Programowanie C++

Wykład 2 (09.03.2016)

Uwaga: slajdy w znacznej części bazują na wykładzie Rafała Wysockiego

Sprawdzanie warunków – `if ()`

```
if (warunek)
    instrukcja;  // A
else
    instrukcja;  // B
```

```
if (warunek) {  // A
    instrukcja;
    ...
} else {  // B
    instrukcja;
    ...
}
```

1. Jeżeli `warunek` ma wartość `true`, zostanie wykonana instrukcja (lub blok) A, zaś w przeciwnym wypadku – instrukcja (lub blok) B.
2. Warunek powinien być wyrażeniem o wartości typu `bool`. Jeżeli tak nie jest, następuje konwersja wyniku do typu `bool` (zgodnie z zasadami przedstawionymi wcześniej).

Sprawdzanie warunków – `if ()` c. d.

Klauzula (*ang. clause*) `else` oraz instrukcja lub blok występujący po niej są opcjonalne.

Przykład

```
x = funkcja(arg);
if (x < 0)
    x = 0;
```

W niektórych przypadkach użycie trójargumentowego operatora prowadzi do bardziej zwartego kodu.

Przykład

```
x = y < z ? z : y;
```

Pętla while ()

```
while (warunek)
    instrukcja;
```

```
while (warunek) {
    instrukcja;
    ...
}
```

1. Powtarzaj instrukcję lub blok tak długo, jak długo `warunek` ma wartość `true`.
2. Warunek powinien być wyrażeniem o wartości typu `bool`. Jeżeli tak nie jest, następuje konwersja wyniku do typu `bool` (zgodnie z zasadami przedstawionymi wcześniej).

Pętla do ... while ()

```
do
    instrukcja;
while (warunek)
```

```
do {
    instrukcja;
    ...
} while (warunek);
```

1. Powtarzaj instrukcję lub blok tak długo, jak długo `warunek` ma wartość `true`.
2. Warunek powinien być wyrażeniem o wartości typu `bool`. Jeżeli tak nie jest, następuje konwersja wyniku do typu `bool` (zgodnie z zasadami przedstawionymi wcześniej).
3. Różni się od `while ()` tym, że instrukcja (lub blok) będzie wykonana co najmniej raz, niezależnie od początkowej wartości warunku.

Pętla for ()

```
for (ins1; war; ins2)
    instrukcja;
```

```
for (ins1; war; ins2) {
    instrukcja;
    ...
}
```

1. Wykonaj instrukcję `ins1`;
2. Oblicz `war` i jeżeli ma on wartość `false` lub 0, przerwij pętlę.
3. Wykonaj instrukcję (lub blok) za nawiasem.
4. Wykonaj instrukcję `ins2`.
5. Przejdź do kroku 2.

Pętle `for ()` i `while ()` c. d.

Przykład

```
for (int i = 1; i <= 10; i++)
    cout << i << "^2 = " << i*i << endl;
```

```
int i = 1;
while (i <= 10) {
    cout << i << "^2 = " << i*i << endl;
    i++;
}
```

W jednym i w drugim przypadku zostaną wydrukowane kwadraty liczb od 1 do 10.

Zasięg deklaracji zmiennych

Zasady określania zasięgu (*ang. scope*) deklaracji

1. Zmienna (lub stała) zadeklarowana wewnątrz bloku nie może być wykorzystywana poza tym blokiem.
2. Zmienna zadeklarowana w pierwszej instrukcji pętli `for ()` jest traktowana tak, jakby została zadeklarowana wewnątrz bloku obejmującego tę pętlę.
3. Instrukcje zapisane przed deklaracją zmiennej nie mogą zawierać odwołań do niej.
4. Jeżeli w bloku zadeklarowana jest zmienna o takiej samej nazwie, jaką ma zmienna zadeklarowana poza tym blokiem, to instrukcje w bloku będą odwoływać się w pierwszej kolejności do zmiennej zadeklarowanej wewnątrz bloku (przesłanianie deklaracji).

"Skracanie" kroku pętli – continue

continue

Powoduje przejście do następnego kroku pętli:

- Dla pętli `while ()` i `do ... while()` powoduje przejście do sprawdzania warunku.
- Dla pętli `for ()` powoduje przejście do kroku 4 w algorytmie wykonywania jej (wykonywanie instrukcji w trzecim polu w nawiasie).

Przykład

```
for (j = 1; j <= 20; j++) {
    if (j % 2)
        continue;
    count << j << "^2 = " << j*j << endl;
}
```

"Skracanie" pętli – break

break

Powoduje natychmiastowe przerwanie wykonywania pętli.

Przykład

```
k = 0;
a_k = 1;
suma = 1;
while (k < N) {
    k++;
    a_k /= k;
    suma += a_k;
    if (a_k < epsilon)
        break;
}
```

switch - przykład

```
switch (req) {
case RPM_REQ_NONE:
    break;
case RPM_REQ_IDLE:
    rpm_idle(dev, RPM_NOWAIT);
    break;
case RPM_REQ_SUSPEND:
    rpm_suspend(dev, RPM_NOWAIT);
    break;
case RPM_REQ_AUTOSUSPEND:
    rpm_suspend(dev, RPM_NOWAIT | RPM_AUTO);
    break;
case RPM_REQ_RESUME:
    rpm_resume(dev, RPM_NOWAIT);
}
```

Łączenie wyrażeń – , (przecinek)

, (przecinek)

1. Oblicz wyrażenie (lub wykonaj instrukcję) po lewej stronie przecinka (łącznie ze wszystkimi efektami ubocznymi).
2. Zaniedbaj wynik obliczonego wyrażenia.
3. Oblicz wyrażenie (lub wykonaj instrukcję) po prawej stronie przecinka.

Przykład

Następujący kod spowoduje wydrukowanie liczb 20 i 30 (w tej kolejności):

```
int i, b = 20, c= 30;
i = b, c;
cout << i << endl;
i = (b, c);
cout << i << endl;
```

Definiowanie funkcji z argumentami

Argumenty funkcji

1. Definiuje się w nagłówku funkcji, w nawiasie okrągłym za nazwą.
2. Ich definicje rozdziela się przecinkami.
3. Są definiowane podobnie, jak zmienne wewnątrz funkcji.
4. Mogą być wykorzystywane jako zmienne wewnątrz funkcji.

Przykład – funkcja z jednym argumentem

```
double f(double x)
{
    return x*x + 1;
}
```

Zwraca wynik będący kwadratem jej argumentu zwiększonym o 1.

Struktura programu

Włączenie pliku nagłówkowego biblioteki

Używamy *przestrzeni nazw* std

```
#include <iostream>
using namespace std;

int main() {

   int a = 10;
   a++;
   cout << "Hello world\n";
   cout << a;

   return 0;
}
```

Funkcja main()

Przekazywanie wartości argumentów do funkcji

Podczas wykonywania programu

1. Tworzone są zmienne, których nazwy i typy danych odpowiadają nazwom i typom danych argumentów funkcji.
2. Wartości znajdujące się na pozycjach odpowiadających argumentom w zapisie wywołania funkcji (np. `suma += sin(a);`) stają się początkowymi wartościami tych zmiennych.
3. Wykonywany jest kod reprezentowany przez instrukcje w treści funkcji, odwołujący się do tych zmiennych.
4. Wykonanie tego kodu może skończyć się wygenerowaniem wyniku, który jest wykorzystywany w sposób określony przez kod wywołujący funkcję.

Przekazywanie wartości argumentów do funkcji – przykład

Całka trapezowa dla funkcji `f()` z poprzedniego przykładu.

```
double trapez(double a, double b, int N)
{
    double delta, suma, x_j;
    int j;

    suma = (f(a) + f(b)) / 2;
    delta = (b - a) / N;
    for (j = 1, x_j = a; j < N; j++) {
        x_j += delta;
        suma += f(x_j);
    }
    suma *= delta;
    return suma;
}
```

```
double srednia(double x, double y) {
     return (x+y) / 2;
}


double srednia(double x) {
     return x;
}


double srednia(double x, double y, double z) {
     return (x+y+z) / 3;
}
```

```
// Funkcja zwraca srednia arytmetyczna lub geometryczna
double srednia(double x, double y, bool arytmetyczna = true) {
  if(arytmetyczna)
         return (x+y)/2;
  else
  {
        if(x < 0)
              return 0;
        if(y < 0)
              return 0;
        return sqrt(x*y);
  }
}
```