Funkcja → wysoce niezależny blok definicji i instrukcji programu (podprogram)

Każdy program napisany w języku C/C++ zawiera przynajmniej jedną funkcję o predefiniowanej nazwie: **main( )**. Najczęściej wykorzystuje się również wiele innych predefiniowanych funkcji np. **printf(...)**, **scanf(...)**, **abs(...)**, **sin(...)**, itp. Można również definiować nowe–własne funkcje.

Składnia definicji funkcji:

```
zwracany_typ  NAZWA_FUNKCJI (  lista parametrów  )
{
   instrukcja lub sekwencja instrukcji ;
}
```

przykład:

```
int  MAX (  int liczba_1 ,  int liczba_2  )
{
   if( liczba_1 > liczba_2 )
      return liczba_1 ;
   else
      return liczba_2 ;
}
```

⇒ lista parametrów może być pusta lub zawierać opisy kolejnych parametrów (pooddzielane przecinkami):

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

⇒ parametry definiowane są tak jak zmienne. Uwaga: nie można grupować sekwencji parametrów tego samego typu:

int MAX (~~**int** liczba_1, liczba_2~~) ← źle !

⇒ „ciało" funkcji jest zawarte pomiędzy nawiasami: **{ ... }** (bez średnika na końcu)

⇒ działanie funkcji kończy się po napotkaniu polecenia **return** lub po wykonaniu sekwencji wszystkich instrukcji zawartych w ciele funkcji,

⇒ jeżeli funkcja jest typu **void**, to używamy samego słowa **return**, bez żadnego wyrażenia po nim,

⇒ jeżeli funkcja jest typu innego niż **void** to po poleceniu return **musi** się pojawić wyrażenie odpowiedniego typu (może być w nawiasach), np.:

return liczba**_1;** *lub* **return**(liczba_1) **;**

Prototyp funkcji → deklaracja „uprzedzająca”, określa tylko nazwę funkcji oraz typy zwracanej wartości i parametrów (sam nagłówek funkcji zakończony średnikiem)

Deklaracja funkcji jest konieczna w przypadkach, gdy wywołanie funkcji występuje wcześniej niż jej definicja. Np.

```
// program wyznaczający maksimum 3 liczb poprzez wywołanie funkcji MAX

#include <stdio.h>

int  MAX (  int ,  int ) ;                    //  Prototyp - deklaracja funkcji MAX


void main( void )
{
   int a , b , c , m. ;
   printf( " Podaj liczbe A = " );
   scanf( " %d " , &a );
   printf( " Podaj liczbe B = " );
   scanf( " %d " , &b );
   printf( " Podaj liczbe C = " );
   scanf( " %d " , &c );

   m = MAX( a , b );                                  // Wywolanie funkcji MAX

   printf( " \n\nMaksimum z liczb A i B rowna sie = %d " , m ) ;

   printf( " \n\nMaksimum z liczb B i C rowna sie = %d " , MAX( b,c ) ) ;

   printf( " \n\nMaksimum z A,B,C rowna sie = %d " ,  MAX( a, MAX(b,c) ) ) ;

   flushall();
   getchar();
}



int  MAX (  int liczba_1,  int liczba_2 )          // Definicja funkcji MAX
{
   if( liczba_1 > liczba_2 )
      return liczba_1 ;
   else
      return liczba_2 ;
}
```

FUNKCJE / PRZEKAZYWANIE PARAMETRÓW

1. Funkcja bezparametrowa nie zwracająca żadnej wartości

```
void nazwa_funkcji(void)
{
   • • •
   return;  // powoduje natychmiastowe zakończenie wykonywania funkcji
}           // na końcu funkcji można pominąć
```

przykład

```
void odwrotność(void)
{                          // obliczenie odwrotności liczby wczytanej z klawiatury
   double liczba;
   scanf( "%lf" , &liczba );
   if( liczba == 0 )
      return;
   printf( "%f" , 1/liczba );
   return;    // to «return» można pominąć
}
```

2. Funkcja pobierająca parametr i zwracająca wartość

UWAGA ! w języku C parametry przekazywane są tylko **przez wartość**

tzn. po wywołaniu funkcji tworzone są nowe zmienne (lokalne), których zawartość inicjowana jest wartościami parametrów (zmiennych, stałych lub wyrażeń) podanych przy wywołaniu.

przykład a)

```
double odwrotność( double liczba )        // definicja funkcji «odwrotność»
{
   if( liczba == 0 )
      return( 0 );
   else
      return( 1/liczba );
}

void main( void )
{
   double x=10, y;
   y = odwrotnosc( 20 );      // przykłady wywoływania funkcji «odwrotnosc»
   y = odwrotnosc( x );
   odwrotnosc(  3*(15-x)  );
}
```

przykład b)

```
// przykład funkcji zwracającej wartość większego z argumentów
double maksimum( double a, double b )
{
   if( a > b)
      return( a );
   return( b );
}
```

przykład c)

```
void posortuj_1 ( double a, double b )
{                                                  // UWAGA !!!
   double buf;                          // błędny sposób przekazywania
   if( a > b)                           // paramerów (przez wartość).
      {                                 // Sortowane są zmienne lokalne a i b
         buf = a;                       // (kopie parametrów x i y).
         a = b;                         // Zawartość x i y nie ulegnie zmianie !
         b = buf;
      }
}

void main( void )
{
   double x=7, y=5;
   posortuj_1( x, y );   // do funkcji przekazywane są wartości zmiennych
}
```

przykład d)

```
void posortuj_2 ( double *a, double *b )
{                                  // przekazywanie parametrów „przez adres”
   double buf;
   if( *a > *b)                    // porównywane są zawartości miejsc
      {                            // wskazywanych przez wskazniki a i b
         buf = *a;
         *a = *b;
         *b = buf;
      }
}

void main( void )
{
   double x=7, y=5;
   posortuj_2( &x, &y );  //do funkcji przekazywane są adresy zmiennych
}
```

W języku **C++** parametry mogą być przekazywane **przez wartość** lub **przez referencję**

(przekazywanie przez referencję jest odpowiednikiem przekazywania przez zmienną)

Typ referencyjny → zmienne tego typu nie zajmują nowego miejsca w pamięci, służą do reprezentacji innych zmiennych w programie.

nazwa_typu nazwa_zmiennej; ← utworzenie zwykłej zmiennej

nazwa_typu & nazwa_zmiennej_referencyjnej = nazwa_zmiennej;

(jest to zdefiniowanie aliasu – innej nazwy dla tej samej zmiennej)

przykład

```
int  wzrost;
int  sredni_wzrost = wzrost;
int&  wysokosc = wzrost;          // utworzenie zmiennej referencyjnej
                                  // związanej z tym samym obszarem
                                  // pamięci co wzrost
wysokosc = wysokosc + 1;          // równoważne:  wzrost = wzrost + 1
```

przykład e)

```
void posortuj_3 ( double & a, double & b )
{
   double buf;                    // przekazywanie parametrów
   if( a > b)                     // przez referencję
      {
         buf = a;                 // a i b są referencyjnymi nazwami x i y
         a = b;
         b = buf;
      }
}


void main( void )
{
   double x=7, y=5;
   posortuj_3( x, y );       // parametry x i y inicjują zmienne referencyjne
}
```

FUNKCJE OPERUJĄCE NA PAMIĘCI <mem.h>

void ∗memset (**void** ∗wsk_pocz, **int** wartosc, **size_t** dlugosc)

*(obszar wskazywany przez **wsk_pocz** o długości **dlugosc** jest wypełniany wartością **wartosc**)*

np.
```
int i, tab[1000];
memset( &i , 0, sizeof( i ) );          // równoważne: i = 0
memset( &i , 1, sizeof( i ) );          // równoważne: i = 257 = 1∗256 + 1
memset( tab , 0, sizeof( tab ) );       // wypełnienie tablicy zerami
```

---

void ∗memcpy (**void** ∗wsk_dokąd, **void** ∗wsk_skąd, **size_t** dlugosc)

*(„**mem**ory **c**o**py**" → skopiowanie **dlugosc** bajtów spod adresu **wsk skąd** pod adres **wsk dokąd**)*

np.
```
int i, j=10, tab1[1000], tab2[1000];
memcpy( &i , &j, sizeof( i ) );          // równoważne: i = j ;
memcpy( tab1 , tab2, sizeof( tab1 ) );   // skopiowanie zawartości
                                         // tablicy tab2 do tab1
```

---

int memcmp (**void** ∗obszar_1, **void** ∗obszar_2, **size_t** dlugosc)

*(„**mem**ory **c**o**mp**are" → porównanie **dlugosc** bajtów spod adresu **obszar 1** oraz adresu **obszar 2**)*

funkcja zwraca wartość:
- < 0 gdy zawartość obszar_1 < obszar_2
- = 0 gdy zawartość obszar_1 == obszar_2
- > 0 gdy zawartość obszar_1 > obszar_2

np.
```
int i, j, tab1[1000], tab2[1000];
if( memcmp( &i , &j, sizeof( int ) ) )   // równoważne porównaniu i != j
   printf( "te zmienne mają rozna wartosc" );
memcmp( tab1 , tab2, sizeof( tab1 ) );   // porównanie zawartości
                                         // tablicy tab1 oraz tab2
```

---

void ∗memmove (**void** ∗wsk_dokąd, **void** ∗wsk_skąd, **size_t** dlugosc)

*(„**mem**ory **move**" → kopiowanie ze sprawdzaniem „zachodzenia się" obszarów)*

---

void ∗memccpy (**void** ∗dokąd, **void** ∗skąd, **int** znak, **size_t** dlugosc)

*(„**mem**ory **c**har **c**o**py**" → kopiowanie ograniczone ilością bajtów lub skopiowaniem znaku)*

---

void ∗memchr (**void** ∗wsk_pocz, **int** znak, **size_t** dlugosc)

*(„**mem**ory **ch**ar search" → szukanie pozycji wystąpienia bajtu o zadanej wartości)*

PRZYKŁADY WYKORZYSTANIA FUNKCJI „MEM”

```
// załóżmy następującą definicję tablicy:
   long tab[11] = { –5, –4, –3, –2, –1, 0, 1, 2, 3, 4, 5 } ;
```

```
// po operacji:
   memcpy( &tab[ 0 ], &tab[ 5 ], sizeof(long) );
// lub:
   memcpy( tab + 0,  tab + 5, sizeof(long) ) ;
// zawartość tablicy jest równa: { 0 , –4, –3, –2, –1, 0, 1, 2, 3, 4, 5 }
```

```
// po operacji:
   memcpy( tab + 0,  tab + 6 ,  5 * sizeof(long) ) ;
// zawartość tablicy jest równa: { 1, 2, 3, 4, 5 , 0, 1, 2, 3, 4, 5 }
```

```
// po operacji:   ( ← ← ← )
   memcpy( tab + 0,  tab + 1 ,  10 * sizeof(long) ) ;
// zawartość tablicy jest równa: { –4, –3, –2, –1, 0, 1, 2, 3, 4, 5 , 5 }
```

```
// po operacji:   ( → → → )
   memcpy( tab + 1,  tab + 0 ,  10 * sizeof(long) ) ;
// zawartość tablicy jest równa: { –5, –5, –5, –5, –5, –5, –5, –5, –5, –5, –5 }
```

```
// po operacji:   ( → → → )
   memmove( tab + 1,  tab + 0 ,  10 * sizeof(long) ) ;
// zawartość tablicy jest równa: { –5, –5, –4, –3, –2, –1, 0, 1, 2, 3, 4 }
```

```
// skopiowanie zawartości tablicy A do tablicy B :
   long  A[ 100 ],  B[ 100 ] ;
// poprzez operację:
   memcpy( B,  A ,  100 * sizeof(long) ) ;
// lub:
   memcpy( B,  A, sizeof( B ) ) ;    // lub:    memcpy( B,  A, sizeof( A ) ) ;
```

```
// UWAGA !!! przy kopiowaniu zawartości tablicy, która jest parametrem funkcji :
void funkcja( long  A[ 100 ] )
{
   long B[ 100 ] ;
   memcpy( B,  A , ~~sizeof( A )~~ ) ;   // ŹLE !!! bo A jest zmienną zawierającą adres,
                                         //     sizeof( A ) jest równe 2 (zamiast 400)
   memcpy( B,  A , sizeof( B ) ) ;       // ← dobrze
}
```