Kompendium na temat pętli i instrukcji warunkowych w C++

wersja artykułu: 1.0

Instrukcje warunkowe

Chciałem tutaj przedstawić podstawy dotyczące funkcji oraz instrukcji warunkowych w języku C++. Zacznijmy może jednak od tych drugich. A mianowicie od instrukcji **if – else**, jej skróconej wersji w postaci operatora warunkowego oraz instrukcji warunkowej **switch – case**.

Wymienione wyżej instrukcje pozwalają nam na rozbudowanie naszej aplikacji o warunki – dzięki czemu nasza aplikacja przestaje być liniowa i zaczyna rozrastać się o nowe możliwości, dając użytkownikowi możliwości wyboru co chce wykonać. Najbardziej podstawową jest instrukcja warunkowa **if – else**, jej budowa jest bardzo prosta:

```
if (warunek) { blok_warunku; }
```

Oraz w rozszerzonej wersji:

```
if (warunek) { blok_warunku; } else { blok_alternatywny; }
```

Oczywiście możemy stosować również kilka warunków do sprawdzenia, gdy pierwszy nie jest spełniony. W praktyce wygląda to mniej więcej tak:

```
if (warunek1) { blok_warunku1; } else if (warunek2 ) { blok_warunku2; } else { blok_alternatywny; }
```

Oczywiście blok alternatywny jak i w poprzednim przykładzie, tak i tutaj nie jest konieczny. Dobrze, koniec teorii, przejdźmy więc do przykładu. A więc exercise:

```
...
int a = 10;
if ( a == 9 )
{
        cout << "Ten kod sie nie wykona, gdyz a jest rowne 9, a nie 10" << endl;
        cout << a << endl;
} if (a >= 10 )
{
        cout << "Natomiast ten kod sie wykona, gdyz a jest rowne 10, a warunkiem jest liczba rowna
10 lub wieksza od 10" << endl;
} else {
        cout << "Blad - podana liczba nie spelnia zadnego z warunkow" << endl;
 }
...
```

W tym przypadku dla demonstracji przy każdym wywołaniu kod akcji umieściłem w bloku, jednak gdy kod wykonujący się po spełnieniu warunku nie jest większy niż jedno polecenie, to nie musimy umieszczać go w znacznikach bloku "{ }".

Operator warunku

Jak wiadomo, język C++ jest językiem, który dąży do jak największej minimalizacji. Dlatego też i instrukcja warunkowa ma swoją skróconą wersję w postaci operatora warunkowego. Jego budowa nie jest skomplikowana:

```
( warunek ) ? kod_do_wykonania : kod_alternatywny;
```

Jak widzimy jest on bardzo podobny w budowie do poprzednika, jednak słowa zamienione zostały znakami interpunkcyjnymi. Możemy go używać np. przy operatorach wyjścia - jak w przykładzie:

```
...
int a = 10;
cout << ( ( x & 1 ) ? "Podana liczba jest parzysta" : "Podana liczba jest nieparzysta" ) << endl;
...
```

W tym przykładzie warunkiem jest iloczyn bitowy podanej liczby i liczby 1 w systemie binarnym, czyli oznaczone wartością 1 są bity, które w obu liczbach się powtarzają. Dla przykładu 1 w systemie binarnym to 00000001, a 10 to 00001010, czyli żaden z bitów się nie powtarza - dlatego też wynikiem będzie 00000000, czyli 0. Gdybyśmy zamiast 10 użyli 11, to wyglądałoby to następująco: 00001011, więc pierwszy bit zwracałby 1, a otrzymana liczba byłaby równa 1. Wszystkie liczby nieparzyste w systemie binarnym na pierwszym bicie mają jedynkę, dlatego też na tej podstawie możemy stwierdzić która jest parzysta, a która nie.

Instrukcja decyzyjna

No dobrze, ale co zrobić, gdy mamy dużo możliwych warunków i kod zaczyna być strasznie nieczytelny, a drabinka strasznie szeroka? Tutaj z pomocą przychodzi nam instrukcja warunkowa **switch - case**. Zwana też instrukcją decyzyjną. Dzięki niej możemy określić kilka warunków dla jednej zmiennej. Jej budowa jest nieco bardziej rozwinięta, niż w przypadku if, lecz równie prosta do zapamiętania:

```
switch ( zmienna ) {
        case wartosc_warunku1: instrukcja1; break;
        case wartosc_warunku2: instrukcja2; break;
        default: instrukcja_alternatywna;
}
```

O co tutaj chodzi? To bardzo proste. Rozpoczynamy naszą instrukcję słowem kluczowym **switch**, następnie w nawiasie podajemy wyrażenie do którego chcemy przypisać warunki, po czym po słowie **case** ustalamy wartość jaka musi być spełniona oraz po dwukropku instrukcję do wykonania, gdy dany warunek jest wykonany. Natomiast polecenie **break** nakazuje instrukcji switch wyjście z danej etykiety case. **Default** odpowiada za wartość domyślną, jaka będzie wykonywana w przypadku, gdy żaden z warunków nie zostanie spełniony. Nie jest to obowiązkowa wartość, jednak

warto jej użyć w przypadku, gdy może nastąpić okoliczność przypisania innej wartości, niż wymienione w programie.

Oto mały przykład użycia w praktyce:

```
...
int a = 10;
switch ( a ) {
        case 9: cout << "to nie zostanie wykonane, gdyz a nie wynosi 9" << endl; break;
        case 10: case 11: cout << "natomiast to zostanie wykonane, gdyz a wynosi 10 lub 11" <<
        endl; break;
        default: cout << "twoja wartosc to nie 9, ani 10" << endl;
}
...
```

W tym przykładzie wykonana zostanie oczywiście druga instrukcja, gdyż spełniony jest warunek, iż a jest równe 10 lub 11. Zaobserwujmy tutaj, że gdy nie zakończymy danej wartości case komendą break, to możemy przejść do następnej gdy ta nie zostanie spełniona i przypisać jej ten sam wynik. To by było niestety na tyle, jeżeli chodzi o instrukcje warunkowe. Teraz więc należało by przejść do pętli.

Pętla For

Podobnie jak w języku C# oraz większości języków tego typu tak i tutaj mamy trzy podstawowe rodzaje pętli. Pętle służą do powtarzania ustalonych czynności określoną, bądź też nieokreśloną ilość razy. W tym wypadku należałoby zacząć od pętli **For**, jest to chyba najczęściej używana pętla - cechuje się ona tym, że wykonuje daną instrukcję określoną ilość razy. Nie koniecznie musi być ona określona w programie, może być pobierana od użytkownika - jednak musi być określona przed jej rozpoczęciem. Ułatwia nam to bardzo życie, gdy jesteśmy zmuszeni powtórzyć jakąś czynność n razy i nie widzi nam się przepisywanie tysiąc razy tej samej instrukcji. W C++ wygląda ona identycznie jak w prędzej wspomnianym C#, a mianowicie:

```
for ( wartosc_poczatkowa; warunek; wartosc_przyrostowa ) { instrukcja; }
```

Wbrew pozorom nie taka skomplikowana jaką się może wydawać. Kolejno omówmy nawias z wartościami. Pierwsza wartość_początkowa jest to wartość, od której startujemy - najczęściej jest to 1. Następnie warunek, to operator relacji z wartością końcową (np. **i < 10**, czyli do 10 - nie wliczając samej 10 lub **i <= 10**, czyli do 10 - łącznie z 10). Ostatnia wartość wskazuje nam na to, o ile wartości ma przeskakiwać nasza pętla (np. x++, czyli przeskocz o jedną wartość). W praktyce wygląda to następująco:

```
...
for ( int i = 1; i < =10; i++ ) cout << "Ten tekst pojawi sie 10 razy na ekranie" << endl;
...
```

Przy pętlach bardzo przydatne są tablice - przy działaniach wymagających stworzenia większej ilości zmiennych, do których powinny zostać przypisane wartości, na których później chcemy pracować. Np.:

```
...
int i = 0, j, tablica[10][10];
for ( ; i < 10; i++ )
{
	for ( j = 0; j < 10; )
	{
		tablica[i][j]=(i+1)*(j+1);
		printf("%d\t", tablica[i][j]);
		j++;
	}
	printf("\n");
}
...
```

Powyższy przykład tworzy nam prostą tablicę z tabliczką mnożenia. Specjalnie jednak w tym przypadku zastosowałem dwie pętle (tzw. pętla w pętli) oraz opuściłem w ich listach wartości pewne pozycje - aby pokazać jak w inny sposób można zapisać jeszcze te pętle. Gdy jednak opuszczamy którąś z wartości pętli, musimy pamiętać o zostawieniu średnika we właściwym miejscu w nawiasie oraz uzupełnienia tej wartości we właściwym momencie kodu aplikacji. Np. gdy zrezygnujemy z wartości odpowiedzialnej za określenie liczby przeskoczeń o określoną ilość pozycji zawsze możemy zrobić to w bloku instrukcji chociażby poprzez podanie na jego końcu inkrementacji zmiennej.

Pętla For ma jeszcze swój odpowiednik do poruszania się po tablicach, a mianowicie **for each**, ale w języku C++ nie jest to jedna ze standardowych pętli więc nie będziemy się nią zajmowali. Działa ona głównie w Visual'u i jest to pętla rodem z C#, czy też VBA, gdyż Microsoft stara się teraz prowadzić środowiska swoich kompilatorów na równym poziomie opcjonalności.

Pętla While

Innym przykładem pętli jest pętla while, w przeciwieństwie do pętli for w niej nie określamy ilości powtórzeń, a będzie się ona wykonywała tak długo, aż wskazany warunek nie zostanie spełniony - tak więc w przeciwieństwie do pętli for nie jest sterowana licznikiem, tylko warunkiem, lecz podobnie jak tamta może się nie wykonać ani razu. Jej budowa wygląda następująco:

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

Nie jest zbyt skomplikowana, w miejscu warunku musimy wstawić warunek, jaki musi być spełniony aby pętla mogła się wykonać, po przekroczeniu którego zakończy swoją pracę. Natomiast w bloku instrukcji wpisujemy instrukcje do wykonania, które mają być powtórzone przez pętlę - oczywiście klamry są zbędne, gdy chodzi o jedną instrukcję.

Więc mały przykład zastosowania takiej pętli:

```
...
int i = 0, j = 0;
```

```
while ( i < 2 && j != 2 )
{
	cout << "i wynosi: " << i << ", j wynosi: " << j << endl;
	i++;
	j++;
}
...
```

Przykład pokazuje, że możemy spokojnie łączyć warunki ze sobą tworząc kilka kryteriów do wykonania. W tym wypadku warunkiem wykonania jest, gdy zmienna i jest mniejsza od 2 oraz j nie jest liczbą 2. Jeżeli któryś z tych warunków przestanie być spełniany pętla zakończy swoją pracę.

Pętla Do .. While

Ostatnią pętlą o jakiej chciałem wspomnieć jest pętla **do .. while**, czyli rozszerzona pętla while. Różni się od poprzedniczki tym, że musi zostać wywołana przynajmniej jeden raz. Spowodowane to jest tym, że najpierw wykonywany jest blok instrukcji, a dopiero później sprawdzany zostaje warunek. W odróżnieniu od poprzedniczek jest to pętla, którą należy zakończyć średnikiem. Jest to spowodowane tym, że kończy się komendą, a nie znacznikiem zamknięcia bloku. Poza tymi dwiema różnicami działa zupełnie tak samo jak matczyna pętla while.

Ogólna budowa pętli do .. while:

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

A w praktyce wygląda to tak:

```
...
int ilosc = 0;
float ocena, srednia = 0;
cout << "[Aby zakonczyc wpisywanie ocen wpisz 0]" << endl;
do
{
	cout << "Podaj ocene: ";
	cin >> ocena;
	if ( ocena > 0 )
	{
		srednia += ocena;
		ilosc += 1;
	}
} while ( ocena != 0 );

if ( ilosc > 0 )
{
	srednia /= ilosc;
```

```
        cout << "Twoja srednia ocen: " << srednia << endl;
} else {
        cout << "Nie podales ocen" << endl;
}
...
```

Wyjaśniać działania tego kodu chyba nie trzeba? Bo chyba każdy swojego czasu wykonywał podobne działania w szkole na koniec każdego półrocza i domyśla się do czego ona służy.

Polecenie Break

Gdy pracujemy z pętlami bardzo przydatnym poleceniem jest polecenie **break;** - dzięki niemu możemy zakończyć pętlę w każdym momencie. Gdy zostanie wywołane to polecenie, dalsza część kodu umieszczonego w pętli nie zostanie już wykonana.

Przykład:

```
...
int i;
for ( i = 1; i <= 10; )
{
        cout << "i wykosi: " << i << endl;
        if ( i == 3 )
                break;
        i++;
}
...
```

W tym przykładzie mimo, iż pętla ma wyznaczone 10 powtórzeń, to zakończy się po trzecim, gdyż wywołana zostanie komenda break, gdy zmienna "**i**" osiągnie wartość 3 (czyli po trzecim powtórzeniu).

Komenda Goto

Inną przydatną komendą w pracy z pętlami jest komenda **goto**. Używamy jej wtedy, gdy mamy zamiar przeskoczyć do jakiegoś konkretnego miejsca w kodzie (np. poza pętlę). Pomijając tym samym kolejne działania, które by miała wykonać aplikacja pomiędzy miejscem jej użycia, a miejscem do którego ma przeskoczyć lub końcem kodu. By jej użyć musimy nadać jej jakąś nazwę zaraz po komendzie **goto** (w miejscu, gdzie program ma się zatrzymać), a następnie odwołać się do tej nazwy w miejscu, do którego ma przeskoczyć (wrócić lub iść dalej).

Poniżej przykład jej użycia:

```
...
int i;
for ( i = 0; i < 10; i++ )
 {
```

```
            cout << "i = " << i << endl;
            if ( i == 3 )
            goto stop;
 }
printf( "ten kod nie zostanie pokazany, gdyz zostanie pominiety, i = ", i );

stop:
printf( "pojawi się dopiero od tego momentu, lecz i = ", i );
...
```

Komenda Continue

Na koniec chciałem jeszcze wspomnieć o rzadko stosowanej komendzie, acz może się komuś przydać kiedyś w pracy z C++. Mianowicie jest to komenda **continue**. Stosujemy ją wtedy, kiedy chcemy przejść dalej. Zastosowana w pętli albo przechodzi od razu do następnej inkrementacji (jak jest to w przypadku pętli for) lub od razu przechodzi do sprawdzania warunku pomijając instrukcje znajdujące się po niej w danym wywołaniu pętli (gdy mówimy o pętli while). Najczęściej stosowana z instrukcją warunkową, w celu określenia wytycznych czasu wystąpienia.

Przykład:

```
...
for ( int i = 0; i < 10; i++ )
{
            if ( i == 3 || i == 8 )
                        continue;
            cout << "i = " << i << endl;
}
...
```

W tym przypadku pętla nie wykona instrukcji i przejdzie do następnego wywołania gdy zmienna "**i**" osiągnie wartość 3 lub 8.

W tym artykule to by było na tyle. Mam nadzieję, iż komuś się przyda wiedza zawarta w tym artykule.

Prawa autorskie