ЦОТ “БЕЛХАРД”

А.А. СИНКЕВИЧ, Г.Ф. САРКИСЯН

**ОСНОВЫ ПРОГРАММИРОВАНИЯ**

**ЯЗЫК “С”**

Учебно-методическое пособие для слушателей курса

“Основы программирования на языке С”

**Минск – 2015**

**Оглавление**

[§1 ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ 5](#_Toc428286162)

[1.1 Скачивание и установка IDE Visual C++ 2008 Express Edition 5](#_Toc428286163)

[1.2 Создание решения (solution), проекта и файла с исходным кодом 6](#_Toc428286164)

[1.3 Простые программы: печать стоки текста 10](#_Toc428286165)

[1.4 Понятие переменной. 12](#_Toc428286166)

[1.5 Целый тип данных 12](#_Toc428286167)

[1.6 Функции ввода (scanf) 16](#_Toc428286168)

[1.7 Числа с плавающей точкой 17](#_Toc428286169)

[1.8 Символьный тип 20](#_Toc428286170)

[1.9 Некоторые функции библиотеки <math.h> 21](#_Toc428286171)

[1.10 Русский язык в консоли С/C++. 21](#_Toc428286172)

[§2 УПРАВЛЯЮЩИЙ СТРУКТУРЫ 25](#_Toc428286173)

[2.1 Структура выбора if (если) и if/else(если - иначе) 25](#_Toc428286174)

[Упражнения для закрепления темы Ветвления 31](#_Toc428286175)

[2.2 Структуры повторений for (цикл) 33](#_Toc428286176)

[2.3 Генерация случайных чисел. 35](#_Toc428286177)

[2.4 Проект для закрепления «Таблица умножения». 37](#_Toc428286178)

[Упражнения для закрепления темы Циклы 38](#_Toc428286179)

[2.5 Структура повторения while (пока) 40](#_Toc428286180)

[2.6 Структура многомерного выбора switch 44](#_Toc428286181)

[2.7 Операторы continue и break 47](#_Toc428286182)

[2.8 Контрольные вопросы: 50](#_Toc428286183)

[§3 МАССИВЫ 52](#_Toc428286184)

[3.1 Одномерные массивы 52](#_Toc428286185)

[3.2 Примеры использования массивов 55](#_Toc428286186)

[Упражнения для закрепления темы массивы. 59](#_Toc428286187)

[3.3 Сортировки массивов 61](#_Toc428286188)

[3.4 Двумерные массивы 64](#_Toc428286189)

[§4 МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ. ФУНКЦИИ 68](#_Toc428286190)

[4.1 Примеры. Правила оформления и вызова функций 69](#_Toc428286191)

[4.2 Примеры. Функции, отличные от void. Оператор return 72](#_Toc428286192)

[4.3 Примеры. Область действия переменных. Локальные и глобальные переменные 75](#_Toc428286193)

[4.4 Примеры. Упражнения. 78](#_Toc428286194)

[4.5Рекурсия. 80](#_Toc428286195)

[*4.6* Классы памяти*.* 83](#_Toc428286196)

[4.7 Указатели 85](#_Toc428286197)

[4.8 Переменные перечисляемого типа 87](#_Toc428286198)

[*4.9* Связь массивов и указателей. 91](#_Toc428286199)

[*4.10* Передача массивов функции 91](#_Toc428286200)

[4.11 Контрольные вопросы: 93](#_Toc428286201)

[§5 СИМВОЛЬНАЯ СТРОКА 95](#_Toc428286202)

[*5.1* Символы и работа с ними 95](#_Toc428286203)

[5.2 Объявление символьной строки 96](#_Toc428286204)

[*5.3* Функции работы со строками из библиотеки <string.h> 100](#_Toc428286205)

[Упражнения для закрепления темы Строки символов 104](#_Toc428286206)

[5.4 Массив строк 105](#_Toc428286207)

[§6 ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ПАМЯТИ 107](#_Toc428286208)

[*6.1* Функция malloc() 107](#_Toc428286209)

[*6.2* Функция сalloc() 110](#_Toc428286210)

[*6.3* Функция realloc*()* 111](#_Toc428286211)

[§7 СТРУКТУРА 113](#_Toc428286212)

[*7.1* Объявление структуры 113](#_Toc428286213)

[7.2 Объединения 127](#_Toc428286214)

[§8 ФАЙЛЫ И ПОТОКИ 130](#_Toc428286215)

[8.1 Объявления файла 130](#_Toc428286216)

[8.2 Работа с текстовым файлом 132](#_Toc428286217)

[8.3 Работа с бинарным файлом 142](#_Toc428286218)

[8.4 Пример решения большой программы по бинарным файлам и структурам. 149](#_Toc428286219)

[8.5 Kонтрольные вопросы 162](#_Toc428286220)

[ПРИЛОЖЕНИЕ 1. Функции ввода/вывода. 163](#_Toc428286221)

[ПРИЛОЖЕНИЕ 2. 165](#_Toc428286222)

[Директивы Препроцессора 165](#_Toc428286223)

§1 ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

# 1.1 Скачивание и установка IDE Visual C++ 2008 Express Edition

Язык "C"(произносится "Си") - это универсальный язык программирования, для которого характерны экономичность выражения, современный поток управления и структуры данных, богатый набор операторов. Язык "C" не является ни языком "очень высокого уровня", ни "большим" языком, и не предназначается для некоторой специальной области применения. Но отсутствие ограничений и общность языка делают его более удобным и эффективным для многих задач, чем языки, предположительно более мощные."C" показал себя приятным, выразительным и разносторонним языком на широком множестве разнообразных программ. Его легко выучить, и он не теряет своих качеств с ростом опыта программиста.

Все программы (например: Internet Explorer, блокнот) первоначально представлены простым текстом. **Этот текст называется исходным кодом программы.** Чтобы исходный код превратился в полноценную программу, его нужно скомпилировать. Компиляцией занимается компилятор. Если в программе несколько файлов с исходным кодом, то помимо компилятора нужна ещё одна программа - компоновщик. Компоновщик связывает несколько файлов с исходным кодом между собой. Очень часто случается, что в исходном коде содержатся ошибки. Чтобы ошибки было проще отслеживать, нужен отладчик.

Итак, чтобы написать более-менее сложную программу, нужны четыре отдельных программы:

1. Редактор текста (text editor), чтобы написать исходный код.
2. Компилятор (compiler) того языка программирования, на котором написан исходный код. В нашем случае - компилятор языка C++.
3. Компоновщик (linker), чтобы связать все файлы программы между собой.
4. Отладчик (debugger), чтобы выловить все ошибки.

Крупные корпорации выпускают интегрированные среды разработки для разных языков программирования.

Для языка программирования C++ существует несколько сред. Самая популярная создана компанией Microsoft и называется Visual Studio (visual - визуальный, studio - студия). В Visual Studio можно разрабатывать не только программы на языке C++.

Все программы мы будем создавать в Visual Studio 2008. Недавно появилась версия Visual Studio 2010.

Вам необязательно устанавливать именно эту версию. Большинство примеров будут работать и на предыдущих версиях, например на: Visual C++ 2005, Visual C++ 2003, Microsoft Visual Studio 6.0. Незначительно может отличаться создание проекта.

Visual Studio выходит в нескольких редакциях (edition - редакция): Express Edition, Professinal, Premuim, Ultimate. Первая - бесплатная. Вторая стоит 800$, последняя - 12000$.

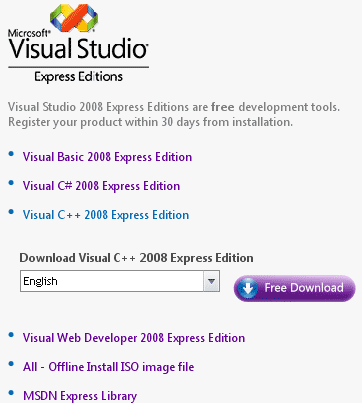
Мы будем использовать Visual C++ 2008 Express Edition - это бесплатная версия. Конечно же она урезана. Но в рамках нашего курса данной программы хватит.

Итак, со средой разработки разобрались. Будем использовать Visual C++ 2008 Express Edition. В неё входят все четыре перечисленные выше программы: текстовый редактор, компилятор, компоновщик, отладчик. И всё это бесплатно!!!

## Установка Microsoft Visual C++ 2008 Express Edition

Переходим по ссылке: <http://www.microsoft.com/express/download/default.aspx>:

Или <http://www.microsoft.com/visualstudio/en-us/products/2008-editions/express>



**Замечание**  
    Перейдя по ссылке, возможно, вы увидите, что страница оформлена не так, как на картинке выше. Microsoft довольно часто меняет эту страницу.

Возможны два варианта установки:

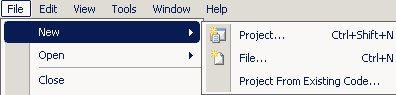
* Установка только Visual C++ 2008. В данном случае все файлы будут скачиваться в процессе установки. Чтобы воспользоваться этим вариантом, щёлкните на **Visual C++ 2008 Express Edition** и выберите язык. Затем нажмите на Free download (скачать бесплатно).
* Скачивание файла-образа всей Visual Studio 2008 Express Edition. И хотя в данном случае вам придётся скачать гораздо больший объём, но вы сможете установить среду разработки, находясь в оффлайне. Чтобы воспользоваться данным пунктом, щёлкните на **All - Offline Install ISO image file** и выберите язык. Затем нажмите на Free download (скачать бесплатно).

Далее устанавливаем программу на ваш компьютер и запускаем **IDE Microsoft Visual C++ 2008 Express Edition.** Также требуется зарегистрировать вашу копию программы через интернет.

Прежде чем начать писать код, нужно подготовить проект программы:

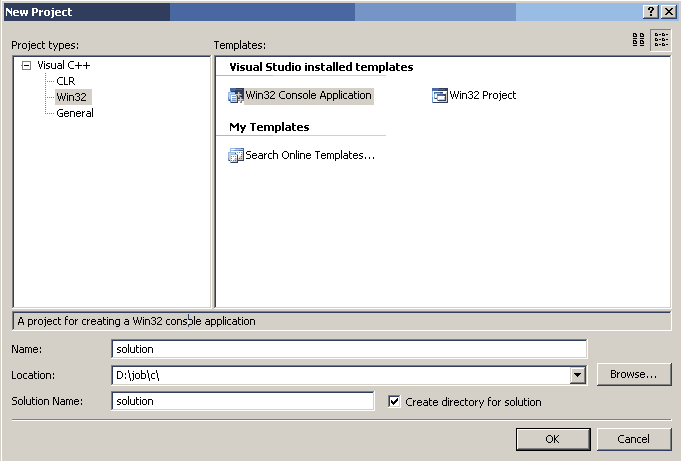
# 1.2 Создание решения (solution), проекта и файла с исходным кодом

Выберите пункт меню File → New → Project (Файл → Создать → Проект). Горячие клавиши для данного пункта - Ctrl+Shift+N.



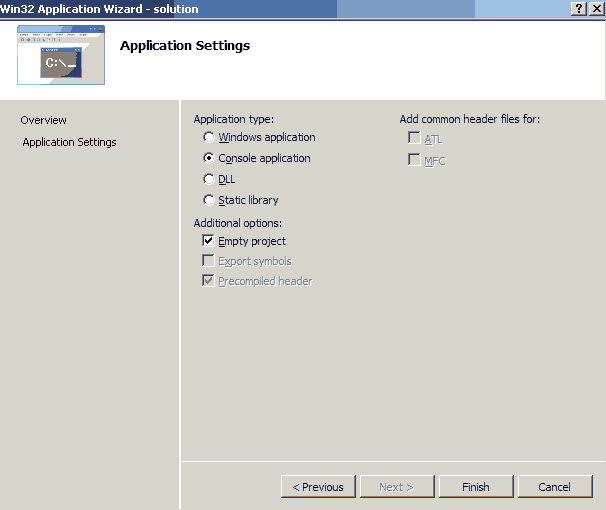
Откроется окно **New Project** (Новый проект). В левой части (Project types - Типы проектов) выбираем Win32. В правой (Templates - Шаблоны) - Win32 Console Application. В нижней части окна нужно ввести: имя проекта, папку для сохранения и имя решения. У меня эти пункты выглядят так (имя проекта (Name) необязательно должно совпадать с именем решения (Solution Name)):

Name: solution  
Location: d:\job\c  
Solution Name: solution



Жмём OK.

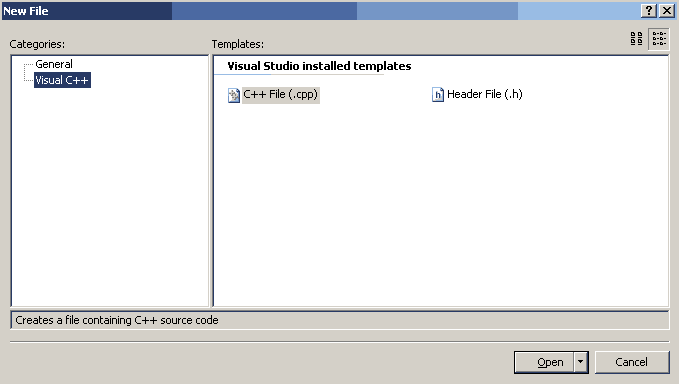
В следующем окне нажимаем Next (Далее) и в окне **Win32 Application Wizard - solution** выбираем следующее: Application type (Тип приложения) - Console application и ставим флажок в Additional options (Дополнительные свойства) - Empty project (пустой проект):



Запомните эти настройки. В ближайшее время мы будем создавать именно такие программы.

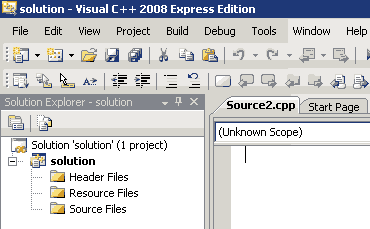
Жмём Finish

Мы создали решение **solution** и одноимённый проект для него. Теперь в проект нужно добавить файл. Выбираете пункт меню File → New → File (Файл → Создать → Файл). Откроется окно **New File** (Новый файл). В Categories (Категории) выберите Visual C++. В Templates (Шаблоны) - C++ File (.cpp).

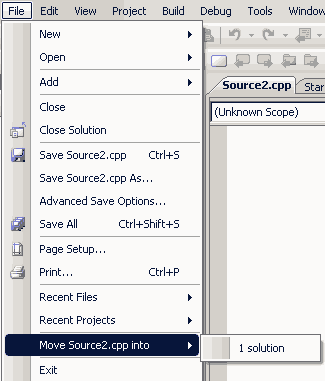


Жмём Open (Открыть).

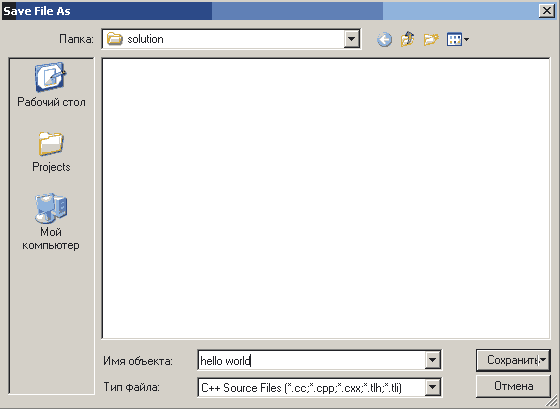
Посмотрите на обозреватель решений (окно слева). Первая строка - **Solution 'solution' (1 project)**. Это как нетрудно догадаться решение. На следующей строчке находится узел **solution** - это проект. Раскрыв данный узел, можно увидеть содержимое проекта. Проект включает в себя несколько типов файлов: Header Files (Заголовочные файлы), Resource Files (Файлы ресурсов), Source Files (Файлы исходного кода).



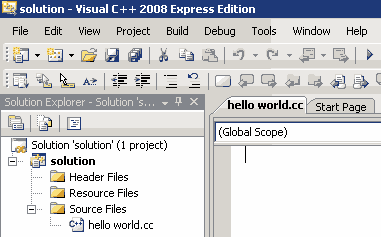
Теперь файл который мы создали (его имя можно увидеть над редактором на вкладке рядом с Start Page - Начальная страница) нужно добавить к проекту. Для этого выберите пункт меню File → Move Source2.cpp into → 1 solution.



Откроется окно сохранения файла (Save File As). В поле "Имя объекта" введите имя файла. Например - "hello world". Щёлкаем "Сохранить"

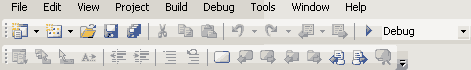


Ещё раз взгляните на обозреватель решений. В Source Files проекта появился новый файл "hello world.cc". Файлы с расширениями .cpp и .cc - это файлы, содержащие исходный код программы.



## Элементы интерфейса IDE Visual C++ 2008 Express Edition

Теперь рассмотрим элементы интерфейса Visual C++ 2008. Вверху, под строкой заголовка, расположено меню как со стандартными для Windows пунктами: File (Файл), Edit (Правка), View (Вид), Tools (Сервис), Window (Окно), Help (справка), так и с уникальными для Visual C++ Express Edition: Project (Проект), Build (Построение), Debug (Отладка).



Под меню расположена довольно стандартная панель инструментов.

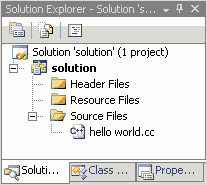
В центре окна в данный момент открыта вкладка **hello world.cc**.



Перейдите на вкладку **Start Page**. На этой странице есть окно - Recent Projects (Последние проекты). В нём перечисляются решения (solutions) и проекты с которыми вы недавно работали. Когда вы в следующий раз откроете IDE, выберите в данном окне решение **solution**.

Вернитесь на вкладку **hello world.cc**. Под вкладками расположен редактор. Здесь пишется код программы.

Слева расположено окно - **Solution Explorer** (Обозреватель решений). В нём в виде древовидной структуры представлено ваше решение. Термин решение используется в Visual Studio для организации проектов т.е. в одном решении может быть несколько проектов. Один проект, обычно, представляет собой одну программу.



В самом верху обозревателя решений написано имя нашего решения - *Solution 'solution'*, а в скобках указано количество проектов. После решения идёт несколько узлов - в зависимости от количества проектов в решении. У нас пока только один проект - "solution". Для каждого проекта на следующих уровнях показаны файлы данного проекта.

# 1.3 Простые программы: печать стоки текста

Любая “C”-программа, состоит из вызовов функций, которые могут находиться в самой программе или в библиотечных (головных) файлах. Итак, любая "C"-программа, каков бы ни был ее размер, должна содержать следующие строки

#include<stdio.h>

void main()

{

сама программа

}

Строка **#include<stdio.h>**является *директивой препроцессора*, т.е. строка, начинающая с #, обрабатывается препроцессором перед компиляцией программы – это означает включить в программу содержание библиотечного файла потока ввода/вывода. Выполнение программы начинается с функции **main(),** т.е. сама программа пишется в фигурных скобках {….}

Напишите следующий код программы в созданный файл Ex1.

#include<stdio.h>

void main()

{

printf("Privet Minsk!!!");

}

Чтобы запустить программу, нажимаемCtrl+F5 или кнопочку

После запуска программы мы увидим следующий экран результата

Privet Minsk!!!Press any key to continue\_

Хотя тут есть небольшая проблема. При нажатии на клавишу  , наше окно консоли откроется, выведет текст и сразу закроется. Чтобы этого избежать, дополним нашу программу функцией задержки.

#include<stdio.h> //Полдключение файла

#include<conio.h> //В этом файле описана функция getch();

int main() //Головная функция int main()

{

printf( " Privet Minsk!!!"" ); //Функция вывода

getch();//Функциязадержки

return 0; } //Функция возврата( выхода из программы )

Вы заметили, что возможно написание главной функции двумя способами: int main() или void main().

В качестве аргументов, для функции **printf(),** использовали символьную строку. Последовательность, из любого количества символов, заключенных в удвоенные кавычки “...”, называется *символьной строкой* или *строчной константой.* Функция **printf()** не обеспечивает автоматического перехода на новую строку, так что многократное обращение к ней можно использовать для поэтапной сборки выходной строки. Наша первая программа, с точно таким же успехом могла бы быть написана в виде:

#include<stdio.h>

void main()

{

printf("Privet ");

printf("Minsk");

printf("!!!");

}

Для регулировки курсора, на экране результатов, существует следующая последовательность символов, которые тоже должны находиться в “ ”.

**\n** переводит курсор на новую строку

\**t** переводит курсор на горизонтальную табуляцию

Символ \ просто так не используется в функции **printf()**, т.к. \ и любой символ для программы что-то означает, примеры некоторых из них

|  |  |
| --- | --- |
| **\a** | Звуковой сигнал |
| **\b** | Сдвиг текущей позиции влево |
| **\’** | Апостроф |
| **\”** | Кавычки |
| **\\** | Левая косая черта |
| **\r** | Сдвиг в начало строки |

Упражнения для закрепления:

**1.Что делают следующие программы:**

|  |  |
| --- | --- |
| void main()  {  printf("Pri");  printf("vet");  printf("Minsk");  printf("\n");  } | void main()  {  printf("Pri\t");  printf("vet\t\n");  printf("Minsk");  printf("\n");  } |
| void main()  {  printf("P\n\trivet\t Minsk\n!\t!\n");  } |  |

**2 Распечатать свою фамилию следующим образом**

B

u

k

i

n

**3. Распечатать прямоугольник следующим образом**

\*\*\*\*\*\*\*

\* \*

\* \*

\* \*

\*\*\*\*\*\*\*

# 1.4 Понятие переменной.

Данные – это область в оперативной памяти компьютера, где можно хранить некоторое значение для использования его в программе. Данные, значения которых можно изменить во время программы называются *переменными*, а не изменяемые данные называются *константами*. В программе все данные перед их использованием должны быть определены.

Определение переменных имеет следующий формат:

**тип\_данных** имя\_переменной1, имя\_переменной2, имя\_переменной3,…;

**тип\_данных** имя\_переменной1= начальное значение, имя\_переменной2= начальное значение, имя\_переменной3= начальное значение …;

имя\_переменной – это идентификатор. Идентификатором называется последовательность символов, содержащая буквы, цифры и символ подчеркивания ‘\_’, которая не начинается с цифры. Идентификатором не могут быть *ключевые* слова, т.е. это имена, используемые в языке с некоторым заранее определенным смыслом. Для программы, после компиляции имя переменной заменяется адресом в оперативной памяти.

*Ключевые слова*

**auto break case char const continue default do double**

**else enum extern float for if int long register return**

**short signed sizeof struct switch typedef union unsigned void**

**volatile while**

В языке “C” существуют следующие типы данных: **данные целого типа, числа с плавающей точкой, символьные данные, переменные перечисляемого типа, структурный тип данных.** Каждый тип данных имеет размер, т.е. сколько байтов выделяется в оперативной памяти для записи переменной данного типа, и диапазон значений.

# 1.5 Целый тип данных

|  |  |  |
| --- | --- | --- |
| **тип** | **Размер, в байтах** | **Диапазон значений** |
| **short int** | 2 | -32768………32767 |
| **int** | 2 | -32768………32767 |
| 4 | -2 147 483 648…….2 147 483 647 |
| **long int** | 4 | -2 147 483 648…….2 147 483 647 |
| **unsigned long int** | 2 | 0……65 536 |
| 4 | 0……4 294 967 295 |

**unsigned**– это беззнаковое целое, т.е. положительные числа.

## Оператор присваивания. Вывод на экран.

Рассмотрим работу с самыми простыми переменными. В них можно хранить только целые числа.

Для того, чтобы объявить целое число необходимо написать

**int** x; где **int** – это тип переменной, а **x** – это имя переменной.

Для того, чтобы вывести целое число на экран результатов необходимо

printf(“%d”,x);

Для того, чтобы обратиться к целому типу данных, используем следующие символы:

%d **int**

%hd **short int**

%ld **long int**

Рассмотрим пример. Внимательно прочитайте комментарии, которые не воспринимаются компилятором, а пишутся для людей, которые изучают и отлаживают код.

В языке Си есть 2 вида комментариев. Однострочный: начинается с //

Многострочный: /\* комментарий \*/

#include<stdio.h>

void main()

{ //описание переменных. Тут создаются ячейки в памяти с именами

int a,b,c; //имена переменных в списке через запятую

a=5; //присвоение значения переменной а

printf("%d\n",a); //вывод значения переменной, переход на новую строку

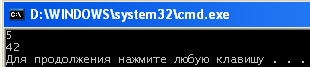
a++; //операция увеличения переменной на 1. Эквивалентна a=a+1;

b=7; //присвоение значения переменной b

c=a\*b;//присвоение значения переменной c, как результат умножения

printf("%d\n",c); //вывод значения переменной, переход на новую строку }

результат:



Справа от оператора присваивания может стоять число или любое выражение. Слева может стоять только имя переменной.

Еще один пример где программа вычисляет сумму 2 чисел.

#include<stdio.h>

#include<conio.h>

int main()

{

int a = 5, b = 8; //Объявление переменных с присвоением

a = a + b; //Присваиваем переменной а новое значение

printf( "Sum a + b is: %d", a ); //Выводим переменную а

getch();

return 0;}

**Задачи:**

1.Что делают следующие программы:

|  |  |  |
| --- | --- | --- |
| int x;  printf(“%d\n”,x); | int x;  x=2.8;  printf(“%d\n”,x); | int x=5;  printf(“%d\n”); |
|  |  |  |
| int x,y;  x=5;  y=7;  printf(“x=%d\ny=%d”,y,x); |  | int x,y;  x=5;  y=7;  printf(“%d+%d=%d\n”,y,x,x+y); |

2.Даны 2 целых числа, например 21 и 8, написать программу, которая считала сумму, разность, умножение, деление данных чисел и распечатала, следующим образом:

21+8=29

21-8=13 и т.д.

3.Дана длина ребра куба(целое число). Найти объем куба и площадь его боковой поверхности.

4. В переменные А и В записаны целые числа (оператором присваивания, например, А=20; В=13). Поменяйте числа в этих переменных местами.

Будьте внимательны! Если записать А=В, вы потеряете число 20 и получите число 13 в двух переменных. Воспользуйтесь третьей переменной Смотри рисунок.

20

13

А

В

С

1

2

3

5\*. Выполните обмен без использования третьей переменной. Используя операции сложения и вычитания.

## Как представляются целые числа в памяти компьютера

Разберемся, что означает размер 2 байта, т.е. 16 битов. Машина все воспринимает в 2-ой системе исчисления, т.е. 0 и 1; вся память разбита на биты, т.е. на ячейки где записываются 1 или 0. В десятеричной системе исчисления число 3567, цифры 3, 5, 6, 7 – это коэффициенты, причем 3 коэффициент 103, 5 – 102, 6- 101, 7 – 100, т.е. 3567=3\*103 + 5\*102 + 6\*101+ 7\*100

**Short int** занимает памяти 2 байта = 16 битов, начинается с 0…..15.

15 бит – для знака, 1 – если число отрицательное

0 - если число положительное

0...14 битах записывается число, переведенное из десятичной системы исчисления в 2-ную.

Пример:

|  |  |
| --- | --- |
| 37 | 2 |
| 2 | 18 |
| 17 |  |
| 16 |  |
| 1 |  |
| 18 | 2 |
| 18 | 9 |
| 0 |  |

|  |  |
| --- | --- |
| 9 | 2 |
| 8 | 4 |
| 1 |  |

|  |  |
| --- | --- |
| 4 | 2 |
| 4 | 2 |
| 0 |  |

|  |  |
| --- | --- |
| 2 | 2 |
| 2 | 1 |
| 0 |  |

Полученные остатки от деления на 2 запишем наоборот и получим 1 0 0 1 0 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |

100101=1\*25+0\*24+0\*23+1\*22+0\*21+1\*20=32+0+0+4+0+1=37

Следовательно, положительное максимальное число, которое поместится в 16 битах и будет:

0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 в 2-ой системе исчисления и 32 767 в десятичной системе исчисления.

Заметим, что представление в памяти и диапазон значений для типов int не определены в языке Си. По умолчанию размер int соответствует размеру целого на данной машине. Например, на 16-разрядной машине тип int всегда равен 16 битов, или 2 байта. На 32-разрядной машине тип int всегда равен 32 бита, или 4 байта.

Переносимость программ можно улучшить путем включения выражений с операцией **sizeof**. Использую операцию **sizeof**, можно определить количество байтов, выделяемых для записи переменных соответствующего типа, например

i=sizeof(int);

l=sizeof(long int);

Операцию можно использовать и для определения длины переменной типа, объявленного пользователем.

# 1.6 Функции ввода (scanf)

Для того чтобы ввести целые числа:

int x,y;

scanf("%d%d",&x,&y);

Обратите внимание! В операторе scanf в кавычках не ставится ничего кроме спецификаторов формата. Часто начинающие программисты хотят поставить внутри кавычек \n, что приводит к ошибке. Также обратите внимание, что перед списком переменных ставится символ &.

Ввод числа заканчивается enter–ом или пробелом и если вместо числа ввести символ, то в области памяти остается старое значение и при этом никаких сообщений выводится не будет (в Вorland C), а в С++ работа прерывается.

Для того чтобы ввести вещественные числа:

double x,y;

scanf("%lf%lf",&x,&y);

Для того чтобы ввести символы:

char x,y;

scanf(“%c%c”,&x,&y);

Если перед вводом символа вводилась что-либо другого типа, то перед вводом символьного типа необходимо очистить буфер клавиатуры с помощью функции **fflush(stdin);**

Про символьный и дробный типы будет написано далее.

Пример: Введем 2 числа и посчитаем сумму.

#include<stdio.h>

#include<conio.h>

int main()

{

int a, b; //Объявление переменных

printf( "Enter A : " ); //Запрос на ввод переменной

scanf( "%d", &a ); //Чтение переменной по адресу а

printf( "Enter B : " );

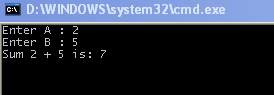
scanf( "%d", &b );

printf( "Sum %d + %d is: %d", a,b,a+b ); //Выводим переменные

getch();

return 0;

}



# 1.7 Числа с плавающей точкой

Числа, в которых положение точки не зафиксировано после некоторого разряда, а указывается специальным числом называются *числа с плавающей точкой*.

**float** занимает 4 байта

**double** занимает 8 байтов

**long double**  занимает 8 байтов

Для того, чтобы обратиться к вещественному типу данных, используем следующие символы:

%f **float**

%lf **double**

%lf **long double**

Для того, чтобы объявить вещественное число необходимо

**double** number=45.6; где **double**– это тип переменной, а **number** – это имя переменной, которой было присвоено значение 45.6

Для того, чтобы вывести данное число на экран результатов необходимо

**printf(“%lf”,number);**

В данном случае программа выведет 45.600000, т.е выводится 6 знаков после запятой, чтобы отрегулировать количество выводимых символов после запятой, в функции **printf(“%.2lf”,number);** пишется % потом ставиться ‘точка’ далее число, которое отражает количество выводимых чисел после ‘точки’ и далее lf или f, в зависимости от того, как был объявлен вещественный тип.

Пример:

#include<stdio.h>

#include<conio.h>

int main()

{

double a, b; //Объявление переменных

printf( "Enter A : " ); //Запрос на ввод переменной

scanf( "%lf", &a ); //Чтение переменной по адресу а

printf( "Enter B : " );

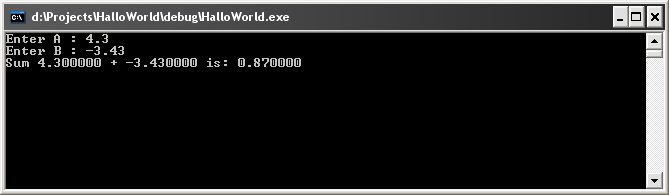
scanf( "%lf", &b );

printf( "Sum %lf + %lf is: %lf", a,b,a+b );//Выводимпеременные

getch();

return 0;

}



Особое внимание следует обратить на операцию деления. Если делятся два целых числа, то будет производится операция целочисленного деления. Например, при делении 1/3 мы получим 0. Для правильного деления хотя бы 1 из аргументов дожен быть преобразован к дробному типу.

Программа, которая правильно считаем деление двух целых чисел.

#include<stdio.h>

void main()

{ int x,y;

double z;

x=5;

y=2;

z=(double)x/y;

//или z=(double)x/(double)y;

//или z=x/(double)y;

printf("%d/%d=%.2lf\n",x,y,z);

}

Сложение, вычитание и умножение целых значений выполняются как обычно, а вот деление целых значений в результате дает опять целое (так называемое *"целое деление"),* например, 5/2 даст в результате 2, а не 2.5 , а 5/(-3) даст -1. Дробная часть числа попросту отбрасывается, происходит усечение частного.

Это странное для математики правило естественно для программирования: если оба операнда имеют один и тот же тип, то и результат имеет тот же тип.

## Математические операторы:

К арифметическим операциям относятся:

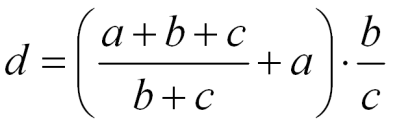
* сложение + (плюс);
* вычитание - (дефис);
* умножение \* (звездочка);
* деление / (наклонная черта – слэш);
* взятие остатка от деления (деление по модулю) % (процент);
* инкремент (увеличение на единицу) ++;
* декремент (уменьшение на единицу) --.

Между сдвоенными плюсами и минусами нельзя оставлять пробелы.

Кроме стандартных операторов умножение, деление, сложение, вычитание, есть оператор остаток от деления. Обозначается символом %

С помощью остатка от деления можно проверять числа на четность, раскладывать целое число на отдельные цифры. Например 7%2=1 или 98%10=8.

Решим пример.



#include<stdio.h>

#include<conio.h>

int main()

{

double a, b, c, d; //Объявление переменных

printf( "Enter A : " ); //Запрос на ввод переменной

scanf( "%lf", &a ); //Чтение переменной по адресу а

printf( "Enter B : " );

scanf( "%lf", &b );

printf( "Enter C : " );

scanf( "%lf", &c );

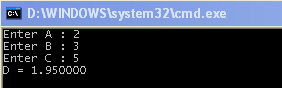
d = ( ( a + b + c ) / ( b + c ) + a ) \* b / c; //Вычисляем выражение

printf( "D = %lf", d ); //Выводим переменные

getch();

return 0;

}



## Обратная запись числа.

Разберем решение сложной задачи. Дано трехзначное число x=abc ( a,b,c - его цифры). Требуется получить число написанное теми же цифрами, но в обратном порядке. То есть, если дано число 128, то надо получить 821. Сначала нужно найти цифры. Выстроить в искомом порядке легко x=100c+10b+a. Проще всего найти цифру c. Она будет равна остатку от деления на 10.

Запомните: Остаток от деления на 10 дает последнюю цифру числа.

Далее, чтобы найти b выполним целочисленное деление на 10. Теперь x=ab.

(Для 128 это будет 12), а затем с помощью остатка от деления найдем b.

Найти а просто – действуем как и с переменной b.

Теперь можно написать всю программу:

#include<stdio.h>

void main()

{

int a, b, c, x; //Объявление переменных

x=128;

c=x%10; //последняя цифра числа

x=x/10; //отбросим последнюю цифру

b=x%10;

x=x/10;

a=x%10;

x=c\*100+b\*10+a;

printf( "x = %d", x ); //Выводим результат

}

# 1.8 Символьный тип

Данные типа **char** занимают в памяти 1 байт. Код от 0 до 255 в этом байте задается один из 256 возможных символов. Тип **char** является типом “целое”. Данные типа **char** могут рассматриваться как данные со знаком (**signed char**) и без знака (**unsigned char**). В случае **signed char** диапазон значений от -128 до 127, а **unsigned char**– от 0 до 255. Константа типа **char** – это символ, заключенный в одинарных кавычках. Каждому символу ставится в соответствие некоторый код. По этому коду из таблицы описания конфигурации символа выбирается изображение данного символа, которое выводится на экран. Коды цифр и латинских букв идут в порядке возрастания, т.е. 0<1<2<3<…<A<B<……<Z<a<……<z

Код символа является его порядковым номером во множестве ASCII кодов.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | nul | soh | stx | etx | eot | enq | ack | bel | bs | ht |
| 1 | nl | vt | ff | cr | so | si | dle | dc1 | dc2 | dc3 |
| 2 | dc4 | nak | syn | etb | can | em | sub | esc | fs | qs |
| 3 | rs | us | sp | ! | “ | # | $ | % | & | ‘ |
| 4 | ( | ) | \* | + | , | - | . | / | 0 | 1 |
| 5 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; |
| 6 | < | = | > | ? | @ | A | B | C | D | E |
| 7 | F | G | H | I | J | K | L | M | N | O |
| 8 | P | Q | R | S | T | U | V | W | X | Y |
| 9 | Z | [ | \ | ] | ^ | \_ | ` | a | b | c |
| 10 | d | e | f | g | h | i | j | k | l | m |
| 11 | n | o | p | q | r | s | t | u | v | w |
| 12 | x | y | z | { | | | } | ~ | del |  |  |

В “С++” используются и двусимвольные константы. Например: ‘db’ ’\r\a’ ‘\n\t’

*Объявление символа.*

char ch1,ch2; // это объявление двух символьных констант.

ch1='a'; //присвоение первоначальных значений

ch2='b';

и т.к. символьная константа является типом “целым”, то символьные константы можно: присваивать друг другу; сравнивать друг с другом.

Для того, чтобы обратиться к символьному типу данных, используем следующий символ %c

Для того, чтобы вывести данные символы на экран результатов необходимо

**printf(“ch1=%с\tch2=%с”,ch1,ch2);**

Пример демонстрирующий арифметических операций

#include<stdio.h>

void main()

{

char a\_ch,b\_ch,c\_ch;

int l;

a\_ch=126;

b\_ch=127;

c\_ch=a\_ch+b\_ch;

l=a\_ch+b\_ch;

printf("%c+%c=%c\n",a\_ch,b\_ch,c\_ch);

printf("%d+%d=%d\n",a\_ch,b\_ch,c\_ch);

printf("%d+%d=%d\n",a\_ch,b\_ch,l);

}

# 1.9 Некоторые функции библиотеки <math.h>

|  |  |  |
| --- | --- | --- |
| **Функция** | **Описание** | **Примеры** |
| **sqrt(x)** | корень квадратный из х | sqrt(900)=30  sqrt(9)=3 |
| **exp(x)** | экспоненциальная функция ex | exp(1)=2.718282  exp(2)=7.389056 |
| **log(x)** | логарифм натуральный x (по основанию e) | log(2.718282)=1  log(7.389056)=2 |
| **log10(x)** | логарифм десятичный x (по основанию 10) | log10(1)=0  log10(10)=1 |
| **fabs(x)** | абсолютное значение x | если x>0, то fabs(x)=x  если x=0, то fabs(x)=0  если x<0, то fabs(x)=-x |
| **ceil(x)** | округление x до наименьшего целого не меньше х | ceil(9.2)=10  ceil(-9.8)=-9 |
| **floor(x)** | округление x до наибольшего целого не больше х | floor(9.2)=9  floor(-9.8)=-10 |
| **pow(x,y)** | x в степени y | pow(2,7)=128  pow(9, 0.5)=3 |
| **fmod(x,y)** | остаток от деления x/y, как число с плавающей точкой | fmod(13.657,2.333)=1.992 |
| **sin(x)** | синус х (х в радианах) | sin(0)=0 |
| **cos(x)** | косинус х (х в радианах) | cos(0)=1 |
| **tan(x)** | тангенс х (х в радианах) | tan(0)=0 |

# 1.10 Русский язык в консоли С/C++.

Думаю вы уже попытались пытались вывести в консольное приложение русский текст, но вместо него выводились разные кракозябры.

К примеру, если мы напишем такой код на С

#include<stdio.h>

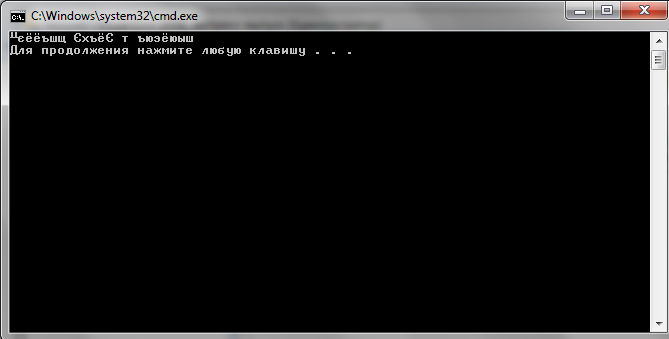
void main()

{

printf("Русский текст в консоли\n” );

}

То после компиляции, вот что будет выдавать нам консольное окно программы.

[](https://lh3.googleusercontent.com/-4rpbUbV5rgo/T0qcy-90mzI/AAAAAAAAAHU/fy0gvNPLsFs/s669/invalid_rus_text.jpg)

Если же мы будем использовать функцию setlocale() внутри функции main()

#include<stdio.h>

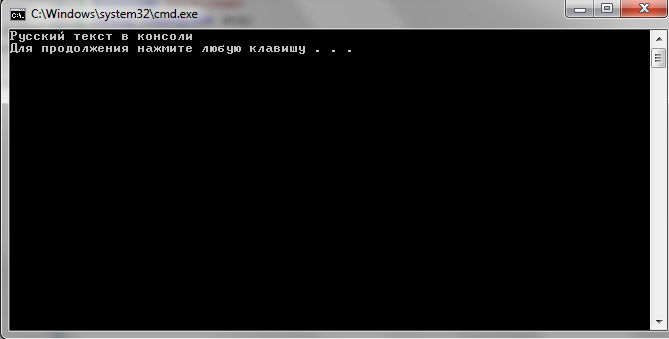
#include <locale.h>

void main()

{

printf("Русский текст в консоли\n” );

}

То получим вместо иероглифов нормальный русский шрифт, вуаля: [](https://lh4.googleusercontent.com/-yiqzUaKusNQ/T0qcy5dP29I/AAAAAAAAAHY/3xHKxUseG5o/s669/normal_rus_text.jpg)

Есть и другой способ. Можно использовать функции из библиотеки <windows.h> Для корректного ввода и вывода кириллицы на консоль надо использовать пару функций: SetConsoleOutputCP() и SetConsoleCP(). В качестве единственного параметра обеим функциям передается номер кодовой страницы. В нашем случае (кириллица) — это 1251.

Проблема заключается в том, что когда в Microsoft придумывали Windows, то попутно придумали новую кодировку для кириллицы. Трудно сказать зачем, но придумали. А старую кодировку, которая использовалась в MS DOS, оставили. Видимо в целях обратной совместимости. С выходом новых версий Windows ситуация только ухудшилась. Т.к. консоль, уже как часть операционной системы, унаследовала кодировку кириллицы от MS DOS.

В итоге сейчас для кириллицы имеем две кодировки: cp866 — старая досовская кодировка и cp1251 (она же windows-1251) — новая, от Windows. Кстати, буковки «cp» в названии кодировки означает codepage — кодовая страница в смысле «страница кодировки символов».

Кроме того, консоль имеет собственную настройку кодовой страницы. Для России по умолчанию — cp866.

//Для практических целей можно использовать шаблон:

#include <windows.h>

#include <stdio.h>

int main()

{ SetConsoleOutputCP(1251);

SetConsoleCP(1251);

// здесь должна быть программа

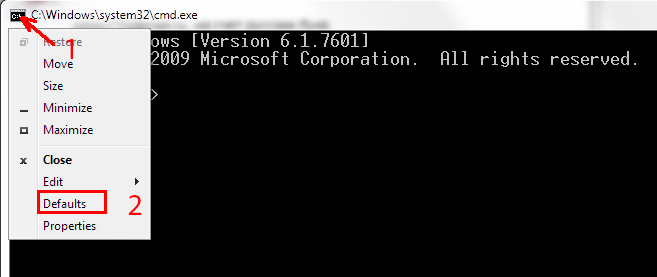
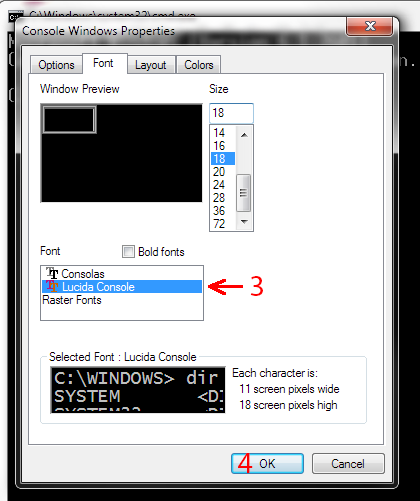
printf("Русский текст в консоли\n" );

system("pause");

return 0;}

Обратите внимание, что при работе с национальными алфавитами: в настройках консоли установите для вывода шрифт True Type.

Сделать это можно следующим способом:

1. Запустите командную строку
2. Откройте контекстное меню и зайдите в настройки консоли   
    
3. Измените шрифт на Lucida Console.   
   

**Задачи:**

1. Программа определяет ваш возраст. Требует ввести год вашего рождения и нынешний год.
2. Ввести три числа и посчитать сумму, разность, произведение и среднее этих чисел.
3. Ввести число и посчитать квадрат этого числа.
4. Ввести 2 целых числа и посчитать деление этих чисел.
5. Дано значение температуры Tf в градусах Фаренгейта. Определить значение этой же температуры в градусах Цельсия. Температура по Цельсию и температура по Фаренгейту связаны следующим соотношением:

Tc =(Tf-32)\*5/9

1. Дан диаметр окружности d. Найти ее длину L =PI\*d. В качестве значения PI использовать 3.14.
2. Даны катеты прямоугольного треугольника a и b. Найти его гипотенузу c и периметр P:
3. Даны переменные A,B,C. Изменить их значения, переместив содержимое A в B, B в C, C в A, и вывести новые значения переменных A,B, C.
4. Скорость первого автомобиля V1 км/ч, второго- V2 км/ч, расстояние между ними S км. Определить расстояние между ними через T часов, если автомобили первоначально движутся навстречу друг к другу. Данное расстояние равно модулю разности начального расстояния и общего пути, проделанного автомобилями;  
   общий путь=время \* суммарная скорость.
5. Напишите программу, которая считывает число из 4 цифр, разделяет это число на отдельные цифры и выводит цифры по отдельности. Пример – для числа 1234 вывод должен быть*:*

1

2

3

4

1. Напишите программу, которая выводит квадраты и кубы чисел от 0 до 10.

0 0 0

1 1 1

2 4 8

3 9 27

4 16 64

И так далее…

§2 УПРАВЛЯЮЩИЙ СТРУКТУРЫ

Обычно операторы программы выполняются в той последовательности, в которой они записаны в программе, что называется *последовательным выполнением*. Однако различные операторы позволяют нам указать, что следующим должен выполняться не очередной оператор в тексте программы, а какой-то другой. Ниже идет описание данных операторов.

# 2.1 Структура выбора if (если) и if/else(если - иначе)

Структура выбора **if** выполняет некоторое, действие если проверяемое условие верно. Структура выбора **if/else** выполняет одно действие, если условие истинно и выполняет другое действие, если условие ложно. Структура **if** называется *структурой с единственным выбором.* Структура **if/else** называется *структурой с двойным выбором*

**if**(условие) **if**(условие)

{ {

операторы1; операторы;

} }

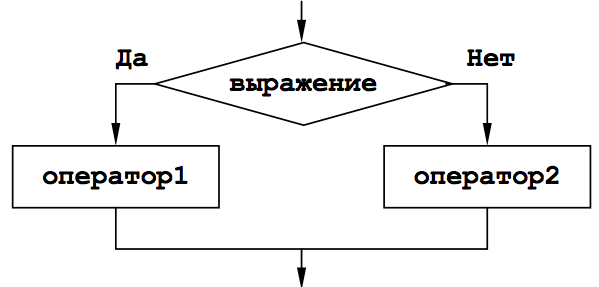
**else**

{

операторы2;

}

Если условие верно, то выполняются операторы1, иначе операторы2.



**Условия:**

В языке Cи шесть обычных операций сравнения чисел по величине:

* больше >;
* меньше <;
* больше или равно >=;
* меньше или равно <=;
* равно ==;
* не равно !=.

Сдвоенные символы записываются без пробелов, их нельзя переставлять местами, запись => будет неверной.

Результат сравнения будет иметь логическое значение: true(1) для выражения 3 != 5 ; или false(0) , например, для 3 == 5. В языке Си нет как такового логического типа. Истина и ложь представляется в памяти числами. То есть ложь дает 0, другие числа дают истину. При вычислении логических операций Истина это 1.

Для записи сложных сравнений следует использовать логичес­кие операции. Например, в вычислениях часто приходится делать проверки вида а < х < b. Подобная запись на языке Си приведет к сообщению об ошибке.

## Логические операции:

&&- И –дает истину если оба условия выполняются одновременно

||- ИЛИ – дает истину, если хотя бы одно условие истинно

В данном случае следует написать выражение (а<х) && (х<b), причем здесь скобки можно опустить, написав а<х && х<b.

Тут важно понять какую логическую операцию надо использовать. Логическое И используем, когда оба условия выполняются одновременно. Например: для поступления в математический класс требуется иметь оценки по физике и математике не ниже 8.

Для логического ИЛИ достаточно выполнения одного условия из нескольких. Например: для поступления в вуз вне конкурса нужно победить в республиканской олимпиаде по профильному предмету или закончить школу с золотой медалью. В данном случае достаточно одного условия.

Для простоты запоминания логическое И можно заменить \*(умножение), а ИЛИ +(сложение). В таблице true(1), false(0)

|  |  |
| --- | --- |
| И | ИЛИ |
| 1\*1=1 | 1+1=1 |
| 1\*0=0 | 1+0=1 |
| 0\*0=0 | 0+0=0 |

## Тернарная операция

Этой операции также нет в других языках, и без неё можно обойтись. С её помощью в более компактном виде в некоторых случаях можно записать полную форму оператора if. Например, вместо if (k>9) R=100; else R=x\*y; можно записать R=k>9 ? 100 : x\*y;.

Это единственная трёхместная операция, так как для неё требуется три операнда. Общий вид операции: выражение1 ? выражение2 : выражение3.

Операция выполняется следующим образом: вычисляется выражение1; если оно истинно (любое ненулевое значение), то вычисляется выражение2, и всё выражение получает это значение; если выражение 1 ложно (нулевое значение), то вычисляется выражение3, и всё выражение получает это значение.

Эта операция не обязательно должна быть связана с присваиванием. Например, известно, что в printf можно записать выражение, а в нём эту операцию без дополнительного присваивания значения выражения. Например:

printf("%d",(x>0 && y>x ? x\*y : x+y)); в зависимости от значений x и y выведут их произведение или сумму. Это равносильно

if (x>0 && y>x)

printf("%d",(x\*y));

else

printf("%d",x+y);

Чтобы хорошо понять работу конструкции, разберите следующие задачи.

**Программы:**

1. Введите число, если оно больше 60 выводит на экран Зачтено и в любых случаях издает звук.

**int x;**

**printf("Введите число: ");**

**scanf("%d",&x);**

**if(x>60)**

**{**

**printf("Зачтено!!!!!\n");**

**}**

**printf("\a\a\n");**

1. Введите число, если оно больше 60 выводит на экран Зачтено, иначе издает звук

**int x;**

**printf(“Введите число: ”);**

**scanf(“%d”,&x);**

**if(x>60)**

**{**

**printf(“Зачтено!!!!!\n”);**

**}**

**else**

**printf(“\a\a\n”);**

1. Введите число и определить является ли число положительным или отрицательным.

Обратите внимание на эту задачу. Тут у нас встречается вложение конструкции if друг в друга. Для наглядного понимания начертим блок-схему.

Ввод числа х

Х>0

положительное

начало

Х<0

отрицательное

ноль

конец

да

да

нет

нет

**#include<stdio.h>**

**void main()**

**{**

**int x,;**

**printf("Введите число: ");**

**scanf("%d",&x);**

**if(x>0)**

**printf("Число положительное\n");**

**else**

**{**

**if(x<0)**

**printf("Число отрицательное\n");**

**else**

**printf("0\n");**

**}**

**}**

1. Введите 2 целых числа и сравнить эти числа, т.е. выводит на экран 5<6, 7>-12, 4=4
2. Введите 3 числа и определить наибольшее из них.

**int x,y,z,max;**

**printf(“введите 3 числа: ”);**

**scanf(“%d%d%d”,&x,&y,&z);**

**max=x;**

**if(max<y)**

**max=y;**

**if(max<z)**

**max=z;**

**printf(“максимальное из чисел :%d %d %d это %d\n”,x,y,z,max);**

1. Введите 3 вещественных числа и определить наименьшее из них
2. Введите 3 целых числа и определить наименьшее, наибольшее, посчитать сумму, произведение и среднее значение.
3. Введите число, если число >60, то выводим отлично, если число больше 30 и меньше 60, то выводим хорошо, если число больше 17 и меньше 30, то выводим удовлетворительно, если число меньше 17, то выводим плохо.

**int x;**

**printf("Введите число: ");**

**scanf("%d",&x);**

**if(x>60)**

**printf("отлично\n");**

**if(x>30 && x<=60)**

**printf("хорошо \n");**

**if(x>17 && x<=30)**

**printf("удовлетворительно \n");**

**if(x<=17)**

**printf("плохо \n");**

1. Программа, которая в зависимости от номера месяца вывести одно из слов: Зима, Весна, Лето или Осень.

**int n;**

**printf(“Введите число: ”);**

**scanf(“%d”,&n);**

**if(n>=3 && n<=5){**

**printf(“Весна\n”);**

**}**

**else{**

**if(n>=6 && n<=8){**

**printf(“Лето\n”);**

**}**

**else{**

**if(n>=9 && n<=11){**

**printf(“Осень\n”);**

**}**

**else{**

**printf(“Зима\n”);**

**}**

**}**

1. Прямоугольник на плоскости задан координатами своего верхнего левого угла (xl,yl) и нижнего правого угла(xr,yr). Введите координаты точки (x,y) и определить принадлежит ли она прямоугольнику.

(xl,yl)

(xr,yr)

(x,y)

**int xl,yl,xr,yr,x,y;**

**printf(“Введите xl, yl:”);**

**scanf(“%d%d”,&xl,&yl);**

**printf(“Введите xr, yr:”);**

**scanf(“%d%d”,&xr,&yr);**

printf(“Введите x, y:”);

**scanf(“%d%d”,&x,&y);**

**if((x>xl) && (x<xr) && (y>yr) && (y<yl))**

**printf(“Принадлежит!!!!\n”);**

**else**

**printf(“Не Принадлежит!!!!\n”);**

1. Ввести координаты точек А(ха, ya) и B(xb,yb). Посчитать расстояние между данными точками.

**#include<stdio.h>**

**#include<math.h>**

**void main()**

**{**

**double xa,ya,xb,yb,ab;**

**printf("xa=");**

**scanf("%lf",&xa);**

**printf("ya=");**

**scanf("%lf",&ya);**

**printf("xb=");**

**scanf("%lf",&xb);**

**printf("yb=");**

**scanf("%lf",&yb);**

**ab=sqrt((xa-xb)\*(xa-xb)+(ya-yb)\*(ya-yb));**

**printf("Расстояние между A(%.2lf;%.2lf) i B(%.2lf;%.2lf)=%.2lf\n",xa,ya,xb,yb,ab);**

**}**

1. Ввести координаты центра окружности O (xo,yo), радиус r. Ввести также и координаты точки А(ха, ya) и определить попадает ли данная точка в окружность или нет.

Упражнения для закрепления темы Ветвления**.**

**Инструкция *if*** Общие замечания

Приступая к решению задач этого раздела, следует вспомнить, что: инструкция if используется для выбора одного из двух направлений дальнейшего хода программы;

1. выбор последовательности инструкций осуществляется в зависимости от значения *условия* — заключенного в скобки выражения, записанного после if;
2. инструкция, записанная после else, выполняется в том случае, если значение выражения *условие* равно нулю, во всех остальных случаях выполняется инструкция, следующая за условием;
3. если при соблюдении или несоблюдении условия надо выполнить несколько инструкций программы, то эти инструкции следует объединить в группу — заключить в фигурные скобки;
4. при помощи вложенных одна в другую нескольких инструкций if можно реализовать множественный выбор.

**Задачи**

1. Написать программу проверки знания даты основания Санкт-Петербурга. В случае неправильного ответа пользователя, программа должна выводить правильный ответ. Ниже приведен рекомендуемый вид экрана во время выполнения программы (данные, введенные пользователем, выделены полужирным шрифтом).

В каком году был основан Санкт-Петербург?

Введите число и нажмите <Enter>

-> 1705

Вы ошиблись, Санкт-Петербург был основан в 1703 году.

1. Написать программу, которая вычисляет частное двух чисел. Программа должна проверять правильность введенных пользователем данных и, если они неверные (делитель равен нулю), вы давать сообщение об ошибке. Ниже приведен рекомендуемый вид экрана во время выполнения программы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление частного.

Введите в одной строке делимое и делитель,

затем нажмите <Enter>.

-> **12 0**

Вы ошиблись. Делитель не должен быть равен нулю.

1. Написать программу, которая переводит время из минут и секунд в секунды. Программа должна проверять правильность введенных пользователем данных и в случае, если данные неверные, выводить соответствующее сообщение. Рекомендуемый вид экрана во время выполнения программы приведен ниже. Ошибочные данные, введенные пользователем, выделены полужирным шрифтом.

Введите время (минут секунд) -> **2 90**

Ошибка! Количество секунд не может быть больше 60

Для завершения нажмите <Enter>

1. Написать программу вычисления стоимости покупки с учетом скидки. Скидка в 10% предоставляется, если сумма покупки больше 1000 руб. Ниже приведен рекомендуемый вид экрана во время выполнения программы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление стоимости покупки с учетом скидки

Введите сумму покупки и нажмите <Enter>

-> **1200**

Вам предоставляется скидка 10%

Сумма покупки с учетом скидки: 1080.00 руб.

1. Написать программу проверки знания даты начала второй мировой войны. В случае неправильного ответа пользователя,  
   программа должна выводить правильный ответ. Ниже приведен рекомендуемый вид экрана во время выполнения программы (данные, введенные пользователем, выделены полужирным шрифтом).

В каком году началась вторая мировая война?

Введите число и нажмите <Enter> '

-> 1939

Правильно.

1. Написать программу, которая сравнивает два введенных с клавиатуры числа. Программа должна указать, какое число больше, или, если числа равны, вывести соответствующее сообщение. Ниже приведен рекомендуемый вид экрана во время работы программы.

Введите в одной строке два целых числа и нажмите <Enter>.

-> 34 67

34 меньше 67

1. Написать программу, которая проверяет, является ли введенное пользователем целое число четным. Ниже приведен рекомендуемый вид экрана программы во время выполнения программы (данные, введенные пользователем, выделены полу жирным шрифтом).

Введите целое число и нажмите <Enter>

-> 23

Число 23 — нечетное.

1. Написать программу, которая проверяет, делится ли на три введенное с клавиатуры целое число. Ниже приведен рекомендуемый вид экрана программы во время ее работы (данные\* введенные пользователем, выделены полужирным шрифтом).

Введите целое число и нажмите <Enter> -> **451**

Число 451 нацело на три не делится.

1. Напишите программу, которая запрашивает у пользователя номер месяца и затем выводит соответствующее название времени года. В случае, если пользователь введет недопустимое число, программа должна вывести сообщение "Ошибка ввода данных". Ниже приведен рекомендуемый вид экрана во время работы программы.

Введите номер месяца (число от 1 до 12)

-> **11**

Зима

1. Написать программу, которая после введенного с клавиатуры числа (в диапазоне от 1 до 999), обозначающего денежную единицу, дописывает слово "рубль" в правильной форме. Например,  
   12 рублей, 21 рубль и т. д.
2. Написать программу, которая запрашивает у пользователя номер дня недели и выводит одно из сообщений: "Рабочий день», «Суббота" ИЛИ "Воскресенье".

# 2.2 Структуры повторений for (цикл)

В этой главе будут рассмотрены циклические алгоритмы и их реализация на языке С/С++.

В жизни мы часто сталкиваемся с действиями которые повторяются много раз. Например, каждый вечер мы ложимся спать или каждое утро мы завтракаем.

В программировании **цикл** – это участок программы, который повторяется заданное или требуемое количество раз.

Далее мы рассмотрим циклы, характерные для языка Си.

Структура повторения for содержит все элементы, необходимые для повторения, управляемые счетчиком.

Общая форма структуры **for**

**for**(**выражение1**; **выражение2**; **выражение3**)

**{**

**операторы; или оператор;**

**}**

**выражение 1** – это начальное значение управляющей переменной

**выражение 2** – это условие продолжения цикла

**выражение 3** – изменяет управляющую переменную

Чтобы проиллюстрировать работу структуры **for** посмотрим работу следующих программ.

**Пример**: Распечатать все числа от 1 до 10

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

printf(“%d ”,i);

При выполнении структуры **for**, управляющей переменной **i** задается начальное значение **1**. Затем проверяется условие продолжения цикла **i<=10**, т.к. начальное значение равно 1, то условие выполняется и следовательно выполняется тело структуры, т.е. срабатывает функция **printf(“%d ”,i**), которая печатает значение переменной **i**, которая равна **1**. Затем управляющая переменная увеличивается на 1 в выражении **i++** и цикл опять начинается с проверки условия его продолжительности. Этот процесс продолжается до тех пор пока переменная **i,** не примет значение **11**, т.к. при этом значении не выполняется условие продолжительности цикла. Следовательно распечатаются все числа от 1 до 10.

[Циклы можно запускать и в обратном порядке.](https://www.google.by/search?espv=2&biw=1920&bih=965&q=%D0%A6%D0%B8%D0%BA%D0%BB%D1%8B+%D0%BC%D0%BE%D0%B6%D0%BD%D0%BE+%D0%B7%D0%B0%D0%BF%D1%83%D1%81%D0%BA%D0%B0%D1%82%D1%8C+%D0%B8+%D0%B2+%D0%BE%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%BC+%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BA%D0%B5.&spell=1&sa=X&ved=0CBcQvwUoAGoVChMI9J-VloCSxwIVqP1yCh2W8A6U)

Распечатать все числа от 10 до 1

**for(int i=10;i>=1;i--)**

**printf(“%d ”,i);**

**Задачи:**

1. Напечатать числа от 3 до 20.
2. Напечатать числа от 50 до 120.
3. Напечатать все четные числа от 10 до 100.

**for(int i=10;i<=100;i+=2)**

**printf(“%d ”,i);**

1. Напечатать все нечетные числа от 150 до 20.
2. Распечатать все числа от 1 до введенного числа и рядом их квадраты:

Если введенное число 4, то

* 1. 1
  2. 4
  3. 9
  4. 16

1. Введите 2 числа и распечатать все числа от наименьшего до наибольшего
2. Введите 2 числа и распечатать все числа от наибольшего до наименьшего
3. Введите 2 числа и посчитать сумму всех чисел между ними

**#include<stdio.h>**

**void main()**

**{**

**inti,n,m,sum=0;**

**printf("Введите два целых числа: ");**

**scanf("%d%d",&n,&m);**

**if(n>m){**

**i=n;**

**n=m;**

**m=i;**

**}**

**for(i=n;i<=m;i++)**

**{**

**if(i!=n)**

**printf("%+d",i);**

**else**

**printf("%d",i);**

**sum+=i;**

**}**

**printf("=%d\n",sum);**

**}**

1. Введите число и посчитать произведение всех чисел от 1 до этого числа
2. Введите число, которое записывается в переменную n, и посчитать 2n.

**#include<stdio.h>**

**void main()**

**{**

**int n,st=2,i;**

**printf("Введите степень:");**

**scanf("%d",&n);**

**for(i=2;i<=n;i++)**

**st\*=2;**

**printf("2^%d=%d\n",n,st);**

**}**

1. Введите 2 числа. Первое число x - основание, второе число - степень, посчитать xn.

# 2.3 Генерация случайных чисел.

**#include<stdio.h>**

**#include<time.h>**

**#include<stdlib.h>**

**void main()**

**{**

**int n;**

**for(int i=1;i<=120;i++)**

**{**

**n=rand();**

**printf("n%d=%d\n",i,n);**

**}**

**}**

Функция **rand()** генерирует числа в диапазоне от 0 до 32767, т.е. до положительного максимального двухбайтного целого значения и прототип функции **rand** и функции **srand** находится в **<stdlib.h>.** Чтобы регулировать ширину диапазона используется операция вычисления остатка от деления %.

Для диапазона генерации от a до b включительно используем формулу:

n = a + rand()%(b-a+1)

**#include<stdio.h>**

**#include<time.h>**

**#include<stdlib.h>**

**void main()**

**{**

**int n;**

**for(int i=1;i<=20;i++)**

**{**

**n=1+rand()%10;**

**printf("n%d=%d\n",i,n);**

**}**

**}**

для того чтобы при новом вызове программы числа были разные необходимо включить функцию srand (unsingnedint), которая получает в качестве аргумента число в пределах от 0 до 65535, если unsingned int в 2-х байтах и от 0 до 4294967295 если в 4-х байтах, и для того чтобы каждый раз не задавать различные числа передаем функцию time(NULL); прототип, которой находится в <time.h>

***Некоторые функции из <time.h>***

**time\_t time(time\_t\*timer);** Функция **time** определяет текущее календарное время. Функция **time** возвращает наилучшее приближение текущего календарного времени, обеспечиваемое реализацией. Если календарное время недоступно, то возвращаемое значение (**time\_t,** которое является арифметическим типом, используемый для предоставления времени) -1. Если **timer** не является нулевым указателем, то возвращаемое значение помещается в объект, на который ссылается указатель.

**#include<stdio.h>**

**#include<time.h>**

**#include<stdlib.h>**

**void main()**

**{**

**int n;**

**srand(time(NULL));**

**for(int i=1;i<=20;i++)**

**{**

**n=1+rand()%10;**

**printf("n%d=%d\n",i,n);**

**}**

**}**

А если хотим узнать длительность выполнения программы, то можем использовать функцию **difftime()**.

**Double difftime(time\_t time1,time\_t time0);** Функция **difftime** вычисляет разность между двумя значениями календарного времени: **time1** и **time0**. Возвращает время, выраженная в секундах, как значение типа **double**.

Например:

**#include <stdio.h>**

**#include<time.h>**

**void main(){**

**time\_t time0,time1;**

**time0=time(NULL);**

**Здесь сама программа**

**time1=time(NULL);**

**printf("\n\n%lf\n",difftime(time1,time0));**

**}**

# 2.4 Проект для закрепления «Таблица умножения».

Чтобы закрепить знания, рассмотрим практический пример. Нужно написать программу-тест на знания таблицы умножения. Ученик последовательно отвечает на 10 вопросов по таблице умножения. После каждого ответа получает подсказку о том верно или нет он ответил. В случае ошибки, выводится правильный ответ. В конце теста программа подсчитывает результат и ставит оценку.

Обдумаем какие переменные нам потребуются.

i – переменная цикла(счетчик)

ball – переменная для хранения количества правильных ответов.

x,y – переменные для двух случайных чисел в диапазоне от 2 до 9

otvet – переменная где храним ответ ученика.

#include <windows.h>

#include <stdio.h>

#include <time.h>

#include <conio.h>

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

srand(time(0));

printf("Вас приветствует программа проверки знания таблицы умножения\n\n" );

printf("вам будет предложено 10 вопросов. \nДля начала теста нажмите enter\n" );

getch();

system("cls"); //очистка экрана консоли

int i=1,ball=0,x,y,otvet;

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

{

printf("Вопрос номер %d\n",i );

x=rand()%8+2;

y=rand()%8+2;

printf("%d\*%d=",x,y);

scanf("%d",&otvet);

if(otvet==x\*y)

{

printf("Верно \n");

ball++;

}

else

printf("Ошибка. Правильный ответ %d\n",x\*y);

}

printf("Ваша оценка %d\n",ball);

getch();

return 0;

}

# Упражнения для закрепления темы Циклы

***For*** Общие замечания

Приступая к решению задач этого раздела, следует вспомнить, что:

* инструкция for используется для организации циклов с фиксированным, известным во время разработки программы, числом повторений;
* количество повторений цикла определяется начальным значением переменной-счетчика и условием завершения цикла;
* переменная-счетчик должна быть целого (int) типа и может быть объявлена непосредственно в инструкции цикла.

**Задачи**

1. Написать программу, которая выводит на экран ваши имя и фамилию 10 раз.
2. Написать программу, которая выводит таблицу квадратов первых десяти целых положительных чисел. Ниже приведен рекомендуемый вид экрана во время работы программы.

Таблица квадратов.

|  |  |
| --- | --- |
| Число | Квадрат |
| 1 | 1 |
| 2 | 4 |
| 3 | 9 |
| 4 | 16 |
| 5 | 25 |
| 6 | 36 |
| *1* | 49 |
| 8 | 64 |
| 9 | 81 |
| 10 | 100 |

1. Написать программу, которая выводит таблицу квадратов первых пяти целых положительных нечетных чисел. Ниже приведен рекомендуемый вид экрана во время работы программы.

Таблица квадратов нечетных чисел.

Число Квадрат

|  |  |
| --- | --- |
| 1 | 1 |
| 3 | 9 |
| 5 | 25 |
| 7 | 49 |
| 9 | 81 |

1. Написать программу, которая вычисляет сумму первых ***n*** целых положительных целых чисел. Количество суммируемых чисел должно вводиться во время работы программы. Ниже приведен рекомендуемый вид экрана (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление суммы положительных чисел.

Введите количество суммируемых чисел -> ***20***

Сумма первых 20 положительных чисел равна 210.

1. Написать программу, которая вычисляет сумму первых ***n*** целых положительных четных целых чисел. Количество суммируемых чисел должно вводиться во время работы программы. Ниже приведен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление суммы четных положительных чисел.

Введите количество суммируемых чисел и нажмите <Enter>

-> ***12***

Сумма первых 12 положительных четных чисел равна 156

1. Написать программу, которая вычисляет сумму первых ***n***  
   членов ряда: 1+1/2+1/3+1/4 … Количество суммируемых членов ряда задается во время работы программы. Ниже приведен рекомендуемый вид экрана (данные, введенные пользователем, выделены полужирным шрифтом).

Вычисление частичной суммы ряда: 1+1/2+1/3+ ...

Введите кол-во суммируемых членов ряда -> 15

Сумма первых 15 членов ряда равна 3.3182

1. Написать программу, которая выводит таблицу степеней  
   двойки от нулевой до десятой. Ниже приведен рекомендуемый  
   вид экрана во время работы программы.

таблица степеней двойки

|  |  |
| --- | --- |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 5 | 32 |
| 6 | 64 |
| 7 | 128 |
| 8 | 256 |
| 9 | 512 |
| 10 | 1024 |

1. Написать программу, которая вводит с клавиатуры 5 дробных чисел и вычисляет их среднее арифметическое. Рекомендуемый вид экрана во время выполнения программы приведен ниже. Данные, введенные пользователем, выделены полужирным шрифтом.

Вычисление среднего арифметического последовательности

дробных чисел. После ввода каждого числа нажимайте <Enter>

***-> 5.4***

***-> 7.8***

***-> 3.0***

***-> 3.5***

***-> 2.3***

Среднее арифметическое введенной последовательности: 4.20

Для завершения нажмите <Enter>

1. Написать программу, которая генерирует последовательность из 10 случайных чисел в диапазоне от 1 до 10, выводит эти числа на экран и вычисляет их среднее арифметическое. Рекомендуемый вид экрана во время выполнения программы приведен ниже.

Случайные числа.

9 2 6 2 8 1 2 1 2 сред, **арифм.** 3.6

# 2.5 Структура повторения while (пока)

|  |  |  |
| --- | --- | --- |
| **while(условие)**  **оператор;** | **while(условие)**  **{**  **оператор1;**  **оператор2; и т.д.**  **}** | **do**  **{**  **оператор;**  **}while(условие);** |

Оператор while организует повторное выполнение одного оператора или несколько операторов, заключенных в фигурных скобках {}, до тех пор пока <условие> верно. Оператор *while,* как и оператор *for* называются *операторами цикла с предусловием*, т.к. истинность условия проверяется перед входом в цикл. Оператор *do … while* называется *оператором цикла с постусловием* и используется в тех случаях, когда необходимо обеспечить выполнение цикла хотя бы один раз.

Выражение вычисляется и анализируется в начале перед выполнением тела цикла. Поэтому тело цикла может ни разу не выполниться, если в самом начале на первом шаге выражение ложно.

Для начинающих изучать программирование с помощью С необходимо обратить внимание на отличие while от сокращённой формы оператора if. Выражение в if анализируется один раз, если, конечно, if не в цикле, и последовательность операторов либо не выполняется, либо выполняется только один раз. Оператор while повторяет как вычисление и анализ выражения, так и выполнение последовательности операторов.

Одна или несколько переменных выражения должны меняться в операторах тела цикла. При этом изменение должно быть таким, чтобы на некотором шаге выражение стало ложным. В противном случае эта часть программы зациклится, то есть будет повторяться бесконечное количество раз.

**Задачи:**

1. Посчитать сумму всех вводимых чисел, ввод чисел завершается, когда вводим 99.

**#include<stdio.h>**

**void main()**

**{**

**int x,sum=0;**

**printf(“Введите число: “);**

**scanf(“%d”,&x);**

**while(x!=99)**

**{**

**sum+=x;**

**printf(“Введите число: “);**

**scanf(“%d”,&x);**

**}**

**printf(“sum=%d\n”,sum);**

**}**

В данной программе число 99 не входит в сумму. А если нужно, чтобы 99 вошло в сумму, то необходимо написать программу следующим образом:

**#include<stdio.h>**

**void main()**

**{**

**int x,sum=0;**

**do**

**{**

**printf(“Введите число: “);**

**scanf(“%d”,&x);**

**sum+=x;**

**} while(x!=99);**

**printf(“sum=%d\n”,sum);**

**}**

1. Посчитать количество всех вводимых чисел, ввод чисел завершается, когда вводим 999.

**#include<stdio.h>**

**void main(){**

**int x,k=0;**

**printf(“Введите число:”);**

**scanf(“%d”,&x);**

**while(x!=999)**

**{**

**k++;**

**printf(“Введите число:”);**

**scanf(“%d”,&x);**

**}**

**printf(“количество введенных чисел = %d\n”,k);**

**}**

1. Посчитать количество всех вводимых нечетных чисел, ввод чисел завершается, когда вводим 22.
2. Посчитать сумму всех вводимых чисел и число четных, ввод чисел завершается, когда вводим 0.
3. Посчитать количество четных, положительных чисел, ввод чисел завершается, когда вводим 0.

**#include<stdio.h>**

**void main()**

**{**

**int x,k=0;**

**printf(“Введите число: ”);**

**scanf(“%d”,&x);**

**while(x)**

**{**

**if(x%2==0 && x>0)**

**k++;**

**printf(“Введите число: ”);**

**scanf(“%d”,&x);**

**}**

**printf(“Количество четных, положительных чисел = %d\n”,k);**

**}**

1. Посчитать количество нечетных, отрицательных чисел и произведение всех чисел, ввод чисел завершается, когда вводим 0.
2. Сумма 1000$ положена в банк, при этом годовой процент 3%. Через сколько лет сумма увеличится в 5 раз.

**#include <stdio.h>**

**void main ()**

**{ int n,r;**

**double s,t;**

**r=5; s=1000;**

**t=r\*s; n=0;**

**while(s<=t)**

**{s+=0.03\*s;**

**n++;}**

**printf(" year=%d\n",n);**

**}**

1. Программа, которая находит наименьшее из нескольких вещественных чисел. Предполагается, что первое введенное число задает количество последующих вводимых чисел.
2. Найти наибольшую цифру целого десятичного числа.
3. Ввести число и распечатать те цифры данного числа, которые делятся без остатка на три (используя / и %)

**#include<stdio.h>**

**void main()**

**{**

**int n,z;**

**printf(“Введите число: “);**

**scanf(“%d”,&n);**

**while(n){**

**z=n%10;**

**if(z%3==0)**

**printf(“%d “,z);**

**n=n/10;**

**}**

**}**

1. Ввести число и распечатать сумму цифр данного числа.
2. Ввести пятизначное число и определить является ли данное число палиндромом или нет. Палиндром – это число или текст, который одинаково читается слева направо и справа налево.
3. Если между цифрами двухзначного числа записать 0, то полученное трехзначное число будет в 7 раз больше за начальное число. Найти это число.

#include <conio.h>

#include <stdio.h>

void main ()

{ int m,c,d,a,b;

m=0;

for( a=1;a<=9;a++)

for( b=0;b<=9;b++)

{

c=10\*a+b;

d=100\*a+b;

if(d==7\*c)

{

m=c;

}

}

printf("число=%d",m);

getch();

}

1. Разработать программу, которая должна определять заработную плату для каждого работника. Компания выплачивает каждому служащему почасовую зарплату за первые 40 часов работы и выплачивает в полуторном размере за все рабочие часы сверх 40. Вам дан список сотрудников, число часов отработанных каждым сотрудником и почасовая ставка каждого сотрудника. Программа должна ввести эти данные для каждого сотрудника, распечатать и вывести на экран его суммарную зарплату. Например:

**Введите число рабочих часов (-1, если ввод закончен): 39**

**Введите почасовую ставку работника: 10.0**

**Заработок: 390.00**

**Введите число рабочих часов (-1, если ввод закончен): 40**

**Введите почасовую ставку работника: 10.0**

**Заработок: 400.00**

**Введите число рабочих часов (-1, если ввод закончен): 41**

**Введите почасовую ставку работника: 10.0**

**Заработок: 415.00**

# 2.6 Структура многомерного выбора switch

В программировании часто встречаются задачи выбора одного варианта решения задачи из многих возможных. Это можно сделать и с помощью вложенных **if……else.** Однако более удобный способ – использование оператора **switсh**, который выполняет одно из множество действий в зависимости от значения проверяемого выражения. **Switсh** отличается от **if** тем, что он выполняет операцию строгого равенства, в то время как оператор **if** может вычислять логические выражения и отношения.

общий формат оператора **switсh** :

**switch (**выражение**)**

{

**case** const1**:**

operator1; **break;**

**case** const2**:**

operator2; **break;**

**case** const3**:**

operator3; **break;**

**case** const4:

**case**const5**:**

operator4; **break;**

**default**

operator5; **break;**

}

**Свойства**:

**Выражение** должно иметь целочисленный тип (допустимо использование символьной константы, перечислений и вызов функции, возвращающей целочисленное значение). Значение выражения сравнивается со всеми находящимися внутри конструкции константами, стоящими после **case.**

**Оператор**, указанный после **case** выполняется, если значение выражения равно соответствующей константе. Случай с const4 и const5 означает, что если значение выражения равно const4 или const5, то выполняется operator4. Если ни с одной из констант совпадений нет, то выполняется оператор стоящий после **default**, если конечно он есть, так как его существование не обязательно. **default** может быть записан в любом месте.

**Константы** сравниваются в той последовательности, в которой они перечислены и не может быть двух констант имеющих одинаковое значение в одном операторе **switch**.

**Break** передает управление за пределами оператора **switch**. Если после какого-либо оператора отсутствует **break**, то выполняется оператор стоящий после следующего **case,** т.е. он автоматически считается правильным.

**Примеры:**

1. Вывести целое однозначное число z прописью.

Это можно сделать с помощью if.

Второй вариант с оператором switch вместо вложенного if нагляднее и легче для понимания и отладки.

**int z;**

**scanf("%d",&z);**

**switch (z)**

**{ case 0: printf(«нуль”); break;**

**case 1: printf(”один”); break;**

**case 2: printf(”два”); break;**

**case 3: printf(”три”); break;**

**case 4: printf(”четыре”); break;**

**case 5: printf(”пять”); break;**

**case 6: printf(”шесть”); break;**

**case 7: printf(”семь”); break;**

**case 8: printf(”восемь”); break;**

**case 9: printf(”девять”); break;**

**default: printf(«Ошибка ”);**

**}** // Конец оператора switch.

Здесь оператор, следующий после switch.

Работает оператор так. Пусть, например, z=3. Тогда первые две ветви пропускаются, выполняется printf(”три”);и break передаёт управление на оператор, следующий после switch. Если z не является однозначным числом, например, числом 30, то первые десять ветвей пропускаются, выводится “Ошибка” и выполняется оператор, следующий после switch.

1. .Вводим числа ввод чисел заканчивается 0. Посчитать сколько раз вводилось каждое из чисел то 5 до 8,.

**#include<stdio.h>**

**void main()**

**{**

**int x,k\_5=0,k\_6=0,k\_7=0,k\_8=0;**

**do**

**{**

**printf("Введите числа от 5 до 8: ");**

**scanf("%d",&x);**

**switch(x)**

**{**

**case 5:**

**k\_5++;**

**break;**

**case 6:**

**k\_6++;**

**break;**

**case 7:**

**k\_7++;**

**break;**

**case 8:**

**k\_8++;**

**break;**

**default:**

**printf("Введено неправильное число:\n");**

**break;**

**}**

**}while(x);**

**printf("Количество:\n5\t%d\n6\t%d\n7\t%d\n8\t%d\n",k\_5,k\_6,k\_7,k\_8);**

**}**

1. Даны три монеты достоинством 5, 10, 15. Напишите программу, которая определит, можно ли заданную сумму набрать из каких–либо двух монет

**int x;**

**printf("Введите сумму: ");**

**scanf("%d",&x);**

**switch(x){**

**case 5+10:**

**printf(“Yes\n”);**

**break;**

**case 5+15:**

**printf(“Yes\n”);**

**break;**

**case 10+15:**

**printf(“Yes\n”);**

**break;**

**default:**

**printf("No:\n");**

**break;**

**}**

1. Переделать программу 3 используя оператор **if/else**
2. Калькулятор1. Ввести число, ввести символ (‘+’, ‘-‘, ‘\*’, ‘/’) затем ввести число. Программа считает и выводит результат.

**#include<stdio.h>**

**void main()**

**{**

**double x,y;**

**char ch;**

**printf("Введите число: ");**

**scanf("%lf",&x);**

**fflush(stdin);**

**printf("Введите действие: ");**

**fflush(stdin);**

**scanf("%c",&ch);**

**printf("Введите число: ");**

**scanf("%lf",&y);**

**switch(ch)**

**{**

**case '+': x+=y;**

**break;**

**case '-': x-=y;**

**break;**

**case '\*': x\*=y;**

**break;**

**case '/': x/=y;**

**break;**

**}**

**printf("=%.2lf\n",x);}**

5. Калькулятор 2. Программа требует ввода числа, символа и числа до тех пор пока не введем символ ‘=’.

6. Калькулятор 3. Программа требует ввода числа, символа и числа до тех пор пока не введем символ ‘=’. Добавить в программу подсчет квадратного корня числа, получения обратного числа, т.е. если число положительное, то становиться отрицательным и наоборот, подсчет процента.

# 2.7 Операторы continue и break

Операторы **break** и **continue** изменяют поток управления. Когда оператор **break** выполняется в структурах **while, for, do/while**или **switch**, происходит немедленный выход из структуры и программа продолжает выполнять операторы, написанные после структуры. Обычное назначение оператора **break** – досрочно прерывать цикл или пропустить оставшуюся часть структуры **switch.** Оператор **continue** выполняется в структурах **while, for, do/while** вызывает пропуск оставшейся части тела структуры и начинает выполнять следующую итерацию цикла. В структурах **while**и **do/while**после выполнения оператора **continue** производится проверка условия продолжения цикла, а в структуре **for**выполняется выражение приращения, а затем проверка условия продолжения цикла.

**Примеры:**

1. Использование оператора **break** в цикле **for**

**#include<stdio.h>**

**void main()**

**{**

**for(int x=1;x<=10;x++)**

**{**

**if(x==5)**

**break;**

**printf("%d ",x);**

**}**

**printf("\nЦикл прерван при x=%d\n",x);**

**}**

2. Использование оператора **break** в цикле **while**

**#include<stdio.h>**

**void main()**

**{**

**int x=1;**

**while(x<=10)**

**{**

**if(x==5)**

**break;**

**printf("%d ",x);**

**x++;**

**}**

**printf("\nЦикл прерван при x=%d\n",x);**

**}**

1. Использование оператора **continue** в цикле **for**

Вывести все числа от 1 до 10 кроме числа 5.

**#include<stdio.h>**

**void main()**

**{**

**for(int x=1;x<=10;x++)**

**{**

**if(x==5)**

**continue;**

**printf("%d ",x);**

**}**

**}**

4. Использование оператора **continue** в цикле **while**

**#include<stdio.h>**

**void main()**

**{**

**int x=1;**

**while(x<=10)**

**{**

**if(x==5)**

**{**

**x++;**

**continue;**

**}**

**printf("%d ",x);**

**x++;**

**}**

**}**

Что получим после выполнения данной программы?

**int x, z=0;**

**for(x=0;x<5;x++)**

**{  
 z++;**

**if(x<3)**

**continue;**

**if(x>3)**

**break;**

**z++;**

**}**

**printf(“z=%d\n”,z);**

1. Написать данную программу с использованием оператора while.

**Задачи:**

**1.** Программа, которая определяет, является ли введенное целое число совершенным или нет. Целое число является совершенным, если его сомножители, включая 1 (но не само число) в сумме дает это число. Например, 6 – это совершенное число, т.к. 6=1+2+3.

**#include<stdio.h>**

**void main()**

**{**

**int n,i,sum=1;**

**printf("Введите число: ");**

**scanf("%d",&n);**

**for(i=2;i<=n/2;i++)**

**{**

**if(!(n%i))**

**sum+=i;**

**}**

**if(sum==n)**

**printf("\a\aЧисло Совершенное!!!!!\n");**

**else**

**printf("нет!!!!!\n");**

**}**

**2.** Программа, которая получает целое число и печатает число в обратном порядке. Например, если ввели 12345, то программа печатает 54321.

**3**. Ввести целое число и определить является ли данное число простым или нет.

**#include<stdio.h>**

**void main()**

**{**

**int n,i,sum=1;**

**printf("Введите число: ");**

**scanf("%d",&n);**

**for(i=2;i<=n/2;i++)**

**{**

**if(!(n%i))**

**break;**

**}**

**if(i==n/2+1)**

**printf("\a\aЧисло ПРОСТОЕ!!!!!\n");**

**else**

**printf("нет!!!!!\n");**

**}**

4. Вычислитьсумму

**int k,n,j,p,x;**

**double a,s;**

**printf("\n input n,x");**

**scanf("%d%d",&n,&x);**

**s=1;**

**for(k=1;k<=n;k++)**

**{p=1;**

**for(j=1;j<=k;j++)**

**p\*=j;**

**a=pow(x,k);**

**s+=a/p; }**

**printf("s=%5.1lf",s);**

5. Вычислитьсумму

# 2.8 Контрольные вопросы:

1. Выполнения каждой программы на “С” начинается с функции \_\_\_\_\_\_\_\_\_\_\_
2. Каждый оператор заканчивается \_\_\_\_\_\_\_\_\_\_
3. Функция \_\_\_\_\_\_\_\_\_ выводит информацию на экран.
4. Функция \_\_\_\_\_\_\_\_\_ вводит информацию с клавиатуры.
5. Какие из следующих операторов верны для подсчета y=ax3 + 7?
   1. y = a \* x \* x \* (x + 7);
   2. y = (a \* x) \* x \* (x + 7);
   3. y = (a \* x) \* x \* x + 7;
   4. y = a \* (x \* x \* x) + 7;
   5. y = a \* x \* (x \* x + 7);

f) y = a \* x \* x \* x +7;

1. Что такое понятие цикла?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. Укажите операторы цикла, которые вы знаете: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. Укажите различия между циклами. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
4. Чем отличаются оператор if оператор switch? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
5. Верны ли следующие утверждения:
   1. В структуре switch обязательно должна быть метка default.
   2. Выражение (x>y&&a<b) истинно, если x>y или a<b
   3. Выражении содержащее операцию || , истинно, если хотя бы один операнд истина.
6. Что напечатает следующая программа?

**int y, x=1,total=0;**

**while(x<=10)**

**{**

**y=x\*x;**

**printf("%d\n",y);**

**total+=y;**

**x++;**

**}**

**printf("\nTotal=%d\n",total);**

1. Имея определения int x = 5, y = 2; float z; Как правильно вычислить деление целых чисел?  
   a) z = x / y;   
   б) z = (float)x / y;   
   в) z = x / (float) y;   
   г) z = (float)(x / y);   
   д)z = (float) x / (float) y;
2. Какие из перечисленных утверждений верны?  
   а) Длина переменной типа short всегда меньше длины переменной типа int   
   б) Длина переменной типа short всегда меньше длины переменной типа long   
   в) Длина переменной типа int всегда меньше длины переменной типа long   
   г) Длина переменной типа float всегда меньше длины переменной типа double   
   д) Длина переменной типа float всегда меньше длины переменной типа long double
3. Какую и сколько из строк case следует удалить, чтобы оставшаяся конструкция стала синтаксически корректной?  
   *switch(a)  
   {  
   case 1: a++; break;  
   case 2:  
   case 3: a--; return;  
   case 4:  
   case 5: a+=1; continue;  
   }*

а) пятую

б) вторую, четвертую и пятую

в) четвертую и пятую

г) все и так нормально

д) вторую и четвертую

1. Может ли оператор continue завершать ветвь case оператора switch, предотвращая, подобно оператору break, выполнение следующей ветви case?

а) да, может

б) нет, не может

в) только если оператор switch вложен в оператор цикла

г) только если оператор цикла вложен в оператор switch

**Задачи:**

1. Напишите программу, которая вводит числа. Ввод чисел завершается, когда вводим 0. Найти наименьшее и наибольшее из введенных чисел.
2. Ввести целое число и распечатать цифры данного числа, через 3 пробела друг от друга. Например, если ввели 234567, то получаем 7 6 5 4 3 2
3. Программа, которая находит все совершенные числа от 1 до 30000.

§3 МАССИВЫ

В этой главе будет введено понятие массивов и показано, как работать с одномерными массивами с фиксированной размерностью, то есть с помощью индексов, без использования указателей.

# 3.1 Одномерные массивы

Массив – это группа расположенных друг за другом в памяти элементов одного типа и имеющих одно общее имя. Определение массива должно содержать информацию о типе, количестве элементов и имени массива. Доступ к элементам массива осуществляется по имени массива и индексу элемента.

*Индекс* – это порядковый номер элемента в массиве; индекс первого элемента всегда 0 далее следуют целые положительные числа.

Доступ к элементам массива в C/С++ осуществляется двумя способами. Первый, с помощью порядкового номера элемента массива (индекса), характерен для многих языков программирования. Он более простой и привычный для тех, кто изучал язык Pascal или другой язык. В качестве индекса можно использовать выражение, в том числе константу или переменную целого или совместимого с ним типа, и нельзя использовать выражение вещественного типа. Профессионально в C/С++, в отличие от более простых языков, есть возможность обрабатывать массивы, используя специальный тип данных, который называется указателем. Переменная такого типа содержит адрес ячейки, то есть в упрощённом виде номер её первого байта. В C/С++ существует связь между массивами и указателями. Этот способ будет рассмотрен позже.

Массив характеризуется следующим:

* все элементы массива имеют одинаковый тип;
* каждый элемент массива занимает одинаковый объём памяти;
* массив располагается в оперативной памяти, а не на внешнем устройстве;
* элементы массива занимают подряд идущие ячейки.

Массивы могут иметь одну или несколько размерностей. Их изучение начинаем с более простого одномерного массива, который называют вектором, подразумевая вектор в n−мерном пространстве. Для доступа к элементу такого массива достаточно одного индекса. Работа с двухмерными массивами (матрицами) рассматривается позже. Теоретически разрешены массивы с тремя и более размерностями, но на практике они используются редко, так как такие массивы занимают большой объём оперативной памяти.

Одномерный массив с фиксированной размерностью, который назовём статический, объявляется в общем виде следующим образом:  
тип\_данных имя\_массива [размер];

размер может быть только целой константой.

Несколько примеров объявления массива:

**int** x[6]; где x – это имя массива, **int** – это тип, 6 – это количество элементов, или размер массива.

x

|  |  |
| --- | --- |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |

**int** x[2\*4];

**int** x[]={1,3,5,6,7,8};

x

|  |  |
| --- | --- |
| 0 | 1 |
| 1 | 3 |
| 2 | 5 |
| 3 | 6 |
| 4 | 7 |
| 5 | 8 |

**Int** x[15]={1,5,-90,7}; в этом случае первые 4 элемента будут соответственно 1,5,-90,7, а остальные будут 0

т.к. размером массива может быть только константа или константное выражение, то

**const int** n =7;

**int** x[n];

*следующее объявление массива неверно:*

**int** n =7;

**int** x[n];

n — размерность (или размер) массива в виде целочисленной константы или константного выражения. Эта величина определяет количество элементов массива, а, следовательно, и количество ячеек оперативной памяти, зарезервированной для массива. Например: float A[10];. Но лучше второй вариант: const int n=10; float A[n]; и вместо константы 10 используется её имя, то есть n. Преимущество второго способа следующее. Если надо будет изменить размерность массива, то это достаточно сделать в одном месте программы при определении константы, В первом способе в тексте программы надо найти несколько раз встречающуюся константу 10 и заменить её другим числом.

С точки зрения времени (этапа), когда распределяется память под массив, существуют два их вида. Кроме рассматриваемого статического можно создавать динамический массив, размерность которого может быть не только константой, а и переменной. Память для такого массива выделяется во время выполнения программы, и если массив не нужен, память для него можно освободить.

В отличие от динамического массива, для статического память для размещения n чисел указанного типа (10 вещественных чисел) резервируется на этапе компиляции. Для массива требуется память объёмом k\*n =4\*10=40 байт, где k — количество байт для размещения одного элемента указанного типа (в примере одного числа типа float). Эта память сохраняется на всё время выполнения функции или блока, где описан массив.

Во многих современных системах программирования, в том числе и в С/С++, нумерация элементов массива начинается с 0. Тогда если n — количество элементов, то A[n-1] — последний элемент массива. Это связано с использованием указателей при работе с массивами (см. вторую часть книги). Поэтому в нашем примере индекс изменяется от 0 до n-1 включительно, то есть индекс последнего элемента массива на единицу меньше его размерности. Элементы массива обозначаются следующим образом: A[0], A[1], A[2] ,…, A[n-1]. В C/С++ отсутствует проверка границ массивов. Можно выйти за его границу и записать значение в некоторую переменную или даже в код программы. О таком контроле должен позаботиться сам программист.

Обратите внимание НУМЕРАЦИЯ ЭЛЕМЕНТОВ МАССИВА НАЧИНАЕТСЯ С 0! ( нулевой элемент существует ).

При использовании статических массивов возникают проблемы в случае, если размер массива заранее мы не знаем. В таком случае объявляем массив максимальной размерности, которая для некоторых задач может быть известна. Реальную размерность вводим и используем её далее в программе, например, в циклах и для других целей:

const nmax=100;

float X[nmax];

int n;

printf("введите размер массива array <100");

scanf("%d",&n);

Дальше работаем с n (а не с nmax) элементами массива. Такой способ проще, но неэффективен с точки зрения распределения памяти, так как мы “запрашиваем” больше памяти, чем реально используем. В таких случаях профессионально используются более эффективные динамические массивы.

Ввод массива осуществляется следующим образом:

**for(i=0;i<n; i++)**

**{**

**printf("[%d]=",i);**

**scanf("%d",&x[i]);**

**}**

Вывод массива осуществляется следующим образом:

**for(i=0;i<n; i++)**

**printf("[%d]=%d\n", i, x[i]);**

# 3.2 Примеры использования массивов

1. Создать массив, заполнить его нулями и вывести на экран содержимое.

#include<stdio.h>

#include<conio.h>

int main()

{

int m[ 10 ]; //Объявляем массив m из 10 элементов

int i; //Переменная цикла

//Инициализация массива :

for( i = 0; i < 10; i++ ) //От 0 до 9 с шагом 1

m[ i ] = 0; //Заполняем массив нулями

//Вывод массива :

for( i = 0; i < 10; i++ ) //От 0 до 9 с шагом 1

printf( "m[ %d ] = %d \n", i, m[ i ] ); //Выводим элементы массива на экран

getch();

return 0;

}

1. Создать массив, инициализировав его с помощью списка инициализации и вывести на экран содержимое.

#include<stdio.h>

#include<conio.h>

int main()

{

int m[ 10 ]; //Объявляем массив m из 10 элементов

int i; //Переменная цикла

//Инициализация массива :

for( i = 0; i < 10; i++ ) //От 0 до 9 с шагом 1

m[ i ] = 0; //Заполняем массив нулями

//Вывод массива :

for( i = 0; i < 10; i++ ) //От 0 до 9 с шагом 1

printf( "m[ %d ] = %d \n", i, m[ i ] ); //Выводим элементы массива на экран

getch();

return 0;

}

1. Создать массив, заполнить его четными числами от 2 до 20 и вывести на экран содержимое.

#include<stdio.h>

#include<conio.h>

int main()

{

int m[ 10 ]; //Объявляем массив m из 10 элементов

int i; //Переменная цикла

//Инициализация массива :

for( i = 0; i < 10; i++ ) //От 0 до 9 с шагом 1

m[ i ] = ( i + 1 ) \* 2; //Заполняем массив четными числами от 2 до 20

//Вывод массива :

for( i = 0; i < 10; i++ ) //От 0 до 9 с шагом 1

printf( "m[ %d ] = %d \n", i, m[ i ] ); //Выводим элементы массива на экран

getch();

return 0;

}

1. Изменить программу так, чтобы она заполняла массив по убыванию. То есть нулевой элемент массива – 10, первый – 9, второй – 8 и т.д.
2. Сумма элементов массива.

#include<stdio.h>

#include<conio.h>

int main()

{

int m[ 10 ]; //Объявляем массив m из 10 элементов

int i; //Переменная цикла

int sum = 0; //Переменная для суммы массива

for( i = 0; i < 10; i++ ) //От 0 до 9 с шагом 1

{

printf("[%d]=",i);

scanf( "%d", &m[ i ] ); //Вводим элементы массива

sum += m[ i ]; //Суммируем каждый элемент массива

}

printf( "Sum is : %d", sum ); //Выводим значение суммы на экран

getch();

return 0;

}

1. Программа запрашивает с клавиатуры десять вещественных чисел, а затем выводит их в обратном порядке.
2. Посчитать сумму элементов массива целых чисел с четным индексом

**#include<stdio.h>**

**void main()**

**{**

**const int n=6;**

**int x[n],i,sum=0;**

**for(i=0;i<n;i++){**

**printf("x[%d]=",i);**

**scanf("%d",&x[i]);**

**if(i%2==0)**

**sum+=x[i];**

**}**

**printf("sum=%d\n",sum);**

**}**

1. Ввести массив вещественных чисел, размерностью 6. Посчитать: а) количество отрицательных элементов массива; б) сумма положительных элементов массива; в) произведение элементов находящихся под нечетным индексом.
2. В школе провели опрос о качестве питания в столовой. Опросили 40 школьников, каждый из которых должен был выставить балл от 1 до 10. Написать программу, которая подсчитывает результаты опроса. В школе провели опрос о качестве питания в школьной столовой. Ученики оценивали от 1 до 10 качество питания.

#include<stdio.h>

#include<conio.h>

#define vs 11

#define rs 40

int main()

{

int m[ vs ]; //Массив счетчиков ответов

unsigned responce[ rs ] = {3,2,4,5,8,5,7,

4,8,4,2,1,7,4,8,6,4,3,8,4,5,7,9,1,5,4

,2,6,10,5,3,7,5,3,4,6,7,4,6,7}; //Инициализация массива ответов

int i; //Переменная цикла

for( i = 0; i < vs; i++ ) //От 0 до количества баллов vs с шагом 1

m[ i ] = 0; //Обнуляем счетчики ответа

for( i = 0; i < rs; i++ ) //От 0 до количества ответов rs с шагом 1

m[ responce[ i ] ]++; //Считаем количество ответов для каждого балла

for( i = 1; i < vs; i++ ) //От 0 до количества баллов vs с шагом 1

printf( "Rating %2d is : %2d \n", i, responce[i] ); //Выводим на экран количество ответов за каждый балл

getch();

return 0;

}

1. Задан массив размерностью 10. Ввести число, которое необходимо записать под 5-им индексом, причем следующим образом введенное число записывается на 5-ом месте, число записанное на 5-ом месте смещается на 6-ое, 6-ое на 7-ое и т.д., в последствии последнее число которое было записано в массиве теряется.
2. Нахождение максимального элемента массива

**#include<stdio.h>**

**void main()**

**{**

**const int n=6;**

**int x[n],i,max;**

**for(i=0;i<n;i++){**

**printf("x[%d]=",i);**

**scanf("%d",&x[i]);**

**}**

**max=x[0];**

**for(i=1;i<n;i++)**

**{**

**if(x[i]>max)**

**max=x[i];**

**}**

**printf("max=%d\n",max);**

**}**

1. Нахождение минимального элемента массива вещественных чисел.
2. Нахождение максимального элемента массива вещественных чисел и его место

**#include<stdio.h>**

**void main()**

**{**

**const int n=6;**

**double x[n],max;**

**int i,i1;**

**for(i=0;i<n;i++){**

**printf("x[%d]=",i);**

**scanf("%lf",&x[i]);**

**}**

**max=x[0];**

**i1=0;**

**for(i=1;i<n;i++){**

**if(x[i]>max){**

**max=x[i];**

**i1=i;**

**}**

**}**

**printf("x[%d]=max=%.2lf\n",i1,max);**

**}**

1. Нахождение минимального элемента массива и его место
2. Ввести массив из 20 элементов и найти максимальный элемент от 0 до 9 и минимальный от 10 до 19 и поменять их местами.

Прежде чем двигаться дальше обязательно выполните упражнения для закрепления.

# Упражнения для закрепления темы массивы.

1. Написать программу, которая вводит с клавиатуры одномерный массив из 5 целых чисел, после чего выводит количество ненулевых элементов. Перед вводом каждого элемента должна выводиться подсказка с номером элемента.

Ввод массива целых чисел.

После ввода каждого числа нажмите <Enter>

а[0] -> 12

а[1] -> 0

а[2] -> 3

а[3] -> -1 . .

а[4] -> 0

В массиве 3 ненулевых элемента.

2. Написать программу, которая выводит минимальный элемент введенного с клавиатуры массива целых чисел. Ниже приведен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Поиск минимального элемента массива.

Введите в одной строке элементы массива (5 целых чисел) и нажмите <Enter>

-> **23 0 45 -5 12**

Минимальный элемент массива: **-5**

3. Написать программу, которая вычисляет среднее арифметическое ненулевых элементов введенного с клавиатуры массива целых чисел. Ниже приведен рекомендуемый вид экрана во время работы программы (данные, введенные пользователем, выделены полужирным шрифтом).

Введите элементы массива (10 целых чисел) в одной строке и

нажмите <Enter>.

-> 23 0 45 -5 12 0 -2 30 0 64

Сумма элементов массива: 184

Количество ненулевых элементов: 7

Среднее арифметическое ненулевых элементов: 23.86

4. Написать программу, которая вычисляет среднее арифметическое элементов массива без учета минимального и максималь­ного элементов массива. Ниже приведен рекомендуемый вид экрана во время работы программы

Среднее арифметическое без учета min и max значений.

Введите массив (10 целых чисел в одной строке)

->12 10 5 7 15 4 10 17 23 7

Минимальный элемент: 4

Максимальный элемент: 23

Среднее арифм. без учета min и max значений: 10.36

5. Написать программу, которая вычисляет среднюю (за неделю) температуру воздуха. Исходные данные должны вводиться во время работы программы. Рекомендуемый вид экрана приведен ниже

Введите температуру воздуха за неделю.

Понедельник -> 12

Вторник -> 10

Среда -> 16

Четверг -> 18

Пятница -> 17

Суббота -> 16

Воскресенье -> 14

Средняя температура за неделю: 14.71 град.

6. Известно, что в массиве есть только числа 0, 1, 2 в любом количестве и порядке следования. Преобразовать массив следующим образом: в начало поместить единицы, затем двойки и, наконец, нули. Например, для массива {2, 1, 1, 2, 0, 2, 0, 1, 2, 2} получим {1, 1, 1, 2, 2, 2, 2, 2, 0, 0}.

Дополнительный массив не формировать.

7. Для массива с числами вывести гистограмму массива в виде:

Element Histogram

10 \*\*\*\*\*\*\*\*\*\*

3 \*\*\*

8 \*\*\*\*\*\*\*\*

4 \*\*\*\*

6 \*\*\*\*\*\*

2 \*\*

8. Считайте пятизначное число и запишите каждую цифру числа в отдельный элемент массива.

# 3.3 Сортировки массивов

1. Сортировка массива по возрастанию, пузырьковым методом.

Просматривается весь массив “сверху вниз” и меняются стоящие рядом элементы в том случае, когда “нижний” элемент меньше, чем “верхний”. Таким образом, после первого просмотра, самый “тяжелый” элемент массива попадает на самое последнее место, т.е. выталкивается в низ. Далее алгоритм повторяется для оставшихся n-1 элементов.

**#include<stdio.h>**

**void main()**

**{**

**const int n=6;**

**int x[n],i,buf;**

**for(i=0;i<n;i++){**

**printf("x[%d]=",i);**

**scanf("%d",&x[i]);**

**}**

**for(int j=1;j<=n;j++){**

**for(i=0;i<n-j;i++){**

**if(x[i]>x[i+1]){**

**buf=x[i];**

**x[i]=x[i+1];**

**x[i+1]=buf;**

**}**

**}**

**}**

**for(i=0;i<n;i++)**

**printf("x[%d]=%d\n",i,x[i]);**

**}**

1. Сортировка массива по убыванию, пузырьковым методом. [Для изменения типа сортировки нужно изменить всего 1 символ.](https://www.google.by/search?espv=2&biw=1920&bih=965&q=%D0%94%D0%BB%D1%8F+%D0%B8%D0%B7%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F+%D1%82%D0%B8%D0%BF%D0%B0+%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8+%D0%BD%D1%83%D0%B6%D0%BD%D0%BE+%D0%B8%D0%B7%D0%BC%D0%B5%D0%BD%D0%B8%D1%82%D1%8C+%D0%B2%D1%81%D0%B5%D0%B3%D0%BE+1+%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB.&spell=1&sa=X&ved=0CBgQBSgAahUKEwiLl_2a2JTHAhWEfnIKHf1-DGo)
2. Сортировка массива по возрастанию, выбором наименьшего элемента.

Последовательно проходим весь массив, каждый раз сравнивая i-ый элемент со всеми, находящимися после него, и, найдя наименьший, переставляет его с i-им. Таким образом, после первого просмотра на первом месте оказывается наименьший элемент массива.

**#include<stdio.h>**

**void main()**

**{**

**const int n=6;**

**int x[n],i,buf,j,k;**

**for(i=0;i<n;i++){**

**printf("x[%d]=",i);**

**scanf("%d",&x[i]);**

**}**

**for( i=0;i<=n-1;i++){**

**k=i;**

**for(j=i+1;j<n;j++){**

**if(x[k]>x[j])**

**k=j;**

**}**

**buf=x[i];**

**x[i]=x[k];**

**x[k]=buf;**

**}**

**for(i=0;i<n;i++)**

**printf("x[%d]=%d\n",i,x[i]);**

**}**

1. Сортировка массива по убыванию, выбором наибольшего элемента
2. Сортировка массива по методу Шелла, по возрастанию.

На начальном этапе сравниваются элементы отстоящих на n/2 друг от друга и если есть необходимость, происходит перемещение этих элементов. Причем сравнение элементов на расстоянии n/2 происходит до тех пор, пока не было ни одного обмена. Далее сравниваются элементы, отстоящие на n/4 и т.д. пока не будут сравниваться соседние элементы.

Данная сортировка считается высоко производительной.

**#include<stdio.h>**

**void main()**

**{**

**const int n=6;**

**int x[n],i,buf,j,k,k1;**

**for(i=0;i<n;i++){**

**printf("x[%d]=",i);**

**scanf("%d",&x[i]);**

**}**

**for( k=n/2;k>0;k/=2){**

**do{**

**k1=0;**

**for(i=0,j=k;j<n;i++,j++){**

**if(x[i]>x[j]){**

**buf=x[i];**

**x[i]=x[j];**

**x[j]=buf;**

**k1++;**

**}**

**}**

**}while(k1);**

**}**

**for(i=0;i<n;i++)**

**printf("x[%d]=%d\n",i,x[i]);**

**}**

1. Сортировка массива по методу Шелла, по убыванию.
2. Введите два индекса, и поменять местами элементы.
3. Ввести массив из 20 элементов и отсортировать от 0 до 9 пузырьковым методом по возрастанию, и от 10 до 19 нахождением минимального элемента.
4. Введите два массива целых чисел. Второй массив упорядочен по возрастанию своих значений. Определить, какие числа первого массива входят и во второй массив.

**#include<stdio.h>**

**#define MAXSIZE 100**

**void main(){**

**int i,sz1,sz2,k,k1,j;**

**int arr1[MAXSIZE],arr2[MAXSIZE],buf;**

**do{**

**printf("\nВведите размеры массивов: ");**

**scanf("%d%d",&sz1,&sz2);**

**}while((sz1>MAXSIZE)||(sz2>MAXSIZE));**

**printf("\nВвод I-го массива:\n");**

**for(i=0;i<sz1;i++)**

**scanf("%d",&arr1[i]);**

**printf("\nВвод II-го массива:\n");**

**for(i=0;i<sz2;i++)**

**scanf("%d",&arr2[i]);**

**//Сортировка II-го массива**

**for(i=0;i<sz2-1;i++){**

**for(k=i,j=i+1;j<sz2;j++)**

**if(arr2[k]>arr2[j])**

**k=j;**

**buf=arr2[i];**

**arr2[i]=arr2[k];**

**arr2[k]=buf;**

**}**

**printf("Числа входящие в оба массива:\n");**

**for(i=0;i<sz1;i++){**

**if(arr1[i]==arr2[sz2/2]){**

**printf("%d ",arr1[i]);**

**continue;**

**}**

**if(arr1[i]>arr2[sz2/2]){**

**k=sz2/2+1;**

**k1=sz2;**

**}**

**else{**

**k=0;**

**k1=sz2/2;**

**}**

**for(j=k;j<k1;j++)**

**if(arr1[i]==arr2[j]){**

**printf("%d ",arr1[i]);**

**continue;**

**}}}**

1. Введите два массива вещественных чисел. Второй массив упорядочен в порядке убывания. Определить, какие числа первого массива входят и во второй массив.
2. Даны координаты n точек. x1,y1,x2,y2,…….xn,yn. Найти координаты точек, которые находятся на максимальном расстоянии друг от друга.

**#include<stdio.h>**

**void main()**

**{**

**const int n=4;**

**int x[2\*n]={-8,-5,**

**5,10,**

**10,8,**

**8,-2};**

**int max\_ab,max1,max,i\_a,i\_b,i1,i2,i,j;**

**max\_ab=max=0;**

**i\_a=0;**

**i\_b=0;**

**for(i=0;i<2\*n-2;i+=2)**

**{**

**i1=i;**

**for(j=i+2,i2=j;j<2\*n;j+=2)**

**{**

**max1=(x[i]-x[j])\*(x[i]-x[j])+(x[i+1]-x[j+1])\*(x[i+1]-x[j+1]);**

**if(max<max1)**

**{**

**max=max1;**

**i1=i;i2=j;**

**}**

**}**

**if(max>max\_ab)**

**{**

**max\_ab=max;**

**i\_a=i1;**

**i\_b=i2;**

**}**

**}**

**printf("max (%d,%d) i (%d,%d)\n",x[i\_a],x[i\_a+1],x[i\_b],x[i\_b+1]);**

**}**

# 3.4 Двумерные массивы

a00 a01a02 a03 матрица А размерности 3х4

a10 a11 a12 a13

a20a21a22 a23

Для того, чтобы ввести и вывести двумерный массив по строкам необходимо

**const int n=3,m=4;**

**int x[n][m],i,j;**

**for(i=0;i<n;i++)**

**{**

**for(j=0;j<m;j++)**

**{**

**printf("x[%d][%d]=",i,j);**

**scanf("%d",&x[i][j]);**

**}**

**}**

**for(i=0;i<n;i++)**

**{**

**for(j=0;j<m;j++)**

**printf("x[%d][%d]=%d ",i,j,x[i][j]);**

**printf(“\n”);**

**}**

Для того, чтобы ввести и вывести двумерный массив по столбцам, а вывести по строкам необходимо

**const int n=3,m=4;**

**int x[n][m],i,j;**

**for(j=0;j<m;j++)**

**{**

**for(i=0;i<n;i++)**

**{**

**printf("x[%d][%d]=",i,j);**

**scanf("%d",&x[i][j]);**

**}**

**}**

**for(i=0;i<n;i++)**

**{**

**for(j=0;j<m;j++){**

**printf("%d\t",x[i][j]);  
}**

**printf(“\n”);**

**}**

**Задачи:**

1. Посчитать сумму всех элементов 2-го массива
2. Найти max-ый и min-ый элементы массива и их места.
3. Сумму элементов выше диагонали, место мах выше диагонали и место min ниже диагонали.
4. Сумма столбцов
5. Сумма строк

**const int n=3,m=4;**

**int x[n][m],i,j,y[n];**

**for(i=0;i<n;i++){**

**for(j=0;j<m;j++){**

**printf("x[%d][%d]=",i,j);**

**scanf("%d",&x[i][j]);**

**}**

**}**

**for(i=0;i<n;i++){**

**y[i]=0;**

**for(j=0;j<m;j++)**

**y[i]=y[i]+x[i][j];**

**}**

1. Сортировка строк

**const int n=3,m=4;**

**int x[n][m],i,j,y[n],buf,buf1;**

**for(i=0;i<n;i++)**

**{**

**for(j=0;j<m;j++)**

**{**

**printf("x[%d][%d]=",i,j);**

**scanf("%d",&x[i][j]);**

**}**

**}**

**for(i=0;i<n;i++){**

**y[i]=0;**

**for(j=0;j<m;j++)**

**y[i]=y[i]+x[i][j];**

**}**

**for(int t=0;t<n;t++){**

**for( i=0;i<n-1;i++){**

**if(y[i]>y[i+1]){**

**buf1=y[i];**

**y[i]=y[i+1];**

**y[i+1]=buf1;**

**for(j=0;j<m;j++){**

**buf=x[i][j];**

**x[i][j]=x[i+1][j];**

**x[i+1][j]=buf;**

**}**

**}**

**}**

**}**

**puts("");**

**for(i=0;i<n;i++){**

**for(j=0;j<m;j++)**

**printf("x[%d][%d]=%d\n",i,j,x[i][j]);**

**}**

1. Сортировка столбцов
2. Рассортировка всех элементов матрицы

**#include<stdio.h>**

**void main()**

**{**

**const int n=4,m=3;**

**int mat[n][m];**

**int i,j,temp;**

**for(i=0;i<n;i++)**

**{**

**printf("\nВвод %d-ой строки: ",i+1);**

**for(j=0;j<m;j++)**

**scanf("%d",&mat[i][j]);**

**}**

**for(i=0;i<n;i++){**

**printf("\n\n");**

**for(j=0;j<m;j++)**

**printf("%d ",mat[i][j]);**

**}**

**for(i=0;i<n\*m;i++){**

**for(j=0;j<m\*n;j++){**

**if(mat[i/m][i%m]<mat[j/m][j%m]){**

**temp=mat[i/m][i%m];**

**mat[i/m][i%m]=mat[j/m][j%m];**

**mat[j/m][j%m]=temp;**

**}**

**}**

**}**

**for(i=0;i<n;i++){**

**printf("\n\n");**

**for(j=0;j<m;j++)**

**printf("%d ",mat[i][j]);**

**}**

**}**

1. Перемножение матриц.

**#include<stdio.h>**

**void main()**

**{**

**const int n=3,m=3,m1=3;**

**double a[n][m],b[m][m1],c[n][m1];**

**int i,j,k;**

**printf("I massiv:\n");**

**for(i=0;i<n;i++){**

**for(j=0;j<m;j++){**

**printf("a[%d][%d]=",i,j);**

**fflush(stdin);**

**scanf("%lf",&a[i][j]);**

**}**

**}**

**printf("\n II massiv:\n");**

**for(i=0;i<m;i++){**

**for(j=0;j<m1;j++){**

**printf("b[%d][%d]=",i,j);**

**fflush(stdin);**

**scanf("%lf",&b[i][j]);**

**}**

**}**

**i=0;**

**while(i<n){**

**j=0;**

**while(j<m1){**

**c[i][j]=0;**

**k=0;**

**while(k<m){**

**c[i][j]+=a[i][k]\*b[k][j];**

**k++;**

**}**

**j++;**

**}**

**i++;**

**}**

**for(i=0;i<n;i++){**

**for(j=0;j<m1;j++){**

**printf("c[%d][%d]=%.2lf\n",i,j,c[i][j]);**

**} }**

**}**

§4 МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ. ФУНКЦИИ

Принцип программирования на языке “C”(“C++”) основан на понятии функции. Любая "C"-программа, каков бы ни был ее размер, состоит из одной или более "функций", указывающих фактические операции компьютера, которые должны быть выполнены. Обычно вы можете давать функциям любые имена по вашему усмотрению, но **main()** - это особое имя; выполнение вашей программы начинается сначала с функции **main().** Это означает, что каждая программа должна в каком-то месте содержать функцию с именем **main()**. Для выполнения определенных действий функция **main()** обычно обращается к другим функциям, часть из которых находится в той же самой программе, а часть - в библиотеках, содержащих ранее написанные функции.

main()

function1();

function2();

printf(“str”);

function3();

function4();

Одновременно с возникновением языков высокого уровня (Aлгол, Basic, Фортран, PL/1, Pascal, C и др.) широкое распространение получил метод модульного программирования. Согласно ему, проект (задача, программа) разбивается на логически завершённые части, которые оформляются по определённым правилам. В некоторых других системах их называли подпрограммами. Во многих языках использовались и остались два их вида, например, процедуры и функции на языке Pascal, которые отличаются правилами их оформления и вызова.

С точки зрения использования рассматриваемой здесь технологии можно выделить следующие уровни программирования:

1. без использования подпрограмм, кроме, быть может, встроенных (стандартных), как было показано в первых трех главах;
2. с подпрограммами; Это будет рассмотрено в этой главе;
3. подпрограммы можно объединять в библиотеки (язык Фортран), модули (Pascal) и т. п.;
4. с возникновением объектно-ориентированного программирования подпрограммы включаются в классы и называются их методами.

С одной стороны, на языке С/С++ формально нет деления подпрограмм на два вида, как это сделано, например, в Pascal. Здесь можно составлять и использовать только функции, которые могут быть как самостоятельными, не включёнными в класс, так и его методами. Но по аналогии с другими языками для эффективного их изучения функции можно также разделить на два вида. Функция типа void аналогична процедуре Pascal (procedure), а функция с возвращаемым с помощью return единственным значением частично похожа на function языка Pascal, но проще.

Одним способом обмена данными между функциями является передача посредством аргументов. Круглые скобки, следующие за именем функции, заключают в себе список аргументов; здесь **main()** - функция без аргументов, что указывается как (). Операторы, составляющие функцию, заключаются в фигурные скобки { и }. Обращение к функции осуществляется указанием ее имени, за которым следует заключенный в круглые скобки список аргументов. Круглые скобки должны присутствовать и в том случае, когда функция не имеет аргументов.

Строка

**printf**("HELLO, WORLD\n");

является обращением к функции, которое вызывает функцию с именем **printf()** и аргументом "HELLO, WORLD\n". Функция **printf()** является библиотечной функцией, которая выдает данные на экран результатов. В данном случае печатается строка символов, являющаяся аргументом функции.

Любая переменная имеет область действия (т.е. это та часть в программе, где данная переменная доступна) и время действия (т.е. в течении какого времени переменная хранит свое значение). Переменные в языках “С/C++” могут быть **локальными** или **глобальными**. Для локальной переменной область действия и время действия в той функции или в том блоке где данная переменная определена. Глобальные переменные объявляются вне функции и для глобальной переменной область действия и время действия в течение всей программы, кроме того блока или функции, где данная переменная переопределена.

# 4.1 Примеры. Правила оформления и вызова функций

**тип\_возвращаемого\_значения имя\_функции (список параметров)**

{

объявления и операторы, т.е. тело функции

}

**тип\_возвращаемого\_значения** - это тип данных результата, возвращаемого из функции. Тип возвращаемого значения может быть любой тип\_данных или **void.** Если возвращаемое значение **void**, то это означает, что функция не возвращает никакого значения, или на прямую не возвращает никакого значения.

**имя\_функции** – это любой правильно написанный идентификатор.

**список параметров** – это список разделенных запятыми объявлений тех параметров, которая получает функция при вызове. Если функция не получает никаких значений, список параметров задается как **void**, т.е. список параметров пустой.

Функция используется следующим образом:

1. прототип функции.
2. вызов функции.
3. описание самой функции (реализация)

**Прототип функции:** является одной из наиболее важных особенностей языка, т.к. прототип функции указывает компилятору тип данных, возвращаемый функцией, количество параметров, которое ожидает функция, тип параметров и ожидаемый порядок их следования.

**Оператор return .** Оператор *return* завершает выполнения функции, в которой он встретился, и передает управление в вызывающую программу в точку вызова. Оператор return имеет следующий вид: **return выражение;** Выражение может быть как константой и вычисляемым выражением. Оператор return в теле функции может встречаться несколько раз.

**Примеры:**

***Пример 1***. Пусть необходимо дважды вывести горизонтальную линию, состоящую из 40 символов псевдографики “−” с кодом 196.

// 1-й вариант (без функции)

int main()

{ printf("Testing of function: \n the first line\n");

for (int i=1; i<=40; i++) printf("-");

printf("\n");

printf(" The second line\n");

for (int i=1; i<=40; i++) printf("-");

printf("\n");

getch();

return 0;

}

Выделенная часть, как видно, повторяется дважды. Оформим её в виде функции, которая только выводит и никаких результатов, в смысле значений переменных, не получает. В виде функции можно оформить любую, не обязательно повторяющуюся, логически завершённую часть программы.

// 2-й вариант (с функцией)

#include<stdio.h>

#include<conio.h>

void LINE1();// Прототип (объявление) функции.

int main()

{ printf("Testing of function: \n the first line\n");

LINE1(); // Вызов функции.

printf(" The second line\n");

LINE1(); // Вызов функции.

getch();

return 0;

}

void LINE1() //реализация функции

{

for (int i=1; i<=40; i++)

printf("-");

printf("\n");

}

Перед функцией main() записываем прототип (объявление, заголовок) функции. Ключевое слово void в данном примере означает, что функция не возвращает результаты. В следующем параграфе будет показано, что функцию типа void можно использовать и для других целей. Далее следует имя функции по обычным правилам записи идентификаторов. Пустые круглые скобки означают, что функция не имеет никаких параметров, ни входных, ни выходных (результатов). Слово void при этом в круглых скобках (но не в начале!) можно не писать, как это было в “старом” языке С. Заметим, что после прототипа обязательно записывается символ “;” (точка с запятой), а при описании функции после main этот символ не пишем.

Описание функции размещаем после текста функции main(). Повторяем заголовок, но без “;” в конце, и затем в фигурных скобках записываем её текст. При необходимости внутри функции объявляются так называемые локальные переменные (переменная i для счётчика в нашем примере). Они не являются ни входными, ни выходными переменными для функции, а предназначены для хранения промежуточных величин. Такие переменные можно использовать только внутри данной функции. Здесь, как и раньше, переменную i можно было объявить в заголовке оператора for.

***Пример 2.*** Усложним задачу. Пусть надо первый раз вывести, например, 20 символов “\*”, а второй — линию из 40 символов псевдографики “–”. Поэтому функция будет иметь два входных параметра: Len для длины линии, в которой будем выводить линию, и ch для выводимого символа.

#include<stdio.h>

#include<conio.h>

void LINE2(int LEN,char ch);// Прототип (объявление) функции.

int main()

{ printf("Testing of function: \n the first line\n");

LINE2(20,'\*'); // Вызов функции.

printf(" The second line\n");

LINE2(40,'-'); // Вызов функции.

getch();

return 0;

}

void LINE2(int LEN,char ch)//реализация функции.

{

for (int i=1; i<=LEN; i++)

printf("%c",ch);

printf("\n");

}

В прототипе функции и при её описании в заголовке обязательно надо записывать типы параметров, даже если они повторяются. Имена параметров обязательны только в заголовке описания, а в прототипе можно указать только их типы: void LINE2(int , char).

Вызов функции типа void выполняется так, как вызов процедуры (Procedure) на языке Pascal, без использования дополнительного оператора, как это имеет место в некоторых других системах. Достаточно записать имя функции и в скобках, если есть, входные и выходные (в нашем примере их нет) фактические параметры. Если параметры отсутствуют, то и в прототипе, и в описании, и при вызове функции необходимо обязательно записать пустые круглые скобки.

Выполнение программы с функцией всегда начинается с main. Когда встретится вызов функции (а не её описание), выполняется следующая последовательность действий:

1. запоминается точка возврата, то есть адрес следующей команды;
2. значение каждого фактического параметра значения (но не параметра-ссылки, не указателя и не элементы массива) копируется на место формального. Заметим, что значения формальных входных параметров в функции определять вводом или другим способом не надо, хотя это синтаксически и не запрещается;
3. управление передаётся на первый выполняемый оператор функции;
4. выполняется функция;
5. если выполнился последний оператор функции или встретился return, управление передаётся в ту точку, откуда функция была вызвана. Заметим, что передача управления из точки вызова функции и обратно выполняется автоматически, без дополнительных операторов.

Если встретился повторный вызов этой же функции, описанные выше действия повторяются.

# 4.2 Примеры. Функции, отличные от void. Оператор return

***Пример 3.*** Ввести последовательность целых чисел (не массив), пока не введём 0 (нуль). Для каждого из них найти и вывести сумму составляющих это число десятичных цифр.

При решении задач с использованием любых типов функций необходимо ответить на следующие вопросы:

1. Какие функции, для каких целей будем составлять? Может быть несколько вариантов ответа на такой вопрос. Для определённости составим функцию для вычисления суммы десятичных цифр одного числа с именем SUMDIG. Эта функция будет вызываться в цикле несколько раз для каждого введённого числа.
2. Определяем входные и выходные параметры функции, то есть что для неё дано, и что функция получает. У нас дано одно (!) целое число, а не массив, и получаем сумму его цифр, то есть тоже одно (!) целое число.
3. Определив входные и выходные параметры для каждой функции, записываем их прототипы перед main. Так как функция имеет единственный результат, её лучше оформить как функцию, отличную от void. Тогда прототип записывается так: тип имя\_функции(входные\_параметры);, где тип —тип единственного возвращаемого значения. В конце прототипа обязательна “;”(точка с запятой).
4. Составляем алгоритм функции SUMDIG и записываем его на языке C после main. Для вычисления суммы цифр выделяем их в обратном порядке справа налево, используя получение остатка от деления на 10 (операция %) и целочисленное деление на 10 (операция /). Так как размерность числа может быть любой, то получается цикл с неизвестным количеством повторений. Получение цифр продолжаем, пока не получим при целочисленном делении нуль.
5. Рекомендуется сначала проверить нашу функцию, вызвав её один раз без цикла, и отладить её (см. первый вариант main).
6. Убедившись в правильности работы функции, решаем нашу задачу. В головной функции в цикле вводим целое число, вызываем SUMDIG и выводим результат. Если введём 0, цикл и программа в целом прекращает работу.

Из двух вариантов main в проект включаем один.

#include<stdio.h>

#include<conio.h>

int SUMDIG(int );

//Первый вариант main для проверки функции

/\*int main() { printf(SUMDIG(220352); getch(); return 0; }\*/

/\*Второй окончательный вариант main. Когда функция отлажена, первый удаляем, или “закрываем” комментариями.\*/

int main() {

int X=1;

printf( "For exit -- 0\n\n");

while (X){

scanf("%d",&X);

printf("sum=%d\n",SUMDIG(X));

}

return 0; }

int SUMDIG (int num) { int s=0;

while (num) { s+= num%10; num/=10; }

return s; }

В прототипе функции, возвращающей единственный результат, записываем его тип вместо void, имя функции и в скобках только входные параметры с их типами по тем же правилам, что и раньше. В скобках единственный выходной параметр не записывается. Аналогично записываем заголовок функции при её описании.

В тексте такой функции должен быть оператор вида return выражение; где выражение должно быть того же или совместимого типа, что и тип функции в прототипе и заголовке функции перед её именем. Оператор осуществляет выход из функции, то есть прекращает её работу, передаёт управление в точку вызова (в нашем примере в головную функцию, но не обязательно) и возвращает в неё значение указанного выражения.

**Особенности return***:*

* как частный случай выражения, можно возвращать константу или значение одной переменной;
* оператор может быть единственным в функции:

float MyFun1 (float x, float y) { return x+y; }

* наоборот, таких операторов может быть несколько и он не обязательно должен быть в конце функции. Но выполняется всегда один из них. Например, следующий вариант

float MyFun2A (float x, float y)

{ if (x>y) return x+y; else return x\*y; }

короче, чем

float MyFun2B (float x, float y)

{ float r; if (x>y) r= x+y; else r= x\*y; return r; }

Это же можно реализовать ещё компактнее и оригинальнее:

float MyFun2 (float x, float y)

{ return x>y ? x+y : x\*y; } // Тернарная операция

* оператор return обязателен, если возвращается одно значение, то есть в заголовке тип результата отличен от void.

Вызов такой функции с единственным возвращаемым значением записывается непосредственно в выражении, где используется её результат. Кроме “команды вывода”, как было показано в примере, обращение к функции с одним результатом можно записать

* в операции присваивания, например sd=SUMDIG(1234);
* в операторе if, не присваивая предварительно значение функции переменной: if (SUMDIG(a)>10)…, где a — переменная целого типа;
* в операторе while, например, while (SUMDIG(a\*a)!=num)…или   
  while (SUMDIG(a\*a)-num)…
* в операторе switch: switch(SUMDIG(a)) {……} Но вызов функции нельзя записать после ключевого слова case в этом операторе, так как полученное с помощью функции значение не является константой;
* при вызове другой функции в качестве фактического параметра например: LINE2 (SUMDIG(a), ‘.’);

Необязательный оператор return без возвращаемого значения можно записать и в функции типа void. В таком случае он прекращает работу функции, передаёт управление в точку вызова, не возвращая никакого значения. Оператор return имеет смысл использовать, если надо выйти из функции в разных её ветвях в зависимости от некоторого условия. Если он не записан, то работа функции типа void заканчивается после выполнения последнего оператора.

Заметим, что во всех предыдущих примерах был рассмотрен самый простой вариант с одной функцией пользователя, которая вызывалась из main. В более сложных задачах может быть несколько функций и не все они должны вызываться из головной функции. Одна из них может вызывать другую. Такие возможности будут рассмотрены позже.

# 4.3 Примеры. Область действия переменных. Локальные и глобальные переменные

И Существенной особенностью функций языка С++, упрощающей их изучение, является то, что функции не могут быть вложенными. Другими словами, невозможно определить (описать) одну функцию внутри другой. В таком случае говорят, что все функции находятся на одном уровне видимости. Поэтому вопрос области действия имён решается проще, чем на языке Pascal.

Есть несколько мест объявления локальных переменных.

В скобках в прототипе и в заголовке при описании функции перечисляются только те переменные, которые должны быть переданы в функцию, то есть входные, и её результаты (выходные параметры), если такие есть. Такие переменные, названные формальными параметрами, доступны (видны, их можно использовать) только в той функции, в заголовке которой они описаны. Повторно в теле функции формальные параметры объявлять не надо! В прототипе функции имена параметров можно не писать, достаточно указать только их типы. Эта информация используется компилятором при проверке соответствия типов и количества фактических и формальных параметров. При описании функции в её заголовке имена параметров и повторно их типы записываются обязательно; Как видно из предыдущих примеров, здесь объявляются не все используемые в функции переменные.

Переменные, содержащие некоторые промежуточные значения, объявляются в любом месте тела функции до первого их использования, доступны только в ней и являются также локальными. Они не являются формальными параметрами, в заголовке функции не записываются, при вызове функции не передаются. В теле функции, отличной от void, можно объявить также переменную для результата, возвращаемого с помощью return переменная.

Как частный случай последней возможности переменные можно объявить внутри блока. Блок или составной оператор — это последовательность операторов, заключённая в фигурные скобки. Таким блоком может быть, например, тело любого из циклов, одна из ветвей if или switch. Например,

if (…) { float t; …}. Такую переменную (t) можно использовать только в этом блоке и во всех внутренних (вложенных) по отношению к нему. Вне данного блока она недоступна. Этим мы гарантируем, что вне блока такая переменная не будет изменена. Такая локальная переменная часто используется в заголовке for.

…… //Здесь i недоступна

for (int i=0; i<n;i++) // i видна в заголовке

{ …… // i видна здесь

}

…… // Здесь i использовать нельзя

При таком объявлении переменную (в примере i) можно использовать только внутри цикла и в любом из трёх выражений заголовка цикла.

Объявив её вне заголовка цикла, получим:

……// Здесь i использовать нельзя

int i;

… ... //Здесь i видна

for (i=0; i<n;i++) // i видна и в заголовке цикла

{ …… /\* i видна и в цикле \*/ }

… … // i доступна и после цикла здесь

Тогда она видна не только в цикле, а и в любом месте после объявления, как перед for, так и после фигурных скобок для цикла.

**Локальные** переменные имеют следующие особенности:

* их можно объявить в любом месте функции или блока до первого их использования, то есть не обязательно в самом начале. Это, конечно, не относится к формальным параметрам, которые объявляются всегда в скобках заголовка;
* локальные переменные создаются при входе в функцию или блок и уничтожаются при выходе из них. При этом не просто теряются значения, но и освобождается память;
* локальные переменные в разных функциях или блоках, не вложенных друг в друга, могут иметь одинаковые идентификаторы. Но на самом деле это разные переменные, для них отводится разная память.

Переменные, объявленные в main, также являются локальными и их можно использовать только в основной функции.

**Глобальные** переменные можно объявить вне всех функций, в том числе вне main, например, в самом начале перед прототипами функций. Кроме этого их разрешается объявлять также между прототипами, непосредственно перед функцией main или между текстами функций. Особенности таких переменных:

* “видны” из любой функции, размещенной после описания переменной;
* память занята на всё время выполнения проекта;

как правило, использование локальных переменных **предпочтительнее** глобальных.

Примеры:

1. используя глобальную переменную, программа считает квадрат вводимого числа с использованием функции.

**#include<stdio.h>**

**void kvadrat();// прототип функции**

**int x;// глобальная переменная, следовательно она понимается как в функции main(), //так и в функции kvadrat()**

**void main(){**

**printf("x=");**

**scanf("%d",&x);**

**kvadrat();//вызов функции**

**}**

**void kvadrat(){// описание функции**

**printf("%d^2=%d\n",x,x\*x);**

**}**

1. Неправильный пример, с использованием локальной переменной.

**#include<stdio.h>**

**void kvadrat();**

**void main(){**

**int x;// локальная переменная, для функции main(), поэтому она не доступна в функции kvadrat()**

**printf("x=");**

**scanf("%d",&x);**

**kvadrat();**

**}**

**void kvadrat(){**

**printf("%d^2=%d\n",x,x\*x);**

**}**

1. Правильный пример, с использованием локальной переменной

**#include<stdio.h>**

**void kvadrat(int);//функция с параметром**

**void main(){**

**int x;// локальная переменная, для функции main(), поэтому она не доступна в функции kvadrat()**

**printf("x=");**

**scanf("%d",&x);**

**kvadrat(x);**

**}**

**void kvadrat(int a){**

**printf("%d^2=%d\n",a,a\*a);**

**}**

При вызове функции мы передаем ей переменную, с которой она должна работать, т.е. создается локальная переменная а, которая получает то же значение, что и х. После вызова функции kvadrat() все содержания локальных переменных уничтожаются автоматически.

1. Пример с использованием возвращаемого значения и параметром

**#include<stdio.h>**

**int kvadrat(int);//функция с параметром и возвращаемым значением**

**void main(){**

**int x,kv;// локальная переменная, для функции main(), поэтому она не доступна в функции kvadrat()**

**printf("x=");**

**scanf("%d",&x);**

**kv=kvadrat(x);**

**printf(“%d^2=%d\n”,x,kv);**

**}**

**int kvadrat(int a){**

**int y;**

**y= a\*a;**

**return y;// return a\*a;**

**}**

# 4.4 Примеры. Упражнения.

1. Ввести 3 числа x1,x2,x3а и определить максимальное из x1,x2,x3. Использовать функцию, которая получает 3 числа и возвращает максимальное из них

#include<stdio.h>

#include<conio.h>

float maximum( float a, float b, float c );//Прототип функции

int main()

{

float val1, val2, val3; //Декларация переменных

printf( "Enter three floats:\n" );//Вывод текста на экран

scanf( "%f%f%f", &val1, &val2, &val3 ); //Чтение трех дробных чисел

printf("Maximum is: %f",maximum(val1,val2,val3));//Вызов функции max и вывод

//на экран самого большого числа

getch();

return 0;

}

float maximum( float a, float b, float c ) //Заголовок функции

{

float max = a;//Переменная для самого большого числа – изначально равна a

if( b > max ) //Если b больше самого большого числа

max = b; //Помещаем значение b в самое большое число

if( c > max ) //Если с больше самого большого числа

max = c; //Помещаем значение с в самое большое число

return max; //Возврат самого большого числа

}

1. Ввести 5 чисел x1,x2,x3,x4,x5 и определить минимальное из x1,x2,x3; из x4,x2,x3; из x4,x5,x3. Использовать функцию, которая получает 3 числа и возвращает минимальное из них
2. Ввести 5 чисел x1,x2,x3,x4,x5 и определить минимальное и максимальное из x1,x2,x3; из x4,x2,x3; из x4,x5,x3. Использовать 2 функции, 1-ая получает 3 числа и возвращает минимальное из них, 2-ая получает 3 числа и возвращает максимальное из них
3. Написать функцию, которая считает факториал числа. Ввести 3 числа, посчитать их факториалы.
4. Решение квадратного уравнения, используя 3 функции: 1-ая печатает уравнение, 2-ая считает дискриминант, 3-я выводит корни уравнения.
5. Функция, которая получает степень n и основание x и возвращает xn.
6. Функция получает 2 целых числа и возвращает деление этих чисел.
7. Функция, которая получает число и определяет, является ли данное число совершенным или нет. Используя эту функцию в программе, которая определяет и печатает все совершенные числа от 1 до 1000. Напечатайте все сомножители каждого совершенного числа, чтобы убедится, чтобы убедиться, что число действительно совершенно.

# 4.5Рекурсия.

Фактически, рекурсия – это использование функции самой в себе. Практически, рекурсия – это метод решения задач, построенный на том, что рекурсивная функция может решить напрямую только частную задачу. Все остальные задачи функция делит на 2 части: часть, которую можно решить напрямую и часть, которую напрямую решить нельзя. Притом вторая часть должна являться такой задачей, которую можно было бы решить вызовом этой же функции.



В понимании рекурсия намного сложнее обычных вызовов функции. Рассмотрим ее на примере вычисления факториала числа.

Предположим, что нам надо найти значение 5!. Математически выводим, что:

5! = 5 \* 4 \* 3 \* 2 \* 1

5! = 5 \* ( 4 \* 3 \* 2 \* 1 )

5! = 5 \* 4!

И т.д.

То есть функция должна постепенно “упрощать” поставленную задачу. В конце концов она дойдет до значения 1! = 1, после чего произойдет обратная подстановка.

Результат рекурсии

5 \* 4!

4 \* 3!

3 \* 2!

2 \* 1!

1

2\*1=2

3\*2=6

4\*6=24

4\*24=120

120

5!

5

4!

4

3!

3

2!

2

1!

1

#include<stdio.h>

#include<conio.h>

unsigned long long int fact( unsigned a ); //Прототип функции факториала

int main()

{

for( int i = 0; i <= 20; i++ ) //Проходим от 0 до 20

printf( "%d! = %llu\n", i, fact( i ) ); //Выводим значение факториала i!

getch();

return 0;

}

unsigned long long int fact( unsigned a ) //Заголовок функции

{

if( a > 1 ) //Если у нас не частный случай -

return a \* fact( a - 1 ); //Разбиваем на 2 части : a \* (a–1)!

else //Иначе – если у нас частный случай

return 1; //Возвращаем 1 ( т.к. 0!=1 и 1!=1)

}

Обратите внимание что значения факториала растут очень быстро и числа выходят большие чем можно поместить в обычный тип int. В данной программе используется беззнаковый длинный целый тип данных.

Еще один интересный пример рекурсии – числа Фибоначи.

***Нахождение чисел Фибоначчи.***

*Числа Фибоначчи – это числа вида : 0, 1, 1, 2, 3, 5, 8, 13, 21… Т.е. два первых числа последовательности – 0 и 1, а каждое последующее число получается как сумма двух предыдущих.*

*Скажем, нам надо найти 8-мое число из ряда Фибоначчи. Итак:*

*F(8) = F(7) + F(6) = (F(6)+F(5))+(F(5)+F(4)) = ((F(5)+F(4)) + (F(4)+F(3))) + ((F(4)+F(3)) + (F(3)+F(2)))*

#include<stdio.h>

#include<conio.h>

unsigned int fibon( unsigned a ); //Прототип функции

int main()

{

for( int i = 1; i <= 20; i++ ) //Выводим числа Фибоначчи от 1 до 20

printf( "fibon(%2d) = %u\n", i, fibon( i ) );

getch();

return 0;

}

unsigned int fibon( unsigned a ) //Заголовок функции

{

if( a > 2 ) //Если a - не первый или второй элемент -

return fibon( a - 1 ) + fibon( a - 2 ); //Считаем его как сумму двух предыдущих

else //Иначе

return a-1; //Возвращаем значение. Для a=1 возвращаем 0, для a=2 возвращаем 1

}

**F(8)**

**F(7)**

**+**

**F(6)**

**F(6)**

**+**

**F(5)**

**F(5)**

**+**

**F(4)**

**F(3)**

**+**

**F(2)=1**

**F(2)=1**

**+**

**F(1)=0**

# *4.6* Классы памяти*.*

В языках программирования C/C++ имеются четыре спецификации класса памяти: **auto, register, extern и static.** Спецификация класса памяти переменной помогает определить его класс памяти и область действия. **Класс памяти переменной** определяет его время жизни или время существования этой переменной, связанная с понятием блока программы. Одни переменные живут не долго, другие неоднократно создаются и уничтожаются, а третьи существуют на протяжении всей программы. **Область действия** (область видимости) **переменной** – это область программы в которой на данную переменную можно сослаться, т.е. на некоторые переменные сослаться в любом месте программы, тогда как на другие только в определенных частях программы.

**Автоматическая переменная(auto)** – это всегда локальная переменная, и память для выделяется в оперативной памяти автоматически при входе в блок и освобождается тоже автоматически при выходе из блока. К переменным типа auto доступ из внешнего блока невозможен.

**Регистровая переменная(register)** – это автоматическая переменная, только ключевое слово register – лишь указывает компилятору, что эту переменную необходимо поместить в регистр, а не в оперативную память, что приводит к уменьшению затраты времени при решении задачи. Регистровой переменной могут быть переменные целого типа. Если вы объявите переменную как register, то нет уверенности в том, что она будет помещена в регистре, т.к. не может быть места в регистре. Некоторые новые компиляторы сами способны распознавать часто используемую переменную и помещать ее в регистр.

К классу **extern** относятся автоматически имена функций и глобальные переменные. Но есть различие между классом памяти и областью определения, к примеру, если есть блок, где объявлена локальная переменная под тем же именем, что и глобальная, то в этом блоке нет доступа к глобальной переменной.

**static**– локальные переменные объявленные ключевым словом static известны только в той функции, где они определены, но в отличии от локальных переменных объявленных, как auto переменные static сохраняют свое значение в течении всего времени выполнения программы, причем при каждом последующем вызове функции сохраняет предыдущее значение которое оно имела при предыдущем вызове. Переменные объявленные как static автоматически получают нулевые значения.

Рассмотрим следующий пример:

**#include<stdio.h>**

**void a();**

**void b();**

**void c();**

**int x=1;**

**void main(){**

**int x=5;**

**printf("Локальная переменная х при входе в функцию main() %d\n",x);**

**{**

**int x=7;**

**printf("Локальная переменная x в блоке{}, который находиться в функции main() %d\n",x);**

**}**

**printf("Локальная переменная х внутри функции main() %d\n",x);**

**a();**

**b();**

**c();**

**a();**

**b();**

**c();**

**printf("Локальная переменная х при выходе из функции main() %d\n",x);**

**}**

**void a(){**

**int x=25;**

**printf("Локальная переменная х при входе в функцию a() %d\n",x);**

**x++;**

**printf("Локальная переменная х при выходе из функции a() %d\n",x);**

**}**

**void b(){**

**static int x=50;**

**printf("Локальная переменная х при входе в функцию b() %d\n",x);**

**x++;**

**printf("b() %d\n",x);**

**}**

**void c(){**

**printf("Глобальная переменная х при входе в функцию c() %d\n",x);**

**x\*=10;**

**printf("Глобальная переменная х при выходе из функции c() %d\n",x);**

**}**

**Локальная переменная х при входе в функцию main() 5**

**Локальная переменная х в блоке{}, который находиться в функции main() 7**

**Локальная переменная х внутри функции main() 5**

**Локальная переменная х при входе в функцию а() 25**

**Локальная переменная х при выходе из функции a() 26**

**Локальная переменная х при входе в функцию b() 50**

**Локальная переменная х при выходе из функции b() 51**

**Глобальная переменная х при входе в функцию c() 1**

**Глобальная переменная х при выходе из функции с() 10**

**Локальная переменная х при входе в функцию а() 25**

**Локальная переменная х при выходе из функции a() 26**

**Локальная переменная х при входе в функцию b() 51**

**Локальная переменная х при выходе из функции b() 52**

**Глобальная переменная х при входе в функцию c() 10**

**Глобальная переменная х при выходе из функции с() 100**

**Локальная переменная х при выходе в функции main() 5**

# 4.7 Указатели

*Указатель* – это особый вид переменной, который хранит адрес другой переменной.

*Определение* указателя:

**тип\_данных \*имя\_указателя;**

тип\_данных – это тип на который будет указывать указатель;

\* - определяет тип указатель;

имя\_указателя – это имя переменной, которая хранит адрес переменной, на которую будет указывать указатель;

\*имя\_указателя – это содержание адреса, на который указывает указатель.

Указатель можно представить в виде стрелки. При объявлении указателя можно задавать начальное значение, которое может быть либо адрес переменной того же типа или значение нуль, который записывается двумя способами либо просто число 0, либо **NULL.**

**Что распечатает данная программа?**

**#include<stdio.h>**

**void main()**

**{**

**int x=67,\*p;//х - это переменная целого типа, а p указатель на целый тип**

**p=&x;// присвоение адреса переменной х переменной p**

**printf("\n%d\n",\*p);**

**\*p=\*p+3;**

**printf("%d\n",x);**

**}**

Операция & является адрес переменной в памяти и может использоваться практически со всеми типами данных, кроме констант и битовых полей. Значение указателя можно вывести на экран с помощью спецификации %p

(pointer), результат выводится в шестнадцатеричном виде.

Рассмотрим фрагмент программы:

int a=5, \*p, \*p1, \*p2;

p=&a; p2=p1=p;

++p1; p2+=2;

printf(“a=%d, p=%d, p=%p, p1=%p, p2=%p.\n”, a, p, p, p1, p2);

Результат выполнения: a=5, \*p=5, p=FFC8, p1=FFCC, p2=FFD0.

Указатель часто используется для обмена данных с функциями. В то время, как в функцию можно передавать столько аргументов сколько требуется, а с помощью оператора **return** возвращается только одно значение. Когда возникает необходимость вернуть более, чем одного значения с помощью одной функции используются указатели.

**Пример:**

Функция, которая получает 2 целых числа и возвращает сумму, произведение, разность и деление данных чисел.

**#include<stdio.h>**

**void fff(int,int,int\*,int\*,int\*,double\*);**

**void main()**

**{**

**int x,y,sum,pr,raz;**

**double del;**

**printf("Введите 2 целых числа: ");**

**scanf("%d%d",&x,&y);**

**fff(x,y,&sum,&pr,&raz,&del);**

**printf("%d+%d=%d\n",x,y,sum);**

**printf("%d-%d=%d\n",x,y,raz);**

**printf("%d\*%d=%d\n",x,y,pr);**

**printf("%d/%d=%.2lf\n",x,y,del);**

**}**

**void fff(int a,int b,int\*p1,int\*p2,int\*p3,double\*p4)**

**{**

**\*p1=a+b;**

**\*p2=a\*b;**

**\*p3=a-b;**

**\*p4=(double)a/b;**

**}**

**Задачи:**

1. В **main()** объявляем 2 переменный, потом вызываем функцию, где вызываем функцию **scanf()** и записывает в эти переменные числа, которые распечатываются в **main()**.
2. Функция которая получает только 3 значения из **main()** и возвращает в первую переменную выражение a+b+c –(a\*a); во вторую a\*a +b\*b-c\*c, а в третью a\*b\*c, т.е если начальные значения a=3, b=2,c=1, то теперь в a храниться -3, b храниться 14, а в c 6.
3. Функция получает два числа и меняет местами значения переменны.

**Задачи:**

**1.** Программа выбирает число в пределе [7,57] и мы угадываем это число. Причем программа начало спрашивает число [7,57], потом если мы ввели число больше чем она выбрала, то спрашивает число [7,введенное нами число], если введенное нами число меньше то [введенное нами число, 57]

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<time.h>**

**void main(){**

**int n,x,a,b,k=1;**

**a=7;**

**b=57;**

**srand(time(NULL));**

**n=7+rand()%51;**

**printf("Введите число [%d,%d] ",a,b);**

**scanf("%d",&x);**

**while(x!=n)**

**{**

**k++;**

**if(x<n)**

**a=x;**

**if(x>n)**

**b=x;**

**printf("Введите число [%d,%d] ",a,b);**

**scanf("%d",&x);**

**}**

**printf("Vi ugadali s %d popitki!!!\n",k);**

**}**

Данная программа работает не совсем правильно, т.е. если мы введем значение выходящие за пределы [a,b], то программа будет нас запутывать, а не помогать. Исправить данную ошибку.

**2.** Программа, которая помогает выучить таблицу умножения. Программа печатает вопрос типа

Сколько будет 6 \* 7?

Затем вводится ответ. Если ответ правильный, то программа печатает “МОЛОДЕЦ!!!!”, если же неверно, то печатает “попробуй еще раз” и дает возможность ввода нового числа, до тех пор пока не получит правильный ответ.

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<time.h>**

**void main()**

**{**

**int n1,n2,n;**

**srand(time(NULL));**

**n1=1+rand()%9;**

**n2=1+rand()%9;**

**printf("Chemy ravno\n%d \* %d ?",n1,n2);**

**printf("\nВведите otvet: ");**

**scanf("%d",&n);**

**while(n!=n1\*n2)**

**{**

**printf("\a\aVi otvetili neverno!!!!\nPopitaytes eshe: ");**

**scanf("%d",&n);**

**}**

**printf("\nMOLODEC!!!!\n");**

**}**

# 4.8 Переменные перечисляемого типа

В программе можно ввести свой тип данных называемый *перечисляемым*, который объявляется следующим образом:

**enum имя\_типа {именная константа\_1, …, именная константа\_n};**

например:

**enum name{Anna, Vera, Masha, Nina, Ira};**

Переменная типа **name**определяется следующим образом:

**enumname**girl; т.е. переменная girl может принимать значения из списка перечисленных в фигурных скобках. Каждому значению из списка соответствует порядковое число *начиная с нуля*, т.е. в фигурных скобках определены именные константы, т.е. константы целого типа: Anna=0, Vera=1 и т.д.

В программе можно определить переменные типа перечисление без задания типа (только одного). Определение имеет вид:

**enum { Anna, Vera, Masha, Nina, Ira } girl\_1,girl\_2,girl\_new=Nina;** т.е. girl\_1, girl\_2, girl\_new переменные данного типа.

Именные константы могут задаваться не по порядку, например:

**enum name{ Anna=1, Vera, Masha=5, Nina, Ira=9 } girl\_1,girl\_2,girl\_new=Nina**

В этом случае значения именных констант будут следующие:

Anna=1, Vera=2, Masha=5, Nina=6, Ira=9

Значения констант могут повторяться, например:

**enum name{ Anna, Vera=1, Masha=5, Nina=Vera-1, Ira=9 };**

В этом случае значения именных констант будут следующие:

Anna=0, Vera=1, Masha=5, Nina=0, Ira=9

Именные константы списка имеют тип **int**. Переменные типа **enum** могут использоваться в индексах выражения, как операнды в арифметических выражениях и в операции отношения. Именные константы из списка перечисление эквивалентно ее численному значению. Именные константы можно устанавливать как положительные, так и отрицательные значения.

При объявления типа перечисление необходимо соблюдать следующие правила:

* Идентификаторы в списке перечисление не должны совпадать с другими идентификаторами той же области видимости;
* Объявляемый тип перечисление (имя) должно быть отлично от других имен типов вводимых пользователем в той же области видимости;
* Именные константы можно устанавливать одинаковые (повторяющиеся) значения, однако имена в списке должны быть разными.

2.Игрок бросает кости. Каждая кость имеет 6 граней, т.е. 1, 2, 3, 4, 5, 6. После броска вычисляется сумма цифр. Если сумма первого броска сумма = 7 или 11, то игрок выиграл. Если сумма первого броска сумма = 2, 3 или 12, то игрок проиграл. (т.е. выиграл “банк”). Если сумма после первого броска сумма = 4, 5, 6, 8, 9 или 10, то эта сумма становится “очком” игрока. Чтобы выиграть, игрок бросает кости до тех пор, пока не выпадет сумма равная его очкам. Игрок проигрывает если в течении бросков выпадет сумма =7.

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<time.h>**

**int rollDice();**

**void main()**

**{**

**enum Status {CONTINUE, WON, LOST};**

**int sum,myPoint;**

**Status gameStatus;**

**srand(time(NULL));**

**sum=rollDice();**

**switch(sum)**

**{**

**case 7: case 11:**

**gameStatus = WON;**

**break;**

**case 2: case 3: case 12:**

**gameStatus= LOST;**

**break;**

**default:**

**gameStatus= CONTINUE;**

**myPoint=sum;**

**printf(“Ochki: %d\n”,myPoint);**

**break;**

**}**

**while(gameStatus== CONTINUE)**

**{**

**sum=rollDice();**

**if(sum==myPoint)**

**gameStatus=WON;**

**else**

**if(sum==7)**

**gameStatus=LOST;**

**}**

**if(gameStatus==WON)**

**printf(“You are WON!!!!\n”);**

**else**

**printf(“You are LOST!!!!\n”);**

**}**

**int rollDice(){**

**int die1,die2,workSum;**

**die1=1+rand()%6;**

**die2=1+rand()%6;**

**workSum=die1+die2;**

**printf(“Brosok igroka %d + %d = %d\n”,die1,die2,workSum);**

**return workSum;**

**}**

1. Преобразовать программу 1, так чтобы мы сами вводили интервал, т.е. **[a,b]**, причем программа должна определять: 1) если **a** или **b** отрицательны или больше, чем 32767, то требовала нового ввода чисел; 2) в какой поочередности бы мы не вводили числа программа должна правильно выводить интервал; 3) если при выполнении программы интервал, например **[10,45]** и ввели число **5**, то интервал был **[10,45]** и, если ввели **60**, то интервал, так же не менялся; 4) определяла длительность выполнения программы

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<time.h>**

**void main()**

**{**

**time\_t t0,t1;**

**t0=time(NULL);**

**int n,x,a,b,k=1,min,max;**

**printf(“a=”);**

**scanf(“%d”,&min);**

**while(min<0 || min>32767)**

**{**

**printf(“a=”);**

**scanf(“%d”,&min);**

**}**

**printf(“b=”);**

**scanf(“%d”,&max);**

**while(max<0 || max>32767)**

**{**

**printf(“b=”);**

**scanf(“%d”,&max);**

**}**

**a=b=min;**

**if(a>max)**

**a=max;**

**else**

**b=max;**

**srand(time(NULL));**

**n=a+rand()%(b-a+1);**

**printf(“Введите число [%d,%d] “,a,b);**

**scanf(“%d”,&x);**

**while(x!=n)**

**{**

**k++;**

**if(x<n && x>a)**

**a=x;**

**if(x>n && x<b)**

**b=x;**

**printf(“Введите число [%d,%d] “,a,b);**

**scanf(“%d”,&x);**

**}**

**t1=time(NULL);**

**printf(“Vi ugadali s %d popitki!!!\n%lf”,k,difftime(t1,t0));**

**}**

# *4.9* Связь массивов и указателей.

В языке “С”, имя массива это скрытый указатель на первый (с нулевым индексом) элемент массива. Любой доступ к элементам массива, осуществляемый операцией индексирования, может быть выполнен и при помощи указателя. Декларация

**int x[10]**;

определяет массив ‘**x**’ из 10 элементов, т.е. блок из10 последовательных элементов с именем **x[0],x[1]……,x[9]** и определяет значение указателя **x**, которое равно адресу элемента **x[0**]. Значение указателя x менять нельзя, т.к. оно является константой, но ее можно присвоить другому указателю и изменить его же значение, а также использовать в выражениях.

**x[6]** тоже самое, что и **\*(x+6)**,

**&x[6]**  тоже самое, что и **(x+6)**

**array**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 45 | 56 | -89 | 6 | 12 | -4 | 56 | -6 | 7 | 123 |

**int** array[10]**;**

**int \*p;**

Если объявлены массив **array** и указатель того же типа, то

**p=&array[0];** или **p=array;** мы направляем указатель **p** на массив.

И **(p+6), &array[6], &p[6]** таким образом мы обращаемся к адресу 6 элемента, а \***(p+6), array[6], p[6]** таким образом мы обращаемся к самому элементу, который записан под 6 индексом.

# *4.10* Передача массивов функции

***Прототип функции, которой передается массив***

**возвр *\_*знач имя\_функции (тип\_данных \*, const int);**

Пример: Ввода и вывода одномерного целочисленного массива, с помощью функции

**#include<stdio.h>**

**void vvod\_mas(int \*,const int);**

**void vivod\_mas(int \*,const int);**

**void main()**

**{**

**const int n=5,m=6;**

**int x[n],y[m];**

**vvod\_mas(x,n);**

**vvod\_mas(y,m);**

**printf("Содержание первого массива:\n");**

**vivod\_mas(x,n);**

**printf("Содержание второго массива:\n");**

**vivod\_mas(y,m);**

**}**

**void vvod\_mas(int \*p,const int k){**

**for(int i=0;i<k;i++){**

**printf("[%d]=",i);**

**scanf("%d",(p+i));**

**}**

**}**

**void vivod\_mas(int \*p,const int k){**

**for(int i=0;i<k;i++)**

**printf("[%d]=%d\n",i,\*(p+i));**

**}**

**Задача :**

1. Написать следующие функции для одномерного массива:

* + - * ввод массива,
      * вывод массива,
* нахождения и максимального элемента и его место,
  + - * нахождения и минимального элемента и его место
      * сортировка массива по убыванию 6. сортировка массива по возрастанию.

2.Переделать задачу один, так чтобы в программе мы использовали оператор **switch**, если мы введем 1, то показывает максимальный элемент и его место

2, то показывает минимальный элемент и его место

3, то сортирует массив по убыванию

4, то сортирует массив по возрастанию

5, то работа программы завершена

# 4.11 Контрольные вопросы:

1. Функция активизируется с помощью \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. Переменная, которая известна только внутри функции, в которой она определена, называется \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. Оператор \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ в функции используется, чтобы передать значение выражения в функцию, в которой осуществлялся вызов.
4. Ключевое слово \_\_\_\_\_\_\_\_\_\_\_\_ используется в прототипе функции, чтобы указать, что функция не возвращает значений.
5. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ позволяет компилятору проверить количество, типы и порядок следования параметров, передаваемых функции.
6. Укажите все классы памяти, которые Вы знаете \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
7. Переменная, объявленная в блоке или в списке параметров функции, имеют класс памяти \_\_\_\_\_\_\_\_\_\_\_\_\_, если не указано иное.
8. Переменная, объявленная вне любого блока или функции, является \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
9. Для того, чтобы локальная переменная сохраняла свое значения между вызовами функции, ее необходимо объявить как \_\_\_\_\_\_\_\_\_\_\_\_\_ переменную.
10. Определить содержат ли следующие фрагменты ошибки и укажите правильную версию описание функции,

a) которая получает три целых числа и возвращает сумму полученных чисел

int sum (a, b, c)

{

return a+b+c;

}

b) которая получает два целых числа и печатает деление этих чисел.

void printResult ( int a, int b)

{

printf(“%d/%d=%lf\n”,a,b,a/b);

return a/b;

}

1. Указатель – это переменная, которая содержит в качестве своего значения \_\_\_\_\_\_\_\_\_\_\_ другой переменной.
2. Для присвоения указателю начального значения можно использовать три значения \_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_ и \_\_\_\_\_\_\_\_\_\_\_\_\_.
3. Напишите прототип функции exchange, которая получает в качестве параметров два указателя на числа с плавающей точкой x и y и не возвращает никакого значения.
4. Объявите массив типа float с именем number с 10 элементами и присвойте начальные значения 0.0, 1.1, 2.2, 3.3, …., 9.9.

а) приведите два различных оператора, которые присваивают начальный адрес массива number указателю float \*pf \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

б) напечатайте элементы массива обращаясь к ним через указатель pf \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Пусть в функции объявлены переменные double a[10]; int i; Какие из операторов цикла обеспечат присвоение 10 числовых значений всем элементам массива:

а) for(i=0; i<10; i++)

scanf(“%lf”,a+i);

б) for(i=0; i<10; i++)

scanf(“%lf”,a[i]);

с) for(i=1; i<=10; i++)

scanf(“%lf”,&a[i]);

д) for(i=0; i<10; i++)

scanf(“%lf”,&a[0]+i);

16 Функция \_\_\_\_\_\_\_\_\_\_\_ используется для получения случайных чисел.

17 Функция \_\_\_\_\_\_\_\_\_\_\_\_ для того, чтобы при каждом вызове программы выдавались различные числа.

**Задачи:**

1. Программа, которая через функции вводит массив и сортирует I массив в возрастающем порядке, а второй II в убывающем порядке и функция вывода массива. А третий массив переписать число из двух массивов, так чтобы они были записаны в возрастающем порядке.

2. Введите два массива вещественных чисел, размерностью 5 и 7. Отсортировать первый массив по убыванию и распечатать только, те числа, которые находятся в обоих массивах.

3. Найти третий по величине элемент одномерного массива. В массиве могут быть повторяющиеся элементы

4. Вводим число, например 45678. Найти сумму данного числа и числа обратного данному числу. (т.е. 45678+87654)

§5 СИМВОЛЬНАЯ СТРОКА

# *5.1* Символы и работа с ними

Основным символьным типом в С является тип char. Переменная типа char может принимать значения от -128 до +127. Каждая цифра из этого диапазона соответствует какому-то символу. Таблица перевода из числового значения к символьному называется таблицей ASCII-кодов.

Одиночные символы используются в языке в одинарных кавычках – ‘a’, ‘B’, ‘8’, ‘+’.

Служебные символы, такие как ‘\n’ или ‘\t’ также являются литералами.

Простые примеры использования символов:

1. Ввести символ. Вывести его код и сам символ.

#include<stdio.h>

#include<conio.h>

#include <locale.h>

int main()

{

setlocale(LC\_ALL,"Russian");

char c1; //Объявляем переменные для символов

printf( "введите символ " );

scanf("%c",&c1);

printf("вы ввели символ %c его код %d\n",c1,c1); //Выводим

getch();

return 0;

}

2. Ввести все буквы от А до Z.

#include<stdio.h>

#include<conio.h>

#include <locale.h>

int main()

{

setlocale(LC\_ALL,"Russian");

char i; //Объявляем переменные для символов

for(i=65;i<=90;i++)

printf(" %c \t",i); //Выводим

getch();

return 0;

}

3.Вывести в обратном порядке маленькие английские буквы от z до a.

4.Написать функцию, которая определяет, является ли введенный символ буквой или нет.

#include<stdio.h>

#include<conio.h>

int IsLetter( char c );

int main()

{

char c1='+',c2=';',c3='z';

printf("a - %d \n",IsLetter('a')); //Проверяем, является ли символ буквой

printf("G - %d \n",IsLetter('G'));

printf("5 - %d \n",IsLetter('5'));

printf("+ - %d \n",IsLetter( c1 ));

printf("; - %d \n",IsLetter( c2 ));

printf("z - %d \n",IsLetter( c3 ));

getch();

return 0;

}

int IsLetter( char c )

{

if( (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') ) //Если символ между a и z или между A и Z

return 1; //Он является буквой. Возвращаем 1

return 0; //Иначе возвращаем 0

}

5.Написать 3 функции, определяющие:

* Является ли символ цифрой.
* Является ли символ буквой в верхнем регистре.
* Является ли символ буквой в нижнем регистре.

# 5.2 Объявление символьной строки

Строка – это нуль или более символов, заключенных в двойные кавычки. Например: “Привет!!!”, и кавычки не входят в строку. Фактически строковая константа – эта массив символов. Во внутреннем представлении строки в конце автоматически помещается нуль-символ **‘\0’**.

Нулевой байт – это байт, каждый бит которого равен нулю, для него определена символьная константа ´\0´ (признак окончания строки или нуль-терминатор). Поэтому, если строка должна состоять из *k* символов, то в описании массива необходимо указать размер *k*+1.

Например, описание char a[7], означает, что строка содержит шесть символов, а последний байт отведен под нуль-терминатор.

Строка доступна через указатель на первый символ в строке, а значением строки является адрес ее первого символа. В этом смысле строки подобны массивам, потому что массив так же является указателем на свой первый элемент. Если задается размер массива, а строка длиннее, то выдается сообщение об ошибке. Если задан размер массива, а строка короче, то лишние элементы массива содержат так называемый “мусор”.

Необходимо запомнить, что символьная константа и строка, содержащая один символ *не одно и тоже!!!*, т. е. ‘x’!=”x”, т.к. “x” – это массив символов, который кроме символа x содержит еще и символ ‘\0’.

***Объявления символьной строки***

**charcolor[]=”синий”;**

**char color[6]=”синий”;**

**char \*color=”синий”;**

**charcolor[]=” ’с’ ‘и’ ‘н’ ‘и’ ‘й’ ‘\0’ ”;**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |
| ‘с’ | ‘и’ | ‘н’ | ‘и’ | ‘й’ | ‘\0’ |

Все три объявления строки возможны, но предпочтительны первые два.

***Функции ввода символьной строки***

**char\* gets( char \*buffer);** считывает символьную строку из стандартного входного потока и помещает ее по адресу, заданному указателем **buffer**. Прием строки заканчивается, если функция обнаружит символ ‘\n’ и данный символ заменяется на символ ‘\0’.

*Возвращает* указатель на считанную строку.

**int scanf(constchar \* format, \*buffer)** считывает символьную строку из стандартного входного потока и помещает ее по адресу, заданному указателем **buffer**. Прием строки заканчивается, если функция обнаружит символ ‘\n’или” ” и данный символ заменяется на символ ‘\0’.

*Возвращаемое значение* – число считанных полей

***Функции вывода символьной строки***

**int puts( constchar \*string );** записывает символ в стандартный поток данных и символ ‘\0’заменяется на ’\n’.

*Возвращает* код переданного символа..

**int printf(constchar \* format, ……);** записывает символ в стандартный поток вывода символ ‘\0’ игнорируется.

*Возвращаемое значение* – число записанный в поток символ

**int sprintf( char \*s, const char \* format )** Альтернатива printf, но результат помещается в строку s, а не отображается на экране**.**

Примеры:

1. Ввести слово и напечатать его на экране.

#include<stdio.h>

#include<conio.h>

int main()

{

char str1[20];

scanf( "%s", str1 ); //Ввод строки с клавиатуры

printf( "String is : %s", str1 ); //Вывод строки на экран

getch();

return 0;

}

Обратите внимание, что при вводе нескольких слов, выведется только первое. Для вывода строки целиком, используйте gets();

1. Сформировать строку с математическими действиями и вывести ее на экран

#include<stdio.h>

#include<conio.h>

int main()

{

char str1[20]; //строка

int a = 5; //Переменные для расчета и вывода

int b = 10;

sprintf( str1, "%d + %d = %d", a, b,a + b);//Записываем в строку str1 выражение

puts( str1 ); //Выводим строку на экран

getch();

return 0;}

1. Ввести строку и распечатать каждый ее символ с новой строки

**#include<stdio.h>**

**void main()**

**{**

**char x[100];**

**int i;**

**puts("Введите строку: ");**

**gets(x);**

**for(i=0;x[i]!='\0';i++)**

**printf("%c\n",x[i]);**

**}**

1. Ввести строку определить, сколько символов содержит данная строка

**#include<stdio.h>**

**void main()**

**{**

**char x[100];**

**int i;**

**puts("Введите строку: ");**

**gets(x);**

**for(i=0;x[i]!='\0';i++);**

**printf("\n%s - содержит %d символов\n",x,i);**

**}**

1. Ввести строку и распечатать данную строку в обратном порядке.
2. Ввести строку и переписать ее в обратном порядке в новую строку.
3. Ввести строку заменить символ «а» на символ «!» в данной строке.
4. Посчитать сколько раз в словах встречается буква «а» и поменять первый и последний символ
5. Посчитать число слов
6. Написать функцию присвоения строк.   
   Прототип функции **char\* StrCopy( char \*s1, char \*s2 );** т.е. присвоить вторую строку первой и вернуть первую строку.

#include<stdio.h>

#include<conio.h>

char\* StrCopy( char \*s1, char \*s2 );

int main()

{

char str1[20] = "First string"; //Создаем 2 строки

char str2[20] = "Second string";

printf( "Before copy :\n" ); //Выводим строки до копирования

printf( "str1 - %s\n", str1 );

printf( "str2 - %s\n", str2 );

StrCopy( str1, str2 ); //Копируем строку 2 в строку 1

printf( "\nAfter copy :\n" ); //Выводим строки после копирования

printf( "str1 - %s\n", str1 );

printf( "str2 - %s\n", str2 );

getch();

return 0;

}

char\* StrCopy( char \*s1, char \*s2 )

{

int i = 0; //Счетчик

do

{

s1[ i ] = s2[ i ]; //Копируем все символы из s2 в s1

}while( s2[ i++ ] != '\0' ); //Пока не дойдем до \0

return s1; //возвращаем указатель на строку

}

1. Написать функцию добавления строк. Прототип функции **char\* dobav(char\*,char\*);** т.е. добавить вторую строку первой и вернуть первую строку.
2. Написать функцию, которая считает количество введенных символов. Прототип функции **int StrLength( char \* str );**получает строку и возвращает количество введенных символов

#include<stdio.h>

#include<conio.h>

int StrLength( char \* str );

int main()

{

char \*sptr = "String Pointer"; //Создаем строку через указатель

char sarr[] = "String Array"; //Создаем строку как массив

char csarr[] = {'a','r','r','a','y','\0'}; //Создаем строку как массив

printf( "ABCDEF - %d\n", StrLength( "ABCDEF" ) );//Выводим длины строк

printf( "String Pointer - %d\n", StrLength( sptr ) );

printf( "String Array - %d\n", StrLength( sarr ) );

printf( "array- %d\n", StrLength( csarr ) );

getch();

return 0;

}

int StrLength( char \* str )

{

int i = 0; //Счетчик

while( str[i] != '\0' ) //Пока очередной символ не равен '\0'

{

i++; //Увеличиваем счетчик

}

return i; //Возвращаем значение счетчика

}

1. Написать функцию, которая сравнивает строки и возвращает: **0** если строки равны; **1** если в первой строке встретился символ больше чем во второй строке и **-1** в противном случае. Прототип функции **int\* srav(char\*, char\*);**

**#include<stdio.h>**

**int srav(char \*,char \*);**

**void main()**

**{**

**char x[100],y[100];**

**int l;**

**puts("Введите строку: ");**

**gets(x);**

**puts("Введите строку: ");**

**gets(y);**

**l=srav(x,y);**

**if(l==0)**

**printf("строки одинаковы==\n");**

**else**

**printf("строки не одинаковы!=%d\n",l);**

**}**

**int srav(char \*x,char \*y)**

**{**

**for(int i=0;x[i] || y[i];i++)**

**{**

**if(x[i]>y[i])**

**return 1;**

**if(x[i]<y[i])**

**return -1;**

**}**

**return 0;}**

# *5.3* Функции работы со строками из библиотеки <string.h>

**char\*strcpy(char\*s1,constchar\*s2);** копирует строку **s2** в строку **s1**, причем массив строк **s1** должен быть достаточно большим, чтобы хранить строку **s2** и символ конец строки содержащий в копирующей строке. Возвращает значение строки **s1.**

**char\*strcnpy(char\*s1,constchar\*s2, size\_tn);** копирует не более **n**символов строки **s2** в строку **s1**, причем массив строк **s1** должен быть достаточно большим, чтобы хранить строку **s2** и символ конец строки содержащий в копирующей строке. Возвращает значение строки **s1.**

**char\*strcat(char\*s1,constchar\*s2);** добавляет строку **s2** строке **s1**, причем массив строк **s1** должен быть достаточно большим, чтобы хранить строки **s1** и s2, а так же символ конец строки. Возвращает значение строки **s1.**

**char\*strncat(char\*s1,constchar\*s2, size\_tn);** добавляет не более **n**символов строки **s2** в строке **s1**, причем массив строк **s1** должен быть достаточно большим, чтобы хранить строки **s1** и s2, а так же символ конец строки. Возвращает значение строки **s1.**

**Int strcmp(constchar\*s1,constchar\*s2);** сравнивает посимвольно строки **s1** и **s2**, возвращает **0**, если строки одинаковы, **1** если встречается хотя бы один символ в строке **s1**, код которого больше, чем код символа строки **s2** и **-1,** если встречается хотя бы один символ в строке **s1**, код которого меньше, чем код символа строки **s2.**

**Int strncmp(constchar\*s1,constchar\*s2,size\_tn);** сравнивает не более **n** символов строки **s1** и **s2**, возвращает **0**, если строки одинаковы, **1** если встречается хотя бы один символ в строке **s1**, код которого больше, чем код символа строки **s2** и **-1,** если встречается хотя бы один символ в строке **s1**, код которого меньше, чем код символа строки **s2.**

**size\_t strlen(constchar\*s);** определяет длину строки **s** и возвращает количество символов, предшествующих символу конец строки. **sise\_t**– это целый без знака тип, который определен в **<stddef.h>**

**char\*strtok(char\*s1,constchar\*s2);** функция разбивает строку **s1** на *лексему* (последовательность символов, отделенной символом *разделителем*, разделителем обычно служат пробелы или знаки пунктуации и задается как строка **s2**). Если последовательно вызывать **strtok,** можно разбить строку **s1** на логические куски, разделенные символом, содержащим в строке **s2**. Первый вызов содержит в качестве первого аргумента строку **s1**, а последующие вызовы для обработки той же строки, содержит в качестве первого аргумента **NULL**. При каждом вызове возвращает указатель на текущую лексему.

**#include<stdio.h>**

**#include<string.h>**

**void main()**

**{**

**char x[]="afs,gaf, agshdfah, ash%%gdha, as%gda%";**

**char \*ptr;**

**printf("разбитие строки на лексемы:\n");**

**ptr=strtok(x,",");**

**while(ptr!=NULL)**

**{**

**printf("%s\n",ptr);**

**ptr=strtok(NULL,",");**

**}**

**}**

**В результате работы данной программы получим:**

***разбитие строки на лексемы:***

***afs***

***gaf***

***agshdfah***

***ash%%gdha***

***as%gda%***

**Задачи:**

1. Имеем 4 строки, например **charx[20], y[20], z[20], t[80];** В первую строку вводим фамилию, во-вторую имя, а в третью отчество и используя функции из библиотеке **<string.h>** записываем в строку **z** сначала фамилию потом пробел, имя пробел и отчество пробел. Распечатываем только строку **z**.
2. Введите строку. Если длина строки а) больше 10, то удалить два последних символа; б) меньше 10, то удалить два первых символа; в) равно 10, то удалить символ по середине.
3. Ввести число и распечатать, те цифры данного числа, которые делятся на 2.
4. Ввести число и посчитать произведение цифр данного числа.
5. Ввести число и распечатать цифры данного числа через два пробела. Например, если ввели 123456, то печатает 1 2 3 4 5 6.
6. Ввести строку, которая содержит целые числа и преобразовать ее в число.

**#include<stdio.h>**

**void main()**

**{**

**char m[100];**

**int n=0,i;**

**printf("Ввести строку: ");**

**scanf("%s",m);**

**for(i=0;m[i];i++)**

**{**

**n\*=10; n+=m[i]-'0';**

**}**

**printf("\n%d\n",n);**

**}**

1. Ввести дату в строку следующим образом: 12/05/1956 и программа распечатывает строку следующим образом: 12 мая, 1956 года (использовать оператор **switch**).
2. \*Ввести число, например 3451 программа должна напечатать три тысячи четыреста пятьдесят один.
3. Ввести число, посчитать, сумму цифр введенного числа используя **char**

**#include<stdio.h>**

**void main()**

**{**

**char s[50];**

**int x,i,sum=0;**

**printf("Введите число: ");**

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

**for(i=0;s[i]!='\0';i++)**

**{**

**if(s[i]=='-' && i==0)**

**continue;**

**x=s[i]-'0';//преобразования символа в число**

**sum+=x;**

**}**

**printf("summa=%d\n",sum);**

**}**

10. *С помощью данной программы можно определить код символа.*

**#include<stdio.h>**

**void main()**

**{**

**int x;**

**char ch;**

**x=getchar();**

**ch=(char)x;**

**printf("%c=%d\n",ch,x);**

**}**

11. За один просмотр исходного текста определить, сколько раз встречается каждый символ

**#include<stdio.h>**

**void main()**

**{**

**char arr[256],text[500];**

**int i,j;**

**for(i=0;i<256;i++)**

**arr[i]=0;//Обнуляем массив встречаемости символов**

**printf("\nВведите текст:\n");**

**gets(text);**

**for(j=0;text[j];j++)**

**{**

**i=(unsigned char)text[j];//аналогично i=(int)text[j];**

**arr[i]++;**

**}**

**for(i=0;i<256;i++)**

**{**

**if(arr[i])**

**printf("\nСимвол '%c' встречается %d раз",i,arr[i]);// если в printf ‘%c’,то i выводится как //символ.**

**}**

**}**

12.Ввести строку. Написать функцию, которая получает данную строку и выводит на экран результатов каждое слова с новой строки. Например, если ввели

Москва столица России!!! 123 456, то получаем

Москва

столица

России!!!

123

456

Прежде чем двигаться дальше, обязательно выполните задания на закрепление темы строки.

# Упражнения для закрепления темы Строки символов

Приступая к решению задач этого раздела, следует вспомнить, что:

1. Строка представляет собой массив символов, заканчивающихся ноль - символом. Нумерация элементов начинается с ноля.
2. Для просмотра всех символов используется цикл от 0 до конца строки.
3. Для вывода строки на экран и получения строки с клавиатуры, используются операторы puts() и gets();

**Задачи**

1. Составить программу, которая запрашивает имя человека и повторяет его на экране с приветствием.
2. Составить программу, которая запрашивает название футбольной команды и повторяет его на экране со словами «…-это чемпион!»
3. Составить программу, которая запрашивает строку. Посчитать сколько раз в словах встречается буква «а» и поменять первый и последний символ.
4. Составить программу, которая запрашивает фразу. Записать фразу в обратном порядке.
5. Составить программу, которая запрашивает строку. Уплотнить текст, убрав из него все пробелы.
6. Составить программу, которая запрашивает строку. Посчитать число слов.
7. Составить программу, которая запрашивает строку. Вывести на экран третий символ.
8. Дано слово. Вывести на экран буквосочетание из его второй и четвертой буквы.
9. Составить программу формирующую строку состоящую из указанного в программе количества символов \*. Например: если введем 12, то строка на экране \*\*\*\*\*\*\*\*\*\*\*\*.
10. \*Дано предложение. Определить сколько в нем гласных букв.

# 5.4 Массив строк

**char array[3][50];** объявляется двумерный массив **array**, состоящий из трех строк по 50 байт.

Примеры:

1. Ввод и вывод массива строк.

**#include<stdio.h>**

**void main()**

**{**

**char string[5][20];**

**int i;**

**for(i=0;i<5;i++)**

**{**

**printf("Строка %d\n",i+1);**

**gets(string[i]);**

**}**

**for(i=0;i<5;i++)**

**puts(string[i]);**

**}**

1. Ввести массив из пяти строк, рассортировать в алфавитном порядке

**#include<stdio.h>**

**#include<string.h>**

**void main()**

**{**

**char string[5][20],buf[20];**

**int i,j,k;**

**for(i=0;i<5;i++)**

**{**

**printf("Строка %d\n",i+1);**

**gets(string[i]);**

**}**

**for(i=0;i<4;i++)**

**{**

**for(k=i,j=i+1;j<5;j++)**

**if(strcmp(string[k],string[j])>0)**

**k=j;**

**strcpy(buf,string[i]);**

**strcpy(string[i],string[k]);**

**strcpy(string[k],buf);**

**}**

**puts("");**

**for(i=0;i<5;i++)**

**puts(string[i]);**

**}**

1. Ввести массив строк, в который записываются целые числа, преобразовать их в массив чисел и вывести.

**#include<stdio.h>**

**void main()**

**{**

**char m[5][100],\*s;**

**int n[5],i;**

**for(i=0;i<5;i++)**

**scanf("%s",m[i]);**

**for(i=0;i<5;i++)**

**{**

**for(s=m[i],n[i]=0;\*s;s++)**

**{**

**n[i]\*=10;**

**n[i]+=\*s-'0';**

**}**

**}**

**puts("");**

**for(i=0;i<5;i++)**

**printf("%d\n",n[i]);**

**}**

§6 ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ПАМЯТИ

Одним из способов хранение информации, является использование системы динамического выделения памяти языка “С”. При этом память выделяется из свободной области памяти по мере надобности и возвращается назад, т.е. освобождается, когда необходимость в ней исчезает. Т.к. память выделяется и освобождается, то можно использовать ту же самую память в другой момент времени и для других целей в другой части программы. Ядром динамического выделения памяти в “С” являются функции, объявленные в стандартной библиотеке в загаловочном файле **stdlib.h.**

# *6.1* Функция malloc()

Функция **malloc()** осуществляется запрос на выделение свободной памяти и при наличии таковой, запрошенный объем свободной памяти выделяется на нужды программы. Когда надобность в памяти отпадает, ее нужно освободить при помощи функции **free()**, при этом освобожденная память возвращается назад системе и вновь доступна для использования в этой или других программах.

**void \* malloc(unsignedsize)**

Функция **malloc**выделяет область памяти размером *size* байт. В случае успеха, malloc возвращает указатель на начало выделенного блока памяти. Если для выделенной памяти не хватает места, возвращает **NULL**.

**void free(free \*ptr)**

Функция *освобождает* область памяти, ранее выделенной при помощи функции **malloc** на которую указывает ptr. Если ptr- **NULL**, то free ничего не выполняет. Если ptr не является указателем, проинициализированным ранее одной из функций выделения памяти, то поведение функции не определено. Функция free не располагает средствами передачи ошибки, возможно возникающей при ее выполнении, равно как возвращаемым значением.

1. объявляется указатель на какой-то тип данных.

тип\_данных \*имя\_указателя;

2. вызов функции **malloc()** для выделения области памяти

имя\_указателя=(тип\_данных \*)**malloc**(количество\*sizeof(тип\_данных));

3. проверка, выделилась ли память

if(!имя\_указателя)

{

**printf**(“Нет места!!!!\n”);

**exit**(1);// прекращение работы программы, причем в качестве аргумента, может быть //использовано любое целое число. Одна по соглашению было принято передавать 1, если причиной //прерывания программы послужила ошибка и 0 в остальных случаях.

}

4. использование выделенной памяти.

5. освобождение памяти

**free(**имя\_указателя**);**

Пример:

1. Программа, которая динамически выделяет массив целых чисел.

**#include<stdio.h>**

**#include<stdlib.h>**

**#include <malloc.h>**

**void main(){**

**int \*ptr,n,i;**

**printf("Введите размер массива: ");**

**scanf("%d",&n);**

**ptr=(int \*)malloc(n\*sizeof(int));**

**if(!ptr)**

**{**

**printf("Нет места!!!\n");**

**exit(1);**

**}**

**for(i=0;i<n;i++)**

**{**

**printf("[%d]=",i);**

**scanf("%d",&ptr[i]);**

**}**

**free(ptr);**

**}**

2. Программа, которая вводит строку, считает количество введенных символов, выделяет место и переписывает введенную строку в обратном порядке.

**#include<stdio.h>**

**#include<stdlib.h>**

**#include <malloc.h>**

**#include<string.h>**

**void main(){**

**char x[100];**

**char \*ptr;**

**int n,i,j;**

**printf("Введите строку: ");**

**gets(x);**

**n=strlen(x);**

**ptr=(char \*)malloc((n+1)\*sizeof(char));//для символьной строки, необходимо выделять на1 символ**

**//больше, для символа конец строки**

**if(!ptr)**

**{**

**printf("Нет места!!!\n");**

**exit(1);**

**}**

**for(j=0,i=n-1;i>=0;i--,j++)**

**{**

**ptr[j]=x[i];**

**}**

**ptr[j]='\0';**

**puts(ptr);**

**free(ptr);**

**}**

**Задачи:**

1. Написать программу, которая выделяет места, для двух целочисленных массивов. Ввод данных и сортировка, массивов в порядке возрастания, с использованием функций. За один просмотр получить общий рассортированный массив.

**#include<stdio.h>**

**#include<stdlib.h>**

**#include <malloc.h>**

**void vvod\_mas(int\*,int);**

**void sort\_vozr(int\*,int);**

**void main()**

**{**

**int \*ptr1,\*ptr2,\*ptr3,n1,n2,i1=0,i2=0,i3=0;**

**printf("Введите размер I-го массива: ");**

**scanf("%d",&n1);**

**ptr1=(int \*)malloc(n1\*sizeof(int));**

**if(!ptr1)**

**{**

**printf("Нет места!!!\n");**

**exit(1);**

**}**

**printf("Введите размер II-го массива: ");**

**scanf("%d",&n2);**

**ptr2=(int \*)malloc(n2\*sizeof(int));**

**if(!ptr2)**

**{**

**printf("Нет места!!!\n");**

**exit(1);**

**}**

**ptr3=(int \*)malloc((n2+n1)\*sizeof(int));**

**if(!ptr3)**

**{**

**printf("Нет места!!!\n");**

**exit(1);**

**}**

**printf("Введите I-ий массив:\n");**

**vvod\_mas(ptr1,n1);**

**printf("Введите II-ой массив:\n");**

**vvod\_mas(ptr2,n2);**

**sort\_vozr(ptr1,n1);**

**sort\_vozr(ptr2,n2);**

**while((i1<n1) && (i2<n2))**

**{**

**if(ptr1[i1]<ptr2[i2])**

**ptr3[i3]=ptr1[i1++];**

**else**

**ptr3[i3]=ptr2[i2++];**

**i3++;**

**}**

**while((i1<n1) || (i2<n2))**

**{**

**if(i1<n1)**

**ptr3[i3++]=ptr1[i1++];**

**else**

**ptr3[i3++]=ptr2[i2++];**

**}**

**printf("\nРезультат:\n");**

**for(i1=0;i1<n1+n2;i1++)**

**printf("%d ",ptr3[i1]);**

**printf("\n");**

**free(ptr1);**

**free(ptr2);**

**free(ptr3);**

**}**

**void sort\_vozr(int\*x,int n)**

**{**

**int k,buf,i,j;**

**for(i=0;i<n-1;i++)**

**{**

**for(k=i,j=i+1;j<n;j++)**

**{**

**if(x[k]>x[j])**

**k=j;**

**}**

**buf=x[k];**

**x[k]=x[i];**

**x[i]=buf;**

**}**

**}**

**void vvod\_mas(int\*x,int n)**

**{**

**int i;**

**for(i=0;i<n;i++)**

**{**

**printf("[%d]=",i);**

**scanf("%d",&x[i]);**

**}**

**}**

1. Написать программу, которая вводит строку и проверяет, является ли данная строка палиндромом. Палиндромом называется та строка, которая при считывании и с начало и с конца одинакова, например 1234321 или сос. После ввода строки считаем длину строки и динамически выделяем место, куда переписываем данную строку и проверяем, совпали они или нет.
2. Ввести строку. Если длина строки а) >10, то удалить 2 первых символа строки и 3 последних символа в строке б)< 10, то удалить символ, находящийся по середины строки.

# *6.2* Функция сalloc()

Функция **сalloc()** как и функция **malloc()** осуществляется запрос на выделение свободной памяти и при наличии таковой, запрошенный объем свободной памяти выделяется на нужды программы, но различие в том что выделяемая память инициализируется нулями.

**void \* сalloc(unsigned num ,unsigned size)**

Функция **calloc** выделяет область памяти размером num\**size* байтов. В случае успеха, calloc возвращает указатель на начало выделенного блока памяти. Если для выделенной памяти не хватает места, возвращает **NULL**.

1. объявляется указатель на какой-то тип данных.

тип\_данных \*имя\_указателя;

2. вызов функции **calloc()** для выделения области памяти

имя\_указателя=(тип\_данных \*)**calloc**(количество, sizeof(тип\_данных));

3. проверка, выделилась ли память

if(!имя\_указателя)

{

**printf**(“Нет места!!!!\n”);

**exit**(1);// прекращение работы программы, причем в качестве аргумента, может быть //использовано любое целое число. Одна по соглашению было принято передавать 1, если причиной //прерывания программы послужила ошибка и 0 в остальных случаях.

}

4. использование выделенной памяти.

5. освобождение памяти

**free(**имя\_указателя**);**

Функцию **calloc()** можно использовать как и функцию **malloc().**

# *6.3* Функция realloc*()*

**void \* realloc(void \*ptr ,unsigned size)**

Функция **realloc**изменяет размер *динамически выделенной памяти*, на которую указывает ptr на новый размер - *size* байтов. Т.е для того чтобы можно было изменить размер памяти его необходимо обязательно динамически выделить с помощь функции **malloc**или функции **calloc** и при изменении размера выделяется новая память, а не меняется размер выделенной памяти следовательно необходимо переадресовывать указатель. Рассмотрим пример использования функции **realloc.**

void main(){

int\*p1, \*p2;

p1= (int \*)malloc(5\*sizeof(int));

if(!p1){

printf(“Error!!!!”);

exit(1);}

……………………

p2=(int \*)realloc(p1,10\*sizeof(int));//10 это новый размер массива, т.е. добавляется //еще 5 елементов

if(p2)//если память выделилась

p1=p2;//переадресация указателя

else{

printf(“Error!!!!”);

exit(1);}

}

Задача: Вести размер массива и динамически выделить память. Заполнить элементы массива и удалить все 5 находящиеся в массиве.

**#include<stdio.h>**

**#include<stdlib.h>**

**void vvod\_mas(int \*,const int);**

**void vivod\_mas(int \*,const int);**

**void main(){**

**int n,k=0;**

**int \*x,\*y;**

**printf("Введите размер массива: ");**

**scanf("%d",&n);**

**x=(int \*)malloc(n\*sizeof(int));**

**if(!x)return;**

**vvod\_mas(x,n);**

**printf("Содержание массива:\n");**

**vivod\_mas(x,n);**

**for(int i=0;i<n;i++){**

**if(x[i]==5){**

**k++;**

**printf("\nk=%d\n",k);**

**for(int j=i+1;j<n-k;j++)**

**x[j-1]=x[j];**

**} }**

**n=n-k;**

**y=(int\*)realloc(x,n\*sizeof(int));**

**if(y)**

**x=y;**

**else**

**return;**

**printf("Содержание нового массива:\n");**

**vivod\_mas(x,n);**

**}**

**void vvod\_mas(int \*p,const int k){**

**for(int i=0;i<k;i++){**

**printf("[%d]=",i);**

**scanf("%d",(p+i));**

**}**

**}**

**void vivod\_mas(int \*p,const int k){**

**for(int i=0;i<k;i++)**

**printf("[%d]=%d\n",i,\*(p+i));**

**}**

§7 СТРУКТУРА

# *7.1* Объявление структуры

Структура – это составной объект языка Си, представляющий собой совокупность логически связанных данных различного типа, объединенных в группу под одним идентификатором. Данные, входящие в эту группу, называют полями.

Термин «структура» в языке Си соответствует двум разным по смыслу понятиям:

– структура – обозначение участка оперативной памяти, где располагаются конкретные значения данных; в дальнейшем – это структурная переменная, поля которой располагаются в смежных областях памяти;

– структура – правила формирования структурной переменной, которыми руководствуется компилятор при выделении ей места в памяти и организации доступа к ее полям.

Определение объектов типа структуры производится за два шага:

– декларация структурного типа данных, не приводящая к выделению участка памяти;

– определение структурных переменных с выделением памяти.

**Структура – это тип данных**, созданный нами, построенный с использованием других типов. Ключевое слово **struct** начинает определение структуры.

struct имя\_структуры

{

тип1 имя\_переменной;

тип2 имя\_переменной;

тип3 имя\_переменной; и т.д.

};

**Например:**struct Man{

char fam[20];

char name[15];

int year, day, month;

};

Среди типов данных структуры могут также присутствовать, кроме стандартных типов данных (int, float, char и т.д.) ранее определенные типы, но структура не включать в себе переменные своего типа, примеры:

1. struct date

{

int year, day, month;

};

struct Man

{

char fam[20];

char name[15];

date birthday;

};

2. struct date

{

int year, day, month;

};

struct Man

{

char fam[20];

char name[15];

date birthday;

man any;//неправильно

};

При непосредственной работе со структурой, доступ к компонентам структуры осуществляется с помощью оператора “точка”, а при использовании указателя на структуру “->”. Синтаксис для доступа к компонентам структуры следующий:

имя\_переменной\_структуры. член\_данных

имя\_переменной\_структуры.->член\_данных

Пример определения структурного типа – необходимо создать шаблон, описывающий информацию о студенте: номер группы, Ф.И.О. и средний балл.

Один из возможных вариантов:

struct Stud\_type {

char Number[10];

char Fio[40];

double S\_b;

};

Интерпретация объекта типа Stud\_type:

|  |  |  |
| --- | --- | --- |
| Number | Fio | S\_b |
| 10 | 40 | 8 |

длина в байтах

Структурный тип данных удобно применять для групповой обработки логически связанных объектов. Параметрами таких операций являются адрес и размер структуры.

Примеры групповых операций:

– захват и освобождение памяти для объекта;

– запись и чтение данных, хранящихся на внешних носителях как физические и/или логические записи с известной структурой (при работе с файлами). Так как одним из параметров групповой обработки структурных объектов является размер, не рекомендуется декларировать поле структуры указателем на объект переменной размерности, поскольку в данном случае многие операции со структурными данными будут некорректны.

Структурный тип может быть использован для декларации массивов, эле-

ментами которых являются структурные переменные, например:

struct person spisok[100]; – spisok – массив структур;

или

struct person {

char fio[40];

int day, month, year;

} spisok[100];

В данном случае обращение к полю, например, day i-й записи может быть

выполнено одним из следующих способов:

spisok[i].day, \*(spisok+i).day, (spisok+i)->day.

Рассмотрим это на примере.

1. Пусть есть список из трех человек. Нужно хранить и выводить на экран данные о каждом человеке. Рост, вес, фамилию и имя.

#include<stdio.h>

#include <conio.h>

struct Man

{

char name[50];

char fam[50];

double rost,ves;

};

void main()

{

Man ob[3];

int i;

for( i=0;i<3;i++)

{

printf("%d\nName: ",i+1);

scanf("%s",ob[i].name);

printf("Family: ");

scanf("%s",ob[i].fam);

printf("Rost: ");

scanf("%lf",&ob[i].rost);

printf("VES: ");

scanf("%lf",&ob[i].ves);

}

for(i=0;i<3;i++)

{

printf("%d\n%s\t%s\n%.1lf\t%.1lf\n",i+1,ob[i].fam,ob[i].name,ob[i].rost,ob[i].ves);

}

getch();

}

Пример вложения структуры внутрь другой структуры.

2. Есть данные о студенте, его дате рождения, группе и оценке. Осуществить ввод/вывод данных.

#include<stdio.h>

#include <conio.h>

#include<stdio.h>

struct date

{

int day,month,year;

};

struct Person

{

char fam[20];

char name[15];

};

struct Student

{

int grup;

int mark;

struct date birthday;

struct Person man;

};

void main()

{

Student ob;

printf("Name: ");

scanf("%s",ob.man.name);

printf("Family: ");

scanf("%s",ob.man.fam);

printf("Day: ");

scanf("%d", &ob.birthday.day);

printf("Month: ");

scanf("%d",&ob.birthday.month);

printf("Year: ");

scanf("%d",&ob.birthday.year);

printf("Mark: ");

scanf("%d",&ob.mark);

printf("Gruppa: ");

scanf("%d",&ob.grup);

printf("Student is gruppi %d \n%s \t%s \n%d.%d.%d\n Mark:%d \n", ob.grup, ob.man.fam, ob.man.name,ob.birthday.day,ob.birthday.month,ob.birthday.year,ob.mark);

getch();

}

Пример использования указателей на структуры.

3. Осуществить доступ к полям структуры через указатель. Вввод/вывод фамилии и имени человека.

#include<stdio.h>

#include <conio.h>

#include<stdio.h>

struct Person

{

char fam[20];

char name[15];

};

void main()

{

Person ob1;

Person \*p;

p=&ob1;

printf("Name: ");

gets(p->name);

printf("Familya: ");

gets(p->fam);

printf("%s\t%s\n",ob1.fam,ob1.name);

getch();

}

Можно использовать оператор присвоения к структурам, при этом копируются значения одной структуры переменной в другую переменную, если обе структуры-переменные относятся к одному и тому же типу. Однако в этом случае происходит поверхностное копирование, т.е. копирует бит за битом и могут возникнуть проблемы с такими членами данных, как указатели.

**#include<stdio.h>**

**#include<string.h>**

**struct Person**

**{**

**char \*fam;**

**char \*name;**

**};**

**void main()**

**{**

**Person ob1,ob2;**

**char x[20];**

**int l;**

**printf("Name: ");**

**scanf("%s",x);**

**l=strlen(x)+1;**

**ob1.name= new char[l];**

**if(!ob1.fam)**

**{**

**printf("\a\a\a\a\n");**

**return;**

**}**

**strcpy(ob1.name,x);**

**printf("Family: ");**

**scanf("%s",x);**

**l=strlen(x)+1;**

**ob1.fam= new char[l];**

**if(!ob1.fam)**

**{**

**printf("\a\a\a\a\n");**

**return;**

**}**

**strcpy(ob1.fam,x);**

**ob2=ob1;**

**printf("%s\t%s\n%s\t%s\n",ob1.fam,ob1.name,ob2.fam,ob2.name);//здесь возможна //ошибка**

**}**

Структуру как параметр можно передавать функции, компилятор передает данные всей структуры в функцию со всеми полями которые в нее входят, а так же и быть возвращающим значением функции

**#include<stdio.h>**

**struct Man**

**{**

**char fam[20];**

**int tel;**

**};**

**struct Man func1(struct Man);**

**void func2(struct Man \*p);**

**void main()**

**{**

**struct Man ob;**

**puts("Vizov func1 ili func2 - ? ");**

**fflush(stdin);**

**switch(getchar())**

**{**

**case '1': ob=func1(ob);**

**break;**

**case '2': func2(&ob);**

**break;**

**}**

**printf("sotrudnik - %s tel.- %d\n",ob.fam,ob.tel);**

**}**

**struct Man func1(struct Man ob1)**

**{**

**puts("Familiya:");**

**fflush(stdin);**

**gets(ob1.fam);**

**puts("Telefon:");**

**fflush(stdin);**

**scanf("%d",&ob1.tel);**

**return ob1;**

**}**

**void func2(struct Man \*p)**

**{**

**puts("Familiya:");**

**fflush(stdin);**

**gets(p->fam);**

**puts("Telefon:");**

**fflush(stdin);**

**scanf("%d",&p->tel);**

**}**

Число байтов, выделенных под структуру не всегда равна сумме длин отдельных ее элементов из-за некоторых особенностей работы процессора с данными с фиксированной и плавающей точкой, что приводит к так называемому “выравниванию”, размещению элементов с четного адреса. Выделение памяти осуществляется в последовательности объявления данных. При размещении структурной переменной в памяти для выравнивания на границу слова компилятор оставляет между ее элементами и элементами массива структурных переменных пустые байты для соблюдения следующих правил:

* Структурная переменная (элемент массива структур) начинается на границе слова, т.е. с четного адреса;
* Любой элемент, кроме элементов типа char, так же располагаются с четного адреса и соответственно имеет четное смещение от начала структурной переменной;
* При необходимости в конце структурной переменной добавляется неиспользуемый байт для того, чтобы общее число байтов, занимаемых переменной, было четным.

Между структурами и элементами структуры могут быть неиспользованные байты памяти.

**Задачи:**

1. Создать массив данных о студентах группы. О каждом студенте записать имя, фамилию, год рождения, оценки по 5-ти экзаменам и определить средний балл оценок.

**#include<stdio.h>**

**struct Student**

**{**

**char fam[20],name[20];**

**int year,marks[5],average;**

**};**

**void func\_vvoda(Student \*);**

**void func\_vivoda(Student );**

**void main()**

**{**

**Student ob[10];**

**for(int i=0;i<10;i++)**

**func\_vvoda(&ob[i]);**

**for(i=0;i<10;i++)**

**func\_vivoda(ob[i]);**

**}**

**void func\_vivoda(Student ob1)**

**{**

**puts("");**

**printf("%s\t%s\t%d\t%d\n",ob1.fam,ob1.name,ob1.year,ob1.average);**

**}**

**void func\_vvoda(Student \*p)**

**{**

**int sum=0;**

**puts("Familiya:");**

**fflush(stdin);**

**gets(p->fam);**

**puts("Name:");**

**fflush(stdin);**

**gets(p->name);**

**puts("5 marks:");**

**fflush(stdin);**

**for(int i=0;i<5;i++)**

**{**

**scanf("%d",&p->marks[i]);**

**sum+=p->marks[i];**

**}**

**p->average=sum/5;**

**puts("Year:");**

**fflush(stdin);**

**scanf("%d",&p->year);**

**}**

1. Создать массив данных о студентах. О каждом студенте записать имя, фамилию, группу, оценки по 5-ти экзаменам и определить средний балл оценок. И распечатать имя, фамилию и группу всех тех студентов, кто получил максимальный средний балл.

Аналогично, что и в предыдущей задаче

**#include<stdio.h>**

**struct Student**

**{**

**char fam[20],name[20];**

**int group,marks[5],average;**

**};**

**void func\_vvoda(Student \*);**

**void func\_vivoda(Student );**

**int func\_max(Student ob1[],int n);**

**void main()**

**{**

**Student ob[4];**

**int max;**

**for(int i=0;i<4;i++)**

**func\_vvoda(&ob[i]);**

**max=func\_max(ob,4);**

**for(i=0;i<4;i++)**

**if(ob[i].average==max)**

**func\_vivoda(ob[i]);**

**}**

**int func\_max(Student ob1[],int n)**

**{**

**int max=ob1[0].average;**

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

**if(max<ob1[i].average)**

**max=ob1[i].average;**

**return max;**

**}**

1. Создать массив данных о студентах. О каждом студенте записать имя, фамилию, год рождения, оценки по 5-ти экзаменам и определить сумму оценок. И отсортировать список по сумме баллов.
2. Ввести массив структур, используя динамическое распределение памяти. Рассортировать массив в алфавитном порядке.

**#include<stdio.h>**

**#include<string.h>**

**#include<malloc.h>**

**struct Student**

**{**

**char name[80];**

**int age;**

**};**

**void main()**

**{**

**struct Student \*p,ob;**

**int i,j,n;**

**do**

**{**

**printf("kol. studentov: ");**

**scanf("%d",&n);**

**}while(n<=0);**

**p=(Student \*)malloc(n\*sizeof(Student));**

**if(!p)**

**{**

**printf("Net mesta\a\a\a\n");**

**return;**

**}**

**for(i=0;i<n;i++)**

**{**

**printf("Name: ");**

**scanf("%s",p[i].name);**

**printf("Age: ");**

**scanf("%d",&p[i].age);**

**}**

**for(i=0;i<n-1;i++)**

**{**

**for(j=i+1;j<n;j++)**

**if(strcmp(p[i].name,p[j].name)>0)**

**{**

**ob=p[i];**

**p[i]=p[j];**

**p[j]=ob;**

**}**

**}**

**for(i=0;i<n;i++)**

**{**

**printf("\n%s\t\t%d\n",p[i].name,p[i].age);**

**}**

**}**

1. Для массива структур, состоящих из полей: указатели на две структуры 1-я студент и номер группы, 2-я 4 оценки, используя указатели организовать поиск и вывод на экран информации о студентах с max суммой баллов.

**#include<stdio.h>**

**#include<malloc.h>**

**#include<stdlib.h>**

**struct student**

**{**

**char name[20];**

**char fam[30];**

**int group;**

**};**

**struct mark**

**{**

**int marks[4];**

**};**

**struct Man**

**{**

**student \*p;**

**mark \*p1;**

**};**

**void vvod(Man \*);**

**int max\_ball(Man \*,int);**

**void vivod(Man \*,int ,int);**

**void main()**

**{**

**const int n=10;**

**int max;**

**Man p2[n];**

**for(int i=0;i<n;i++)**

**vvod(&p2[i]);**

**max=max\_ball(&p2[0],n);**

**printf("Studenti s max ballom:\n");**

**vivod(&p2[0],n,max);**

**}**

**void vivod(Man \*f,int k,int max)**

**{**

**int sum;**

**for(int i=0;i<k;i++)**

**{**

**sum=(f+i)->p1->marks[0]+(f+i)->p1->marks[1]+(f+i)->p1->marks[2]+(f+i)->p1->marks[3];**

**if(sum= =max)**

**printf("%s\t%s - %d\n",(f+i)->p->fam,(f+i)->p->name,(f+i)->p->group);**

**}**

**}**

**int max\_ball(Man \*f,int k)**

**{**

**int max;**

**int \*n1;**

**n1=(int \*)malloc(k\*sizeof(int));**

**if(!n1)**

**{**

**printf("\a\a\a\a\n");**

**exit(1);**

**}**

**for(int i=0;i<k;i++)**

**{**

**\*(n1+i)=0;**

**for(int j=0;j<4;j++)**

**\*(n1+i)+=(f+i)->p1->marks[j];**

**}**

**max=n1[0];**

**for(i=1;i<k;i++)**

**{**

**if(max<n1[i])**

**max=n1[i];**

**}**

**free(n1);**

**return max;**

**}**

**void vvod(Man \*f)**

**{**

**f->p=(student \*)malloc(sizeof(student));**

**f->p1=(mark \*)malloc(sizeof(mark));**

**printf("Name: ");**

**scanf("%s",f->p->name);**

**printf("Familiya: ");**

**scanf("%s",f->p->fam);**

**printf("Gruppa: ");**

**scanf("%d",&f->p->group);**

**printf("4 ocenki: ");**

**for(int j=0;j<4;j++)**

**scanf("%d",&f->p1->marks[j]);**

**}**

6. Используя объединение полями которого являются: имя фигуры, структура (содержащая параметры геом. фигур – прямоугольник, трапеция) и переменная площадь. Написать программу, которая вычисляет площади 10 фигур и выводит на экран наименование – площадь. Использовать функции для ввода данных, посчитать площадь и для распечатки данных.

**#include<stdio.h>**

**#include<string.h>**

**struct prm**

**{**

**double dlina,shirina;**

**};**

**struct trap**

**{**

**double visota,osnavanie;**

**};**

**struct FIGURA**

**{**

**char name[15];**

**double ploshad;**

**prm f1;**

**trap f2;**

**};**

**void vvod(FIGURA \*);**

**void pl(FIGURA \*,int);**

**void vivod(FIGURA \*,int);**

**void main()**

**{**

**const int n=10;**

**FIGURA ob[n];**

**int i;**

**for(i=0;i<n;i++)**

**{**

**printf("VVrdite dannie %d fuguri\n",i+1);**

**vvod(&ob[i]);**

**}**

**pl(&ob[0],n);**

**vivod(&ob[0],n);**

**}**

**void vivod(FIGURA \*f,int k)**

**{**

**puts("");**

**for(int i=0;i<k;i++)**

**{**

**printf("%s - %.1lf\n",(f+i)->name,(f+i)->ploshad);**

**}**

**}**

**void pl(FIGURA \*f,int k)**

**{**

**for(int i=0;i<k;i++)**

**{**

**if(!strcmp((f+i)->name,"PRYAMOUGOLNIK"))**

**(f+i)->ploshad=(f+i)->f1.dlina\*(f+i)->f1.shirina;**

**else**

**(f+i)->ploshad=(f+i)->f2.osnavanie\*(f+i)->f2.visota;**

**}**

**}**

**void vvod(FIGURA \*f)**

**{**

**char ch[3];**

**printf("Введите имя фигуры:pr–если ПРЯМОУГОЛЬНИК, и tr-если ТРАПЕЦИЯ :");**

**scanf("%s",ch);**

**if(!strcmp(ch,"pr"))**

**{**

**strcpy(f->name,"PRYAMOUGOLNIK");**

**printf("DLINA: ");**

**scanf("%lf",&(f->f1.dlina));**

**printf("SHIRINA: ");**

**scanf("%lf",&(f->f1.shirina));**

**}**

**else**

**{**

**strcpy(f->name,"TRAPECIYA");**

**printf("VISOTA: ");**

**scanf("%lf",&(f->f2.visota));**

**printf("OSNOVANIE: ");**

**scanf("%lf",&(f->f2.osnavanie));**

**}**

**}**

7. Создать массив структур. Элементами структуры являются: фамилия автора книги и ее наименование. Написать функции 1) добавления данных; 2) удаления указанного автора; 3) вывод данных.

**#include<stdio.h>**

**#include<string.h>**

**struct book**

**{**

**char fam[30];**

**char naz[10];**

**};**

**void mas\_str(book\*,book,int\*);**

**void delet(book\*,char \*,int \*);**

**void show\_str(book\*);**

**void main()**

**{**

**book katal[10],bk;**

**int i,kol=0,pr;**

**char fio[30];**

**printf("Enter:\n1 - DOBAVLENUE\n2 - Prosmotr\n3 - UDALENIE\n4 - exit\n");**

**while(1)**

**{**

**puts("pr-?");**

**scanf("%d",&pr);**

**switch(pr)**

**{**

**case 1:**

**do**

**{**

**fflush(stdin);**

**printf("FAM: ");**

**gets(bk.fam);**

**if(strcmp(bk.fam,""))**

**{**

**printf("KNIGA: ");**

**gets(bk.naz);**

**mas\_str(katal,bk,&kol);**

**}**

**}while(strcmp(bk.fam,""));**

**break;**

**case 2:**

**if(kol)**

**for(i=0;i<kol;i++)**

**puts(katal[i].fam);**

**else**

**puts("PUST!!!!");**

**break;**

**case 3:**

**puts("FAM UDALENIE:");**

**fflush(stdin);**

**gets(fio);**

**if(kol)**

**delet(katal,fio,&kol);**

**else**

**puts("PUST VVV!!!");**

**break;**

**case 4:**

**return;**

**}**

**}**

**}**

**void mas\_str(book\* katal,book bk,int\*kol)**

**{**

**int i=\*kol;**

**while((\*kol)<10 && i<(\*kol) && strcmp(katal[i].fam,"") && (!strcmp(katal[i].fam,bk.fam)) && strcmp(katal[i].naz,bk.naz))**

**i++;**

**katal[i]=bk;**

**(\*kol)++;**

**}**

**void delet(book\* katal,char \*fio,int \*kol)**

**{**

**int i=-1,k=\*kol;**

**if(\*kol)**

**{**

**while(i++,i<k)**

**if(!strcmp(katal[i].fam,fio))break;**

**if(!strcmp(katal[i].fam,fio))**

**{**

**if(i==k)**

**{**

**\*kol=0;return;**

**}**

**if(i<k)**

**{**

**while(++i,i<\*kol)**

**katal[i-1]=katal[i];**

**(\*kol)--;**

**}**

**else**

**printf("NET!!!\n");**

**}**

**}**

**}**

Ключевое слово **typdef** предоставляет программисту механизм для создании синонимов (или псевдокодов) для ранее определенного типа. Объявление нового имени с помощью **typedef** не создает нового типа, а создает лишь имя, являющееся синонимом для определенного ранее типа.

**Моделирование высокоэффективной тасовки и раздачи карт**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<time.h>**

**struct card{**

**char \*face;**

**char \*suit;**

**};**

**typedef struct card Card;**

**void fillDeck(Card \*,char \*[],char \*[]);**

**void shuffle(Card \*);**

**void deal(Card \*);**

**void main()**

**{**

**Card deck[52];**

**char \*face[]={"Tuz", "2", "3", "4", "5",**

**"6", "7", "8", "9", "10", "Valet", "Dama","Korol"};**

**char \*suit[]={"Chervi", "Bubni", "Trefi", "Piki"};**

**srand(time(NULL));**

**fillDeck(deck,face,suit);**

**shuffle(deck);**

**deal(deck);**

**}**

**void fillDeck(Card \*wDeck,char \*wFace[],char \*wSuit[])**

**{**

**int i;**

**for(i=0;i<52;i++)**

**{**

**wDeck[i].face = wFace[i%13];**

**wDeck[i].suit = wSuit[i/13];**

**}**

**}**

**void shuffle(Card \*wDeck)**

**{**

**int i,j;**

**Card temp;**

**for(i=0;i<52;i++)**

**{**

**j=rand()%52;**

**temp=wDeck[i];**

**wDeck[j]=temp;**

**}**

**}**

**void deal(Card \*wDeck)**

**{**

**int i;**

**for(i=0;i<52;i++)**

**{**

**printf("%5s of %-8s%c",wDeck[i].face,wDeck[i].suit, (i+1)%2 ? '\t' : '\n');**

**}**

**}**

# 7.2 Объединения

*Объединение –* произвольный тип данных, подобный структуре, элементы которого разделяют одну и ту же область памяти. На различных этапах выполнение программы одни переменные могут оказаться невостребованными, в то время как другие, наоборот, используются только в этой части программы, поэтому объединения экономят пространство вместо того, чтобы впустую тратить память на не использующиеся в данный момент переменные. Элементы объединения могут принадлежать любому типу. Число байтов, используемая для хранения объединения, должно быть, по крайней мере, достаточным для хранения наибольшего из элементов. Ссылаться в данный момент времени можно только на один элемент и, соответственно, только один тип данных. Объединение объявляется с помощью ключевого слова **union**. Формат объединения тот же, что и в случае структуры.

**#include**<stdio.h>

**union** number{

**in**t x;

**float** y;

};

**void** main()

{

**union** number value;

value.x=100;

printf("Put a value in the integer member\nand printf both members.\nint: %d\nfloat: %f",value.x,value.y);

value.y=100.0;

printf("\n\nPut a value in the floating member\nand printf both members.\nint: %d\nfloat: %f",value.x,value.y);

}

Допускается определение *неименованного битого поля;* такое поле используется как заполнитель структуры и служит для выравнивания следующего битого поля по границе новой ячейки памяти. Например:

**struct** example

{

**unsigned** a : 13;

**unsigned** : 0;

**unsigned** b : 4;

};

объявление битовое поле без имени длиной 0 бит для того, чтобы перескочить через оставшиеся биты (столько, сколько их будет) ячейки памяти, в которой хранится **а**, и выровнять **b** по границе следующей ячейки.

Пример моделирование высокоэффективной тасовки и раздачи карт с разъясняющий тип объединения:

**#include<stdio.h>**

**struct bitCard{**

**unsigned face : 4;**

**unsigned suit : 2;**

**unsigned color : 1;**

**};**

**typedef struct bitCard Card;**

**void fillDeck(Card \*);**

**void deal(Card \*);**

**void main(){**

**Card deck[52];**

**fillDeck(deck);**

**deal(deck);**

**}**

**void fillDeck(Card \*wDeck){**

**int i;**

**for(i=0;i<52;i++)**

**{**

**wDeck[i].face = i%13;**

**wDeck[i].suit = i/13;**

**wDeck[i].color= i/26;**

**}**

**}**

**void deal(Card \*wDeck){**

**int k1,k2;**

**for(k1=0,k2=k1+26;k1<26;k1++,k2++)**

**{**

**printf("Card:%3d Suit:%2d Color:%2d ",wDeck[k1].face,wDeck[k1].suit, wDeck[k1].color);**

**printf("Card:%3d Suit:%2d Color:%2d\n",wDeck[k2].face,wDeck[k2].suit, wDeck[k2].color);**

**}**

**}**

§8 ФАЙЛЫ И ПОТОКИ

# 8.1 Объявления файла

Файл – это набор данных, размещенный на внешнем носителе и рассматриваемый в процессе обработки как единое целое. В файлах размещаются данные, предназначенные для длительного хранения.

Различают два вида файлов: текстовые и бинарные. Текстовые файлы представляют собой последовательность ASCII символов и могут быть просмотрены и отредактированы с помощью любого текстового редактора.

Бинарные (двоичные) файлы представляют собой последовательность данных, структура которых определяется программно.

В языке Си имеется большой набор функций для работы с файлами, большинство которых находятся в библиотеке stdio.h

В языке С любой файл рассматривается как последовательный поток байтов. Каждый файл оканчивается или м*аркером конца файла,* или особым байтом, определенным в работающей файлами программе. В начале исполнения программы автоматически открываются три файла и связанные с ними потоки – *стандартный ввод* (указатель файла **stdin**)*,стандартный вывод* (указатель файла **stdout**)*, и стандартный ошибка*(указатель файла **stderr**)*.*

Если хотим вывести строку на экран результатов, то можно написать следующим образом:

**fputs(const char\*, stdout); -** это тоже, что написать **puts(const char\*);**

Если хотим с клавиатуры ввести строку, то можно написать следующим образом:

**fgets(char \*, int, stdin); -** это тоже, что написать **gets(const char\*);**

**fprintf(stdout, const char\*,…); -** это тоже, что **printf(const char\*,…);**

**fscan(stdin, const char\*,…); -** это тоже, что**scan(const char\*,…);**

*Файл* – это именованный объект, хранящий данные, на каком либо носителе (дискета, винчестер, CD). В переменных и массивах данные хранятся временно. Файлы предназначены для постоянного хранения больших объемов данных Файлы, в отличии от массивов, могут располагаться не только в оперативной памяти (электронном диске), но и на жестких или на внешних носителях. Файлы не имеют фиксированной длины, т.е. могут увеличиваться или уменьшаться. Перед работой с файлом его необходимо открыть, а после работы закрыть.

Различаются два вида файлов: *текстовые и бинарные*. *Текстовый файл* может быть просмотрен и отредактирован с клавиатуры любым текстовым редактором и имеет простую структуру, которая состоит из последовательности символов. Эта последовательность символов можно разбить на строки, каждая строка заканчивается двумя кодами: 10 (0хD) и 13 (0xA). *Бинарный файл* – это файл, который не имеет структуру текстового файла. Каждая программа для своего бинарного файла определяет свою структуру. Существуют функции для работы как с текстовыми, так и с бинарными файлами.

Для работы с файлами необходимо включить библиотеку <stdio.h>, где находятся прототипы функций работы с файлами. Для открытия файла используют функцию fopen().

**FILE \* fopen(char \*filename, char \* mode);**

Возвращаемое значение (FILE \*) – это указатель на структуру типа FILE, который называется *указатель на файл*. Возвращаемое значение нужно хранить: во-первых, для проверки открытия файла, т.е. если возвращаемое значение равно NULL, то произошла ошибка при открытии и следовательно нужно остановить работу программы; во-вторых для работы с файлом в программе.

**char** \*filename задает путь, т.е. местонахождение файла и имя открываемого файла.

**char** \*mode – это тип доступа к файлу, который может принимать следующие значения:

|  |  |
| --- | --- |
| “rt” | Открыть текстовый файл для чтения |
| “wt” | Открыть текстовый файл для записи, если файл существует, то его содержание теряется |
| “at” | Открыть текстовый файл для записи в конец файла, если файл существует. Если файл не существует, то он создается |
| “rt+” | Открыть текстовый файл для записи и чтения. Файл должен существовать. |
| “wt+” | Открыть текстовый файл для записи и чтения. Если файл существует, то его содержание теряется |
| “at+” | Открыть текстовый файл для записи в конец файла и для чтения, если файл существует. Если файл не существует, то он создается |
| “rb” | Открыть бинарного файл для чтения |
| “wb” | Открыть бинарного файл для записи, если файл существует, то его содержание теряется |
| “ab” | Открыть бинарного файл для записи в конец файла, если файл существует. Если файл не существует, то он создается |
| “rb+” | Открыть бинарного файл для записи и чтения. Файл должен существовать. |
| “wb+” | Открыть бинарного файл для записи и чтения. Если файл существует, то его содержание теряется |
| “ab+” | Открыть бинарного файл для записи в конец файла, если файл существует. Если файл не существует, то он создается |

Т.е. “t” для текстового режима, а “b” для бинарного. Если режим не указан, то автоматически берется текстовый.

После работы с файлом его необходимо обязательно закрыть с помощью функции **fclose(FILE \*)**, параметром функции является указатель на файл, который был получен при открытии файла**.**

Например:

FILE \*k;

k=fopen(“aaa.bbb”,”w”);

if(!k)

{

printf(“Открытие файла невозможно!!!!\n”);

return;

}

………..

………..

fclose(k);

**int feof(FILE \*);**

функция feof() возвращает ненулевое значение в том и только в том случае, если индикатор конца файла для этого потока показывает конец файла.

# 8.2 Работа с текстовым файлом

Для работы с текстовым файлом наиболее часто используются следующие функции:

**int fprintf(FILE \*, const char format, …);**

Функция fprintf() выводит данные в файл на который указывает заданный указатель FILE \*; const char format задает способ преобразование заданных аргументов при выводе. Функция fprinf() заканчивает свою работу, как только достигает символа конца файла.

Пример:

**int x;**

**FILE \*k;**

**k=fopen(“aaa.bbb”,”w”);**

**if(!k)**

**{**

**printf(“Открытие файла невозможно!!!!\n”);**

**return;**

**}**

**scanf(“%d ”,&x);**

**fprintf(k,”%d”,x); т.е. целое число записанное в х преобразовать в символ и записать в файл на который указывает k,**

**fclose(k);**

**Пример создания последовательного файла:**

Записываем в текстовый файл номер имя и баланс клиента. Ввод данных прекращается, когда введем сочетание клавиш <Ctrl+z>, что в системе IBMPC означает конец файла, т.е. EOF.

**#include<stdio.h>**

**void main()**

**{**

**int account;**

**char name[30];**

**double balance;**

**FILE \*cfPtr;**

**if((cfPtr=fopen("clients.dat","w"))== NULL)**

**{**

**printf("File could not be opened\n");**

**return;**

**}**

**printf("Enter the account, name and balance.\n");**

**printf("Enter EOF to end input.\n");// в системе IBM PC EOF–это сочетание клавиш <ctrl+z>**

**printf("? ");**

**scanf("%d%s%lf",&account,name,&balance);**

**while(!feof(stdin))**

**{**

**fprintf(cfPtr,"%d %s %.2lf\n",account,name,balance);**

**printf("? ");**

**scanf("%d%s%lf",&account,name,&balance);**

**}**

**fclose(cfPtr);**

**}**

**int fputs(const char \*, FILE \*);**

Возвращает положительное число, а если произошла ошибка, то возвращает EOF. Функция fputs() выводит строку, которая задается constchar без символа конец строки в файл на который указывает указатель заданный FILE \*.

Пример:

**char s[100];**

**FILE \*k;**

**k=fopen(“aaa.bbb”,”w”);**

**if(!k)**

**{**

**printf(“Открытие файла невозможно!!!!\n”);**

**return;**

**}**

**gets(s);**

**fputs(s,k); т.е. строка s записывается в файл на который указывает k,**

**fclose(k);**

**int fscanf( FILE \*, const char \*format, …..);**

Функция fscanf() вводит данные из файла, на который указывает заданный указатель FILE \*; const char format задает способ преобразование заданных аргументов при выводе. Функция fprinf() заканчивает свою работу, как только достигает символа конца файла. Возвращает положительное число, а если произошла ошибка, то возвращает EOF.

int x;

FILE \*k;

k=fopen(“aaa.bbb”,”r”);

if(!k)

{

printf(“Открытие файла невозможно!!!!\n”);

return;

}

fscanf(k,”%d”,&x); т.е. преобразовать символ из файл на который указывает k в целое число и записанное в х

fclose(k);

**Программа, которая считывает числа с заранее созданного файла.**

**#include<stdio.h>**

**void main()**

**{**

**FILE \*k;**

**int x;**

**k=fopen("aaa.bbb","r");**

**if(!k)**

**{**

**printf("\a\a\a\n");**

**return;**

**}**

**fscanf(k,"%d",&x);**

**while(!feof(k))**

**{**

**printf("%d ",x);**

**fscanf(k,"%d",&x);**

**}**

**fclose(k);**

**}**

**Чтение и распечатка последовательного файла**

Программа, которая считывает данные из заранее созданного файла.

**#include<stdio.h>**

**void main()**

**{**

**int account;**

**char name[30];**

**double balance;**

**FILE \*cfPtr;**

**if((cfPtr=fopen("clients.dat","r"))== NULL)**

**{**

**printf("File could not be opened\n");**

**return;**

**}**

**printf("%-10s%-13s%-17s\n","Account","Name", "Balance");**

**fscanf(cfPtr,"%d%s%lf",&account,name,&balance);**

**while(!feof(cfPtr))**

**{**

**printf("%-10d%-13s%7.2lf\n",account,name , balance);**

**fscanf(cfPtr,"%d%s%lf",&account,name,&balance);**

**}**

**fclose(cfPtr);}**

**char \* fgets(char \*,int , FILE \*);**

функция fgets() считывает из файла на единицу меньше символов указанных с помощью int или как только встречает символ новой строки или символ конец файла.

**Пример, когда считываются строки из файла.**

**#include<stdio.h>**

**void main()**

**{**

**FILE \*k;**

**char s[200];**

**k=fopen("aaa.bbb","r");**

**if(!k)**

**{**

**printf("\a\a\a\n");**

**return;**

**}**

**while(!feof(k))**

**{**

**fgets(s,100,k);**

**puts(s);**

**}**

**fclose(k);**

**}**

**int fseek(FILE \*,long ,int );**

Первый параметр- это указатель на файл; следующие параметр задает количество байтов, на которые необходимо сместить указатель в направлении указанном последним параметром.

Последним параметром могут быть:

**SEEK\_SET 0** Смещение выполняемся от начало файла

**SEEK\_CUR 1** Смещение выполняемся от текущей позиции

**SEEK\_END 1** Смещение выполняемся от конца файла

Функция возвращает отличное от нуля результат только для запроса, который не может быть выполнен.

Для текстового файла второй параметр функции **fseek** должно быть равно нулю, либо значению которое вернула функция **ftell**, а последний параметр должен быть **SEEK\_SET**

**long int ftell(FILE \*);**

Функция возвращает значение индикатора текущей позиции в файле.

**Задачи:**

1. Написать программу, в которой создаются два файла, для записи 10 целых чисел.
2. Используя первую задачу, в третий файл переписать поочередно числа из заранее созданных файлов.

**#include<stdio.h>**

**void main()**

**{**

**FILE \*k1,\*k2,\*k3;**

**int x;**

**k1=fopen("aaa.txt","w+");**

**if(!k1)**

**{**

**puts("FILEa ne najden!!!");**

**return;**

**}**

**k2=fopen("bbb.txt","w+");**

**if(!k2)**

**{**

**puts("FILE b ne najden!!!");**

**return;**

**}**

**k3=fopen("ccc.txt","w");**

**if(!k3)**

**{**

**puts("FILE c ne najden!!!");**

**return;**

**}**

**for(int i=1;i<=10;i++)**

**{**

**fprintf(k1,"%d ",i);**

**}**

**for(i=11;i<=20;i++)**

**{**

**fprintf(k2,"%d ",i);**

**}**

**fseek(k1,0,0);**

**fseek(k2,0,0);**

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

**{**

**fscanf(k1,"%d",&x);**

**fprintf(k3,"%d ",x);**

**fscanf(k2,"%d",&x);**

**fprintf(k3,"%d ",x);**

**}**

**fclose(k1);**

**fclose(k2);**

**fclose(k3);**

**}**

1. Сначала создать 2 файла для записи, потом 3-ий файл для записи из этих 2-х файлов, причем длина файлов может быть не одинаковой.

**#include<stdio.h>**

**void main()**

**{**

**FILE \*k1,\*k2,\*k3;**

**int x,y;**

**k1=fopen("aaa.txt","w+");**

**if(!k1)**

**{**

**puts("FILEa ne najden!!!");**

**return;**

**}**

**k2=fopen("bbb.txt","w+");**

**if(!k2)**

**{**

**puts("FILEb ne najden!!!");**

**return;**

**}**

**k3=fopen("ccc.txt","w");**

**if(!k3){**

**puts("FILEc ne najden!!!");**

**return;**

**}**

**puts("Enter: I-ij FILE");**

**do{**

**scanf("%d",&x);**

**fprintf(k1,"%d ",x);**

**}while(x);**

**puts("Enter: II-oj FILE");**

**do{**

**scanf("%d",&x);**

**fprintf(k2,"%d ",x);**

**}while(x);**

**fseek(k1,0,0);**

**fseek(k2,0,0);**

**fscanf(k1,"%d",&x);**

**fscanf(k2,"%d",&y);**

**while(!feof(k1) && !feof(k2)){**

**fprintf(k3,"%d ",x);**

**fprintf(k3,"%d ",y);**

**fscanf(k1,"%d",&x);**

**fscanf(k2,"%d",&y);**

**}**

**if(feof(k1)){**

**while(!feof(k2)){**

**fprintf(k3,"%d ",y);**

**fscanf(k2,"%d",&y);**

**}**

**}**

**if(feof(k2)){**

**while(!feof(k1)){**

**fprintf(k3,"%d ",x);**

**fscanf(k1,"%d",&x);**

**}**

**}**

**fclose(k1);**

**fclose(k2);**

**fclose(k3);**

**}**

Задачи:

1. Создать файл для записи текста. text.txt

**#include<stdio.h>**

**void main()**

**{**

**FILE \*k;**

**char x[200];**

**k=fopen("text.txt","w");**

**if(!k){**

**printf("\a\a\a\n");**

**return;**

**}**

**printf("Введите stroku: vved prekrachaetsya pri vvode 0\n");**

**gets(x);**

**while(x[0]!='0'){**

**fputs(x,k);**

**fputs(“\n”,k);**

**gets(x);**

**}**

**fclose(k);**

**}**

1. Посчитать количество лишних пробелов в тексте, записанном в файле text.txt.

**#include<stdio.h>**

**void main()**

**{**

**FILE \*k;**

**char ch;**

**int kol=0;**

**k=fopen("text.txt","r");**

**if(!k){**

**printf("\a\a\a\n");**

**return;**

**}**

**fscanf(k,"%c",&ch);**

**while(!feof(k)){**

**if(ch==' '){**

**fscanf(k,"%c",&ch);**

**while(ch==' '){**

**kol++;**

**fscanf(k,"%c",&ch);**

**}**

**}**

**fscanf(k,"%c",&ch);**

**}**

**printf("Kol ' ' = %d\n",kol);**

**fclose(k);**

**}**

1. Переписать из файла text.txt в файл text1.txt без лишних пробелов.

**#include<stdio.h>**

**void main()**

**{**

**FILE \*k,\*k1;**

**char ch;**

**int kol=0;**

**k=fopen("text.txt","r");**

**if(!k){**

**printf("\a\a\a\n");**

**return;**

**}**

**k1=fopen("text1.txt","w");**

**if(!k1){**

**printf("\a\a\a\n");**

**return;**

**}**

**fscanf(k,"%c",&ch);**

**while(!feof(k)){**

**if(ch==' '){**

**fprintf(k1,"%c",ch);**

**fscanf(k,"%c",&ch);**

**while(ch==' '){**

**kol++;**

**fscanf(k,"%c",&ch);**

**}**

**}**

**fprintf(k1,"%c",ch);**

**fscanf(k,"%c",&ch);**

**}**

**printf("Kol ' ' = %d\n",kol);**

**fclose(k);**

**fclose(k1);**

**}**

4. Файл, где хранятся данные о студенте. Функции создания файла и добавления в файл.

**#include<stdio.h>**

**#include<stdlib.h>**

**struct Student{**

**char fam[20];**

**int marks[4];**

**float sr\_ball;**

**};**

**char name1[]="student1.txt";**

**void fill\_file(FILE \*);**

**void dobav\_file(FILE \*);**

**void show\_file(FILE \*);**

**void main()**

**{**

**int kod;**

**FILE \*k1;**

**k1=fopen(name1,"r+t");**

**if(!k1){**

**puts("File student1 ne otkrilsay!!!");**

**exit(0);**

**}**

**puts("Enter:\n1 - Sozdanie");**

**puts("2 - Dobavlenie");**

**puts("3 - Prosmotr");**

**puts("0 - exit");**

**while(1){**

**scanf("%d",&kod);**

**switch(kod)**

**{**

**case 1:**

**fill\_file(k1);**

**break;**

**case 2:**

**dobav\_file(k1);**

**break;**

**case 3:**

**show\_file(k1);**

**break;**

**case 0:**

**return;**

**}**

**}**

**fclose(k1);**

**}**

**void show\_file(FILE \*k1)**

**{**

**char x[100];**

**Student ob;**

**k1=fopen(name1,"rt");**

**if(!k1){**

**puts("File student\_t nt otkrilsay!!!");**

**exit(0);**

**}**

**fgets(x,100,k1);**

**puts(x);**

**fscanf(k1,"%s%d%d%d%d%f", ob.fam, &ob.marks[0], &ob.marks[1], &ob.marks[2], &ob.marks[3], &ob.sr\_ball);**

**while(!feof(k1))**

**{**

**printf("%-20s%2d%2d%2d%2d%9.2f\n", ob.fam, ob.marks[0], ob.marks[1], ob.marks[2], ob.marks[3], ob.sr\_ball);**

**fscanf(k1,"%s%d%d%d%d%f", ob.fam, &ob.marks[0], &ob.marks[1], &ob.marks[2], &ob.marks[3], &ob.sr\_ball);**

**}**

**fclose(k1);**

**}**

**void dobav\_file(FILE \*k1)**

**{**

**Student ob;**

**k1=fopen(name1,"at");**

**if(!k1){**

**puts("File student\_t nt otkrilsay!!!");**

**exit(0);**

**}**

**puts("Fam: Vvod zavesshaetsya - 0");**

**fflush(stdin);**

**scanf("%s",ob.fam);**

**while(ob.fam[0]!='0'){**

**puts("Marks:");**

**fflush(stdin);**

**ob.sr\_ball=0.0;**

**for(int i=0;i<4;i++){**

**scanf("%d",&ob.marks[i]);**

**ob.sr\_ball+=ob.marks[i];**

**}**

**ob.sr\_ball/=4;**

**fprintf(k1,"%-20s%2d%2d%2d%2d%9.2f\n",ob.fam,ob.marks[0],ob.marks[1],ob.marks[2],ob.marks[3],ob.sr\_ball);**

**puts("Fam: Vvod zavesshaetsya - 0");**

**fflush(stdin);**

**scanf("%s",ob.fam);**

**}**

**fclose(k1);**

**}**

**void fill\_file(FILE \*k1)**

**{**

**Student ob;**

**k1=fopen(name1,"wt");**

**if(!k1){**

**puts("File student\_t nt otkrilsay!!!");**

**exit(0);**

**}**

**fprintf(k1,"%-21s%-11s%s\n","Familiay","Marks","Sr\_ball");**

**puts("Fam: Vvod zavesshaetsya - 0");**

**fflush(stdin);**

**scanf("%s",ob.fam);**

**while(ob.fam[0]!='0'){**

**puts("Marks:");**

**fflush(stdin);**

**ob.sr\_ball=0.0;**

**for(int i=0;i<4;i++){**

**scanf("%d",&ob.marks[i]);**

**ob.sr\_ball+=ob.marks[i];**

**}**

**ob.sr\_ball/=4;**

**fprintf(k1,"%-20s%2d%2d%2d%2d%9.2f\n",ob.fam,ob.marks[0],ob.marks[1],ob.marks[2],ob.marks[3],ob.sr\_ball);**

**puts("Fam: Vvod zavesshaetsya - 0");**

**fflush(stdin);**

**scanf("%s",ob.fam);**

**}**

**fclose(k1);**

**}**

**Создание файла произвольного доступа.**

Создавая с помощью функции форматированного вывода **fprintf** записи необязательно имеют одинаковую длину, а например отдельные записи в *файле с произвольным доступом* обычно имеют фиксированную длину, что позволяет получить к ним непосредственный (и следовательно, быстрый) доступ без поиска по всем записям. Для этого создаются бинарные файлы.

# 8.3 Работа с бинарным файлом

unsigned fread(void \*ptr,unsigned size,unsigned n, FILE \*stream);

Первый параметр- это указатель на буфер, в котором будут помещены данные из файла; следующие два параметра задают размер блока и количество читаемых блоков соответственно; последний параметр – это указатель на файл с которого считываются данные. Возвращает функция число успешно прочитанных элементов данных.

Индикатор текущей позиции в файле передвигается на число успешно прочитанных символов. В случае возникновения ошибки значение индикатора становится неопределенным.

unsigned fwrite(void \*ptr,unsignedsize,unsignedn, FILE \*stream);

Первый параметр- это указатель на буфер, из которого будут помещены данные в файл; следующие два параметра задают размер блока и количество записываемых блоков соответственно; последний параметр – это указатель на файл в который записываются данные. Возвращает функция число успешно прочитанных элементов данных.

Индикатор текущей позиции в файле передвигается на число успешно записанных символов. В случае возникновения ошибки значение индикатора становится неопределенным.

**Программа, которая создает бинарный файл для записи и чтения, записывает туда числа и считывает.**

**#include<stdio.h>**

**#include<stdlib.h>**

**void main()**

**{**

**FILE \*f1;**

**int n;**

**if(!(f1=fopen("int.bin","w+b"))){**

**puts("FILE ne otkrit!!!!!!");**

**exit(1);**

**}**

**puts("Zapis` v file: ");**

**do{**

**scanf("%d",&n);**

**fwrite(&n,sizeof(int),1,f1);**

**}while(n);**

**fseek(f1,0,0);**

**puts("\nSoderzanie faila:");**

**do{**

**fread(&n,sizeof(int),1,f1);**

**printf(" %d",n);**

**}while(n);**

**fclose(f1);**

**}**

**Программа, которая читает 10 чисел с конца из заранее созданного файла.**

**#include<stdio.h>**

**char name[]="int.bin";**

**void main()**

**{**

**int i,n=10,x;**

**FILE \*f;**

**f=fopen(name,"rb");**

**if(!f)**

**return;**

**for(i=1;i<=n;i++){**

**fseek(f,-i\*sizeof(int),SEEK\_END);**

**fread(&x,sizeof(int),1,f);**

**printf("\n%d\n",x);**

**}**

**fclose(f);**

**}**

**Функция для записи в файл.**

**void** fill\_file (FILE \* f){

**int** x;

scanf(“%d”,&x);

**while**(x){

fwrite(&x,**sizeof(int),**1,f);

scanf(“%d”,&x);

}

}

**Функция определение длины файла.**

**int** getFileSize(FILE \*f){

**int** count=0,tmp;

fseek(f,SEEK\_SET,0);

**while**(!feof(f)){

fread(&tmp**,sizeof(int**),1,f);

count++;

}

**return** count-1;

}

**Функция, читающая элемент заданной позиции.**

**int** getElementAtPosition**(int** i,FILE\*f){

**int** buf;

fseek(f,i\***sizeof(int**),SEEK\_SET);

fread(&buf,**sizeof(int**),1,f);

**return** buf;

}

**Функция, записывающая элемент в заданную позицию.**

**void** setFileElements **(int** pos **, int** element ){

fseek (f , pos \* **sizeof (int**) , SEEK\_SET );

fwrite (&element ,sizeof (**int**) , 1 , f);

}

**Функция, вывода элементов файла**

**void** print\_file (FILE \* f){

fseek(f,SEEK\_SET,0);

**for** (**int** i=0; ! feof (f); i++){

**int** x;

**if**( ! fread (&x , sizeof (int) , 1 , f) ) **break**;

printf ("element # %d = %d\n" , i , x);

}

}

**Программа с использованием данных функций.**

**#include<stdio.h>**

**#include<stdlib.h>**

**int getFileSize(FILE \*f);//Считает количество чисел в файле**

**int getElementAtPosition(int i,FILE\*f);//Возвращает число записанное на i-ой позиции**

**void setFileElements (int pos , int element , FILE \* f);//Заменяем число на позиции pos на число element**

**void print\_file (FILE \*f);//выводит содержание файла**

**void main()**

**{**

**FILE \*f1;**

**int n,n1;**

**if(!(f1=fopen("int.bin","w+b")))**

**{**

**puts("FILE ne otkrit!!!!!!");**

**exit(1);**

**}**

**puts("Zapis` v file: ");**

**do**

**{**

**scanf("%d",&n);**

**fwrite(&n,sizeof(int),1,f1);**

**}while(n);**

**puts("\nSoderzanie faila:");**

**fseek(f1,0,0);**

**print\_file(f1);**

**puts("\nKolichestvo elementov:");**

**fseek(f1,0,0);**

**printf("%d\n",getFileSize(f1));**

**puts("\nВведите nomer elementa konori xotite uvidet: ");**

**scanf("%d",&n);**

**fseek(f1,0,0);**

**printf("[%d]=%d\n",n,getElementAtPosition(n,f1));**

**puts("\nВведите element konori xotite pomestit v file: ");**

**scanf("%d",&n);**

**puts("\nВведите kuda vstavit element: ");**

**scanf("%d",&n1);**

**fseek(f1,0,0);**

**setFileElements(n1,n,f1);**

**puts("\nSoderzanie faila:");**

**fseek(f1,0,0);**

**print\_file(f1);**

**puts("\nKolichestvo elementov:");**

**fseek(f1,0,0);**

**printf("%d\n",getFileSize(f1));**

**fclose(f1);**

**}**

**int getFileSize(FILE \*f)**

**{**

**int count=0,tmp;**

**fseek(f,SEEK\_SET,0);**

**while(!feof(f))**

**{**

**fread(&tmp,sizeof(int),1,f);**

**count++;**

**}**

**return count-1;**

**}**

**int getElementAtPosition(int i,FILE\*f)**

**{**

**int buf;**

**fseek(f,i\*sizeof(int),SEEK\_SET);**

**fread(&buf,sizeof(int),1,f);**

**return buf;**

**}**

**void setFileElements (int pos , int element , FILE \* f){**

**fseek (f , pos \* sizeof (int) , SEEK\_SET );**

**fwrite (&element ,sizeof (int) , 1 , f);**

**}**

**void print\_file (FILE \*f){**

**fseek(f,SEEK\_SET,0);**

**for (int i=0; ! feof (f); i++){**

**int x;**

**if ( ! fread (&x , sizeof (int) , 1 , f) ) break;**

**printf ("element # %d = %d\n" , i , x);**

**}**

**}**

Задачи:

1. Программа, которая заполняет файл, потом считывает данные с файл и с начало и с конца. Использовать функции.

**#include<stdio.h>**

**#include<stdlib.h>**

**void fill\_file(FILE \*,int \*);**

**void print\_file(FILE \*,int \*);**

**void print\_file\_end(FILE \*,int \*);**

**char name[]="chisla.bin";**

**void main(){**

**int n;**

**FILE \*k;**

**k=fopen(name,"w+b");**

**if(!k)**

**return;**

**fill\_file(k,&n);**

**print\_file(k,&n);**

**print\_file\_end(k,&n);**

**fclose(k);**

**}**

**void fill\_file(FILE \*k,int \*p){**

**int x,i;**

**printf("Vvedive kol: ");**

**scanf("%d",p);**

**for(i=0;i<\*p;i++){**

**scanf("%d",&x);**

**fwrite(&x,sizeof(int),1,k);**

**}**

**}**

**void print\_file(FILE \*k,int \*p){**

**puts("\n");**

**int x,i;**

**fseek(k,0\*sizeof(int),SEEK\_SET);**

**for(i=0;i<\*p;i++){**

**fread(&x,sizeof(int),1,k);**

**printf("[%d]=%d\n",i,x);**

**}**

**}**

**void print\_file\_end(FILE \*k,int \*p){**

**puts("\n");**

**int x,i;**

**for(i=1;i<=\*p;i++){**

**fseek(k,-i\*sizeof(int),SEEK\_END);**

**fread(&x,sizeof(int),1,k);**

**printf("%d\n",x);**

**}**

**}**

2. Программа, которая получает 2 файла в которых записаны числа, причем оба файла отсортирован по возрастанию. Переписать данные в третий файл, так чтобы они были по возрастанию. Не использовать дополнительных файлов, массивов и т.д.

**#include<stdio.h>**

**#include<stdlib.h>**

**void print\_file(FILE \*);**

**char name1[]="int1.bin";**

**char name2[]="int2.bin";**

**char name3[]="int3.bin";**

**void main(){**

**int x1,x2;**

**FILE \*k1,\*k2,\*k3;**

**k1=fopen(name1,"rb");**

**if(!k1)**

**return;**

**k2=fopen(name2,"rb");**

**if(!k2)**

**return;**

**k3=fopen(name3,"w+b");**

**if(!k3)**

**return;**

**fread(&x1,sizeof(int),1,k1);**

**fread(&x2,sizeof(int),1,k2);**

**while(!feof(k1) && !feof(k2)){**

**if(x1<x2){**

**fwrite(&x1,sizeof(int),1,k3);**

**fread(&x1,sizeof(int),1,k1);**

**}**

**else{**

**fwrite(&x2,sizeof(int),1,k3);**

**fread(&x2,sizeof(int),1,k2);**

**}**

**}**

**if(feof(k1)){**

**while(!feof(k2)){**

**fwrite(&x2,sizeof(int),1,k3);**

**fread(&x2,sizeof(int),1,k2);**

**}**

**}**

**if(feof(k2)){**

**while(!feof(k1)){**

**fwrite(&x2,sizeof(int),1,k3);**

**fread(&x2,sizeof(int),1,k1);**

**}**

**}**

**print\_file(k1);**

**print\_file(k2);**

**print\_file(k3);**

**fclose(k1);**

**fclose(k2);**

**fclose(k3);**

**}**

**void print\_file(FILE \*k){**

**puts("\n");**

**int x;**

**fseek(k,0\*sizeof(int),SEEK\_SET);**

**fread(&x,sizeof(int),1,k);**

**while(!feof(k)){**

**printf("%d ",x);**

**fread(&x,sizeof(int),1,k);**

**}**

**}**

3\*. Программа, которая получает 2 файла в которых записаны числа, причем 1- ый файл отсортирован по возрастанию, а 2-ой по убыванию. Переписать данные в третий файл, так чтобы они были по возрастанию. Не использовать дополнительных файлов, массивов и т.д.

4. Отсортировать существующий бинарный файл, причем дополнительных массивов и файлов не использовать.

**#include <stdlib.h>**

**int getElementAtPosition (int i, FILE \*f);**

**void setFileElements (int pos , int element , FILE \* f);**

**int getFileSize (FILE \*f);**

**void print\_file ();**

**void main(){**

**FILE \* f = fopen ("111.bin" , "r+b");**

**if (f == NULL){**

**printf ("cannot open file");**

**exit (1);**

**}**

**int n = getFileSize (f);**

**int j;**

**for (int i=1; i < n; i++){**

**int elt = getElementAtPosition (i , f);**

**int jelt;**

**for (j=i-1; j >= 0 && (jelt = getElementAtPosition (j , f)) > elt; j--)**

**setFileElements (j+1 , jelt , f);**

**setFileElements (++j , elt , f);// поместить в позицию j+1 первый (текущий) элемент**

**}**

**fclose (f);**

**print\_file ();**

**}**

**int getElementAtPosition (int i, FILE \*f){**

**fseek (f , i \* sizeof (int), SEEK\_SET);**

**int buf;**

**fread (&buf , sizeof (int) , 1 , f);**

**return buf;**

**}**

**int getFileSize (FILE \*f){**

**int count = 0;**

**fseek (f , SEEK\_SET , 0);**

**int tmp;**

**while (!feof (f))**

**{**

**fread (&tmp , sizeof (int) , 1, f);**

**count++;**

**}**

**return count - 1;**

**}**

**void setFileElements (int pos , int element , FILE \* f){**

**fseek (f , pos \* sizeof (int) , SEEK\_SET );**

**fwrite (&element ,sizeof (int) , 1 , f);**

**}**

**void print\_file (){**

**FILE \* f = fopen ("111.bin", "rb");**

**if (f == NULL){**

**printf ("cannot open file");**

**exit (1);**

**}**

**for (int i=0; ! feof (f); i++){**

**int x;**

**if ( ! fread (&x , sizeof (int) , 1 , f) ) break;**

**printf ("element # %d = %d\n" , i , x);**

**}**

**fclose (f);**

**}**

**Задачи:**

1. В файле записаны числа, причем первое число целое и показывает сколько вещественных чисел записана в файле. Написать программу, которая считывает первое число и динамически выделяет массив вещественных чисел, куда переписывает данные из файла. Найти максимальный элемент массива, а также отсортировать данный массив в возрастающем порядке.
2. В файле записаны данные о студентах, причем первое число целое и показывает количество студентов в файле. Написать программу, которая считывает первое число и динамически выделяет массив структуры, куда переписывает данные из файла. Найти всех тех студентов, кто получил минимальный средний балл и отсортировать в алфавитном порядке по фамилии студентов.

# 8.4 Пример решения большой программы по бинарным файлам и структурам.

Данный раздел будет особенно полезен для студентов, которые выполняют курсовую работу по С. Эта задача один из вариантов курсовой работы для студентов-заочников.

*База данных содержит информацию о направлениях на пересдачу: номер (целое число), группа (строка 10 символов), предмет (строка 20 символов), ФИО преподавателя (строка 20 символов), ФИО студента (строка 20 символов), дата выписки (дд.мм.гггг).*

*Дополнительно программа должна реализовывать сервис выписки и просмотра направлений на нескольких студентов.*

\*/

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <windows.h>

void select(); //данный метод(функция) помогает сформировать пользовательское меню

void create\_new(); //создание новой базы данных

void view\_old(); //просмотр ранее созданной базы данных

void correct\_old(); //корректировка ранее созданной базы данных из этого метода вызываются void modify\_old();void append\_old();

void sortfind(); //тут сортировки и поиск данных в базе

void modify\_old(); //изменение записи

void append\_old(); //добавление записи

FILE\* open\_file(char\*); //метод открывает файл

void view\_file(FILE\*); //метод показывает файл

char\* rus(const char\*); //универсальный метод русификации, в отличие от setlocale и SetConsoleCp, работает всегда и везде

//-----------------------Русификация--------------------

char bufRus[256];

char\* rus(const char\* text)

{

CharToOem(text, bufRus);

return bufRus;

}

char\* rus1(const char\* text)

{

OemToChar(text, bufRus);

return bufRus;

}

//-----------------------конец русификации---------------

//----------- тип данных ЗАПИСЬ student ----------

typedef struct {

char fios[80]; //ФИО студента (строка 80 символов)

char group[11]; //группа (строка 10 символов)

int num; //номер (целое число)

char data[20]; //дата выписки (дд.мм.гггг)

char predmet[21]; //предмет (строка 20 символов)

char fiop[80]; //ФИО преподавателя (строка 80 символов),

}student;

/\*\_\_user's menu\_\_\_\*/

void select()

{

system("cls");

puts(rus("1 - Создать базу"));

puts(rus("2 - Просмотр базы"));

puts(rus("3 - Коррекция базы"));

puts(rus("4 - Поиск и сортировки на переcдачу"));

puts(rus("5 - Выход"));

printf("\n");

puts(rus("Выберите действие"));

char sel=getch();

switch (sel) {

case '1':create\_new(); break;

case '2':view\_old(); break;

case '3':correct\_old(); break;

case '4':sortfind(); break;

case '5':puts(rus("Уверены, что хотите выйти? (y/n)")); fflush(stdin);

if ((getch()=='y')||(getch()=='Y'))

{

puts(rus("Спасибо за работу!"));

getch();

exit(0);

}

else break;

}

}//\_\_end\_user's menu\_\_\_\_

/\*---------суммируем------------\*/

void sortfind()

{

FILE\* sum\_file;

FILE\* new\_file;

char obrazec[80];

student sp[500],bufer; //для сортировки

sum\_file=open\_file("rb");

system("cls");

view\_file(sum\_file);

fseek(sum\_file, 0, SEEK\_SET);

puts(rus("1 - Фильтровать записи по студенту "));

puts(rus("2 - Фильтровать записи по преподавателю"));

puts(rus("3 - Фильтровать записи по предмету"));

puts(rus("4 - сортировать записи по студенту "));

puts(rus("5 - сортировать записи по преподавателю"));

puts(rus("6 - сортировать записи по предмету"));

puts(rus("7 - вывести студентов в текстовый файл output.txt"));

puts(rus("8 - Отмена"));

printf("\n");

puts(rus("Выберите действие"));

fflush(stdin);

int i,j,n,size\_stud=sizeof(student);

char sel=getch();

switch (sel) {

case '1':

//начало поиска

puts(rus("фильтр по фамилии: "));

puts(rus("Введите фамилию студента: "));

gets(obrazec);

size\_stud=sizeof(student);

student stud;

i=1;

printf(rus("N\tФИО | группа | предмет | дата | фио препод. |\n")); //!!!!!!!!!!!!!!!

printf(rus("-----------------------------------------------------------\n"));

fread(&stud,size\_stud,1,sum\_file);

if(strcmp(obrazec,stud.fios)==0)

printf("%d%13s|%8s|%12s|%12s|%13s|\n",i,stud.fios, stud.group, stud.predmet, stud.data,stud.fiop);

while(2)

{

i++;

if(!fread(&stud,size\_stud,1,sum\_file)) break;

if(strcmp(obrazec,stud.fios)==0)

printf("%d%13s|%8s|%12s|%12s|%13s|\n",i,stud.fios, stud.group, stud.predmet, stud.data,stud.fiop);

}

//\_\_\_\_\_конец печати\_\_\_\_\_\_

printf("\n");

getch();

//конец поиска

break;

case '2':

//начало поиска

puts(rus("фильтр по фамилии: "));

puts(rus("Введите фамилию преподавателя: "));

gets(obrazec);

size\_stud=sizeof(student);

//student stud;

i=1;

printf(rus("N\tФИО | группа | предмет | дата | фио препод. |\n")); //!!!!!!!!!!!!!!!

printf(rus("-----------------------------------------------------------\n"));

fread(&stud,size\_stud,1,sum\_file);

if(strcmp(obrazec,stud.fiop)==0)

printf("%d%13s|%8s|%12s|%12s|%13s|\n",i,stud.fios, stud.group, stud.predmet, stud.data,stud.fiop);

while(2)

{

i++;

if(!fread(&stud,size\_stud,1,sum\_file)) break;

if(strcmp(obrazec,stud.fiop)==0)

printf("%d%13s|%8s|%12s|%12s|%13s|\n",i,stud.fios, stud.group, stud.predmet, stud.data,stud.fiop);

}

//\_\_\_\_\_конец печати\_\_\_\_\_\_

printf("\n");

getch();

//конец поиска

break;

case '3':

//начало поиска

puts(rus("фильтр по предмету для перездачи: "));

puts(rus("Введите предмет: "));

gets(obrazec);

size\_stud=sizeof(student);

//student stud;

i=1;

printf(rus("N\tФИО | группа | предмет | дата | фио препод. |\n")); //!!!!!!!!!!!!!!!

printf(rus("-----------------------------------------------------------\n"));

fread(&stud,size\_stud,1,sum\_file);

if(strcmp(obrazec,stud.predmet)==0)

printf("%d%13s|%8s|%12s|%12s|%13s|\n",i,stud.fios, stud.group, stud.predmet, stud.data,stud.fiop);

while(2)

{

i++;

if(!fread(&stud,size\_stud,1,sum\_file)) break;

if(strcmp(obrazec,stud.predmet)==0)

printf("%d%13s|%8s|%12s|%12s|%13s|\n",i,stud.fios, stud.group, stud.predmet, stud.data,stud.fiop);

}

//\_\_\_\_\_конец печати\_\_\_\_\_\_

printf("\n");

getch();

//конец поиска

break;

case '4':

//----------------------сортируем---------------

i=0;

fread(&stud,size\_stud,1,sum\_file);

sp[i]=stud;

while(2)

{

i++;

if(!fread(&stud,size\_stud,1,sum\_file)) break;

sp[i]=stud;

}

n=i;

for(j=1;j<n;j++)

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

if(strcmp(sp[i].fios,sp[i+1].fios)>0)

{

bufer=sp[i];

sp[i]=sp[i+1];

sp[i+1]=bufer;

}

printf(rus("N\tФИО | группа | предмет | дата | фио препод. |\n"));

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

printf("%d%13s|%8s|%12s|%12s|%13s|\n",i,sp[i].fios, sp[i].group, sp[i].predmet, sp[i].data,sp[i].fiop);

getch();

//----------------------сортируем конец---------------

break;

case '5':

//----------------------сортируем---------------

i=0;

fread(&stud,size\_stud,1,sum\_file);

sp[i]=stud;

while(2)

{

i++;

if(!fread(&stud,size\_stud,1,sum\_file)) break;

sp[i]=stud;

}

n=i;

for(j=1;j<n;j++)

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

if(strcmp(sp[i].fiop,sp[i+1].fiop)>0)

{

bufer=sp[i];

sp[i]=sp[i+1];

sp[i+1]=bufer;

}

printf(rus("N\tФИО | группа | предмет | дата | фио препод. |\n"));

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

printf("%d%13s|%8s|%12s|%12s|%13s|\n",i,sp[i].fios, sp[i].group, sp[i].predmet, sp[i].data,sp[i].fiop);

getch();

//----------------------сортируем конец---------------

break;

case '6':

//----------------------сортируем---------------

i=0;

fread(&stud,size\_stud,1,sum\_file);

sp[i]=stud;

while(2)

{

i++;

if(!fread(&stud,size\_stud,1,sum\_file)) break;

sp[i]=stud;

}

n=i;

for(j=1;j<n;j++)

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

if(strcmp(sp[i].predmet,sp[i+1].predmet)>0)

{

bufer=sp[i];

sp[i]=sp[i+1];

sp[i+1]=bufer;

}

printf(rus("N\tФИО | группа | предмет | дата | фио препод. |\n"));

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

printf("%d%13s|%8s|%12s|%12s|%13s|\n",i,sp[i].fios, sp[i].group, sp[i].predmet, sp[i].data,sp[i].fiop);

getch();

//----------------------сортируем конец---------------

break;

case '7':

i=0;

fread(&stud,size\_stud,1,sum\_file);

sp[i]=stud;

while(2)

{

i++;

if(!fread(&stud,size\_stud,1,sum\_file)) break;

sp[i]=stud;

}

n=i;

printf(rus("печать базы в текстовый файл!!!\n"));

new\_file=open\_file("wt");

fprintf(new\_file,"N\tФИО | группа | предмет | дата | фио препод. |\n");

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

{

OemToChar(sp[i].fios,sp[i].fios);

OemToChar(sp[i].group,sp[i].group);

OemToChar(sp[i].predmet,sp[i].predmet);

OemToChar(sp[i].fiop,sp[i].fiop);

fprintf(new\_file,"%d%13s|%8s|%12s|%12s|%13s|\n",i,sp[i].fios, sp[i].group, sp[i].predmet, sp[i].data,sp[i].fiop);

}

puts(rus("Информация записана!"));

getch();

fclose(new\_file);

break;

case '8': break;

}

}

/\*\_\_\_open\_file\_\_\_\*/

FILE\* open\_file(char\* mode)

{

FILE\* file\_;

char filename[30];

puts(rus("Введите имя файла: "));

scanf("%s",filename);

if (!(file\_=fopen(filename, mode)))

{

fprintf(stderr,rus("Ошибка! Файл не %s"),(mode=="wb")?"cozdan":"naiden");

getch();

exit(1);

}

return file\_;

}

/\*\_\_\_viewing file\_\_\_\*/

void view\_file(FILE\* dest\_file)

{

puts(rus("Просмотр файла:"));

printf("\n");

//\_\_\_\_\_\_печатаем старый файл\_\_\_\_

//int i, size\_int=sizeof(int);

int size\_stud=sizeof(student);

student stud;

int i=1;

printf(rus("N\tФИО | группа | предмет | дата | фио препод. |\n")); //!!!!!!!!!!!!!!!

printf(rus("-----------------------------------------------------------\n"));

fread(&stud,size\_stud,1,dest\_file);

printf("%d%13s|%8s|%12s|%12s|%13s|\n",i,stud.fios, stud.group, stud.predmet, stud.data,stud.fiop);

while(2)

{

i++;

if(!fread(&stud,size\_stud,1,dest\_file)) break;

printf("%d%13s|%8s|%12s|%12s|%13s|\n",i,stud.fios, stud.group, stud.predmet, stud.data,stud.fiop);

}

//\_\_\_\_\_конец печати\_\_\_\_\_\_

printf("\n");

}

/\*\_\_\_creating\_\_\_\*/

void create\_new()

{

FILE\* new\_file;

new\_file=open\_file("wb");

student stud;

puts(rus("Введите записи о студентах. Выход - введите 1"));

//int i, size\_int=sizeof(int);

int size\_stud=sizeof(student);

fflush(stdin);

while (1)

{

printf("\n");

puts(rus("ФИО студента: "));

gets(stud.fios);

fflush(stdin);

if ((stud.fios[0]=='1')) break;

puts(rus("группа: "));

scanf("%s",stud.group);

fflush(stdin);

puts(rus("ФИО преподавателя: "));

gets(stud.fiop);

fflush(stdin);

puts(rus("предмет: "));

gets(stud.predmet);

fflush(stdin);

puts(rus("дата перездачи(дд.мм.гггг): "));

gets(stud.data);

fflush(stdin);

fwrite(&stud, size\_stud, 1, new\_file);

fflush(stdin);

}

fclose(new\_file);

puts(rus("Информация записана!"));

getch();

}

/\*\_\_\_viewing\_\_\_\*/

void view\_old()

{

FILE\* old\_file;

old\_file=open\_file("rb");

view\_file(old\_file);

puts(rus("нажмите клавишу.."));

getch();

}

/\*\_\_\_correcting\_\_\_\*/

void correct\_old()

{

puts(rus("6 - Изменить базу"));

puts(rus("7 - Добавить в базу"));

puts(rus("8 - Отмена"));

printf("\n");

puts(rus("Выберите действие"));

fflush(stdin);

char sel=getch();

switch (sel) {

case '6':modify\_old(); break;

case '7':append\_old(); break;

case '8': break;

}

}

/\*\_\_\_modify\_\_\_\*/

void modify\_old()

{

FILE\* modif\_file;

int ch\_num;

student stud;

int size\_stud=sizeof(student);

modif\_file=open\_file("r+b");

//int i, size\_int=sizeof(int);

view\_file(modif\_file);

//modif\_file=freopen();

puts(rus("Введите номер записи, которую нужно изменить: "));

fflush(stdin);

scanf("%d",&ch\_num);

puts(rus("Введите новое значение:"));

fflush(stdin);

//----новое--

{

printf("\n");

puts(rus("ФИО студента: "));

gets(stud.fios);

puts(stud.fios);

fflush(stdin);

puts(rus("группа: "));

scanf("%s",stud.group);

fflush(stdin);

puts(rus("ФИО преподавателя: "));

gets(stud.fiop);

fflush(stdin);

puts(rus("предмет: "));

gets(stud.predmet);

fflush(stdin);

puts(rus("дата перездачи(дд.мм.гггг): "));

gets(stud.data);

fflush(stdin);

}

//---конец нового

if (!fseek(modif\_file, (long)size\_stud\*(ch\_num-1), SEEK\_SET))

fwrite(&stud,size\_stud,1,modif\_file);

else puts(rus("Ошибка! Неверные данные!"));

puts(rus("Нажмите клавишу.."));

getch();

fclose(modif\_file);

}

/\*\_\_append\_\_\_\*/

void append\_old()

{

FILE\* app\_file;

app\_file=open\_file("a+b");

student stud;

int size\_stud=sizeof(student);

view\_file(app\_file);

puts(rus("Введите новое значение:"));

fflush(stdin);

//----новое--

{

printf("\n");

puts(rus("ФИО студента: "));

gets(stud.fios);

puts(stud.fios);

fflush(stdin);

puts(rus("группа: "));

scanf("%s",stud.group);

fflush(stdin);

puts(rus("ФИО преподавателя: "));

gets(stud.fiop);

fflush(stdin);

puts(rus("предмет: "));

gets(stud.predmet);

fflush(stdin);

puts(rus("дата перездачи(дд.мм.гггг): "));

gets(stud.data);

fflush(stdin);

}

//---конец нового

if (!fseek(app\_file, 0 ,SEEK\_END))

fwrite(&stud,size\_stud,1,app\_file);

else puts(rus("Ошибка! Неверные данные!"));

puts(rus("Нажмите клавишу.."));

getch();

fclose(app\_file);

}

/\*\_\_\_main\_\_\*/

void main()

{

while (1)

{

select();

fflush(stdin);

}

}

Результаты тестирования (с приведением скриншотов интерфейса).

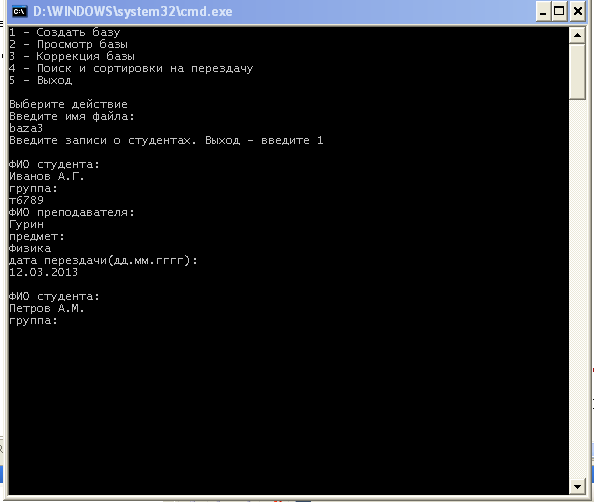


Рисунок Создание новой базы данных

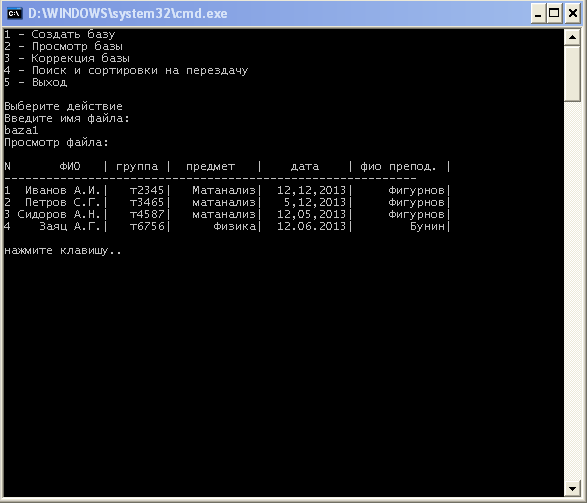


Рисунок Открытие существующей БД

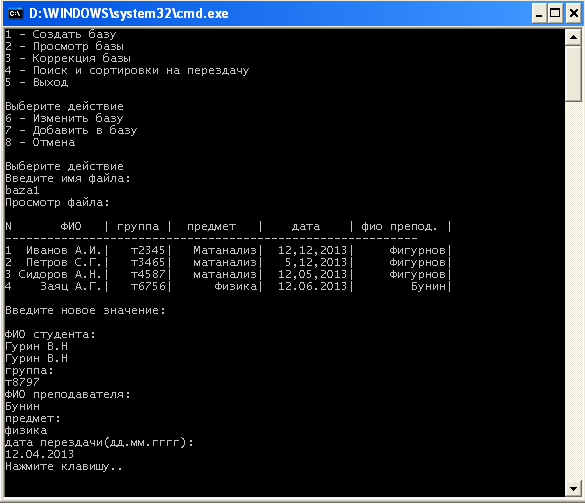


Рисунок Добавление записи в БД

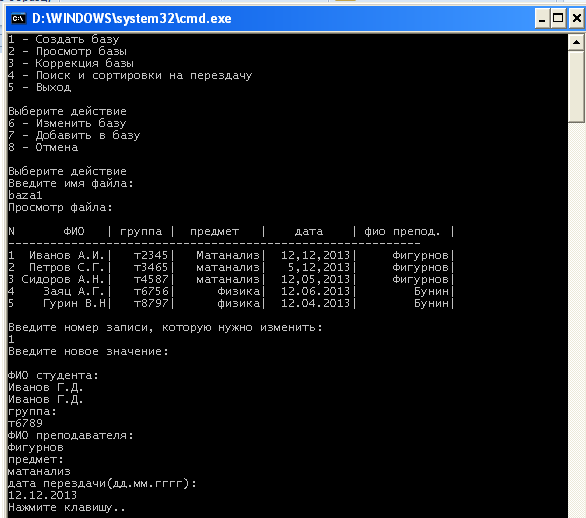


Рисунок Редактирование записи

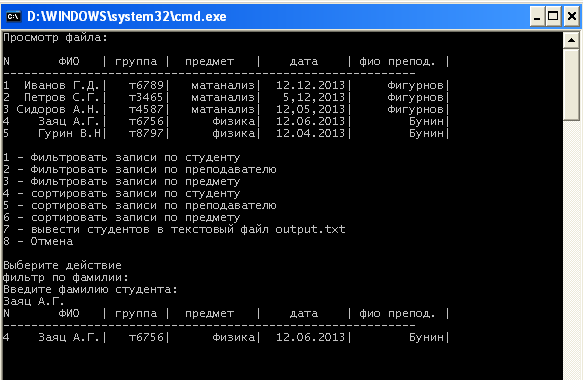


Рисунок Фильтр по фамилии студента

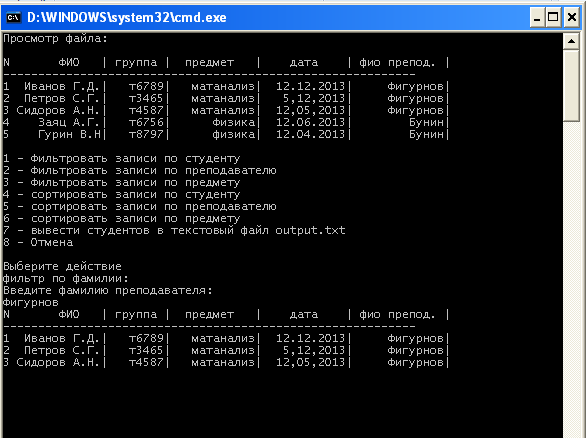


Рисунок Фильтр по фамилии преподавателя

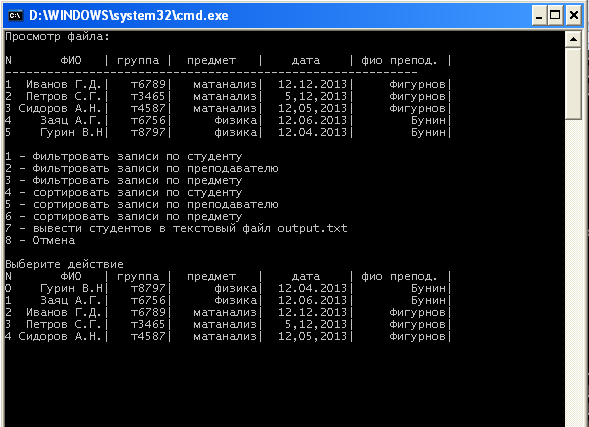


Рисунок Сортировка записей по фамилии студента

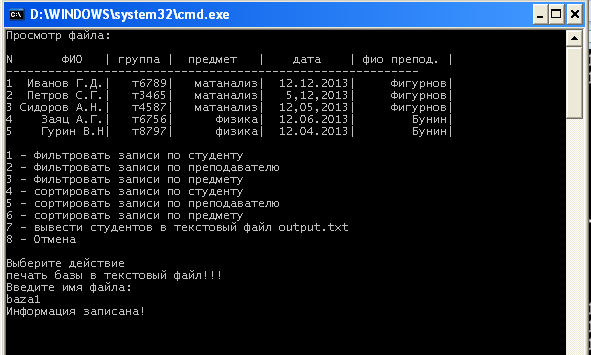


Рисунок Экспорт БД в текстовый файл

# 8.5 Kонтрольные вопросы

1. **Какие операции определены со строками?**

а) Ни какие

б)Операции сравнения

в)Арифметические операции

г)Логические операции

д)Операция присваивания

2. **Пользователь вводит строку "Иванов Иван Иванович". Эту строку программа читает после нажатия клавиши Enter с помощью функции scanf("%s", str). Что будет занесено в массив str?**

а) Ничего, так как перед именем str не хватает символа &  
б) Ничего, так как неверно задан формат ввода строки   
в) "Иванов Иван Иванович\n"   
г) "Иванов Иван Иванович"   
д) "Иванов\n"   
е) "Иванов"

3 **Какой результат вернет функция strcmp("a","b"); ?**

а) Положительное число

б) Отрицательное число

с) Минус один

д) Плюс один

4 **Динамически выделенная память**

а) автоматически освобождается после завершения работы функции

б) автоматически инициализируется нулевыми значениями

с) может быть освобождена с использованием стандартных функций

д) выделяется для специально описанных динамических переменных

е) выделяется с использованием стандартных функций

5 **Функция void free(void \*ptr);**

а) очищает динамически выделенную память, заполняя ее нулевыми значениями

б) выделяет динамическую память

с) освобождает динамически выделенную память

д) устанавливает указатель ptr в NULL

6 **Заполните пробелы в следующих утверждениях:**

а) ключевое слово \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ начинает определение структуры.

б) прямой доступ к элементам структуры посредством операций \_\_\_\_\_\_\_\_\_\_\_\_\_\_

в) доступ по указателю \_\_\_\_\_\_\_\_\_\_\_\_\_\_

7 **Заполните пробелы в следующих утверждениях:**

а) ключевое слово \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ начинает определение структуры.

б) прямой доступ к элементам структуры посредством операций \_\_\_\_\_\_\_\_\_\_\_\_\_\_

в) доступ по указателю \_\_\_\_\_\_\_\_\_\_\_\_\_\_

8 **В чем различие между структурой и объединением** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

9 **Какие файлы вы знаете? Укажите все различия между ними.** \_\_\_\_\_\_\_\_

10 **Укажите функции чтения из** а) текстового файла \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

б) бинарного файла \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

11 **Укажите функции вывода**

а) в текстовый файл \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

б) в бинарный файл \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

12 **Функция, которая перемещает курсор в файле** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# ПРИЛОЖЕНИЕ 1. Функции ввода/вывода.

Прототип функции **printf()**

**int printf(“constchar \* string”,…); %[флаг][ширина][точность][модификатор типа] тип\_формата**

**возвращаемое значение – это число выведенных символов.**

**тип\_формата**

|  |  |
| --- | --- |
| Тип  формата | Представление данных при выводе |
| **с** | отдельный символ |
| **s** | символьная строка |
| **d,i** | Целое десятичное число int |
| **u** | Целое беззнаковое десятичное число unsignedint |
| **o** | Целое беззнаковое восьмеричное число |
| **x** | Целое беззнаковое шестнадцатеричное число (для вывода используются 0-f) |
| **X** | Целое беззнаковое шестнадцатеричное число (для вывода используются 0-F) |
| **f** | числа с плавающей запятой в записи с фиксированной десятичной точкой float,double |
| **e** | Значение со знаком в формате [-]d.dddde[+|-]ddd |
| **E** | Значение со знаком в формате [-]d.ddddE[+|-]ddd |
| **g** | Значение со знаком в формате ‘e’ или ‘f’, в зависимости от значения и указанной точности |
| **G** | Значение со знаком в формате ‘E’ или ‘F’, в зависимости от значения и указанной точности |
| **p** | Значение указателя, т.е. адрес |
| **n** | Число записываемых в данный момент символов |
| **[….]** | Соответствует самой длинной строке, которая состоит из перечисленных в скобках символов |
| **[^…]** | Соответствует самой длинной строке, которая не содержит перечисленных в скобках символов |
| **%** | Сам знак %, преобразование не производит |

**Флаги**

|  |  |
| --- | --- |
| **знаки** | **Действие** |
| **-** | Выравнивание в лево выводимого числа в пределах выделенного поля. Правая сторона выделенного поля заполняется пробелами. Если знак не стоит, то по умолчанию производится выравнивание вправо, т.е. перед числом. |
| **+** | Выводит знак числа. При отрицательных значениях знак выводится всегда и не зависит от наличия данного флага |
| **Пробел** | Выводит знак пробела перед положительным числом |
| **0** | Заполняет поле нулями |
| **#** | зависит от типа формата: для целых чисел выводит идентификатор системы счисления; 0- перед восьмеричным числом; 0х или 0Х – перед шестеричным. При указании типа формата e, E или f происходит вывод десятичной точки. Действие данного символа при использовании формата g или G идентично действию при e и E |

**Ширина** – в данном поле устанавливается минимальная ширина поля в символах. Если число подлежит символов меньше, чем указанно в этом поле, то слева или справа добавляются пробелы для достижения указанного значения. Если перед числом стоит 0, то вместо пробелов добавляются нули.

**Точность** – задает число подлежащих выводу десятичных знаков и должно начинаться точкой. Действие поля зависит от типа поля.

|  |  |
| --- | --- |
| **Символ** | **Действие** |
| **d, i, u, o, x, X** | Указывает минимальное число выводимых цифр |
| **E, e, f** | Указывает число цифр, которые выводятся после десятичной точки. Последняя цифра округляется |
| **G,g** | Выводит указанное число значащих цифр |
| **c** | Не действует. Выводится соответствующий символ |
| **s** | Указывает максимальное число выводимых цифр |

**Модификатор типа**

|  |  |
| --- | --- |
| **Модифи- катор** | **Значение** |
| **H** | Если тип формата - d, i, o, x или X, то тип параметра – short int. При типе формата u тип параметра unsigned short int |
| **L** | Если тип формата - d, i, o, x или X, то тип параметра – longint. При типе формата u тип параметра unsignedlong. При типе формата – e, E, f, g лил G тип параметра – double вместо float |
| **L** | При типе формата – e, E, f, g лил G тип параметра – long double |

**Пример использования функции printf().**

**#include<stdio.h>**

**void main()**

**{**

**printf("%-21s%-10s%s\n","Name","Marks","SR\_BALL");**

**printf("%-20s%2d%2d%2d%2d%8.2lf","Lena",4,4,4,7,3.45);**

**}**

Прототип функции **scanf()**

**int scanf(constchar \* format, ……); возвращаемое значение – число считанных полей**

**format - %d,%f,%s,%lf,%c**

# ПРИЛОЖЕНИЕ 2.

# Директивы Препроцессора

Директивы препроцессора представляют собой инструкции, записанные в тексте программы на СИ, и выполняемые до трансляции программы. Директивы препроцессора позволяют изменить текст программы, например, заменить некоторые лексемы в тексте, вставить текст из другого файла, запретить трансляцию части текста и т.п. Все директивы препроцессора начинаются со знака #. После директив препроцессора точка с запятой не ставятся.

#### Директива #include

Директива #include включает в текст программы содержимое указанного файла. Эта директива имеет две формы:

#include "имя файла"

#include <имя файла>

Имя файла должно соответствовать соглашениям операционной системы и может состоять либо только из имени файла, либо из имени файла с предшествующим ему маршрутом. Если имя файла указано в кавычках, то поиск файла осуществляется в соответствии с заданным маршрутом, а при его отсутствии в текущем каталоге. Если имя файла задано в угловых скобках, то поиск файла производится в стандартных директориях операционной системы, задаваемых командой PATH.

Директива #include может быть вложенной, т.е. во включаемом файле тоже может содержаться директива #include, которая замещается после включения файла, содержащего эту директиву.

Директива #include широко используется для включения в программу так называемых заголовочных файлов, содержащих прототипы библиотечных функций, и поэтому большинство программ на С начинаются с этой директивы.

#### Директива #define

Директива #define служит для замены часто использующихся констант, ключевых слов, операторов или выражений некоторыми идентификаторами. Идентификаторы, заменяющие текстовые или числовые константы, называют именованными константами. Идентификаторы, заменяющие фрагменты программ, называют макроопределениями, причем макроопределения могут иметь аргументы.

Директива #define имеет две синтаксические формы:

#define идентификатор текст

#define идентификатор (список параметров) текст

Эта директива заменяет все последующие вхождения идентификатора на текст. Такой процесс называется макроподстановкой. Текст может представлять собой любой фрагмент программы на С, а также может и отсутствовать. В последнем случае все экземпляры идентификатора удаляются из программы.

Пример:

#define WIDTH 80

#define LENGTH (WIDTH+10)

Эти директивы изменят в тексте программы каждое слово WIDTH на число 80, а каждое слово LENGTH на выражение (80+10) вместе с окружающими его скобками.

Скобки, содержащиеся в макроопределении, позволяют избежать недоразумений, связанных с порядком вычисления операций. Например, при отсутствии скобок выражение t=LENGTH\*7 будет преобразовано в выражение t=80+10\*7, а не в выражение t=(80+10)\*7, как это получается при наличии скобок, и в результате получится 780, а не 630.

Во второй синтаксической форме в директиве #define имеется список формальных параметров, который может содержать один или несколько идентификаторов, разделенных запятыми. Формальные параметры в тексте макроопределения отмечают позиции на которые должны быть подставлены фактические аргументы макровызова. Каждый формальный параметр может появиться в тексте макроопределения несколько раз.

При макровызове вслед за идентификатором записывается список фактических аргументов, количество которых должно совпадать с количеством формальных параметров.

Пример:

#define MAX(x,y) ((x)>(y))?(x):(y)

Эта директива заменит фрагмент

t=MAX(i,s[i]);

на фрагмент

t=((i)>(s[i])?(i):(s[i]);

Как и в предыдущем примере, круглые скобки, в которые заключены формальные параметры макроопределения, позволяют избежать ошибок связанных с неправильным порядком выполнения операций, если фактические аргументы являются выражениями.

Например, при наличии скобок фрагмент

t=MAX(i&j,s[i]||j);

будет заменен на фрагмент

t=((i&j)>(s[i]||j)?(i&j):(s[i]||j);

а при отсутствии скобок - на фрагмент

t=(i&j>s[i]||j)?i&j:s[i]||j;

в котором условное выражение вычисляется в совершенно другом порядке.

#### Директива #undef

Директива #undef используется для отмены действия директивы #define. Синтаксис этой директивы следующий #undef идентификатор

Директива отменяет действие текущего определения #define для указанного идентификатора. Не является ошибкой использование директивы #undef для идентификатора, который не был определен директивой #define.

Пример:

#undef WIDTH

#undef MAX

Эти директивы отменяют определение именованной константы WIDTH и макроопределения MAX.