Функции

Функции

- В C++ программа состоит только из функций.
- ***Функция*** - это независимая совокупность объявлений и операторов, обычно предназначенная для выполне ния определенной задачи.
- Каждая функция имеет имя, которое используется для вызова функции.

Объявление функции

- Формат описания функции:

[тип] <имя функции> (список параметров)
{
тело функции
}

Объявление функции

- Тип определяет тип значения, которое возвращает функция с помощью оператора **return**.
- Если тип не указан, то по умолчанию предполагается, что функция возвращает целое значение (типа **int**).
- Список параметров состоит из перечня типов и имен параметров, разделенных запятыми.
- Функция может не иметь параметров, но круглые скобки необходимы в любом случае.

Объявление функции

- В списке параметров для каждого параметра должен быть указан тип:

f(int x, int y, float z)

Функции

- Когда функция не возвращает никакого значения, она должна быть описана как функция типа **void** (пустая).
- Если не объявлять функцию типа **void**, тогда она по умолчанию будет иметь тип **int** и не возвращать никакого значения. Это вызовет предупреждающее сообщение компилятора, но не будет препятствием для компиляции.

Рекурсивные функции

Рекурсивные функции

- В языке C++ функции могут вызывать сами себя.
- Функция называется рекурсивной, если оператор в теле функции содержит вызов этой же функции.
- Классический пример рекурсивной функции - это вычисление факториала числа **n!=1*2*3*…*n.**
- Вызов функции в рекурсивной функции не создает новую копию функции, а создает в памяти новые копии локальных переменных и параметров.
- Из рекурсивной функции надо предусмотреть выход, иначе это вызовет “зависание” системы.

Пример

```
factorial(int n)
    {
    int a;
    if (n==1) return 1;
    a=factorial(n-1)*n;
    return a;
    }
```

Директивы препроцессора

Препроцессор

- Препроцессор языка C++ -это программа, выполняющая обработку входных данных для другой программы.
- Препроцессор просматривает программу до компилятора, заменяет аббревиатуры в тексте программы на соответствующие директивы, отыскивает и подключает необходимые файлы, может влиять на условия компиляции.
- Директивы препроцессора не являются языком С, они расширяют область действия среды программирования среды С.
- Все директивы начинаются с символа #.

Директивы препроцессора

Директива	Описание
#define	Описание макроса
#undef	Отмена определения макроса
#if	Включение объекта-заголовка
#ifdef	Компиляция, если выражение истинно
#else	Компиляция, если выражение в if ложно
#elif	Составная директива else/if
#endif	Окончание группы компиляции по условию
#error	Формирование ошибок трансляции
#pragma	Действие определяется реализацией
#	Null-директива

Директива #define

#define

- Директива **#define** вводит макроопределение (макрос) или символическую константу.
- Формат:

  #define <имя макроса> <последовательность символов>
- Последовательность символов иногда называют строкой замещения.
- Когда препроцессор находит в исходном тексте программы имя макроса (в дальнейшем макрос), он заменяет его на последовательность символов - ***макроподстановка***.
- Для имени макроса принято использовать прописные буквы.

Пример

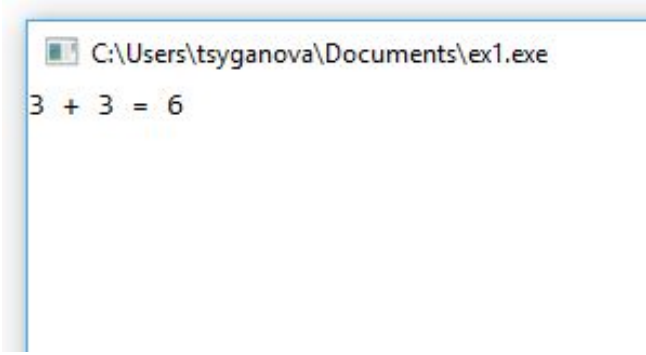
#define TRUE 1; //1 и 0 заменяют в исходном файле

#define FALSE 0; //имена TRUE и FALSE.

#define MAX 100;

Пример

```
#include <stdio.h>
#define A 3
int main(){
  printf("%d + %d = %d", A, A, A+A); // 3 + 3 = 6
  getchar();
  return 0;
}
```



#define

В зависимости от значения константы компилятор присваивает ей тот или иной тип.

С помощью суффиксов можно переопределить тип константы:

- U или u - целая беззнаковая(unsigned);
- F или f - вещественная типа float;
- L или l - long int / long double.

Пример

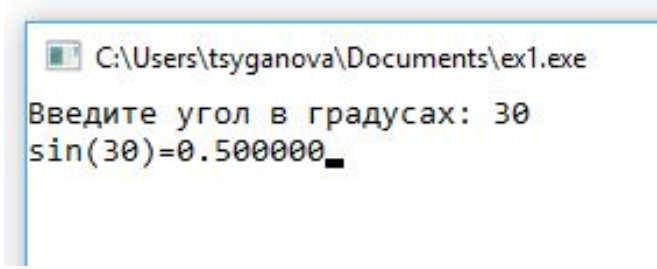
```
#define A 280U       // unsigned int
#define B 280LU      // unsigned long int
#define C 280     // int (long int)
#define D 280L    // long int
#define E 28.0    // double
#define F 28.0F   // float
#define G 28.0L      // long double
```

#define

- Второй вариант синтаксиса определяет макрос, подобный функции, с параметрами.
- Данная форма допускает использование необязательного списка параметров, которые должны находиться в скобках.

Пример

```
//Вычисление синуса угла
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.14159265
#define SIN(x) sin(PI*x/180)
int main() {
  int c;
  printf("Введите угол в градусах: ");
  scanf("%d", &c);
  printf("sin(%d)=%lf", c, SIN(c));
  getchar();
  return 0;
}
```

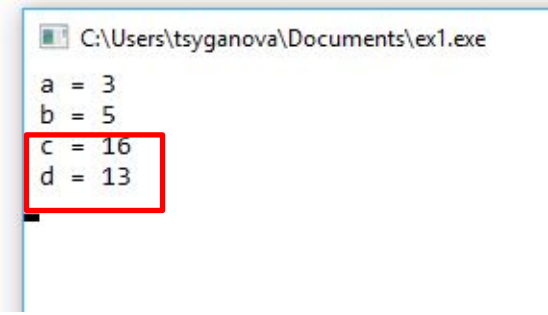


#define

- Отличием макроопределений от функций является то, что на этапе компиляции каждое вхождение идентификатора замещается соответствующим кодом, т.е. программа может иметь несколько копий одного и того же кода, соответствующего идентификатору.
- В случае работы с функциями программа будет содержать 1 экземпляр кода, реализующий указанную функцию, и каждый раз при обращении к функции ей будет передано управление.
- Отменить макроопределение можно с помощью директивы **#undef**.

Пример неудачного использования

```
#include <stdio.h>
#define sum(A,B) A+B
int main(){
  int a,b,c,d;
  a=3; b=5;
  printf(" a = %d\n b = %d\n", a, b);
  c = (a + b)*2; // c = (a + b)*2
  d = sum(a, b) * 2; // d = a + b*2;
  printf(" c = %d \n d = %d \n", c, d);
  getchar();
  return 0;
}
```



Условная компиляция

Условная компиляция

Директивы ***#if*** или ***#ifdef / #ifndef*** вместе с директивами ***#elif***, ***#else*** и ***#endif*** управляют компиляцией частей исходного файла.

Если указанное выражение после ***#if*** имеет ненулевое значение, в записи преобразования сохраняется группа строк, следующая сразу за директивой ***#if***.

Отличие директив ***#ifdef/#ifndef*** заключается в том, что константное выражение может быть задано только с помощью #define.

Условная компиляция

Синтаксис:

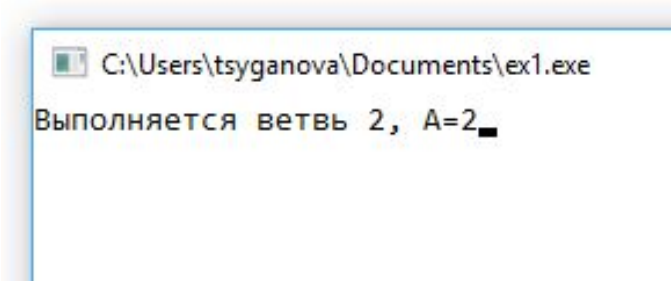
#if константное выражение
группа операций
#elif константное выражение
группа операций
#else
группа операций
#endif

Условная компиляция

- У каждой директивы ***#if*** в исходном файле должна быть соответствующая закрывающая директива ***#endif***.
- Между директивами ***#if*** и ***#endif*** может располагаться любое количество директив ***#elif***, однако допускается не более одной директивы ***#else***.
- Директива ***#else***, если присутствует, должна быть последней перед директивой ***#endif***.

Пример

```
#include <stdio.h>
#include <stdlib.h>
#define A 2
int main() {
#if A==1
  printf("Выполняется ветвь 1");
#elif A==2
  printf("Выполняется ветвь 2, A=%d", A);
#else
  printf("Выполняется третья ветвь, A=%d", A);
#endif
  getchar();
  return 0;
}
```



Директива #include

#include

- Директива **#include** подключает к исходному тексту заданные в директивы файлы.
- Данные файлы называют подключаемыми, заголовочными файлами и заголовками.
- Часто в качестве подключаемых файлов используются заголовочные файлы библиотек языка С.
- Формат:

#include <имя заголовка>;

#include "имя заголовка";

#include

- Если имя файла заключено в угловые скобки (<>), считается, что нам нужен некий стандартный заголовочный файл, и компилятор ищет этот файл в предопределенных местах.
- Двойные кавычки означают, что заголовочный файл - пользовательский, и его поиск начинается с того каталога, где находится исходный текст программы.
- Каждая библиотечная функция, определенная стандартом С, имеет прототип в соответствующем заголовочном файле.
- В соответствии со стандартом ANSI имеет 15 заголовочных файлов.

Основные заголовочные файлы стандарта ANSI

stdio.h	Библиотека стандартного ввода/вывода
stdlib.h	Функции общего назначения
string.h	Функции работы со строками символов
time.h	Функции работы с датами и временем
errno.h	Проверка ошибок
float.h	Работа с числами с плавающей точкой
limits.h	Определение размеров циклических типов
locale.h	Поддержка интернациональной среды
math.h	Математическая библиотека
setjmp.h	Возможности нелокальных переходов
signal.h	Обработка сигналов
stdarg.h	Поддержка функций с неопределенным числом параметров

Ввод/вывод данных

Работа с файлами

Ввод/вывод данных

- Стандарт ANSI называется бу феризированным (buffered) или форматированным (formated) вводом/выводом.
- В тоже время используется и другой метод ввода/вывода, UNIX-подобный, или неформатированный (небуферизированный) ввод/вывод.

Стандарт ANSI

- Система ввода/вывода языка C++ поддерживает интерфейс, не зависящий от того, какое в действительности используется физическое устройство ввода/вывода, т.е. есть абстрактный уровень между программистом и физическим устройством. Данная абстракция и называется ***потоком***.
- Способ хранения информации на физическом устройстве называется ***файлом***.

Стандарт ANSI

- Стандарт ANSI языка С связывает каждое из различных устройств (дисковод, клавиатура, терминал, и тд.) с логическим устройством, называемым потоком. Так как потоки не зависят от физических устройств, то одна и та же функция может записывать информацию на диск или выводить ее на экран.
- В языке существует два типа потоков: *те кстовый (text)* и *двоичный (binary).*

Стандарт ANSI

- Текстовый поток - это последовательность символов. При этом может не быть взаимооднозначного соответствия между символами, которые передаются в потоке и выводятся на экран. Среди символов пара может соответствовать возврату каретки или символу табуляции.
- Двоичный поток - это последовательность байтов, которые взаимно - однозначно соответствуют тому, что находится на внешнем устройстве.

Стандарт ANSI

- Поток может быть связан с файлом с помощью оператора открытия файла. Как только файл открыт, то информация может передаваться между ним и вашей программой.
- Все файлы разные по своей сути. Из файла на диске можно выбрать 5-ю запись или заменить 10-ю запись. В то же время в файл, связанный с печатающим устройством, информация может передаваться только последовательно в том же порядке.
- Главное различие между потоками и файлами: все потоки одинаковы, все файлы разные.
- Каждый поток, связанный с файлом, имеет структуру называемую **FILE**.

Консольный ввод/вывод

Консольный ввод/вывод

- К консольному вводу/выводу относятся операции ввода с клавиатуры и вывода на экран.
- Технически функции, осуществляющие эти операции, связывают консоль со стандартными потоками в/в.
- Во многих системах стандартный в/в может быть перенаправлен.
- Для простоты будем предполагать, что стандартный ввод - это ввод с клавиатуры, а стандартный вывод - это вывод на экран.

Функции ввода/вывода

- **getche()** - читает символ с клавиатуры и отображает введенный символ на экране. Прототип в файле **CONIO.H;**
- **putchar()** - выводит символ, который является ее аргументом, на экран в текущую позицию курсора. Прототип в файле **STDIO.H**;
- **getchar()** - читает символ с клавиатуры, но требует нажатия клавиши ENTER. Прототип в файле **STDIO.H**;
- **getch()** - читает символ с клавиатуры, но не выводит символ на экран (без эхо -возврата). Прототип в файле **STDIO.H**;
- **gets()** - ввод строки символов с клавиатуры. Прототип в файле **STDIO.H**;
- **puts()** - вывод строки символов на экран. Прототип в файле **STDIO.H**.

Функции ввода/вывода

- В дополнение к расмотренным функциям кон сольного ввода/вывода, библиотека содержит две функции, которые выполняют форматированный в/в.
- Форматированный в/в означает, что функции могут читать и выводить данные в разном формате, которым можно управлять.
- Функция **printf()** выполняет форматированный вывод данных на консоль, имеет прототип в файле **STDIO.H**.
- Функция **scanf()** выполняет ввод с консоли и автоматически преобразует введенное число в заданный формат. Прототип в файле **STDIO.H**.

Работа с файлами

Работа с файлами

- Библиотека ввода-вывода C++ включает средства для работы с последовательными файлами.
- Логически последовательный файл можно представить как именованную цепочку (ленту, строку) байтов, имеющую начало и конец.
- Последовательный файл отличается от файлов с другой организацией тем, что чтение (или запись) из файла (в файл) ведется байт за байтом от начала к концу.

Работа с файлами

- В каждый момент позиции в файле, откуда выпол няется чтение и куда производится запись, определяются значениями указателей позиций записи и чтения файла (в дальнейшем указатель на файл - **file pointer**).
- Указатель на файл является связующим звеном между файлом и потоком.
- Указатель на файл определяет не только текущую позицию записи (чтения), а также имя файла на диске, структуру типа **FILE**.
- Структура типа файл определена в заголовке **stdio.h**. В этом же файле **stdio.h** определены функции для работы с файлами

Функции для работы с файлами

Функция	Действие функции
fopen()	Открыть файл
fclose()	Закрыть файл
fputc()	Записать символ в файл
fgetc()	Прочитать символ из файла
fseek()	Изменить указатель позиции файла на указанное место
fprintf()	Форматная запись в файл
fscanf()	Форматное чтение из файла
feof()	Возвращает значение TRUE, если достигнут конец файла
ferror()	Возвращает значение FALSE, если обнаружена ошибка

Функции для работы с файлами

Функция	Действие функции
fread()	Читает блок данных из потока
fwrite()	Пишет блок данных в поток
rewind()	Устанавливает указатель позиции файла на начало
remove()	Уничтожает файл

Открытие и закрытие файла. Функции FOPEN() и FCLOSE()

Открытие и закрытие файла

- Перед началом работы с файлом его надо создать (открыть), а по окончании работы закрыть.
- Перед началом работы с файлом надо создать указатель на структуру данных типа **FILE.**
- Затем необходимо вызвать функцию **fopen(),** которая может создать новый файл для записи в него, либо открыть существующий на диске файл для записи или (и) чтения.
- После вызова этой функции создается структура типа **FILE** и указатель **f** содержит адрес начала этой структуры в памяти. Кроме того, в ОП отводится 512 байт для обмена данными между файлом на диске и программой. Этот массив называется буфером.
- При закрытии файла, память отведенная под эту структуру и буфер, обнуляется, указатель **f** также обнуляется, это означает, что указатель **f** ни на что не указывает.

Функция **fopen()** - функция открытия файла

- Функция **fopen()** - функция открытия файла.

Формат:

fopen(<имя файла>,<аргумент>),

где аргумент - символьная константа, определяющая режим открытия файла.

Функция **fopen()** - функция открытия файла

Режимы открытия файла:

- **r** - чтение;
- **w** - запись;
- **a** - добавление;
- **r+** - чтение и запись с обновлением;
- **w+** - запись с обновлением;
- **a+** - чтение и добавление;
- и др.

Пример

```
FILE *out;
out=fopen(“q1","w");
```

Примечание:

- Служебное слово **FILE** записывается заглавными буквами;
- Аргументы, определяющие режим доступа в функции **fopen()** записываются только маленькими буквами.

Пример

```
// Пример - Открытие (создание) нового файла
 #include <stdio.h>
main ()
{
int n;
FILE *f;
f=fopen("test.dat","w");
fclose(f);
}
```

Запись и чтение символа

Запись и чтение символа

Для побайтной записи в файл используется функция **fputc().**

Формат:

fputc(<переменная>,<имя файла>);

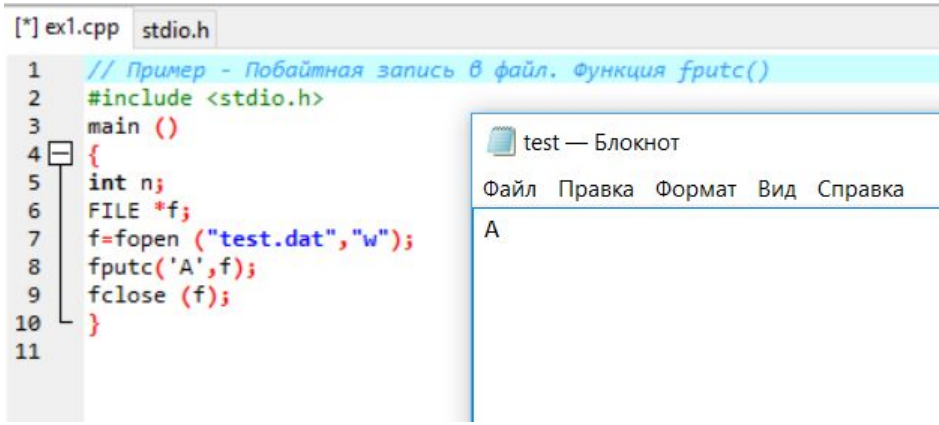
где:

переменная - символьная переменная, значение которой записывается в файл;

имя файла - файл, в который выполняется запись символа.

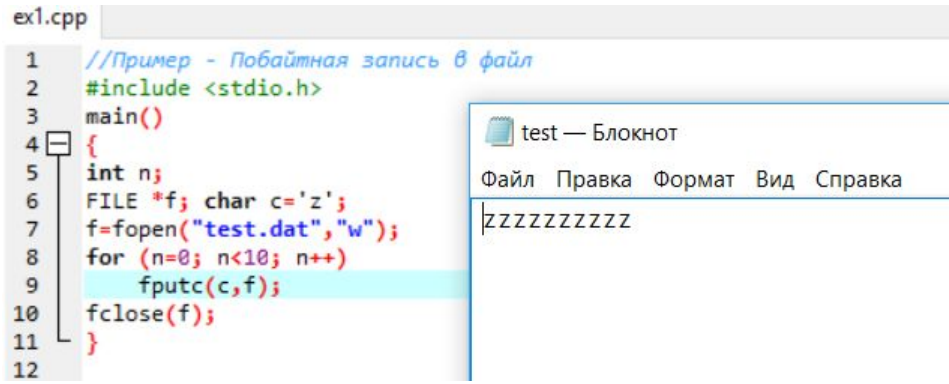
Пример

```
// Пример - Побайтная запись в файл. Функция fputc()
#include <stdio.h>
main ()
{
int n;
FILE *f;
f=fopen ("test.dat","w");
fputc('A',f);
fclose (f);
}
```



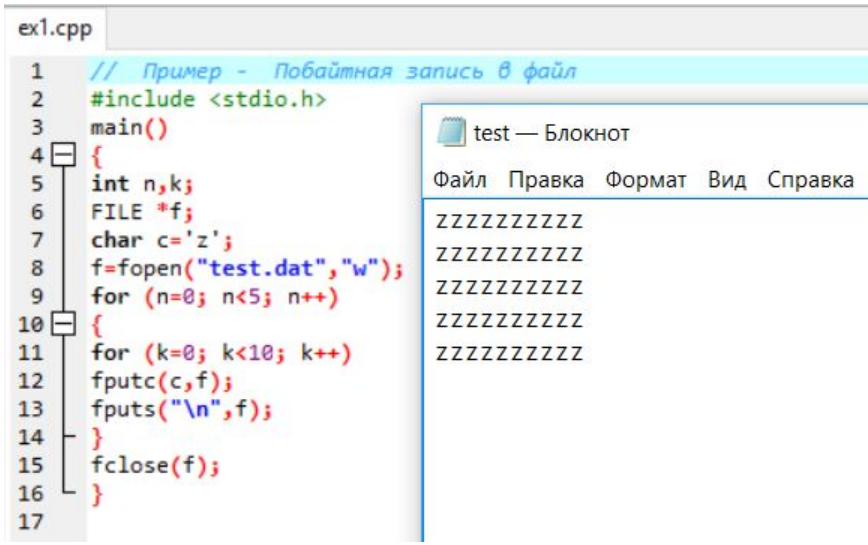
Пример

```
/ /Пример 31. Побайтная запись в файл
#include <stdio.h>
main()
{
int n;
FILE *f; char c=‘z’;
f=fopen(“test.dat”,”w”);
for (n=0; n<10; n++)
        fputc(c,f);
fclose(f);
}
```



Пример

```
//  Пример 32.  Побайтная запись в файл
#include <stdio.h>
main()
{
int n,k;
FILE *f;
char c='z';
f=fopen("test.dat","w");
for (n=0; n<5; n++)
{  for (k=0; k<10; k++)
fputc(c,f);
fputs("\n",f); }
fclose(f);
}
```



Примечание

В программе используется функция **fputs()**, которая записывает в файл последовательность символов - **\n** (функция **fputc()** записывает один символ).

Управляющий символ **\n** при записи в файл превращается в два символа:

- символ возврата каретки **(код 0D)**;
- символ перевода строки **(код 0A)**.

Запись и чтение символа

- Чтение символа из файла. Функция **FGETC()**
- Формат:

fgetc(<имя файла>);

Определение конца файла

- В операционной системе MS-DOS принято соглашение: признаком конца текстового файла является символ

  CTRL-Z (код 26).
- Т.о., если из файла будет прочитан байт, в котором хранится число 26, то функция **fgetc()** примет значение (-1), что означает конец файла.
- В файле **stdio.h** объявлена константа **EOF**, равная (-1).

Пример

```
/* Пример - Побайтное чтение из файла. */
include <stdio.h>
main()
{
int c;
FILE *f;
clrscr();
f=fopen(“test.dat”,”r”);
printf("Результаты чтения данных из файла test.dat:\n”);
while ((c=fgetc(f)) !=EOF)
      printf(“%c”,c);
fclose(f);
}
```