

Podstawy Programowania

Zajęcia laboratoryjne 4

I rok Bioinformatyki Politechniki Poznańskiej

Funkcje

1 Definicja, deklaracja i wywołanie funkcji

Funkcje to opisy czynności (instrukcje) do wykonywania. Funkcje mogą przyjmować różne parametry (argumenty), następnie je przetwarzać w celu zwrócenia lub wyświetlenia wyniku. Argumenty mogą być przekazywane przez wartość lub przez adres. W pierwszym przypadku modyfikacja parametru nie spowoduje jego zmiany po zakończeniu działania funkcji, ponieważ do funkcji przekazywane są kopie wartości argumentów. Natomiast modyfikacja parametru przekazanego przez adres spowoduje zmianę wartości zmiennej, która znajduje się pod przekazanym adresem parametru. Funkcje mogą, ale nie muszą zwracać wartości określonego typu. Jednak to czy funkcja coś zwraca musi być określone przy deklaracji funkcji (słowo kluczowe `void` określa funkcje, które nie zwracają wartości).

Tworząc własną funkcje należy pamiętać o **deklaracji funkcji** (tzw. **prototyp funkcji**), **definicji funkcji** oraz instrukcji związanej z wywołaniem funkcji.

- Deklaracja funkcji (prototyp funkcji) określa zwracany typ funkcji, nazwę funkcji oraz deklarację parametrów (liczbę argumentów jakie przyjmuje oraz ich typ).
- Definicja funkcji dodatkowo określa jakie jest jej zadanie (instrukcje zapisane w ciele funkcji).

Definicja pojawia się w kodzie tylko jeden raz. Deklaracja prototypu nie musi wystąpić w ogóle, jednak jest często używana np. dla uporządkowania kodu lub w przypadku gdy funkcje wywołują siebie nawzajem. W przypadku gdy jedna funkcja wywołuje drugą funkcję w swoim ciele, może się okazać że funkcja zagnieżdżona jest nieznaną w momencie kompilacji, co spowoduje błędy. Do poprawnego działania kodu konieczna jest deklaracja prototypu funkcji zagnieżdżonej przed definicją funkcji, która w swoim ciele wywoła funkcję zagnieżdżoną. Celem uniknięcia tego typu błędów zaleca się aby pamiętać o tworzeniu prototypów funkcji, a następnie o tworzeniu definicji funkcji.

- Wywołanie funkcji w głównej części programu (w funkcji `main()`) polega na podaniu nazwy funkcji i określeniu wartości argumentów (podanych w nawiasie, oddzielonych przecinkami).

Przykład deklaracji, definicji i wywołania funkcji (kod 1 dostępny w materiałach na stronie):

```
1 #include <stdio.h>
2
3 //deklaracja funkcji
4 float funkcja(float fahr); //prototyp funkcji koniecznie zakonczony srednikiem!
5
6 int main()
7 {
8     float fahr, celsius;
9
10    for(fahr = 0.0; fahr <= 200; fahr = fahr + 20)
11    {
12        //wynik funkcji typu float mozna przypisac do zmiennej takiego samego typu
13        celsius = funkcja(fahr); //wywołanie funkcji: nazwa(argument)
14        printf("%3.0f %6.1f \n", fahr, celsius);
15    }
16
17    return 0;
```

```
18 }
19
20 float funkcja(float fahr) //definicja funkcji NIE konczy sie srednikiem jak prototyp
21 {
22     return (5.0/9.0)*(fahr - 32.0); //funkcja zwraca wynik typu float
23 }
```

2 Funkcje ze zwracaniem vs. funkcje bez zwracania

Zarówno w prototypie funkcji jak i w samej definicji należy określić typ funkcji. Funkcje bez zwracania przyjmują typ void, podczas gdy funkcje zwracające mogą być typu int, float, double, itd. Różnica w składni polega przede wszystkim na pojawieniu się słowa kluczowego return. Dzięki temu, że funkcja zwraca wartość określonego typu to może być przypisana do zmiennej w głównej części programu lub może zostać wykorzystana jako argument dla innej funkcji. Poniżej przykładowy kod (kod 2 dostępny w materiałach na stronie):

```
1 #include <stdio.h>
2
3 int funkcjaZeZwracaniem(int a, int b);
4 void funkcjaBezZwracania(int a, int b);
5
6 int main()
7 {
8     int a = 45, b = 25;
9     int wynik = funkcjaZeZwracaniem(2,3);
10    printf("Wynik funkcji typu int: %d\n", wynik);
11    funkcjaBezZwracania(a,b);
12
13    return 0;
14 }
15
16 int funkcjaZeZwracaniem(int a, int b)
17 {
18     int wynik = a + b;
19     return wynik;
20 }
21
22 void funkcjaBezZwracania(int a, int b)
23 {
24     printf("Wynik funkcji typu void: %d\n", a + b);
25 }
```

3 Zadania

Zad 1. Napisz program, w którym zaimplementujesz dwie funkcje, obie przyjmują jako argumenty zmienne a i b , będące liczbami całkowitymi. Wartości dla zmiennych a i b powinny być wprowadzone przez użytkownika. Pierwsza funkcja powinna wyświetlić, która z wprowadzonych liczb jest większa. Natomiast druga funkcja powinna zwrócić większą wartość.

Zad 2. Napisz funkcję, która jako argument przyjmuje liczbę całkowitą dodatnią $n (> 0)$, gdzie n jest podana przez użytkownika, a następnie wyświetla wszystkie jej dzielniki.

Zad 3. Wykorzystaj zadanie 6. z poprzednich zajęć (prosty kalkulator, zadanie z protokołu dla zajęć laboratoryjnych 3) i wyodrębnij funkcje zwracające rozwiązania dla każdego działania: dodawania, odejmowania, mnożenia i dzielenia. Funkcja dzielenia powinna zawierać zabezpieczenie dla dzielenia przez 0.

Zad 4. Napisz funkcję, która otrzymuje jako argumenty liczby całkowite dodatnie a i b , ich wartości powinny być podane przez użytkownika, a następnie zwraca wynik działania a^b .

Zad 5. Napisz funkcję, która otrzymuje jako argument liczbę naturalną n , gdzie n jest podana przez użytkownika, a następnie zwraca wynik działania $n!$.

Zad 6. Napisz funkcję, która otrzymuje jako argument liczbę naturalną n , gdzie n jest podana przez użytkownika, a następnie zwraca wynik działania 2^n .

1. Rozwiąż zadanie nie wykorzystując funkcji bibliotecznych.
2. Użyj gotowych funkcji.

Zad 7. Napisz funkcję, która jako argument przyjmuje liczbę całkowitą (szczęśliwy numerek), a następnie losuje 10 liczb całkowitych z przedziału $[1, 10]$ i zlicza ile razy podczas tego losowania pojawił się podany szczęśliwy numerek oraz ile stanowi to procent wszystkich wylosowanych liczb. Wartość argumentu funkcji powinna być podana przez użytkownika i powinna należeć do przedziału $[1, 10]$.

Zad 8. Napisz program, który dla 100 kolejnych liczb naturalnych wykona następujące działania: jeśli liczba jest parzysta to program obliczy $n/2$, jeśli liczba jest nieparzysta to obliczy $(n - 1)/2$. Niech oba te działania będą zaimplementowane w oddzielnych funkcjach zwracających odpowiednie rozwiązania.

4 Zadanie domowe

Napisz program, który będzie przeliczał temperatury w różnych skalach. Dla ułatwienia wyodrębniono poszczególne zadania. Pamiętaj jednak, że rozwiązaniem jest jeden wspólny kod.

Zad 1. Wykorzystaj przeliczenia temperatur (kod 1 dostępny w materiałach na stronie) i zmodyfikuj go tak aby utworzyć następujące funkcje przeliczające temperaturę:

- FtoC $C = (F - 32.0) * 5.0/9.0$
- FtoK $K = (F + 459.67) * 5.0/9.0$
- CtoF $F = C * 9.0/5.0 + 32.0$
- CtoK $K = C + 273.15$
- KtoC $C = K - 273.15$
- KtoF $F = K * 9.0/5.0 - 459.67$

Zad 2. Za pomocą instrukcji `switch ... case` dodaj menu w programie:

- 1 - przelicz F na C
- 2 - przelicz F na K
- 3 - 6 - pozostałe funkcje
- 7 - zakończ działanie programu

Zad 3. Przerób powyższy program tak aby najpierw wyświetlił menu, prosząc użytkownika o wybranie opcji przeliczenia. Następnie (przed wywołaniem odpowiedniej funkcji) program prosi użytkownika o podanie odpowiedniej temperatury do przeliczenia np. dla funkcji FtoC program prosi o podanie F, a przelicza C. Program na końcu wyświetla zarówno wartość podaną jak i wartość przeliczoną.

Zad 4. Do programu przeliczającego temperaturę dodaj funkcję pomocniczą *int sprawdz(float temp, char stopnie)*, która sprawdza, czy temperatura (zmienna temp) podana przez użytkownika mieści się w zadanej skali (C, F lub K). Jeśli:

- $temp < 0$ oraz stopnie = "K"
- $temp < -273.15$ oraz stopnie = "C"
- $temp < -469.67$ oraz stopnie = "F"

wówczas funkcja powinna zwrócić pewną ustaloną wartość, na podstawie której program wypisze komunikat: Nie ma takiej temperatury.