Funkcje

W matematyce pod pojęciem funkcji rozumiemy twór, który pobiera pewną liczbę argumentów i zwraca wynik. Jeśli dla przykładu weźmiemy funkcję *sin(x)* to *x* będzie zmienną rzeczywistą, która określa kąt, a w rezultacie otrzymamy inną liczbę rzeczywistą - sinus tego kąta.

W C **funkcja** (czasami nazywana podprogramem lub procedurą) to wydzielona część programu, która przetwarza argumenty i ewentualnie zwraca wartość, która następnie może być wykorzystana jako argument w innych działaniach lub funkcjach. Funkcja może posiadać własne zmienne lokalne. W odróżnieniu od funkcji matematycznych, funkcje w C mogą zwracać dla tych samych argumentów różne wartości.

Z poprzednich instrukcji zapewne pamiętasz takie funkcje jak:

- funkcja printf(), drukująca tekst na ekranie,
- funkcja scanf(), umożliwiająca pobieranie danych od użytkownika,
- funkcja getChar(), umożliwiająca odczytanie/pobranie znaku,
- funkcja main(), czyli główna funkcja programu (funkcja uruchomieniowa).

Główną motywacją tworzenia funkcji jest unikanie powtarzania kilka razy tego samego kodu.

Przykład

```
for(i=1; i <= 5; ++i) {
  printf("%d ", i*i);
}
for(i=1; i <= 5; ++i) {
  printf("%d ", i*i*i);
}
for(i=1; i <= 5; ++i) {
  printf("%d ", i*i);
}
```

W powyższym przykładzie widzimy, że pierwsza i trzecia pętla *for* są takie same. Zamiast kopiować fragment kodu kilka razy (co jest mało wygodne i może powodować błędy) lepszym rozwiązaniem mogłoby być wydzielenie tego fragmentu tak, by można go było wywoływać kilka razy. Tak właśnie działają funkcje.

Innym, nie mniej ważnym powodem używania funkcji jest rozbicie programu na fragmenty wg ich funkcjonalności. Oznacza to, że jeden duży program dzieli się na mniejsze funkcje, które są "wyspecjalizowane" w wykonywaniu określonych czynności. Dzięki temu łatwiej jest zlokalizować błąd. Ponadto takie funkcje można potem przenieść do innych programów.

Postać ogólna funkcji

Ogólna postać funkcji wygląda następująco:

```
typ identyfikator (typ1 argument1, typ2 argument2, typ_n argument_n)
{
  /* instrukcje */
}
```

Każda funkcja składa się z:

- nagłówka funkcji (linia 1) zbudowanego z:
  - typu danych zwracanych przez funkcję,

  - identyfikatora (nazwy) funkcji,
  - argumentów umieszczonych w nawiasach okrągłych,
- ciała funkcji (bloku).

Oczywiście istnieje możliwość utworzenia funkcji, która nie posiada żadnych argumentów. Definiuje się ją tak samo, jak funkcję z argumentami z tą tylko różnicą, że między okrągłymi nawiasami nie znajduje się żaden argument lub pojedyncze słówko *void* - w definicji funkcji nie ma to znaczenia, jednak w deklaracji puste nawiasy oznaczają, że prototyp nie informuje jakie argumenty przyjmuje funkcja, dlatego bezpieczniej jest stosować słówko *void*.

Funkcje definiuje się poza główną funkcją programu (*main*).

Przykłady

```c
int suma(int a, int b)
{
    int wynik = a+b;
    return wynik;
}

int kwadrat(int x)
{
    return x*x;
}

int min(int a, int b)
{
    int wynik;
    if(a<b)
        wynik = a;
    else
        wynik = b;
    return wynik;
}

double jeden(void)
{
    return 1.0;
}
```

W powyższych przykładach funkcja *suma()* pobiera 2 argumenty: *int a, int b* oraz wylicza sumę podanych argumentów i przypisuje do zmiennej *wynik*. Następnie otrzymany rezultat zostaje zwrócony za pomocą instrukcji *return*.

Z kolei funkcja *kwadrat()* pobiera 1 argument, a następnie od razu zwraca wynik działania *x*x*, bez przypisywania go do dodatkowej zmiennej.

Nieco bardziej rozbudowana jest funkcja *min()*, która wyznacza minimum spośród 2 liczb całkowitych. W ciele tej metody użyta została instrukcja warunkowa, która w zależności od podanych argumentów funkcji zwraca wartość jednego z tych argumentów (wcześniej przypisaną do zmiennej *wynik*).

Funkcja *jeden()* nie pobiera żadnych argumentów. Jej zadaniem jest zwracanie wartości *1.0* (typ *double*).

Definiowanie i deklarowanie funkcji

Deklaracja funkcji (zwana czasem prototypem) to inaczej ‘poinformowanie’ programu o tym, że za chwilę pojawi się funkcja przyjmująca parametry (argumenty) określonych typów i zwracająca wynik danego typu. Deklaracji funkcji zwykle dokonujemy na początku programu, po sekcji dyrektyw preprocesora.

Z kolei definicja funkcji to de facto utworzenie funkcji przyjmującej konkretne argumenty oraz zwracającej konkretne rezultaty.

Przykład

```c
#include<stdio.h>

// deklaracja funkcji
int suma(int, int);

int main()
{
    printf("suma 1 + 2 = %d\n", suma(1,2)); // przyklad uzycia funkcji
    return 0;
}

// definicja funkcji
int suma(int a, int b)
{
    int wynik = a+b;
    return wynik;
}
```

Deklaracja może zawierać tylko typy przyjmowanych argumentów (w nawiasach okrągłych) – jak w powyższym przykładzie, lub typy i nazwy argumentów.

W praktyce jednak można nie stosować kodu deklaracji funkcji, ale wówczas definicja funkcji musi pojawić się przed funkcją, w której następuje użycie definiowanej funkcji.

Przykład

```c
#include<stdio.h>

// definicja funkcji
int suma(int a, int b)
{
    int wynik = a+b;
    return wynik;
}

int main()
{
    printf("suma 1 + 2 = %d\n", suma(1,2));
    return 0;
}
```

Rozważmy teraz następujący kod:

```c
int a()
{
  return b(0);
}

int b(int p)
{
  if( p == 0 )
    return 1;
  else
    return a();
}

int main()
{
  return b(1);
}
```

Funkcja *a()* zwraca wynik wywołania (użycia) funkcji *b()* z argumentem *0*. Z kolei funkcja *b()* zwraca wywołanie funkcji *a()* jeśli p!=0. W tym przypadku nie jesteśmy w stanie zamienić funkcji a i b

kolejnością, gdyż obie funkcje korzystają z siebie nawzajem. W tym przypadku konieczne jest wcześniejsze zadeklarowanie funkcji.

Wywołanie funkcji, przekazywanie parametrów, zasięg zmiennych

Wywołanie funkcji to inaczej jej użycie z konkretnymi argumentami w ciele innej funkcji, np. *main()*. Ogólna postać wywołania wygląda następująco:

```
identyfikator (argument1, argument2, argumentn);
```

Jeśli chcemy przypisać do zmiennej wartość, którą zwraca funkcja należy użyć następującej formy:

```
zmienna = funkcja (argument1, argument2, argumentn);
```

Przykłady

```c
#include<stdio.h>
double sum(double a, double b)
{
    return a+b;
}

int get_10(void)
{
    return 10;
}

char ascii(int ch)
{
    return (char)ch; // rzutowanie int na char
}

int main()
{
    double r1 = sum(1.092, 3.45);
    int r2 = get_10();
    char r3 = ascii(65);
    printf("%g\n%d\n%c\n", r1, r2, r3);
    return 0;
}
```

```c
#include<stdio.h>
double sum(double a, double b)
{
    return a+b;
}

int get_10(void)
{
    return 10;
}

char ascii(int ch)
{
    return (char)ch; // rzutowanie int na char
}

int main()
{
    printf("%g\n%d\n%c\n", sum(1.092, 3.45), get_10(), ascii(65));
    return 0;
}
```

Powyższe przykłady dają takie same rezultaty, chociaż kod po prawej stronie jest nieco krótszy – funkcja *main()*. Została tam użyta (wywołana) funkcja *printf()*, wewnątrz której wywołane zostały 3 funkcje: *sum()*, *get_10()* i *ascii()*. Jest to przykład obrazujący, że wywołanie jednej funkcji może być parametrem drugiej.

Rozważmy teraz przykład:

```c
#include<stdio.h>
int sum(int a, int b)
{
    return a+b;
}

int get_10(void)
{
    return 10;
}

char ascii(int ch)
{
    return (char)ch;
}

int main()
{
    printf("%c\n", ascii(sum(87, get_10())));
    return 0;
}
```

W funkcji *main()* pojawia się rozbudowane wywołanie funkcji *printf()*, w którym:

- wywołana została funkcja *ascii()*, a jej parametrem jest wynik zwrócony przez wywołanie funkcji *sum()*
- funkcja *sum()* z kolei przyjmuje 2 parametry: liczbę *87* oraz wynik wywołania funkcji *get_10()*

Zatem aby mogła wykonać się funkcja *printf()*, to:

- najpierw musi nastąpić wywołanie funkcji *get_10()* – zwróci *10*
- dalej wywołana zostaje funkcja *sum()* – zwróci *87+10=97*
- następnie funkcja *ascii()* – zwróci *char(97)='a'*
- i finalnie *printf()* – wypisze *„a"*

Spójrzmy na jeszcze jeden przykład:

```c
#include<stdio.h>
void fun(int a)
{
    a++;
    printf("Wartosc a wewnatrz funkcji fun(): %d\n", a);
}

int main()
{
    int a = 10;
    fun(a);
    printf("Wartosc a poza funkcja fun(): %d\n", a);
    return 0;
}
```

Program ten wypisuje następujące wyniki:

```
Wartosc a wewnatrz funkcji fun(): 11
Wartosc a poza funkcja fun(): 10
```

Być może zastanawiasz się dlaczego zmienna *a* wypisana w *main()* ma wartość *10*, podczas gdy w *fun()* wynosi ona *11*. Przecież przy deklaracji *a=10*, następnie *a* zostaje przekazane jak parametr do *fun()* i tam zostaje zinkrementowane. Wydaje się, że *a* wypisywane w *main()* powinno wynosić *11*.

Otóż nie. Zmienna *a* zadeklarowana w funkcji *main()* ma wartość *10* i w żaden sposób wartość tej zmiennej nie zostaje zmieniona w tej funkcji. Wartość zmiennej *a* z *main()* zostaje też co prawda przekazana jako parametr do funkcji *fun()*, ale *a* z funkcji *fun()* to zupełnie inna zmienna, chociaż ma tę samą nazwę.

Parametr funkcji *fun()* – zmienna *a* – widoczna jest tylko w tej funkcji. Jest to zmienna lokalna tej funkcji i nie można jej używać poza tą funkcją.

Aby to lepiej zrozumieć rozważmy ten przykład, ale zmieńmy nazwę parametru funkcji *fun()*.

```c
#include<stdio.h>
void fun(int b)
{
    b++;
    printf("Wartosc b wewnatrz funkcji fun(): %d\n", b);
}

int main()
{
    int a = 10;
    fun(a);
    printf("Wartosc a poza funkcja fun(): %d\n", a);
    return 0;
}
```

```
Wartosc b wewnatrz funkcji fun(): 11
Wartosc a poza funkcja fun(): 10
```

Zauważ, że uruchamiając w *main()* linię *fun(a)* zostaje wywołana funkcja *fun()* z parametrem o wartości *10*. Dalej wartość ta zostaje przypisana do zmiennej *b* i to właśnie na niej operujemy wewnątrz funkcji *fun()* – zmienna *a* pozostaje nietknięta.

Procedury

Przyjęło się, że **procedura** od funkcji różni się tym, że ta pierwsza nie zwraca żadnej wartości (typ zwracany to *void*). Zatem, aby stworzyć procedurę należy napisać:

```
void identyfikator (typ1 argument1, typ2 argument2, typn argument_n)
{
  /* instrukcje */
}
```

Oczywiście w ciele procedur mogą być realizowane różne obliczenia czy deklarowane zmienne. Jednak procedura nie zawiera instrukcji *return*.

Przykłady

```
#include<stdio.h>

// procedura suma()
void suma(int a, int b)
{
    printf("%d + %d = %d\n", a, b, a+b);
}

// procedura gwiazdka()
void gwiazdka(void)
{
    int i;
    for(i=0; i<10; i++)
        printf("*");
}

int main()
{
    suma(1,2);
    printf("\n");
    gwiazdka();
    return 0;
}
```

Zadania

1. Przeanalizuj i sprawdź działanie wszystkich przykładów zawartych w tej instrukcji.
2. Napisz funkcję o następującej deklaracji:
   int czyParzysta(int x);
   która sprawdza czy podana liczba jest parzysta. Jeśli tak, to zwróci 1, w przeciwnym razie 0.
   Wywołaj utworzoną funkcję w *main()* z różnymi wartościami parametru.
3. Napisz program, który wczytuje ze standardowego wejścia liczbę całkowitą n i wypisuje na standardowe wyjście wartość bezwzględną z n. Do rozwiązania zadania nie używaj funkcji bibliotecznych za wyjątkiem operacji wejścia/wyjścia. W programie użyj samodzielnie zaimplementowanej funkcji liczącej wartość bezwzględną.
4. Napisz funkcję zwracającą wartość największą spośród 3 liczb całkowitych a, b, c. Liczby te mają być parametrami funkcji. Wywołaj utworzoną funkcję kilkukrotnie z różnymi parametrami.
5. Napisz funkcję o następującej deklaracji:
   int potega(int x, int n);
   która zwracać będzie *n*-tą potęgę liczby *x*.

Funkcja powinna sprawdzać na początku, czy podany wykładnik jest nieujemny. Jeśli tak, to oblicza i zwraca odpowiedni wynik, jeśli nie – zwraca wartość -1 (minus jeden) oznaczającą błędne parametry.
Wywołaj utworzoną funkcję kilkukrotnie z różnymi parametrami.