

**Function** → a block of statements grouped together, to perform some specific task

Each program written in C / C++ includes at least one function with pre-defined name: **main()**. In our previous programs, we used many other predefined functions, for example: **printf(...)**, **scanf(...)**, **getchar(...)**, **rand(...)**, **sin(...)**.

You can also define a whole list of new – your own functions.

**The syntax for function definition:**

```
the_name_of_returned_type FUNCTION_NAME ( parameter list )  
{  
    single instruction or sequence of instructions ;  
}
```

example:

```
int MAX ( int number_1 , int number_2 )  
{  
    if( number_1 > number_2 )  
        return number_1 ;  
    else  
        return number_1 ;  
}
```

⇒ parameter list may be empty,  
or includes detailed description of the parameter name and type  
(separated by commas):

main( )      main( **void** )      main( **int** argc , **char\*** argv[ ] )

⇒ parameters are defined very much like a new variables,  
(Note: You can not group the sequence of parameters of the same type):

**int** MAX ( ~~**int** number\_1, number\_2, number\_3~~ )      ← bad !  
**int** MAX ( **int** number\_1, **int** number\_2, **int** number\_3 )      ← good

⇒ the „body” function is included within the brackets: { ... }

⇒ function ends when it encounters command: **return**  
or after execution of all instructions contained in the body of the function,

⇒ if the function is of type **void**, we use only the separate word **return**,

⇒ if the function is of the type **not-void**, then just after keyword **return**  
there must appear value or expression of the appropriate type

eg.:      **return** result\_value;      *or*      **return** ( value\*5 – 12 ) ;

### 1) prog. without sub-functions

```
#include <iostream>

int a,b,c,sum;
float average;

int main(void)
{
    //--- load data ---
    printf( "Enter three numbers: " );
    scanf( "%d%d%d", &a, &b, &c );

    //--- calculate formulas ---
    sum = a+b+c;
    average = sum / 3.0;

    //--- visualise results ---
    printf( "Sum = %d\n", sum );
    printf( "Avg = %f\n", average );

    //--- exit the program ---
    printf( "\n\nPress ENTER" );
    fflush( stdin );
    getchar();
    return 0;
}
```

### 2) parameter-less functions

```
#include <iostream>

int a,b,c,suma;
float srednia;

void LOAD_DATA (void)
{
    printf( "Enter three numbers: " );
    scanf( "%d%d%d", &a, &b, &c );
}

void CALCULATE (void)
{
    sum = a+b+c;
    average = sum / 3.0;
}

void PRINT_RESULTS(void)
{
    printf( "Sum = %d\n", sum );
    printf( "Avg = %f\n", average );
}

int main(void)
{
    LOAD_DATA ();
    CALCULATE();
    PRINT_RESULTS ();

    printf( "\n\nPress ENTER" );
    fflush( stdin );  getchar();
    return 0;
}
```

### 3) func. with explicit list of parameters

```
#include <iostream>

void LOAD_DATA(int& x, int& y, int&z)
{
    printf( "Enter three numbers: " );
    scanf( "%d%d%d", &x, &y, &z );
}

void SUM(int a,int b,int c, int& s)
{ s = a+b+c; }

float CALCULATE_AVG(int x, int y, int z)
{
    int sum = x+y+z;
    // or alternatively → SUM( x, y, z, sum );
    return sum/3.0;
}

void PRINT(int sum, float avg)
{
    printf( "Sum = %d\n", sum );
    printf( "Avg = %f\n", average );
}

int main(void)
{
    int a,b,c,sum;
    float average;

    LOAD_DATA(a,b,c);
    SUM(a,b,c,sum);
    average =CALCULATE_AVG(a,b,c);
    PRINT(sum,average);
    return 0;
}
```

## Definition ↔ Call of the Function ↔ Prototype

**Function prototype** → "anticipating" declaration, specifies only the function name and the types of returned value and given parameters  
(only **function header** completed with a semi-colon)

Such a function declaration is necessary in cases, where the function call is earlier than its definition. eg.

```
// Program calculating a maximum of three numbers, by calling the function MAX
```

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

```
// implementation in C
```

```
int MAX ( int , int );
```

```
// Prototype – declaration of the MAX function
```

```
int main( void )
```

```
{
```

```
    int a , b , c , m. ;
```

```
    printf( " Enter the value: A = " );
```

```
    scanf( " %d " , &a );
```

```
    printf( " Enter the value: B = " );
```

```
    scanf( " %d " , &b );
```

```
    printf( " Enter the value: C = " );
```

```
    scanf( " %d " , &c );
```

```
    m = MAX( a , b );
```

```
// Call of function MAX
```

```
    printf( " \n\n Maximum of A and B equals = %d " , m );
```

```
    printf( " \n\n Maximum of B and C equals = %d " , MAX( b,c ) );
```

```
    printf( " \n\n Maximum of A,B,C equals = %d " , MAX( a, MAX(b,c) ) );
```

```
    fflush();
```

```
    getchar();
```

```
    return 0;
```

```
}
```

```
int MAX ( int number_1, int number_2 )
```

```
// Definition of function MAX
```

```
{
```

```
    if(number_1 > number_2 )
```

```
        return number_1 ;
```

```
    else
```

```
        return number_2 ;
```

```
}
```

## FUNCTIONS / PARAMETERS PASSING

### 1. Parameterless function – not returning any value – procedure (?)

```
void function_name (void)
{
    ...
    return; // causes an immediate termination of the function,
           // can be omitted (while at the end of function body)
}
```

#### example

```
void inverse_value (void)
{
    // calculate the inverse of the number of loaded from keyboard
    double number;
    scanf( "%lf" , &number );
    if( number == 0 )
        return;
    printf( "%f" , 1/number );
    return; // this «return» can be omitted
}
```

### 2. Function receiving a list of parameters and returning some result value

**NOTE !** in C language, parameters are generally passed **by value**  
ie. execution of function call creates new variables (local / parameters)  
the content of which is initialized by values of arguments  
(constants, variables or expressions) given in calling instruction

#### example a)

```
double inverse ( double number ) // definition of function «inverse»
{
    if( number == 0 )
        return( 0 );
    else
        return( 1/number );
}

int main( void ) {
    double x=10, y;
    y = inverse( 20 ); // example calls of function «inverse»
    y = inverse( x );
    y = inverse( 3*(15-x) );
    return 0;
}
```

#### example b)

```
// example of a function "maximum", that returns a value of a bigger number
double maximum( double a, double b )
{
    if( a > b)
        return( a );
    return( b );
}
```

#### example c)

```
void sort_1 ( double a, double b )
{
    double buf;
    if( a > b)
    {
        buf = a;
        a = b;
        b = buf;
    }
}

int main( void )
{
    double x=7, y=5;
    sort_1( x, y );
    return 0;
}
```

*// WARNING !!!  
// wrong way of passing parameters  
// (by value)  
// There are sorted values of local variables **a** , **b**  
// (copies of arguments: **x** and **y**).  
// The content of **x** and **y** will not change !*

*// arguments: 7, 5; actual values of parameters are passed*

#### example d)

```
void sort_2 ( double *a, double *b )
{
    double buf;
    if( *a > *b)
    {
        buf = *a;
        *a = *b;
        *b = buf;
    }
}

int main( void ) {
    double x=7, y=5;
    sort_2( &x, &y );
    return 0;
}
```

*// passing variables „by address”  
// comparison of values (from variables **x**, **y**)  
// indicated by pointers **a** and **b***

*// passing the addresses of variables (not values)*

In C++ parametrs can be passed by value or by reference

reference type → variables of this type do not occupy a new place in memory, they are used to represent the other variables in the program.

**type\_name** variable\_name; ← creation of standard variable

**type\_name &** reference\_name = variable\_name;

*(this is a definition of an alias → alternative name for the same variable)*

example

```
int height;
int average_height = height;
int& size = height;           // creation of reference « size »
                              // related to the same memory area
                              // which is assigned to « height »

size = size + 1;              // equivalent to: height = height + 1
```

example e)

```
void sort_3 ( double & a, double & b )
{
    double buf;                // passing parameters (variables)
    if( a > b )                  // by reference
    {
        buf = a;                // a and b are reference names for x , y
        a = b;
        b = buf;
    }
}

int main( void )
{
    double x=7, y=5;
    sort_3( x, y );             // actual arguments x , y → initialize parameters a,b
    return 0;
}
```

## Additional examples

### 1) arguments by „value”

```
int FunVal1(int a) //receive „value”
{
    a = a+1;
    return a;    //return a „value”
}
```

```
int FunVal2(int a, int b)
{
    if( a > b )
        return a;    //return a „value”
    else
        return b;
}
```

```
int main(void)
{
    int x=0, y=2, z;
    z = FunVal1( x );    //x==0, z==1
    z = FunVal2( x, y );    //z==2
    FunVal2( x, y ) = 101; //2==101?
    return 0;
}
```

### 2) arguments by „pointer”

```
int FunAdr1(int* pa) //receive „pointer”
{
    *pa = *pa+1;
    return *pa;    //return a „value”
}
```

```
int* FunAdr2(int* pa, int* pb)
{
    if( *pa > *pb )
        return pa;    //return a „pointer”
    else
        return pb;
}
```

```
int main(void)
{
    int x=0, y=2, z;
    z = FunAdr1( &x );    //x==1, z==1
    z = *FunAdr2( &x, &y );    //z==2
    *FunAdr2( &x, &y ) = 102; //y=102 !
    return 0;
}
```

### 3) arguments by „reference”

```
int FunRef1(int &ra) //receive a „reference”
{
    ra = ra+1;
    return ra;    //return a „value”
}
```

```
int& FunRef2(int &ra, int &rb)
{
    if( ra > rb )
        return ra;    //return a „reference”
    else
        return rb;
}
```

```
int main(void)
{
    int x=0, y=2, z;
    z = FunRef1( x );    //x==1, z==1
    z = FunRef2( x, y );    //z==2
    FunRef2( x, y ) = 103;    //y=103 !
    return 0;
}
```