

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ»
Институт компьютерных технологий и защиты информации

Индекс по учебному плану): Б1.О.12.02

Специальность: 10.05.02 Информационная безопасность телекоммуникационных
систем

Специализация: "Разработка защищенных телекоммуникационных систем"

Методические указания к лабораторным работам
по дисциплине

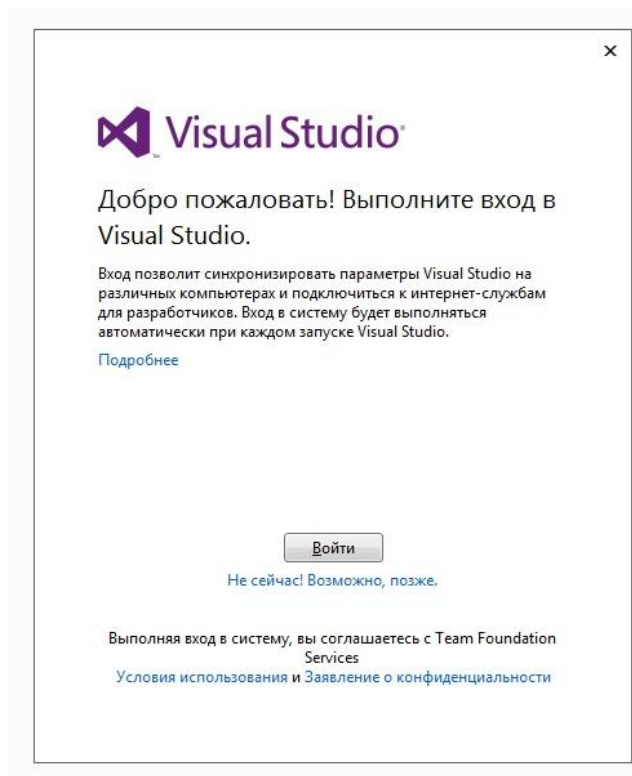
ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ

РАБОТА В СРЕДЕ *MICROSOFT VISUAL STUDIO*

Microsoft Visual Studio - это интегрированная среда разработки программного обеспечения. *Visual Studio* содержит редактор исходного текста программы и встроенный отладчик.

Начало работы

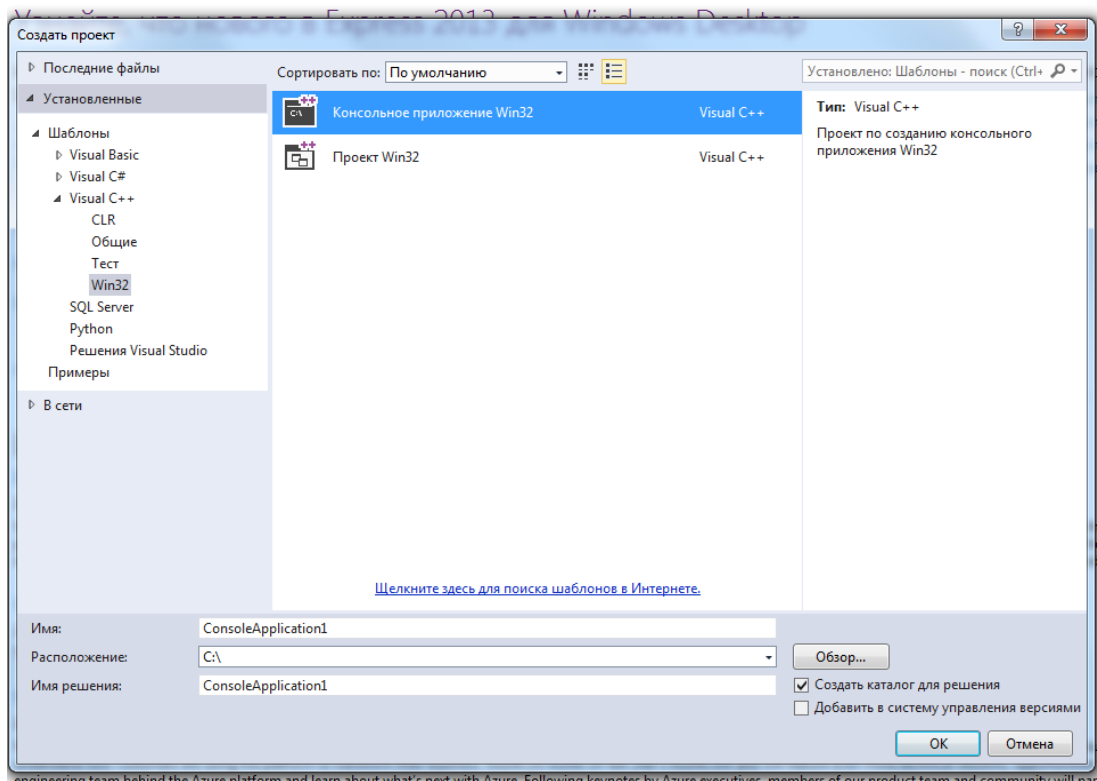
1. Нажать кнопку **«Пуск»/Все программы/Visual Studio**. Откроется окно.



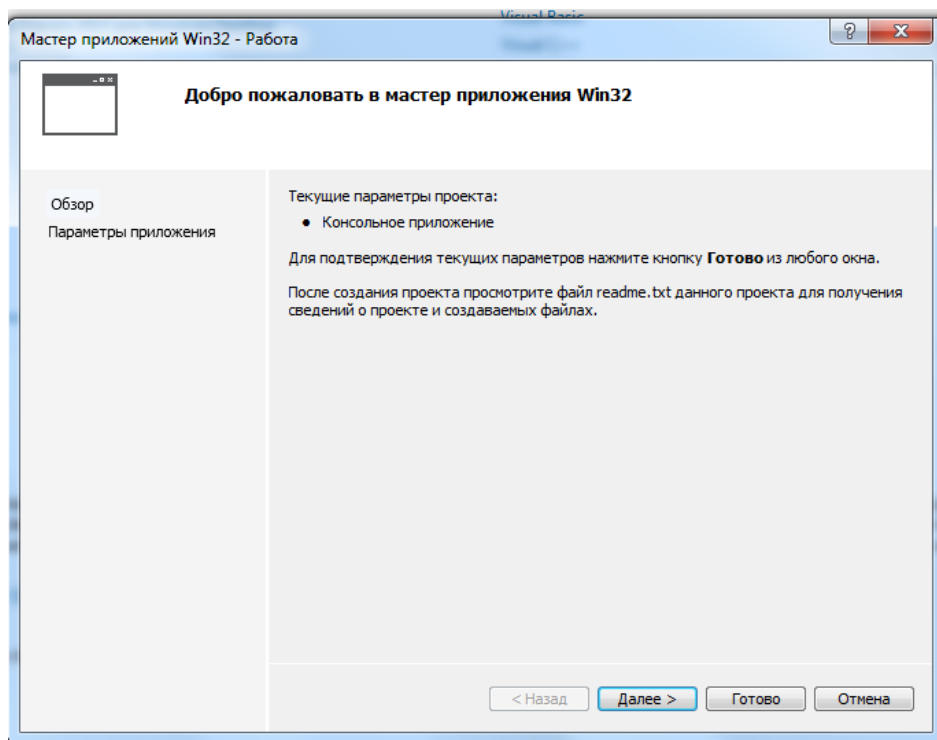
Нажать **Войти**.

2. В окне **Начальная страница** выбрать **Создать проект**. Проект включает в себя файлы исходных текстов программы и файлы, которые необходимы для создания программы.

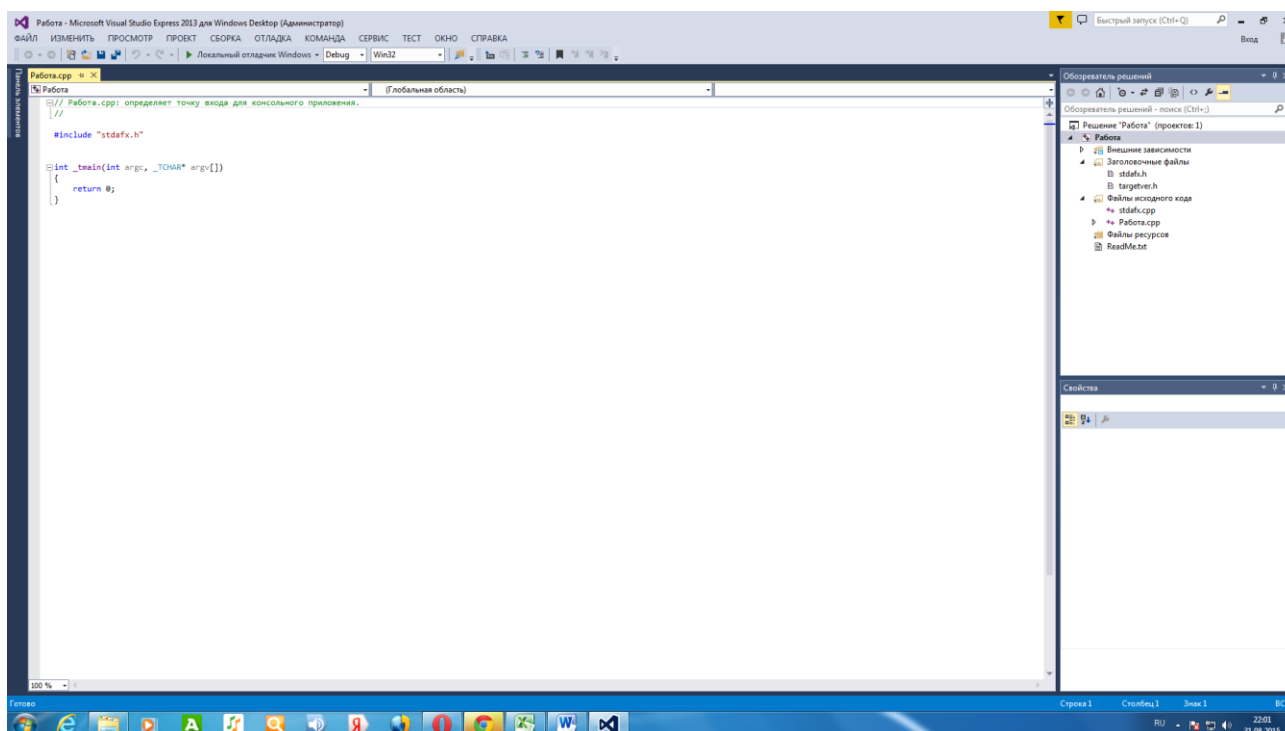
3. В диалоговом окне **Создать проект** выбрать **Шаблоны/Visual C++** и рядом в поле выбрать **Консольное приложение Windows**.



4. В поле **Имя** ввести имя файла. Например, *Работа*.
5. В поле **Расположение** указать расположение файла. Для этого при помощи кнопки **Обзор** выбрать нужную папку. Например: *C:\Stud\4104*.
6. Нажать **ОК**.
7. Откроется диалоговое окно **Мастер приложений Win32**. Нажать **Готово**.



8. Откроется окно, содержащее заголовок, строку меню, панель инструментов и несколько вкладок, в том числе вкладку файла исходного текста программы *Работа*.



Ввод исходного текста программы.

Текст программы вводится в окне файла исходного текста программы *Работа*. В окне уже содержится шаблонный текст следующего вида:

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    return 0;
}
```

Операторы программы вводятся между фигурными скобками. Допускается указывать несколько операторов в одной строке, но лучше размещать каждый оператор в отдельной строке. При вводе операторов следует придерживаться определенного стиля, например, закрывающую фигурную скобку располагать под открывающей. Операторы следует располагать с отступами от левой границы, соответствующими их уровням вложенности. Чем больше отступ, тем больше уровень

вложенности. Операторы, имеющие одинаковые уровни вложенности, лучше размещать в одной вертикальной позиции.

Например:

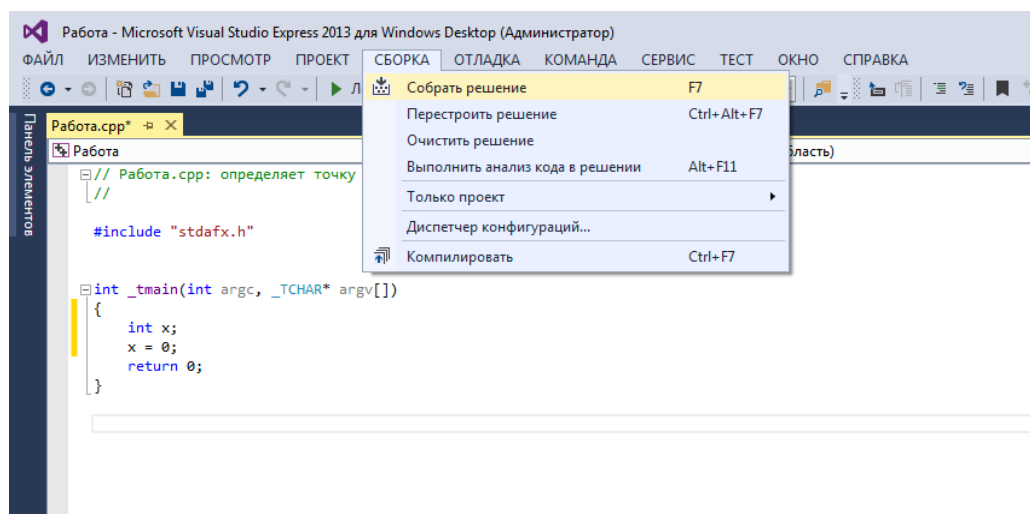
```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    int x;
    x = 0;
    return 0;
}
```

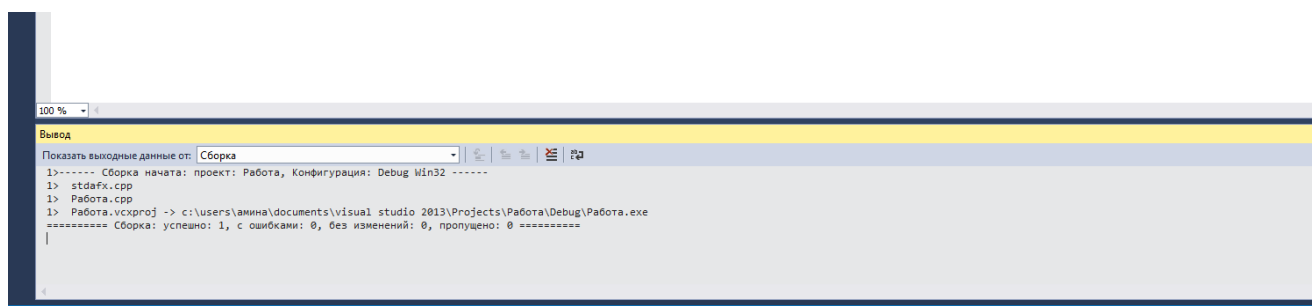
После ввода исходного текста программы необходимо перевести его на машинный язык, то есть выполнить компиляцию программы.

Компиляция программы

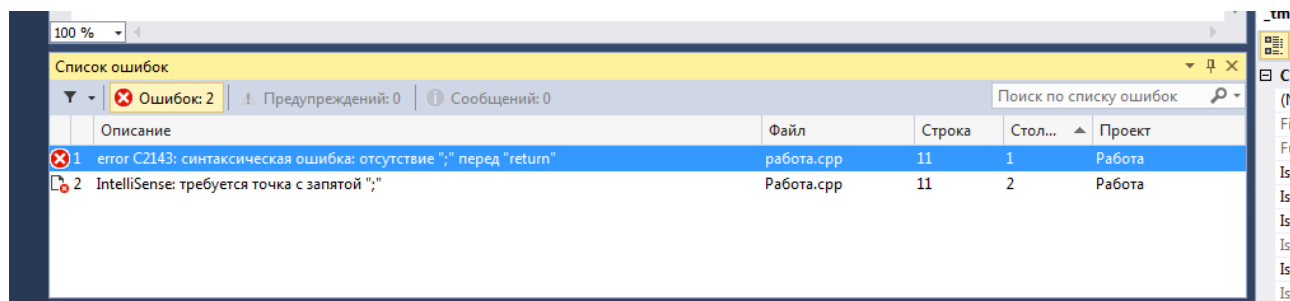
1. Дать команду *Сборка/Собрать решение* или нажать клавишу **F7**.



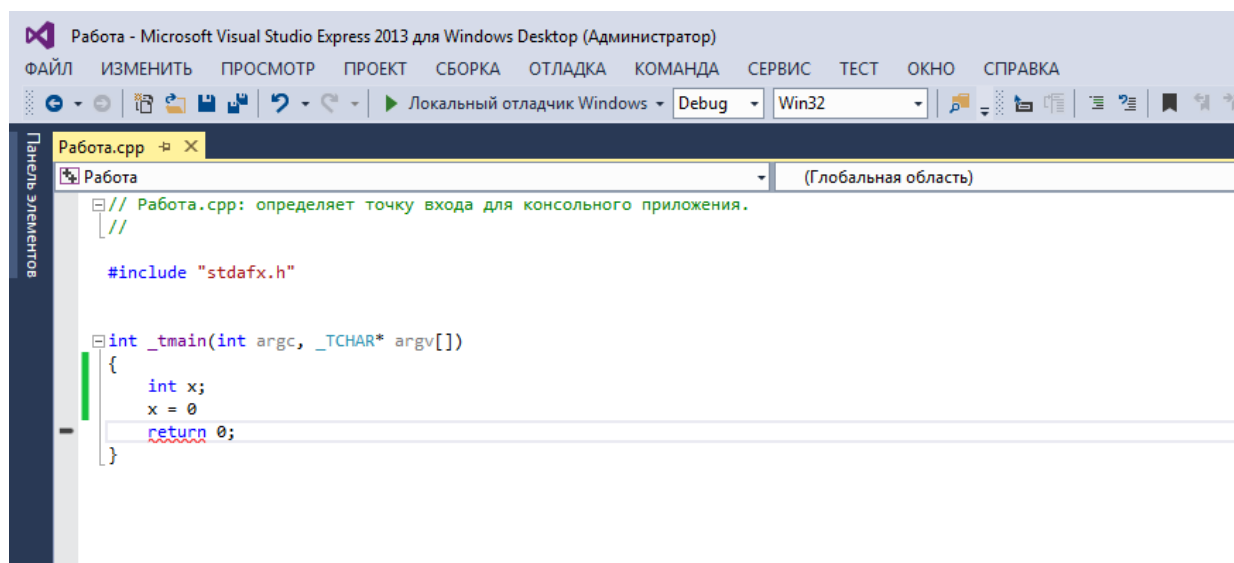
2. Если компиляция завершится успешно, то в нижней части экрана появится вкладка **Вывод** с сообщением вида:



3. Если в ходе компиляции будут обнаружены ошибки, то в нижней части экрана появится вкладка **Список ошибок** с сообщениями об ошибках. Каждое сообщение содержит номер, описание ошибки и номер строк и столбца, где была обнаружена ошибка.

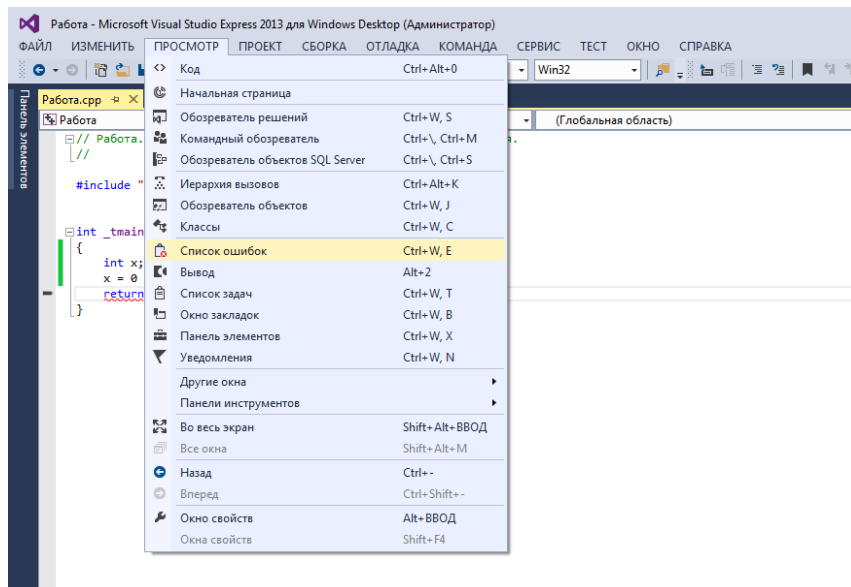


4. Если дважды щелкнуть по сообщению левой клавишей мыши, то в окне исходного текста текстовый курсор будет установлен в точке, где была обнаружена данная ошибка. Это необязательно та позиция в тексте программы, где эта ошибка была допущена.



5. Исправить все ошибки и повторить пункт 1.

Если окно с ошибками не появилось на экране, его можно вызвать, дав команду **Просмотр/Список ошибок** или **Просмотр/Другие окна/Список ошибок**.

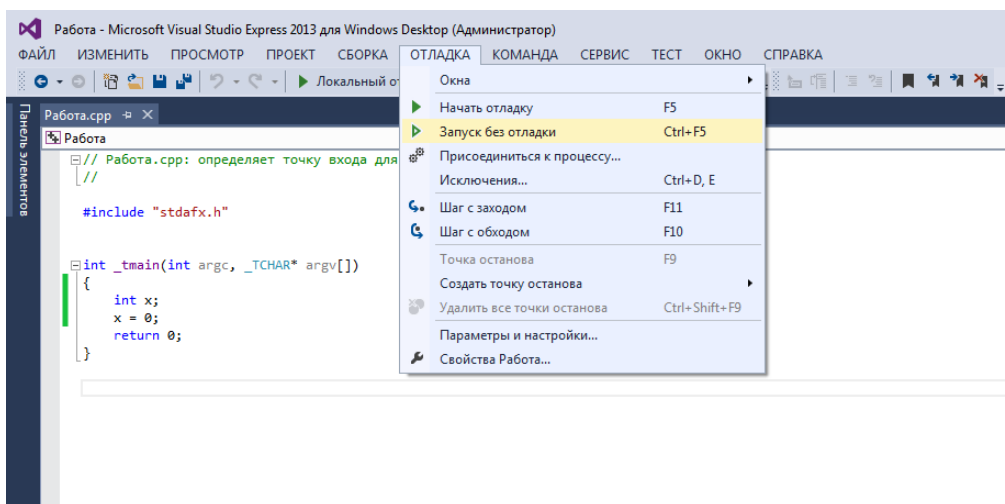


Чтобы зафиксировать положение окна в нижней части экрана, надо установить курсор на заголовке окна, щелкнуть правой клавишей мыши и в контекстном меню выбрать **Закрепить**.

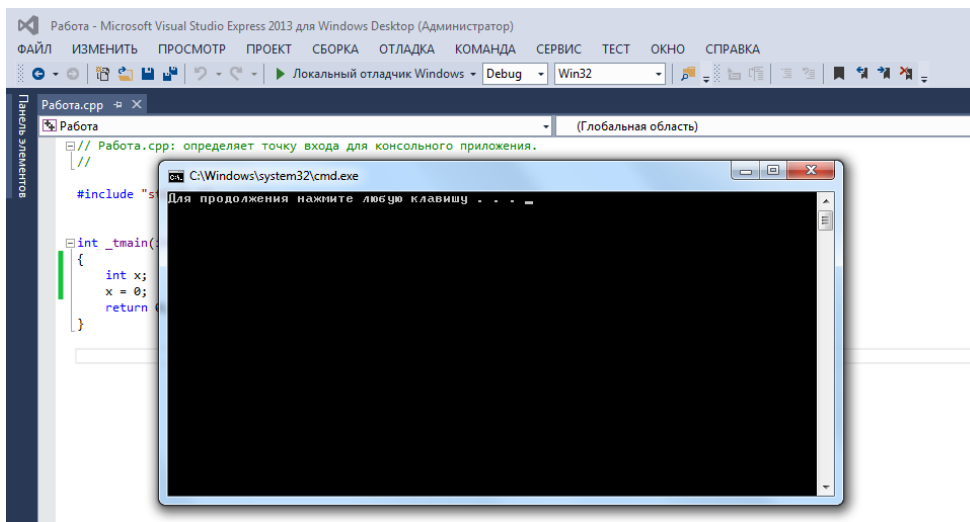
После компиляции можно приступить к выполнению программы. Если вы уверены в правильности программы, то можно выполнить ее без отладки.

Выполнение программы без отладки

1. Дать команду **Отладка/без отладки**.



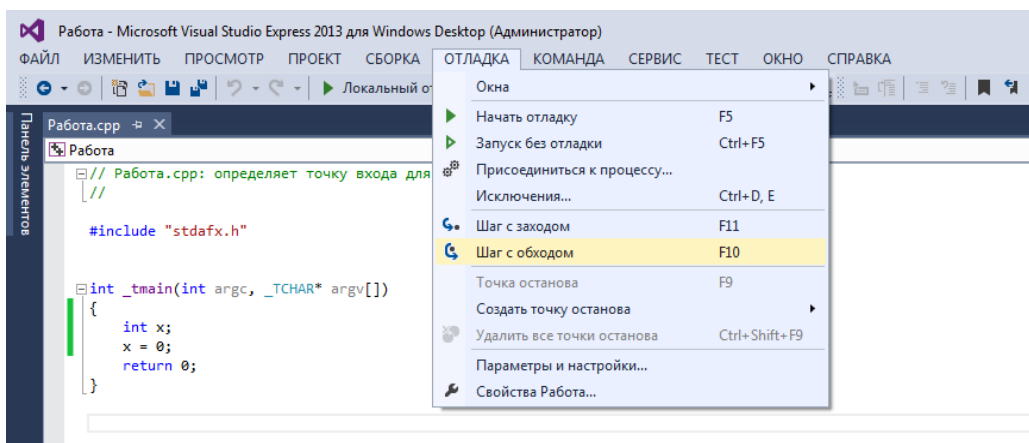
2. Откроется окно консоли. В этом окне будет выполнен вывод информации. В конце появится текст «Для продолжения нажмите любую клавишу». После нажатия клавиши окно консоли будет закрыто.



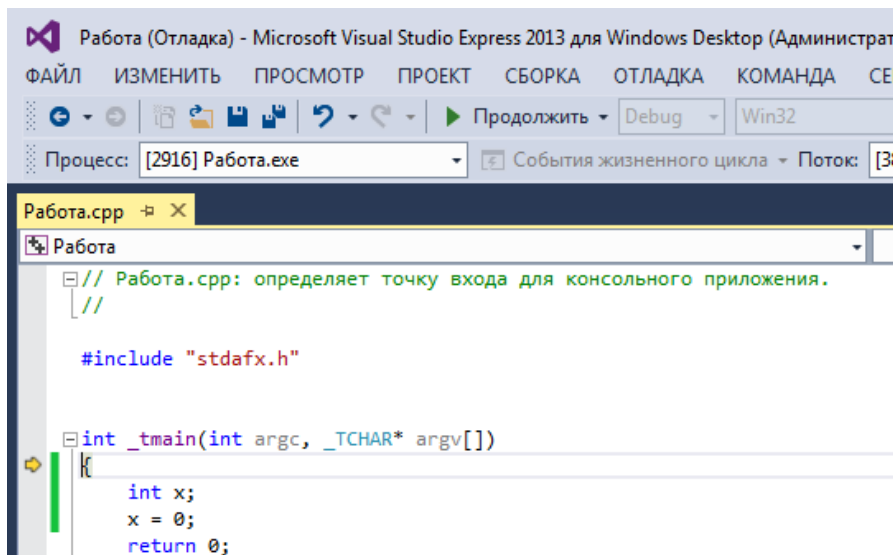
Любая сколько-нибудь сложная программа, как правило, содержит ошибки, которые могут быть выявлены только в ходе ее отладки.

Пошаговое выполнение программы

1. Дать команду *Отладка/с обходом* или нажать клавишу **F10**.

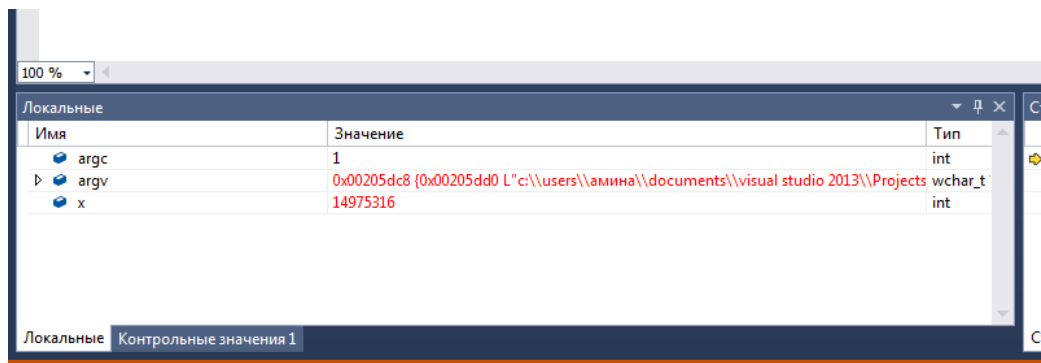


2. Перед первым оператором программы появится желтая стрелка. Эта стрелка указывает на оператор, который будет выполнен при нажатии клавиши **F10**.

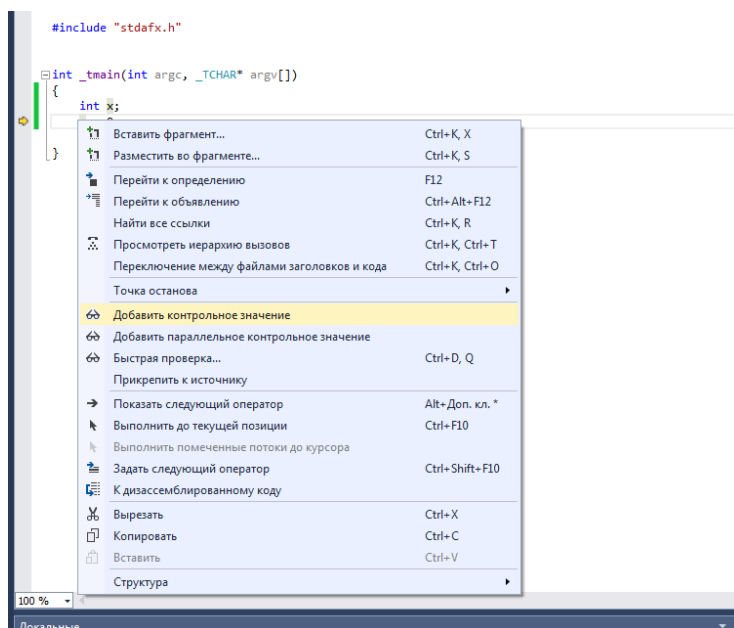


Так, нажимая клавишу **F10**, можно шаг за шагом выполнить всю программу.

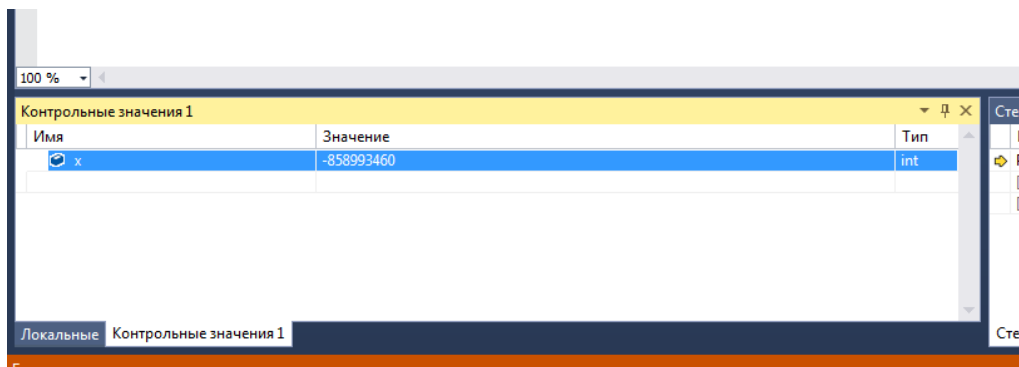
3. В нижней части экрана появится вкладка **Локальные переменные**. В этой вкладке отображаются текущие значения переменных, которые видимы в данный момент.



4. При желании можно сформировать свой список из имен интересующих вас переменных. Для этого надо подвести курсор к имени переменной, нажать правую клавишу мыши и выбрать **Добавить контрольное значение**.

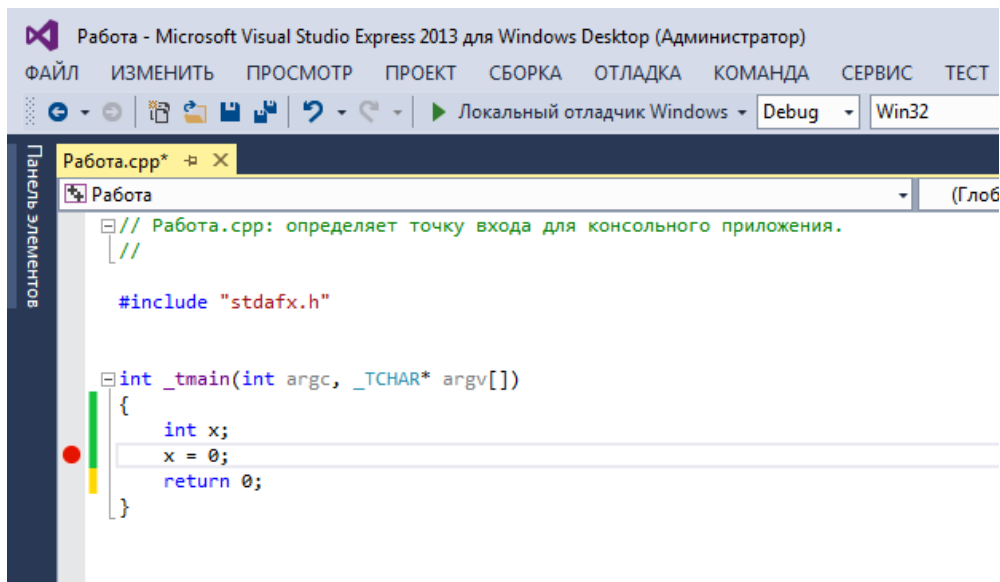


5. Появится новая вкладка **Контрольные значения**.

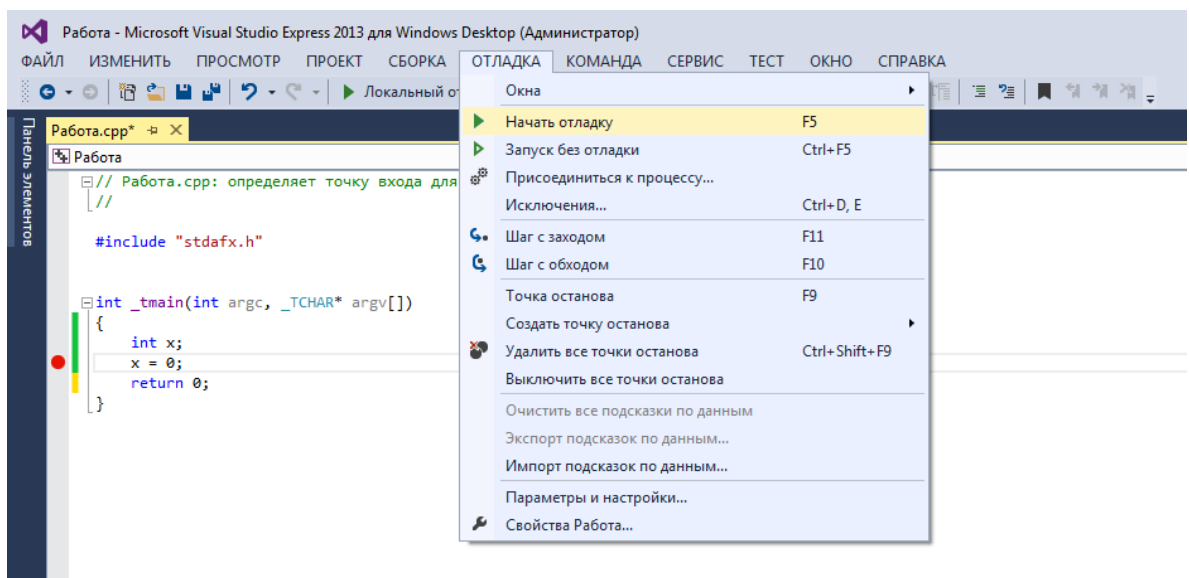


Точки останова

1. Если требуется выполнять программу с остановками в определенных местах, то надо устанавливать точки останова перед соответствующими операторами. Для этого надо щелкнуть мышью перед оператором в поле, выделенном серым цветом. Если точка останова установлена, то перед оператором появится красная точка. Щелчком по красной точке можно отменить точку останова.

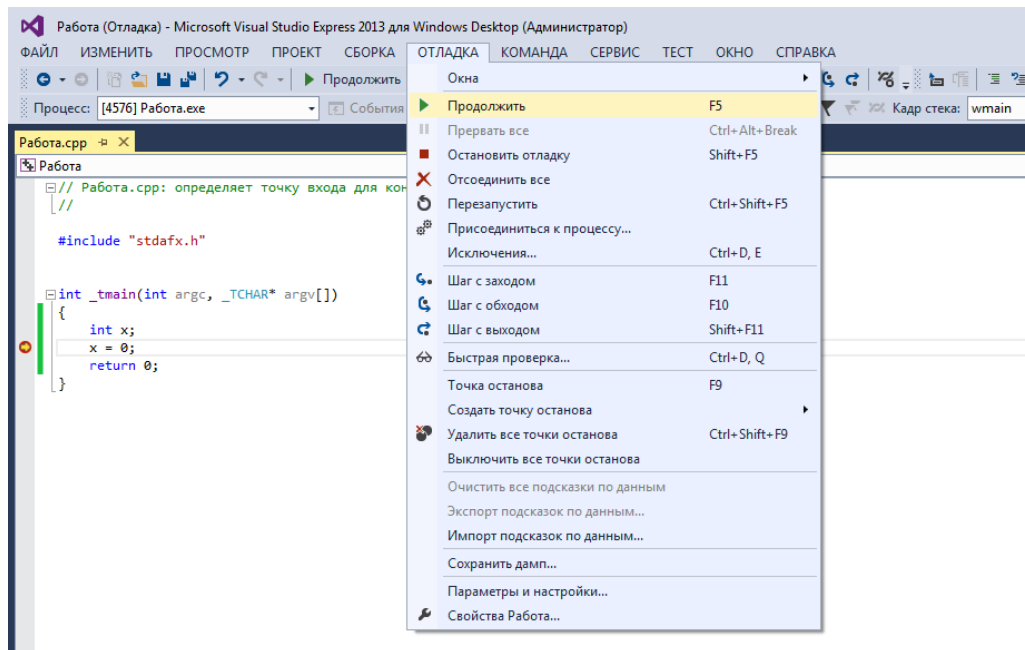


2. Дать команду *Отладка/Начать отладку*. Выполнение программы будет остановлено перед оператором, помеченным красной точкой.

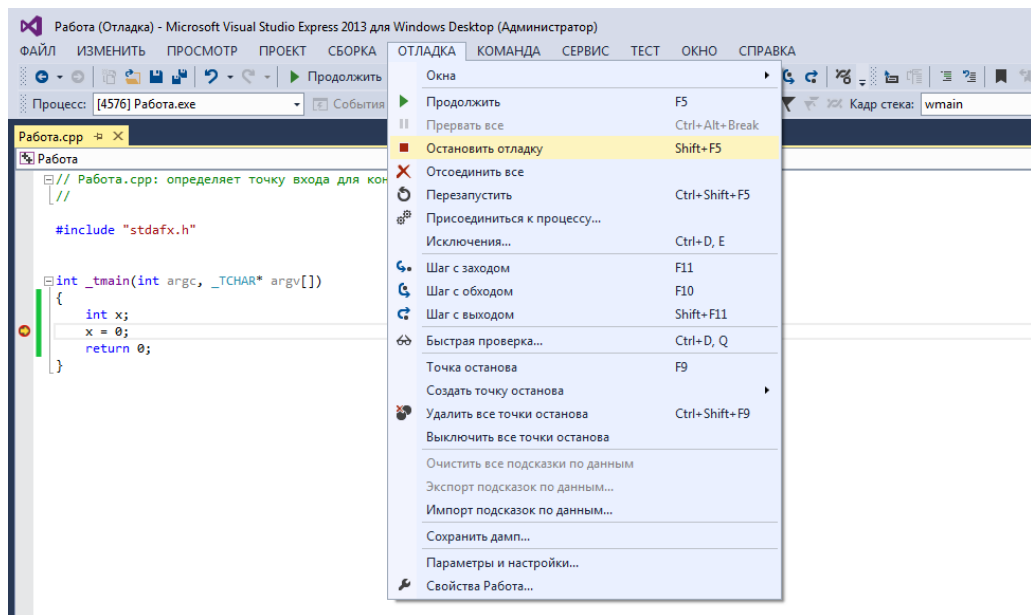


3. Для пошагового выполнения программы нажимать клавишу **F10**. Для продолжения выполнения программы дать команду *Отладка/Продолжить*. Вы-

полнение программы будет продолжено до следующей точки останова или до конца программы.



4. Чтобы прекратить отладку, надо дать команду **Отладка/Остановить отладку**.



ЛАБОРАТОРНАЯ РАБОТА № 1

ТИПЫ ДАННЫХ. ОПЕРАТОР ВЫРАЖЕНИЕ. УСЛОВНЫЙ ОПЕРАТОР. ЛИНЕЙНЫЕ АЛГОРИТМЫ. АЛГОРИТМЫ С РАЗВЕТВЛЕНИЯМИ

Цель работы. Создание линейных программ и программ с разветвлениями.

1. Теоретическая часть

Алфавит языка C

Алфавит – это символы, которые применяются при написании программ на языке C. К ним относятся

- латинские буквы прописные и строчные и знак подчеркивания,
 - арабские цифры от 0 до 9,
 - специальные знаки
- “ “ { } [] () < > = + - / * ‘ : ; . , \ ? ! & # ~ ^ % .

Идентификаторы

Идентификатор – это имя переменной, константы, типа, функции и т.п. Идентификаторы состоят из латинских букв, цифр и знаков подчеркивания. Прописные и строчные буквы различаются. Первый символ – буква или знак подчеркивания. Пробелы внутри идентификатора не допускаются. Например:

AI, Kol_vo, x - правильные идентификаторы.

23a - неправильный идентификатор (начинается с цифры).

a.b - неправильный идентификатор (содержит точку).

сумма - неправильный идентификатор (не латинские буквы)

Константы

Константы – это неизменяемые величины. Они бывают целые, вещественные, символьные и строковые. Константы целого типа бывают десятичные, восьмеричные и шестнадцатеричные.

Десятичные целые константы состоят из цифр. Первая цифра не должна быть нулем. Например:

24 1000 -56

Вещественные константы могут быть записаны двумя способами: в форме с десятичной точкой и в экспоненциальной форме. Например:

Запись в форме с десятичной точкой	Значение	Запись в экспоненциальной форме	Значение
-4.78	4, 78	10.3e-6	$10,3 \cdot 10^{-6}$
.89	0,89	-.13E25	$-0,13 \cdot 10^{25}$
56.05	56,05	1E+12	$1 \cdot 10^{12}$
12.	12,0	5.e18	$5 \cdot 10^{18}$

Символьная константа – это один или два символа, заключенные в апострофы. Например: 'A', 'F', 'o', 'n'.

Последовательности, которые начинаются с обратной косой черты, «\» называются **Esc-последовательностями**. Это управляющие последовательности. Они интерпретируются как один символ.

	наименование
\n	перевод строки
\t	горизонтальная табуляция
\'	апостроф
\"	кавычка
\?	? вопросительный знак

Строковая константа – это последовательность символов, заключенная в кавычки. Например:

“Это лекция” , “\t y=\xF5\n” .

Комментарий

Начинается с 2-х символов «косая черта» и заканчивается символом перехода на новую строку или заключается между символами /* и */

Например:

// это комментарий, который располагается на одной строке

```
/* это комментарий,  
который располагается  
на трех строчках */
```

Структура программы

Программа состоит из директив препроцессора, описаний и функций. Директивы препроцессора используются для присоединения специальных файлов, содержащих описания используемых в программе функций. Например:

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

Это директива, при помощи которой присоединяются функции для ввода-вывода информации.

Если программа состоит из одной функции, то эта функция должна иметь имя ***main***. Формат функции ***main***:

```
void main ( ) {  
операторы, составляющие тело функции  
}
```

void – это тип, используемый для описания функций, которые не возвращают значения.

Операторы

Операторы делятся на исполняемые и неисполняемые. Исполняемые операторы задают действия над данными. Неисполняемые операторы – это операторы описания данных (переменных). Каждый оператор заканчивается знаком « ; » (точкой с запятой).

Оператор описания переменной

Переменная - это поименованная область памяти, в которой хранятся данные определенного типа. Любая переменная должна быть описана перед ее использованием.

Формат оператора описания переменной:

```
[класс памяти] [const] тип имя [инициализатор];
```

Тип определяет внутреннее представление данных в памяти, множество значений, которые могут принимать данные этого типа, а также операции и функции, которые можно применять к величинам этого типа. Основные типы данных приводятся в таблице 1.

Таблица 1.

Тип	Диапазон		Размер в байтах	Назначение
	от	до		
<i>signed char</i>	- 128	+127	1	Для представления символов или целых чисел
<i>unsigned char</i>	0	255	1	
<i>signed short int</i>	- 32768	32767	2	целое со знаком
<i>unsigned short int</i>	0	65535	2	целое без знака
<i>signed long int</i>	- 2 147 483 848	2 147 483 848	4	целое со знаком
<i>unsigned long int</i>	0	4 294 967 295	4	целое без знака
<i>float</i>	3.4e-38	3.4e+38	4	вещественное одинарной точности
<i>double</i>	1.7e-308	1.7e+308	8	вещественное удвоенной точности.
<i>long double</i>	3.4e-4932	1.1e+4932	10	вещественное максимальной точности.

Инициализатор – позволяет присвоить переменной начальное значение при ее описании. Например:

float x = 7.453;

const – модификатор, который указывает, что переменную нельзя изменять. Это константа. Константа обязательно должна быть инициализирована при описании. Например:

```
const int h = 100;
```

В одном операторе можно описать несколько переменных одного типа (разделяя их запятыми). Например:

```
float x1,y1,x2,y2;
```

Выражения

Выражение состоит из операндов и знаков операции и используется для вычисления значения определенного типа. Каждый операнд может быть выражением, или в частном случае – константой или переменной.

Операции

Операции задают действия над операндами. Операции бывают унарные (над одним операндом, бинарные – с двумя операндами, тернарная – с тремя операндами). Знак операции состоит из одного или двух символов. Пробелы между символами в знаке операции не допускаются. Список операций приводится в таблице 2.

Таблица 2.

Приоритет	Действие
Унарные операции	
2	++ увеличить на 1
	-- уменьшить на 1
2	~ поразрядное отрицание
	! логическое отрицание
	- арифметическое отрицание
2	+ унарный плюс
2	& взятие адреса
2	* разыменование
2	(type) преобразование типа
2	sizeof размер
Бинарные и тернарные операции	
3	* умножение

	/	деление
	%	остаток от деления
4	+	сложение
	-	вычитание
5	<<	сдвиг влево
	>>	сдвиг вправо
6	<	меньше
	<=	меньше или равно
	>	больше
	>=	больше или равно
7	= =	равно
	! =	не равно
8	&	поразрядная конъюнкция И
9	^	поразрядное исключающее ИЛИ
10		поразрядная дизъюнкция
11	&&	логическое И
12		логическое ИЛИ
13	? :	условная операция (тернарная)
14	=	присваивание
	* =	умножение с присваиванием
	/ =	деление с присваиванием
	% =	остаток от деления с присваиванием
	+ =	сложение с присваиванием
	- =	вычитание с присваиванием
	<< =	сдвиг влево с присваиванием
	>> =	сдвиг вправо с присваиванием
	& =	поразрядное И с присваиванием
	=	поразрядное ИЛИ с присваиванием
	^ =	поразрядное искл. ИЛИ с присваиванием

15	, последовательное вычисление

Если операции одного приоритета, то справа налево вычисляются унарные операции, условная операция и операция присваивания. Слева направо вычисляются все остальные. Для изменения порядка вычислений используют круглые скобки.

Операции инкремент и декремент имеют две формы записи: префиксную и постфиксную. При префиксной форме операнд изменяется, новое значение является результатом выражения. При постфиксной форме результатом выражения является исходное значение операнда, после вычисления выражения операнд изменяется. Например:

$b=1;$

$a=++b; \quad // \quad a=2, b=2$

$c=b++; \quad // c=2, b=3$

При логическом отрицании операнды, равные 0, рассматриваются как ложь, а не равные 0 – как истина. Например:

$x=15;$

$y=!x; \quad // y=0$

Логические операции выполняются в соответствии с их таблицами истинности. Операнды, равные 0, рассматриваются как ложь, а не равные 0 – как истина.

Логическое И

&&	0	1
0	0	0
1	0	1

Результат выражения « истина », если истинны все входящие в него отношения

Например:

$x>20 \&\& x<50$ – это выражение истинно, если x находится в интервале от 20 до 50.

Логическое
ИЛИ

	0	1
0	0	1
1	1	1

Результат выражения « истина », если истинно хотя бы одно входящее в него выражение

Например:

$x < 20 \parallel x > 50$ – это выражение истинно, если x не входит в интервал от 20 до 50.

Операции деления. Операция `%` (остаток от деления) применима только к целым операндам. Результатом деления целых чисел при помощи операции `/` является целое число.

Исполняемые операторы

Исполняемые операторы задают действия над данными. Операторы могут быть объединены в составной оператор, или блок. Операторы, составляющие блок, заключаются в фигурные скобки. В этом случае они рассматриваются как один оператор.

Линейные алгоритмы

Это алгоритмы, в которых действия выполняются последовательно одно за другим от начала до конца. Обычно это ввод-вывод данных и присваивание каких-либо значений переменным.

Оператор «выражение»

Любое выражение, которое заканчивается точкой с запятой, рассматривается как оператор, выполнение которого заключается в вычислении выражения. Например:

`x=10;`

`a++;`

`y=a==b;`

`c==d` // не оператор : нет точки с запятой

Пример:

Дано двухзначное целое число x . Написать программу, которая вычисляет сумму цифр этого числа.

Решение

Для решения задачи нам понадобятся 4 переменные целого типа:

x – исходное число,

d – количество десятков,

e – количество единиц.

s – сумма цифр.

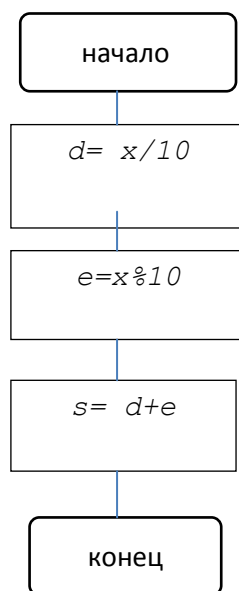
Двухзначное число x можно представить в виде

$$x = d * 10 + e * 1,$$

Таким образом, чтобы определить цифры числа надо поделить число x на 10.

Частное будет равно количеству десятков (так как при делении целого на целое получается результат целого типа), а остаток от деления – количеству единиц.

Блок-схема



Программа

```
void main()      //заголовок функции
{
    //неисполняемые операторы
    int x; // объявление переменной целого типа
    int s,  // сумма
        d, //количество десятков
        e; // количество единиц

    //исполняемые операторы «выражение»
    d= x/10;
```

```

    e=x%10;
    s= d+e;
}

```

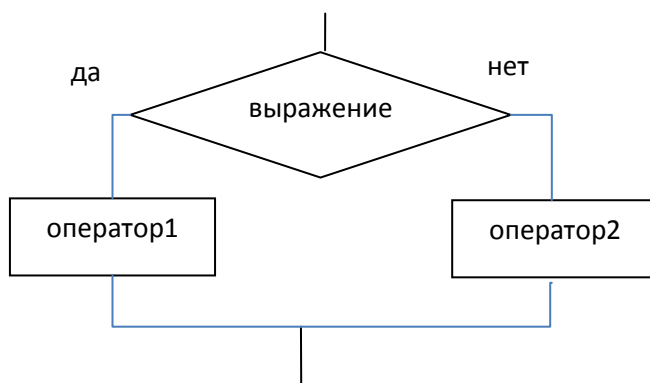
Алгоритмы с разветвлениями

Линейные программы встречаются редко. Обычно в процессе решения задачи приходится проверять различные условия, от которых зависит ход решения и результат. При анализе условий происходит разветвление процесса решения на два или несколько направлений. Для реализации ветвлений в языке С предусмотрены условный оператор и оператор переключатель.

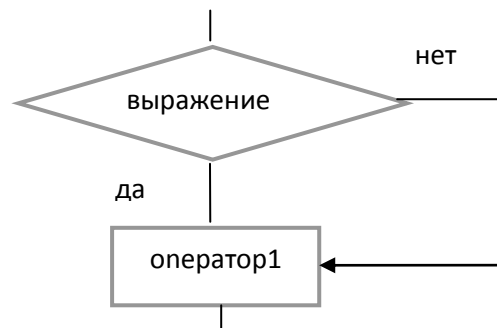
Условный оператор *if*

Этот оператор используется для разветвления процесса вычислений на два направления. Формат оператора:

```
if ( выражение) оператор1; [else оператор2;]
```



Если *else* опущено, то



Выражение должно иметь арифметический тип. Если оно не равно нулю, то выполняется оператор1, иначе - оператор2. После этого управление передается следующему оператору. Одна из ветвей может отсутствовать. Если в какой-либо ветви требуется выполнить несколько операторов, их необходимо заключить в фигурные скобки.

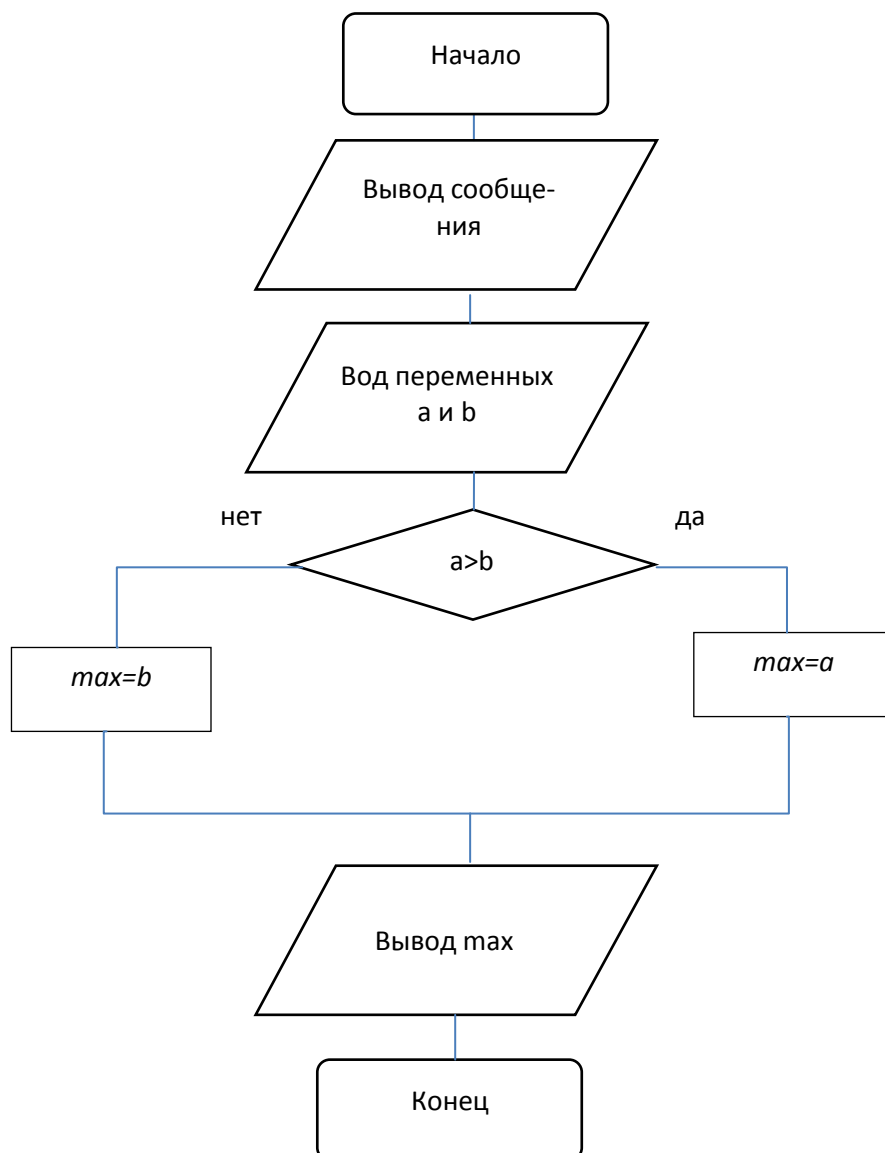
Пример:

Даны два целых числа a и b . Присвоить переменной max значение наибольшего из них.

Решение

Нам потребуются три переменные: max – для максимального значения, a – для первого числа и b – для второго числа. Вначале вводим с клавиатуры значения чисел a и b . Затем сравниваем a с b и, если a больше b , то присваиваем переменной max значение a , иначе – значение b .

Блок-схема



Программа

```
#include <stdio.h> // директива, присоединяющая функции для
```

ВВОДА-ВЫВОДА

```
int main() {  
    int a,b, max; // объявление переменных целого типа  
    printf("\n Введите a и b: "); //вывод сообщения на экран  
    scanf("%d%d",&a,&b); //ввод с клавиатуры значений a и b  
    if (a>b) max = a;      //если a>b, то max = a  
    else max=b;           // иначе max=b  
    printf("\nmax=%d ",max); //вывод на экран значения max  
}
```

Оператор переключатель *switch*

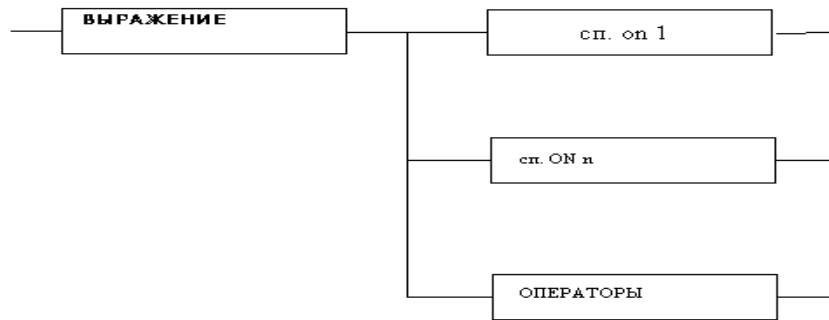
Оператор *switch* (переключатель) предназначен для разветвления процесса вычислений на несколько направлений. Формат оператора:

```
switch ( выражение ){  
    case константное_выражение_1: [список_операторов_1]  
    case константное_выражение_2: [список_операторов_2]  
    case константное_выражение_п: [список_операторов_п]  
    [default: операторы]  
}
```

Выражение должно быть целочисленным. Все константные выражения должны иметь разные значения, но быть одного и того же целочисленного типа.

Если значение выражения совпадает с одним из константных выражений, то управление передается первому оператору из соответствующего списка, а затем последовательно выполняются все остальные ветви. Иначе выполняются операторы, расположенные после слова *default*. Если ветвь *default* опущена, то управление передается оператору, следующему за *switch*.

Выход из переключателя обычно выполняется с помощью операторов *break*. Этот оператор передает управление оператору, следующему за оператором *switch*.



Пример.

Написать программу, которая выполняет над двумя целыми числами заданную арифметическую операцию.

Программа

```

#include <stdio.h>
void main() {
    int a,b, // аргументы
        res; // результат
    char op; // переменная символьного типа – знак операции
    printf("\n Введите знак операции: ");
    scanf("%c",&op); // ввод с клавиатуры знака операции
    printf("\n Введите значения аргументов: ");
    scanf("%d%d",&a,&b); // ввод с клавиатуры значений a и b
    switch (op){
        case '+': res = a + b; break;
        case '-': res = a - b; break;
        case '*': res = a * b; break;
        case '/':
        case ':': res = a / b; break;
        default : printf("\n Известная операция");};
}

```

} ОПЕРАТОРЫ ЦИКЛА. ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ

Циклические алгоритмы

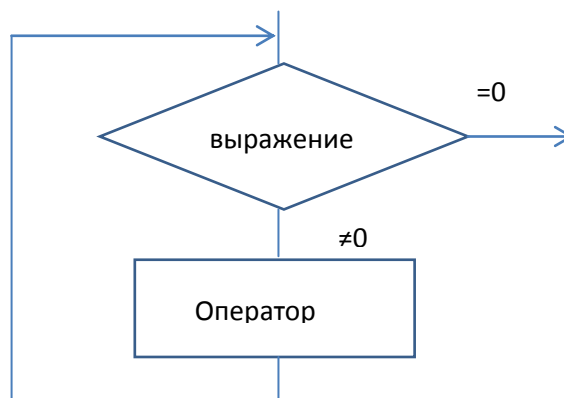
Использование компьютеров наиболее эффективно при многократном повторении, возможно с небольшими изменениями, простых действий.

Многократное повторение оператора или группы операторов называется циклом. Тело цикла – это повторяемые операторы. Однократное повторение тела цикла называется итерацией. В языке *C* есть три оператора цикла: цикл с предусловием *while*, цикл с постусловием *do while* и цикл с параметром *for*.

Цикл с предусловием

Синтаксис оператора:

while (выражение) оператор;



Выражение должно быть арифметическим. Выражение вычисляется, и если значение выражения не равно нулю, то выполняется оператор. После этого управление снова передается на проверку условия. Тело цикла выполняется до тех пор, пока значение выражения не станет равным нулю. Если в цикле надо выполнить несколько операторов, то их заключают в фигурные скобки.

Пример:

Дана последовательность из 10 целых чисел. Найти сумму этих чисел и вывести ее на экран.

Решение.

При обработке последовательностей все элементы размещаются в одной и той же ячейке, последовательно сменяя друг друга. Так как при вводе нового элемента последовательности предшествующее значение теряется, то надо описать об-

работку одного элемента и повторить эти действия столько раз, сколько элементов содержится в последовательности.

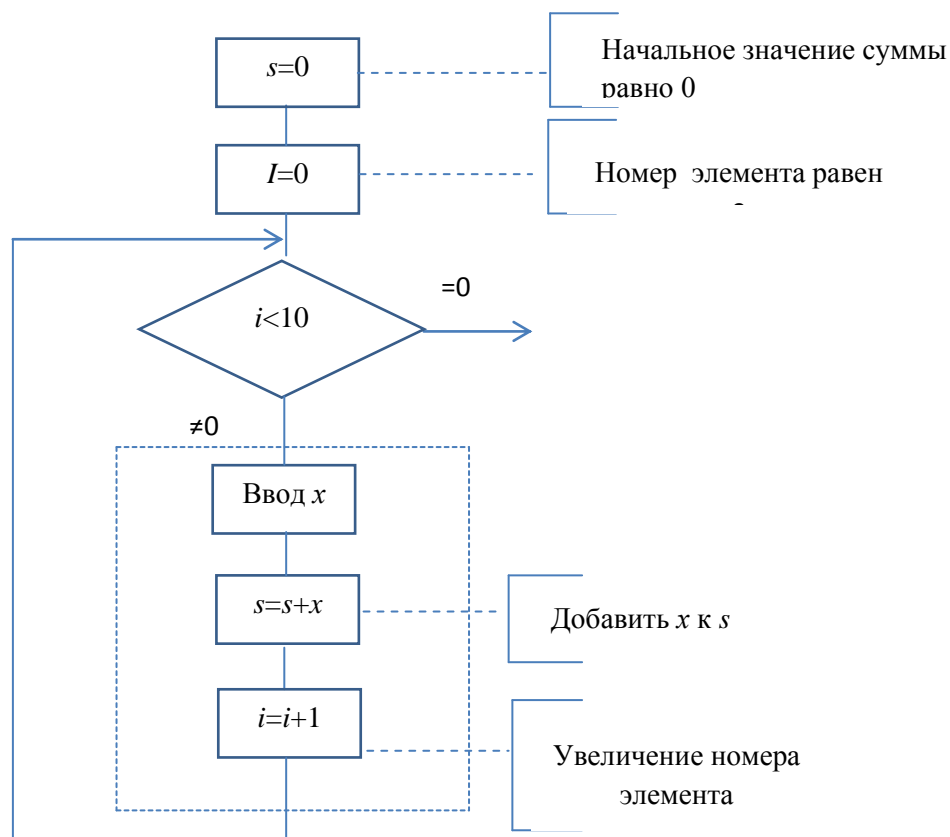
Нам понадобятся три целочисленные переменные: x – для хранения элемента последовательности, s – для суммы элементов последовательности и i – для номера текущего элемента. Вначале $i=0$ и $s=0$.

Будем выполнять следующие действия, пока не рассмотрим все 10 элементов (т.е. пока номер текущего элемента i меньше 10) :

- ввод значение переменной x ,
- добавление значения x к переменной s ,
- увеличение номера элемента i на единицу.

Таким образом, после завершения цикла в переменной s будет находиться сумма всех элементов последовательности.

Блок-схема:



Пунктирной линией выделено тело цикла. Оно состоит из трех операторов, поэтому эти операторы надо взять в фигурные скобки.

Программа.

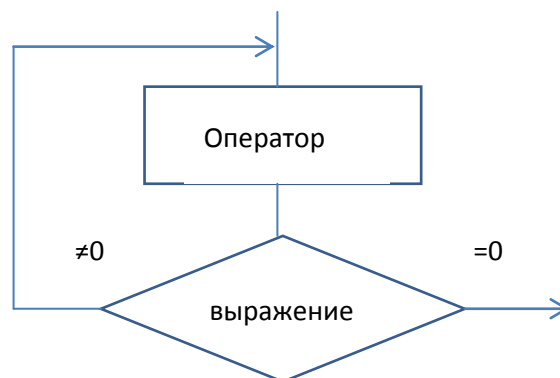
```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    int x,    //значение элемента последовательности
    i=0,      //счетчик - номер элемента
    s=0;      //сумма элементов последовательности
    while (i<10) {
        scanf ("%d",&x) ; //ввод элемента
        s+=x;              //добавление элемента к сумме
        i++;               //увеличение счетчика
        printf ("\ns=%d",s) ; //печать результата
    }
    return 0;}
```

Цикл с постусловием

Синтаксис оператора:

do оператор;while(выражение);



Оператор выполняется до тех пор, пока значение выражения не станет равным нулю. Если в цикле надо выполнить несколько операторов, то их надо заключить в фигурные скобки.

Особенность этого оператора цикла в том, что тело цикла выполняется хотя бы один раз.

Пример:

Дана последовательность символов, которая заканчивается точкой. Подсчитать, сколько раз в этой последовательности встречается буква 'а'.

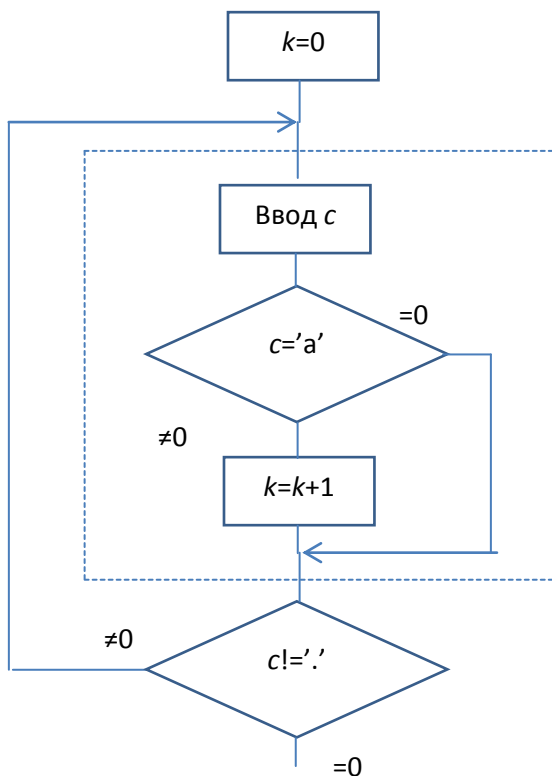
Решение.

Нам понадобятся две переменные: x – символьная переменная и k – счетчик количества символов – переменная целого типа.

Вводим с клавиатуры символьную переменную. Если значение переменной равно 'а', то увеличиваем счетчик на единицу. Будем повторять эти действия, пока значение переменной не станет равным '.'.

Блок-схема.

Пунктирной линией выделено тело цикла. Оно состоит из двух операторов, поэтому эти операторы надо взять в фигурные скобки.



Программа

```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{
    int k=0;
```

```

char c;
do {
    scanf("%c",&c);
    if(c=='a') k++;
    }while(c!='. ');
printf("\ns=%d",k);
return 0;
}

```

Цикл с параметром

Синтаксис оператора:

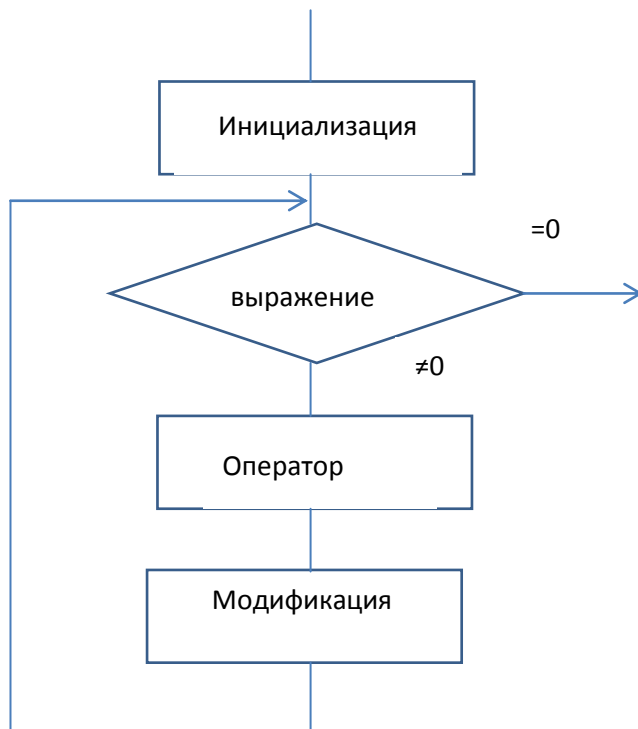
for (инициализация; выражение; модификации) оператор;

Инициализация – используется для объявления и присвоения начальных значений величинам, используемым в цикле. Может содержать несколько операторов через запятую (операция **последовательное выражение**). Инициализация выполняется один раз в начале исполнения цикла. Область действия переменных, объявленных в части инициализации – это цикл.

Выражение - определяет условия выполнения цикла. Цикл реализован как цикл с предусловием.

Модификация выполняется после каждой итерации цикла и служит для изменения параметров цикла. В части модификации может быть записано через запятую несколько операторов.

Оператор – простой или составной оператор. Составляет тело цикла.



Любая из составных частей **for** может быть опущена, но точки с запятой надо оставить на местах. Например:

for ($x=1$; $x \leq 10$; $x+=0.5$) $y=y+x$;

for ($x=1$; $x \leq 10$;){ $x+=0.5$; $y=y+x$.}

for ($x=1$; $x \leq 10$; $x+=0.5$, $y=y+x$);

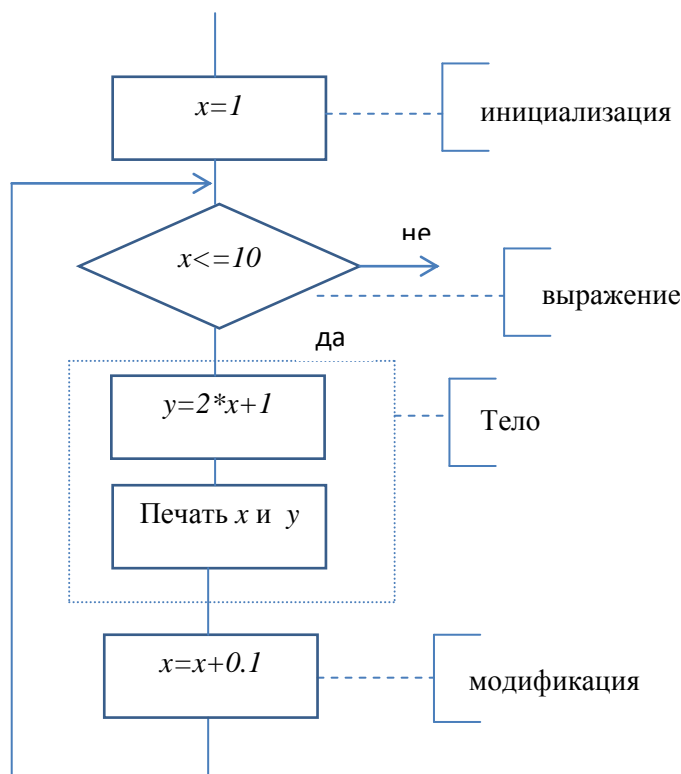
for ($x=1$; ; $x+=0.5$, $y=y+x$)

if ($x \leq 10$) *break*:

Пример 1:

Протабулировать функцию $y=2^x+1$ на интервале от 1 до 10 с шагом 0,1. Вычислить и напечатать значения x и y .

Блок-схема



Программа

```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{ float x,y;
  for(x=1;x<=10;x+=0.1){
    y=2*x+1;
    printf("\nx=%f \ty=%f",x,y);
    return 0;}
```

Пример 2:

Найти и вывести на экран сумму элементов последовательности

$$x + x^3 + x^5 + x^7 + x^9 \dots$$

с точностью до 0.001 при $0 < x < 1$.

Решение.

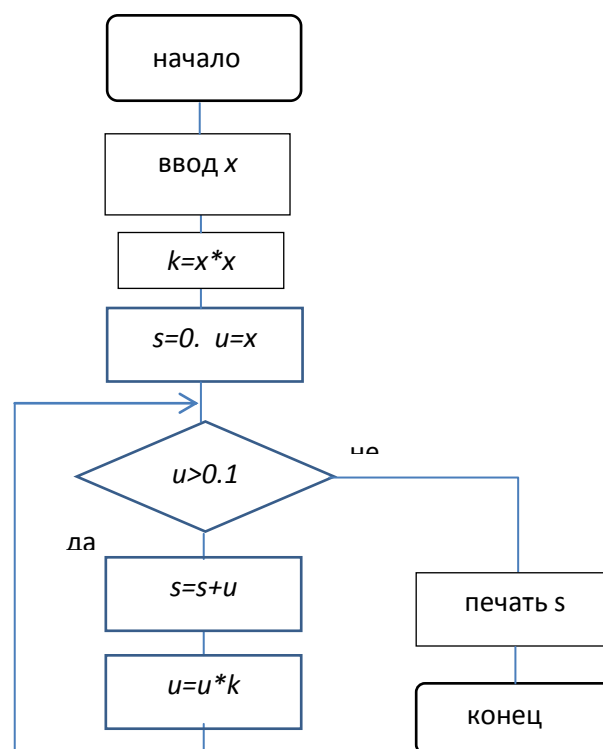
Последовательность при $0 < x < 1$ является убывающей: значение очередного элемента будет меньше предыдущего. Найти сумму элементов с точность до 0.001

означает, что добавление элементов к текущей сумме надо прекратить, когда значение очередного элемента последовательности станет меньше 0.001.

Нам понадобятся три переменные: x – исходное число, s – сумма элементов последовательности, u – значение очередного элемента последовательности.

Каждый элемент представляет собой степень числа x , причем степень очередного элемента на 2 больше степени предыдущего. Таким образом, можно выразить значение следующего элемента через значение предыдущего следующим образом: $u = u * x^2$. Вначале u принимаем равным 1.

Блок-схема



Программа.

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{ float x,
    s, //сумма элементов последовательности
    u, // значение текущего элемента последовательности
    k;

    scanf ("%f", &x); //ввод значения x
```



```

k=x*x;
for (s=0,u=x;u>0.01;u*=k)
    s+=u;
printf("\nx=%f",s);
return 0;
}

```

Вложенные циклы

Если в теле одного цикла содержится другой оператор цикла, то говорят о вложенных циклах.

Пример.

Вывести на экран таблицу умножения.

Решение.

Чтобы получить таблицу, надо каждое число в интервале от 1 до 9 умножать последовательно на все числа от 1 до 9 и выводить на экран полученные произведения.

Программа

```

#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{ int a,b;
printf("\n  |");
for(a=1;a<10;a++) // печать строки заголовка
    printf("%3d",a); // значения 2-го сомножителя
printf("\n");
for(a=0;a<10;a++)
    printf("___");
for(a=1;a<10;a++) { //внешний цикл– изменение 1-го сомножителя
    printf("\n\n%2d| ",a); //печать значения 1-го сомножителя
    for(b=1;b<10;b++) //внутренний цикл – изменение 2го сомножителя
        printf("%3d",a*b); //печать произведения

```

```

}
return 0;
}

```

2. Практическая часть

1. Дано 4-значное целое число. Вычислить и вывести на экран сумму цифр этого числа.
2. Дано 4-значное целое число. Определить, равна ли сумма старших цифр сумме младших.
3. Определить число, полученное выписыванием в обратном порядке цифр заданного трехзначного числа, и вывести его на экран.
4. Найти и вывести на экран произведение цифр заданного четырехзначного числа.
5. Дано:

$$b=3, a=80.$$

Ввести с клавиатуры значение x , вычислить и вывести на экран значение y .

$$y = \begin{cases} 7x+25, & \text{если } x \leq b \\ x^2, & \text{если } b < x \leq a \\ -x^3, & \text{если } x > a \end{cases}$$

6. Значение x ввести с клавиатуры. Вычислить и вывести на экран значение y .

$$y = \begin{cases} x, & \text{если } x \leq 0 \\ x^2, & \text{если } 0 < x \leq 1 \\ x^3, & \text{если } x > 1 \end{cases}$$

7. Ввести x и вычислить y :

$$\left\{ \right.$$

$$y = \begin{cases} 2x+1, & \text{если } x \leq 0 \\ 1/x, & \text{если } x > 0 \end{cases}$$

Вывести на экран x и y .

8. Ввести число x – возраст человека.

Определить, ходит ли он

- в ясли (от 1 до 3 лет),
- в детский сад (от 4 до 6 лет),
- в школу (от 7 до 16 лет),
- окончил школу.

9. Ввести число t от 0 до 24 – время дня. Определить, какому времени суток соответствует этот час:

- от 0 до 3 – ночь,
- от 4 до 10 – утро,
- от 11 до 17 – день,
- от 18 до 21 – вечер,
- от 22 до 24 – ночь.

10. Дано: t – температура воздуха. Вывести на экран сообщение о погоде: холодно, прохладно, тепло, жарко.

11. Даны 2 целых числа a и b . Сделать так, чтобы в a находилось большее, а в b – меньшее из чисел.

12. Определить, имеет ли функция $y = 2 \cdot x - 37$ корень на отрезке (a, b) .

13. Проверить, можно ли вложить лист бумаги размером $a \times b$ в конверт размером $c \times d$. Края листа бумаги должны быть параллельны краям конверта.

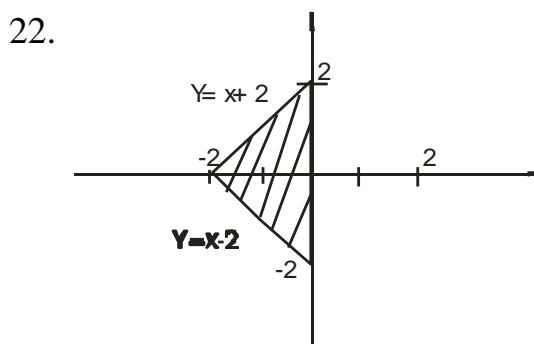
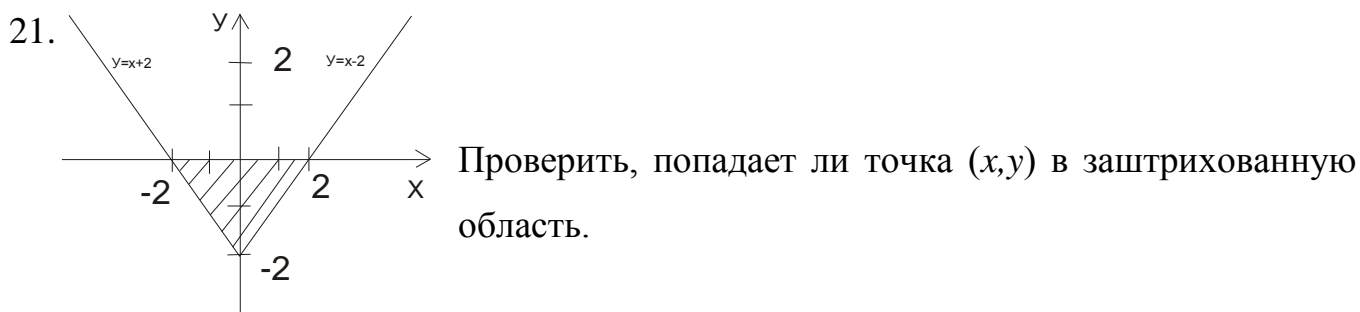
14. Дано: a , b и c – углы треугольника в градусах. Определить:

- а) Можно ли построить такой треугольник
- б) Если можно, то определить тип треугольника.

15. Дано: a , b и c – стороны треугольника. Проверить, можно ли построить треугольник с такими сторонами, и если можно, то определить его тип (равносто-

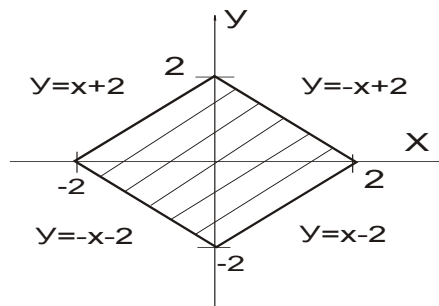
ронный, равнобедренный, разносторонний).

16. Даны 4 числа: оценки студента на экзаменах. Если он получил хотя бы одну оценку «2» - его отчислят. Если он получил хотя бы одну оценку «3» - его лишат стипендии. Если он получил все оценки «5» - он получит повышенную стипендию. Вывести на экран сообщение о решении деканата.
17. Дано 3 целых числа. Вывести на экран «Да» или «Нет» в зависимости от того, имеют 3 числа одинаковую четность или нет
18. Даны целые числа a , b , c и d . Найти и вывести на экран наибольшее из них.
19. Даны целые числа a , b , c и d . Проверить, есть ли среди них пары одинаковых чисел.
20. Даны целые числа a , b , c и d . Найти и вывести на экран число, которое ближе всего к среднему арифметическому этих чисел.



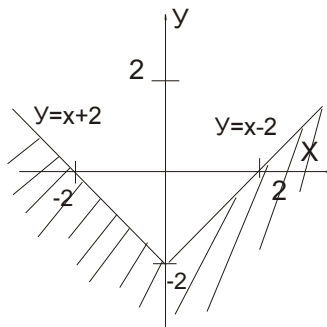
Проверить, попадает ли точка (x,y) в заштрихованную область

23.



Попадает ли точка с координатами (x, y) в заштрихованную область.

24.



Проверить попадает ли точка (x, y) в заштрихованную область.

25. Дан номер месяца. Определить и вывести на экран количество дней в этом месяце.

1. Вычислить и вывести на экран значения функции

$$y = \begin{cases} 3x^2 + 2 & 1 \leq x < 2 \\ x & 2 \leq x < 4 \\ 5/x - 8 & 4 \leq x \leq 9 \end{cases}$$

при x , который изменяется на интервале от 1 до 9 с шагом 1.

2. Найти на интервале от 9,1 до 10 корень уравнения $x^3 + x = 1000$ с точностью до 0.0001.

3. Дана непустая последовательность положительных целых чисел, за которыми следует 0 (признак конца последовательности). Вычислить среднее арифметическое этих чисел.

4. Вывести на экран латинские буквы, коды которых четны.

5. Вывести на экран символ, порядковый номер которого является сред-

ним арифметическим для кодов цифр.

6. Дана дата в виде: день, месяц. Напечатать, сколько дней прошло от начала года.
7. Дано целое $n > 0$, за которым следует n вещественных чисел. Определить, сколько среди них отрицательных.
8. Дана последовательность из N вещественных чисел. Определить, каких значений больше – положительных или отрицательных.
9. Дана последовательность из N целых чисел. Найти среднее арифметическое максимального и минимального элементов последовательности
10. Дана последовательность из N целых чисел. Определить, сколько элементов последовательности находится в интервале от -1 до 45.
11. Найти значения 40 членов ряда $1; \frac{3}{4}; \frac{4}{8}; \frac{5}{16} \dots$
12. Найти значения 10 членов ряда $1; \frac{1}{4}; \frac{1}{6}; \frac{1}{8} \dots$

13.
$$y = \frac{x}{x+1} + \frac{x^2}{x+2} + \frac{x^3}{x+3} + \dots$$

$x = 0,59.$

Вычислить y с точностью до $\epsilon = 0,000001$.

14.
$$a_n = \frac{(-1)^n \cdot 2^{2n}}{n(n+1)(n+2)}, \text{ где } n = 1, 2, 3, \dots$$

Найти такое n , для которого $|a_{n+1}| < 10^{-5}$.

15. Найти сумму:

$$1 - \frac{x^3}{1 \cdot 3} + \frac{x^5}{2 \cdot 5} - \frac{x^7}{3 \cdot 7} + \dots$$

при $x=0,1$ с точностью 0,0001.

16. Найти сумму:

$$1 - \frac{x^3}{1 \cdot 3} + \frac{x^5}{2 \cdot 5} - \frac{x^7}{3 \cdot 7} + \dots$$

при $x=0,1$ с точностью 0,0001.

17. Найти сумму:

$$1 + \frac{x}{2!} + \frac{x^2}{4!} + \frac{x^3}{6!} + \dots$$

при $x=0,7$ с точностью 0,00001 и определить количество слагаемых.

18. Найти сумму:

$$1 + \frac{x^3}{2!} + \frac{x^5}{2! \cdot 3!} + \frac{x^7}{3! \cdot 4!} + \dots$$

при $x=0,9$ с точностью 0,00001.

19. Дана последовательность символов, которая заканчивается точкой. Определить, сколько раз в этой последовательности встречается сочетание “no”.

20. Дана последовательность из n целых чисел. Определить, является ли она невозрастающей.

21. Дана последовательность из целых чисел, которая заканчивается нулем. Определить, является ли она неубывающей

22. Дано: $c_0=1.5$, $c_i=2*(c_{i-1}+4)$. Найти значение суммы 20 членов ряда.

23. Дано: x – целое число. Определить, является ли оно простым.

24. Вычислить и вывести на экран значения функции

$$a=x*\sin(y)+y*\cos(x) \text{ при}$$

при x , который изменяется на интервале от -3 до 4 с шагом 0,2 и

y , который изменяется на интервале от -2 до 1 с шагом 0,5

25. Для каждого целого числа из интервала a до b , вывести на экран все его делители.

26. Дано целое число x . Вывести на экран все его делители.

3. Порядок выполнения и сдачи работы

1. Изучить теоретическую и практическую часть.

2. Сдать преподавателю теорию работы путем ответа на контрольные вопросы.

3. Получить вариант задания и выполнить задание в Visual Studio 2013.

4. Предъявить преподавателю результат выполнения задания для самостоятельной работы

Оформить отчет.

Содержание отчета

Отчет готовится в электронном виде и распечатывается. Он должен содержать текст задания, скриншот программного кода и скриншот результата .

4. Контрольные вопросы

1. Какую структуру имеет программа на языке Си?
2. Какие требования предъявляются к идентификаторам?
3. Как описываются переменные?
4. Как обозначаются логические операции?
5. Как задаются вещественные константы?
6. Какие типы данных называются основными?
7. Как типы применяются для описания вещественных переменных?
8. Каков синтаксис условного оператора?
9. Тело какого цикла выполняется хотя бы один раз?
10. Какого типа выражение используется в операторе while?
11. Для чего служит раздел инициализации оператора for?
12. Сколько операторов может быть в разделе модификации оператора for?
13. Каким знаком отделяются друг от друга составные части оператора for?
14. Можно ли опустить одну из составных частей оператора for?

ЛАБОРАТОРНАЯ РАБОТА № 2

МАССИВЫ

Цель работы. Создание массивов.

1. Теоретическая часть

Массив – это конечная поименованная последовательность однотипных величин.

Описание массива:

<тип> <имя> [<размерность>];

[] – это часть синтаксиса.

Размерность массива вместе с типом его элементов определяет объём памяти, который необходим для размещения массива. Поэтому размерность массива может быть указана только константой или константным выражением. Например:

const int N=15;

char mas[N]; массив из 15 символов.

float a[10]; массив из 10 вещественных чисел.

Инициализирующие значения записываются в фигурных скобках. Значения элементам присваиваются по порядку. Инициализаторов должно быть не больше количества элементов массива. Если элементов в массиве больше, чем инициализаторов, элементы, для которых значение не указано, обнуляются.

Например:

int b[5] = {4, 3, 1};

Элементы массива *b*

4	3	1	0	0
---	---	---	---	---

Если размерность не указана, то должен быть указан инициализатор. В этом случае память выделяется по количеству инициализирующих значений.

Например:

int a[] = {4, 7, 3, 1, 2}; будет создан массив из 5 элементов

Чтобы обратиться к элементу массива, надо после имени массива в квадратных скобках указать номер элемента (индекс). Элементы массива нумеруются от 0.

Например:

Номер элемента

0	1	2	3	4
---	---	---	---	---

i					
Элементы массива					
a	4	7	3	1	2
Обращение к i элементу	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$

Массивы обрабатываются поэлементно. Чтобы обратиться к каждому элементу используют циклы, в которых номер (индекс) элемента меняется от 0 до количества элементов массива.

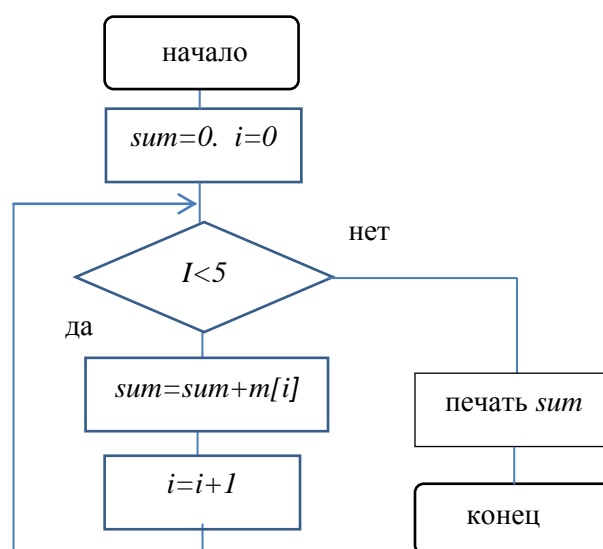
Пример.

Вычислить и вывести на экран сумму элементов массива, состоящего из 5 элементов вещественного типа. Массив проинициализировать при объявлении.

Решение

Нам понадобятся три переменные: m – массив элементов вещественного типа, i – индекс элемента массива (значение целого типа), sum – сумма элементов массива (значение вещественного типа). Вначале s и i равны 0.

Блок-схема



Программа

```
void main ( ) {
```

```

    int ;
    float, sum;
    float m [5] = { 3.6, -0.53, 4.123, 3.906, -76.4 };
    for ( i = 0, sum = 0; i < 5 ; i ++ )
        sum + = m[i];
    printf( "%f", sum);
}

```

Инициализация элементов массива случайными значениями

Генератор случайных чисел предназначен для получения случайных чисел. Для работы с генератором необходимо присоединить заголовочный файл `<stdlib.h>`.

Функция `rand()` возвращает псевдослучайное число в интервале от 0 до 32 767. Изменить интервал можно, используя деление по модулю. Например, получить число в интервале от 0 до 100 можно так: `x=rand()%100`.

Получить число в интервале от - 100 до 100 можно так: `x= rand()%200-100`.

Функция `srand (int)` устанавливает исходное число для последовательности, генерируемой функцией `rand()`.

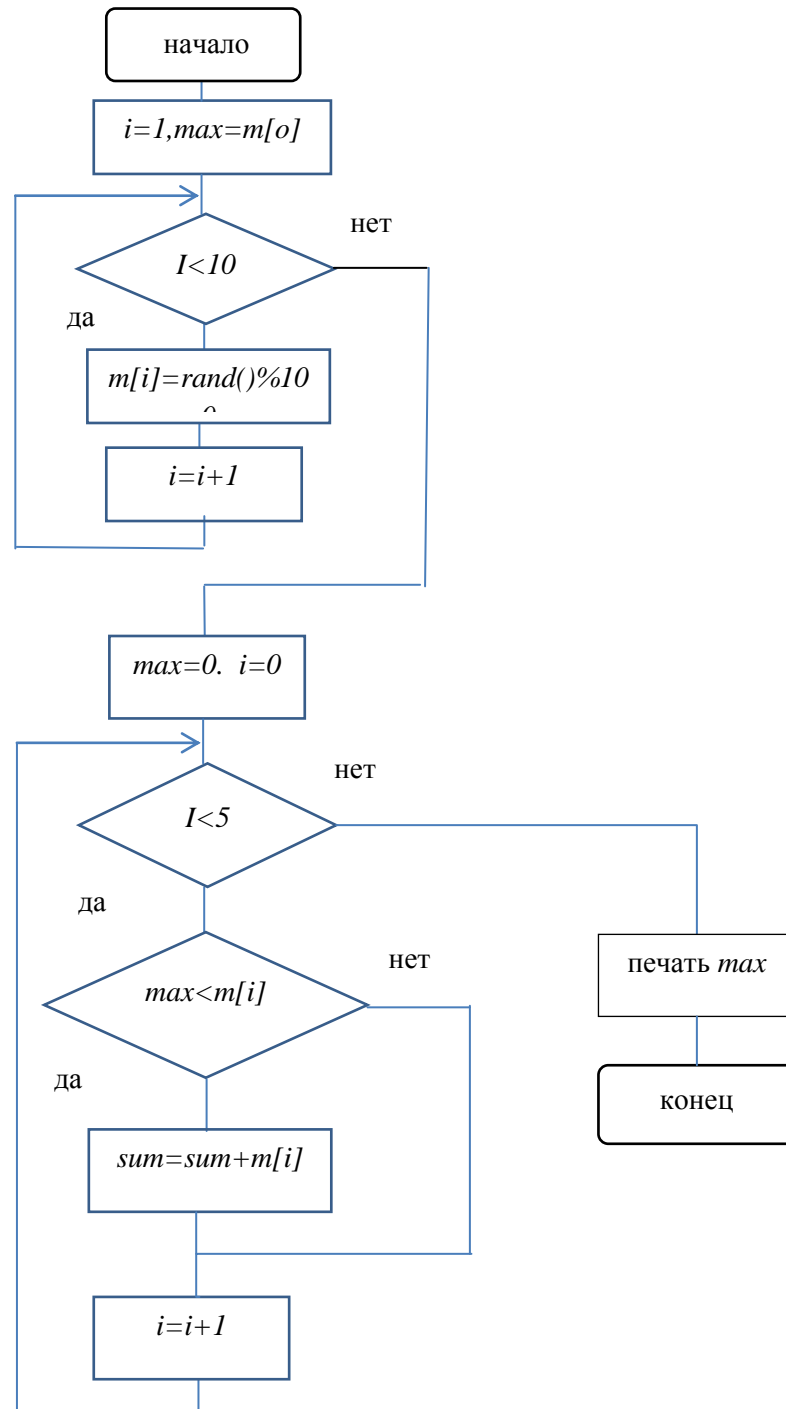
Пример.

Найти максимальный элемент массива, состоящего из 10 целых чисел в интервале от 0 до 100. Заполнить массив случайными числами.

Решение

Нам понадобятся три переменные: `m` – массив элементов целого типа, `i` – индекс элемента массива (значение целого типа) , `max` – значение максимального элемента массива (значение целого типа). Переменной `max` вначале присвоим значение первого элемента массива. Затем будем последовательно сравнивать `max` с остальными элементами, и если значение какого-либо элемента окажется больше `max`, то присвоим `max` значение этого элемента. Таким образом, в конце концов в `max` окажется значение наибольшего элемента массива.

Блок-схема



Программа

```
#include <stdio.h>
#include <stdlib.h>
void main ( ){
    const int n = 10;
    int i, max;
```

```
int m [ n ];
```

```
for ( i = 0; i < n ; i ++ ) //Заполнение массива случайными числами
```

ми

```
m[i]=rand()%100;
```

```
for ( i = 1, max=m[0]; i < n ; i + + )
```

```
if(max<m[i])max= m[i];
```

```
printf(“maximum=%d”,max);
```

```
}
```

МНОГОМЕРНЫЕ МАССИВЫ

Цель работы. Создание многомерных массивов.

1. Теоретическая часть

Многомерные массивы задаются указанием каждого измерения в квадратных скобках;

<тип> <имя> [размерность][размерность];

/

\

к-во

к-во

строк

столбцов

Например:

`int m [6][8];` таблица из 6 строк и 8 столбцов.

Теоретически размерность массивов не ограничена.

В памяти массив размещается построчно. При переходе к очередному элементу быстрее всего меняется самый правый элемент.

Для доступа к элементу надо указать все его индексы.

`m [2][4];` 4 элемент 2 строки

`m [i][j];` j элемент i строки

Инициализация двухмерного массива

Задается общий список элементов, в том порядке, как они располагаются в памяти.

```
int a[ 3 ][ 5 ] = { 1, 2, 4, 6, 1, 5, 7, 3, 0, 1, 7, 2 };
```

```
1 2 4 6 1
```

```
5 7 3 0 1
```

```
7 2 0 0 0
```

Двухмерный массив можно представить как массив массивов, каждый массив заключается в свои фигурные скобки. При этом начальные значения можно указывать не для всех элементов.

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

```
1 2 3 0 0
```

```
1 2 0 0 0
```

```
1 0 0 0 0
```

Если для каждой строки задать хотя бы одно инициализирующее значение, то количество строк можно не указывать.

```
int a [ ][ 5 ] = {{ 1, 2 }, { 1 }, { 3, 4, 5, 6 }};
```

```
1 2 0 0 0
```

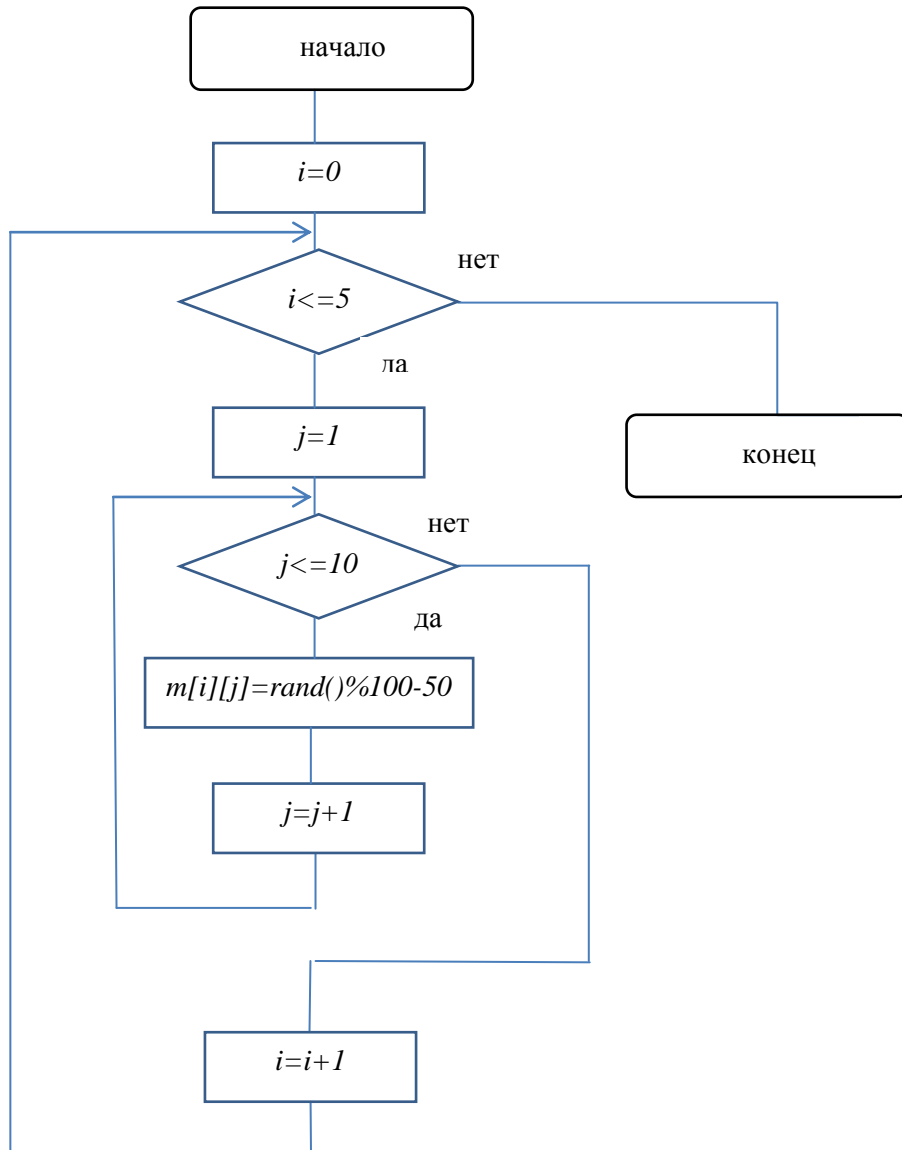
```
1 0 0 0 0
```

```
3 4 5 6 0
```

Пример.

Заполнить двухмерный массив, состоящий из 5 строк и 10 столбцов, случайными числами в интервале от -50 до 50. Тип элементов массива – целый.

Блок-схема



Программа

```
#include "stdafx.h"
#include <stdlib.h>
int _tmain(int argc, _TCHAR* argv[])
{int m[5][10],i,j,k;
for(i=0;i<5;i++)
    for(j=0;j<10;j++)
        m[i][j]=rand()%100-50;
return 0;}
}
```

Пример:

Дана матрица размером 10x10. Тип элементов – целый. Найти сумму элементов, лежащих над пересечением диагоналей.

Решение

	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.
0.										
1.										
2.										
3.										
4.										
5.										
6.										
7.										
8.										
9.										

На рисунке нужные элементы заштрихованы. Видно, что будут суммироваться элементы первых 4 строк матрицы. Чем больше номер строки, тем она короче. Составим таблицу, в которой для каждой строки укажем начальный и конечный номер столбца.

Номер строки i	Начальный номер столбца $j_{\text{нач}}$	Конечный номер столбца $j_{\text{кон}}$	$i+j_{\text{кон}}$
0	1	8	$0+8=8$
1	2	7	$1+7=8$
2	3	6	$2+6=8$
3	4	5	$3+5=8$

Из таблицы видно, что можно выразить начальный и конечный номер столбца через номер строки:

$$j_{\text{нач}} = i + 1,$$

$$j_{\text{кон}} = 8 - j_{\text{нач}}.$$

Программа

```
#include "stdafx.h"
#include <stdlib.h>

int _tmain(int argc, _TCHAR* argv[])
{int m[10][10], i, j, k;
//заполнение массива случайными числами
for(i=0; i<10; i++)
    for(j=0; j<10; j++){
        m[i][j]=rand()%100-50;}
//обнуление элементов, лежащих выше пересечения диагоналей
for(i=0; i<5; i++)
    for(j=i+1; j<9-i; j++)
        m[i][j]=0;
//вывод массива на экран
for(i=0; i<10; i++){
    printf("\n");
    for(j=0; j<10; j++)
        printf("%4d", m[i][j]);}
    return 0;
}
```

2. Практическая часть

Задания

1. Заполнить массив $A(10)$ случайными числами. Напечатать количество и сумму отрицательных элементов этого массива.
2. Заполнить массив $M(10)$ случайными числами. Подсчитать и вывести на экран сумму положительных элементов массива.
3. Заполнить массив $mas(10)$ случайными числами. Подсчитать и вывести на экран количество четных элементов массива.
4. Заполнить массив $A(10)$ случайными числами. Подсчитать и вывести на экран количество элементов массива, кратных 7 и не кратных 3.
5. Заполнить массив $X(10)$ случайными числами. Подсчитать и вывести на экран количество элементов массива, значения которых лежат в интервале от 25 до 50.
6. Заполнить массив $A(10)$ случайными числами. Подсчитать и вывести на экран сумму четных положительных элементов массива.
7. Дан одномерный массив $p(10)$. Элементы массива проинициализировать. Переставить элементы массива в обратном порядке.
8. Заполнить массивы $X(15)$ и $Y(15)$, случайными числами. Получить все числа массива Y , которые не входят в массив X .
9. Заполнить массив $C(20)$ случайными числами. Получить числа, взятые по одному из каждой группы равных членов и указать их количество (и количество элементов в группе).
10. Дан массив $E(10)$. Значения элементов массива ввести с клавиатуры. Выяснить, сколько чисел входит в массив более чем по одному разу.
11. Дан массив $E(10)$. Значения элементов массива ввести с клавиатуры. Выяснить, сколько чисел встречается в массиве только один раз.
12. Даны 3 массива $A(5)$, $B(5)$ и $C(5)$. Заполнить эти массивы случайными числами. Получить новый массив D по следующему правилу:

$$D_1=A_1,$$

$$D_2=B_1,$$

$$D_3=C_1,$$

$$D_4=A_2,$$

$$D_5=B_2 \quad \text{и т. д.}$$

13. Дан массив $A(10)$. Заполнить массив случайными числами. Получить новый массив B по следующему правилу:

$$B_1=A_1+A_{10},$$

$$B_2=A_2+A_9,$$

и так далее.

14. Дан массив $A(30)$. Элементы массива проинициализировать. Пусть L - максимальный, а M - минимальный элемент из массива A . Получить в порядке возрастания все целые числа из интервала $[M, L]$, которые не входят в массив A .

15. Дан массив $R(5)$. Значения элементов массива ввести с клавиатуры. Найти сумму значений $|R_i - R_j|$ где $0 \leq i < j < 5$. Функция, которая возвращает абсолютную величину целочисленного аргумента, называется *abs* и содержится в заголовочном файле *math.h*. Ее формат: `int abs(int x)`.

16. Даны число X и массив $Z(10)$. Элементы массива проинициализировать. Найти в последовательности Z два числа, среднее арифметическое которых ближе всего к X .

17. Заполнить массив $A(10)$ случайными числами. Упорядочить элементы массива по возрастанию.

18. Заполнить массив $A(10 \times 10)$ следующим образом:

0 0 0 0 ... 0

0 1 0 0 ... 0

0 0 2 0 ... 0

.....

0 0 0 0 ... 9

Вывести на экран полученный массив

19. Дан массив $F(5, 5)$. Вывести его на экран. Вывести на экран индексы элементов, лежащих в интервале от 10 до 30.

20. Дан массив $C(6,6)$ и массив $B(6,6)$. Образовать новый массив A , такой, что $A_{ij} = \min(B_{ij}, C_{ij})$. Вывести на экран полученный массив.

4. Дан массив $X(10,10)$. Вывести его на экран. Найти и вывести на

экран максимальный элемент массива.

21. Дан массив $A(5,5)$. Вывести его на экран. Увеличить в 2 раза каждый элемент 3 строки массива. Вывести на экран полученный массив.

6. Дан массив $D(10,10)$. Найти максимальный элемент в нижней половине и минимальный элемент в верхней половине матрицы и поменять их местами. Вывести на экран исходный и результирующий массив.

max

22. Дана матрица $A(10 \times 10)$. Транспонировать матрицу. Вывести на экран исходную и результирующую матрицу.

23. Дан массив $A(10 \times 10)$. Найти сумму отрицательных элементов из заштрихованной на рисунке области.

24. Даны массивы $A(2,2)$ и $B(2,2)$.

Вывести на экран числа, которые находятся одновременно в этих двух массивах

25. Даны массивы $A(3,3)$ и $B(3,3)$.

Вывести на экран элементы массива A , которые не встречаются в массиве B .

26. Дан массив $A(5,5)$. Вывести его на экран. Найти и вывести на экран сумму и произведение положительных элементов строк, которые начинаются с отрицательного числа.

27. Дан массив $A(10 \times 10)$. Найти количество положительных элементов из заштрихованной на рисунке области.

28. Дан массив $A(10 \times 10)$. Найти количество отрицательных элементов из заштрихованной на рисунке области.

29. Дан массив $A(10 \times 10)$. Упорядочить по возрастанию строки массива. Вывести на экран исходный и результирующий массив.

30. Дан массив $A(10 \times 10)$. Поменять местами 3 строку и 7 столбец. Вывести на экран исходный и результирующий массив.

31. Дан двумерный массив.

Преобразовать его в одномерный массив, читая по спирали. Вывести на экран полученный одномерный массив.

32. Даны два массива А (2,3) и В (3,4).

Найти их произведение и вывести на экран полученный массив.

3. Порядок выполнения и сдачи работы

1. Изучить теоретическую и практическую часть.
2. Сдать преподавателю теорию работы путем ответа на контрольные вопросы.
3. Получить вариант задания и выполнить задание в Visual Studio 2013.
4. Предъявить преподавателю результат выполнения задания для самостоятельной работы

Оформить отчет.

Содержание отчета

Отчет готовится в электронном виде и распечатывается. Он должен содержать текст задания, скриншот программного кода и скриншот результата .

4. Контрольные вопросы

1. Как задается размерность массива?
2. Каким может быть тип элементов массива?
3. Как выполняется инициализация элементов массива?
4. Как выполняется доступ к элементу массива?
5. Каков индекс первого элемента массива?
6. Сколько размерностей может быть у массива?
7. Как выполнить инициализацию двухмерного массива?
8. В каком случае можно опустить первую размерность при объявлении массива?
9. Как размещаются в памяти элементы массива?
10. Как обратиться к элементу многомерного массива ?

ЛАБОРАТОРНАЯ РАБОТА № 3

ФУНКЦИИ

Цель работы. Создание функций.

1. Теоретическая часть

Функция – это последовательность описаний и операторов, которые выполняют какое-либо законченное действие. Функция может возвращать значение.

Любая функция должна быть объявлена и определена. Объявлений может быть несколько. А определение – только одно. Объявление должно появиться в тексте программы до первого ее вызова.

Объявление функции задает ее имя, тип возвращаемого значения и список передаваемых параметров.

Определение функции состоит из объявления и тела функции. Тело функции – это последовательность операторов и описаний в фигурных скобках.

[класс] тип имя ([список параметров])

{ тело функции }

Класс – необязательный параметр, который позволяет задать область видимости функции.

Тип возвращаемого значения может быть любым, кроме массива и функции (но может быть указателем на массив или функцию). Если функция не должна возвращать значение – указывается тип *void*.

Список параметров определяет величины, которые должны передаваться в функцию при ее вызове. Элементы списка разделяются запятыми. Для каждого параметра, передаваемого в функцию, указывается его тип и имя.

В определении, в объявлении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать. Имена могут различаться, так как в объявлении имена игнорируются, а вызывать функцию можно с разными именами.

Все величины, описанные внутри функции, а также ее параметры, являются локальными. Их область действия – это функция.

Для возврата результата в вызывающую программу используется оператор `return [<выражение>];`

Выражение преобразуется к типу возвращаемого значения и передается в точку вызова. Поэтому функцию можно использовать как аргумент (параметр) в выражении. Выражение не указывается, если функция описана как *void*.

Пример

Создать функцию *sum*, которая возвращает в вызывающую программу значение суммы двух заданных целых чисел. Вызвать функцию *sum* из функции *main*.

Решение

Описание функции *sum* разместим в программе перед функцией *main*. Так как сумма целых чисел является целым числом, то тип возвращаемого значения будет *int*. Список параметров функции состоит из двух значений целого типа.

Программа

```
#include <stdio.h>

int    sum    (int a, int b){ return    a+b;} // Описание функции
void main ( ){
    int  x=68, y=42, z;
    z = sum ( x, y );// Вызов функции
}
```

Операторов *return* может быть несколько. Для функции типа *void* оператор *return* можно опустить, если он стоит перед закрывающей фигурной скобкой.

Пример

Создать функцию, которая возвращает наибольшее из 2х чисел (тип параметров – *double*). Применить функцию *mas* при выводе на экран наибольшего из чисел *a* и *b*.

Программа

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

```
double max (double x, double y ){
    if ( x > y ) return x; else return y ; }
void main(){
    double a,b,c;
    scanf ( " %f %f", &b, &c);
    printf(“max=%f”,max(b,c));
}
```

Функцию можно определить после функции *main*, тогда перед первым использованием ее следует объявить. В объявлении функции имена параметров можно не указывать.

Пример

Написать функцию *divid*, которая печатает значение частного и остатка от деления двух целых чисел. Определить функцию *divid* после функции *main*.

Решение

Функция только печатает результаты и не возвращает результат в вызывающую программу, поэтому тип функции будет *void*. По заданию определение функции располагается после функции *main*, значит функцию надо объявить перед функцией *main*. Список параметров функции состоит из двух значений целого типа.

Программа

```
#include <stdio.h>
void divid ( int , int );    // объявление функции
void main ( ) {
    int x = 18, y = 97, z;
    divid ( x, y ); // вызов функции
}
//описание функции
void divid ( int a, int b) {
    if(!b) printf(“частное от деления= %d, остаток =%d”,a/b,a%b) ;
    else printf(“деление на 0”);}
```


Передача параметров в функцию

Параметры в функцию можно передавать 2 способами:

1. По значению.
2. По адресу.

Передача по значению: в стек записываются копии значений параметров. Фактические параметры, соответствующие формальным параметрам, передаваемым по значению, могут быть выражениями или, в частном случае, переменными или константами. Во всех предыдущих примерах выполнялась передача по значению.

Передача по адресу: в стек заносятся адреса аргументов, функция по этим адресам обращается к параметрам, и все изменения, которым подвергаются эти параметры в функции, передаются в вызывающую программу (сохраняются после возврата управления вызывающей программе). Фактические параметры, соответствующие формальным параметрам, передаваемым по адресу, могут быть только переменными.

Передача по адресу выполняется с помощью указателя. При этом:

- в заголовке формальный параметр объявляется как указатель на тип аргумента,
- в теле функции для обращения к параметру используется операция разыменования,
- в обращении (при вызове) используется операция взятия адреса.

Пример

Написать функцию, которая возвращает сумму и разность 2х заданных целых чисел.

Решение

При помощи оператора *return* в вызывающую программу можно вернуть только одно значение, поэтому принимаем решение вернуть результаты через список параметров. Поскольку мы не применяем оператор *return*, тип функции будет *void*. Список параметров будет состоять из четырех элементов: первые два параметра (заданные числа) будут передаваться по значению, а вторые два (через которые будут возвращаться результаты) – по адресу.

Программа

```

# include <stdio.h>

// в заголовке функции параметры S и R объявляются как указатели на тип аргу-
мента
void SR ( int a, int b, int * S, int * R ) {
// обращения к параметрам S и R используется операция разыменования
    * S = a + b;
    * R = a - b;
}

void main(){
    int x=10, y=17, sum, sub;
// для передачи двух последних аргументов используется операция взятия адреса
    SR(x, y, &sum, &sub);
    printf("сумма x и y равна %d", sum);
    printf("разность x и y равна %d", sub);
}

```

Передача одномерных массивов

Одномерные массивы в функцию передаются по адресу. Информация о количестве элементов массива при этом теряется, поэтому ее надо передавать отдельно.

Пример функции, которая выводит на экран элементы одномерного массива, состоящего из целых чисел.

```

// m-адрес начала массива, n — количество элементов массива
void vivod(int *m, int n){
    for(int i=0; i<n; i++)
        printf("\n %d", m[i]);
}

```

В заголовке функции вместо `int *m` можно писать `int m[]`. Прямоугольные скобки указывают на то, что *m* представляет собой адрес массива.

При вызове в функцию передается адрес начала массива. А адрес начала массива – это его имя. Например, вызвать функцию *vivod* для массива *x*, состоящего из 10 целых чисел, можно так:

```
int mas[10];
vivod (mas,10);
```

Передача статических двумерных массивов

Элементы статических двумерных массивов в памяти располагаются построчно. Поэтому в функцию такие массивы можно передавать как одномерные. При этом в функцию надо передавать количество строк и столбцов массива, а адресом начала массива будет адрес его первого элемента. Для доступа к элементу массива внутри функции можно использовать пересчет индексов двумерного массива в индекс одномерного.

Например:

Двухмерный массив $m[3][4]$

	0	1	2	3
строки 0				
1				
2				

Размещение массива $m[3][4]$ в памяти

Строка 0				Строка 1				Строка 2			
0	1	2	3	4	5	6	7	8	9	10	11

Пересчет индексов двумерного массива в индекс одномерного массива:

$$k=i*K+j=2*4+1=9,$$

где i -номер строки в матрице,

j -номер столбца в матрице.

K -количество столбцов матрицы,

k - индекс элемента в одномерном массиве.

Пример:

Написать функцию для вывода на экран любого двухмерного массива, состоящего из целых чисел.

Программа

```
#include "stdafx.h"
```

```
//s– количество строк, k– количество столбцов, m– адрес начала массива
```

```
void f(int *m,int s,int k){
```

```
    for(int i=0;i<s;i++){
```

```
        printf("\n");
```

```
        for(int j=0;j<k;j++){
```

```
            printf("%d ",m[i*k +j]);
```

```
        }
```

```
    }
```

```
int _tmain(int argc, _TCHAR* argv[]){
```

```
int m1[3][5];
```

```
int m2[2][8];
```

```
f(&m1[0][0],3,5); // &m1[0][0]-адрес первого элемента массива m1
```

```
f(&m1[0][0],2,8);
```

```
}
```

Если в функцию предполагается передавать двухмерные массивы с заданным количеством столбцов k , то их передают как одномерные массивы, элементами которых в свою очередь являются одномерные массивы, состоящие из k элементов, где k – это константа. Внутри функции обращение к элементам двухмерного массива выполняется обычным способом.

Пример.

Написать функцию, которая возвращает в вызывающую программу сумму элементов двумерного массива, содержащего 3 столбца. Тип элемента – целый.

Решение.

В списке формальных параметров функции будет два параметра:

- *int n* - количество строк двумерного массива,
- *int m[][3]* – адрес начала массива. Первые скобки остаются пустыми. Даже если указать значение, оно будет проигнорировано – скобки просто показывают, что параметр является адресом массива. Фактическим параметром, соответствующим этому формальному параметру будет имя любого двумерного массива, имеющего 3 столбца и состоящего из целых чисел.

Программа.

```
#include "stdafx.h"

int sum(int m[][3],int n){
    int s=0;
        for(int i=0;i<n;i++){
            for(int j=0;j<3;j++){
                s+=m[i][j];
            }
        }
        return s;
    }

void main(){
    int massiv[10,3],sm;
    int mas[6][3],x;
    sm=sum(massiv,10);
    x=sum(mas,6);
}
```

Передача динамических двумерных массивов

Динамические двумерные массивы передаются по адресу. Кроме того, в функцию передается информация о количестве строк и столбцов двумерного мас-

сива. Внутри функции обращение к элементам двумерного массива выполняется обычным способом.

Пример:

Написать функцию для вывода на экран динамического двумерного массива, состоящего из целых чисел.

Программа:

```
void f2(int **m,int s,int k){
    for(int i=0;i<s;i++){
        printf("\n");
        for(int j=0;j<k;j++){
            printf("%d ",m[i][j]);
        }
    }
}

int _tmain(int argc, _TCHAR* argv[])
{int n,k;
scanf("%d",&n);
scanf("%d",&k);
int **m=(int **)malloc(n*sizeof(int*));
for(int i=0;i<n;i++)
    m[i]=(int*)malloc(k*sizeof(int));
for(int i=0;i<n;i++)
    for(int j=0;j<k;j++)
        m[i][j]=i*10+j;

f2(m,n,k);
return 0;
}
```

2. Практическая часть

Задания

Функция *main* должна вызвать указанную функцию и вывести на экран полученный результат.

1. Написать функцию, которая возвращает в вызывающую программу площадь прямоугольника. Тип параметров – вещественный.
2. Написать функцию, которая возвращает в вызывающую программу периметр треугольника. Тип параметров – вещественный.
3. Написать функцию, которая возвращает в вызывающую программу площадь круга радиуса r . Тип параметров – вещественный.
4. Написать функцию, которая возвращает в вызывающую программу длину окружности радиуса r . Тип параметров – вещественный.
5. Написать функцию, которая возвращает в вызывающую программу объем параллелепипеда. Тип параметров – целый.
6. Написать функцию, которая возвращает в вызывающую программу количество секунд, которое содержится в t часах. Тип параметров – целый.
7. Написать функцию, которая возвращает в вызывающую программу количество часов, которое содержится в n днях. Тип параметров – целый.
8. Написать функцию, которая возвращает в вызывающую программу сумму и произведение двух целых чисел. Результаты возвращать через параметры, передаваемые по адресу.
9. Написать функцию, которая возвращает в вызывающую программу квадрат и куб аргумента. Результаты возвращать через параметры, передаваемые по адресу.
10. Написать функцию, которая возвращает в вызывающую программу сумму и среднее арифметическое двух целых чисел. Результаты возвращать через параметры, передаваемые по адресу.

11. Написать функцию, которая возвращает в вызывающую программу наибольшее и наименьшее из трех целых чисел. Результаты возвращать через параметры, передаваемые по адресу.
12. Написать функцию, которая возвращает в вызывающую программу номер дня и месяца для заданного дня в году. Результаты возвращать через параметры, передаваемые по адресу.
13. Написать функцию, которая возвращает в вызывающую программу площадь поверхности и объем куба. Результаты возвращать через параметры, передаваемые по адресу.
14. Написать функцию, которая возвращает в вызывающую программу два простых числа, ближайших к заданному целому числу (большее и меньшее). Результаты возвращать через параметры, передаваемые по адресу.
15. Написать функцию, которая возвращает в вызывающую программу два целых числа, между которыми находится заданное вещественное число. Результаты возвращать через параметры, передаваемые по адресу.

Функция `main` должна вызвать указанную функцию и вывести на экран полученный результат.

16. Написать функцию, которая возвращает сумму элементов указанной строки двумерного массива, содержащего 3 столбца. Тип элементов массива – вещественный.
17. Написать функцию, которая возвращает сумму и количество отрицательных элементов двумерного массива, содержащего 2 столбца. Тип элементов массива – целый.
18. Написать функцию, которая возвращает значение минимального элемента двумерного массива, содержащего 4 столбца. Тип элементов массива – вещественный.
19. Написать функцию, которая возвращает сумму элементов указанного столбца двумерного массива, содержащего 3 столбца. Тип элементов массива – целый.

20. Написать функцию, которая увеличивает в два раза элементы, лежащие на главной диагонали двумерного массива размера $N \times N$. Тип элементов массива – вещественный.
21. Написать функцию, которая обнуляет элементы главной диагонали двумерного массива размера $N \times N$. Тип элементов массива – целый.
22. Написать функцию, которая возвращает произведение элементов указанного столбца двумерного массива. Тип элементов массива – вещественный.
23. Написать функцию, которая возвращает сумму и количество положительных элементов динамического двумерного массива. Тип элементов массива – целый.
24. Написать функцию, которая возвращает значение максимального элемента динамического двумерного массива. Тип элементов массива – вещественный.
25. Написать функцию, которая возвращает произведение элементов указанной строки динамического двумерного массива. Тип элементов массива – вещественный.
26. Написать функцию, которая обнуляет кратные трем элементы, динамического двумерного массива. Тип элементов массива – целый.

2. Порядок выполнения и сдачи работы

1. Изучить теоретическую и практическую часть.
2. Сдать преподавателю теорию работы путем ответа на контрольные вопросы.
3. Получить вариант задания и выполнить задание в Visual Studio 2013.
4. Предъявить преподавателю результат выполнения задания для самостоятельной работы

Оформить отчет.

Содержание отчета

Отчет готовится в электронном виде и распечатывается. Он должен содержать текст задания, скриншот программного кода и скриншот результата .

3. Контрольные вопросы

1. Как описывается функция?
2. Какого типа значение может вернуть функция?
3. Какой оператор используется для возврата значения в вызывающую программу?
4. В каком случае оператор return может быть опущен?
5. Для чего предназначен список формальных параметров?
6. Какие существуют способы передачи параметров в функцию?
7. Как передаются массивы в функцию?
8. Как передается в функцию двухмерный массив с известным числом столбцов?
9. Как передается в функцию произвольный двухмерный массив?
10. Как передается в функцию динамический двухмерный массив?

ЛАБОРАТОРНАЯ РАБОТА № 4 СТРОКА

Цель работы. Обработка строк.

1. Теоретическая часть

Строка – это массив символов, который заканчивается нуль-символом. Нуль-символ – это символ с кодом, равным нулю. Он записывается в виде управляющей последовательности `"\ 0"`. По положению нуль-символа определяется фактическая длина строки.

Например:

`char str[10];` - это строка длиной 10 символов. Из них значащими являются девять символов, десятый – это нуль-символ.

Ввод - вывод строк

int scanf(const char * fmt,...);

Первый параметр – это строка формата. Второй параметр – это список переменных (адреса). Функция возвращает количество тех элементов данных, которым были присвоены значения.

Ввод строки выполняется со спецификатором %s. Ввод строки выполняется до первого пробельного символа (пробела, табуляции или символа перевода строки). Это значит, что *scanf* нельзя использовать для ввода строк, содержащих пробелы. Прочитанные символы помещаются в строку, а после введенных символов добавляется символ конца строки. В спецификаторах формата можно указать модификатор максимальной длины поля. Это целое число, которое располагается между % и спецификатором s. Оно ограничивает количество считываемых символов.

Например:

```
char s[10];  
scanf ("%9s" , s);
```

В строку будет прочитано не более 9 символов. Имя строки – это адрес начала строки, поэтому операция взятия адреса не нужна.

int scanf_s(const char * fmt,...);

Отличается от *scanf* тем, что после имени переменной в списке ввода указывается длина строки. Например:

```
scanf ("%9s" , s, 9);
```

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

Первый параметр – это строка формата. Второй параметр – это список выражений. Функция возвращает количество выведенных символов или отрицательное число в случае ошибки.

Вывод строки выполняется со спецификатором %s. При выводе можно задать спецификатор длины поля. Если заданного кол-ва позиций недостаточно, оно игно-

рируется, и строка выводится целиком. Строка выравнивается по правому краю отведенного поля.

Например.

```
char s[10];  
printf ("%10s", s);
```

char * gets(char *s);

Функция *gets* читает символы с клавиатуры до появления символа новой строки и помещает их в строку *s* (сам символ новой строки в строку не включается, вместо него в строку заносится нуль-символ). Функция возвращает указатель на строку *s*, а в случае возникновения ошибки или конца файла- *NULL*.

Если количество вводимых символов превышает заданную длину строки, то ответственность лежит на программисте.

puts

```
int puts (const char *s);
```

Функция *puts* выводит строку *s* на стандартное устройство вывода, при этом завершающий нуль заменяется символом новой строки. Если вывод успешный, то возвращается неотрицательное значение. Если возникла ошибка при выводе - возвращается *EOF*.

Например. Ввести строку с клавиатуры и вывести ее на экран.

```
# include < stdio.h >  
void main () {  
    char s[80];  
    gets (s); //ввод строки  
    puts (s); //вывод строки  
}
```

При работе со строками их можно обрабатывать как массивы символов или применять специальные функции.

Пример

Присвоить строке *s* значение строки *t* и вывести ее на экран.

Решение

В цикле будем присваивать посимвольно элементам массива *s* значения соответствующих элементов массива *t*. Цикл завершится, когда закончится строка *t*, а именно, когда будет скопирован нуль-символ.

Программа

```
#include "stdafx.h"

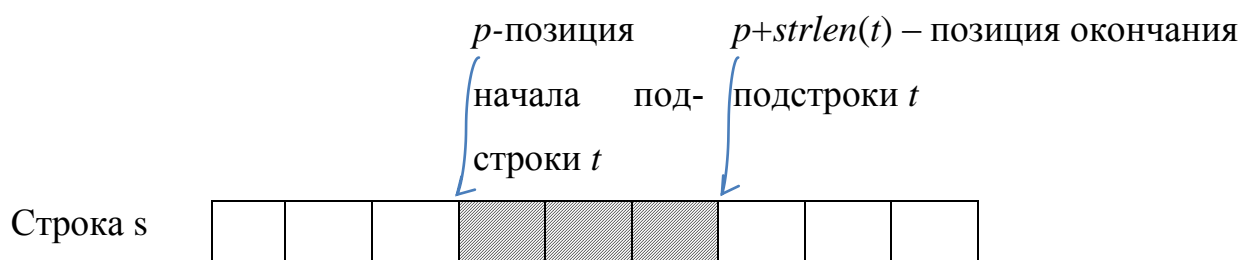
int _tmain(int argc, _TCHAR* argv[]){
    char s[10] , t[10]="abcd" ;
    int i=0 ;
    while ((s[i] = t[i]) != '\0' )
        i ++;
    puts (s); //вывод строки
}
```

Пример

Удалить из строки *s* длиной 80 символов все вхождения подстроки *t* длиной 10 символов и вывести ее на экран. Строки *s* и *t* вводить с клавиатуры.

Решение

Пока значение позиции *p*, с которой в строке *s* начинается подстрока *t*, не окажется равной *NULL*, будем удалять из строки *s* подстроку *t*. Для поиска подстроки воспользуемся функцией *strstr*. Подстроку *t* можно удалить, скопировав при помощи функции *strcpy* в позицию *p* символы строки *s*, которые располагаются после подстроки *t*.



Программа

```
#include "stdafx.h"
#include <string.h>
int _tmain(int argc, _TCHAR* argv[])
{
    char s[80], t[10], *p;
    scanf_s("%80s", s, 80);
    scanf_s("%10s", t, 10);
    while (( p=strstr(s,t))!= NULL )
        strcpy_s(p, 80, p+strlen(t));
    return 0;
}
```

Функции работы со строками и символами

Для работы со строками предназначены функции, которые находятся в заголовочных файлах *string.h* и *cstring.h*.

Функция	Пояснение
Копирование строк	
char * strcpy (char * S1, char * S2);	Копирует S2 в S1 и возвращает S1. Если строки перекрываются, то результат неопределен.
char *strncpy(char * S1, const char * S2, size_t n);	копирует до n байт строки из S2 в S1

Длина строки	
Size_t strlen (char * S);	Возвращает длину строки (без учета символа завершения строки).
Конкатенация строк	
char * strcat (char * S1, char * S2);	Добавляет S2 к S1 и возвращает S1. В конец результирующей строки добавляется ноль - символ.,
char *strncat(char * S1, const char * S2, size_t n);	дописывает не более n начальных символов строки S2 (или всю S2, если ее длина меньше) в конец S1. Результат сохраняется в S1
Поиск символа в строке.	
char * strchr (char * S, int ch);	Возвращает указатель на первое вхождение символа ch в строку S, если его в строке нет, то возвращается NULL.
char *strrchr(const char *S, int c);	то же, что strchr, но находит последнее вхождение символа c в строку S.
Поиск подстроки в строке	
char * strstr (char * S1, char * S2);	Возвращает указатель на элемент из S1, с которого начинается S2 или NULL, если элемент не найден.
Сравнение строк	
int strcmp(const char *S1, const char*S2);	сравнивает две строки. Возвращает целое меньше нуля, если S1 < S2, равное нулю, если S1 == S2, и большее нуля, если S1 > S2. С учётом регистра
int stricmp(const char *S1, const char*S2);	сравнивает строку S1 со строкой S2 и возвращает результат типа int: 0 –если строки эквивалентны, >0 – если S1<S2, <0 — если S1>S2. Без учёта регистра

int strncmp(const char *S1, const char*S2);	сравнивает n символов строки S1 со строкой S2 и возвращает результат типа int: 0 –если строки эквивалентны, >0 – если S1<S2, <0 — если S1>S2. С учётом регистра
int strnicmp(const char *S1, const char*S2);	сравнивает n символов строки S1 со строкой S2 и возвращает результат типа int: 0 –если строки эквивалентны, >0 – если S1<S2, <0 — если S1>S2 Без учёта регистра
Функции поиска	
size_t strspn(const char *S1, const char *S2);	возвращает длину начального сегмента строки S1, содержащего только те символы, которые входят в строку S2
size_t strcspn(const char *s, const char *reject);	возвращает длину начального сегмента строки S1, содержащего только те символы, которые не входят в строку S2
char *strpbrk(const char *S1, const char *S2);	Возвращает указатель первого вхождения любого символа строки S2 в строке S1
char *strtok(char *S1, const char *S2);	сканирует первую строку в поисках первого участка, не содержащего символов из S2. Первый вызов функции возвращает указатель на начало первого участка и записывает 0 в S1 сразу после конца участка. Последующие вызовы с 0 в качестве 1-го аргумента обрабатывают строку дальше, пока еще есть такие участки. Если их нет, возвращается 0. Функцию применяют для выделения слов из предложения S1. В строке S2 находятся символы-разделители

Функции преобразования <stdlib.h>	
<code>double atof (const char *s);</code>	Преобразует строку S в тип <code>double</code> , если строка содержит недопустимое значение, то возвращаемое значение не определено
<code>int atoi (const char *S);</code>	Преобразует строку S в значение типа <code>int</code> , если строка содержит недопустимое значение, то возвращаемое значение не определено
<code>long int atol (const char *S);</code>	преобразует строку S в тип <code>long int</code>
Обработка символов <ctype.h>	
<code>isalnum(c)</code>	возвращает значение <code>true</code> , если c является буквой или цифрой, и <code>false</code> в других случаях
<code>isalpha(c)</code>	возвращает значение <code>true</code> , если c является буквой, и <code>false</code> в других случаях
<code>isdigit(c)</code>	возвращает значение <code>true</code> , если c является цифрой, и <code>false</code> в других случаях
<code>islower(c)</code>	возвращает значение <code>true</code> , если c является буквой нижнего регистра, и <code>false</code> в других случаях
<code>isupper(c)</code>	возвращает значение <code>true</code> , если c является буквой верхнего регистра, и <code>false</code> в других случаях
<code>isspace(c)</code>	возвращает значение <code>true</code> , если c является пробелом, и <code>false</code> в других случаях
<code>toupper(c)</code>	если символ c, является символом нижнего регистра, то функция возвращает преобразованный символ c в верхнем регистре, иначе символ возвращается без изменений.

Функции стандартной библиотеки ввода/вывода <stdio.h>	
getchar(c)	считывает символ с со стандартного потока ввода, возвращает символ в формате int
gets(s)	считывает поток символов со стандартного устройства ввода в строку s до тех пор, пока не будет нажата клавиша ENTER

УКАЗАТЕЛЬ НА ФУНКЦИЮ

Имя функции – это указатель на функцию или адрес первого байта выполняемого кода функции. Указатель на функцию содержит адрес в кодовом сегменте, по которому располагается исполняемый код функции (т.е. адрес, по которому передается управление при вызове функции).

В общем виде указатель на функцию объявляется так:

тип (* имя) ();

Имя - это указатель на функцию, которая возвращает значение типа **тип**.

Например:

void (pf1) ();* *pf1* – указатель на функцию без параметров, которая не возвращает значения.

int (pf2) (float r);* *pf2* - указатель на функцию с одним параметром типа *float*, которая возвращает значение типа *int*..

При помощи указателей на функции можно одни функции передавать как параметры в другие функции.

Пример

Написать функцию, выводящую на экран значение заданной функции с двумя целочисленными параметрами, возвращающей в вызывающую программу значение целого типа. Вызвать эту функцию из программы *main* для двух различных функций.

Программа

```
#include <stdio.h>

//объявили функции с двумя целочисленными параметрами,
//возвращающие в вызывающую программу значение целого типа.
int SUM (int a, int b) {return a + b;}
int PROD (int a, int b) {return a * b; }

void PRINT (int (* pf) (int, int ), int x, int y)//
    // pf - указатель на функцию
{PRINT f ("результат = % d", pf (x, y)) ;}
void main ( ) {
    int m, n;
    PRINT (SUM, m, n); // вызов PRINT для печати суммы аргументов
    PRINT (PROD, 20, 132); // вызов PRINT для печати произведения аргу-
ментов
}
```

Предложение *typedef*

При помощи спецификатора *typedef* можно определить новый тип (дать ему имя). Предложение *typedef* имеет вид:

typedef тип новое имя [размерность];

Например,

```
typedef unsigned short int USINT;
```

определяет *USINT* как тип короткого целого без знака. Теперь можно использовать это имя для объявления новых переменных:

```
UINT k, t;
```

Пример. Объявить тип *MAS* массива, состоящего из 100 значений типа *float*:

```
typedef float MAS[100];
```

Пример. Объявить тип указателя на функцию с одним параметром целого типа, которая не возвращает значения в вызывающую программу. Создать массив из 4 указателей такого типа.

```
typedef void A (int);  
A mas[4];
```

Возврат указателя на функцию в качестве результата

Прототип такой функции должен иметь следующий вид

тип (* имя (список1)) (список2);

Здесь объявляется функция **имя** со списком формальных параметров **список1**, которая возвращает указатель на функцию со списком формальных параметром **список2**, возвращающую значение типа **тип**.

Пример.

Написать функцию vf с одним параметром символьного типа, которая возвращает указатель на функцию от двух целочисленных аргументов. Вызвать эту функцию из функции *main*.

Программа.

```
# include <stdio.h>  
  
int f1 (int a, int b) {return a * b;}  
int f2 (int a, int b) {return a+b;}  
int f3 (int a, int b) {return a-b;}  
  
// vf- функция, которая возвращает указатель на функцию, выполняющую  
операцию op.  
  
int (* vf (char op)) (int ,int ) {  
    switch(op){  
        case '*': return f1;  
        case '+': return f2;  
        case '-': return f3;  
    }  
}  
  
void main ( )  
{
```

```
printf ("результат = % d",  vf('*')(8,4));  
}
```

2. Практическая часть

Задания

1. Написать собственный вариант стандартной функции. Сравнить результаты выполнения собственной функции и стандартной функции.
 - 1.1. *strchr*
 - 1.2. *strstr*
 - 1.3. *strlen*
 - 1.4. *strcmp*
 - 1.5. *strcat*
 - 1.6. *strncat*
 - 1.7. *strpbrk*
 - 1.8. *strncmp*
 - 1.9. *strncat*
 - 1.10. *strrchr*
2. Дана строка длиной 80 символов. Словом называется последовательность непробелов, окруженная пробелами. Написать функцию, которая
 - 2.1. Вставляет слово «номер» перед словами, состоящими из цифр.
 - 2.2. Переносит первое слова в конец предложения.
 - 2.3. Переносит последнее слово в начало предложения.
 - 2.4. Удаляет все символы между скобками.
 - 2.5. Все слова, которые начинаются с буквы «а», переводит в верхний регистр.
 - 2.6. Вычисляет сумму чисел, которые встречаются в предложении.
 - 2.7. Находит самое длинное слово в предложении.
 - 2.8. Находит самое короткое слово в предложении.
 - 2.9. Все слова, которые заканчиваются на букву «а», начинает с прописной буквы.

2.10. Выводит на экран все слова, содержащие цифры.

2.11. Подсчитывает количество слов в строке.

2.12. Удаляет лишние пробелы между словами.

2.13. Удаляет все слова, в которых встречается заданный символ

3. Написать функцию, которая возвращает значение выражения

3.1. $f(x)=2*x;$

3.2. $f(x)=5+12*x;$

3.3. $f(x)=7-x;$

3.4. $f(x)=5*x-32;$

3.5. $f(x)=7*(1-x);$

3.6. $f(x)=15-4*x;$

3.7. $f(x)=(5-x)/x;$

где x – целочисленная переменная.

4. Написать функцию *Function*, которая возвращает в вызывающую программу значение выражения

4.1. $f1(a)+f2(a);$

4.2. $f1(a)-f2(a);$

4.3. $f1(a)*f2(a);$

4.4. $f1(a)/f2(a);$

4.5. $(f1(a)+f2(a))/f1(a);$

4.6. $1/(f1(a)+f2(a));$

Здесь $f1(a)$ и $f2(a)$ – это функции с одним параметром целого типа, которые возвращают целочисленное значение (функции из пункта 1). Передавать эти функции в *Function* при помощи указателя на функцию. Написать функцию *main*, которая печатает результаты функции *Function*.

5. Объявить тип указателя *TF* на функцию с одним параметром целого типа, которая возвращает целочисленное значение. Изменить заголовок функции *Function*: передавать функции при помощи параметра типа *TF*. Написать функцию *main*, которая печатает результаты функции *Function*.

6. Написать программу, которая реализует меню из 4 функций, которые выполняют следующие действия:

6.1. Возведение a в степень b .

6.2. Умножение a на b .

6.3. Получение остатка от деления a на b .

6.4. Проверка кратности a и b .

Здесь a и b – положительные целые числа.

3. Порядок выполнения и сдачи работы

1. Изучить теоретическую и практическую часть.

2. Сдать преподавателю теорию работы путем ответа на контрольные вопросы.

3. Получить вариант задания и выполнить задание в Visual Studio 2013.

4. Предъявить преподавателю результат выполнения задания для самостоятельной работы

Оформить отчет.

Содержание отчета

Отчет готовится в электронном виде и распечатывается. Он должен содержать текст задания, скриншот программного кода и скриншот результата .

4. Контрольные вопросы

1. Что такое строка?

2. Что такое нуль-символ?

3. Как определяется текущая длина строки?

4. Как выполняется копирование строк?

5. Как выполняется инициализация строк?

6. Как строки передаются в функцию?

7. В каком заголовочном файле содержится функция для обработки строк?

8. Как выполняется объявление указателя на функцию?

9. Как передать в функцию другую функцию в виде параметра?
10. Как объявить тип указателя на функцию?
11. Как вернуть в качестве результата указатель на функцию?
12. Как выполнить косвенный вызов функции?

ЛАБОРАТОРНАЯ РАБОТА № 5

СТРУКТУРЫ

Цель работы. Работа со структурами.

1. Теоретическая часть

Структура – это совокупность переменных, которые объединяются под одним именем. Элементы структуры называются полями. Поля могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на тип этой же структуры. В общем случае описание структуры имеет вид:

```
struct имя типа {  
    тип1 элемент 1;  
    тип2 элемент 2;  
    ...  
    тип3 элемент 3;} список описателей;
```

Можно опустить имя типа или список описателей, но нельзя опустить то и другое одновременно. Например,

```
struct zap {  
    int a;  
    char b;  
    float c;};
```

определяет тип структуры с именем *zap*. Далее можно определить переменные типа структура. Например,

```
struct zap x, y[10], *z;
```

объявляет переменную *x* типа структура, массив структур *y* и указатель *z* на структуру.

Инициализация структуры

Значения элементов структуры при инициализации перечисляют в фигурных скобках в порядке их описания. Например,

```
struct zap {  
    int a;  
    char b;  
    float c;} x={10, 'a', 3.14};
```

При инициализации массивов структур в фигурные скобки заключается каждый элемент массива. Например,

```
struct {  
    int a;  
    char b;} y[3]={{1, 'a'}, {2, 'b'}, {3, 'c'}};
```

Доступ к полям структуры

При обращении к полю структуры применяется операция выбора:

Имя переменной.имя поля

```
struct zap {  
    int a;  
    char b;  
    float c;} x;  
x.a=10;  
x.b='l';  
x.c=7.67;
```

При обращении к полю через указатель используется знак ->.

```
struct zap {  
    int a;  
    char b;  
    float c;} *z;  
z->a=100;  
z->b='8';  
z->c=78.45;
```

Действия над структурами

Для переменных одного и того же структурного типа определена операция присваивания. При этом происходит поэлементное копирование.

Структуры нельзя сравнивать на равенство или неравенство.

К структуре можно применять операцию вычисления адреса.

Структуру можно передавать в функцию и возвращать в качестве результата функции.

Размер структуры может быть не равен сумме размеров ее элементов (т.к. элементы могут быть выровнены по границу слова).

2. Практическая часть

Задания

1. Создать и проинициализировать массив структур, состоящих из трех полей:

- Название книги.
- Автор.
- Год издания.

Вывести на экран

1.1.Количество книг, изданных в указанном году.

1.2.Количество книг указанного автора.

1.3.Год издания указанной книги.

2. Создать массив структур, состоящих из трех полей:

- Название детали.
- Цена детали.
- Количество.

Размер массива – 5 элементов. Заполнить массив значениями, вводимыми с клавиатуры. Вывести на экран

2.1.Название самой дорогой детали.

2.2.Цену указанной детали.

2.3.Количество штук указанной детали.

3. Объявить массив *Mas*, состоящий из 5 элементов типа структура вида:

3.1.Марка машины.

3.2.Пробег.

3.3.Государственный номер.

Написать функцию Vs , которая заполняет поля структуры значениями, вводимыми с клавиатуры, и возвращает структуру в вызывающую программу. Заполнить массив Mas при помощи функции Vs и вывести его на экран.

Структура с адресами полей

4. Объявить тип структура T вида

- Название страны
- Столица страны
- Численность населения страны

Создать динамический массив Mas из n элементов типа T . Заполнить массив значениями, вводимыми с клавиатуры. Вывести на экран название столицы указанной страны.

5. Объявить тип структура T вида

- Указатель на строку.
- Указатель на целое.

Создать массив Mas из 4 элементов типа T . Заполнить массив значениями, вводимыми с клавиатуры. Вывести на экран полученный массив.

6. Объявить тип структура T вида

- Фамилия.
- Возраст.

Создать массив Mas из 4 указателей на структуры типа T . Заполнить массив значениями, вводимыми с клавиатуры. Вывести на экран полученный массив.

3. Порядок выполнения и сдачи работы

1. Изучить теоретическую и практическую часть.
2. Сдать преподавателю теорию работы путем ответа на контрольные вопросы.
3. Получить вариант задания и выполнить задание в Visual Studio 2013.

4. Предъявить преподавателю результат выполнения задания для самостоятельной работы

Оформить отчет.

Содержание отчета

Отчет готовится в электронном виде и распечатывается. Он должен содержать текст задания, скриншот программного кода и скриншот результата .

4. Контрольные вопросы

1. Для чего предназначена структура?
2. Каким может быть тип поля структуры?
3. Чем отличаются структура и объединение?
4. Как обратиться к полю структуры?
5. Как выполняется инициализация структуры?

ЛАБОРАТОРНАЯ РАБОТА № 6 СТАНДАРТНЫЙ ВВОД-ВЫВОД

Цель работы. Организация ввода данных с клавиатуры и вывода на экран.

1. Теоретическая часть

Программа на С рассматривает ввод и вывод как некоторый поток данных. Источником потока при вводе может быть файл или устройство ввода (клавиатура). Местом назначения потока при выводе может быть файл или устройство отображения (экран).

Стандартные файлы

При работе на языке С автоматически открываются 3 файла.

stdin – стандартный ввод (клавиатура).

stdout – стандартный вывод.

stderr – стандартный вывод ошибок.

Ввод-вывод буферизован. Буфер – это промежуточная область памяти. Информация поступает из файла в буфер порциями по 512 байтов и более, а программа обращается за данными к буферу. Это ускоряет работу программы

Функции символьного небуферизованного ввода-вывода

```
int getchar (void);
```

Возвращает очередной символ в форме *int* из стандартного ввода. Если символ не может быть прочитан – возвращается значение *EOF*.

```
int putchar (int ch);
```

Выводит символ *ch* на стандартное устройство вывода. Если вывод успешный, то возвращается значение *ch*, иначе – *EOF*.

Эти функции содержатся в заголовочном файле *conio.h*/

Функции форматированного ввода-вывода

```
int scanf(const char * fmt,...);
```

Первый параметр – это строка формата. Второй параметр – это список переменных (адреса). Функция возвращает количество тех элементов данных, которым были присвоены значения.

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

Первый параметр – это строка формата. Второй параметр – это список выражений.

Функция возвращает количество выведенных символов или отрицательное число в случае ошибки.

Спецификации формата

Специ- фика- ция	Пояснение
<i>c</i>	Аргумент рассматривается как отдельный символ
<i>d, i</i>	Аргумент преобразуется к десятичному виду

<i>e, E</i> (вывод в экспо- ненци- альной форме)	Аргумент, который рассматривается как переменная типа <u>float</u> или <u>double</u> , преобразуется в десятичную форму в виде [-] <i>m.nnnnnne e</i> [+ -] <i>xx</i> <i>m</i> – мантисса <i>xx</i> - порядок <i>nnnnnn</i> – дробная часть мантиссы по умолчанию точность – 6 знаков после запятой
<i>f</i>	Аргумент, который рассматривается как переменная типа <u>float</u> или <u>double</u> , преобразуется в десятичную форму в виде [-] <i>mmm.nnnn</i> Точность по умолчанию = 6
<i>g, G</i>	Используется тот из форматов <i>%e</i> или <i>%f</i> , который короче Назначающие нули не печатаются
<i>O</i>	Аргумент преобразуется в беззнаковую восьмеричную форму (без лидирующего нуля)
<i>p</i>	Вывод указателя в шестнадцатеричном формате (в стандарт не входит, но есть во всех реализациях)
<i>s</i>	Аргумент является строкой; символы строки печатаются, пока не будет достигнут нулевой символ или не будет напечатано количество символов, указанное в спецификации точности.
<i>u</i>	Аргумент преобразуется в беззнаковую десятичную форму
<i>x ;X</i>	Аргумент преобразуется в беззнаковую шестнадцатеричную форму (без лидирующих 0X)
<i>%</i>	Выводится символ %

Модификаторы формата

Применяются для управления шириной поля, которое отводится для размещения значения. Модификаторы – это одно или два числа: Первое – задаёт *min* количество позиции, которое отводится под число. Второе определяет, сколько из них отводится под дробную часть. Если указанного количества позиции недостаточно, то автоматически выделяется большее число позиций.

% - *min C* или % *min C*

% - *min. precision C* или % *min. precision C*

C – спецификация.

min - минимальная ширина поля.

precision зависит от C:

- при выводе строки (% *s*) – это max число символов для вывода

- при выводе вещественного числа (%*f* или %*e*) – это количество цифр после десятичной точки.

- при выводе целого числа (%*d* или %*i*) – это min количество выводимых цифр.

Если число выводится меньшим числом цифр – выводится начальные нули.

- при выводе вещественного числа (% *d* или %*G*) – это max количество значащих цифр, которые будут выводиться.

Символ “-” (минус) указывает на то, что значения выравниваются по левому краю и дополняются при необходимости пробелами справа.

Если минуса нет, то значения выравниваются по правому краю и дополняются пробелами слева.

Работа с буферизованным вводом

При вводе информации может возникнуть необходимость очистки буфера в следующих случаях:

- При вводе символа при помощи функции *getchar()*. Если завершить ввод нажатием клавиши *Enter*, то в буфер заносится символ конца строки, который и будет прочитан при очередном выполнении функции *getchar()*.

- При вводе информации поочередно при помощи функций *scanf()* и *getchar()*. Эти функции по-разному обрабатывают пробельные символы. *getchar()* считывает каждый символ, а *scanf()* пропускает при вводе пробельные символы.

В таких случаях следует пропускать символы входной строки, включая и символ новой строки, при помощи цикла *while*:

```
while(getchar() != '\n');
```

или

```
while(getchar() != '\n')
```


Информация об обнаружении конца файла возвращается в программу при помощи специального значения *EOF*. Это значение возвращается такими функциями как *getchar()* или *scanf()*.

Пример.

Чтение с клавиатуры до достижения конца ввода. Конец ввода моделируется при нажатии *Ctrl+Z*.

Программа

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    char ch;
    while((ch=getchar())!=EOF)
        putchar(ch);
    return 0;
}
```

Верификация данных – проверка допустимости входных данных

При выполнении программ, требующих ввода информации с клавиатуры необходима проверка допустимости входных данных во избежание ошибок в ходе выполнения программы.

Например, можно проверить, не выходит ли введенное значение за границы интервала допустимых значений и т.д.

Часто ошибки связаны с тем, что вводимые значения не соответствуют типам переменных, которым они должны быть присвоены. В таких случаях в программе надо обеспечить проверку значения, возвращаемого функцией *scanf()*. Эта функция возвращает количество значений, которые были успешно прочитаны. В случаях, когда это значение не соответствует ожидаемому, можно прекратить работу программы с выдачей сообщения об ошибке, или организовать цикл, повторяя ввод, до тех пор, пока не будет введено корректное значение.

Пример.

Ввести с клавиатуры целое число. При вводе нечислового значения пропускаются все символы до конца строки и ожидается ввод с начала новой строки.

```
int x, k;  
while ((k = scanf("%d", &x)) != 1)  
    while(getchar() != '\n');
```

ФАЙЛОВЫЙ ВВОД-ВЫВОД

Существует 2 представления файлов: текстовое и двоичное. В двоичном представлении каждый байт доступен из программы. В текстовом представлении распознаются и правильно обрабатываются такие элементы как конец. Обычно текстовое представление используется для текстовых файлов, двоичное – для двоичных, но не обязательно.

Файл становится доступным для работы после процедуры открытия.

Открытие файла

```
errno_t fopen_s (FILE *file, const char * fname, const char *  
fmode);
```

FILE это предопределенная структура, которая содержит информацию, необходимую для работы с файлом.

fname – строка, содержащая имя файла, или адрес строки, которая содержит имя файла.

fmode – строка, которая определяет режим, в котором надо открыть файл.

Режимы открытия:

“r” – открыть файл для чтения.

“w” – открыть файл для записи. Если файл существует, то он усекается до нуля. Если файл не существует, то он создается.

“a” - открыть файл для записи. Данные добавляются в конец файла. Если файл не существует, то он создается.

“r+” - открыть файл для обновления (для чтения и записи).

“w+” – открыть файл для обновления (для чтения и записи). Если файл существует, то он усекается до нуля. Если файл не существует, то он создается.

“a+” – открыть файл для обновления (для чтения и записи). Данные добавляются в конец файла. Читать можно весь файл. Если файл не существует, то он создается.

Функция *fopen_s* создает указатель на структуру, содержащую информацию о файле, если файл успешно открыт.

Функция *fopen_s* возвращает 0, если файл успешно открыт, или ненулевое значение в противном случае.

Закрытие файла

Файл закрывается либо при завершении программ, либо с помощью функции *fclose*.

```
int fclose ( FILE * );
```

Функция возвращает значение 0, если файл закрыт успешно, и *EOF* в противном случае. Перед закрытием потока информация из связанных с ним буферов выгружается на диск. Поэтому, чтобы избежать потерь информации, рекомендуется явным образом закрывать потоки.

Функции форматированного ввода-вывода

Эти функции отличаются от функций *scanf_s* и *printf* только наличием первого аргумента для идентификации файла. Первый параметр – указатель файла. Второй параметр – это строка формата. Третий параметр – это список выражений.

```
int fscanf_s( FILE *stream, const char *fmt,... );
```

Функция возвращает количество тех элементов данных, которым были присвоены значения.

```
int fprintf ( FILE *stream, const char *fmt,... );
```

Функция возвращает количество выведенных символов или отрицательное число в случае ошибки.

Пример

Создать текстовый файл с именем «a1», содержащий из 10 целых чисел, разделенных пробелами.

Программа

```
#include "stdafx.h"
```

```

FILE * f;
/*
f – это файловая переменная, указатель на предопределенную структуру FILE, со-
держащую информацию о файле
*/
int n, i, k;
void main(){
fopen_s( &f, "a1", "w"); // открыли файл для записи
                        //(создали новый файл с именем «a1»)
for (i=0;i<10;i++){
    scanf_s( "%d", &n); // ввели с клавиатуры значение n
    fprintf(f, "%d ", n); // записали значение n в файл
}
fclose(f); // закрыли файл
}

```

Признак конца файла

При чтении из файла программа должна остановиться при достижении конца файла. Она обнаруживает, что конец файла достигнут, только после попытки чтения за пределами файла.

```
int feof (FILE*f );
```

Функция *feof* используется для определения конца файла и возвращает значения $\neq 0$ или *EOF*, если достигнут конец файла, иначе возвращается 0.

Чтобы избежать проблемы чтения пустого файла, при вводе необходимо использовать цикл с предусловием. Чтение из файла надо выполнять до входа в тело цикла.

Пример.

Вывести на экран файл «a1», содержащий целые числа.

Программа.

```
#include "stdafx.h"
```

```

FILE * f;
int n, i;
void main(){
    if (fopen_s( &f, "a1", "r")) printf("\n ошибка открытия");
    else {
        fscanf_s(f, "%d", &n); // выполняется пробное чтение из файла
        while(!feof(f)){ // пока не достигнут конец файла
            printf("%d\n", n); // выводится на экран прочитанное значение
            fscanf_s(f, "%d", &n); } // выполняется чтение из файла нового значения
        fclose(f); //файл закрывается
    }
}

```

ДВОИЧНЫЙ ВВОД-ВЫВОД

Цель работы. Организация произвольного доступа к файлу.

1. Теоретическая часть

При двоичном вводе-выводе информация выводится в файл в ее внутреннем представлении. Прочитать такой файл при помощи текстового редактора нельзя. Для выполнения двоичного ввода-вывода предназначены функции *fread* и *fwrite*.

size_t fread (void *buffer, size_t size, size_t count, FILE *stream);

Функция считывает *count* элементов *size* байтов в область, заданную указателем *buffer*, из потока *stream*. Функция возвращает количество прочитанных элементов, которое может быть меньше *count*, если при чтении произошла ошибка или встретился конец файла.

size_t fwrite (const void *p, size_t size, size_t n, FILE *f);

Функция записывает *n* элементов длиной *size* байт из буфера, заданного указателем *p*, в поток *f*. Возвращает число записанных элементов.

Пример.

Создать двоичный файл с именем «a1» и записать в него три переменные типа структура.

```
#include "stdafx.h"
FILE * f;
struct zap {
    char name[10];
    int n;} z; // z- переменная типа структура
int i;
void main(){
    fopen_s( &f, "a1", "w");
    for (i=0;i<3;i++){
        scanf_s( "%9s%d", z.name, 10, &z.n);
        /*ввод с клавиатуры значений  в поля структуры z */
        fwrite( &z, sizeof(z), 1, f);
        /*вывод структуры z в файл в двоичном виде – выполняется вывод в файл f
        одной записи) размера sizeof(z из области по адресу &z*/
    }
    fclose(f);}
```

Пример.

Вывод на экран информации из двоичного файла

```
#include "stdafx.h"
FILE * f;
struct zap {
    char name[10];
    int n;} z;
int i, k;
void main(){
    if(!(fopen_s( &f, "a1", "r"))){
        fread( &z, sizeof(z), 1, f);
```

```

/* пробное чтение из файла f одной записи размера sizeof(z) в область по
адресу &z (ввод в переменную z)/
while(!feof(f)){ //пока не конец файла
    printf("%9s %d\n", z.name, z.n);
    /*вывод полей структуры z на экран */
    fread(&z, sizeof(z), 1, f)
    /* чтение из файла f одной записи размера sizeof(z) в область по адре-
су &z*/
}
fclose(f);}

}

}

```

Произвольный доступ к файлу.

Операции ввода/вывода выполняются, начиная с текущей позиции потока. Ее определяет указатель потока. Указатель изменяется автоматически после каждой операции ввода/вывода. При открытии файла указатель автоматически указывает на начало или конец файла. При изменении направления движения информации необходимо явным образом устанавливать позицию указателя файла.

Позицию файла можно считывать и устанавливать при помощи специальных функций. Эти функции нельзя использовать со стандартными потоками.

long int ftell (FILE *f);

ftell возвращает текущую позицию указателя в файле.

int fseek (FILE *f, long off, int org);

fseek перемещает текущую позицию в файле, связанном с потоком *f*, на новую позицию *off*, отсчитываемую от значения *org*, которое должно быть равно одной из 3х констант, определенных в *stdio.h*:

SEEK_CUR – от текущей позиции указателя 1

SEEK_END – от конца файла 2

SEEK_SET – от начала файла 0

int fgetpos (file *F, fpos_t *POS);

fgetpos возвращает текущую позицию в файле , связанном с потоком *f*, и копирует значение по адресу *pos* . Возвращаемое значение имеет тип *fpos_t*

```
int fsetpos (FILE *f, const fpos_t *pos);
```

fsetpos перемещает текущую позицию в файле, связанном с потоком *f*, на позицию **pos*, предварительно полученную с помощью функции *fgetpos*. Возвращает 0 при успешном завершении и ненулевое значение в противном случае.

Пример.

Изменение значения поля *name* в 3-й записи файла «a1»

```
#include "stdafx.h"

FILE * f;

struct zap {
    char name[10];
    int n;} z;

int i,k;

void main(){
if (!(fopen_s(&f, "a1", "r+"))){
    /*открыли файл для чтения и записи */
    k=fseek(f,3*sizeof(z),SEEK_SET);
    /*установили указатель файла на третью запись от начала файла */
    if (k==0){
        /* если удалось установить указатель */
        fread(&z,sizeof(z),1,f);
        /* прочитать из файла информацию в структуру z */
        printf("%9s %d\n",z.name,z.n);
        /* вывести z на экран */
        printf("\nname: ");
        scanf_s("%5s",z.name,10);
        /* ввод нового значения в поле name структуры z */
        fseek(f,3*sizeof(z),SEEK_SET);
        /* установка указателя на третью запись */
```



```

    fwrite(&z, sizeof(z), 1, f);
    /* вывод в файл структуры z */
    }
fclose(f);
}

```

2. Практическая часть

Задания.

1. Написать функцию для угадывания числа. Загаданное число передавать в функцию в виде параметра.

Функция должна выдавать подсказки вида:

- Загаданное число больше введенного.
- Загаданное число меньше введенного.
- Введено не число.

Предусмотреть возможность отказа от игры.. Возвращать 1, если число угадано, и 0 – в противном случае.

2. Описать структуру T , содержащую следующие сведения:

- фамилия водителя,
- категория,
- стаж.

Создать массив, содержащий N элементов типа T . Для заполнения полей структуры создать функцию. Выполнять верификацию данных при вводе. Вывести на экран созданный массив в виде таблицы.

3. Описать структуру T , содержащую следующие сведения:

- Название детали,
- Цена,
- Вес,

Создать массив, содержащий 5 элементов типа T и проинициализировать его.

Написать функцию P , которая выводит на экран значение типа структура T .

При выводе

- 3.1.выполнять выравнивание по левой границе, ширина поля вывода для строковых данных 10 знаков, ширина поля вывода для целочисленных данных 5 знаков, ширина поля вывода для вещественных данных 6 знаков, количество знаков после запятой для вещественных чисел -1, отображать незначащие нули,
- 3.2.выполнять выравнивание по правой границе, ширина поля вывода для строковых данных 5 знаков, ширина поля вывода для целочисленных данных 3 знаков, ширина поля вывода для вещественных данных 5 знаков, количество знаков после запятой для вещественных чисел -2 отображать незначащие нули,
- 3.3.выполнять выравнивание по левой границе, ширина поля вывода для строковых данных 10 знаков, ширина поля вывода для целочисленных данных 6 знаков, ширина поля вывода для вещественных данных 5 знаков, количество знаков после запятой для вещественных чисел -3, не отображать незначащие нули,
- 3.4.выполнять выравнивание по левой границе, ширина поля вывода для строковых данных 5 знаков, ширина поля вывода для целочисленных данных 6 знаков, ширина поля вывода для вещественных данных 5 знаков, количество знаков после запятой для вещественных чисел -4, не отображать незначащие нули.

Вывести массив на экран при помощи функции *P*.

4. Описать структуру *T*, содержащую следующие сведения: фамилия автора, название книги, издательство, год издания. Написать функции, выполняющие следующие действия:
 - 4.1.Создание двоичного файла *FB.txt*, содержащего *N* элементов типа *T*. Функция возвращает 0 при успешном завершении и -1 в противном случае.
 - 4.2.Вывод на экран содержимого файла *FB.txt*.. Функция возвращает в вызывающую программу количество записей через параметр *k*, а также 0 при успешном завершении и -1 в противном случае.

4.3. Возврат в вызывающую программу следующего значения:

4.3.1. Количество книг указанного автора.

4.3.2. Количество книг указанного года издания.

4.3.3. Количество книг указанного издательства.

4.3.4. Количество книг с указанным названием.

4.4. Возвращение в вызывающую программу записи под номером n .

4.5. Изменение значения следующего поля для n -го элемента файла:

4.5.1. Фамилия автора.

4.5.2. Название книги.

4.5.3. Издательство.

4.5.4. Год издания.

3. Порядок выполнения и сдачи работы

1. Изучить теоретическую и практическую часть.
2. Сдать преподавателю теорию работы путем ответа на контрольные вопросы.
3. Получить вариант задания и выполнить задание в Visual Studio 2013.
4. Предъявить преподавателю результат выполнения задания для самостоятельной работы

Оформить отчет.

Содержание отчета

Отчет готовится в электронном виде и распечатывается. Он должен содержать текст задания, скриншот программного кода и скриншот результата .

4. Контрольные вопросы

1. Что такое поток?
2. Какие существуют стандартные потоки?
3. Что такое буферизация?
4. Когда применяется полностью буферизованный ввод/вывод?

5. Когда применяется построчный буферизованный ввод/вывод?
6. Что такое верификация ввода?
7. Какие существуют режимы открытия файла?
8. Как выполняется открытие файла?
9. Что такое признак конца файла?
10. Для чего используется функция закрытия файла?
11. Как выполняется форматированный ввод/вывод?
12. В каком виде выполняется вывод информации при использовании двоичного вывода в файл?
13. Какие функции применяются для двоичного ввода/вывода?
14. Какие функции применяются для позиционирования в файле?
15. Каким образом выполняется изменение направления движения информации при обращении к файлу?
16. Что означает константа SEEK-SET?

ЛАБОРАТОРНАЯ РАБОТА № 7

ДИРЕКТИВЫ ПРЕПРОЦЕССОРА

Цель работы. Применение макросов.

1. Теоретическая часть

Директивы препроцессора

Препроцессор – это первая фаза компиляции.

Инструкции препроцессора называются директивами. Директивы препроцессора начинаются с #.

#define – **именованные константы**.

#define макрос (аббревиатура) список замены (тело)

Имя макроса не содержит пробелов и должно включать только букв, цифры и знак подчеркивания. Первый символ не должен быть цифрой. В конце строки команды точка с запятой не ставится.

Если препроцессор находит в программе указанный макрос, то он замещает его списком замены.

Пример

```
#include <stdio.h>
#define A 10
#define C A+1
#define D1 "A-C"
#define F "\n x=%d y= %d D D1"
void main(){
int x,y,z;
char c;
x=A;      /* x=10 */
y=C;      /* y=11 */
printf(F,x,y); /* x=10 y=11 D D1 */
getchar(); }
```

Макросы-функции или макросы с параметром

Параметр и все выражение в списке замены рекомендуется обрамлять круглыми скобками, так как препроцессор не выполняет расчетов, он только выполняет предложенные подстановки. Например, пусть

```
#define sq(x) x*x
int a=2;
int y=sq(a+3);
```

После вычисления выражения получим неверный результат.

$y = sq(a+3) = a+3 * a+3 = 11.$

Возьмем в скобки параметр и все выражение в списке замены:

```
#define sq(x) ((x)*(x))
```

После вычисления выражения получим верный результат.

$y = sq(a+3) = ((a+3)*(a+3)) = 25.$

Директивы условной компиляции

Директивы условной компиляции дают возможность выборочного компилировать часть кода программы.

Это директивы ***#if***, ***#else***, ***#elif***, ***#endif***

#if **константное выражение**

Последовательность операторов

#endif

Если константное выражение истинно, то компилируется код, который находится между этим выражением и ***#endif***. Иначе этот промежуточный код опускается. В выражении могут быть только константы и ранее определенные идентификаторы, но не переменные.

#else работает как ***else***, т.е. дает альтернативу на тот случай, когда не выполняется условие ***if***.

#elif означает ***else if***, после ***#elif*** находится константное выражение. Если это выражение истинно, то компилируется находящийся за ним блок кода, и больше не проверяются никакие другие выражения ***elif***. Иначе проверяется следующий блок последовательности.

Каждая директива ***#endif***, ***#else*** или ***#elif*** относится к ближайшей директиве ***#if*** или ***#elif***.

Пример.

```
#include <stdio.h>
int x=10,y=20,z=30;
void main(){
#define T '+'
#define F '-'
#define A '*'
#define P 10
#define S +
enum {Z,Q='*'};
#if Q==T
z=x+y;
```

```

#elif  $Q == F$ 
 $z = x - y;$ 
#else
 $z = x * y;$ 
#endif
 $z = x \ S \ y;$ 
getchar();
}

```

Многофайловые проекты

Программы, разработанные для решения реальных задач, могут быть очень сложными. В процессе решения поставленная задача разбивается на подзадачи. Над отдельной подзадачей может трудиться свой коллектив разработчиков, создавая файл, содержащий функции для решения поставленной подзадачи. Таким образом, окончательный текст программы состоит из нескольких файлов. Каждый из таких файлов называется исходным модулем и имеет расширение *c* или *cpp*. Прототипы всех функций исходного файла выносят в отдельные файлы, которые называются заголовочными файлами.

Если функции одного модуля обращаются к функциям из другого модуля, то в вызывающий модуль включается директива *include* с именем заголовочного файла, содержащего прототипы вызываемых функций.

#include имя файла

Эта директива дает указание читать еще один исходный файл, в дополнение к тому, в котором находится сама эта директива. Имя исходного файла должно быть заключено в двойные кавычки или в угловые скобки. Например:

```

#include <stdio.h>
#include "stdio.h"

```

Если имя заключено в угловые скобки, то поиск ведется в специальном каталоге. Если имя заключено в кавычки – то поиск ведется сначала в текущем каталоге, а если его там нет, то в специальном каталоге.

Создание многофайлового проекта в среде Microsoft Visual studio

1. Создать файл с функцией ***main()***.
2. Включить дополнительные файлы ***.cpp***
 - 1.1.Щелкнуть по кнопке ***Add New Item..***
 - 1.2.Выбрать ***C++ File (.cpp)***
 - 1.3.Ввести имя файла ***.cpp*** и нажать кнопку ***Add***
3. Включить заголовочный файл ***.h***
 - 1.4.Щелкнуть по кнопке ***Add New Item..***
 - 1.5.Выбрать ***Header File (.h)***
 - 1.6.Ввести имя файла ***.h*** и нажать кнопку ***Add***

Пример:

Файл с именем ***mod.cpp*** содержит функцию ***fun***

```
int fun(int a,int b){  
    if (a>b)return a;  
    else return b;}  

```

Заголовочный файл с именем ***mod.h*** содержит заголовок функции ***fun:***

```
int fun(int ,int);  

```

Файл с именем ***prog.cpp*** содержит функцию ***main***, из которой вызывается функция ***max***, и директиву с именем заголовочного файла ***mod.h:***

```
#include <stdio.h>  
#include "mod.h"  
void main(){  
int x=10,y=7,z=0;  
    z=fun(x,y);  
    printf("\n fun(x,y)=%d, x=%d y=%d",z,x,y);  
}
```

КЛАССЫ ПАМЯТИ

Классы памяти определяют область видимости, связывание и продолжительность хранения.

Область видимости определяет участок программы, из которого можно получить доступ к конкретному идентификатору.

Продолжительность хранения бывает статическая и автоматическая. Статическая продолжительность означает, что переменная существует на всем протяжении времени выполнения программы. Автоматическая продолжительность означает, что при входе в блок память выделяется, при выходе из блока – освобождается.

Автоматические переменные: класс памяти *auto*

Любая переменная, объявленная в блоке или в заголовке функции, является автоматической по умолчанию, но можно использовать слово *auto* для явного объявления

Продолжительность хранения – автоматическая, Видимость – в пределах блока, это значит, что только в пределах этого блока к этой переменной можно обращаться по имени.

Такие переменные инициализируются только явно. Инициализирующие значения могут быть неконстантными выражениями, при условии, что переменные, входящие в выражение уже были ранее объявлены.

Регистровые переменные: класс памяти *register*

Регистровые переменные это автоматические переменные, которые желательно разместить в регистрах микропроцессора. Спецификатор *register* можно применять только к локальным переменным и к формальным параметрам функций.

Статические переменные с областью видимости в пределах блока:

класс памяти *static*

Продолжительность хранения – статическая, видимость – в пределах блока,

Такие переменные существуют на всем протяжении времени выполнения программы. Иначе говоря, это локальные переменные, которые сохраняют значения между вызовами функции.

Такие переменные инициализируются один раз во время компиляции функции. Если переменные не инициализируются явно, то им присваивается значение 0. Инициализирующие значения могут быть неконстантными выражениями, при условии, что переменные, входящие в выражение уже были ранее объявлены.

Статические переменные с внешним связыванием

Продолжительность хранения – статическая, видимость – в пределах файла, связывание – внешнее, это значит, что переменная может использоваться в любом месте многофайловой системы.

Внешняя переменная объявляется вне пределов всякой функции. Если переменная определена в другом файле, то внутри функции, использующей эту переменную, она объявляется с ключевым словом *extern*.

Внешние переменные неявно инициализируются нулями. При явной инициализации инициализирующие значения могут быть только константными выражениями.

Статические переменные с внутренним связыванием

Продолжительность хранения – статическая, видимость – в пределах файла, связывание – внутреннее, это значит, что переменная может использоваться только в пределах одного файла.

Такая переменная объявляется вне пределов всякой функции с ключевым словом *static* и может использоваться только в функциях одного и того же файла.

Классы памяти функции

Функция может быть внешней (по умолчанию или статической с квалификатором *static*).

К внешней функции имеют доступ функции из других файлов. При объявлении функции, определенной в другом файле, указывается ключевое слово *extern* для наглядности.

Статическая функция может использоваться только в файле, в котором она определена, поэтому в других файлах можно использовать функции с тем же именем.

2. Практическая часть

Задание

1. Написать программу, которая выводит на экран результат вычисления заданной функции. Описание функции сохранить в файле, который включается в программу при помощи директивы *include*. Функция возвращает в вызывающую программу
 - 1.1. сумму двух вещественных чисел,
 - 1.2. квадрат аргумента целого типа,
 - 1.3. разность двух целых чисел,
 - 1.4. произведение двух целых чисел,
 - 1.5. максимальное из двух вещественных чисел,
 - 1.6. минимальное из двух целых чисел.
2. Написать макрос для вычисления площади круга. Написать программу, которая выводит на экран площадь круга заданного радиуса. Число π задавать при помощи директивы *define*.
3. Написать программу, которая заполняет одномерный массив из N элементов случайными числами в интервале от 1 до 100, и выводит созданный массив на экран. Значение N можно задать при помощи директивы *define* или принять равным 10. Использовать директивы условной компиляции для объявления массива.

3. Порядок выполнения и сдачи работы

1. Изучить теоретическую и практическую часть.
2. Сдать преподавателю теорию работы путем ответа на контрольные вопросы.
3. Получить вариант задания и выполнить задание в Visual Studio 2013.
4. Предъявить преподавателю результат выполнения задания для самостоятельной работы

Оформить отчет.

Содержание отчета

Отчет готовится в электронном виде и распечатывается. Он должен содержать текст задания, скриншот программного кода и скриншот результата .

4. Контрольные вопросы

1. Что такое препроцессор?
2. Для чего предназначена директива #include?
3. Для чего предназначена директива #define?
4. Для чего предназначены директивы условной компиляции?
5. Как создаются многофайловые проекты?

ЛАБОРАТОРНАЯ РАБОТА № 8 АЛГОРИТМЫ СОРТИРОВКИ

Цель работы. Сортировка числовых последовательностей

Сортировка методом прямого включения (сортировка вставками).

Элементы просматриваются по одному, и каждый новый элемент вставляется в подходящее место среди ранее упорядоченных элементов. Упорядоченный список ранее пуст.

```
void vstavka(int * m, int n){  
    int    i,j,t;  
    for( i=0;i<n;i++){  
        t=m[i];  
        j=i-1;  
        while ((j>=0)&&(t<m[j])){  
            m[j+1]=m[j];  
            j--;  
        }  
        m[j+1]=t;  
    }  
}
```

```

        }
        m[j+1]=t;
    }
}

```

Метод прямого включения с делением пополам

```

void binins(a[],int n){
    int i,j,x,m,l,r;
    for (i=1;i<n;i++){
        { x=a[i];l=0;r=i;
        while(l<r){
            m=(l+r)/2;
            if (a[m]<=x)
                l=m+1;
            else r=m;
        }
        for (j=i;j>=r;j--){
            a[j]=a[j-1];
        }
        a[r]=x;
    }
}

```

Сортировка выбором

Сначала выделяется наибольший элемент и меняется местами с первым элементом. Затем выделяется наибольший среди оставшихся меняется местами со вторым элементом и т.д..

```

void sortvb(int *m,int n){
    int i,j,y,k;
    for( i=0;i<n;i++){

```

```

    y=m[i];
    k=i;
    for( j=i;j<n;j++)
        if(y>m[j]){
            y=m[j]; k=j;}
    m[k]=m[i];
    m[i]=y;
}
}

```

Сортировка обменом (сортировка методом пузырька)

Если два элемента расположены не по порядку, то они меняются местами. Процесс повторяется, пока элементы не будут упорядочены. В результате одного прохода через последовательность от ее конца к началу самый меньший элемент оказывается в начале последовательности (или самый большой оказывается в конце при просмотре от начала к концу).

```

void sortp(int *m,int n){
    int i,j,y;
    for( i=0;i<n;i++)
        for( j=0;j<(n-i-1);j++)
            if(m[j]>m[j+1]){
                y=m[j]; m[j]=m[j+1]; m[j+1]=y;}
}

```

Алгоритм можно улучшить, если учесть, что если в очередном проходе не было обмена, то последовательность упорядочена.

Шейкерная сортировка

Особенностью сортировки методом пузырька является то, что легкий элемент за один проход занимает свое место сразу, а тяжелый перемещается за один проход на одну позицию. Если изменить направление просмотра, то тяжелый элемент тоже занял бы свое место за один проход. Алгоритм, в котором чередуется направление последовательных просмотров называется шейкерной сортировкой. Такая сортировка эффективна, когда элементы почти упорядочены.

Алгоритм быстрой сортировки (алгоритм Хоара).

Идея алгоритма. Выделяется средний элемент последовательности. Все элементы, меньшие выделенного, переносятся влево от него, а все элементы большие выделенного вправо от него. Затем рекурсивно сортируются правая и левая подпоследовательность. -

```
#include <stdio.h>
#include <stdlib.h>
int  m[10] ;
int  i;
void sort(int l,int r){
int  i,j,x,y;
i=l; j=r; x=m[(l+r)/2];
do {
while (m[i]<x) i++;
while (x<m[j]) j--;
if (i<=j){
y=m[i]; m[i]=m[j]; m[j]=y;
i++; j--;
}
}while (i<j);
if (l<j) sort(l,j);
if (i<r) sort(i,r);
```

```

}
void quicksort(int * m,int Lo,int Hi){
    sort(Lo,Hi);
}

void main(){
    randomize();
    for( i=0;i<max;i++) m[i]=random(30);
    for( i=0;i<max;i++) printf("%d ",(m[i]));
    printf("\n");
    quicksort(m,0,9);
    printf("\n");
    for( i=0;i<max;i++) printf("%d ",(m[i]));
}

```

Сортировка Шелла.

Сначала сортируются элементы, отстоящие друг от друга на 9 позиций, потом 5,3,2 и 1. Конечная последовательность может быть другой, но последний шаг всегда должен быть равен 1.

```

void shell(int * m, int n){
    const int a[5]={9,5,3,2,1 };
    int i,j,z,k;
    int x;
    for (k=0;k<5 ;k++){
        z=a[k];
        for (i=z;i<20 ;i++){
            x=m[i];
            j=i-z;
            while ((x<m[j])&&(j>=0)){
                m[j+z]=m[j];
                j=j-z;
            }

```



```

        }
    m[j+z]=x;
    }
}
}

```

Задание

1. Создать массив из 20 элементов и заполнить его случайными числами в интервале от 0 до 100. Выполнить сортировку массива заданными методами и сравнить эффективность алгоритмов (подсчитать количество перестановок).
 - a) Написать функцию для сортировки массива методом пузырька. Прекратить сортировку, когда во внутреннем цикле не удастся сделать ни одну перестановку.
 - b) Написать функцию для сортировки массива методом шейкерной сортировки. Предусмотреть прекращение сортировки, когда массив будет упорядочен.
 - c) Написать функцию для сортировки массива методом выбора.
 - d) Написать функцию для сортировки массива методом вставки.
 - e) Написать функцию для сортировки массива методом прямого включения с делением пополам.
 - f) Написать функцию для сортировки массива методом быстрой сортировки.
 - g) Написать функцию для сортировки массива методом Шелла., когда шаги образуют ряд 1, 3, 7, 15, 31.
2. Создать массив структур, содержащих 3 поля:
 - Фамилия студента,
 - Номер группы,
 - Рейтинг.

Сначала отсортировать массив

- a. По фамилиям,
- b. По номеру группы,
- c. По рейтингу.

Выполнить сортировку методом

- a. выбора,
- b. пузырька,
- c. вставки

и определить, является ли данный метод сортировки устойчивым.

Список литературы

1. Подбельский В.В., Фомин С.С. Программирование на языке Си: Учебное пособие для студ. вузов. М. : Финансы и статистика, 2003.-600 с.
2. Прата С. Лекции и упражнения: Учебник. СПб. ДиаСофтЮП.2002. - 896с.
3. Павловская Т.А. С/С++. Программирование на языке высокого уровня: Учебник для вузов.-СПб.:Питер, 2004. -461 с.
4. Гагарина Л.Г. Алгоритмы и структуры данных: учебное пособие. – М.:Финансы и статистика:НФРА-Ъ, 2009. 304 с.