Podstawy programowania: Laboratorium nr 3

Funkcje.

21 października 2016

mgr inż. Przemysław Walkowiak
dr inż. Michał Ciesielczyk

Instrukcja

W czasie pisania programu pamiętaj o:

1. dbaniu o czytelność kodu (odpowiednie formatowanie kodu, nazewnictwo zmiennych adekwatne do ich znaczenia, komentarze),
2. dbaniu o czytelność interfejsu z użytkownikiem (w sposób jawny pytaj użytkownika jakie dane ma podać oraz opisuj wyniki, które zwracasz),
3. przed fragmentem implementującym poszczególne zadania umieść komentarz: `/*Zadanie X */` oraz wypisz na ekranie analogiczny komunikat (X jest numerem zadania): `std::cout << "Zadanie X"<< std::endl;` ,
4. umieszczeniu wszystkich rozwiązań w jednym pliku, chyba, że w poleceniu napisano inaczej.
5. w zadaniach wymagających udzielenia komentarza bądź odpowiedzi, należy umieścić go w kodzie programu (np. w postaci komentarza albo wydrukować na ekranie).

Wprowadzenie

Deklaracja

W języku C++ deklaracja funkcji wygląda następująco:

```
typ_zwracany nazwa_funkcji(typ_argumentu nazwa_argumentu);
```

gdzie `typ_zwracany` jest nazwą typu wartości jaką funkcja ma zwrócić. Następnie podajemy nazwę funkcji, która musi być poprawnym identyfikatorem języka C++. W nawiasach okrągłych ( ) umieszcza się listę argumentów funkcji w postaci par: typ_argumentu – nazwa_argumentu. Lista taka może być listą pustą – mamy wtedy do czynienia z funkcją bezargumentową.

Jeżeli `typ_zwracany` jest typem **void** taka funkcja nie zwraca żadnej wartości.

Przykładowe deklaracje funkcji:

```
int main();
int generate_number();
double sum(double x, double y);
void print(std::string napis);
```

Deklaracje funkcji należy umieścić przed pierwszym odwołaniem się do danej funkcji — najlepiej na początku pliku źródłowego lub pliku nagłówkowego.

Ciało funkcji

Ciało funkcji - definicję, umieszcza się w nawiasach klamrowych po nagłówku funkcji:

```
typ_zwracany nazwa_funkcji(typ_argumentu nazwa_argumentu) {
    // cialo funkcji
}
```

Przykładowe definicje funkcji dla wyżej zadeklarowanych mogą wyglądać następująco:

```
int main() {
    int value = generate_number();
    int value2 = generate_number();

    double result = sum(value, value2);

    print("Hello World");

    return 0;
}

int generate_number() {
    return 4;
}

double sum(double x, double y) {
    return x + y;
}

void print(std::string napis) {
    std::cout << napis;
}
```

Argumenty funkcji

Argumenty funkcji mogą być dowolnego, poprawnie zdefiniowanego typu języka C++ (np. typy proste jak **int**, albo złożone jak `std::string`. Nazwa parametru musi być, tak samo jak przy deklaracji zmiennej, poprawnym identyfikatorem języka C++.

Argumenty funkcji dostępne są tylko w obrębie ciała danej funkcji, tj. nie będzie ich widać na zewnątrz. Np.:

```
int sum(int x, int y) {
    int z = x + y;
    int c = a + b; // BLAD: zmienne 'a' i 'b' są zdefinowane w funkcji
                   // main (ktora wywoluje funkcje 'sum'), jednakze
                   // nie sa one widoczne w tym miejscu!
    return z;
}
```

```
int main() {
    int a = 4;
    int b = 5;
    int x = 3;

    int r1 = sum(a, b);
    int r2 = sum(a, x); // zmienna 'x' jest inna zmienna
                        // anizeli argument 'x' w funkcji 'sum'

    std::cout << z; // BLAD: w funkcji 'main' nie mam zdefiniowanej
                    // zmiennej 'z' - ona jest tylko widoczna
                    // w funkcji 'sum'

    return 0;
}
```

Zalecenia przy pisaniu funkcji

- Nazywaj funkcję zgodnie z ich przeznaczeniem. Np. `dodaj` – dodaje liczby. `wyswietl`/`print` - wyświetla na ekranie, `czy_trojkat(int a, int b, int c)` – sprawdza czy można zbudować trójkąt.
- Unikaj bardzo ogólnych nazw: `function`, **do**,
- Funkcja wykonuje jakość operację/czynność — z tego względu nazwa funkcji powinna zawierać czasownik lub być frazą czasownikową: np.: `calculate_age`, `send_message`, `fill_array_with_prime_numbers`. Źle: `dog`, `student`, `bike`.
- Staraj się również by nazwy funkcji nie były za długie — łatwo się pogubić przy jej czytaniu.
- Staraj się ograniczać liczbę argumentów do maksymalnie 3. W kodzie powinno być najwięcej funkcji 0,1 i 2 argumentowych. Jeżeli chcesz zaimplementować funkcję z większą liczbą argumentów, pomyśl czy nie można jej podzielić na dwie.
- Ważne! Przy pisaniu funkcji staraj się by robiła ona tylko to o czym mówi jej nazwa, nic więcej i nic mniej, np.:

```
int sum(int x, int y) { // dobrze
    return x + y;
}
int multiply(int x, int y) { // źle
   std::cout << "Mnoze " << x << " oraz " << y;
   int z = x * y;
   std::cout << "Wynik: " << z; // funkcja ma pomnozyc wartosc
                                // a nie dodatkowo wyswietlac wynik
    return z;
```

```
10      }
```

Wskazówka przydatna przy pisaniu programu

Oprócz umieszczenia zadań w blokach **case** konstrukcji **switch** spróbuj umieścić kod w oddzielnej funkcji. Najprostszą funkcje definiuje się w sposób następujący:

```
void zadanie1() {
    // ciało funkcji
}
```

Następnie, wywołuje się ją w następujący sposób:

```
zadanie1();
```

UWAGA! Pamiętaj o tym by deklarować funkcje zanim będziesz je wywoływać w swoim kodzie źródłowym.

Więcej informacji: `http://en.cppreference.com/w/cpp/language/functions` oraz `http://www.cplusplus.com/doc/tutorial/functions/`

Zadania

Zadanie 1

Wykorzystując pętlę **do–while** oraz konstrukcję **switch**, w funkcji głównej zaimplementuj menu wyboru zadania. Uruchamiany program powinien kolejno:

1. zapytać użytkownika które zadanie chce wykonać,
2. wywołać funkcję `zadanieX()`, gdzie X to numer zadania wybrany przez użytkownika; w przypadku wybrania nieodpowiedniego numeru zadania wyświetlić odpowiedni komunikat.
3. zapytać użytkownika czy chce zakończyć program,
4. wrócić odpowiednio do pkt 1. lub zakończyć działanie programu.

Możesz wykorzystać tę implementację na kolejnych laboratoriach.

Zadanie 2

Zdefiniuj dwie funkcje:

(a) **bool** `triangleExists(`**float** `a,` **float** `b,` **float** `c)` – zwracającą wartość **true** jeżeli można zbudować trójkąt z odcinków o długościach `a`, `b`, i `c`, oraz **false** w przeciwnym wypadku.

(b) `float triangleArea(float a, float b, float c)` – zwracającą pole trójkąta zbudowanego z odcinków o długościach `a`, `b`, i `c`.

Wczytaj długości trzech odcinków a, b, c. Następnie, korzystając z wcześniej zdefiniowanych funkcji sprawdź czy można z nich zbudować trójkąt. Jeżeli tak to oblicz jego pole. Wyświetl na ekranie stosowny komunikat z wynikiem działania programu.

Wskazówka 1 Trójkąt można zbudować z trzech odcinków wtedy i tylko wtedy gdy długość każdego boku jest mniejsza niż suma długości dwóch pozostałych boków.

Wskazówka 2 Pole można obliczyć korzystając ze wzoru Herona

$$P = \sqrt{p\,(p-a)\,(p-b)\,(p-c)}, \tag{1}$$

gdzie $p = \frac{1}{2}\,(a+b+c)$ jest połową obwodu trójkąta.

Wskazówka 3 Do wyznaczenia pierwiastka możesz skorzystać z wbudowanej funkcji `sqrt`, przykładowo `sqrt(9)` zwróci `3`.

Zadanie 3

Zdefiniuj funkcję `float toDegree(float rad)` przyjmującą wartość kąta wyrażoną w radianach oraz zwracającą wartość kąta wyrażoną w stopniach. Przetestuj swoją implementację dla przykładowych danych (np. $1\ rad \approx 57,3°$).

Wskazówka 1 Aby zamienić radiany na stopnie skorzystaj ze wzoru:

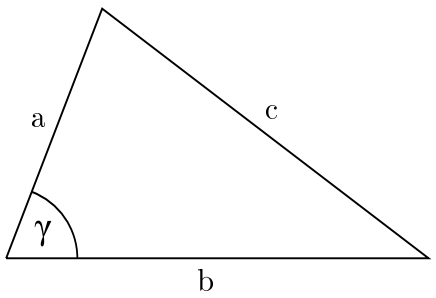
$$\alpha[°] = \alpha[rad] \cdot \frac{180°}{\pi}$$

Zamiast π możesz użyć wartość przybliżoną np. 3.1415.

Zadanie 4

Zdefiniuj funkcję `calculateAngle` zwracającą wartość kąta γ (wyrażoną w stopniach) przyległego do boków a i b, oraz przeciwległego do c. Do obliczenia kątów wykorzystaj twierdzenie kosinusów:

$$c^2 = a^2 + b^2 - 2ab\cos\gamma$$



Zastanów się, jakie argumenty (oraz jakiego typu) powinna przyjmować funkcja `calculateAngle`.

Wczytaj długości trzech boków trójkąta a, b, c, a następnie wyznacz wartości jego wszystkich trzech kątów (wyrażone w stopniach) wykorzystując funkcję `calculateAngle`. Wyświetl wynik działania programu na ekranie.

Wskazówka 1 Skorzystaj z funkcji trygonometrycznej `acos` (arcus cosinus) z biblioteki `cmath`. Zwróć uwagę, że zwraca ona kąt w radianach, a nie stopniach.

Wskazówka 2 Aby móc skorzystać z biblioteki `cmath` należy dodać na samym początku programu dyrektywę: `#include <cmath>`

Dodatkowe informacje:

- Biblioteka `cmath` - `http://www.cplusplus.com/reference/clibrary/cmath/`.

Zadanie 5

Zaimplementuj dwie funkcje liczące $n!$ (silnia liczby naturalnej n):

(a) `recursiveFactorial` – wykorzystującą mechanizm rekurencji, oraz

(b) `iterativeFactorial` – wyznaczającą silnię w wersji iteracyjnej.

Porównaj obie implementacje przeprowadzając testy dla dużych wartości n. Sprawdź dla jakich wartości zwracane są poprawne odpowiedzi. Wykorzystaj typ danych odpowiedniego rozmiaru. Obserwacje zapisz w komentarzach.

Zadanie 6

Wykorzystując mechanizm rekurencji zaimplementuj funkcję `recursiveFibonnaci` liczącą n-ty wyraz ciągu Fibonnaciego. Wypisz na ekranie wszystkie wyrazy ciągu Fibonacciego mniejsze niż 300.

Zadanie 7

Zaimplementuj funkcję `iterativeFibonnaci` liczącą n-ty wyraz ciągu Fibonnaciego w wersji iteracyjnej. Następnie, porównaj ją z wersją rekurencyjną (przeprowadź testy przy dużych wartościach n). Zwróć uwagę na czas wykonania. Obserwacje zapisz w komentarzach.

Na następne zajęcia

- Tablice statyczne: std::array.
- Tablice dynamicznie: std::vector.
- Pętla range-based for.