  //Множественное объявление переменных (auto)
 printf("    Введите через пробел 2 числа:");
 scanf("%f %f",&a,&bf);
 rez=MAX(af,bf);     //af, bf-фактические параметры
 printf("Максимальное число: %4.2f ",rez);
 getche();
}
/*Значения переменных после инициализации:
      x            y            min             k          i       j
 0.00000   10.00000   1.00000e+05   0.00000   0      0
 Введите значения переменных типа float:
         a,  b,  c,  d:
         3   4   5   6
 Значения переменных после ввода с клавиатуры,
      a           b           c                   d
 3.00000   4.00000   5.00000e+00    6.00000
 ЗНАЧЕНИЕ Y=3456.000000
 ЗНАЧЕНИЕ min=3.45600e+03
 ЗНАЧЕНИЕ x=  10.10000
 ЗНАЧЕНИЕ k=0.0
 Введите через пробел 2 числа:        23.4 56.89
 Максимальное число: 56.89                                  */
```

Ход работы

1. Изучить основные типы данных языка **C**, директивы процессора.
2. Разработать программу с использованием переменных различных типов. Индивидуальное задание приведено в табл.2. При объявлении переменных использовать **typedef** . Создать и использовать именованные константы. Для ввода и вывода значений использовать функции форматированного ввода-вывода **scanf()** и **printf()** с представлением значений в десятичном, шестнадцатеричном и восьмеричном виде**,** а также **getchar(), getche(), putchar()** и др.
3. Для переменных разного типа определить их размер в байтах и вывести значения на экран дисплея.
4. Использовать управляющие последовательности (эскейп последовательности) различного типа при выводе сообщений на экран.
5. К переменным, указанным в индивидуальном задании, добавить переменные других типов.
6. Оформить отчет и сделать выводы о проделанной работе, изучив контрольные вопросы по теме.

Индивидуальное задание к лабораторной работе №3

Составить программу для хранения и обработки информации включающей различные типы данных. Индивидуальное задание приведено в таблице 3.2.

Таблица 3.2 - Индивидуальное задание

Вариант	Номер и содержание данных						
	1	2	3	4	5	6	7
1	ФИО	Рост	Вес	Год рождения	Пол	Рейтинг	…
2	Название ЭВМ	Тип процессора	Объем памяти	Тип дисплея	Количество дисководов	Стоимость	…
3	Тип автомобиля	Цвет	Количество колес	Количество мест	Грузоподъемность	Стоимость	…
4	Тип автобуса	Количество мест	Грузоподъемность	Номер маршрута	Пункт назначения	Время отправления	…
5	ФИО	Номер школы	Класс	Средний бал аттестата	Любимый предмет	Нелюбимый предмет	…
6	Название магазина	Вид товара	Адрес	Время работы	Количество продавцов	Номер магазина	…
7	ФИО	Вид спорта	Личный рекорд	Иностранный язык	Срок занятий	Количество знакомых слов	…
8	Название фирмы	Объем годового оборота	ФИО директора	Штат	Стаж работы	Возраст директора	…
9	ФИО	Наличие братьев и сестер	Число	Месяц	Год рождения	Вес	…
10	Название книги	Автор	Издательство	Дата издания	Страна	Количество страниц	…
11	ФИО	Номер в группе	Название группы	Курс	Оценки	Рейтинг	…
12	Название велосипеда	Количество колес	Диаметр колес	Цвет	Грузоподъемность	Скорость	…
13	Название программного продукта	Область применения	Объем занимаемой памяти	Операционная система	Режим: текстовый или графический	Стоимость	…
14	Название	Дата соз-	Стиль	Состав	Количество	Стоимость	…

Вариант	Номер и содержание данных						
	1	2	3	4	5	6	7
	рок группы	дания		группы	альбомов	билета	
15	ФИО	Номер зачетной книжки	Любимый предмет	Оценки по математике	средний бал	Язык программирования	…
16	Название журнала	Возраст читателей	Количество страниц	Начало издания	Тираж	Подписной индекс	…
17	Название самолета	Дальность полета	Количество мест	Количество двигателей	Время вылета	Время в воздухе	…
18	Название утюга	Цена	Страна производитель	Вес	Температура	Наличие регулятора	…
19	Город	Страна	Область	Почтовый индекс	Число жителей	Площадь	…
20	Операционная система	Многозадачность	Объем памяти	Версия	Фирма разработчик	Стоимость	…

Требования к содержанию отчёта приведены в лабораторной работе №1.

Контрольные вопросы для подготовки и самостоятельной работы

1 Что означают символы**: ’\r’, ’\f ’, ’\b’, ’\a’, ’\n’, ’\\’, ’\’’ ’, ’\’ ’, ’\000’, ’\0x00’, ’\0x0a’, ’\0x5c’?**

2 Какие переменные относятся к скалярным, арифметическим, составным, символьным ?

3 Почему в строке **’’c:\\bc\\test\\f.c’’** знак \ повторяется два раза ?

4 Что означает ключевое слово **signed** ?

5 Что означает ключевое слово **#define,** для чего используется **?**

6 Что означает ключевое слово **const,** для чего используется **?**

7 Что означает ключевое слово **typedef,** для чего используется**?**

8 Что означает ключевое слово **#include,** для чего используется **?**

9 Правильно ли выражение **#include ’’c:\bc\test\my.h’’?**

10 Что означают записи: **1254L, 34l, 0xf9ac, 0875, 76678UL ?**

11 Какой размер в байтах имеют базовые типы **Си** ?

12 Где можно объявить переменную? Какой тип имеет переменная по умолчанию?

13 Что означает множественное объявление переменных **?**

14 Как определить размер типа в байтах ?

15 Происходит ли инициализация переменной по умолчанию? В каких случаях?
16 Назовите класс памяти переменной по умолчанию. Зависит ли он от места объявления?

Лабораторная работа №4

Разработка программ с циклическими вычислительными процессами

(2 часа)

Цель работы: Изучить написание программ на языке **C**, используя итерационные (циклические) методы, освоить основные операторы, поддерживающие работу с циклами **(for, while, do... while)**. Научиться писать программы, используя данные операторы.

Теоретические сведения

Каждый оператор **C** заканчивается оператором **(;)**, который является пустым оператором. Циклы или итерационные процедуры позволяют выполнять отдельные операторы или блоки операторов (составные операторы **{…}**). Число повторов определяется выражением в скобках, значение которого сравнивается с ∅ (∅-прекращение цикла). Циклы бывают с проверкой значения выражения перед началом выполнения тела цикла (с предусловием), по окончанию выполнения тела (с постусловием) или внутри тела цикла. В **C** определено три вида операторов цикла: оператор цикла с предусловием - **while**, оператор цикла с постусловием **do... while** и оператор цикла **for.** Рассмотрим эти операторы детально:

Цикл **while** имеет следующий формат:

while (<выражение>) <оператор>

Условие выполнения итерации в цикле **while** предварительно проверяется и в случае истинности выражения (не равно ∅) выполняются те операторы, которые описаны в теле цикла. В противном случае цикл заканчивается, и программа продолжает свое выполнение с того места, где закончилось тело цикла. Если тело цикла содержит несколько операторов, оно ограничивается операторными скобками: **"{"** и **"}"** и является составным оператором.

Прервать выполнение цикла можно, используя операторы **break**, **goto, return.** Ниже продемонстрировано два примера работы с циклом **while** в виде фрагментов программ.

Первый пример демонстрирует применение операторов **while** и **break,** перекрывающего цикл. Второй пример показывает использование цикла **while** для проверки ответа, который вводится с клавиатуры.

Пример 1

```
…
int i=10,k,s =∅;
printf ("ввести шаг приращения к: ");
scanf("%i",&k );
while (i)      //выполнять цикл, пока i не
// равно ∅
{ s+=k;        // увеличить s на k
i- - ;         // уменьшить i на 1
if (i=k) break;  //выйти, если выполнится
// условие: i=k
}...
```

Пример 2

```
...
char ch =’a’;
do          // начать цикл do while
{ printf("\n  Отвечайте yes или no (y/n):");
scanf(ch!=’y’ &&ch!= 'n');  // выполнять
цикл до тех пор, пока не будет нажата
// буква ‘y’ или ‘n’
```

Оператор цикла **do while()** является циклом с постусловием. В этом случае тело цикла всегда выполнится хотя бы один раз.

Формат оператора цикла следующий:

do <оператор>
while(<выражение>);

Телом цикла может являться один или несколько операторов (составной оператор). Ниже приведены аналогичные примеры реализации цикла с постусловием.

Пример 1

```
int i=10, k, s=∅;
  printf ("ввести шаг приращения к: ");
scanf("%i", &k);
  do                  // начать цикл do while
  { s+=k;          //увеличить s на k
    i-- ;                    //уменьшить i
    if (i=k) break; //выйти, если
// выполнится условие: i= =k
  while (i);  // выполнять цикл, пока i
// не равно ∅
```

Пример 2

```
char ch;
  do                  //начать цикл do while
printf ("\n Отвечайте yes или no (y/n):");
scanf("%c",&ch);
  while (ch!='y' && ch!='n'); // выполнять
//цикл до тех пор, пока не будет нажата
//буква 'y' или 'n'
```

Оператор цикла **for** является наиболее удобным и мощным средством организации циклических вычислений в **C**.

Формат цикла **for:**

for([<выражение1>];[<выражение2>];[<выражение3>]) <оператор>

где : <**выражение1>** - производит инициализацию тех переменных, которые будут непосредственно изменяться в цикле, в частности, задаётся начальное значение переменной-счётчика

<выражение2> - определяет условие выхода из цикла. При равенстве его значения ∅ цикл прерывается;

<выражение3> - это выражение задает изменение на каждом шаге переменных, проинициализированных в **<выражении1>.** Можно задавать также изменение других переменных, не связанных с **<выражением1>**.

Каждое из описанных выражений может отсутствовать. Оператор цикла **for,** заданный как **for(;;){<тело_цикла>},** является бесконечным циклом.

***Алгоритм работы оператора цикла* for ():**

1 Вычисляется первое выражение (если оно присутствует).
2 Вычисляется второе выражение (если оно присутствует), проверяется условие окончания цикла**:(<выражение 2>= =∅ “ Ложно ”)** т.е. Если оно не равно ∅ **(“Истинно”)** – происходит переход к пункту **3**, иначе выход из цикла.
3 Исполняется тело цикла.
4 Вычисляется третье выражение, если оно присутствует.
5 Переход к пункту **2.**

Появление в любом месте тела цикла оператора **continue** - приводит к прекращению выполнения операторов в теле цикла 4 немедленному переходу к шагу **2.**

Выход из цикла возможен аналогично **while** с помощью операторов **break, goto, return**.

Пример 1 (вариант 1)

```
#include <stdio.h>
 void main (void)
{// Программа выводит четные числа
//в диапазоне от 100 до 0, в порядке
// убывания.
 For (int i=100;i>=0;i-=2)
printf("\n%d",i);
 }  i - =2
```

(вариант 2)

```
f r (i=100; i>=0; i--)
if (i%2= =1) continue;  o // оператор
// continue прерывает текущую
// итерацию
else printf("\n%d",i);
```

Для демонстрации гибкости оператора **for** перепишем пример в следующем виде (**вариант 3):**

```
for(int i=100;i>=0; printf("\n%d",i), i - =2);
```

Пример

```
/*    ЗАНЯТИЕ N 4
  Разработал .....
 Строка символов записывается в обратном порядке с
 применением различных операторов организации циклов */
        #include <stdlib.h>
        #include <stdio.h>
        #include <conio.h>
        #include <math.h>
        # define N 23
        main()
        {int t,b;
         char string[]="\"Инициализация строки\"";
         char m;
         clrscr();
    //  Цикл for -с предусловием
         printf("Вывод исходной строки\n");
         for (t=0;t<N;t++)
         printf("%c ",string[t]);
         for (t=0,b=N-2;t<(N-1)/2;t++,b--)
         {m=string[t];
          string[t]=string[b];
          string[b]=m;
         }
         printf("\nВывод строки в обратном порядке\n");
         for (t=0;t<N;t++) printf("%c ",string[t]);
         printf("\nВвод новой строки -y, нет-n:\t");
```

```
        m=getche();
        if(m!='n')
new_str: scanf("%s",string);
        else      printf("\n");
  //   Цикл while -с предусловием
        t=0;
        while(string[t]!='\0')
         {printf("%c",string[t]);
          t++;
         }
         printf("\nКоличество символов =%2d\n",t);
  //    Цикл d0...while  -с постусловием
        printf("Вывод строки в обратном порядке\n");
        b=t-1;t=0;
        do
        {m=string[t];
         string[t]=string[b];
         string[b]=m;
         t++;b--;
        }while(t<b);
        printf("%s ",string);
        printf("\nВвод новой строки -y, нет-n:\t");
        m=getche();
        if(m=='y') goto new_str; //new_str -метка
        else    getch();
     }
    /*  Вывод исходной строки
    " И н и ц и а л и з а ц и я   с т р о к и "
         Вывод строки в обратном порядке
    " и к о р т с   я и ц а з и л а и ц и н И "
         Ввод новой строки -y, нет-n:    n
    "икортс яицазилаицинИ"
         Количество символов =22
         Вывод строки в обратном порядке
    "Инициализация строки"
         Ввод новой строки -y, нет-n:    y
    qwert12345
         Количество символов =10
         Вывод строки в обратном порядке
    54331trewq
         Ввод новой строки -y, нет-n:    n      */
```

Ход работы

1 По индивидуальному заданию преподавателя составить программу с различными вариантами применения операторов цикла: **while, do… while, for**.

Предусмотреть дополнительные возможности выхода из циклов (операторы **break, goto**).

2 Набрать программу на компьютере, выявить и устранить ошибки.

3 Ознакомиться с работой операторов цикла в языке **С.**

4 Получить результаты работы программы.

5 Оформить отчет и сделать выводы о проделанной работе, изучив контрольные вопросы по теме.

Требования к содержанию отчёта приведены в лабораторной работе №1.

Индивидуальное задание к лабораторной работе №4

Найти сумму ряда

$y = \sum \frac{f_1}{f_2}$, где $a \le x \le b, \quad \Delta x = c.$

Варианты заданий приведены в таблице 4.1.

Таблица 4.1 - индивидуальное задание

Вар.	f_1	f_2	***A***	***b***	***c***
1	$3x-1$	$e^{-1/x}+x/(x+1)$	3	5	0,5
2	X^3-3x^2	x^4+2x^2+3	1	3	0,2
3	$e^{-x}+4x$	$\sqrt{(1+x^3+4x^2)}$	0,6	4,2	0,3
4	$Sin^2(x+4x^3)$	$(x+2x^3)\sqrt{x}$	0,5	4,8	0,2
5	$Xsinx^3-ln2x$	$arctgx/4+e^{-x+2}$	2	6,3	0,4
6	X^4-cosx	$tgx+2x$	1	5	0,5
7	$2x+sin^2x$	$\sqrt{(2x+lnx)}$	5	8	0,3
8	$ln(4x+8)$	$e^{-x}+sin2x$	1	4	0,2
9	$x^3ln(2x)$	$4x^2+6x^3-2$	0,5	6	0,3
10	x^2+sin^3x	$cos3x+e^{-2x}$	-2	3	0,4
11	xe^{-x}	$sin4x+x^3$	1,5	5	0,3
12	$\sqrt{x^2+1\|}$	$arctgx/5+2x$	0,6	4	0,2
13	$x^2/(3x+2)$	$sin^2(\pi x+1)$	0,5	5,2	0,3
14	$\sqrt{2+x^3}$	$3^x/(x-2)$	1,2	6,3	0,4
15	$x^{3x+1}+8x$	$\|x-8\|+sinx$	4	7,5	0,3
16	x^4+e^{x+3}	$xarctg(x/3)$	2	6,4	0,2
17	$ln^2(x+4)$	$sin^3(x/5)$	1	6,8	0,3
18	$e^{x-2}+x^3$	$x-ln\|x-1\|$	0	4	0,4
19	$2cos(x+3)$	$4x^2/(3+x^3)$	2	5	0,3
20	$\sqrt{1+x^4})$	$tg^2(x+4)-e^{-x}$	1	6	0,4
21	$3+2sin^2(x-3)$	$4+x/10$	2	7	0,5
22	$ln(1(1+2^x)$	$sin^2(4x+1)$	1,5	6,8	0,4
23	$\sqrt{\|x\|}+e^{-x}$	$5arctg(4x)$	2	7	0,5
24	$arcsin(x+2)$	$3(x-4)/(x^2+1)$	3	8	0,2

25	$e^{\|x+2\|}$	$ln^2(x+4)$	-2	6	0,3
26	$(4-x)cos2x$	$\sqrt{\|x+1\|}+e^{-3x}$	1	7	0,4
27	$\sqrt{\|x\|+2^x}$	$sinx^4-4$	-2	5	0,2
28	$2^{x+4}+cos^2x$	$ln\|x+8\|$	-4	2	0,5
29	$(x+2)/sin^3x$	$\sqrt{x+tg^2x}$	1	4	0,3
30	$e^{x+3}+4x^2$	$arcsinx^3$	2	5	0,2

Контрольные вопросы для подготовки и самостоятельной работы

1 С помощью каких операторов можно досрочно завершать выполнение операторов цикла?
2 Какие выражения можно использовать в операторе цикла?
3 Как интерпретируются значения **<выражений>** операторов цикла? Какого типа могут быть эти **<выражения>?**
4 Можно ли записать следующие операторы: **for (; ;); while (∅); do while(i--); for (; i && j ; i++, j- -); for (int i, k ; ; i+=2) j++ ; k-=5;**?: Объясните, почему можно или нельзя ?
5 Объясните результат работы приведенных операторов.
6 В чём разница в работе операторов с предусловием и с постусловием?
7 Каково назначение выражений в операторе **for**?
8 Зачем используются составные операторы (операторные скобки)?
9 С помощью какого оператора можно прекратить выполнение текущей итерации в цикле?
10 Назовите порядок вычисления и интерпретации выражений в операторе **for**.
11 Объясните работу операторов в примерах.

Лабораторная работа №5

Разветвляющийся вычислительный процесс с различными логическими условиями: оператор if... else, условная операция (?:), оператор switch, оператор break, оператор goto

(2 часа)

Цель работы: Изучить реализацию в языке ветвящихся вычислительных процессов . Научиться писать программы, используя операторы: ветвления **if...else**, переключения **switch** в паре с оператором **break,** оператор перехода **goto**, тернарную условную операцию **(?:).**

Теоретические сведения

Разветвляющийся вычислительный процесс применяется в тех случаях, когда необходимо произвести выбор одного из вариантов дальнейших действий или вычислений в зависимости от текущих значений переменных и логических условий. Например, произвести вычисление по одной или по другой формуле.

Оператор if…else (если…иначе)

Формат оператора **if…else** приведен ниже:

if(<выражение>) <оператор 1>
[else <оператор2>]

Действия оператора зависят от значения выражения. Реализация различных возможностей выполняется следующими способами. Если **<выражение>** в скобках не равно ∅ **(“Истинно”),** то будет выполнятся **<оператор1>.** В противном случае, если указанное **<выражение> равно ∅ (“Ложно”),** то будет выполняться **<оператор2>** в блоке **else,** если он присутствует.

В теле оператора **if** может находиться один или более операторов. Если должны выполняться два или более операторов, их необходимо заключить в операторные скобки: **"{"** и "}". **<Выражение>** представляет собой запись логического условия или условий. Например: **1) if(i<=j)...; 2) if(size= =a)...; 3) if(t>10 && v<3) ..; 4) if(a){…}else{…}; 5) if(!a){…}else{…}.**

В первом случае **<выражение>** истинно, если значение переменной **i** будет меньше или равно значению переменной **j;** во втором случае условие будет истинно, если значения переменных **size** и **a** будут равны; в третьем выражении истина будет соблюдаться, когда переменная **t** будет больше десяти “И” **(&&)** переменная **v** будет меньше трех. В четвертом случае первый блок будет выполняться при **a** не равном ∅ , в противном случае выполняется блок после **else**. В пятом выполняются действия, обратные, указанные в четвертом варианте. Вместо переменных можно использовать выражения, которые будут непосредственно вычисляться, например: **if ((i+2)/3 < 4*j**)...выделение. Основные операции, которые ставятся между сравниваемыми величинами, следующие :

= = - знак "равно" (Не путать с присваиванием **"=(!)**);	<= - знак "меньше или равно";
! = - знак "не равно";	> - знак "больше";
< - знак "меньше";	> = - знак "больше или равно".

Основные логические операции, которые ставятся между сравниваемыми выражениями (если их два как в примере 3 или более), следующие :

! - операция **"НЕ".** Пример*:* **if (!(num % 2))...** // истина -если **num** нечетно.

&& - операция "**И**". Пример 3 (см. выше).

|| - операция "**ИЛИ**". Пример: **if (i+n*2<l || i%2=0)...**

Детально логические операции будут рассмотрены в следующей лабораторной работе. Ниже приведен фрагмент программы, который выполняет следующие действия: запрашивает на ввод три числа и выдает в результате сравнения наименьшее из них. Оператор **if…else** допускает возможность вложенности при ветвлении, что обеспечивает гибкость реализации логики и компактность записи операторов.

Пример

```
...
printf("Ввести три числа через пробел: ");
scanf(" %d%d%d ", &a, &b, &c);
if (a<=b && a<=c) printf("\ n Наименьшее: %d", a);
else
if (b<=a && b<=c) printf("\ n Наименьшее: %d", b);
else printf("\ n Наименьшее: %d", c);
...
```

Условная тернарная операция (?:)

Синтаксис данной операции следующий:

<выражение1> ?<выражение2> : <выражение3>;

Результат операции будет равен **<выражению2>** в том случае, если **<выражение1>** истинно, в противном случае результат будет равен **<выражению3>**. Ниже показан пример реализации данной конструкции.

int result = (i<j) ? i : j ; // Переменной **result** присваивается наименьшее значение // **(i** или **j),** если **i** < **j** то **i**, иначе.

Оператор варианта switch

Оператор выбора варианта **switch** заменяет несколько операторов **if…else**. Обычно оператор **switch** используется тогда, когда требуется выбор и выполнение только одной последовательности операторов из нескольких возможных, хотя возможности оператора **switch** не ограничиваются этим случаем.

Синтаксис оператора **switch**:

```
switch (<выражение>)
        {
        case <константное_выражение_1>:
                <оператор 1>
        [case <константное_выражение_2>:
                <оператор 2>]
        ...
        [case <константное_выражение_n>
                <оператор n>]
```

[default:
<оператор>]
}

После вычисления **<выражения>** типа **int** в **(** в заголовке оператора **switch)** результат сравнивается последовательно с **<константными_выражениями>,** стоящими после зарезервированных слов **case**, которые вместе с **<константными_выражениями>** являются внутренними метками оператора, После **<константных_выражений>** обязательно ставится признак метки **":"**. Сравнение значения **<выражения>** начинается с самого верхнего **<константного_выражения>** и далее, пока не будет установлено их соответствие. Тогда выполняются операторы после соответствующей метки **case,** на которую передается управление. Для того чтобы прекратить последовательное выполнение операторов и выйти из оператора **switch**, необходимо после группы операторов, принадлежащих выбранному **case**, поставить оператор **break**. Если **break** отсутствует, то последовательно выполняются все операторы, а метки не учитываются.

Последовательность операторов, стоящих после слова **default** (умолчание), выполняется тогда, когда значение **<выражения>** не совпадает ни с одним <**константным_выражением>.** Пример показывает возможности оператора **switch**. Функция **error_message** выводит одно из трех сообщений в зависимости от значения, когда ошибки (параметр **error_code)**.

Пример

```
void error_message(int error_code)
{ switch(error_code)
 { case 1: printf("\n сообщение 1"); break;
 case 2: printf("\n сообщение 2"); break;
 case 3: printf("\n сообщение 3"); break;
 default : printf("\n неверный код ошибки");
 }
}
```

Пример

```
/*   ЗАНЯТИЕ N 5
 Выполнил студент группы .....   Петров Ю.В.
 Применение операторов if, switch, тернарной операции (? :)
 на примере простого калькулятора. Более сложные условия
 для оператора switch приведены в тексте программы      */

#include <stdio.h>
#include <conio.h>
#include <math.h>
int main(void)
  {  float a,b,rez;
     int s,p,h;
     char oper, flag=' ';
```

```
    clrscr();
    printf("\n Введите через пробел два числа:  ");
    scanf(" %f %f ", &a, &b);
    printf(" Введите знак операции (+ - * /): ");
oper1: oper=getche();
    switch(oper)
        { case '+': rez=a+b; break;
          case '-': rez=a-b; break;
          case '*': rez=a*b; break;
          case '/': rez=a/b; break;
          default : printf("\n Повторите ввод знака: ");
                    goto oper1;
         }
    printf("\n Результат операции: %5.2f\n",rez);
oper2:printf("\n Введите цифру в интервале (-9...+9) :  ");
    scanf("%d",&s);     //Условия для оператора switch:
    if(s<0)             //если    s=0, то - a++, p++, h++
     { s=abs(s);        //если    s=1, то -      p++, h++
      flag='-';         //если    s=2, то -          h++
     }                  //если    s=3, то - a--
    else flag=' ';      //если    s=4, то -      p--
    if(s<=9)            //если    s=5, то -          h--
                        //если s=6..7, то - a=1, p=1, h=1
    switch(s)           //если    s=8, то - a=0, p=0, h=0
      {case 0:                          a++;
       case 1:                          p++;
       case 2: printf("\tВы ввели %c%d",flag,s); h++; break;
       case 3: printf("\tВы ввели %c%d",flag,s); a--; break;
       case 4: printf("\tВы ввели %c%d",flag,s); p--; break;
       case 5: printf("\tВы ввели %c%d",flag,s); h--; break;
       case 6: case 7: printf("\tВы ввели %c%d",flag,s);
                                    a=1; p=1; h=1; break;
       case 8: printf("\tВы ввели %c%d",flag,s);
                                    a=0; p=0; h=0; break;
       default: printf("\tВы ввели %c%d \n\
       на границе интервала",flag,s);
                goto oper2;
        }
       else { printf("\tВы ввели число вне интервала (-9...+9)\n");
              goto oper2;
            }
       printf("\n Результат switch: a=%5.2f p=%2d h=%2d\n",a,p,h);
       getch();
       if (p==0) {b=10; a=10;}
       else      {b+=b/p; a+=a/p;}
       printf("\n Результаты вычислений: a=%5.2f b=%5.2f",a,b);
```

```
        rez=(b<a)?b:a;
        printf("\n Результат тернарной операции: res=%5.2f",rez);
        getch();
        return 0;
}
/*Введите через пробел два числа:  54 32
  Введите знак операции (+ - * /): /
  Результат операции:  1.69

  Введите цифру в интервале (-9...+9) :  -9
     Вы ввели -9  на границе интервала
  Введите цифру в интервале (-9...+9) :  -6
     Вы ввели -6
  Результат switch: a= 1.00 p= 1 h= 1

  Результаты вычислений: a= 2.00 b=64.00
  Результат тернарной операции: res= 2.00    */
```

Ход работы

1. В соответствии с индивидуальным заданием разработать программу с применением операторов ветвления, выбора варианта, разрыва, перехода (при составлении программы использовать операторы **if…else, switch, break, goto,** операцию **(?:)** в двух-трех вариантах).
2. Набрать программу и устранить ошибки.
3. Изучить работу операторов, различные возможности их применения.
4. Получить результаты.
5. Оформить отчет и сделать выводы по работе, изучив контрольные вопросы по теме.

Требования к содержанию отчёта приведены в лабораторной работе №1.

Индивидуальное задание к лабораторной работе №5

Вычислить значение функции

$$y = f(x), \text{ где}$$

$$x = \begin{cases} f_1(z), \text{ если } z < 0; \\ f_2(z), \text{ если } 0 \le z \le 8; \\ f_3(z), \text{ если } z > 8; \end{cases}$$

$$z = cos(c).$$

Значения функций приведены в таблице 5.1.

Таблица 5.1 - индивидуальное задание

Вар.	***f(x)***	***$f_1(z)$***	***$f_2(z)$***	***$f_3(z)$***	***c***
1	x^2+8x-6	z^3-3z^2	$zln(z)$	E^z-e^{-z}	5,1
2	X^3lnx^2	$e^{-z}+3z$	$ln\|z\|$	$Cosz+z^2$	5,4

3	$X^{1/4}+\sin x$	$2z-\ln\lvert z\rvert$	$tgz-2z$	$\sin^3 z$	4,1
4	$X^4+2\sin x^2$	$\sin z+tgz$	$\cos^3 z+3/z$	$Z^2+\ln z^2$	3,2
5	$\cos x^3$	$z^2+2\sin z$	$\ln z+2z$	E^z+1/z	4,7
6	$\sin x+2\ln x$	$2z+tgz$	$\ln z^4+2z$	$\cos z+2z$	1,3
7	$\sin^4 x^2$	$\sin z^2-z^3$	$\sqrt{\lvert\cos z\rvert}$	$2\sin z^2$	1,6
8	$Tgx-4x^3$	$1/\cos^2 z$	$z-\ln\lvert z\rvert$	$Z^3+\sin z$	1,5
9	$\ln x-e^{2x}$	z^2+e^z	$\cos^4 z/z^3$	$Tg(z+1/z)$	2,7
10	$2x-\ln x$	$2\cos z+1/z$	$z^3-2\ln\lvert z\rvert$	$Tg2z+z^3$	3,8
11	$3x-\sin x$	$3tg^3 z$	$1/\cos^4 z$	$E^{2z}+\sin z$	1,6
12	$4x^2+\cos x$	$3z/\sin z$	$z^2+2\sin z$	$2z-\ln\lvert z\rvert$	2,4
13	$\sqrt{x}+\cos x$	$z^2+\ln z^2$	e^z+1/z	$Z^4-\sin z$	4,1
14	$x^{1/3}+2x$	$\ln\lvert\cos z\rvert$	$2z+e^z$	$Tg^2 z$	2,5
15	$\sin^4 x+2x$	$z^5/\sin 2z$	$e^{-2z}+tgz$	$\cos^4 z+z^{1/3}$	3,2
16	$Tg4x+1/x$	$z/\sin z^{1/5}$	$2ztg^3 z$	$Z\sqrt{\lvert z\rvert}+8$	1,4
17	$\ln(1/x)$	$z\sin^2 z-8$	$\ln\sin z^{0,8}$	$\sqrt{ze^z}-2,5$	2,3
18	$e^{2x}+4x$	$\cos(\pi/4)-z$	$1/(e^z+1)$	$arctg(z+3)$	4,1
19	$\cos x^4+x/2$	$\sin(z+30°)$	$\ln\cos(\pi z/6)$	$e^{-tg(z-2)}$	3,2
20	$2tgx+e^x$	$z+\cos(\pi+z)$	$z^3+z^{1/3}$	$Z^4-\ln z$	2,8
21	$2\ln x^2$	$arccos z^2$	$\sin z+\ln\cos z$	$Z^3-\sin(\pi x)$	1,7
22	$\cos^2 x/3$	$z^2+\ln(z+4)$	$e^{(z-5)}+\sin z$	$\sqrt{z^4+tgz}$	2,2
23	$1/tgx^4$	$e^{-4z+2}+z^2$	$\cos(z^{1/3}+2)$	$\sin(\pi+4z^2)$	5,6
24	$e^{2x}-x^3$	$tg(z^2+\sqrt{z})$	$\ln(\sin z+5)$	$Z^4+z^2-\cos z$	3,4
25	$Tgx-2\ln x$	$arcsin(z+3)$	$z^3-z^2+\cos z$	$Ln(z^3+4z)$	2,5
26	$\cos x^4+x/2$	$\ln\sin z^{0,8}$	$\cos(\pi/4)-z$	$Z/\sin z^{1/5}$	3,7
27	$\ln(x+x^2)$	$2ztg^3 z$	$\sin(\pi+4z^2)$	$Z^3+z^{1/3}$	2,6
28	$\cos x^4+2x$	$tg(z+1/z)$	$e^{2z}+\sin z$	$\cos z^{1/5}$	3,8
29	$\sin^4 x+2x$	$z^2+\ln z^2$	$\cos^3 z+3/z$	$\cos(\pi/4)-z$	5,8
30	$3\ln(x^2+5)$	$z^4-\ln z$	$\sin z+tgz$	$\sin z+\ln\cos z$	3,5

Контрольные вопросы для подготовки и самостоятельной работы

1 Какой тип должно иметь **<выражение>** в операторах **if** и **switch?**

2 Можно ли использовать оператор **goto** для передачи управления на **else**, метку **case…; default:?**

3 С помощью каких операторов можно досрочно завершить выполнение операторов **if…else, switch?**

4 Можно ли использовать в качестве **<выражения>** в операторе **switch** указатель?
5 Можно ли использовать в качестве константного **<выражения>** в операторе **switch** константу?
6 Обязательно ли использовать оператор **break** в операторе **switch**? Каково его действие? Что происходит при отсутствии **break**?
7 Какому из вложенных операторов **if** относится **else** при наличии и отсутствии операторных скобок "{" и "}".
8 Какие типы операндов допустимы в условной операции **(? :)**?
9 Объясните работу операторов в приведенных примерах.
10 Как объявить и использовать метки в программе?

Лабораторная работа №6

Операции С, их приоритеты и использование.
Преобразование типов
(4 часа)

Цель работы: Изучить основные логические, арифметические и другие операции **С,** научиться правильно составлять выражения **С**, изучить приоритеты операций **С**, научиться использовать преобразование типов.

Теоретические сведения

Язык **С** имеет мощную арифметическую и логическую основу, которая позволяет быстро, компактно и эффективно писать код программы. В **С** разработано множество базовых арифметических и логических операций, а также функции библиотеки математической поддержки языка. Операндами операций могут быть выражения определённых видов, зависящих от операции. В простейшем случае операндами являются переменные. Переменные, прежде чем они будут использованы, должны быть объявлены с определённым спецификатором типа.

Таблица 6.1 - допустимые операции над переменными

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ		
Операция	**Пояснение**	**Пример**
*	Умножение	**A=b*c;**
/	Деление (для целых – нацело)	**a=b/c;**
%	Остаток от деления (для целых)	**a=b%c;**
<операция x> **=** **+ = ;** **- = ;** *** = ;** **/ = ;** **% =**	составное присваивание Изменить и заменить	**a+=3**; или **a=a+3; a%=c**; или **a=a%c**;
++	Инкремент(увеличить на 1)	**c++** ; или **c=c+1; + + a**
--	Декремент(уменьшить на 1)	**c-- ;** или **c=c-1; - - a**
+	Бинарный плюс (сложение)	**A=b+c;**
-	Бинарный минус (вычитание)	**A=b-c;**
ЛОГИЧЕСКИЕ ОПЕРАЦИИ		
Операция	**Пояснение**	**Пример**
&&	**"И"**	**a&&b**
||	**"ИЛИ"**	**a||b**
!	**"НЕ"**	**!a**
==	**"РАВНО"**	**a= =b**
!=	**"НЕ РАВНО"**	**a!=b**
>	**"БОЛЬШЕ"**	**a>b**
>=	**"БОЛЬШЕ ИЛИ РАВНО"**	**a>=b**

<	**"МЕНЬШЕ"**	**a<b**
<=	**"МЕНЬШЕ ИЛИ РАВНО"**	**a<=b**
ПОБИТОВЫЕ ОПЕРАЦИИ (ПОРАЗРЯДНЫЕ)		
Операция	**Пояснение**	**Пример**
&	**“И” (and)**	**1&1=1; 1&0=0 ;0&0=0;**
|	**“ИЛИ” (or)**	**1 | 1=1 ; 1 | 0 =1 ; 0 | 0=0;**
^	**“ИСКЛЮЧАЮЩЕЕ ИЛИ”**	**(xor) 1^1 = 0 ; 1 ^ 0=1 ; 0^0=0;**
~	**“ОТРИЦАНИЕ” (not)**	**~1=0; ~0=;**
<<	**“СДВИГ ВЛЕВО” (shl)**	**0001b<<2=0100b**; (буква **b** означает что число двоичное)
>>	**“СДВИГ ВПРАВО” (shr)**	**0010b >> 1=0001b;**
<операция_*x*>=	составное присваивание. Изменить и заменить, где <**операция_*x*>** может быть: **&,|,^,>>,<<.**	**a&=b** или **a = &b** **a^=b** или **a = a^b** **a<<=b** или **a<<b**

Приоритет операций и порядок выполнения (ассоциативность)

Приоритет и ассоциативность операций влияют на порядок группирования операндов и порядок вычислений в выражениях **С**. Например, приоритет выполнения операций необходимо учитывать при составлении сложных арифметических формул.

Операции, приведенные в одной группе таблицы, имеют одинаковый приоритет и ассоциативность. Порядок убывания приоритета сверху вниз.

Таблица 6.2 - ***Приоритет операций***

Приоритет	Знак операции	Тип операции	Ассоциативность (порядок выполнения)
1	**() [ ] .–>**	Первичные	→ слева направо
2	**– ~ ! * & ++ -- sizeof,** приведение типов **( )**	Унарные	← справа налево
3	*** / %**	Мультипликативные	→
4	**+ -**	Аддитивные	→
5	**<< >>**	Сдвиги	→
6	**< > <= > =**	Отношение	→
7	**= = ! =**	Отношение	→
8	**&**	Поразрядное "**и**"	→
9	^	Поразрядное исключающее "**или**"	→
10	|	Поразрядное включающее "**или**"	→
11	**&&**	Логическое "**и**"	→

Продолжение таблицы 6.2

При-оритет	Знак операции				Тип операции	Ассоциативность (порядок выполнения)
12	||				Логическое "**или**"	→
13	**? :**				Условная (тернарная)	←
14	=	* =	/ =	% =	Простое и составное присваивание	←
	+ =	– =	<< =	>> =		
	& =	| =	^ =			
15	,				Последовательное вычисление	→

Если несколько операций одного приоритета встречаются в выражении, то они применяются в соответствии с ассоциативностью.

Примеры. **a = b&∅xFF + 5**; // вычисляется как **a = b&(∅x FF + 5);**

b=a + c >> 1; // как b=(a +c) >> 1;

c = a + + + b/5; // как **c=(a + +) + (b/5);**

Мультипликативные, аддитивные и поразрядные операции обладают свойством коммутативности. Компилятор вычисляет выражения с учётом приоритета в любом порядке, даже если есть скобки. Определённый порядок вычисления (,) операндов гарантируют операции: последовательного вычисления, логические **«И» (&&)** и **«ИЛИ» (||)**, условная операция **(? :).**

Запятые в вызовах функций не являются операциями последовательного вычисления и не обеспечивают гарантий вычисления слева направо. Логические операции вычисляют минимальное число операндов, необходимых для определения результатов выражения.

```
func (i + 1, i = j + 2);      //. Не гарантирует порядок вычисления фактических
                              // аргументов
i= 0;                         // i имеет тип int по умолчанию
a  [++  i]  =  i;             // порядок вычисления левого и правого операндов не
                              // гарантируются a [0] = 0 или  a[1]=1
(x - 5) && ++ i               // Если x =5, то ++ i не вычисляется
int x, y, z, f();
z = x > y || f(x, y);         // Если x > y, то значение z = 1 «Истина», а f( ) –не
                              // вызывается
                              // если x ≤ y, то f() вызывается, тогда z=0,
                              // если f( ) возвращает нулевое значение, или z = 1,
                              // если f( ) возвращает не нулевое значение
                              // printf ("%d %d \n", ++n, p()2, n)
                              // в функцию может передаваться n или n+1.
```

Преобразование типов

В выражениях **C** переменные различных типов в ряде случаев могут использоваться совместно; например, переменные типа **char** могут присутствовать в выражениях одновременно с переменными типа **int.**

Пример совместного использования целых и символьных переменных.

```
char ch='a', ans;        //объявление переменных ch и ans
printf("значение ch + 3 = %d", ch+3);  // вывод значения ch+3
ans = ch % 3;    // определение остатка от целочисленного деления
printf("\n\n значение ans = % d\n", ans);
```

Поскольку **char** это целый тип, для него применимы все операции, операнды которых могут иметь тип **int.** Целые по умолчанию - это величины со знаком **signed**.

С переменными вещественного типа (**float**, **double** и др.) применимы все операции, допустимые для целого типа **int,** за исключением операции остатка от деления **(%)**.

Преобразования типов бывают явные и неявные. Синтаксис операции явного преобразования типа

(<новый_тип>) <операнд>

или

<новый_тип> (<операнд>).

Ряд операций может в зависимости от типов своих операндов вызывать неявное преобразование значения операнда из одного типа в другой (преобразование по умолчанию).

Рассмотрим результаты таких преобразований.

Данные типа **char** или **short int** могут использоваться везде, где используется тип **int**. Во всех случаях значение преобразуется к целому типу

Арифметические операции над числами с плавающей точкой **(float** и **double)** по умолчанию выполняются с двойной точностью.

Преобразования целочисленных значений в вещественные (плавающие) выполняется без осложнений, но возможна потеря точности, если для результата не предусмотрено достаточного количества битов. Преобразование значений с плавающей точкой к целочисленному типу машинно-зависимо. Результат неопределен, если значение не попадает в отведенный диапазон.

Если целое без знака **(unsigned)** используется вместе с простым целым, то простое целое и результат преобразуются в целое без знака.

Подавляющее большинство операций вызывает преобразование и определяет типы результата в соответствии с вышеприведенными правилами. Приведенная ниже схема преобразований реализуется по умолчанию при вычислении выражений **C.**

Сначала любые операнды типов **char**, **unsigned char** или **short** преобразуются в **int**, а любые операнды типа **float** преобразуются в **double**.

Затем, если какой-либо операнд имеет тип **double,** то другой преобразуется к типу **double** и типом результата будет **double**.

В случае, если какой-либо операнд имеет тип **unsigned long**, то другой преобразуется к типу **unsigned long** и это же будет и типом результата.

В случае, если какой-либо операнд имеет тип **long,** то другой преобразуется к типу **long** и это же будет типом результата.

В случае, если операнд имеет тип **unsigned**, то другой операнд преобразуется к типу **unsigned,** и это будет типом результата.

Объект типа **void* (**указатель на пустой**)** может быть объявлен для указания на объекты неизвестного типа.

Преобразование типа такого указателя задаётся с помощью явной операции преобразования типов.

Указатель может преобразовываться в любой из целых типов, достаточно большой для того, чтобы вместить его.

Указатель на один тип может быть преобразован в указатель на другой тип. При этом возможно преобразование указателя в указатель на объект меньшего размера и обратно без изменений.

Например, функция выделения памяти может принимать размер (в байтах) объекта, для которого выделяется память, и возвращать указателю несоответствующий тип, который может таким образом использоваться.

```
extern void* allos ();
doube* dp;

dp = (doube*) allos (sizeof (doube));
*dp = 2,6/8,4
```

Пример

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
 main()
{float r1,r2;
 int a,b,b1;
unsigned c,d;
        char e,f;
        unsigned char g;
        float f1,f2;
 clrscr();
 printf("ввод первого и второго вещественных чисел: ");
 scanf("%f %f",&r1,&r2);
 //printf("\n");
 printf("вывод результатов операций для чисел: %5.2f %5.2f\n",r1,r2);
 printf("!r1=  %d  ",!r1);  printf("!r2=  %d  ",!r2);
 printf("r1>r2  %d  ",r1>r2);  printf("r1<r2  %d\n",r1<r2);
 printf("r1||r2  %d  ",r1||r2); printf("r1&&r2  %d  ",r1&&r2);
 printf("r1==r2  %d  ",r1==r2); printf("r1>=r2  %d\n",r1>=r2);
 printf("r1<=r2  %d  ",r1<=r2); printf("r1!=r2  %d\n",r1!=r2);
 //Вложенный блок, переменные переобъявлены: int r1,r2; float b;
```

```
{int r1,r2;
 float b;
 printf("ввод первого и второго целого числа:  ");
 scanf("%d %d",&r1,&r2);
// printf("\n");
 printf("вывод результатов операций для целых чисел: %2d %2d\n",r1,r2);
 printf("!r1=   %d   ",!r1);   printf("!r2=   %d   ",!r2);
 printf("r1>r2  %d   ",r1>r2);  printf("r1<r2  %d\n"  ,r1<r2);
 printf("r1||r2  %d   ",r1||r2); printf("r1&&r2  %d   ",r1&&r2);
 printf("r1==r2  %d   ",r1==r2); printf("r1>=r2  %d\n"  ,r1>=r2);
 printf("r1<=r2  %d   ",r1<=r2); printf("r1!=r2  %d   ",r1!=r2);
 printf("~r1    %d   ",~r1);   printf("r1|r2  %d\n"  ,r1|r2);
 printf("r1^r2  %d   ",r1^r2);  printf("r1&r2  %d   ",r1&r2);
 printf("r1<<r2  %d   ",r1<<r2); printf("r1>>r2  %d\n"  ,r1>>r2);

 printf("Исходные значения: r1=%d  r2=%d\n",r1,r2);
 r2=r1++;       //Постфиксные операции a1++ a1--
 printf("r2=r1++;        r1=%d r2=%d\n",r1,r2);
 --r1;  r2=++r1; //Префиксные операции ++a1 --a1
 printf("--r1; r2=++r1;   r1=%d r2=%d\n",r1,r2);
 r1-=4;   r2+=5; //Составное присваивание
 printf("r1-=4; r2+=5;    r1=%d r2=%d\n",r1,r2);
 a=r2-=2,r1+=5;  //Составное присваивание
 printf("a=r2-=2,r1+=5;   r1=%d r2=%d a=%d\n",r1,r2,a);
 a=(r1<r2)?r1:r2;//Тернарная операция если r1<r2, то a=r1 иначе a=r2
 printf("a=(r1<r2)?r1:r2; a=%d\n",a);
 a=r2%r1;       //Остаток от деления целых
 printf("a=r1%r2;   "); printf("a=%d\n",r2%r1);
 a=r2/r1;       //Деление целых
 printf("a=r2/r1;        a=%d\n",a);
 b=(float)r2/(float)r1;  //Деление с преобразованием типов
 printf("b=(float)r2/(float)r1; b=%f\n",b);
 }
 float q=1.3,q1=2.4,raz;
 printf("Введите переменные a-(int), \
 c-(unsigned), g-(unsigned char)\n");
 scanf("%i,%u,%uc",&a,&c,&g);
 b = (a & (c<<3));
 b1 = (a & 3) << 7;
 f = (a & 3) << 7;
 f1 = q / (c | 0x3E);
 f2 = a /  (c | 0x3E);
 raz=exp(q+q1)/4;
 printf("g=%u,  q=%5.2f,  q1=%7.2f,  b=%i,  b1=%i, \
  \n",g,q,q1,b,b1);
 printf("f=%i,  f1=%6.3f, f2=%6.3f, raz=%f\n",f,f1,f2,raz);
```

```
 getch(); return 0;
}
/* ввод первого и второго вещественных чисел: 56     7
   вывод результатов операций для чисел:      56.00  7.00
!r1=    0    !r2=    0    r1>r2   1    r1<r2   0
r1||r2  1    r1&&r2  1    r1==r2  0    r1>=r2  1
r1<=r2  0    r1!=r2  1
ввод первого и второго целого числа:        45  2
вывод результатов операций для целых чисел: 45  2
!r1=    0    !r2=    0    r1>r2   1    r1<r2   0
r1||r2  1    r1&&r2  1    r1==r2  0    r1>=r2  1
r1<=r2  0    r1!=r2  1    ~r1     -46  r1|r2   47
r1^r2   47   r1&r2   0    r1<<r2  180  r1>>r2  11
Исходные значения: r1=45  r2=2
r2=r1++;          r1=46  r2=45
--r1; r2=++r1;    r1=46  r2=46
r1-=4; r2+=5;     r1=42  r2=51
a=r2-=2,r1+=5;    r1=47  r2=49  a=49
a=(r1<r2)?r1:r2;  a=47
a=r1%r2;          a=2
a=r2/r1;          a=1
b=(float)r2/(float)r1;    b=1.042553
Введите переменные a-(int),  c-(unsigned), g-(unsigned char)
                   -34          6          7
g=122,  q =1.30,   q1=2.40,     b=512,       b1=256,
f=0,    f1=0.010,  f2=519.000,  raz=10.111827          */
```

Ход работы

1. Изучить теоретические сведения.
2. Для использования арифметических, логических и других операций, приведенных в таблице задаться выражениями, содержащими указанные операции. В качестве базы принять лабораторную работу №5.
3. Ознакомившись с приоритетом операций, показать порядок выполнения операций в конкретных выражениях с использованием скобок.
4. Для преобразования типов переменных использовать явное и неявное преобразование типов.
5. Разработать алгоритм и программу, отладить ее на компьютере.
6. Изучить выполнение операций и тип результата.
7. Получить результаты и сделать выводы по работе.
8. Оформить отчет.
9. Подготовиться к защите лабораторной работы, изучив вопросы по данной теме.

Требования к содержанию отчёта приведены в лабораторной работе №1.

Индивидуальное задание к лабораторной работе №6.

Составить программу для вычисления арифметических, логических и битовых выражений. Преобразовать полученные результаты согласно индивидуальному заданию приведенному в таблице 6.3.

Таблица 6.3 - Индивидуальные задания

вариант	арифметическая операция	арифметическая операция	логическая операция	битовая операция	преобразование: явное
1	(a + b)*c	(c/a^b)	чётное	a << = b	int→short
2	(a*b)-c	a|=b-c	нечётное	a >> = b	long→int
3	(a / b)+++b	a<<=b/c	(a==c)&& (b<a)	a^=b	signed→unsigned
4	++b-(~a)	a%=b	a > b	a%=b+c	double→float
5	(a+b)* sizeof(c)	$(a+b)^{1/c}$	a < b	a>>=5	int→char
6	--c*(*&a+b)	(a+b)/5	a>=b	a&=abs(c)	long double →double
7	$A^2+b^2+c^2$	$15ab-(1/4c)$	a!=b	a<<=6	float →long
8	$5b^3-2a+c$	$c^2+8b+10a$	a||b	a&=b+c	float→char
9	$4a^2+5b^2$	$3a^2+4b-8$	a&&b	a^=b	double→int
10	$3ab-4c$	A^3+b^2-8c	!a	a%=(c+10)	double→unsigned long int
11	c^2+5a^3-b	A^2+b^2-6c	(a<b)|| (c>5)	a|=20	float→unsigned
12	$2a+4c-b^4$	$A+2b+3c$	a>=b	a&=(b+c)	int→char
13	A^2+b^2	$2(a+b)-c^4$	(a>=b)|| (b<c)	a^=abs(b-c)	long double →double
14	$(a+b)^2$	c^2-b^3	кратное a	(a&b)^c	double→float
15	$2ac-3cb$	$3a-4cb$	(c!=b)|| (a==10)	(a|b)>>c	double→unsigned long int
16	$5c+2a^4$	c^5-2ab	(c<=a)&& (b!=a)	(b&&c)|(a--)	signed→unsigned
17	$A+b+c$	$6a+3b^3+c$	(b==0)|| (c<=a)	a|=b+c	int→short
18	$2a+3b+4c$	$4abc$	(a==1)|| (b<c)	a|= (c+10)	double→int
19	$A^2+b^3+c^4$	$A^2+(b-c)^{5/3}$	(a<b)&& (a>c)	a|=20	double→float
20	$A+2b+3c$	$(a+4b)^{1/3}-c^2$	(a>=b)|| (a<=10)	(a&b)^c	int→char
21	$2(a+b)-c^4$	$A^{1/3}+(b^3-c)$	(b<c)&& (b!=a)	a|=b+c	long double →double
22	c^2-b^3	$B^3+(a-4c)^{1/5}$	(b<c)|| (a<b)	a&=b+c	double→float

23	$3a-4cb$	$A+2b+3c$	(a==1)&& (c!=0)	a&=abs(c-b)	int→char
24	c^5-2ab	$2(a+b)-c^4$	(c==0)|| (b!=100)	a%=b+c	long double →double
25	$6a+3b^3+c$	c^2-b^3	(b!=0)&& (b<c)	(a&b)^c	int→char
26	$4abc$	$3a-4cb$	(b!=a)|| (b<=c)	a%=b+c	long double →double
27	$A^2+(b-c)^{5/3}$	c^5-2ab	(c<=12)& &(c>=24)	(b|c)|(a--)	double→unsigned long int
28	$(a+4b)^{1/3}-c^2$	$6a+3b^3+c$	((a-b)<c)|| ((a*c) <100)	a<<=6	float→unsigned
29	$A^{1/3}+(b^3-c)$	$4abc$	(a<10)? (b):(b-c)	(b&c)|(a--)	int→char
30	$B^3+(a-4c)^{1/5}$	$A^2+(b-c)^{5/3}$	(b<=10)|| ((a+b)< (b-c))	a^=abs(b-c)	long double →double

Контрольные вопросы для подготовки и самостоятельной работы

1 Какие операции называются унарными, бинарными, тернарными?
2 Сколько групп приоритетов принято в **C?**
3 В какой последовательности выполняются операции с одинаковым приоритетом?
4 Что означает свойство коммутативности?
5 Какие операции гарантируют порядок вычисления своих операндов?
6 Для чего применяют первичные операции?
7 Какой тип операндов допустим для различных операций?
8 Все ли операнды вычисляются в выражениях, содержащих логические операции?
9 Для чего применяют преобразование типов?
10 Назовите правила неявного преобразования типов. В каких случаях возможна потеря информации при преобразовании типов?
11 В каком порядке будет выполняться конструкция **f(x)&&g(y**) и как будет интерпретироваться результат вызова функций, результат выражения в целом?
12 Объясните примеры, приведенные в теоретической части.

Лабораторная работа №7

Разработка программ с функциями. Объявление, определение и вызов функций

(2 часа)

Цель работы: выработать практические навыки в написании программ с выделением функций, их объявлением, определением и использованием.

Теоретические сведения

Каждая функция в **C** имеет название (имя), класс памяти, тип возвращаемого значения и необязательный список параметров. В функциях можно объявлять локальные константы и переменные. Все функции, за исключением функции **main(),** необходимо объявлять (создавать прототип, декларировать) до использования (вызова).

Объявления (прототипы) функций

C поддерживает предварительное объявление (декларацию) функций, которое также созданием называется прототипом. Предварительное объявление позволяет перечислить список функций в начале исходной программы. Такой список предлагает удобный способ указания того, какие функции будут использованы в программе. Кроме того, использование прототипов заранее уведомляет компилятор о типах возвращаемых значений и формальных параметров различных функций. После объявления функций можно размещать их определения в любом порядке и не волноваться по поводу ошибок компиляции, которые происходят, когда функция вызывается перед объявлением или определением. Общий синтаксис объявления для функции:

[<Класс_памяти>] <Тип_возвращаемого_значения> <имя> ([<Список_типов_параметров >]);

Классы памяти для функций: **extern** и **static.** По умолчанию – **extern.**

Каждая функция имеет **<тип_возвращаемого_значения>**, который указывается перед названием **(<именем>**) функции**. <Список_параметров>** следует за названием функции и заключается в круглые скобки.

Точка с запятой в конце необходима для объявления функции, но не нужна в определении функции. Имена формальных параметров могут отсутствовать или не опадать с именами в определении функции, но типы должны совпадать обязательно. По умолчанию тип возвращаемого значения **int.**

Пример объявления функции:

dauble summa(float a, float b); // два параметра типа **float,** возвращается
// значение типа **dauble**

C использует специальный тип **void**, чтобы указать, что функция не требует параметров или не возвращает значение. Использование оператора **return** в **void**-функции не требуется.

Примеры объявлений функций:

upDate(); // без параметров, // возвращается **int**	**void releesDate(int fop);**// один // параметр

void (void); // без параметров // **getPrint**	**void reseedType(float Seed);** // один // параметр

Синтаксис определения функций в С

Синтаксис определения функций в **С** имеет следующий вид:

[<Класс_памяти>] <Тип возвращаемго_значения> <Имя> ([<Список_объявлений_формальных_параметров>])
{[// - объявления переменных (декларации);]

// - операторы;
[return <возвращаемое_значение_заданного_типа> ;]
}

Функция возвращает результат своей работы, используя оператор **return**, который обычно появляется в конце тела функции. Однако, функция может иметь больше одного оператора **return.**

<Список_объявлений_формальных_параметров> функции может отсутствовать **(void),** содержать одно или больше объявлений формальных параметров, соответствующих (аргументов) данной функции. Параметры в списке разделяются запятой и имеют синтаксис:

[модификатор]<тип_параметра><имя>

В качестве модификаторов могут использоваться следующие ключевые слова **const, near, face, huge.**

Примеры определений функций, объявленных ранее

dauble summa(float x, float y) // требу- // ет два параметра **{return x+y;** // возвращаемое значение преобразуется к типу **dauble** **}**	**void getPrint(void)** //без параметров // и возвращаемого значения **{printf (n\"**Пример**\n");** **}** // **return** отсутствует

Функция **summa()** имеет два параметра и тип возвращаемого значения **dauble**. Функция **getPrint()** не имеет параметров и возвращаемого значения.

Если функция объявлена с ключевым словом **inline,** то компилятор заменяет любой вызов **inline** - функции копией её тела, приведенного в определении.

Пример **inline** - функции: **inline long square(int nNum)** **{return nNum*nNum;** **}**	В примере определена **inline** - функция **square ( )**, которая возвращает квадрат параметра **nNum** типа **int.**

Использование локальных и глобальных переменных в функциях. Вызов функции

В любой функции можно объявлять локальные константы и переменные. Область действия локальных констант и переменных ограничена телом функции-хозяина. Никакая функция не может непосредственно получить доступ к локальным константам и переменным другой функции. Существует два класса памяти локальных переменных*:* **register** и **auto,** которые указываются перед типом переменных. Локальные переменные создаются каждый раз, когда функция начинает выполняться, а когда функция завершает работу, система устраняет локальные переменные.

В отличие от автоматических переменных, статические переменные (с классом памяти **static**) сохраняют своё значение между вызовами функции. Эта особенность позволяет в функциях использовать значения, вычисленные при предыдущем вызове. Статическая переменная инициализируется один раз при первом вызове функции явно или по умолчанию нулевым значением. Инициализация статических переменных позволяет, например, функции-хозяину определить, выполняется ли она впервые.

Пример статических переменных, объявленных в функции

```
int doCalc()
{static int index1=2;
static float my_index;
// другие объявления
// операторы
return …;
}
```

В примере объявлена и явно инициализирована статическая переменная **index1**, а также инициализирована неявно переменная **my_index** (по умолчанию равна 0)**.** Эти переменные сохраняют свои значения между вызовами функции **doCalc ( )**.

Пример

```
/*   ЗАНЯТИЕ N 7
 Выполнил студент группы .....   Петров Ю.В.
 Объявление и определение функций. Применение функций:
 передача переменных в  функцию по значению, по адресу
 и по ссылке, возврат значений из функции. Области
 видимости переменных, примеры операций.
 Выбор функции - по номеру с помощью оператора switch*/

#include <stdio.h>
#include <conio.h>
#include <math.h>
float glob=2.5;
float Fx0(float a, float b, float c) //Определение функции
 { return --b/(2*a)*(++c); }
   //Fx0-Передача параметров и возврат по значению

float* Fx1(float, float, float);      //Объявление функции
```

```
  //Fx1-Передача параметров по значению, возврат указателя

float& Fx2(float, float, float);      //Объявление функции
  //Fx2-Передача параметров по значению, возврат ссылки

  //Передача параметров по значению, последнего по адресу
void changex0(float, float, float, float *);

  //Передача и изменение параметров по адресу
void changex1(float *, float *, float *, float *);

  //Передача  изменение параметров по ссылке
void changex2(float &, float &, float &, float &);

void main()
   { float a1,b1,c1,x1;
     float* px=&x1;
     float& sx=x1;
     char diskr;
   clrscr();
   printf(" Введите значения переменных: a, b, c: ");
   scanf("%f %f %f", &a1, &b1, &c1);
   printf("Введите номер функции (0...5),\
   6 или ESC-конец расчета:  ");
vvod1: diskr=getche();
   switch (diskr)
   {case '0': x1=Fx0(a1,b1,c1);          break;
    case '1': px=Fx1(a1,b1,c1); printf("\nglob= \
  %6.3f *px= %6.3f", glob, *px); break;
    case '2': sx=Fx2(a1,b1,c1); printf("\nglob= \
  %6.3f  sx= %6.3f", glob,  sx); break;
    case '3': changex0(a1,  b1, c1,&x1); break;
    case '4': changex1(&a1,&b1,&c1,&x1); break;
    case '5': changex2(a1,  b1, c1, x1); break;
    case '6':case 27: goto end;
    default:printf("\nВне диапазона, введите другой \
    номер функции (0...5)  ");
    goto vvod1;
   }
   printf("\nРезультат: a1= %5.3f b1= %5.3f c1= %5.3f \
   x1= %5.3f\n",a1,b1,c1,x1);
   printf("Введите другой номер функции: ");
   goto vvod1;
end:;
}
```

```
float* Fx1(float a, float b, float c)   //Определение функции N 1
  {   float*pf;
      *pf=glob+(--b)/(2*a)*(++c);
   //printf("\nglob= %6.3f *pf= %6.3f",glob,*pf);
      return pf;
  }

float&  Fx2(float a, float b, float c)  //Определение функции N 2
  { float& sf=glob;
    sf=(--b-sqrt(abs(++c)))/(4*++a);
    glob+=5;        //printf("\nglob= %6.3f sf= %6.3f ",glob,sf);
    return sf;
  }

void changex0(float a, float b, float c, float *d)         //N 3
     { *d=pow(b,2)-4*--a*++c;          }

void changex1(float *a, float *b, float *c, float *d)      //N 4
     {++*a; ++*b++; ++*c; *d+=*a+*b+*c;      }

void changex2(float &a, float &b, float &c, float &d)      //N 5
     { a+=2; b+=2; c+=2; d-=a+b+c;      }

/*       Введите значения переменных: a, b, c: 3  4  5
 Введите номер функции (0...5), 6 или ESC-конец расчета: 0
 Результат: a1= 3.000  b1= 4.000  c1= 5.000   x1= 3.000
 Введите другой номер функции: 1
glob= 2.500 *px= 5.500
 Результат: a1= 3.000  b1= 4.000  c1= 5.000  x1= 5.500
 Введите другой номер функции: 2
glob= 5.034  sx= 5.034
 Результат: a1= 3.000  b1= 4.000  c1= 5.000  x1= 5.034
 Введите другой номер функции: 3
 Результат: a1= 3.000  b1= 4.000  c1= 5.000  x1=-32.000
 Введите другой номер функции: 4
 Результат: a1= 4.000  b1= 5.000  c1= 6.000  x1=-18.000
 Введите другой номер функции: 5
 Результат: a1= 6.000  b1= 7.000  c1= 8.000  x1=-39.000
 Введите другой номер функции: 8
 Вне диапазона, введите другой номер функции (0...5) 6 */
```

Ход работы

1 Изучить теоретические сведения.

2 В соответствии с индивидуальным заданием разработать алгоритмы для заданных функций и функции **main().** При разработке функции предусмотреть передачу и возврат значений различных типов.
3 Разработать программу с использованием функций.
4 Выполнить определение функции до функции **main()** и после нее.
5 Набрать программу на компьютере и устранить ошибки.
6 Получить результат и сделать выводы по работе.
7 Оформить отчет.
8 Подготовиться к защите лабораторной работы, изучив контрольные вопросы по теме.

Индивидуальное задание к лабораторной работе №7

Составить программу реализующую вызов функций H, a, b, c согласно индивидуальному заданию приведенному в таблице 7.1.

Таблица 7.1 - ***Индивидуальное задание***

Вар.	***H***	***a***	***B***	***c***	***x***
1	A^2+b^2-6c	x^2-e^{-x}	$lnx+\sqrt{x}$	cos^2x+x^5	5,4
2	$c^2+8b+10a$	$sin^2x+x^{1/4}$	$tgx-8x^3$	$x^4+2sinx^2$	1,2
3	$3a^2+4b-8$	$3x-2cos^3x$	$lnx+2e^x$	$x^{1/3}+4x-1$	0,3
4	A^3+b^2-8c	sin^3x+x^4	$\sqrt{x}-lnx$	$4x-5x^3$	1,7
5	$6b^3+4c-2$	$tgx+e^{2x}$	x^2-6x^3	$1/x-2lnx$	4,1
6	$A^2+b^2+c^2$	$e^x+e^{2x}+4$	$x-sin^3x$	x^2/cos^3x	2,4
7	$5b^3-2a+c$	$tgx-2x$	$\sqrt{x}-sinx$	$x^3/7$	5,5
8	$4a^2+5b^2$	$cosx+2x$	$x^4-2x/5$	$2x-5$	4,6
9	$3ab-4c$	sin^2x+5	$cosx^5$	$x^{1/3}+tgx$	1,6
10	c^2+5a^3-b	cos^3x-6x	$-4x^3+lnx$	$e^{2x}+4cosx$	4,6
11	$2a+4c-b^4$	e^x-2lnx	$2x-5/x$	x^5-2lnx	3,9
12	$A^2+b^2+c^2$	$2/x+x^3$	lnx^2-4x	$tgx-sin2x$	4,1
13	$(a+b)^2$	$lnx+2e^x$	$tgx+e^{2x}$	x^2-e^{-x}	3,4
14	$2ac-3cb$	$1/x-2lnx$	$cosx+2x$	$sin^2x+x^{1/4}$	1,9
15	$5c+2a^4$	x^2-2/x	$(2-x)/6$	cos^3x-2x	2,3
16	$A+b+c$	$lnx/2x$	x^3-4x	$tgx-2x$	4,2
17	$2a+3b+4c$	x^2+x^3	$lnx-x^4$	$cos^2(x-4)$	2,8
18	$A^2+b^3+c^4$	$sin^2x+x^{1/4}$	x^3+4x	e^x+2lnx	1,3
19	$A+2b+3c$	$2x-x^{1/4}$	$\sqrt{x}-2cosx$	$tgx-4x$	3,1
20	$2(a+b)-c^4$	$(x^3-x/2)^3$	$lnx-e^{2x}$	$\sqrt{x+x^3}$	2,4
21	c^2-b^3	$2x+sinx^4$	$sin(x-lnx)$	lnx^2+2x	1,1
22	$3a-4cb$	$2cosx^3$	$tgx/4$	$x/5$	3,1
23	c^5-2ab	$1/2sin^3x$	sin^6x/x^3	$x-4sin2x$	1,8
24	$6a+3b^3+c$	$cosx^x+2x$	$sin2x+tgx$	$lnx-e^{-x}$	2,1
25	$4abc$	x^x-sinx^3	$x/2-x^5$	$2x-sin^3x$	4,1
26	$A^2+(b-c)^{5/3}$	$2x^{1/3}+1$	$sin(x^2+4)$	$lncos^3x$	5,3

27	$(a+4b)^{1/3}-c^2$	$tg(2x)/4$	$cosx^2/x^{1/5}$	e^{-2x+1}/x^2	3,8
28	$A^{1/3}+(b^3-c)$	$x+2^{3x}$	$lnsin^3 4x$	$arcsin^2 x$	4,2
29	$B^3+(a-4c)^{1/5}$	$5^{3x}/(3x-1)$	$e^{-5x}+4/x$	$cos(x^{1/3})$	2,6
30	$c^{1/5}-(b+3a)^2$	$\sqrt{\lvert x+2\rvert}+e^x$	$cosx+x^2$	$arctg(x^3)$	1,3

Требования к содержанию отчёта приведены в лабораторной работе №1.

Контрольные вопросы для подготовки и самостоятельной работы

1 Объясните синтаксис объявления, определения и вызова функции.
2 Всегда ли: последним оператором функции всегда должен быть оператор **return**?
3 Как вы называете переданную локальную переменную – аргумент или параметр?
4 Какие типы переменных всегда передаются по адресу?
5 Какие типы переменных можно передать в функцию по значению?
6 Если переменная передается в функцию по значению и там изменяется, будет ли изменена переменная в вызывающей функции?
7 Если переменная передается в функцию по адресу и там изменяется, будет ли изменена переменная в вызывающей функции?
8 Как объявить тип возвращаемого функцией значения?
9 Какой тип возвращаемого значения используется по умолчанию?
10 В чём разница между объявлением или определением функции?
11 Где размещается объявление и определение функций?
12 Где размещается объявление и определение библиотечных функций?
13 В чём разница между формальными и фактическими параметрами?
14 В чём разница между обычными и **inline** функциями?
15 Какой тип имеет имя функции?
16 Какие классы памяти используются при объявлении функции?
17 Какой класс памяти функций используется по умолчанию?
18 Как включить файл объявления библиотечных функций в программу?

Лабораторная работа №8

Разработка программ с указателями
(2 часа)

Цель работы: изучить конструкции и операторы языка **C** для работы с указателями.

Теоретические сведения

Самым мощным инструментом в **C,** безусловно, являются указатели и для того, чтобы овладеть программированием в **C**, необходимо овладеть умением использовать указатели.

С помощью указателей в **C** можно: получать доступ к адресам памяти объектов и манипулировать с ними, строить одномерные, двумерные и многомерные массивы, создавать динамические структуры данных, указатели помогают передавать массивы и функции в другие функции и т.д.

Рассмотрим основные понятия и принципы работы с указателями.

Указатель - это адрес памяти, распределённой для другой переменной заданного типа. Значение указателя сообщает о том, где расположен объект данных, но не говорит о его содержимом.

Синтаксис объявления указателей:

<тип>*<имя_указателя>.

Читается данная запись так:

<имя_указателя> является указателем на **<тип>.** Символ «*» (звездочка) говорит о том, что данная переменная есть указатель на заданный тип.

Пример:

int * x; // x является указателем на тип **int** (целое).

Таким образом, можно объявить указатель на любой тип: стандартный или созданный пользователем, в том числе может быть объявлен указатель на указатель, на любой тип-**void.**

Указатель на тип **void** совместим с любым другим указателем. Например, допустима запись:

int *x;	**void *y;**	**y = x;**

Размер указателя, т.е. размер участка памяти, отведенного под адрес, зависит от модели памяти, в которой пишется программа. Отметим их название и укажем размер указателей в байтах: крошечная **(2)**, маленькая **(2)**, средняя **(4),** компактная **(4),** большая **(4),** огромная **(4).**

Для указания размера указателя используют модификаторы: по умолчанию **near** (2 байта), **far** (4 байта), **huge** (4 байта).

Указатель может использоваться как константный, который связан с одной постоянной ячейкой памяти:

float * const ptr; // константный указатель

Указатель может быть связан и с константами:

const float * ptr; // указатель на константный тип

Возможно также следующее выражение:

const float* const ptr; // константный указатель на константный тип.

Основные операции с указателями

Наиболее важные операции с указателями: операция обращения по адресу "*" (называется также разыменованием указателя, разадресацией, операцией косвенной адресации) и операция определения адреса "&". Операция обращения по адресу служит для доступа к данным описанного типа, т.е. для присвоения и считывания данных.

Пример:

int*x; int d; // не путать разадресацию со звёздочкой при объявлении указателя
int y = *x // y = 4;

В первой строке переменной-указателю **x** присваивается адрес, где хранится значение переменной **d** во второй строке переменной **y** присваивается разыменованное значение указателя **x,** т.е. значение 4. Если первую операцию написать без адреса, т.е. **x = d** (а **x** объявлен как **int*x**), то **x** будет указывать на ячейку памяти с адресом ??:4 (?? - в зависимости от модели памяти) и значение переменной **y** будет равно значению, которое хранится по указанному адресу.

Операция определения адреса & возвращает адрес памяти своего операнда (переменной, константы, функции и т. д.). Формат получения адреса следующий:

<адрес> = & <переменная>;

Пример: **int x, *y; y = &a;** // указатель **y** содержит адрес **x**

Кроме того, указатели можно присваивать друг другу, если они одного типа.

Пример: **int a, *x, *y; x =y;**

Всем указателям для инициализации можно присваивать константу **NULL**, при этом гарантируется, что этот адрес не совпадает ни с каким другим адресом, зарезервированным системой. Операции с указателями будут также рассмотрены при работе с массивами.

Указатели и целые величины

Выражение целого типа может складываться и вычитаться из переменной типа **указатель.** Два указателя на объекты одного и того же типа могут вычитаться; в этом случае результат имеет целый тип.

```
char*a = "Slovo", c;
c = *(a+3);  // c = 'v'
char*a, *b = "ABCDE";
a = b + 2; c = *(- - a;
char k = *a; int d = b-a)
```

Динамическое размещение указателей в памяти

Чтобы не вызвать конфликт в операционной системе или не нарушить работу приложения, нужно выделять место для указанного типа данных в допустимом пространстве памяти, которое называется «**heap**» или **«куча».**

Выделение (если есть возможность) памяти и присвоение её адреса указателю, по которому можно работать с описанным типом осуществляется библиотечной функцией **malloc ( )** или **alloc()**, (заголовочный файл **<mem.h>),** которую необходимо предварительно включить в программу директивой # **include**. Эти функции при последующих выделениях памяти предотвращают конфликты между указателями.

Формат использования данной функции:

<указатель> = (<тип_указателя>*) malloc (<размер_выделяемой_памяти_байт>);

Указатель на один тип может быть преобразован в указатель на другой тип. При этом обеспечивается возможность преобразования указателя на объект данного размера в указатель на объект меньшего размера в указатель на объект меньшего размера и обратного без изменений.

Например, функция динамического распределения памяти может воспринимать размер (в байтах) объекта, для которого выделяется память, и возвращать указатель на тип **void**

double*dp;
d = (double*) malloc (sizeof(double)); // функция **malloc ()** выделяет память
free dp: *dp = 22. / 7.0 // освобождение динамических выделений памяти.

При использовании функции обеспечивается преобразование возвращаемого значения из указателя на тип **void** в указатель на тип **double.**

Пример

```
/*    ЗАНЯТИЕ N 8
 Выполнил студент группы .....   Петров Ю.В.
 Применение указателей при работе с переменными
 различных типов, изменение значений переменных по адресу
 и по ссылке. Примеры операций с указателями  */

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <alloc.h>
  int a;       //Объявление глобальной переменной типа int
int main(void)
  {char c1,c2,buf[20];     //buf-указатель на тип char
   char   *pc;
   char   *pst="\"slovo\"";//Объявление указателей на тип char
   int    *pi= &a;
   float  *pf, f=26.6;     //Объявление указателя на тип float
```

```
double *pd, d;          //Объявление указателя на тип double
double &sd= d;          //Объявление ссылки на тип double
void   *pv;             //Объявление указателя на тип void
char   *pchar=(char *)malloc(20);//Выделение памяти в куче
clrscr();   // Объявление функции void *malloc(size_t size);
pc=&c1;     pf=&f;       pd=&d;
printf(" Ввод переменной c1 типа char:   ");
scanf("%c",pc);   //Функция ввода данных, & - операция взятия
  //адреса отсутствует
printf(" Вывод переменной c1 типа char:  ");
printf("%c\n",*pc);
fflush(stdin);
pc=&c2;
printf(" Ввод переменной c2 типа char:   ");
scanf("%c",pc);   //Функция ввода данных, & - операция взятия
  //адреса отсутствует
printf(" Вывод переменной c2 типа char:  ");
printf("%c\n",*pc);
printf("\n Ввод переменной (a) типа int:  ");
scanf("%d",pi);
a+=5; ++*pi;       //Изменение a = a+5+1
printf(" \t Вывод (a) после изменения a=a+5+1\n");
printf(" Формат вывода (int): +6d     #6o     #8x\n");
printf("\t\t     |%+6d|%#6o|%#8x|\n ",a,*pi,*pi);
printf("\n Вывод исходной строки:  %s\n",pst);
pc=pst;
printf(" Вывод строки в цикле:\n");
while(*pc!='\0') { printf("  %s",pc); pc++; }
printf("\n Ввод строки в массив:    ");
scanf("%s",buf);
pv=buf;             //Использование указателя (pv) на тип void
printf(" Вывод строки из массива: %s\n",(char *)pv); //pv -void
printf(" Ввод строки в динамическую память:   ");
scanf("%s",pchar);
printf(" Вывод строки из динамической памяти: %s\n",pchar);
printf(" Ввод переменных типа float and double (через пробел):\n");
printf("\t\t      ");
scanf("%f %lf",pf,pd);
pv=pf;              //Использование указателя (pv) на тип void
*pf=*(float *)pv*10; //f*=10;
*pd=sd+*(float *)pv; //Использование ссылки (sd) на тип double
pv=pd;              //d+=f;
printf("\t Вывод результатов расчета f*=10   d+=f\n");
printf(" Формат вывода (float):  10.6f       10.6e       +10.6g\n");
printf("\t\t     |%10.6f|%10.6e|%+10.6g|\n",f,*pf,*pf);
printf(" Формат вывода (double): 10.8lf      10.8e       10.8g\n");
```

```
  printf("\t\t     |%10.8lf|%10.8e|%+10.8g|\n ",*pd,*(double *)pv,sd);
  getche();
  return 0;
 }
/* Ввод переменной  c1 типа char:  w
   Вывод переменной c1 типа char:  w
   Ввод переменной  c2 типа char:  t
   Вывод переменной c2 типа char:  t

   Ввод переменной (a) типа int:   40
   Вывод (a) после изменения a=a+5+1
Формат вывода (int): +6d     #6o     #8x
    |   +46|   056|    0x2e|

Вывод исходной строки:  "slovo"
Вывод строки в цикле:
  "slovo"  slovo"  lovo"  ovo"  vo"  o"  "
Ввод строки в массив:    unsigned
Вывод строки из массива: unsigned
Ввод строки в динамическую память:    dinamo
Вывод строки из динамической памяти:  dinamo
Ввод переменных типа float and double (через пробел):
     1.5       20.4
Вывод результатов расчета *pf=*pf*10; *pd=*pd+f;
Формат вывода (float):  10.6f       10.6e       +10.6g
   | 15.000000|1.500000e+01|       +15|
Формат вывода (double): 10.8lf      10.8e        10.8g
   |35.40000000|3.54000000e+01|  +35.4|
 */
```

Ход работы

1. Изучить теоретические сведения.
2. В соответствии с индивидуальным заданием разработать алгоритм применения указателей.
3. Разработать программу, содержащую указатели на скалярные типы данных, показать использование указателей в арифметических операциях.
4. Набрать программу на компьютере и устранить ошибки.
5. .Получить результат.
6. Оформить отчет и сделать выводы по работе.
7. Подготовиться к защите лабораторной работы, изучив контрольные вопросы по данной теме.

Индивидуальное задание к лабораторной работе №8

Присвоить раз именованному указателю на тип P1 значение арифметического выражения AB включающего указатели на типы P2 и P3. Арифметическое

выражение реализовать в виде функции возвращающей указатель на тип P1. Вывести на экран значение указателя P2 и значение на которое он ссылается. Индивидуальные задания взять из таблицы 8.1.

Таблица 8.1 - индивидуальные задания

Вариант	P1	AB	P2	P3
1	long	$(1/\sin((p2)^2))^{p3}$	int	float
2	float	$(\text{abs}(p3))^{1/p2}$	long	double
3	double	$\tan((p3)^2)^{p2/3}$	int	long
4	float	$(\ln(p2)^{p3})^{p2}$	char	unsigned long int
5	long double	$\sin(\text{abs}(p3)^{p2})$	double	long int
6	long	sin(p2)/tan(p3)	int	float
7	unsigned long int	(++p3)/(--p2)	short int	int
8	long double	$((1+(++p2))/p3)^{p2}$	long int	float
9	signed long int	$(\sin(--p2)-(p3))^{p3}$	char	int
10	long int	$(1/\sin(p2))^{p3}$	unsigned long int	int
11	double	$\sin(p3)^{1/p2}$	double	float
12	double	cos(p2/p3)	int	double
13	int	(--p2)+(++p3)	unsigned int	short int
14	signed int	$(\sin(p2)/\tan(p3))^{p3}$	short int	char
15	long double	$\ln(--p2)^{1/p3}$	float	double
16	double	1.2*(10-(--p3))+p3	double	short int
17	double	$\tan((p3)^2)^{p2/3}$	double	float
18	float	$(\ln(p2)^{p3})^{p2}$	int	double
19	long double	$\sin(\text{abs}(p3)^{p2})$	unsigned int	short int
20	long double	$((1+(++p2))/p3)^{p2}$	double	long int
21	signed long int	$(\sin(--p2)-(p3))^{p3}$	int	float
22	long int	$(1/\sin(p2))^{p3}$	short int	int
23	long double	$\sin(\text{abs}(p3)^{p2})$	char	int
24	long	sin(p2)/tan(p3)	unsigned long int	int
25	double	$\sin(p3)^{1/p2}$	double	float
26	double	cos(p2/p3)	int	double
27	int	(--p2)+(++p3)	unsigned int	short int
28	long double	$((1+(++p2))/p3)^{p2}$	short int	char
29	signed long int	$(\sin(--p2)-(p3))^{p3}$	float	double
30	long int	$(1/\sin(p2))^{p3}$	double	short int

Требования к содержанию отчёта приведены в лабораторной работе №1.

Контрольные вопросы для подготовки и самостоятельной работы

1 Дайте определение указателю.
2 Какой синтаксис объявления указателя?
3 С какими модификаторами может быть использован указатель?
4 Чем отличается (*) в объявлении указателя и в выражении, где он используется?
5 Что такое операция обращения по адресу?
6 Что такое операция взятия адреса?
7 Как бы Вы присвоили адрес переменной с плавающей точкой salary указателю с именем pt_sal?
8 Обязательно ли инициализировать указатель при его объявлении?
9 В чём особенности использования указателя на тип void?
10 Для чего применяется константа NULL?
11 Что такое динамическое выделение памяти, где она выделяется и с помощью каких функций?
12 Можно ли присваивать указатели одного типа?
13 Как осуществить приведение типа указателя?
14 Объясните примеры, приведенные в теоретической части.

Лабораторная работа №9

Массивы. Селективная обработка массивов

(2 часа)

Цель работы: изучить работу с массивом как с составным типом данных, приёмы ввода и вывода данных, обработку одномерных массивов.

Теоретические сведения

Массив - это набор объектов (элементов) одинакового типа, доступ к которым осуществляется прямо по индексу в массиве. Обращение к массивам в **C** реализовано с помощью указателей **(pointers).**

Массив в **C** можно объявить следующим образом:

[Класс_памяти>] <Тип_данных> <Имя_массива> [<Размер_массива>];

<Размер_массива> может быть задан константным выражением.

Доступ к элементам массива происходит следующим образом:
<Имя_массива>[<Значение_индекса>], т.е., используя имя массива и индекс от 0 до (**<Размер_массива> -1),** т.е. на единицу меньше, чем <**Размер_массива>.**
Пример объявления массива**:**

```
char name[20];
int grad[125];
float b[30];
```

Массив имеет имя (**name)**, которое является указателем на тип элементов массива и адресует первый элемент с индексом **(∅)**. Имя массива фактически является константным указателем, ссылающимся на начальный адрес данных, с которого расположен сам массив и может быть использовано как указатель. Обращение к элементам может осуществляться следующим образом **name[0]** -1-ый элемент массива, **name[1]-** 2-ой элемент**, name[19]** – последний 20-ый элемент.

При трансляции программы компилятор отводит место под объявленный массив статически, т.е. в области данных участок памяти, выделенный для массива, не может быть динамически изменен. Размер выделяемой памяти под массив можно определить по следующей формуле:

(sizeof(тип)*<Размер_массива>) байтов.

Массив размещается последовательно в памяти, т.е. каждая следующая ячейка расположена сразу после предыдущей.

В **C** не включены стандартные операторы, которые осуществляют работу с массивами, как стандартными типами данных. Чтобы скопировать данные из одного массива в другой необходимо организовать функцию, тело которой должно иметь примерно следующий вид:

```
for(i=0; i<rasm; i++) mas2[i] = mas1[i];
```

где **rasm** - размер массивов, соответствующий указанному в их объявлении

```
# define rasm 20
int mas2[rasm], mas1[rasm];
```

Пример

```
/*    ЗАНЯТИЕ N 9
 Выполнил студент группы .....   Петров Ю.В.
 Применение массивов при работе с переменными
 различных типов (int,float), изменение значений
 переменных. Примеры приемов работы с массивами  */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define N 15
#define M 10
int p[N],   x[N], i, j, j1, count=0;
int y[N*2], z[N]={15,37,10,-20,-10,25,27,30,89,67,\
                  24,-6,22};
int   *pi=y;        //Два последних элемента z[N] равны 0
float *pf, mf[N];
void main()
 { clrscr();   randomize();
   pf=mf;
   printf("Исходный массив p[]: \n");
   for (i=0;i<N;i++)
    { p[i]=random(81)-40;  //Инициализация с помощью функции
      printf("%4d",p[i]);  //random() в интервале +40...-40
      if (p[i]>0)     //Выбор элементов >0 в массив x[]
            { x[count]=p[i];
              mf[count]=x[count]/5.5;
              count++;
            }
     }
    printf("\n");
    printf("Исходный массив z[j]: \n");
    for (i=0;i<N;i++) printf("%4d",z[i]);
    printf("\nРабочий массив x[count]>0 из p[i]: \n");
    for (i=0;i<count;i++) printf("%4d",x[i]);
    j=0;//Выбор элементов в массив y[] из z[] и p[] по условиям
    for (i=0;i<N;i++)
     {if (z[i]>=-M && z[i]<=(M+N) && (z[i]%2==0))
           { *pi = z[i];  pi++; j++;}
      if (p[i]>=-M && p[i]<=(M+N) && (p[i]%2==0))
           { *(y+j)=p[i]; pi++; j++;}
     }
    j1=j;
    printf("\nРабочий массив y[j] из z[i] и p[i] по условиям \n");
    printf(" -M<y[j]<(M+N) и четное:\n");
```

```
    for (i=0;i<j1;i++)  printf("%4d",y[i]);
    printf("\nРабочий массив mf[j]=x[j]/5.5 \n");
    for (i=0;i<count;i++)  printf("%6.4f ",*(pf+i));
      //Сортировка массива x[]
    for (i=0;i<count;i++)
     for (j=i;j<count;j++)
        if (x[i]<x[j])
          {int c=x[i];x[i]=x[j];x[j]=c;
          }
        else continue;
     //Сортировка массива y[]
    for (i=0;i<j1;i++)
     for (j=i;j<j1;j++)
        if (y[i]<y[j])
          {int c=y[i];y[i]=y[j];y[j]=c;
          }
        else continue;
    printf("\nСортированный массив x[]: \n");
    for (i=0;i<count;i++)
      printf("%4d",x[i]);
    printf("\nСортированный массив y[]: \n");
    for (i=0;i<j1;i++)
      printf("%4d",y[i]);
    getch();
 }
/*
Исходный массив p[]:
  32  37 -34  11  30   8 -11 -38   6 -21 -27 -23 -30 -19 -29
Исходный массив z[j]:
  15  37  10 -20 -10  25  27  30  89  67  24  -6  22   0   0
Рабочий массив x[count]>0 из p[i]:
  32  37  11  30   8   6
Рабочий массив y[j] из z[i] и p[i] по условиям
       -M<y[j]<(M+N) и четное:
  10 -10   8   6  24  -6  22   0   0
Рабочий массив mf[j]=x[j]/5.5
  5.8182 6.7273 2.0000 5.4545 1.4545 1.0909
Сортированный массив x[]:
  37  32  30  11   8   6
Сортированный массив y[]:
  24  22  10   8   6   0   0  -6  -10    */
```

Ход работы

1 Изучить теоретические сведения.

2 В соответствии с индивидуальным заданием разработать алгоритм инициализации массива, селективной обработки массива.
3 Разработать и набрать программу, отладить её на компьютере, изучить работу операторов.
4 Получить результаты.
5 Оформить отчет.
6 Подготовиться к защите лабораторной работы, изучив контрольные вопросы по данной теме.

Требования к содержанию отчёта приведены в лабораторной работе №1

Индивидуальное задание к лабораторной работе №9

Составить программу для обработки массива согласно индивидуальному заданию приведенному в таблице 9.1.

Таблица 9.1 - индивидуальное задание

Вар.	Условие задачи
1	Найти сумму четных чисел массива
2	Вычислить произведение отрицательных чисел массива
3	Определить количество нечетных чисел массива
4	Найти сумму отрицательных чисел массива
5	Определить количество отрицательных чисел массива
6	Вычислить произведение положительных чисел массива
7	Найти сумму положительных чисел массива
8	Определить количество четных чисел массива
9	Вычислить произведение четных чисел массива
10	Найти сумму нечетных чисел массива
11	Определить количество кратных 3 чисел массива
12	Вычислить произведение нечетных чисел массива
13	Найти сумму кратных 3 чисел массива
14	Определить количество не кратных 3 чисел массива
15	Вычислить произведение кратных 3 чисел массива
16	Найти сумму не кратных 3 чисел массива
17	Определить количество кратных 4 чисел массива
18	Вычислить произведение не кратных 3 чисел массива
19	Найти сумму кратных 4 чисел массива
20	Определить количество не кратных 4 чисел массива
21	Вычислить произведение кратных 4 чисел массива
22	Найти сумму не кратных 4 чисел массива
23	Определить количество кратных 5 чисел массива
24	Вычислить произведение не кратных 4 чисел массива
25	Найти сумму кратных 5 чисел массива
26	Вычислить среднее арифметическое положительных четных элементов массива
27	Найти среднее геометрическое отрицательных нечетных элементов массива

28	Найти произведение отрицательных не кратных пяти элементов массива
29	Найти среднее арифметическое элементов массива, находящихся в интервале [-10,20]
30	Найти среднее геометрическое элементов массива, находящихся в интервале [5,20]

Контрольные вопросы для подготовки и самостоятельной работы

1 С какого числа начинается индексация массивов в языке С?
2 Как объявляется 1-но мерный массив?
3 Какие типы языка **С** можно и нельзя указывать в качестве типа при объявлении массива?
4 В каких случаях размерность массива при объявлении можно не указывать?
5 Какой тип имеет имя массива?
6 Как осуществляется инициализация элементов массива?
7 Как можно инициализировать массив с элементами типа **char**?
8 Можно ли использовать средство **typedef** для объявления типа “массив”?
9 Какие альтернативные формы записи элементов массива можно использовать? Приведите примеры.
10 Каковы правила использования индексных выражений?
11 Существуют ли операции работы с массивами?
12 Какие классы памяти можно использовать при объявлении массивов?

Лабораторная работа №10

Формирование рабочих массивов с помощью операций селекции исходного массива (2 часа)

Цель работы: изучить и научиться применять обработку массивов по заданным логическим условиям, формирование новых массивов.

Теоретические сведения

Смотри теоретические сведения по предыдущей работе.

Пример

```
/*    ЗАНЯТИЕ N 10
   Разработал Петров Ю.В.
   Объявить массивы заданной размерности, выполнить их
   инициализацию с применением функции random() и явно.
   Получить доступ к элементам массивов с использованием
   операторов организации цикла. Переписать положительные
   элементы массива x[1],x[2],...,x[N] в массив y[t], а
   отрицательные - в массив z[p]. Элементы в массивах y[t]
   и z[p] располагать подряд, элементы массива y1[t]
   рассчитать для положительных x[i] по формуле
   y1[t]=y1[t]+2*exp(b*y[t]-y[t]*y[t]),где b=N/5     */

  #include <stdlib.h>
  #include <stdio.h>
  #include <conio.h>
  #include <math.h>
  # define N 20
  main()
  { clrscr();
    randomize();
    int i,t=0,p=0,b=N/5;
    int x[N],y[N],z[N]; //Объявление одномерного массива (вектора)
    float y1[N]={2.3,3.4,4.0,5.2,6.6,7.9,8.34,9.8,10.4,11.2,\
    10.1,9.9,8.7,7.5}; //Инициализация первых 14, остальные -0
    printf("\nВывод элементов исходного массива y1[ ]\n");
    for(i=0;i<N;i++)
      {printf(" y1[%2d]=%4.1f",i+1,y1[i]);
       if ((i+1)%5==0) printf("\n");
      }
    printf("\nИнициализация одномерного массива x[N]\n");
    for(i=0;i<N;i++)
      { x[i]=random(40)-20;
```

```
        printf(" x[%2d]=%2d ",i+1,x[i]);    //Вывод элементов
        if ((i+1)%5==0) printf("\n");       //массива x[ ]
        if (x[i]>=0)           //Выбор положительных значений
           { y[t]=x[i];    //Расчет элементов массива y1[ ]
            y1[t]=y1[t]+2*exp(b*y[t]-y[t]*y[t]); t++;
           }
        else {z[p]=x[i]; p++;} //Выбор отрицательных значений
     }
   printf("\nВывод элементов массива y[ ]>0\n");
   i=0;
   while ( i<t )
    { printf(" y[%2d]=%2d ",i+1,y[i]);
        if ((i+1)%5==0) printf("\n");
        i++;
    }
   printf("\n");
   printf("\nВывод элементов массива y1[ ] после расчета\n");
   i=0;
   do
    { printf(" y1[%2d]=%3.2f",i+1,y1[i]);
        if ((i+1)%5==0) printf("\n");
        i++;
     } while ( i<t );
   printf("\n");
   printf("\nВывод элементов массива z[ ]<0\n");
   for(i=0;i<p;i++)
    { printf(" z[%2d]=%2d ",i+1,z[i]);
        if ((i+1)%5==0) printf("\n");
    }
   getch();
   return 1;
 }
/*   Вывод элементов исходного массива y1[ ]
  y1[ 1]= 2.3 y1[ 2]= 3.4 y1[ 3]= 4.0 y1[ 4]= 5.2 y1[ 5]= 6.6
  y1[ 6]= 7.9 y1[ 7]= 8.3 y1[ 8]= 9.8 y1[ 9]=10.4 y1[10]=11.2
  y1[11]=10.1 y1[12]= 9.9 y1[13]= 8.7 y1[14]= 7.5 y1[15]= 0.0
  y1[16]= 0.0 y1[17]= 0.0 y1[18]= 0.0 y1[19]= 0.0 y1[20]= 0.0

    Инициализация одномерного массива x[N]
  x[ 1]=  8  x[ 2]= 10  x[ 3]=- 7  x[ 4]= 13  x[ 5]=- 1
  x[ 6]=-14  x[ 7]=  5  x[ 8]= 17  x[ 9]=-14  x[10]=-19
  x[11]= 13  x[12]=  8  x[13]=-10  x[14]=-16  x[15]=  5
  x[16]=  9  x[17]=-11  x[18]=-12  x[19]=-16  x[20]=- 3

    Вывод элементов массива y[ ]>0
  y[1]= 8    y[2]=10    y[3]=13    y[4]= 5   y[5]=17
```

y[6]=13 y[7]= 8 y[8]= 5 y[9]= 9

Вывод элементов массива y1[ ] после расчета
y1[1]=2.30 y1[2]=3.40 y1[3]=4.00 y1[4]=5.21 y1[5]=6.60
y1[6]=7.90 y1[7]=8.34 y1[8]=9.81 y1[9]=10.40

Вывод элементов массива z[ ]<0
z[1]=- 7 z[2]=- 1 z[3]=-14 z[4]=-14 z[5]=-19
z[6]=-10 z[7]=-16 z[8]=-11 z[9]=-12 z[10]=-16
z[11]=- 3 */

Ход работы

1 Изучить теоретические сведения
2 В соответствии с индивидуальным заданием, на основе программы предыдущей работы разработать алгоритм, обеспечивающий формирование рабочих массивов по заданным логическим условиям. Операции с массивами вынести в отдельную функцию.
3 Разработать программу, набрать и отладить программу на компьютере.
4 Изучить работу операторов.
5 Получить результаты.
6 Оформить отчет.
7 Подготовиться к защите лабораторной работы, изучив контрольные вопросы по данной теме.

Требования к содержанию отчёта приведены в лабораторной работе №1.

Индивидуальное задание к лабораторной работе №10

Составить программу для обработки массивов согласно индивидуальному заданию приведенному в таблице 10.1.

Таблица 10.1 - Индивидуальное задание

Вар.	**Условие задачи**
1	Дан массив *X(15)*. Сформировать новый массив из четных чисел исходного
2	Дан массив *X(25)*. Сформировать новый массив из нечетных чисел исходного
3	Дан массив *D(15)*. Сформировать новый массив из кратных 3 чисел исходного
4	Дан массив *A(10)*. Сформировать новый массив из отрицательных чисел исходного
5	Дан массив *Z(15)*. Сформировать новый массив из положительных четных чисел исходного
6	Дан массив *X(25)*. Сформировать новый массив из чисел исходного, лежащих в интервале [-3,7]
7	Дан массив *Y(10)*. Сформировать новый массив из нечетных положитель-

	ных чисел исходного
8	Дан массив *D(12)*. Сформировать новый массив из положительных кратных 3 чисел исходного
9	Дан массив *A(8)*. Сформировать новый массив из отрицательных четных чисел исходного
10	Дан массив *C(15)*. Сформировать новый массив из больших 8 чисел исходного
11	Дан массив *B(21)*. Сформировать новый массив из кратных 4 чисел исходного
12	Дан массив *A(12)*. Сформировать новый массив из отрицательных нечетных чисел исходного
13	Дан массив *X(8)*. Сформировать новый массив из отрицательных не кратных 3 чисел исходного
14	Дан массив *G(9)*. Сформировать новый массив из четных чисел исходного массива, стоящих на нечетных местах
15	Дан массив *Y(15)*. Сформировать новый массив из нечетных, кратных 3 чисел исходного
16	Дан массив *A(18)*. Сформировать новый массив из нечетных, кратных 5 чисел исходного
17	Дан массив *Z(10)*. Сформировать новый массив из четных чисел исходного, лежащих в интервале [1,12]
18	Дан массив *A(11)*. Сформировать новый массив из нечетных чисел исходного, лежащих в интервале [-3,15]
19	Дан массив B*(10)*. Сформировать новый массив из номеров отрицательных четных чисел исходного
20	Дан массив *A(8)*. Сформировать новый массив из номеров отрицательных нечетных чисел исходного
21	Дан массив *C(12)*. Сформировать новый массив из отрицательных чисел исходного, стоящих на четных местах
22	Дан массив *F(13)*. Сформировать новый массив из отрицательных чисел исходного, стоящих на нечетных местах
23	Дан массив *H(12)*. Сформировать новый массив из положительных чисел исходного, стоящих на четных местах
24	Дан массив *V(19)*. Сформировать новый массив из отрицательных чисел исходного, лежащих в диапазоне [-20,-5]
25	Дан массив *N(11)*. Сформировать новый массив из отрицательных кратных 5 чисел исходного
26	Дан массив *K(15)*. Сформировать новый массив из положительных чисел исходного, стоящих на нечетных местах
27	Дан массив *Y(11)*. Сформировать новый массив из отрицательных не кратных 5 чисел исходного
28	Дан массив *Z(14)*. Сформировать новый массив из положительных кратных 5 чисел исходного
29	Дан массив *R(13)*. Сформировать новый массив из отрицательных кратных 10 чисел исходного

30	Дан массив *N(11)*. Сформировать новый массив из отрицательных кратных 8 чисел исходного

Контрольные вопросы для подготовки и самостоятельной работы

1 Как производится доступ к элементам массива?
2 Какое количество операторов цикла необходимо для обработки главной или побочной диагонали массива?
3 Какие методы сортировки элементов Вы знаете?
4 Можно ли использовать указатель на тип элементов массива в качестве имени массива и что для этого необходимо?
5 Адрес какого элемента содержит имя массива?
6 Какие классы памяти можно использовать при объявлении массива?
7 Какие классы памяти используются по умолчанию?
8 Как размещаются элементы массива в памяти?
9 Как определяется количество байтов, на которое смещается указатель индексного выражения? Зависит ли смещение указателя от типа элементов массива?

Лабораторная работа №11

Обработка символьных данных
(2 часа)

Цель работы: изучить и научиться использовать массивы символьных данных.

Теоретические сведения

Теоретические сведенья приведены в лабораторных работах N3, N8 и N9.

Ход работы

1 Изучить теоретические сведения
2 В соответствии с индивидуальным заданием разработать алгоритм и программу для обработки символьных данных, представленных в виде массивов без применения библиотечных строковых функций. Для работы со строками использовать указатели на тип **char,** массивы указателей.
3 Набрать и отладить программу на компьютере.
4 Изучить работу операторов.
5 Получить результаты.
6 Оформить отчет.
7 Подготовиться к защите лабораторной работы, изучив вопросы по данной теме.

Требования к содержанию отчёта приведены в лабораторной работе №1.

Индивидуальное задание к лабораторной работе №11.

Составить программу для обработки символьных данных согласно индивидуальному заданию приведенному в таблице 11.1.

Таблица 11.1

Вариант	Задание
1	Ввести с клавиатуры предложение (слова отделенные пробелом). Пометить местами первое и последнее слова.
2	Ввести с клавиатуры предложение. Поменять местами четные и нечетные, по порядку слова.
3	Ввести с клавиатуры предложение. Произвести перестановку слов в предложении в обратном порядке.
4	Ввести с клавиатуры предложение. Произвести перестановку букв в словах в обратном порядке.
5	Ввести два предложения. Добавить второе предложение к первому, отделив их пробелом.
6	Ввести с клавиатуры предложение. Произвести вставку слова «не» перед

	каждым третьим словом в предложении.
7	Ввести с клавиатуры предложение. Произвести вставку запятой после слов заканчивающихся на «й». Вывести на экран количество таких вставок.
8	Ввести с клавиатуры предложение. Произвести вставку запятой перед словами начинающихся с букв «по».
9	Ввести с клавиатуры предложение. Слова заканчивающиеся на «ие» удалить.
10	Ввести с клавиатуры предложение. В словах заканчивающихся на «е» заменить эту букву на «я».
11	Ввести с клавиатуры предложение и слово. Произвести вставку слова между словами предложениями.
12	Ввести с клавиатуры предложение. Поменять местами второе и последнее слово.
13	Ввести с клавиатуры предложение. Произвести преобразование нижнего регистра в верхний.
14	Ввести с клавиатуры предложение. Произвести преобразование из верхнего в нижний.
15	Вывести строку с буквами верхнего и нижнего регистра. Произвести инвертирования регистра.
16	Ввести с клавиатуры предложение. Произвести перестановку букв в строке согласно таблице.
17	Ввести с клавиатуры предложение. Отсортировать слова в предложении в алфавитном порядке.
18	Ввести с клавиатуры предложение. Отсортировать слова в предложении по возрастанию.
19	Ввести с клавиатуры предложение. Подсчитать количество гласных букв в каждом слове предложения.
20	Ввести с клавиатуры предложение. Подсчитать количество согласных букв в каждом втором слове предложения.
21	Ввести с клавиатуры предложение. Отсортировать буквы в каждом слове предложения в порядке убывания: букву «а» считая последней, букву «я» считая первой.
22	Ввести с клавиатуры предложение. Подсчитать длину каждого слова в предложении. Найти номер самого длинного и самого короткого слова.
23	Ввести с клавиатуры предложение. Найти самое длинное и самое короткие слова и поменять их местами.
24	Ввести с клавиатуры предложение. Сформировать массив из длин слов в предложении.
25	Ввести с клавиатуры предложение. Найти среднюю длину слов в предложении. Вывести на экран самое длинное слово, и самое короткое слово, самое «среднее» слово.
26	Ввести с клавиатуры предложение. Произвести упаковку и распаковку предложений (повторяющиеся комбинации символов заменить каким-либо одним символом).

27	Вывести два предложения. Соединить предложения и отсортировать слова в порядке обратном алфавитному.
28	Ввести с клавиатуры предложение. Разбить предложение на два.

Контрольные вопросы для подготовки и самостоятельной работы

1 Укажите способы объявления символьных строк (переменных и констант).
2 Какой символ ставится в конце строки?
3 Сколько памяти занимает один символ строки?
4 Сколько памяти занимает строка (слово) из **6** букв?
5 Можно ли использовать указатели для адресации символьных строк, содержащихся в массивах?
6 Сколько указателей можно использовать для работы с массивами?
7 Какой объект адресуется указателем, смещенным относительно начала строки на **n** байтов?
8 Какая функция позволяет определить длину строки?
9 Какие операции применимы к указателям и необходимы для работы со строками?
10 Можно ли инициализировать массивы с данными типа **char** строками символов?
11 Можно ли инициализировать указатели на тип **char** строками символов?
12 Какие символьные последовательности называют управляющими, что они означают?

Лабораторная работа № 12

Использование библиотечных функций для работы с символьными данными (2 часа)

Цель работы: выработать практические навыки в написании программ с использованием библиотечных функций для работы с символьными данными.

Теоретические сведения

Функции для работы с символьными данными

Для работы с символьными данными разработан ряд библиотечных функций. Рассмотрим некоторые из них:

char *strcat(char *dest, const char *src); - добавляет копию строки **src** в конец **dest**. Длина результирующей строки равна **strlen(dest) + strlen(src).**

strcat() - возвращает указатель на результирующую строку.

Функция **char *strchr(const char *s, int c)**; - просматривает строку в прямом направлении для отыскания в ней заданного символа. Функция **strchr()** ищет первое вхождение символа **c** в строку **s.** Символ конца строки считается частью строки, поэтому - **strchr(strs, 0)** вернет значение указателя на нулевой конечный символ строки **strs**. Функция **strchr()** возвращает указатель на первое вхождение символа **c** в **s**, если **c** не обнаружен в **s**, то **strchr()** возвращает нуль.

int strcmp(const char *s1, const char *s2); - осуществляет беззнаковое сравнение строк **s1** и **s2**, начиная с первого символа каждой строки, до тех пор, пока очередные соответствующие символы в строках не будут различны или пока не будут достигнуты концы строк. Функция **strcmp**() возвращает значение:

< 0 если **s1** меньше чем **s2;**
= = 0 если **s1** равна **s2;**
> 0 если **s1** больше чем **s2.**

int strcmpi(const char *s1, const char *s2); - сравнивает одну строку с другой аналогично s**trcmp(),** но без различия больших и маленьких букв. Функция **strcmpi()** определена как макрос в **string.h** и преобразует вызовы **strcmpi()** к вызовам **strcmp()**. Макрос обеспечивает совместимость с другими компиляторами **C.**

int strncmp(const char *s1, const char *s2, size_t maxlen); - сравнивает часть одной строки с частью другой. Функция **strncmp()** производит такое же беззнаковое сравнение, как и **strcmp(),** но просматривает не более, чем **maxlen** символов. Она начинает с первого символа каждой строки и заканчивает, когда очередные символы в строках различны или когда проверено **maxlen** символов. Функция **strncmp**() возвращает значение типа **int**, смысл которого аналогичен **strncmp().**

char *strcpy(char *dest, const char *src); - копирует строку **src** в **dest,** завершая работу после копирования символа окончания строки. Функция **strcpy()** возвращает указатель на результирующую строку **(dest).**

size_t strcspn(const char *s1, const char *s2); - ищет в строке первый сегмент, не содержащий ни одного символа из заданного набора символов. Функция

strcspn() возвращает длину первого встретившегося сегмента строки **s1**, состоящего только из символов, не входящих в строку **s2.**

size_t strlen(const char *s); - вычисляет длину строки **s.** Функция **strlen()** - возвращает число символов в **s**, не считая символ конца строки.

char *strlwr(char *s); - преобразует буквы верхнего регистра **(A-Z)** строки **s** в буквы нижнего регистра **(a-z).** Другие символы не изменяются. Функция **strlwr()** возвращает указатель на строку **s.**

char *strnset(char *s, int ch, size_t n); - заменяет заданное количество символов **n** в строке **s** на указанный символ **ch.** Функция **strnset()** копирует символ **ch** в первые **n** байтов строки **s**. Если **n > strlen(s)**, тогда **strnset()** заменяет, все символы и останавливается, когда достигнут конец строки (обнаружен нулевой символ).

Функция **strnset()** - возвращает указатель на изменённую строку **s.**

char *strpbrk(const char *s1, const char *s2); ищет в строке первое вхождение любого символа из переданного функции набора символов. Функция **strpbrk()** - осуществляет поиск в строке **s1** первого вхождения любого из символов, определяемых строкой **s2.** Функция **strpbrk()** - возвращает указатель на первое вхождение любого символа строки **s2**. Если ни один символ из символов **s2** не обнаружен в **s1**, то функция возвращает нуль.

char *strrchr(const char *s, int ch); - ищет в строке последнее вхождение заданного символа. Функция **strrchr()** проверяет строку **s** в обратном направлении, производя поиск заданного символа. Функция **strrchr()** находит последнее вхождение символа **ch** в строку **s.** Предполагается, что символ окончания строки является частью строки. Функция **strrchr()** возвращает указатель на последнее вхождение символа **ch.** Если **ch** не обнаружен в **s**, то **strrchr()** возвращает нуль.

char *strrev(char *s); - переворачивает строку. Функция **strrev(),** переустанавливает все символы в строке в обратном порядке, за исключением завершающего нулевого символа. Например, **"строка\0"** будет преобразована в **"акортс\0"**. Функция **strrev()** возвращает указатель на перевернутую строку.

char *strset(char *s, int ch); - заменяет все символы строки **s** на заданный символ **ch**, заканчивая работу при обнаружении символа конца строки. Функция **strset()** - возвращает указатель **s** на результирующую строку.

char *strstr(const char *s1, const char *s2); - осуществляет поиск в строке **s2** первого вхождения в нее подстроки **s1.** Функция **strstr()** - возвращает указатель на элемент в строке **s2**, с которого начинается **s1** (указатель на **s1** в **s2**). Если **s1** не обнаружена в **s2**, то **strstr()** возвращает нуль.

Пример

```
/*   ЗАНЯТИЕ N 12
  Разработал Петров Ю.В.
Использование библиотечных функций для обработки символьных данных.
Выпполнить сортировку элементов массива list[N][4], использовать
функции strcpy(), qsort(), strcmp(), stricmp(). Разработать функцию
сравнения строк с учетом и без учета регистра -sort_function().  */
#include <stdio.h>
```

```
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#define N 8
 int sort_function( const void *a, const void *b);
 char list[N][4]={"1a3","d34","1c4","235","2h5","012","135","102"};
                  //list[N][4]-глобальный массив
 int registr=1;  //Глобальная переменная для управления режимом
  //работы функции sort_function(): 0 -с учетом регистра, 1-без
int main(void)
{ clrscr();
  int i,j;
  char *pch=NULL;
   printf("\n\tИсходный (глобальный) массив char list[N][4]:\n");
  for (i=0; i<N; i++)  printf("%s ", list[i]);
  printf("\n");
  for (i=0; i<N; i++)                //Сортировка массива
     for (int j1=i+1;j1<N;j1++)
      {j=sort_function((void *)list[i], (void *)list[j1]);
       if(j>0)  //Замена строк: char *strcpy(char *dest,
          { strcpy(pch,list[i]);         // const char *src);
            strcpy(list[i],list[j1]);
            strcpy(list[j1],pch);
          }
      }
  printf("\n\tРезультат сортировки:\n");
  for (i=0; i<N; i++) printf("%s ", list[i]);
  printf("\n");
  printf("\n\tИсходный (локальный) массив char list[N][4]:\n");
  char list[N][4]={"abc","cad","tre","Cab","abb","Abc","cam","Cap"};
  for (i=0; i<N; i++) printf("%s ",list[i]);
  printf("\n");
  qsort((void *)list, N, sizeof(list[0]), sort_function);
   //Функция использует для сортировки алгоритм quicksort
   // void qsort(void *base, size_t nelem, size_t width,
   //int (*fcmp)(const void *, const void *));
  printf("\n\tРезультат сортировки без учета регистра:\n");
  registr=1;
  for (i=0; i<N; i++)  printf("%s ",list[i]);
  printf("\n");
  registr=0;
  qsort((void *)list, N, sizeof(list[0]), sort_function);
  printf("\n\tРезультат сортировки с учетом регистра:\n");
  for (i=0; i<N; i++)  printf("%s ",list[i]);
  printf("\n");
 // Функции  strrev() strcat()
```

```
 char *s = "Герой ";
 char *m = "- Gerakl";
 char *t = "retro";
 char *sapr = "cae";
 printf("\t Исходная строка :    %s \n", t);
 char *g = strrev(t);
 printf("\t Перевернутая строка: %s \n", g);
 s = strcat(s,m);//char *strcat(char *dest, const char *src);
 printf("\t Конкатенация строк : %s\n",s);
 getch();
 return 0;
}

int sort_function(const void *a, const void *b)
  { if (registr==0) //С учетом регистра
      return( strcmp((const char *)a,(const char *)b ));
//int strcmp(const char *s1, const char*s2); Возвращает:
// (<0) если s1<s; (==0) если s1==s2; (>0)  если s1>s2
   else            //Без учета регистра
      return( stricmp((const char *)a,(const char *)b ));

  }
/*  Исходный (глобальный) массив char list[N][4]:
          1a3 d34 1c4 235 2h5 012 135 102

Результат сортировки:
          012 102 135 1a3 1c4 235 2h5 d34

   Исходный (локальный) массив char list[N][4]:
          abc cad tre Cab abb Abc Cap cam

Результат сортировки без учета регистра:
          abb Abc abc Cab cad cam Cap tre

Результат сортировки с учетом регистра:
          Abc Cab Cap abb abc cad cam tre

Исходная строка :    retro
Перевернутая строка: orter
Конкатенация строк : Герой - Gerakl      */
```

Ход работы

1. Изучить теоретические сведения
2. В соответствии с индивидуальным заданием разработать алгоритм решения задачи. На базе лабораторной работы №11 реализовать задание, при этом

использовать библиотечные функции, рассмотренные на лекции и в теоретической части лабораторной работы. Результаты обработки строк (выделенные элементы) представить в массиве указателей. Обеспечить возможность передачи данных из окружения (из командной строки). При работе со строками использовать функции поиска одиночных символов и последовательностей в заданной строке.

3 Разработать программу.
4 Набрать программу на компьютере, отладить её и изучить работу операторов.
5 Получить результаты.
6 Оформить отчет.
7 Подготовиться к защите лабораторной работы, изучив вопросы по данной теме.

Требования к содержанию отчёта приведены в лабораторной работе №1.

Индивидуальные задания к лабораторной работе №12 приведены в предыдущей работе.

Контрольные вопросы для подготовки и самостоятельной работы

1 Какой заголовочный файл необходим для работы с библиотечными функциями обработки символьных данных?
2 Какой символ необходим в конце строки для нормальной работы со строками?
3 Что означает модификатор **const** при объявлении формальных параметров функций?
4 Для чего можно использовать массив указателей при работе с символьными данными?
5 Объясните форматы приведённых в теоретической части функций.
6 Что означает буква **n** в функциях **strncmp(), strnset()**?
7 Что означает буква **i** в **strcmpi()?**
8 Что означает дополнительная буква **r** в функции **strrchr()**?
9 Что адресует указатель на **n**-ный символ в строке?

Лабораторная работа № 13

Вложенные циклы. Многомерные массивы. Массивы указателей (2часа)

Цель работы: изучить конструкции языка **C** и операторы для обработки многомерных массивов с применением оператора цикла **for.**

Теоретические сведения

Массивы и указатели, индексные выражения

Идентификатор массива является указателем на первый элемент этого массива. По определению, оператор индексирования **[ ]** интерпретируется таким образом, что **A[B]** эквивалентно ***(A+B).** В соответствии с правилами преобразования, которые выполняются при операции **"+"**, если **A** является именем массива (имя массива является указателем на тип элементов массива), то индексное выражение **A[B]** указывает на **B**-ый элемент массива **A. B** является индексом массива и имеет целый тип (**int**). Индексирование является коммутативной операцией, поэтому допустимы записи **B[A],** ***(B+A).**

A[ ][ ][ ][k]

В случае использования многомерных массивов интерпретация индексного выражения следующая. Если **A[i][j][$к_i$][k]** представляет собой **n**-мерным массив с рангом **i*j *...*k**, то, если **A** встречается в выражении, он рассматривается как вектор, содержащий **i (n -1)**-мерных массивов с рангом **j*...*k**. Имя массива **A** является указателем на этот вектор. Если к указателю применяется оператор **(*)** в явном или в неявном виде (как результат индексирования), то результат будет указывать на элемент **(n-1)**-мерного массива.

Рассмотрим, например массив **int x[N][M] -** массив целых чисел размерности **N* M.** Массивы в языке **C** хранятся построчно (последний индекс изменяется быстрее), а первый индекс **(N)** в объявлении массива позволяет определить объем необходимой для массива памяти, но не играет никакой роли в вычислении смещения указателя при доступе к элементу массива – **x[i][j]**. Данный элемент расположен фактически в **i+1** строке, **j+1** столбце, т.к. индексы в массиве начинаются с нулевого. Массив рассматривается как одномерный (вектор), содержащий **N** элементов, каждый из которых является массивом (в данном случае) из **M** элементов типа **int.** Имя массива **x** является указателем на нулевой элемент массива . Рассмотрим элемент данного массива **x[i][j].** В выражении ***x[i]**, которое эквивалентно ***(x+i), x –** указатель на нулевой элемент массива. Для доступа к **i**-тому элементу **i** умножается на длину объекта, на который указывает указатель, а именно на длину массива (строки) из **M** элементов типа **int,** т.е. **(i*M*sizeof(int)).** В результате индексной операции получается указатель, который адресует нулевой элемент **i**-го одномерного массива (строки) **x[i][o].** Для второго индекса снова применяется тот же алгоритм.. Указатель смещается на величину **(j* sizeof(int)),** после чего происходит разыменование. На этот раз результат будет иметь тип **int.** Элемент массива можно представить также в виде ***(*(x+i)+j).**

Таким образом, общее смещение в байтах указателя на нулевой элемент массива **X[N][M]** при доступе x[i][j] вычисляется по формуле**((i*M+j)* sizeof(int)).**

Для трёхмерного массива **float x3D[N][M][K]** величина смещения в байтах для доступа к элементу **x3D[i][j][k]** вычисляется по формуле

i*(M+K)+j*R+k)* sizeof (float).

Для доступа к элементу массива **4x3D[i][j][k]** можно также использовать выражение ***(*(*(x3D+i)+j)+k).**

Пример

```
/*   ЗАНЯТИЕ N 13
  Разработал Петров Ю.В.
Объявить массивы различной размерности, выполнить их инициализацию
с применением указателей и массива указателей. Получить доступ к
элементам массивов с использованием различных синтаксических
конструкций                */
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
enum en{K=2,N=3,M=4};//Аналгично #define K 5 #define N 5 #define M 4
typedef int mas1_int[N*M];//Объявление типа 1-мерный массив (вектор)
typedef int mas2_int[N][M];//Объявление типа 2-мерный массив
                //(массив 1-мерных массивов) элементов типа int
typedef int mas3_int[K][N][M]; //Объявление типа 3-мерный массив
                //(вектор 2-мерных массивов) элементов типа int
int * matrix(int n_str, int m_stolb);//Функция инициализации массива
  //с применением указателей и выделением памяти для массива в "куче"
void main()
   { int d,a,b,i,j,k;
     int *pi, *para, *c[N];
     mas1_int mas1;    mas2_int mas2;       mas3_int mas3;
     clrscr();         randomize();
     for(i=0;i<N;i++)  c[i]=&mas2[i][0];
     printf("\nИнициализация одномерного массива:\n");
     for(i=0;i<N*M;i++)
        { mas1[i]=random(10);
          printf("mas1[%d]=%2d ",i,mas1[i]);
          if ((i+1)%5==0) printf("\n");
        }
     printf("\nВведите индекс элемента одномерного массива i<N: ");
     scanf("%d",&i);
     pi=mas1;
     printf("mas1[%d]=%2d смещение: %2d байт\n",i,*(pi+i),i*sizeof(int));
     getch();
```

```
randomize();
printf("Инициализация двумерного массива\n");
printf("с использованием массива указателей:\n");
for(i=0;i<N;i++)
 for(j=0;j<M;j++)
    { *(c[i]+j)=random(10);
      printf("c[%d][%d]=%2d ",i,j,*(c[i]+j));
      if ((j+1)%M==0) printf("\n");
    }
printf("\nВведите индексы элемента двумерного массива i<N, j<M: ");
scanf("%d %d",&i,&j);
printf("mas2[%d][%d]=%2d смещение: %2d байт \n",i,j,mas2[i][j],\
(i*M+j)*sizeof(int));
printf("Другие формы записи доступа к элементам двумерного массива:\n");
printf("*(*mas2+i*M+j)= %2d\n", *(*mas2+i*M+j));
printf("(*mas2)[i*M+j]  = %2d\n", (*mas2)[i*M+j]);
printf("*(*(mas2+i)+j)= %2d\n", *(*(mas2+i)+j));
printf("*(c[i]+j)=      %2d\n", *(c[i]+j));
getch(); randomize();
printf("\nИнициализация двумерного массива в функции\n");
printf("с выделением памяти для массива в \"куче\":\n");
para=matrix(N,M);
for(i=0;i<N;i++)
 for(j=0;j<M;j++)
    { printf("mas2[%d][%d]=%2d ",i,j,*(para+i*M+j));
      if ((j+1)%M==0) printf("\n");
    }
printf("\nВведите индексы элемента двумерного массива i<N, j<M: ");
scanf("%d %d",&i,&j);
printf("mas2[%d][%d]=%2d смещение: %2d байт ",i,j,*(para+i*M+j),\
(i*M+j)*sizeof(int));
free(para); //Освобождение памяти, выделенной для массива в "куче"
getch(); randomize();
printf("\nИнициализация трехмерного массива:\n");
for(i=0;i<K;i++)
 for(j=0;j<N;j++)
    { for(k=0;k<M;k++)
       { mas3[i][j][k]=random(10);
         printf("mas3[%d][%d][%d]=%2d ",i,j,k,mas3[i][j][k]);
         if ((k+1)%4==0) printf("\n");
       }
      if ((j+1)%N==0) printf("\n");
    }
printf("\nВведите индексы элемента трехмерного массива i<K, j<N, k<M: ");
scanf("%d %d %d", &i, &j, &k);
printf("mas3[%d][%d][%d]=%2d смещение: %2d байт\n",i,j,k,\
```

```
      mas3[i][j][k],(i*M*N+j*M+k)*sizeof(int));
      printf("Другие формы записи доступа к элементам трехмерного массива:\n");
      printf("*(**mas3+i*M*N+j*M+k)= %2d\n",*(**mas3+i*M*N+j*M+k));
      printf("*(*(*(mas3+i)+j)+k)= %2d\n",*(*(*(mas3+i)+j)+k));
      getch();
 }//main

int * matrix(int n, int m)
 { int i,j;
   randomize();  //Выделение памяти для массива в "куче"
   int *pa=(int *)malloc(n*m*sizeof(int));
   for(i=0;i<n;i++)
    for(j=0;j<m;j++)   //*(pa+i*m+j)-аналогично pa[i*m+j]
     { *(pa+i*m+j)=random(51)-25;
    // printf("mas[%d][%d]=%2d ",i,j,pa[i*m+j]);
    // if ((j+1)%m==0) printf("\n");
     }
    return pa;
  }
/*   Инициализация одномерного массива:
mas1[0]= 6 mas1[1]= 0 mas1[2]= 0 mas1[3]= 8 mas1[4]= 4
mas1[5]= 0 mas1[6]= 7 mas1[7]= 4 mas1[8]= 0 mas1[9]= 7
mas1[10]= 6 mas1[11]= 6
Введите индекс элемента одномерного массива i<N: 10
mas1[10]= 6 смещение: 20 байт

Инициализация двумерного массива
с использованием массива указателей:
c[0][0]= 7 c[0][1]= 0 c[0][2]= 0 c[0][3]= 1
c[1][0]= 3 c[1][1]= 8 c[1][2]= 6 c[1][3]= 3
c[2][0]= 6 c[2][1]= 9 c[2][2]= 6 c[2][3]= 1

Введите индексы элемента двумерного массива i<N, j<M: 1 3
mas2[1][3]= 3 смещение: 14 байт
Другие формы записи доступа к элементам двумерного массива:
*(*mas2+i*M+j) =  3
 (*mas2)[i*M+j]=  3
*(*(mas2+i)+j) =  3
*(c[i]+j)      =  3

Инициализация двумерного массива в функции
с выделением памяти для массива в "куче":
mas2[0][0]= 7 mas2[0][1]=-1 mas2[0][2]=19 mas2[0][3]=15
mas2[1][0]=-6 mas2[1][1]=-15 mas2[1][2]= 2 mas2[1][3]=11
mas2[2][0]=-24 mas2[2][1]=24 mas2[2][2]=21 mas2[2][3]=-6
```

```
Введите индексы элемента двумерного массива i<N, j<M: 2 0
mas2[2][0]=-24 смещение: 16 байт

Инициализация трехмерного массива:
mas3[0][0][0]= 1 mas3[0][0][1]= 9 mas3[0][0][2]= 3 mas3[0][0][3]= 1
mas3[0][1][0]= 8 mas3[0][1][1]= 7 mas3[0][1][2]= 3 mas3[0][1][3]= 8
mas3[0][2][0]= 0 mas3[0][2][1]= 3 mas3[0][2][2]= 9 mas3[0][2][3]= 0

mas3[1][0][0]= 8 mas3[1][0][1]= 6 mas3[1][0][2]= 8 mas3[1][0][3]= 5
mas3[1][1][0]= 2 mas3[1][1][1]= 6 mas3[1][1][2]= 5 mas3[1][1][3]= 5
mas3[1][2][0]= 4 mas3[1][2][1]= 9 mas3[1][2][2]= 5 mas3[1][2][3]= 4

Введите индексы элемента трехмерного массива i<K, j<N, k<M: 1 2 2
mas3[1][2][2]= 5 смещение: 44 байт
Другие формы записи доступа к элементам трехмерного массива:
*(**mas3+i*M*N+j*M+k)=  5
*(*(*(mas3+i)+j)+k)=    5             */
```

Ход работы

1. Изучить теоретические сведения.
2. В соответствии с индивидуальным заданием разработать алгоритм и программу с применением указателей на массив и массив указателей для работы с двумерным и трёхмерным массивом.
3. Показать использование различных видов синтаксических конструкций, включая индексные выражения и указатели на тип элементов массива для доступа к элементам массива.
4. Набрать и отладить программу на компьютере.
5. Изучить работу операторов.
6. Получить результаты.
7. Оформить отчет.
8. Подготовиться к защите лабораторной работы, изучив контрольные вопросы по данной теме.

Требования к содержанию отчёта приведены в лабораторной работе №1.

Индивидуальное задание к лабораторной работе №13

Составить программу для обработки многомерных массивов с использованием циклов. Индивидуальные задания приведены в таблице 13.1.

Таблица 13.1 - индивидуальное задание

Вар.	**Условие задачи**
1	Вычислить произведение отрицательных чисел строки, у которой второй элемент четный
2	Найти сумму нечетных чисел столбца, у которого первый элемент больше второго
3	Определить количество отрицательных чисел столбца, у которого первый

	элемент меньше последнего
4	Найти сумму положительных кратных 5 чисел столбца, у которого четвертый элемент отрицательный
5	Найти произведение нечетных чисел столбца, у которого первый элемент нуль
6	Найти произведение положительных чисел столбца, последний элемент которого нуль
7	Найти сумму нечетных элементов строки, первый элемент которой кратен 3
8	Найти максимальное отрицательное число строки, у которой второй элемент больше 20
9	Найти сумму положительных четных чисел строки, у которой первый элемент отрицательный
10	Найти минимальное положительное число строки, у которой пятый элемент отрицательный
11	Найти минимальное четное число столбца, у которого первый элемент больше третьего
12	Найти сумму положительных кратных 5 чисел столбца, у которого четвертый элемент отрицательный
13	Найти количество отрицательных не кратных 3 чисел строки, у которой первый элемент нуль
14	Найти количество положительных четных чисел строки, у которой пятый элемент больше 30
15	Найти произведение квадратов положительных четных чисел столбца, у которого второй элемент нуль
16	Найти среднее арифметическое отрицательных элементов строки, у которой четвертый элемент отрицательный
17	Найти разность сумм отрицательных и положительных элементов строки, у которой третий элемент кратен 3
18	Найти среднее геометрическое модулей отрицательных элементов столбца, у которого первый элемент положительный
19	Найти все кратные 7 положительные элементы столбца, у которого третий элемент нечетный
20	Найти среднее арифметическое положительных элементов строки, у которой первый элемент меньше второго
21	Найти среднее геометрическое кратных 3 элементов столбца, у которого шестой элемент не кратен 4
22	Найти частное от деления количества отрицательных элементов столбца, у которого первый элемент нуль, на их сумму
23	Найти все положительные нечетные числа строки, у которой второй элемент не кратен 3
24	Найти количество отрицательных кратных 5 элементов столбца, у которого второй элемент равен третьему
25	Найти номер максимального четного числа строки, у которой первый элемент равен последнему
26	Найти номер минимального нечетного числа строки, у которой первый эле-

	мент равен последнему
27	Найти номер максимального четного числа столбца, у которого первый элемент равен последнему
28	Найти номер минимального нечетного числа столбца, у которого первый элемент равен последнему
29	Найти сумму максимального и минимального чисел строки, у которой второй элемент нечетный
30	Найти номер минимального нечетного числа строки, у которой первый элемент четный

Контрольные вопросы для подготовки и самостоятельной работы

1 Приведите примеры объявления массивов с различной размерностью. Объясните организацию хранения элементов массива.
2 Как происходит масштабирование при последовательном разыменовании указателя (имени массива) в процессе доступа к элементам? Какие синтаксические конструкции можно использовать для доступа к элементам массива?
3 Приведите общую формулу для массива **<Тип> <Имя> [N][M][K]** при доступе к заданному элементу **<Имя> [i][j][k]** и объясните её.
4 Как осуществляется инициализация многомерных массивов?
5 Зависит ли инициализация массива от класса памяти при объявлении?
6 Как использовать средство **typedef** для объявления типа массива.
7 Какой из индексов можно не указывать при явной инициализации массивов?
8 Можно ли указывать не все элементы при инициализации? Как использовать скобки при инициализации?
9 Какой индекс не используется при расчёте величины смещения указателя в процессе доступа к элементу массива?
10 Для чего используется указанный индекс?

Лабораторная работа № 14

Разработка программ с составными типами данных
(2часа)

***Цель*:** выработать практические навыки в написании программ с использованием комбинированных типов данных.

Теоретические сведения

Структуры

С поддерживает определённый пользователем составной тип, объявляемый с ключевым словом **struct,** который определяет структуры. Эти структуры подобны записям, используемым в других языках программирования. Структура содержит данные-члены, которые являются данными базовых типов, либо являются предварительно объявленными структурами. Структуры не могут содержать структуры своего типа, но могут содержать указатели на такие структуры.

Объявление структур

В **С** ключевое слово **struct** используется не только для объявления объектов структурного типа, но и для объявления нового (структурного) типа. Объявленный тип структуры можно использовать для объявления объектов структурного типа.

Представим общий синтаксис для объявления структуры типа **(stt)** и приведём несколько примеров структур

struct sttype **{** **type1 dataMember1;** **type2 dataMember2;** //другие данные-члены **}[<имя_объекта 1>, <имя_объекта 2>];** // объявление типа структуры и необязательное // объявление объектов	**struct Point** **{** **float x;** **float y;** **} Apoint, Bpoint, *Ppoint, Mpoint[10];** объявление типа и объектов	**struct Person** **{char firstname[12];** **char lastname[15];** **int birthday;** **float weight;** **};** // объявление // типа структуры

После объявления типа **(<strucType>),** его можно использовать для объявления объектов, например

sttype Astr, Mstr[5], *pstr;

Объявлены объекты структурного типа:
Astr- структура;
Mstr[5]- массив из **5**-ти структур;
pstr- указатель на структуру данного типа.

Структура типа **Point** имеет два члена-данных типа **float.**

Структура типа **Person** - пример структуры, которая содержит данные-члены, которые являются массивами:

firstname []- массив из 12 символов, в котором хранится имя.
lastname []- массив из 15 символов, в котором хранится фамилия.
birthday типа **int**, в котором хранится год рождения.
weight типа **float**, в котором хранится вес.

Объявление структуры-переменной (объекта структурного типа) не отличается от объявления переменных с базовыми или предварительно определёнными типами.

Общий синтаксис объявления	Пример объявления объектов (переменных) структурного типа
//объявление единственной переменной // sttype **sttype structVar**; //объявление массива структур sttype **sttype stArray[Kol_elem];**	**Point Origin, StartPoint, Points[10];** **Person You, Me, Us[30], *PYou;**

В этом примере объявлены переменные- структуры **Origin**, **StartPoint** типа **Point**, массив **Points [10]**, имеющий **10** элементов - структур типа **Point**, переменные **Me** и **You** типа **Person**, массив **Us [30],** имеющий **30** элементов- структур типа **Person** и указатель на тип **Person.**

C позволяет инициализировать данные-члены структур. Инициализация осуществляется подобно инициализации массивов и следует тем же правилам. Общий синтаксис для инициализации данных-членов структуры:

sttype strucVar ={value1,value2,...};

Компилятор присваивает значение **value1** первому данному-члену структуры **strucVar, value2**- второму данному-члену структуры **strucVar** и т.д. **C** требует соблюдения следующих правил :

1 Присваиваемые значения должны быть совместимы с соответствующими им данными-членами по типу, диапазону и количеству (для массивов).
2 Можно объявлять меньшее количество присваиваемых значений, чем количество данных. Компилятор присваивает нули остальным данным- членам структуры.
3 Нельзя указывать больше инициализирующих значений, чем количество данных-членов.
4 Значения из списка инициализации последовательно присваиваются данным-членам вложенных структур и массивов.

Пример инициализации структуры - **Point fPoint={12.4,34.5};**

В этом примере объявлена переменная **fPoint** типа **Point** и инициализированы данных члены **x, y** значениями **12.4** и **34.5.**

Доступ к данным-членам осуществляется с помощью операции **(.)"точка".**

Общий синтаксис для доступа к данным-членам структуры	Примеры доступа к данным - членам структуры
StrucVar.dataMember **<Имя_структуры> . <данное_член>;**	**Point myPoint;** // объявление **MyPoint. x=10.9;** // доступ к **x** **MyPoint. y=21.89;**

myPoint -структурная переменная, доступ к её данным-членам **x, y** осуществляется с помощью выражения **myPoint**. **x** и **myPoint**. **y** соответственно.

Пример

```
/*   ЗАНЯТИЕ N 14
  Разработал Петров Ю.В.
 Объявить составные типы данных, выполнить их
 инициализацию. Массив структур инициализировать с
 использованием операторов организации цикла. Получить доступ
 к элементам структур. Вывести значения элементов массива
 структур на экран с применением функции.              */

#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <alloc.h>

#define N 20
#define M 5
typedef struct Adr        //Adr -тип структуры
     { char town[N];  char *country;
        char region;   float indeks;   int kolvo;
     } adr;              //adr -тоже тип структуры (синоним Adr)
  //input_st() -функция для инициализация структуры типа adr
adr input_st(char *town, char *coun, char reg, float ind, int kol);
void print_st(adr adr3); //Функция вывода элементов структуры

int main(void)
{  clrscr();
  int a; float f1;
  char str[125];         //Буфер для ввода строки символов
  adr adress[M];         //Массив структур
  adr adr1,adr2={"Kiev","Ukr",'r',12.3,15}; //Явная инициализация
  adr1=adr2;             //Инициализация adr1 присваиванием
  printf("Вывод элементов структуры ");
  print_st(adr1);                    //Вывод adr1
  adr1=input_st("Kr","Ukr",'t',134.5,2);    //Инициализация adr1
  printf("Вывод элементов структуры ");
  print_st(adr1);                    //Вывод adr1
```

```
  int i=0,j;

 while (a&&i<M)           //Ввод элементов массива структур
   {  printf("Ввод города (char array[N])  ");
      scanf("%s",&adress[i].town);        fflush(stdin);
      printf("Ввод страны (char *)  ");
      scanf("%s",str);  fflush(stdin);
      adress[i].country=(char*)malloc(strlen(str)+1);
      strcpy(adress[i].country,str);
      //adress[i].country=strdup(str); //Возможный вариант иниц.
      printf("Ввод кода региона (char) ");
      scanf("%c", &adress[i].region);      fflush(stdin);
      printf("Ввод цифрового кода (float) ");
      scanf("%f",&f1);                    fflush(stdin);
      adress[i].indeks=f1;
      printf("Ввод количества (int)  ");
      scanf("%i",&adress[i].kolvo);        fflush(stdin);
      printf("\n\t\t\t Продолжить ввод ? y/n : ");
      char c=getche();      printf("\n");
      if (c=='n' || c=='N') a=0;
      i++;
    } //end while-----------------------------------------
    printf("Вывод элементов структур\n ");
    j=i;
    for(i=0;i<j;i++)        print_st(adress[i]);
   getche();
   return 0;
}  //end main()-----------------------------------------

 adr input_st(char *t, char *c, char r, float f, int k)
   { adr ad;                //Объявление локальной структуры
     strcpy(ad.town,t);      //Копирование "t" в "ad.town"
     ad.country=(char*)malloc(strlen(c)+1); //Выделение памяти
     strcpy(ad.country, c);  //Копирование "c" в "ad.country"
   // ad.country=strdup(c);   //Возможный вариант инициализации
     ad.region=r;  ad.indeks=f;   ad.kolvo=k;
     return ad;              //Возврат структуры из функции
   } //end  input_st ()-----------------------------------
 void print_st(adr adr3)    //Функция вывода элементов структуры
   { printf("| %s  ", adr3.town);
     printf("| %s  ", adr3.country);
     printf("| %4c | %10f|%6i|\n",adr3.region, adr3.indeks, adr3.kolvo);
   } //end print_st()--------------------------------------

/*Вывод элементов структуры | Kiev  | Ukr  |    r |  12.300000|    15|
  Вывод элементов структуры | Kr    | Ukr  |    t | 134.500000|     2|
```

```
Ввод города (char array[N])  Don
Ввод страны (char *)         Ukr
Ввод кода региона (char)     t
Ввод цифрового кода (float)  45.67
Ввод количества (int)        8
                    Продолжить ввод ? y/n : y
Ввод города (char array[N])  Khar
Ввод страны (char *)         Ukr
Ввод кода региона (char)     g
Ввод цифрового кода (float)  67.84
Ввод количества (int)        4
                    Продолжить ввод ? y/n : n
Вывод элементов структур | Don   | Ukr  |    t |  45.669998|     8|
                         | Khar  | Ukr  |    g |  67.839996|     4| */
```

Ход работы

1 Изучить теоретические сведения.
2 В соответствии с индивидуальным заданием разработать алгоритм решения задачи. Объявить составные (комбинированные) типы, массивы структур, использовать указатели в качестве членов структуры, объявить объединение и выполнить работу с объединением. Результаты инициализации, изменения членов комбинированных типов в процессе вычисления вывести на экран.
3 Разработать программу, набрать программу на компьютере, устранить ошибки.
4 Получить результат.
5 Оформить отчет.
6 Подготовиться к защите лабораторной работы, изучив контрольные вопросы по данной теме.

Требования к содержанию отчёта приведены в лабораторной работе №1.

Индивидуальное задание к лабораторной работе №14 взять из работы №3.

Контрольные вопросы для подготовки и самостоятельной работы

1 Какие типы данных могут быть членами структур?
2 Каков синтаксис объявления структур?
3 Как объявляются переменные (объекты) структурного типа?
4 Какие виды объектов структурного типа можно объявить?
5 Как производится инициализация данных-членов структуры? Каковы особенности инициализации массивов и структур-членов структуры (вложенных структур)?
6 Можно ли производить неполную инициализацию структуры (не для всех членов структуры)?
7 Какие правила инициализации данных-членов структур Вы знаете?

8 Как располагаются данные-члены структур в памяти?
9 Что такое объединение (**union)?**
10 Как располагаются в памяти данные-члены объединения?
11 Каков синтаксис объявления типа и объектов типа объединения?
12 В чём отличие объединения от структуры?
13 Как осуществляется доступ к данным-членам структуры, объединения?
14 Как осуществляется доступ к данным-членам структуры, расположенной в массиве структур?

Лабораторная работа № 15

Использование указателей для работы с составными типами данных (2часа)

Цель работы: выработать практические навыки в использовании указателей при работе с составными (комбинированными) типами данных.

Теоретические сведения

Можно объявить указатели на структуры и получить доступ к данным-членам этих структур, используя указатели. **С** требует, чтобы при этом использовалась операция доступа к указателю **(->)** вместо операции доступа **(.)** "точка". Общий синтаксис для объявления указателя на переменную структурного типа тот же, что и для обычной переменной.

Пример объявления указателя на переменную-структуру и использования его для доступа к данным-членам этой структуры.

Объявление типа структуры	Использование структур и указателей
с помощью **typedef struct myComplex** **{** **float my_real;** **float my_mag;** **} MyComp;** // тип структуры – // **myComp** и **MyComp** синонимы	**void main()** **{ myComp comvar={1.0, 2.8}** **MyComplex * pComplex=&comlexvar;** **cout <<"Complex number = "** **<< pComp ->my_real** **<< pComp ->my_mag;** **}**

В примере объявлена структура **myComp** или **MyComp** с членами данных **my_real** и **my_mag** типа **float**. В функции **main()** объявляется и инициализируется переменная-структура **comvar**. В функции также объявляется указатель **pComp** на тип **myComp** и инициализируется адресом переменной **compvar.** При выводе в поток **cout** используется операция **->** для доступа с помощью указателя **pComp** к данным-членам **my_real** и **my_mag** переменной-структуры **compvar**.

Пример

```
/*   ЗАНЯТИЕ N 15
   Разработал Петров Ю.В.
 Объявить структуры и указатели на них, выполнить инициализацию.
 Массив указателей на структурный тип инициализировать с
 выделением памяти в куче. Получить доступ к элементам структур
 с использованием указателей. Вывести значения элементов
 массива структур на экран с применением функции. Структура
 содержит фамилию и дату рождения (число, месяц, год) студента.
 Ввести информацию о студентах и найти данные о первом
 в массиве структур студенте, который родился заданного числа.*/
#include <stdio.h>
#include <conio.h>
```

```
#define N 3
#define M 20
 struct stud
  { char name[M];
    int  day,month,year;
  };
 stud data[N];      //Глобальный массив из N структур типа stud
 //Функция поиска данных в глобальный массиве data[i].day==j)
 int poisk(int j);
 //print_st() -Функция вывода элементов структуры из глобальнго
 void print_st(int num); //массива, num -кол. структур для вывода
 //init_loc() -функция для инициализация структуры типа stud
 stud *init_ptr(void);
 void print_ptr(stud *);  //Функция вывода элементов структуры

void main()
{ int i=0,j,a;      clrscr();
  stud arrloc[N];   //Массив из N структур
  stud *dloc[N];    //Массив указателей на структурный тип stud
  stud *pstr;       //Указатель на структурный тип stud
  pstr=arrloc;      //Аналогично pstr=&arrloc[0];
  while (a&&i<N)    //Ввод элементов массива структур arrloc[N]
   { printf("Введите фамилию студента      : ");
     scanf("%s",pstr->name);      fflush(stdin);
     printf("Введите дату рождения(чч мм гг) : ");
     scanf("%d%d%d",&pstr->day,&pstr->month,&pstr->year);
  fflush(stdin);
    // printf("%-15s  - %2d.%2d.%4d\n",pstr->name,pstr->day,
    // pstr->month,pstr->year); //Контроль значений при вводе
     printf("t\t Продолжить ввод ? y/n  ");
     char c=getche();             printf("\n");
     if (c=='n' || c=='N') a=0;
     pstr++;//Переход к следующей структуре в массиве arrloc[N]
     i++;
   } //end while-----------------------------------------
  printf("Вывод значений элементов массива структур arrloc[i]\n ");
  printf("с использованием указателя\n ");
  j=i; pstr=&arrloc[0];
  for(i=0;i<j;i++)
   { print_ptr(pstr);
     dloc[i] = new stud; //Выделение памяти для структуры в "куче"
     dloc[i] = pstr;     //Инициализация элементов локального массива
     pstr++;
   }
  printf("Ввод недостающих значений элементов массива структур\n ");
  printf("с использованием массива указателей\n ");
```

```
 for ( i=j; i<N; i++)
     dloc[i]=init_ptr(); //Выделение памяти для структуры в функции
 printf("Вывод значений элементов массива структур\n ");
 printf("с использованием массива указателей\n ");
 for ( i=0; i<N; i++)
  { print_ptr(dloc[i]);
    data[i] = *dloc[i]; //Инициализация элементов глобального
  }                     //массива структур
 printf("Вывод (N-2) первых значений элементов глобального \
 массива структур\n");
 print_st(N-2);
 printf("Введите день для поиска первого в массиве структур \
 студента   : ");
 scanf("%d",&j);    fflush(stdin);
 a=poisk(j);
 if (a!=-1) print_ptr(&data[a]);
    else    printf("Нет таких студентов ");
 for ( i=0; i<N; i++)  delete dloc[i]; //Освобождение памяти в "куче"
 getch();
}//End main()-----------------------------------------------------

int poisk(int j)   //Функция поиска данных (data[i].day==j)
 {int a=-1;        //в глобальном массиве структур data[i]
  for (int i=0; i<N; i++)
   if (data[i].day==j) {a=i; return a;}
  return a;
 }//-------------------------------------------------------
//print_st() -Функция вывода элементов структуры из глобальнго
void print_st(int num)            //массива, num -количество
 { for (int i=0; i<num; i++)      //структур для вывода
   printf("%-15s  - %2d.%2d.%4d\n",data[i].name,data[i].day,
   data[i].month,data[i].year);
 }//-------------------------------------------------------
//init_loc() -функция для инициализация структуры типа stud
stud *init_ptr(void)
 { stud *pst = new stud;      //Выделение памяти в "куче"
   printf("Введите фамилию студента        : ");
   scanf("%s",&pst->name);
   printf("Введите дату рождения(чч мм гг) : ");
   scanf("%d%d%d",&pst->day,&pst->month,&pst->year);
   return pst;
 }//-------------------------------------------------------
void print_ptr(stud *pdt)  //Функция вывода элементов структуры
 { printf("%-15s  - %2d.%2d.%4d\n",pdt->name,pdt->day,
   pdt->month,pdt->year);
 }//-------------------------------------------------------
```

```
/* Введите фамилию студента        : Petrov
   Введите дату рождения(чч мм гг) : 12 7 1980
   Продолжить ввод ? y/n  y
   Введите фамилию студента        : Sidorov
   Введите дату рождения(чч мм гг) : 24 8 1983
   Продолжить ввод ? y/n  n
Вывод значений элементов массива структур arrloc[i]
с использованием указателя
Petrov          - 12. 7.1980
Sidorov         - 24. 8.1983
Ввод недостающих значений элементов массива структур
с использованием массива указателей
   Введите фамилию студента        : Ivanov
   Введите дату рождения(чч мм гг) : 24 5 1982
Вывод значений элементов массива структур
с использованием массива указателей
Petrov          - 12. 7.1980
Sidorov         - 24. 8.1983
Ivanov          - 24. 5.1982
Вывод (N-2) первых значений элементов глобального массива структур
Petrov          - 12. 7.1980
Sidorov         - 24. 8.1983
Введите день для поиска первого в массиве структур студента : 24
Sidorov         - 24. 8.1983          */
```

Ход работы

1. Изучить теоретические сведения.
2. В соответствии с индивидуальным заданием разработать алгоритм решения задачи. Объявить комбинированные типы, массивы структур, указатели на структуру. Использовать указатели в качестве членов структуры, а также для доступа к членам структуры и работы с ними. Объявить объединение и выполнить работу с объединением с использованием указателей. Результаты инициализации, изменения членов комбинированных типов вывести на экран.
3. Разработать и набрать программу на компьютере, устранить ошибки.
4. Получить результат.
5. Оформить отчет.
6. Подготовиться к защите лабораторной работы, изучив контрольные вопросы по данной теме.

Требования к содержанию отчёта приведены в лабораторной работе №1.

Индивидуальное задание к лабораторной работе №15 взять из работы №3.

Контрольные вопросы для подготовки и самостоятельной работы

1 Можно ли использовать указатели в качестве данных-членов структур?
2 Можно ли использовать массивы и структуры в качестве данных-членов структур (вложенные объявления)?
3 Можно ли использовать в качестве вложенных структуры объявляемого (своего) типа, а также указатели на структуры своего типа?
4 Как объявить указатель на структуру, массив указателей на структуры? Существуют ли различные варианты объявления?
5 Можно ли использовать **typedef** для объявления типа структуры?
6 Как получить доступ к элементу структуры с помощью указателя?
7 Как осуществляется доступ к данным-членам структуры при использовании массива указателей на структуры?
8 Можно ли использовать различные классы памяти применительно к переменным структурного типа?
9 Обязательно ли указывать ключевое слово **struct** при объявлении структурного типа, переменных структурного типа?
10 Можно ли хранить адрес структуры в самой структуре?

Лабораторная работа № 16

Использование указателей для работы с функциями (2 часа)

Цель работы: выработать практические навыки в написании программ с функциями и в использование указателей для работы с функциями.

Теоретические сведения

С позволяет использовать указатели на базовые типы, на массивы и структуры как параметры передаваемые в функцию и возвращаемые из функции, а также указатели на функции. Общий синтаксис для объявления указателя на структуру как параметры функции:

[const]<тип>*<имя указателя>

При объявлении функции имя указателя как формального параметра может не указываться аналогично другим видам формальных параметров.

Ключевое слово **const** предохраняет данные от изменения данных в структуре функции, тогда как отсутствие ключевого слова **const** позволяет функции изменять значения данных, т.к. при использовании указателей в теле функции известны физические адреса переменных.

Пример использования указателей на различные типы данных как параметры функции.

```
int funct (float*, int*); // передача указателей на тип float и на тип int
struct Myst
{
//члены;
};
void loadMy(Myst*pmy); // передача указателя pmy на тип структур типа Myst.
```

В примере объявлена структура типа **Myst** и объявлена функция **loadMy()** . Функция имеет формальный параметр-указатель **pmy** на тип **Myst.**

Вызов функции **funct()** и **fn():**

```
Float*pf; int*pl;
Int a = funct(pf, pl); fn(pf);
```

Объявление указателя на функцию аналогично объявлению функций, с которыми он будет использоваться. Например

```
void (* pfunc) (int, int);
```

Объявлен указатель **pfunc** на функцию, которая требует два параметра типа **int** и не возвращает значения **(void).** Типы и количество параметров, а также тип возвращаемого значения должны создать. Для использования указателя объявим функцию

```
void f1 (int, int);
```

Присвоим указателю адрес функции

```
pfunc= f1;
```

и осуществим её вызов

```
(*pfunc)(2,4);
```

Указатели как параметры функции

Указатели широко используются при передаче аргументов в функцию и из функции, создавая двухсторонний поток данных между вызываемой и вызывающей функциями.

Для передачи аргументов в функцию используют:

1 Указатели на простые типы данных, как параметры функции.
2 Указатели на массивы (передавать по значению нельзя).
3 Указатели на перечислимые типы как параметры функции.
4 Указатели на структуры и объединения как параметры функции.
5 Указатели на функции (передавать по значению нельзя).

C позволяет объявлять и использовать указатели на функции. В таком указателе хранится адрес функции, т.е. адрес первого выполнимого оператора. Указатель на функцию можно использовать для её вызова. Общий синтаксис для объявления указателя на функцию:

<тип_возвращаемого_значения> (*<имя_указателя_на_функцию>) (<объявления_типов_параметров>);

При инициализации указателя на функцию нужно использовать имя функции, у которой тип возвращаемого значения и список типов параметров соответствуют объявлению указателя. Имя функции имеет тип указателя на функцию с заданными для неё в объявлении типами параметров и возвращаемого значения.

Инициализация указателя на функцию:

<имя_указателя> = <имя_функции>;

Общий синтаксис для вызова функции с использованием указателя:

(*<имя_указателя>)(<список_фактических_параметров>);

Пример

int linescarch (int)*pstring //объявление функции **linesearch (int*)**
voild main()
{int a; * pstr = "Hello World!"; // объявление переменной и указателя **pstr** на тип **int**
int (*search)(int*); // объявление указателя на функцию
search= linesearch; // присваиваем адрес функции **linesearch ()**указателю **search**
//другие операторы
a = (* search)(pstr); // вызов функции **linesearch ( )** с помощью указателя.
}

Пример

```
/*   ЗАНЯТИЕ N 16
  Разработал Петров Ю.В.
 Объявить заданные функции и указатель на функцию этого типа,
 выполнить определение функций. Объявить массивы и указатели
 на них, выполнить инициализацию в соответствии с заданными
```

```
 зависимостями и использованием указателей. Для одного из
 массивов выделить память в куче. Вывести значения элементов
 массивов на экран. Осуществить вызов функций с применением
 указателя. Вывести результаты работы функций на экран.   */
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <iostream.h>
 const int kx = 6, ky = 6, kw = 8, a1 = 3, a2 = 9,
   b1 = 6, b2 = 5, c1 = 6, c2 = 10;

//Вычисляет сумму отрицательных элементов массива
float summa ( int, float* );
//Вычисляет произведение положительных элементов массива
float prois ( int, float* );
float (*ps)( int, float* ); //Объявление указателя на функцию

void main ()
{
 int i;
 float y[ky+2]; //+2 элемента массива для хранения
// результатов работы функций
 float *px=new float [kx+2];//Выделение памяти для массива в "куче"
 float *xptr;
 clrscr();
 printf("Массив x[]\n");
 for ( i=0; i<kx; i++)
   { *px = a1*i*i - a2*(5-i); //Инициализация динамического массива
     printf("x[%d] = %6.2f %p \n",i, *px, px);
     px++;
   }
 xptr = y;
 gotoxy(25,1);  printf("Массив y[]\n");
 for ( i=0; i<ky; i++)
  { *xptr = b1*sin(2*i) + b2*exp(i-5); //Инициализация массива y[]
    gotoxy(30,i+2);
    printf("y[%d] = %6.2f\n",i,*xptr);
    xptr++;
  }
 px-=kx;    //Установка указателя на нулевой элемент массива
 printf("Указатель на дин. массив содержит адрес px =  %p\n ", px );
 ps=summa;  //Связывание указателя с функцией summa()
 printf("Указатель на функцию содержит адрес ps =  %p\n ", ps );
 printf("Сумма: x[%d] = %6.2f   ", kx, (*ps)(kx, px) );
 printf("Сумма: y[%d] = %6.2f\n ", ky, (*ps)(ky, y) );
 ps=prois;  //Связывание указателя с функцией prois()
```

```
 printf("Указатель на функцию содержит адрес ps =  %p\n ", ps );
 printf("Произведение: x[%d] = %8.2f ", kx+1, (*ps)(kx, px) );
 printf("Произведение: y[%d] = %8.2f\n ", ky+1, (*ps)(ky, y) );
 delete [] px; //Освобождение памяти, выделенной в "куче"для массива
 getch();
}

float summa ( int kol, float* Arr )  //Определение функции
 { float s=0;
   for ( int i=0; i<kol; i++ ) if (Arr[i]<0) s+=Arr[i];
   return s;
 }

 float prois ( int kol, float* Arr )  //Определение функции
 { float s=1;
   for ( int i=0; i<kol; i++ ) if (Arr[i]>0) s*=Arr[i];
   return s;
 }
   /*
      Массив x[]              Массив y[]
   x[0] = -45.00 0AD0          y[0] =   0.03
   x[1] = -33.00 0AD4          y[1] =   5.55
   x[2] = -15.00 0AD8          y[2] =  -4.29
   x[3] =   9.00 0ADC          y[3] =  -1.00
   x[4] =  39.00 0AE0          y[4] =   7.78
   x[5] =  75.00 0AE4          y[5] =   1.74
 Указатель на дин. массив содержит адрес px =  0AD0
 Указатель на функцию содержит адрес     ps =  049C
 Сумма: x[6] = -93.00    Сумма: y[6] =  -5.29
 Указатель на функцию содержит адрес     ps =  04F3
 Произведение: x[7] = 26325.00 Произведение: y[7] = 2.52  */
```

Ход работы

1. Изучить теоретические сведения.
2. В соответствии с индивидуальным заданием по лабораторной работе **№7**. разработать алгоритм.
3. Объявить указатели на функции. Использовать указатели для вызова соответствующих функций. Использовать оператор **switch** для выбора варианта функций.
4. Разработать и набрать программу на компьютере, устранить ошибки.
5. Получить результат.
6. Оформить отчет.
7. Подготовиться к защите лабораторной работы, изучив контрольные вопросы.

Индивидуальное задание к лабораторной работе №16

Составить программу использующую вызовы функций с помощью указателей. Индивидуальные задания приведены в таблице 16.1.

Таблица 16.1 - индивидуальные задания

Вариант	первая функция	вторая функция	третья функция	четвертая функция
1	int (*func1) (int *)	int (*func2) (int *,float *)	int (*func3) (void *)	char *func4 (char *,…)
2	float (*func1) (int *, float *, double *)	float (*func2) (void *,…)	float (*func3) (int, …)	double (*func4 (double, double))[3]
3	void (*func1) (void *, int *)	void (*func2) (int, int *)	void (*func3) (void)	int (*func4 (void))(int)
4	double (*func1) (float *, int)	double (*func2) (double, long int)	double (*func3) (unsigned int *)	int (*func4(int)) (int)[3][5]
5	long double (*func1) (int *, float *)	long double (*func2) (void *, …)	long double (*func3) (int, …)	char * (*func4 (int *, …))(void *)
6	char (*func1) (unsigned int *)	char (*func2) (int *, float *, double *)	char (*func3) (void *,…)	int *func4(char *, int *,…)
7	long int (*func1) (int, int *)	long int (*func2)(double , long int)	long int (*func3)(void)	float (*func4 (double)(void)) [10]
8	unsigned int (*func1) (int *)	unsigned int (*func2) (int, …)	unsigned int (*func3) (int *,float *)	long int (*func4(int)) (void)
9	float (*func1) (int *, float *, double *)	float (*func2) (unsigned int *)	float (*func3) (void)	void *func4 (int *, int, …)
10	unsigned long int (*func1) (int, …)	unsigned long int (*func2) (int *)	unsigned long int (*func3) (double, long int)	float (*func4(long int, int))[5]
11	signed int (*func1) (int *, float *, double *)	signed int (*func2) (int)	signed int (*func3) (void *, …)	double (*func4 (char)(int *))[4]
12	double (*func1) (float *, int)	double (*func2) (double, long int)	double (*func3) (unsigned int *)	double (*func4 (double, double))[3]
13	long double (*func1) (int *, float *)	long double (*func2) (void *, …)	long double (*func3) (int, …)	int (*func4 (void))(int)
14	char (*func1)	char (*func2)	char (*func3)	int (*func4(int))

	(unsigned int *)	(int *, float *, double *)	(void *,…)	(int)[3][5]
15	long int (*func1) (int, int *)	long int (*func2)(double , long int)	long int (*func3)(void)	char * (*func4 (int *, …))(void *)
16	unsigned int (*func1) (int *)	unsigned int (*func2) (int, …)	unsigned int (*func3) (int *,float *)	int *func4(char *, int *,…)
17	int (*func1) (int *)	int (*func2) (int *,float *)	int (*func3) (void *)	float (*func4 (double)(void)) [10]
18	float (*func1) (int *, float *, double *)	float (*func2) (void *,…)	float (*func3) (int, …)	long int (*func4(int)) (void)
19	void (*func1) (void *, int *)	void (*func2) (int, int *)	void (*func3) (void)	int *func4(char *, int *,…)
20	double (*func1) (float *, int)	double (*func2) (double, long int)	double (*func3) (unsigned int *)	float (*func4 (double)(void)) [10]
21	long double (*func1) (int *, float *)	long double (*func2) (void *, …)	long double (*func3) (int, …)	long int (*func4(int)) (void)
22	char (*func1) (unsigned int *)	char (*func2) (int *, float *, double *)	char (*func3) (void *,…)	void *func4 (int *, int, …)
23	unsigned long int (*func1) (int, …)	unsigned long int (*func2) (int *)	unsigned long int (*func3) (double, long int)	float (*func4(long int, int))[5]
24	signed int (*func1) (int *, float *, double *)	signed int (*func2) (int)	signed int (*func3) (void *, …)	double (*func4 (char)(int *))[4]
25	double (*func1) (float *, int)	double (*func2) (double, long int)	double (*func3) (unsigned int *)	char *func4 (char *,…)
26	long double (*func1) (int *, float *)	long double (*func2) (void *, …)	long double (*func3) (int, …)	double (*func4 (double, double))[3]
27	char (*func1) (unsigned int *)	char (*func2) (int *, float *, double *)	char (*func3) (void *,…)	int (*func4 (void))(int)
28	double (*func1) (int *, float *, double *)	double (*func2) (double, long int)	double (*func3) (unsigned int *)	int (*func4(int)) (int)[3][5]
29	void (*func1)	void (*func2)	void (*func3)	char * (*func4

	(void *, int *)	(int, int *)	(void)	(int *, …))(void *)
30	double (*func1) (float *, int)	double (*func2) (double, long int)	double (*func3) (unsigned int *)	int *func4(char *, int *,…)

Требования к содержанию отчёта приведены в лабораторной работе №1.

Контрольные вопросы для подготовки и самостоятельной работы

1 Можно ли использовать указатели для передачи данных в функции?
2 Какие типы данных можно передать в функцию с использованием указателей?
3 Можно ли менять значения данных в функции при использовании указателей при наличии модификатора **const**.
4 Можно ли возвращать указатели из функций?
5 Как объявить указатель на функцию?
6 Можно ли использовать указатели для работы с функциями различного типа?
7 Какой тип имеет имя функции?
8 Как связать указатель с конкретной функцией?
9 Как использовать указатели для вызова функции?
10 Как ограничивается доступ к членам класса?
11 Как связать и использовать указатель на функцию?
12 Как вызвать функцию с использованием указателя?

Лабораторная работа №17

Использование функций высокого и низкого уровня для работы с потоками (файлами)
(2 часа)

Цель работы: научиться использовать функции высокого и низкого уровня при работе с файлами.

Теоретические сведения

Функции для работы с файлами

chmod() **#include <io.h>**
Изменяет режим доступа к файлу.
int chmod(const char *path, int mode);

chsize() **#include<io.h>**
Изменяет размер файла.
int chsize(int handle, long size);

close() **#include<io.h>**
Закрывает файл.
int close(int handle);

creatnew() **#include <dos.h>**
Создает новый файл.
int creatnew(const char *path, int mode);

eof() **#include<io.h>**
Определяет, достигнут ли конец файла.
int eof(int handle);

fclose() **#include <stdio.h>**
Закрывает поток.
int fclose (FILE *stream);

fdopen() **#include<stdio.h>**
Связывает поток с логическим номером файла.
FILE *fdopen(int handle, char *type);

feof() **#include<stdio.h>**

ferror() **#include<stdio.h>**
Обнаруживает ошибки в потоке.
int ferror(FILE *stream);

fgetc() **#include<stdio.h>**
Получает символ из потока.
int fgetc(FILE *stream);

fgetchar() **#include<stdio.h>**
Получает символ из потока **stdin**.
int fgetchar(void);

fgetpos() **#include<stdio.h>**
Возвращает положение указателя текущей позиции в файле.
int fgetpos(FILE *stream, fpos_t *pos);

fgets() **#include<stdio.h>**
Получает строку символов из потока.
char *fgets(char s, int n, FILE *stream);

fopen() **#include <stdio.h>**
Открывает поток.
FILE *fopen(char *filename, char *type);

fprintf() **#include <stdio.h>**
Осуществляет форматированный вывод в поток.
int fprintf(FILE *stream, const char *format [argument,...]);

fputc() **#include <stdio.h>**
Выводит символ в поток.
int fputc(int c, FILE *stream);

fputchar() **#include <stdio.h>**
Выводит символ в поток **stdout**.
int fputchar(int c);

fputs() **#include <stdio.h>**
Выводит строку символов в поток.
int fputs(char *string, FILE *stream);

fread() **#include <stdio.h>**
Считывает данные из потока.
size_t fread(void *ptr, size_t size, size_t n, FILE *stream);

freopen() **#include <stdio.h>**
Связывает с потоком новый файл.
FILE *freopen(char *filename, char *mode, FILE *stream);

fscanf() **#include <stdio.h>**
Выполняет форматированный ввод из потока.
int fscanf(FILE *stream, char *format [adress,...]);

fseek() **#include <stdio.h>**
Устанавливает указатель файла в потоке.
int fseek(FILE *stream, long offset, int fromwhere);

fsetpos() **#include<stdio.h>**
Позиционирует указатель текущей позиции в файле, связанном с потоком **stream.**
int fsetpos(FILE *stream, const fpos_t *pos);

ftell() **#include <stdio.h>**
Возвращает положение указателя текущей позиции файла.
long ftell(FILE *stream);

fwrite() **<stdio.h>#include**
Записывает данные в поток.
size_t fwrite(void *ptr, size_t size, size_t n, FILE *stream);

getc() **#include <stdio.h>**
Вводит из потока символ.
int getc(FILE *stream);

getchar() **#include <stdio.h>**
Вводит символ из потока **stdin**.
int getchar(void);

gets() **#include<stdio.h>**
Вводит строку символов из потока **stdin**.
char *gets(char *s);

getw() **#include <stdio.h>**
Вводит из потока целое число.
int getw(FILE *stream);
Перемещает указатель чтения/записи файла.

lseek() **#include <io.h>**
long lseek(int handle, long offset, int fromwhere);

_open() **#include <fcntl.h>**
Открывает файл для чтения или записи.
int _open(const char *filename, int oflags);

open() **#include <fcntl.h>; #include <sys\stat.h>**;
Открывает файл для чтения
int open(const char *filename, int access [unsigned mode]);
или записи.

putc() **#include <stdio.h>**
Выводит символ в поток.
int putc(int c, FILE *stream);

putchar() **#include <stdio.h>**
Выводит символ в поток **stdout**.
int putchar(int c);

puts() **#include<stdio.h>**
Выводит строку в поток **stdout**.
int puts(const char *s);

putw() **#include <stdio.h>**
Помещает в поток целое значение.
int putw(int w, FILE *stream);

_read() **#include<io.h>**
Считывает данные из файла.
int _read(int handle, void *buf, unsigned len);

read() **#include<io.h>**
Считывает данные из файла.
int read(int handle, void *buf, unsigned len);

rewind() **#include <stdio.h>**
Устанавливает указатель в начало потока.
int rewind(FILE *stream);

setmode() **#include<fcntl.h>**
Устанавливает режим открытия файла.
int setmode(int handle, unsigned amode);

unlink() **#include<io.h>**
Удаляет файл.
int unlink(const char *filename);

vprintf() **#include <stdarg.h>**
Осуществляет форматированный вывод в стандартный поток **stdout**.

int vprintf (const char *format, va_list arglist);

_write() **#include<io.h>**

Записывает данные в файл.

int _write(int handle, void *buf, unsigned len);

write() **#include<io.h>**

Записывает данные в файл.

int write(int handle, void *buf, unsigned len);

Пример

```
/*    ЗАНЯТИЕ N17
  Разработал Петров Ю.В.
 Объявить структуру, содержащую члены типа int, float, char,
 выполнить инициализацию нескольких структур в цикле и запись
 их в файл. Объявить и инициализировать массивы данных для
 членов структур,записать их в отдельные файлы. Ввести значения
 элементов структур, прочитав их из созданных файлов данных и
 дописать в конец файла структур. Прочитать структуры из
 полученного файла и вывести значения членов структур на экран */

#include <stdio.h>
#include <CONIO.H>
#include <CTYPE.H>
#define N 5

int main(void)
{  struct str{ int   ces;
               float ves;
               char  tip;
             } st;
   FILE *fstr;         //Объявление указателя на файл (для структур)
   FILE *fces,*fves,*ftip;//Объявление указателей на файлы
   int a, mces[ ] ={1,   24,   14,   5,  16};
   float  mves[ ] ={2.3, 3.4, 5.3, 5.6, 2.4};
   char   mtip[ ] ="abcde";
   clrscr();
  //   Открытие файлов для записи и чтения ("w+"):
  // FILE *fopen(const char *filename, const char *mode);
   if ((fstr = fopen("fstr.dat", "w+")) == NULL)
    { fprintf(stderr, "Cannot open output file\n");
      return 1;
    }
   if ((fces = fopen("fc.dat", "w")) == NULL) //Только для записи
    { fprintf(stderr, "Cannot open output file\n");
      return 1;
    }
```

```
  if ((fves = fopen("fv.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open output file\n");
     return 1;
   }
  if ((ftip = fopen("ft.dat", "w+")) == NULL)
   { fprintf(stderr, "Cannot open output file\n");
     return 1;
   }
  int i=1;
  do
   {    printf("  Ввод данных в файл структур: %i\n",i);
    met1: printf("Ввод даты (>0, тип - int):");scanf("%i",&st.ces);
          if (st.ces<=0) { fflush(stdin); goto met1;}
          fflush(stdin);
    met2: printf("Ввод веса (>0, тип - float):");scanf("%f",&st.ves);
          if (st.ves<=0) { fflush(stdin); goto met2;}
          fflush(stdin);
    met3: printf("Ввод вида (тип - char):");scanf("%c",&st.tip);
          if (!isalpha(st.tip)) { fflush(stdin); goto met3;}
  // Запись полученной структуры в файл fstr
          fwrite(&st, sizeof(st), 1, fstr);
          i++;
          printf("Выход a==0 Введите a="); scanf("%i", &a);
   } while(a); //Конец записи структур в файл fstr
// Запись в файлы данных из массивов: mces[i], mves[i], mtip[i]
  for(i=0;i<N;i++)
   {//Формат и применение функций записи в файл:
//int fprintf (FILE *stream, const char *format [, argument, ...]);
     fprintf(fces,"%d ", mces[i]); //Из массива mces[i]
//size_t fwrite(const void *ptr, size_t size, size_t n, FILE*stream);
     fwrite (&mves[i], sizeof(float), 1, fves);  //Из массива mves[i]
//int fputc(int c, FILE *stream);
     fputc  ( mtip[i],ftip);       //Из массива mtip[i]
    }

  fclose(fces);       //Закрытие файла fces
// int fseek(FILE *stream, long offset, int whence);
  fseek(fves, 0L,SEEK_SET);//Установка указателя файла fves на начало
  fseek(ftip, 0L,SEEK_SET);//Установка указателя файла ftip на начало
  //   Открытие файла fces для чтения ("r"):
  if ((fces = fopen("fc.dat", "r")) == NULL)
   { fprintf(stderr, "Cannot open input file\n");
     return 1;
   }
  printf("Ввод данных в структуры из файлов данных для отдельных членов:\n");
  for(i=0;i<N;i++)
```

```
    {//Формат и применение различных функций чтения данных
     //из файлов и инициализация членов структуры st :
// int fscanf (FILE *stream, const char *format [, address, ...]);
      fscanf (fces, "%d", &st.ces);
// size_t fread(void *ptr, size_t size, size_t n, FILE *stream);
      fread (&st.ves, sizeof(float), 1, fves);
// int fgetc(FILE *stream);
      st.tip=fgetc (ftip);
// Результаты инициализации членов структуры st
      printf("Дата=%d Вес=%f Вид=%c \n", st.ces, st.ves, st.tip);
// Запись полученной структуры в конец существующего файла fstr
      fwrite (&st, sizeof(st), 1, fstr);
    }
  fseek(fstr, 0L, SEEK_SET);
  printf("\nВывод данных из файла структур: Дата, Вес, Вид \n");
  while(1)
   { a=fread(&st, sizeof(st), 1, fstr);
     if (a==0) goto end;
     printf("Дата= %i Вес= %f Вид=%c\n ", st.ces, st.ves, st.tip);
   }

  end:  fclose(fstr);
  fcloseall(); //fclose(fces); fclose(fves); fclose(ftip);
  getch();
  return 0;
}
/*  Ввод данных в файл структур: 1
  Ввод даты (>0, тип - int):3
  Ввод веса (>0, тип - float):4.6
  Ввод вида (тип - char):g        Выход a==0 Введите a=1
    Ввод данных в файл структур: 2
  Ввод даты (>0, тип - int):19
  Ввод веса (>0, тип - float):2.7
  Ввод вида (тип - char):p        Выход a==0 Введите a=0
  Ввод данных в структуры из файлов данных для отдельных членов:
    Дата=1 Вес=2.300000  Вид=a
    Дата=24 Вес=3.400000 Вид=b
    Дата=14 Вес=5.300000 Вид=c
    Дата=5  Вес=5.600000 Вид=d
    Дата=16 Вес=2.400000 Вид=e
  Вывод данных из файла структур: Дата, Вес, Вид
    Дата= 3  Вес= 4.600000 Вид=g
    Дата= 19 Вес= 2.700000 Вид=p
    Дата= 1  Вес= 2.300000 Вид=a
    Дата= 24 Вес= 3.400000 Вид=b
    Дата= 14 Вес= 5.300000 Вид=c
```

Дата= 5 Вес= 5.600000 Вид=d
Дата= 16 Вес= 2.400000 Вид=e */

Ход работы

1 Изучить теоретические сведения.
2 В соответствии с индивидуальным заданием разработать алгоритм и программу с применением указанных функций высокого и низкого уровня.
3 Набрать программу на компьютере, и устранить ошибки.
4 Получить результат.
5 Оформить отчет.
6 Подготовиться к защите лабораторной работы, изучив контрольные вопросы по данной теме.

Индивидуальное задание к лабораторной работе №17

Составить программу для записи данных структурного типа в файл. Ввод информации осуществлять с использованием функций высокого уровня, вывод информации осуществлять с использованием функций низкого уровня. Индивидуальные задания приведены в таблице 17.1.

Таблица 17.1 - Индивидуальные задания

№ варта	Номер и содержание данных						
	1	2	3	4	5	6	7
1	ФИО	Рост	Вес	Год рождения	Пол	Рейтинг	…
2	Название ЭВМ	Тип процессора	Объем памяти	Тип дисплея	Количество дисководов	Стоимость	…
3	Тип автомобиля	Цвет	Количество колес	Количество мест	Грузоподъемность	Стоимость	…
4	Тип автобуса	Количество мест	Грузоподъемность	Номер маршрута	Пункт назначения	Время отправления	…
5	ФИО	Номер школы	Класс	Средний бал аттестата	Любимый предмет	Нелюбимый предмет	…
6	Название магазина	Вид товара	Адрес	Время работы	Количество продавцов	Номер магазина	…
7	ФИО	Вид спорта	Личный рекорд	Иностранный язык	Срок занятий	Количество знакомых слов	…
8	Название фирмы	Объем годового оборота	ФИО директора	Штат	Стаж работы	Возраст директора	…

№ варта	Номер и содержание данных						
	1	2	3	4	5	6	7
9	ФИО	Наличие братьев и сестер	Число	Месяц	Год рождения	Вес	…
10	Название книги	Автор	Издательство	Дата издания	Страна	Количество страниц	…
11	ФИО	Номер в группе	Название группы	Курс	Оценки	Рейтинг	…
12	Название велосипеда	Количество колес	Диаметр колес	Цвет	Грузоподъемность	Скорость	…
13	Название программного продукта	Область применения	Объем занимаемой памяти	Операционная система	Режим: текстовый или графический	Стоимость	…
14	Название рок группы	Дата создания	Стиль	Состав группы	Количество альбомов	Стоимость билета	…
15	ФИО	Номер зачетной книжки	Любимый предмет	Оценки по математике	средний бал	Язык программирования	…
16	Название журнала	Возраст читателей	Количество страниц	Начало издания	Тираж	Подписной индекс	…
17	Название самолета	Дальность полета	Количество мест	Количество двигателей	Время вылета	Время в воздухе	…
18	Название утюга	Цена	Страна производитель	Вес	Температура	Наличие регулятора	…
19	Город	Страна	Область	Почтовый индекс	Число жителей	Площадь	…
20	Операционная система	Многозадачность	Объем памяти	Версия	Фирма разработчик	Стоимость	…

Требования к содержанию отчёта приведены в лабораторной работе №1.

Контрольные вопросы для подготовки и самостоятельной работы

1 Приведите общий формат объявления функции.
2 Приведите формат определения функции.
3 Какие классы памяти используются при объявлении функций.
4 Охарактеризуйте формальные параметры приведенных функций.
5 Что означает запись **FILE***?

6 Что означает термин-поток?
7 Какие стандартные потоки открываются при выполнении программы?
8 Какие существуют режимы работы с файлами?
9 Какие заголовочные файлы содержат объявления библиотечных функций для работы с потоками.
10 Какие функции обеспечивают форматированный ввод/вывод данных?
11 Что такое буферизация и можно ли ею управлять?

Лабораторная работа № 18

Разработка программ с многофайловой структурой. Заголовочные файлы. Классы памяти переменных и функций (4 часа)

Цель работы: ознакомиться с написанием программ с многофайловой структурой, заголовочными файлами, изучить классы памяти переменных и функций, научиться создавать модульные программы и заголовочные файлы. Изучить область действия и время жизни переменных и функций с различными классами памяти.

Теоретические сведения

Управление многофайловыми проектами

Поскольку большинство программ состоит из нескольких файлов, желательно иметь возможность автоматической идентификации тех файлов, которые должны быть перекомпилированы и скомпонованы. Эти и многие другие обязанности выполняет встроенный администратор (менеджер) проектов системы Borland **C.**

Администратор проектов позволяет задавать те файлы, которые относятся к описываемому проекту. Когда осуществляется перекомпиляция проекта, администратор проектов автоматически обновляет информацию, которая хранится в файле проекта. В файл проекта входит следующая информация:

- имена всех файлов, входящих в проект;
- пути для поиска файлов;
- какие файлы зависят от других файлов, какие должны быть откомпилированы в первую очередь (решаются вопросы, касающиеся автоматически отслеживаемых зависимостей);
- какие компиляторы и параметры командной строки должны использоваться при создании каждой из частей программы;
- куда следует поместить результирующую программу;
- размер кода, размер данных и число строк, полученных в результате последней компиляции.

Использование администратора проектов

Использование администратора проектов не представляет затруднений. Для построения проекта следует:

- выбрать имя файла проекта (с помощью команды **Project|Open Project);**
- добавить к проекту имена исходных файлов (с помощью команды **Project| Add Item);**
- задать системе Borland **C** компиляцию файла (с помощью команды **Compile|Make EXE**).

Затем, когда в меню **Project** станут доступны команды ведения проекта, можно:

- добавлять имена файлов в проект или удалять их из него;

- задавать параметры файла, входящего в проект;
- просматривать содержимое файлов, включенных в конкретный проект.

Пример работы с администратором проектов.

Имеется программа, которая состоит из основного исходного файла с именем **mymain.c**, дополнительного файла **myfuncs.c**, содержащего определения функции и данные, обращения к которым имеются в основном файле, и файла **myfuncs.h**, где находятся объявления функций.

Файл mymain.c выглядит следующим образом:

```
#include <stdio.h>          // заголовочный файл в стандартном каталоге
#include "myfuncs.h"        // заголовочный файл в активном каталоге

main (int argc, char *argv[ ]) // передача в функцию main() параметров из
//окружающей среды: argc – количество параметров, argv[i] - массив строковых
//параметров, разделённых пробелом, argv[o] – имя программы
{
char *s;                    // класс памяти по умолчанию auto
if (argc > 1)
s = argv[1];                // выбор второго строкового параметра из массива
else
s = " вселенной";           // инициализация указателя строковой константой
printf("%s %s.\n",GetString(),s);  // вывод результирующей работы функции
// GetString()  и строки, связанной с указателем s
}
```

Файл myfuncs.c выглядит следующим образом:

```
char ss[] ="Приют на границе";   // массив ss[ ] проинициализирован строковой
// константой
char *GetString(void);      // определение функции
{
return ss;    // доступ к массиву из функции разрешён, т.к. класс памяти по
// умолчанию extern
}
```

А **файл myfuncs.h** выглядит следующим образом:

```
extern char *GetString(void);  // класс памяти глобальный - extern.
```

Пример

```
/*   ЗАНЯТИЕ N18
                    lab18.cpp  -  Основной файл проекта
  Разработал Петров Ю.В.
 Объявить глобальные переменные r (extern) и f (extern и static),
 а также локальные переменные.
 Объявить в отдельном файле функции, выполняющие работу с переменными
 с различными классами памяти. Определение функций также разместить
```

в отдельных файлах. Выполнить инициализацию переменных. Вывести значения рассчитанных глобальных и локальных переменных на экран */

```
#include <iostream.h>
#include <conio.h>
#include "my_18.h"

int r=5;        //Глобальная переменная r
extern float f;  //Глобальная переменная f = 0

void main()
{ auto int r; // Локальная переменная (auto) r
  r = Sum_Variable(4, 5); //Функция объявлена в файле "my_18.h" r=9
  clrscr();
  cout << "Локальная (auto) r=\t" << r << " Глобальная r="<< ::r << endl;
  r+=10;                    //Локальная переменная    r=19
  ::r = Sum_Variable(4, 3);    //Глобальная переменная ::r=7
  cout << "Локальная (auto) r=\t" << r << " Глобальная r=\t"<< ::r << endl;
  cout << "Глобальная f=" << f;
  cout << endl << " Локальная (auto)     f=\t" << Function_1()
               << " Статическая (static) f=\t" << Function_2();
  cout << endl << " Локальная (auto)     f=\t" << Function_1()
               << " Статическая (static) f=\t" << Function_2();
  cout << endl << " Локальная (auto)     f=\t" << Function_1()
               << " Статическая (static) f=\t" << Function_2();
  getch();
}
/* Локальная (auto) r=  9   Глобальная r= 5
   Локальная (auto) r= 19   Глобальная r= 7
   Глобальная       f= 0.2
   Локальная (auto) f= 8    Статическая (static) f= 2
   Локальная (auto) f= 8    Статическая (static) f= 3
   Локальная (auto) f= 8    Статическая (static) f= 4    */
```

//***lab18f1.cpp – первый вспомогательный файл проекта***

```
// Объявление переменной r и определение функций
extern r;          // Глобальная переменная r

float Function_1()
{ auto float f= r; // Локальная переменная f
  return ++f;
}

int Sum_Variable(int a, int b)
 {return a + b;
 }
```

// ***lab18f2.cpp – второй вспомогательный файл проекта***

```
//Определение функции
static float f=1;//Статическая  переменная (static) f
float Function_2()
{  return ++f;
}
```

//***my_18.h – заголовочный файл***

```
//Объявление переменных и функций
int Sum_Variable(int, int);
float Function_1();
float Function_2();
static float f = 0.2;
```

Ход работы

1 Изучить теоретические сведения.
2 В соответствии с индивидуальным заданием разработать алгоритм и программу, содержащую заголовочный файл и несколько модулей (2-3). В каждом модуле разработать отдельные функции.
3 Выполнить объявление переменных на внешних и внутренних уровнях с различными классами памяти.
4 Разработать функции, вызываемые из дополнительных модулей. Использовать объявленные переменные с различной областью действия (классами памяти).
5 Создать заголовочные файлы, содержащие информацию о функциях в дополнительных модулях.
6 Написать основную программу, подключающую необходимые заголовочные модули и использующую функции и переменные из других модулей.
7 Показать изменение переменных в различных областях действия.
8 Проверить доступ к переменным с различными классами памяти (внутри блока, модуля и в других модулях).
9 Как ограничивается доступ к членам класса? Как ограничивается доступ к членам класса?
10 Проверить выполнение инициализации переменных с классом памяти **static**, объявленных на внешнем уровне и внутри функции при нескольких вызовах.
11 Проверить доступ к функциям, объявленным с различными классами памяти.
12 Разработать функции, имеющие одно имя и различные области действия. Осуществить вызов этих функций.
13 Набрать программу на компьютере, и устранить ошибки.
14 Получить результат.
15 Оформить отчет.

16 Подготовиться к защите лабораторной работы, изучив контрольные вопросы по данной теме.

Индивидуальное задание к лабораторной работе №18

Составить проект для многофайловой структуры программы. Разработать заголовочный файл содержащий глобальные данные и объявления внешних функций. Разработать вспомогательный файл содержащий определения внешних функций и объявления и переопределения данных.

Проект должен содержать:

- объявление и использование глобальных и локальных переменных;
- передача глобальных данных в качестве параметров функций;
- переопределение глобальных данных внутри функций;
- вызов из вспомогательного файла внешней функции;
- вызов из основного файла внешних функций;
- переопределение функций во вспомогательных файлах.

Индивидуальные задания находятся в таблицах 18.1 и 18.2

Таблица 18.1 – индивидуальные задания

Варианты	Структура проекта
1-10	основной файл 2 вспомогательных файла заголовочный файл
11-20	основной файл 3 вспомогательных файла заголовочный файл
21-30	основной файл 4 вспомогательных файла

Таблица 18.2 - индивидуальные задания

Вариант	локальная функция	внешняя функция	переопределенная функция	локальные данные	внешние данные	переопределенные данные
1,11,21	auto int first (int, float)	void second (int *,float *)	static float *first(float)	auto int var1	const float var2	static long int var1
2,12,22	auto cdecl int first	void second (int *,float *)	static pascal char	auto long unsigned	double var2	static float var1

	(int, float)		*first(int,…)	int var1		
3,13,23	near auto int first (int, float)	void far second (int *,float *)	huge static int *first(double)	auto int var1	extern float var2	static short unsigned int var1
4,14,24	auto int first (near int, near float)	far void * pascal near second (far int *,far float *)	static unsigned int *first(char)	near auto double var1	extern unsigned short int var2	static char var1
5,15,25	pascal auto int first(int)	char far * (far * second) (int *,float *)	cdecl static far short int *first(float, float)	register char var1	far double var2	static int var1
6,16,26	cdecl auto int first(int, float)	extern near void * second (int *,float *)	static near double * first(void, …)	auto long int var1	float var2	static double var1
7,17,27	huge auto int first(near int *, far int *)	extern void second (int *,float *)	near static float *first(void)	register int var1	extern near float var2	near static double var1
8,18,28	far int * auto pascal huge first(huge int, huge float)	void second (int *,float *)	static short int *first(float *)	huge int var1	far double var2	static long unsigned int var1
9,19,29	auto int first(int, float)	near void * extern far pascal second (int *,float *)	static float *first(float)	auto int var1	far extern float var2	near static char var1
10,20,30	auto far int * (first)(int *, …)	void second (int *,float *)	int far * pascal far first()	auto float var1	extern huge float var2	extern long int var1

Требования к содержанию отчёта приведены в лабораторной работе №1.

Контрольные вопросы для подготовки и самостоятельной работы

1 Какие классы памяти (КП) допустимы применительно к переменным и функциям?
2 Какие классы памяти используются для переменных и функций по умолчанию?

3 Назовите область действия и время жизни данных и функций с различными классами памяти.
4 Можно ли использовать переменные с КП **register** в левой части выражения присваивания?
5 В каких случаях можно использовать переменные, а также функции с одинаковыми именами?
6 Можно ли переобъявить переменную во вложенных блоках? Какова область действия таких переменных?
7 Можно ли получить доступ к глобальной переменной в блоке, где объявлена локальная переменная с тем же именем?
8 Производится ли инициализация переменных с различными классами памяти?
9 Сколько раз производится инициализация переменных с КП **static**, объявленных в теле функции?
10 Какую информацию содержат заголовочные файлы и где они находятся?
11 Как предотвратить повторное включение заголовочных файлов в нескольких модулях?

Лабораторная работа № 19

Изучение графических средств С
(2 часа)

Цель работы: приобрести практические навыки в использовании графических функций языка **С .**

Теоретические сведения

Графические функции предназначены для управления видеорежимами работы дисплея, выводом графической информации на экран.

Графические функции

void far detectgraph(int far *graphdriver, int far *graphmode); – определение доступного видео-драйвера.

void far initgraph(int far * graphdriver, int far *graphmode, char far *pathtodriver); - установка видеорежима.

vo**id far setgraphmode(int mode);** - установка видеорежима.

void far restorecrtmode(void); - временный переход из графического видеорежима в текстовый.

void far closegraph(void); - закрытие графической системы.

void far setvisualpage(int page); - установка активной видеостраницы.

void far setactivepage(void); -вывод на активную видеостраницу.

int far getmaxx(void); - определение максимального значения координаты **x.**

int far getmaxy(void); - определение максимального значения координаты **y.**

void far setviewport(int left, int top, int right, int botton, int clip); - установка нового графического окна.

void far getviewsettings(struct viewporttype far *viewport); - получение параметров текущего окна.

void far moveto(int x, int y); void far moverel(int dx, int dy); - перемещение текущей графической позиции в координаты **x**, **y** или на величину **dx, dy.**

void far setlinestyle(int linestyle, unsigned upattern, int thickness); - установка типа линии.

i**nt far getx(void); -** получить текущую графическую позицию **(x).**

int far gety(void); - получить текущую графическую позицию **(y).**

void far clearviewport(void); - очистка текущего графического окна.

void far cleardevice(void); - очистить активную видеостраницу.

int far getmaxcolor(void); -определить максимальное количество цветов.

void far setpalette(int colornum, int color); - установка палитры.

void far setbkcolor(int color); -установка цвета фона.

Доступ к пикселям

unsigned far getpixel(int x, int y); - получить текущие параметры пикселя.

void far putpixel(int x, int y, int color); - вывести пиксель с параметрами.

Графические примитивы

void far bar(int left, int top, int right, int botton);
void far bar3d(int left, int top, int right, int botton, int depth, int topflag);
void far fillpoly(int numpoints, int far *polypoints);
void far fillelipse(int x, int y, int xradius, int yradius);
void far pielipse(int x, int y, stangle, int endangle, int radius);
void far sector(int x, int y, int stangle, int endangle, int xradius, int yradius);
void far line(int x1, int y1, int x2, int y2);
void far linerel(int dx, int dy);
void far lineto(int x, int y);
void far rectangle(int left, int top, int right, int botton);
void far drawpoly(int numpoints, far *polypoints);
void far circle(int x, int y, int radiuces);
void far arc(int x, int y, int stangle, int endangle, int radius);
void far ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius);

Пример

```
/*    ЗАНЯТИЕ N 19
   Разработал Гармаш В.Н.
 Объявить массивы для вывода текста в графическом режиме,
 выполнить их инициализацию. Инициализировать графический режим работы.
 Выполнить расчеты и построить заданную геометрическую фигуру.
 Вывести тексты на экран с применением необходимых функций.  */
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <DOS.H>
#define ANGLES 15
#define PIXEL_COUNT 1000
#define DELAY_TIME  100  /* in milliseconds */

char *str[]={"А","л","ь","ф","а"," ","Ц","е","н","т","а","в","р","а","."};
char st[]="_";
void demo(int x,int y,int size,int color);
void demo1(void);

void main()
{// Инициализация графического режима работы
 int graphdriver=DETECT,graphmode,errorcode;
 initgraph(&graphdriver,&graphmode,"");
 errorcode = graphresult();
 if (errorcode != grOk)
```

```
   { printf("Graphics error: %s\n", grapherrormsg(errorcode));
     printf("Press any key to halt:");
     getch();
     exit(1);
   }
   //Рисование рамки по контуру экрана
 setcolor(9); //Установка цвета
 setlinestyle(0,0,3); //Установка типа линии
 line(0,0,getmaxx(),0); // Рисование линии
 line(0,0,0,getmaxy());
 line(getmaxx(),0,getmaxx(),getmaxy());
 line(0,getmaxy(),getmaxx(),getmaxy());
 setcolor(10); // Вывод текста "ЗАДАНИЕ 19"
 settextstyle(GOTHIC_FONT,HORIZ_DIR,4);
 outtextxy(260,10,"ЗАДАНИЕ 19");
 demo(320,250,125,11); // Функция вывода рисунка
 demo1();      // Функция вывода текста "Альфа Центавра"
 getch();
 closegraph(); // Окончание графического режима работы
 clrscr();
}

void demo(int x,int y,int size,int color)
 { setcolor(color);
   setlinestyle(0,0,1);
   int xx[ANGLES],yy[ANGLES],i,j;
   for (i=0;i<=ANGLES-1;i++)
     { xx[i]=x+(int)(cos(i*2*M_PI/ANGLES)*size);
       yy[i]=y-(int)(sin(i*2*M_PI/ANGLES)*size);
     }
   for (i=0;i<=ANGLES-1;i++)
     { for (j=0;j<=ANGLES-1;j++)
       if (i!=j)
           { line(xx[i],yy[i],xx[j],yy[j]);
           }
     }
 }

void demo1(void)
 { int f1 = installuserfont("rtri.CHR"); //Установка шрифта
  unsigned int sz;
  void far *ptr;
  sz=imagesize(10,10,50,50);
  ptr=malloc(sz);
  getimage(10,10,50,50,ptr);
  int size = 2,tt,ff,i;
```

```
 while(!kbhit())
  { tt=0;
    settextstyle(DEFAULT_FONT, HORIZ_DIR, size);
    for(i=0; i<3; i++)
     { setcolor(3); outtextxy(260,60,st); delay(90);
        setcolor(0); outtextxy(260,60,st); delay(90);
     }
    setcolor(3); ff=260;
   for(i=0; i<15; i++)
    { setcolor(3);
     settextstyle(f1, HORIZ_DIR, 4);
     outtextxy(260+tt, 50, str[i]);
     tt+=textwidth(str[i]);
     sound(30); delay(20); nosound();
     settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
     outtextxy(260+tt+4, 60, st); delay(40);
     setcolor(0);
     outtextxy(260+tt+4, 60, st); delay(10);
    }
   settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
   for(i=1;i<4;i++)
    { setcolor(3); outtextxy(260+tt+4, 60, st); delay(90);
      setcolor(0); outtextxy(260+tt+4, 60, st); delay(90);
    }
   setcolor(3);
   ff=tt+260; tt=0;
   for(i=14;i>=0;i--)
    { settextstyle(f1, HORIZ_DIR, 4);
     tt+=textwidth(str[i]);
     sound(30); delay(20); nosound();
     putimage(ff-tt, 50, ptr, COPY_PUT);
     settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
     setcolor(3); outtextxy(ff-tt-4, 60, st); delay(40);
     setcolor(0); outtextxy(ff-tt-4, 60, st); delay(10);
    }
  }
}
```

Ход работы

1. Изучить теоретические сведения.
2. В соответствии с индивидуальным заданием разработать алгоритм решения задачи и оформление интерфейса программы.
3. Подготовить и разметить на экране эскиз чертежа детали в масштабе 1:1.
4. Составить программу с использованием графических функций языка **C** для вывода на экран подготовленной графической информации. Размеры, указанные на чертеже, ввести с клавиатуры.

5 Набрать программу на компьютере и устранить ошибки.
6 Получить результат.
7 Оформить отчет.
8 Подготовиться к защите лабораторной работы, изучив контрольные вопросы по теме.

Индивидуальное задание к лабораторной работе №19.
Варианты индивидуальных заданий находятся в таблице 19.1.
Таблица 19.1 - индивидуальные задания

Вариант	Номер рисунка	Вариант	Номер рисунка
1	а	16	г
2	б	17	д
3	в	18	е
4	г	19	ж
5	д	20	з
6	е	21	и
7	ж	22	к
8	з	23	л
9	и	24	м
10	к	25	а
11	л	26	б
12	м	27	в
13	а	28	г
14	б	29	д
15	в	30	е

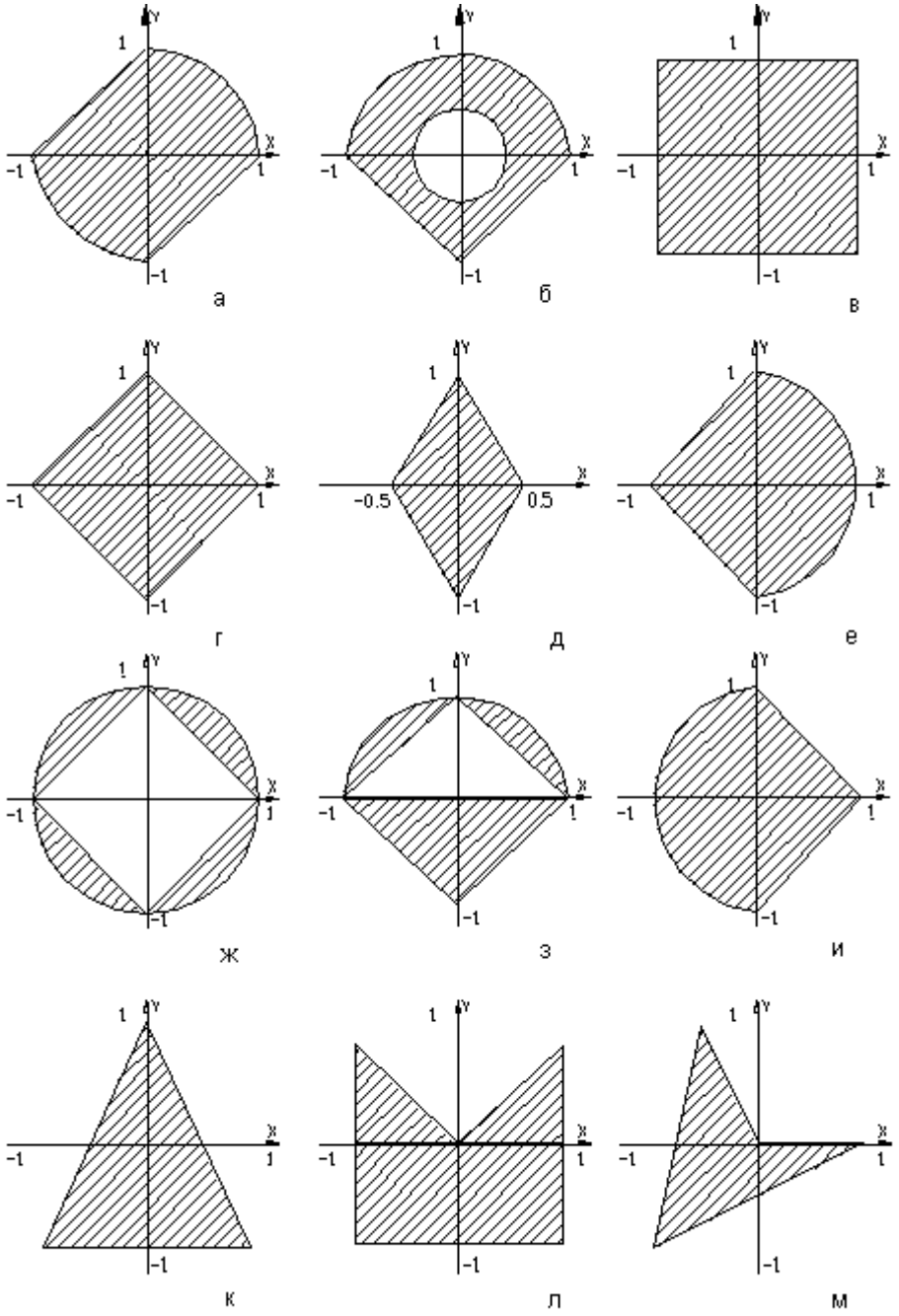

Рисунок 19.1 - индивидуальные задания

Требования к содержанию отчёта приведены в лабораторной работе №1.

Контрольные вопросы для подготовки и самостоятельной работы

1 Какая функция применяется для установки видеорежима, инициализации графического режима работы?
2 Что означают параметры функций в приведенном примере?
3 Как закрыть графический режим работы?
4 Можно ли получить и установить координаты курсора на экране? В чём измеряются эти координаты?

5 Какие функции позволяют устанавливать цвета для выводимой информации, цвет фона, осуществлять различные виды заливки изображения?
6 Какие графические примитивы можно изобразить с помощью библиотечных функций? Какие параметры необходимы для построения этих примитивов?

Лабораторная работа №20

Разработка программ с использованием классов (2 часа)

Цель работы: изучить синтаксические конструкции для объявления, определения и использования классов. Разобраться с особенностями использования классов в языке **C.**

Теоретические сведения

Объект - это абстрактная сущность, наделённая характеристиками объектов реального мира. В **C** объекты играют очень важную роль. Всё, чем манипулирует программа, может рассматриваться как объект (экземпляр) определённого класса. При выполнении программы объекты создаются и удаляются. Они взаимодействуют с другими объектами и могут быть помещены в массивы, списки, группы, коллекции, и т.д.

Объекты в **C** - это программные конструкции (переменные), формируемые так называемыми классами. Определение переменной класса (объекта) также называется созданием экземпляра класса. За создание своих классов полную ответственность несёт сам программист. Но он может получить доступ и к классам, разработанным другими программистами. Например, к классам, которые находятся в библиотеке контейнеров или библиотеке потоков компилятора **C.**

Объявление и определение класса

Класс - это пользовательский тип данных, (аналогично структуре), который содержит (включает, инкапсулирует) не только объявления данных, но и функции. Эти функции называются функциями-членами класса и определяют, что может делать класс. Структуры в **C** также могут содержать функции.

Для того, чтобы использовать класс, его нужно вначале объявить точно так же, как это делается со структурами. И так же как для структур, полное объявление класса может появиться в программе только один раз. Рассмотрим пример объявления простого класса:

```
class Counter {  // имя типа Counter
long  count;// данное-член класса, объявлено в разделе private по умолчанию
public:       // разделён public, данные доступны из программы
void SetValue (long);  // функции-члены класса, объявлены в разделе public
long GetValue ();
};
```

Ключевое слово **class** вводит объявление класса. Далее следует имя класса **(Counter).** Тело класса должно заключаться в фигурные скобки, после которых стоит точка с запятой. Классы могут содержать не только объявления функций, но и их полные определения **inline** - функции. Функции внутри классов могут быть настолько длинными и сложными, насколько это необходимо.

Переменные, объявленные внутри класса, принадлежат этому классу. В некоторых случаях переменные могут разделяться (использоваться) различными эк-

земплярами класса **(**с классом памяти **static).** Идентификаторы (имена) переменных и функций внутри класса застрахованы от конфликтов с идентификаторами других классов. Класс - это замкнутый программный комплект с собственными идентификаторами.

Для идентификаторов класса применимы те же правила, что и для остальных типов или имён переменных. В **C** для идентификаторов предельная длина не определена, но в Borland **C** максимальная длина равна 32 символам. По умолчанию все 32 символа являются значащими. Регистры букв (строчная или прописная) в идентификаторах различаются.

Тело класса

Переменная **count** объявлена внутри класса. Таким образом, **count** - это переменная-член **- (**данное-член**)** класса. Любая переменная, определённая в классе, имеет область видимости класса**.** Область видимости переменной-члена простирается от точки её объявления в классе до конца объявления класса.

В **C** данные и функции-члены, объявленные внутри класса, не содержат спецификацию класса памяти (кроме **static***)* и лексически принадлежат области действия данного класса.

Класс имеет столько переменных, сколько необходимо. Переменные могут быть любого типа, включая другие классы, указатели на объекты классов и даже указатели на динамически распределяемые объекты.

Класс **Counter** содержит объявление функций **SetValue ()** и **GetValue (),** которые называются функциями-членами класса. Эти функции пока не определены, они только объявлены. Как и другие функции в C, функции-члены должны быть объявлены до использования. Объявление должно быть полным, включая тип возвращаемого значения и типы аргументов.

Пример

```
void Counter::Setvalue (long value)
{
count = value;  // изменение значения
// данного-члена count
}
```

```
long Counter::GetValue ()
{
return count; // вернуть значение
// данного члена count
 }
```

При определении функции-члена после типа возвращаемого значения указывается, членом какого класса является функция. Для этого нужно написать имя класса и поставить за ним два двоеточия – оператор разрешения области видимости.

Формат определения функции –члена класса.

Определение функций для класса **Counter** обычно осуществляется в других модулях:

<Тип_ возвращаемого_значения> <Имя_ класса> ::
<Имя_функции> (<Объявления_ параметров>*)*
{<Тело _функции>}

Доступ к членам класса в общем случае осуществляется с помощью нотации

<Имя_объекта> <Имя_члена_класса>,

кроме функций-членов, которые имеют прямой доступ к данным-членам.

Класс, объявленный внутри другого класса, называется вложенным классом. Его имя является локальным для охватывающего класса. Вложенный класс имеет область действия, лежащую внутри области действия охватывающего класса. Эта вложенность является чисто лексической. Вложенный класс не имеет никаких дополнительных привилегий в доступе к элементам охватывающего класса (и наоборот). Уровень вложенности классов указанным образом является любым. Например:

```
struct outer
{
 typedef  int tint;  // выражение outer::tint - это синоним типа int tint x;
struct inner         // выражение outer::inner - это вложенная структура (класс)
{static int x;};      // статическая переменная (член) структуры типа inner
tint y;
int f();              // функция-член структуры типа outer.
}

int outer::f()        // определение функции-члена структуры типа outer
{
tint y1=x;          // x видим и доступен здесь, т.к. функции–члены имеют доступ к
                      //данным-членам класса (структуры)
return y1;
}
outer::inner::x=5;  // определение статического данного-члена класса
outer::  tint y=2;
```

Доступ к статическому данному-члену класса осуществляется по имени класса (**outer:: inner::),** а не по имени объекта класса.

Использование класса

Для того чтобы использовать класс, нужно объявить (создать) объект этого класса. Объекты класса определяются точно так же, как структурные или скалярные переменные. Объявление класса должно предшествовать использованию класса. Пользователю предоставляется описание класса, но не обязательно его внутренняя реализация. Чтобы объявить переменную (объект) **people** типа **Counter,** используется следующую запись:

```
Counter people;  // аналогично объявлению любой переменной int a;
Counter* ptr      // аналогично int* p;
```

Для вызова функции-члена объекта класса используется та же запись, что и для обращения к элементу структуры: за точкой следует имя элемента **people.GetValue()**. Аналогичен вызов и при использовании указателей **ptr-> GetValue()** В остальном использование функций-членов ничем не отличается от использования традиционных функций **C.**

Инкапсуляция. Управление доступом к членам класса

Класс включает как данные, так и функции (код). Доступ к элементам класса управляем. Это управление касается не только данных, но и кода.

Чтобы использовать класс необходимо знать, какие функции и какие данные доступны. Набор используемых функций и доступных данных называется пользовательским интерфейсом класса.

Главная забота при создании класса - скрыть как можно больше информации. Это накладывает ограничения на использование данных или кода внутри класса. Существует три вида пользователей класса:

- сам класс (функции-члены имеют прямой доступ к членам класса);
- обычные пользователи (доступ из функции **main()** и функций);
- производные классы (доступ из классов-наследников к членам базовых классов)

Каждый вид пользователей обладает разными привилегиями доступа. Каждый уровень привилегий доступа ассоциируется с определенным ключевым словом. Уровней привилегий доступа в **C** всего три, они определяются ключевыми словами:

- **private** – закрытые члены класса;
- **protected** – защищённые члены класса;
- **public** – открытые члены класса.

public - элемент может использоваться любой функцией и операторами программы;
private- элемент может использоваться только функциями-членами и "друзьями" класса, в котором они объявлены;
protected - то же самое, что для **private**, но кроме того, элемент может быть использован функциями–членами и функциями-«друзьями» классов, производных от объявленного класса.

Элементы класса по умолчанию имеют атрибут **private,** поэтому спецификаторы доступа **public** и **protected** должны задаваться явно.

Элементы **struct** по умолчанию имеют атрибут **public**, но вы можете переопределить доступ при помощи спецификаторов доступа **private** или **protected.**

Элементы **union** по умолчанию имеют атрибут **public**. Переопределить его нельзя. Задавать спецификаторы доступа для элементов объединения недопустимо.

Модификатор доступа (по умолчанию или заданный) остается действительным для всех последующих объявлений элементов, пока не встретится другой модификатор доступа. Например:

```
class X {
int i;      // X::i по умолчанию private
char ch;    // X::ch аналогично X::i
public:
int j;    // следующие два
// элемента - public
int k;     // открытые данные-члены
protected:
int l;      // X::l - protected
};
```

```
struct Y {
int i;      // Y::i по умолчанию public
private:
int j;      // Y::j - private
public:
int k;      // Y::k - public
};
union Z {
int i;  // public по умолчанию,
// других вариантов нет
double d; }; // public
```

Спецификаторы доступа могут следовать и повторяться в любой удобной последовательности.

Пример:

```
class   AccessControlExample {
int  value_1; // приватные члены-класса по умолчанию, доступны только
void f_1 (long);  //функциям – членам класса
private:
int   value_2;// тоже приватные, доступны только
intf_2 (char *);   //функциям–членам класса
public:
char*  value_3;   // общедоступные члены класса
long f_3 ();
protected:
int  value_4; // защищенные члены, доступны только функциям – членам
void  f_4 (long);   //класса и классов-наследников (производных классов)
};
```

Приватные (**private**) члены класса имеют наиболее ограниченный доступ. Только сам класс или классы, объявленные как дружественные (**friend**) имеют доступ к приватным членам. Производные классы не имеют доступа к приватным членам родительского класса. Концепция сокрытия информации реализована в языке только частично, чтобы предотвратить нечаянный доступ к внутренним переменным или функциям класса, поскольку преднамеренный доступ можно всегда получить к любой части класса в обход обычных способов.

Для того, чтобы использовать общедоступные члены класса внутри функции, необходимо иметь доступ к данным-членам или функциям-членам или и к тому, и к другому. Чтобы сделать члены общедоступными, их нужно объявить в секции **public.**

Когда класс используется как базовый для других классов, можно сделать его члены доступными только для функций производных классов с помощью слова **protected** .

Классы памяти для объектов

Класс, в смысле распределения памяти, рассматривается как вид структуры. Как и структуры, классы могут быть объявлены с такими классами памяти, как автоматический **(auto),** внешний **(extern)** или статический **(static).**

Переменные (Идентификаторы), объявленные внутри класса, видимы только в сочетании с объектами класса. При неоднозначном толковании должен использоваться оператор разрешения области видимости **(::)**.

Использование данных-членов класса

Инкапсуляция позволяет скрыть какие-то данные и функции внутри класса. Данных может быть столько, сколько нужно и сколько позволяет память. Однако их может и не быть вообще.

Попытка инициализировать данные внутри объявления класса ошибочна. Класс - это не объект и память для него не будет выделена до тех пор, пока не будет создан экземпляр этого класса. Данные, объявленные в классе, следует рассматривать как поля структуры, а не как переменные. Как и для структуры, нужно объявить объект с типом (именем) класса, а затем инициализировать данные члены**.**

Данные-члены создаются с тем же классом памяти, что и объект класса. Если объект объявляется автоматическим, то все его данные будут автоматическими. Статические данные-члены являются исключением из этого правила: когда создаётся объект со статическими членами (данными), память под них не выделяется, потому что это приведёт к появлению нескольких копий статических данных. Если в классе данное объявляется статическим **(static)**, то все экземпляры класса будут разделять одно и тот же данное-член. Статический член данных, как и глобальная переменная, размещается в фиксированной области памяти на стадии компоновки. Для объявления или инициализации статических данных используется в точности та же нотация, что и для глобальных переменных **(extern*)*.**

Если данное-член класса объявлен как статический, то компоновщик выделяет под него память только один раз, когда производится объявление этого данного в классе. Поэтому инициализировать статическую переменную можно в момент её объявления. Статические элементы всегда остаются статическими.

Привилегии доступа к статическим данным отличаются от привилегий доступа к нестатическим. К статическому данному необходимо всегда обращаться через имя класса и оператор разрешения области видимости **(::*),*** независимо от того, объявлен он как приватный, общедоступный или защищенный**.**

Оператор разрешения области видимости **(::)** – используется для доступа к данным- членам:

- через функцию-член класса;
- через класс, объявленный дружественным для данного класса.

Для доступа к открытым членам класса используется оператор точка **(.)** с объектами или оператор **(->)** с указателями на объекты класса.

Для доступа к закрытым и защищённым членам класса используются функции-члены открытой части класса.

Ход работы

1 Изучить теоретические сведения.
2 В соответствии с индивидуальным заданием разработать структуру класса, сделать определение функций-членов класса **(class)**, разработать алгоритм использования объектов и указателей на объекты класса для доступа к данным и функциям- членам. Проверить возможность доступа к членам класса в разделах **private, public, protected.** В разделах объявить минимум по одному данному-члену, включая статические **(static).**
3 Набрать программу на компьютере.
4 Устранить ошибки.
5 Получить результат.
6 Оформить отчет.
7 Подготовиться к защите лабораторной работы, изучив контрольные вопросы по данной теме.

Индивидуальное задание к лабораторной работе №20.

Составить программу для объявления и использования данных типа класс согласно индивидуальному заданию, приведенному в таблице 20.1.

Таблица 20.1 - индивидуальные задания

Базовый класс	Производный класс	Производный класс
1 источник света	костёр	фонарь
2 хранилище	зернохранилище	элеватор
3 грузоподъемное средство	кран	мостовой кран
4 животное	кот	кот сиамский
5 растение	дерево	дуб
6 транспортное средство	самолет	дельтоплан
7 транспортное средство	автомобиль	легковой автомобиль
8 транспортное средство	корабль	танкер
9 топливо	нефть	бензин
10 средство информации	книга	журнал
11 хранилище информации	жесткий диск	дискета
12 средство визуализации	электронно-лучевая трубка	жидкокристаллический экран
13 источник тепла	солнце	костер
14 источник тока	батарея	аккумулятор
15 продукты	овощи	картофель
16 таймер	часы	часы наручные
17 морская фауна	рыба	колбаса
18 пишущее устройство	ручка	шариковая ручка
19 режущее устройство	нож	ножницы
20 продукты	хлеб	батон

21 инструмент	напильник	надфиль
22 металл	сталь	закалённая сталь
23 мебель	кресло	стул
24 здание	цех	кузнечный цех
25 стройматериалы	кирпич	фасонный кирпич
26 одежда	куртка	пиджак
27 устройство передачи крутящего момента	редуктор	червячный редуктор
28 обувь	ботинки	кеды
29 головной убор	шапка	папаха
30 погода	дождь	ураган
31 спорт инвентарь	лыжи	санки
32 средство связи	телефон	радиотелефон
33 колесо	зубчатое колесо	косозубое колесо
34 водоём	море	водохранилище
35 оптический прибор	труба	бинокль
36 флора	лес	куст
37 посуда	кастрюля	тарелки

Требования к содержанию отчёта приведены в лабораторной работе №1.

Контрольные вопросы для подготовки и самостоятельной работы

1 Дайте определение понятиям: объект, класс, данные-члены класса, функции-члены класса.
2 В чём отличие между классом и объектом класса?
3 Можно ли определить функцию внутри класса?
4 Можно ли использовать в различных классах одинаковые имена данных и функций-членов?
5 Какие классы памяти можно использовать при объявлении объектов?
6 Можно ли использовать классы памяти при объявлении членов класса, почему?
7 Что означает термин инкапсуляция?
8 Как производится управление доступом к элементам класса?
9 Что означают спецификаторы доступа **public, protected, private**
10 Назовите спецификатор доступа по умолчанию к членам класса и структуры.
11 Ограничивается ли количество и порядок следования разделов в классе?
12 Назовите особенности использования статических **(static*)*** членов класса.
13 Для чего используется оператор разрешения области видимости **(::)**?
14 Могут ли структуры содержать функции в качестве элементов?
15 Могут ли структуры и классы быть вложенными?
16 Какова область действия членов класса?
17 Как получить доступ к статическим членам класса из программы?
18 Можно ли объявить тип в теле класса при объявлении класса?

Лабораторная работа №21

Использование конструкторов и деструкторов (2 часа)

Цель работы: изучить и научиться использовать механизм работы с конструкторами и деструкторами.

Теоретические сведения

Конструкторы и деструкторы

Существует несколько специальных функций-членов, определяющих, каким образом объекты класса создаются, инициализируются, копируются и разрушаются. Конструкторы и деструкторы являются наиболее важными из них. Они обладают большинством характеристик обычных функций-членов: объявляются в классе и определяются в пределах класса или вне его. Однако, они обладают и некоторыми уникальными свойствами:

1 Они имеют имя, совпадающее с именем класса **(**деструкторы с символом тильды ~)
2 Конструкторов может быть несколько, деструктор только один.
3 Они не имеют объявлений типа возвращаемых значений (даже **void**).
4 Они не могут быть унаследованы, хотя производный класс может вызвать конструкторы и деструктор базового класса.
5 Конструкторы, как и большинство функций языка **C,** могут иметь аргументы по умолчанию или использовать списки инициализации элементов.
6 Деструкторы могут иметь атрибут **virtual**, но конструкторы не могут.
7 Нельзя работать с адресами этих функций.

Пример

void *ptr=base::base; // недопустимо**, base()–**функция-конструктор класса **base**

8 Вызвать конструктор тем же образом, что и обычную функцию, нельзя. Вызов деструктора допустим только с полностью уточненным именем.

Примеры

```
X *p;          // указатель на класс X
X x;           // объявление объекта x класса X
p=&x;          //указатель p хранит адрес объекта x
p->X::~X(); // допустимый вызов деструктора
X::X();        // недопустимый вызов конструктора
```

9 При объявлении объектов компилятор вызывает один из конструкторов автоматически.
10 Аналогично при уничтожении объектов компилятор автоматически вызывает деструктор.
11 Конструкторы и деструкторы при необходимости динамического выделения объекту памяти могут выполнять вызовы операторов **new** и **delete.**
12 Конструктор создает объект **x** и инициализирует его данные-члены. При копировании объекта данного класса происходит неявный вызов соответствующего конструктора. Данные-члены объекта класса, у которого все эле-

менты общедоступны **(public)** и без конструкторов или базовых классов (обычно это структура) могут инициализироваться при помощи списка инициализаторов. Деструктор разрушает объект класса, созданный конструктором, когда этот объект выходит из области видимости.

Конструкторы глобальных объектов вызываются до вызова функции **main**().

Локальные объекты создаются, как только становится активной область действия (видимости) объекта. При этом конструктор вызывается каждый раз.

```
class X
{...public:
X();              // конструктор класса X по умолчанию
                  // X(X); недопустимо
X(const X&); // конструктор копирования – передаётся ссылка на X
~X( );            // деструктор класса X
};
```

Формальные параметры конструктора при его объявлении могут быть любого типа, за исключением класса, элементом которого является данный конструктор. Конструктор копирования принимает в качестве параметра ссылку на свой собственный класс. Конструктор, не воспринимающий параметров, называется конструктором по умолчанию: **X::X().**

X x; // объявлен объект **x** класса **X**, вызывается конструктор по умолчанию

Как и все функции-члены, конструкторы могут иметь аргументы, используемые по умолчанию. Например, возможны следующие объявления конструкторов в классе:

Объявление класса	Определение конструкторов вне класса
class X { **int I, j** по- умолчанию **private** **public:** **X();** // конструктор по умолчанию **X(int);** // перегруженные конструкторы **X(int, int);** // могут вызываться с одним // **X(int**) или двумя **X(int, int)** аргументами **X(int a = 5, int в= 6);** //может вызываться без аргументов или с одним или двумя // аргументами **X(const &);** // конструктор копирования **};**	**X::X{i=∅; j=∅}** **X::X(int a) {i=a;** **j=∅;}** **X::X(int a, int b)** **{i=a; j=b;}**

При вызове конструкторов следует избегать неоднозначностей **main()**

Инициализация объектов класса

Объекты классов с конструкторами могут инициализироваться при помощи задаваемых в круглых скобках списков инициализаторов. Этот список используется как список передаваемых конструктору аргументов

```
main()
{
X one; // вызывается конструктор X::X() по умолчанию для объекта one
// класса X
X two(1);     // используется конструктор X::X(int)
X three = 1; // вызывается X::X(int) – альтернативная форма со знаком (=)
X three(10, 8) // использовать нельзя, неоднозначно задан конструктор
// X::X(int a=5, int b=6)
X four = one; // вызывается X::X(const X&) для копирования
// объекта one в four
X five(two);  // вызывает X::X(cont X&) для копирования объекта two в five
   }
```

Конструктор воспринимает значения в качестве параметров и выполняет присваивание данным членам в теле функции конструктора. Определение конструктора как и любой функции-члена может производиться в теле класса

```
class X
{
int i, j;
public:
X(int a, int b) { i = a; j = b } // определение конструктора в классе, он имеет
      // доступ к i и j
X(int a, int b) : a(i), b(j) {}  // альтернативная форма для инициализации a и b
};      i(a), j(b)
```

В обоих случаях инициализации **X x(1, 2)** присваивает значение **x::i=1** и значение **x::j=2.**

Деструкторы

Деструктор класса вызывается для разрушения элементов объекта до уничтожения самого объекта. Деструктор представляет собой функцию-член, имя которой совпадает с именем класса, перед которым стоит символ тильды (~). Деструктор не может воспринимать каких-либо параметров, а также не объявляет возвращаемого типа или значения.

```
class X
{…
public:
~X();  // деструктор класса X
};
```

Если деструктор не объявлен для класса явно, компилятор генерирует его автоматически.

Вызов деструкторов

Вызов деструктора выполняется неявно, когда объект выходит из своей объявленной области действия. Для локальных переменных деструкторы вызываются, когда перестает быть активным блок, в котором они объявлены. В случае глобальных переменных деструкторы вызываются как часть процедуры выхода после функции **main()**. Необходимость деструкторов в значительной степени связана с указателями и динамическим выделением памяти.

Когда указатели, связанные с объектами, выходят за пределы области действия, неявного вызова деструктора не происходит. Это значит, что для уничтожения такого объекта оператор **delete** должен задаваться явно.

Деструкторы вызываются в обратной последовательности относительно вызова конструкторов, т.е. сначала вызываются деструкторы производных классов, потом базовых – вверх по иерархии.

При выходе из программы с использованием функции **exit()** деструкторы для каких-либо локальных переменных в текущей области действия не вызываются. Глобальные переменные уничтожаются в обычной последовательности.

При вызове **abort**() где-либо из программы деструкторы не вызываются даже для переменных глобальной области действия.

Деструктор может вызваться одним из двух способов: косвенно, через вызов **delete,** или непосредственно, путем задания полностью уточненного имени деструктора. **Delete** используется для уничтожения тех объектов, для которых память распределялась при помощи **new**. Явный вызов деструктора необходим только в случае объектов, которым при помощи вызова **new** динамически распределять память.

Примеры:

```
class X {
{...
~X();
...
};

void* operator new(size_t size, void *ptr)
{
return ptr;
}
char buffer[sizeof(X)];
```

```
main()
{
X* pointer = new X;  // память для
// объекта класса X выделена в
куче
X* exact_pointer;

exact_pointer =
(X)new(sizeif (X)&buffer) X;
// указатель инициализируется ад-
ресом // буфера
...
delete pointer;      // delete служит
// для разрушения указателя
exact_pointer->X::-X(); // прямой
вызов для отмены распределения
// памяти
}
```

Ход работы

1 Изучить теоретические сведения.
2 В соответствии с индивидуальным заданием для предыдущей лабораторной работы разработать конструкторы и деструктор для заданного класса. Осуществить инициализацию объектов класса различными конструкторами. Разместить объект класса в динамической памяти и разрушить его после использования в программе.
3 Набрать программу на компьютере и устранить ошибки.
4 Получить результат.
5 Оформить отчет.
6 Подготовиться к защите лабораторной работы, изучив контрольные вопросы по данной теме.

Требования к содержанию отчёта приведены в лабораторной работе №1.

Контрольные вопросы для подготовки и самостоятельной работы

1 Зачем используются конструкторы и деструкторы?
2 Какое имя имеет конструктор и деструктор?
3 Сколько конструкторов и деструкторов может быть в классе?
4 Можно ли выполнить инициализацию данных-членов без конструктора?
5 Назовите отличия конструкторов и деструкторов от других функций
6 Можно ли явно вызвать конструктор и деструктор?
7 Можно ли передать параметры в конструкторы, использовать параметры по умолчанию?
8 Как определить вызываемый конструктор, если их несколько?
9 Что такое конструктор по умолчанию, конструктор копии?
10 Приведите синтаксис объявления, определения и использования конструкторов, какие альтернативные варианты допустимы?
11 Объясните приведенные примеры.
12 Для чего необходимы оператора **new** и **delete.**
13 Когда вызываются деструкторы для локальных и глобальных переменных
14 Как происходит вызов деструкторов при выходе из программы, при вызове функций **exit(), abord()?**

Лабораторная работа №22

Использование наследования для создания иерархии классов (2 часа)

Цель работы: получить навыки в использовании наследования для создания производных классов при простом наследовании.

Теоретические сведения

При объявлении производного класса **D** перечисляются базовые классы **B1, B2** ... в разделяемом запятой **"базовом_списке"**. Синтаксис объявления производного класса.

class <Имя_класса_D>:<Базовые_классы> {<список_элементов>};

При объявлении класса **D** перед классами в базовом списке вы можете задать спецификатор доступа **public,** или **private**, или **protected**.

```
class D : public B1, private B2, ... {
...
};
```

Эти спецификаторы не изменяют доступа к элементам с точки зрения базового класса, но могут изменить доступ к элементам базовых классов из производных классов.

Если **D** представляет собой объявление класса (**class)**, то по умолчанию используется **private**, а если **D** - объявление структуры (**struct),** то **public**.

Класс **D** наследует все элементы базовых классов. (Переопределенные элементы базовых классов наследуются и при необходимости доступ к ним возможен при помощи переопределений области действия). **D** может использовать элементы базовых классов только с атрибутами **public** и **protected.**

Производный класс наследует атрибуты доступа базового класса следующим образом:

1 Если базовый класс указан в списке как **public**:

- элементы **public** базового класса становятся элементами **public** производного класса.
- элементы **protected** базового класса становятся элементами **protected** производного класса.
- элементы **private** базового класса остаются **private** для производного класса.

2 2. Если базовый класс **protected:**

- элементы базового класса общедоступные **(public)** и защищенные **(protected)** становятся защищенными **(protected)** элементами производного класса.
- элементы **private** базового класса остаются для произвольного класса **private.**

3 3. Если базовый класс **private:**

- общедоступные **(public)** и защищенные **(protected)** члены базового класса становятся частными **(private**) элементами производного класса.

– элементы **private** базового класса остаются для производного класса **private.**

Во всех рассмотренных случаях отметим, что элементы **private** базового класса были и остаются недоступными для функций-членов производного класса, пока в производном классе не использован оператор разрешения области видимости (::), пока в описании доступа базового класса не будут явно заданы объявления **friend**. Например:

```
class X : A {  // класс X является производным от класса A (простое
 // наследование), причём по умолчанию спецификатор - private A
…
}
class Y : B, public C {  // класс Y является производным (множественное
 // наследование) от B и C. По умолчанию - private B, спецификатор для C- public
}
struct S : D {   // struct S - производная от D, по умолчанию для структур
 // struct  - public D
..,
}
struct T : private D, E {  // struct T является производной (множественное
 // наследование) от  D и E. Спецификатор для D – private D. По умолчанию,
 // E  - public E
}
```

Действие спецификаторов доступа к элементам базовых классов при наследовании можно скорректировать при помощи оператора разрешения области видимости (::) в разделах **public** или **protected** для производного класса. Например:

```
class B {// базовый класс
int a;  // по умолчанию
 // private
public:
int b, c;
int Bfunc(void);
};
```

```
class X : private B {   // теперь члены базового класса
// B::a, b, c и B func() - стали private в классе
//наследнике X. Переменная a в X недоступна
int d;      // по умолчанию private
public:
B::c;          // переменная c была private; теперь она
public:
int e;
int Xfunc(void);
};
```

```
int Efunc(X& x);       // внешняя по отношению к классам B и X, требует
// ссылку на объект x класса X.
```

Функция **Efunc()** может использовать только имена членов класса **X** с атрибутом **public**, например **x.c, x.e** и **x.Xfunc**.

Функция **Xfunc()** в **X** является функцией-членом, поэтому она имеет доступ:

1 к собственным **private** и **public** элементам: **d, e** и **Xfunc().**
2 к старшим **private** элементам базового класса **B: b** и **Bfunc()**.
3 Однако, **Xfunc** не имеет доступа к **private** относительно **B** элементу **a**.

4 к переменной **c** класса **B** явно заданной **public** с помощью **(::) – B::c.**

Конструкторы базового класса должны объявляться с атрибутами **public** или **protected.** Причём: конструкторы производных классов обязательно вызывают конструкторы базовых классов.

Примеры:

```
class base1
{
int x;
public:
base1(int i) { x = i; }
};
```

```
class base2
{
int x;
public:
base2(int i) : x(i) {}
};
```

```
class top : public base1, public base2
{
int a, b;
public:
top(int i, int j) : base(i*5), base2(j+i), a(i) { b = j;}
};
```

В случае такой иерархии классов объявление **top one(1, 2)** приведет к инициализации base**1 ::x** значением **5**, а **base2 ::x** значением **3**. Методы инициализации могут комбинироваться друг с другом (чередоваться).

Базовые классы инициализируются нижнего с верхнего в иерархии в последовательности, указанной в объявлении производного класса при множественном наследовании. Инициализация элементов происходит в последовательности их объявления независимо от последовательности их расположения в списке инициализации.

```
class X
{
int a, b;
public:
X(int i, j) : a(i), b(a+j) {}
};
```

Объявление **X xobj (2.1)** приведет к присваиванию **xobj::a** числа **2** и **xobj::b** числа **3.**

Конструкторы базовых классов вызываются перед конструированием любых элементов производных классов. Значения данных-членов производного класса не могут изменяться и затем влиять на создание базового класса.

```
class base
{
int x;
public:
base(int i) : x(i) {}
};
```

```
class derived : base
{
int a;
public:
derived(int i) : a(i*10), base(a) {} // нельзя, т.к.
// конструктору base будет передано неинициализированное a
}
```

При внешнем определении конструктора **derived** он также вызывает конструктор базового класса

```
derived::derived(int i) : base(i)
{
...
```

}

"Друзья" классов (friend)

Если в объявлении или определении функции в пределах класса **X** используется спецификатор **friend**, то такая функция становится "другом" класса **X**.

"Друг" **F** класса **X** - это функция или класс, который, не являясь функцией-элементом **X,** имеет право доступа к элементам **X,** включая разделы **private** и **protected**. Во всех прочих отношениях **F()** - это обычная с точки зрения области действия, объявления и определения функция.

Поскольку функция **F()** не является элементом **X**, она не лежит в области действия **X** и поэтому для **X xobj, *xptr**; не может вызываться операциями выбора **xobj. F()** и **xptr->F** (где **xobj** - это объект класса **X**, а **xptr** - это указатель на класс **X**).

Дружественная функция, определенная в пределах класса (**inline**), подчиняется тем же правилам встраивания, что и функции-элементы класса. На дружественные функции не действуют спецификаторы доступа. Например:

Объявление класса	Определения для функций
class X { **int i;** // **private** **friend void friend_func(X*, int);** // **friend_func** не является **private**, // хотя она и объявлена в разделе // **private** **public:** **void member_func(int);** **};**	**void friend_func(X* xptr, int a) { xptr->i = a; }** // доступ к **private int i** открыт **void X::member_func(int a) { i = a; }** // в функцию-член не нужно // передавать указатель или ссылку на // класс, т.к. доступ к ней // осуществляется через объект класса // операцией выбора (.)

X xobj; // объявление объекта **xobj** класса **X**
Отметим различие в вызовах функций:
friend_func(&xobj, 6); // вызов без имени объекта как обычная функция
xobj.member_func(6); // вызов с именем объекта класса

Можно сделать все функции класса **Y** дружественными для класса **X** в одном объявлении:

```
class Y; // неполное объявление
class X {
friend Y; // класс Y является
// дружественным для X
int i;
void member_funcX();
};
```

```
class Y; { friend void X::member_funcX();
public:
void fr_X1(X&);
void fr_X2(X*);
...
};
```

Функции **fr_x1()** и **fr_x2(),** объявленные в **Y,** являются дружественными для **X**, хотя они и не имеют спецификаторов **friend**. Они имеют доступ к частным элементам **X (private),** таким как **i** и **member_funcX().** Кроме того, отдельные функции-члены класса **X** также могут быть дружественными для класса **Y**.

"Дружественность" классов не транзитивна: если **X** является дружественным для **Y**, а **Y** - дружественный для **Z**, это не означает, что **X** - дружественный **Z.** Однако, "дружественность" наследуется.

Ход работы

1 Изучить теоретические сведения.
2 В соответствии с индивидуальным заданием, разработать структуру базового класса и наследников (не менее 3-х производных классов на двух уровнях иерархии). Использовать конструкторы и деструкторы для инициализации данных и уничтожения объектов классов. Использовать замещающие функции-члены для работы с объектами классов.
3 Разработать алгоритм решения задачи и программу.
4 Набрать программу на компьютере и устранить ошибки.
5 Получить результат.
6 Оформить отчет.
7 Подготовиться к защите лабораторной работы, изучив контрольные вопросы по данной теме.

Требования к содержанию отчёта приведены в лабораторной работе №1.

Контрольные вопросы для подготовки и самостоятельной работы

1 Что означает оператор **(::)**?
2 Что означает понятие наследования?
3 Какой класс называется базовым?
4 Какой класс является наследником?
5 Сколько базовых классов может быть у производного класса?
6 Может производный класс быть базовым?
7 Можно ли задавать спецификаторы для базовых классов при наследовании (объявление произвольного класса)?
8 Как изменяется доступ к элементам базового класса при наследовании с различными спецификаторами доступа: из разделов класса, из программы, из других классов?
9 В чём разница между простым и множественным наследованием?
10 Что означает выражение «неполное объявление» класса?
11 Можно ли уточнить доступ к членам базового класса в производном классе? Как это осуществляется?
12 Какие функции называются друзьями класса?
13 Как объявляются и определяются функции–друзья класса?
14 Может ли класс быть дружественным?
15 Могут ли два класса быть друзьями друг другу?
16 Можно ли из класса-наследника получить доступ к **private** части базового класса, если спецификатор доступа при наследовании **private** ?

Таблица 22.1 - ***Варианты понятий для базовых классов***

Вариант	Понятие	Вариант	Понятие
1	Растения	15	Мебель
2	Животные	16	Строения
3	Небесные тела	17	Мосты
4	Спортивные соревнования	18	Бритвы
5	Печатная продукция	19	Принтеры
6	Промышленное производство	20	Плоттеры
7	Телефоны	21	Разъёмы электрические
8	Железнодорожно-транспортные средства	22	Манипуляторы для ввода информации
9	Автомобильный транспорт	23	Устройства записи информации
10	Осветительные приборы	24	Сканеры
11	Средства связи	25	ЭВМ
12	Телевизоры	26	Нагревательные устройства
13	Корабли	27	Устройства передачи крутящего момента
14	Мебель мягкая	28	

Лабораторная работа №23

Использование виртуальных и указателей для работы с объектами классов (2 часа)

Цель работы: изучить и научиться использовать виртуальные функции в языке **C.**

Теоретические сведения

Виртуальные функции-члены объявляются в классе с ключевым словом **virtual.**

Если базовый класс **(БК) base** содержит виртуальную функцию **(virtual) vf ()** и производный класс **(ПК) derived** также содержит эту функцию, то при вызове функции **vf()** для объекта базового класса мы получим **base:: vf(),** а для объекта производного класса мы получаем **derived::vf().** Например:

Базовый класс	Производный класс	
struct base **{…..** **virtual void vf (void);** **void f (void);** **};** **base_1** **class** **{…** **public:** **virtual void vf(void)=0** **void f(void);** **};**	**struct derived : public base** **{…..** **virtual void vf (void);** **// virtual** в последнем // **ПК** можно опустить **void f (void);** **};**	**derived d;** // объект производ- // ного класса **d.vf();** // вызов функции класса **d** **// erived::vf()** **d.f();** // вызов функции класса // **derived::f()** **base* bp = &d;** // указатель на // **БК** адресует объект **ПК** **bp->vf ();** // вызов виртуальной // функции **derived::vf()** **bp->f ();** // вызов функции-члена // класса **base** // **base:: f()**

Тип объектов классов с виртуальными функциями определяется во время выполнения программы.

Поэтому при вызовах с помощью указателя **bp** на **БК** функций **vf()** и **f()** для объекта **ПК** с именем **d** вызываются, соответственно, **derived::vf(),** но **base::vf(). В**ызов нужной виртуальной функции **vf()** зависит от типа объекта, для которого она вызывается **(derived d)**, в то время как вызов невиртуальной функции **f()** зависит только от типа указателя **(base*bp)**, адресующего данный объект.

Если производный класс содержит функцию с тем же именем, что и имя виртуальной функции в базовом классе, то они должны иметь один и тот же тип. Функция **vf()** в **ПК** от **БК**, в котором содержится виртуальная функция **vf(),** также считается виртуальной. Виртуальная функция может быть определена в базовом классе. Виртуальную функцию, которая уже определена в базовом классе, в производном классе можно не определять. В этом случае при использовании указателя на БК для адресации ПК при всех обращениях используется функция, определенная в базовом классе. Если виртуальная функция в классе заканчивается нулём **(=0),** то она называется чистой виртуальной функцией. Чистая виртуальная функ-

ция не имеет определения в базовом классе, но определяется в производных. Класс, содержащий хотя бы одну такую функцию, называется абстрактным. Нельзя создать объект абстрактного класса.

Виртуальные базовые классы

При множественном наследовании базовый класс не может задаваться в производном классе более одного раза. Однако, базовый класс можно передавать производному классу более одного раза косвенно:

```
class B { ... };
class D : B, B { ... }:
 // недопустимо
```

```
class X : public B { ... };
class Y : public B { ... };
class Z : public X, public Y { ... }; // допустимо
```

В данном случае каждый объект класса **Z** будет иметь два подобъекта класса **B**. Для устранения этой проблемы к спецификатору базового класса добавляют ключевое слово **virtual**. Например:

```
class X : virtual public B { ... }; // теперь B является виртуальным базовым
 классом
class Y : virtual public B { ... };
class Z : public X, public Y { ... } // класс Z имеет только один подобъект класса B.
```

Виртуальные деструкторы

Конструкторы не могут быть виртуальными. Деструктор может быть объявлен как виртуальный **(virtual).** Это позволяет указателю на базовый класс вызывать необходимый деструктор в случае, когда указатель ссылается на объект производного класса. Деструктор производного класса от базового класса с виртуальным деструктором является виртуальным.

```
class color
{...
public:
virtual ~color(); // виртуальный
// деструктор для класса color
};
```

```
class red : public color
{...
public:
virtual~red();    // деструктор для red
// также является виртуальным
};
```

```
class brightred : public red
{...
public:
virtual~brightred();      // деструктор для brightred также виртуальный
};
```

Рассмотрим работу с объектами объявленных классов. Указатель на базовый класс может адресовать объекты производных классов

```
Color *palette[3]; // объявление массива указателей на базовый класс
Palette[0] = new red; // создание объекта класса red в куче
palette[1] = new brightred; // создание объекта класса brightred в куче
```

palette[2] = new color; // создание объекта базового класса в куче

Применение оператора **delete**

delete palette[0]; // вызывается деструктор для объекта класса **red**

delete palette[1]; // деструктор для объекта класса **brightred**

delete palette[2]; // запуск деструктора для объекта класса **color**

Однако, если ни один из деструкторов не был объявлен виртуальным, то выражения **delete palette[0**], **delete palette[1]** и **delete palette[2]** вызывают только деструктор для базового класса **color,** на который объявлен массив указателей. Это приведет к неправильному уничтожению первых двух элементов, которые фактически имели тип **red и brightred**. Вызовы виртуальных деструкторов компонуются во время выполнения программы и объекты сами определяют, какой деструктор надо вызвать.

Ход работы

1 Изучить теоретические сведения.
2 В соответствии с индивидуальным заданием на базе лабораторной работы **№22** разработать алгоритм работы с объектами базового и производных классов с использованием указателей на базовые и производные классы. При необходимости довести иерархию классов до 3-4-х уровней.
3 Набрать программу на компьютере и устранить ошибки.
4 Получить результат.
5 Оформить отчет.
6 Подготовиться к защите лабораторной работы, изучив контрольные вопросы по данной теме.

Требования к содержанию отчёта приведены в лабораторной работе №1.

Контрольные вопросы для подготовки и самостоятельной работы

1 Какие функции-члены называются встроенными **(inline)?**
2 Какие функции-члены называется перегруженными?
3 Какие функции-члены называются замещающими?
4 Какие функции- члены называется виртуальными?
5 Можно ли адресовать объекты ПК с помощью указателей на общий БК, на предыдущий БК по иерархии?
6 Назовите правила использования указателей для работы с объектами БК и ПК**.**
7 Когда необходимо определение виртуальной функции в базовом классе?
8 Какой класс называется абстрактным?
9 Можно ли создать объект абстрактного класса?
10 Какая функция называется чисто виртуальной?

11 Можно ли установить в процессе компиляции какая функция будет вызываться при использовании указателей для работы с объектами?
12 Могут ли конструкторы и деструкторы быть виртуальными? Чем это вызвано?
13 Как производится размещение объектов классов в "куче"?
14 Как производится выделение и освобождение памяти для динамически создаваемых объектов?
15 Когда производится нахождение виртуальной функции, которую необходимо вызвать для заданного объекта ПК, если для адресации объекта ПК используется указатель на БК.?
16 Какая невиртуальная функция будет вызвана в указанном случае и почему?
17 Для чего или как объявляются виртуальные классы?

Литература

1 Керниган Б., Ритчи Д Язык программирования Си. – 2-изд. –М.: Финансы и статистика, 1992 – 272 с.
2 Страуструп Б. Язык программирования С++. –М.: Радио и связь.1991 – 352с.
3 Прокофьев Б.П., Сухарев Н.Н. и др. Графические средства Turbo С++. М.:ФИС. 19992 – 160 с.
4 Романовская Л.М., Русс Т.В., Святковский С.Г. Программирование в среде Си для ПЭВМ ЕС М.: ФИС, 1992 – 352с.
5 Берри Р., Микин Б. Языки Си. Введение для программирования. –1998 – 198с.
6 Шилдт Г. Язык "Си" для профессионалов. –М.: ИВК-СОФТ. 1992 – 319 с.
7 Ален И. Голуб. С и Си++ . Правила программирования. / Под ред. В. Костенко. М : БИНОМ. 1996 – 272 с.
8 Бочков С.О., Субботин Д.М. Язык программирования Си для персонального компьютера. / Под ред. А.И.Садчикова – Диалог. Радио и связь, 1990 – 384с.
9 Болски М.И. Языки программирования Си: Справочник. Перевод с английского. Радио и связь, 1988 – 96с.
10 Лукас П. С++ под рукой: Справочник. _ К: ДиаСофт, 1993 – 176 с.
11 Проценко В.С., Чаленко И.П., Ставровский А.Б. Техніка програмування мовою Сі.- К: Либідь, 1993 – 224с.
12 Пол Ирэ. Объектно–ориентированное программирование с использованием С++. – К.: НИПФ "ДиаСофт Лтд." 1995. – 480с.
13 Гради Буч. Объектно–ориентированное проектирование. – К.: Диалектика . ИВК (Москва), 1992 – 519с.
14 Сван Т. Основание Borland С++ 4.5. Практический курс, в 2-х томах. 2 – изд.К.: Диалектика, 1996 – 544с.
15 Стивен Поттс, Т.С. Монк. Borland С++ в примерах. / Минск: Попури, 1996 – 752с.

Содержание

Стр.