

XI. Tworzenie warunków złożonych if ... else

11.1. Co to warunek złożony

Pierwsze słowa wymagają małego sprostowania odnośnie warunków złożonych. Warunki złożone nie istnieją (bynajmniej tak mi się wydaje...). Do czego zmierzam? Nazwę warunków złożonych wprowadziłem ja, w celu podzielenia materiału związanego z warunkami na bardziej przystępne części do nauki. Jeśli nie zrozumiałeś rozdziału poświęconego warunkom prostym, polecam się do niego teraz wrócić, ponieważ ten rozdział będzie bazował głównie na informacjach o których wspominałem we wcześniejszym rozdziale.

Warunek złożony, to złożenie kilku warunków prostych, **połączonych** operatorem logicznym.

11.2. Podstawy logiki

Warunki proste już znamy, pora zapoznać się z trzema podstawowymi operatorami logicznymi.

`&&` // nazwa angielska: AND; tłumaczenie: operator logiczny "i"

`||` // nazwa angielska: OR; tłumaczenie: operator logiczny "lub"

`!` //nazwa angielska: NEG; tłumaczenie: negacja

`/* Logika (dział matematyki):`

```
-----  
| a | b | (!a) | (!b) | (a && b) | (a || b) |  
-----  
| 0 | 0 | 1 | 1 | 0 | 0 |  
| 0 | 1 | 1 | 0 | 0 | 1 |  
| 1 | 0 | 0 | 1 | 0 | 1 |  
| 1 | 1 | 0 | 0 | 1 | 1 |  
-----  
*/
```

Wydaje mi się, że tabelka nie wymaga komentarza. Jednak dla osób, które nie miały jeszcze logiki postaram się trochę rozjaśnić sytuację. Przyjmijmy, że **a** i **b** są zmiennymi. Zmienne te mogą przyjmować wartości **0** i **1**. Wartość **0** symbolizuje fałsz (**false**), natomiast wartość **1** symbolizuje prawdę (**true**). W logice wartością przeciwną do 0 jest 1 i na odwrót dla wartości 1 przeciwną wartością jest 0. Wartość przeciwną nazywamy negacją, a w C++ negację zapisujemy znakiem wykrzyknika (!).

11.3. Warunki złożone w praktyce

Założmy teraz, że znamy wartości dwóch zmiennych (czyli zmiennej **a** i **b**) i jednocześnie chcemy od tych zmiennych uzależnić dalszy przebieg naszego programu. Jeżeli nastąpi taka sytuacja, że wszystkie wartości zmiennych będą prawdziwe (**1**), to ma wykonać określony dodatkowy blok instrukcji. Aby uzyskać ten efekt, w C++ należy napisać następujący kod:

```
//... tu jakiś kod wcześniejszy
```

```
if(a&&b)
```

```
{//początek bloku
```

```
    //tu kod dodatkowy, który chcemy wykonać,
```

```
    //gdy obie zmienne będą miały wartość 1
```

```
    /*-----
```

```
    informacja:
```

```
    precyzyjniejszym sformułowaniem byłoby, że obie
```

```
    zmienne będą różne od zera, ponieważ dla C++,
```

```
    każda wartość różna od 0 jest wartością
```

```
    prawdziwą, czyli true (1).
```

```
    Nie chciałem Ci tego jednak wcześniej powiedzieć,
```

```
    ponieważ nie chciałem zaciemniać Ci teorii logiki.
```

```
    -----*/
```

```
}//koniec bloku
```

```
//... tu dalszy kod programu
```

Teraz wystarczy zastąpić literki **a** i **b** warunkami i mamy gotowy poprawny warunek złożony. Wprowadziłem te symbole po to, żebyś nie koncentrował się za bardzo na wiedzy z poprzedniego rozdziału. W praktyce symbole **a** i **b** mogą być fizycznie zmiennymi, jednak nie będziemy raczej mieli powodów, by zastosowane symbole były faktycznie zmiennymi.

11.4. Utrwalenie wiadomości przez analizę przykładów

Jedyną co pozostaje teraz zrobić, to prześledzić zamieszczone przykłady i napisać kilka zadań w celu utrwalenia wiedzy zdobytej w tym rozdziale.

```
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    float a,b;
    cout<<"Podaj a: ";
    cin>>a;
    if((a>=50)&&(a<=100))
    {
        cout<<"Podana liczba mieści się w przedziale (a>=50)&&(a<=100)"<<endl;
    }else cout<<"ERROR! Liczba nie mieści się w przedziale (a>=50)&&(a<=100)"<<endl;
    getch();
    return(0);
}

#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    float a,b;
    cout<<"Podaj a: ";
    cin>>a;
    if( ((a>=50)&&(a<=100)) || ((a>10)&&(a<30)) )
    {
        cout<<"Podana liczba mieści się w jednym z przedziałów: ((a>=50)&&(a<=100)) lub
(a>10)&&(a<30))"<<endl;
    }else cout<<"ERROR! Liczba nie mieści się w żadnym z przedziałów ((a>=50)&&(a<=100))
(a>10)&&(a<30))"<<endl;
    getch();
    return(0);
}

#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    float a,b;
    cout<<"Podaj a: ";
    cin>>a;
    if((a>=50)&&(a<=100)&&(a!=75))
    {
        cout<<"Podana liczba mieści się w przedziale (a>=50)&&(a<=100) i jest różna od 75"<<endl;
    }else cout<<"ERROR! Liczba nie mieści się w przedziale (a>=50)&&(a<=100) lub jest równa
75"<<endl;
    getch();
    return(0);
}
```

11.5 Operator ?: - operator wyboru.

Jest to operator, który może zastąpić instrukcję **if else**. Jest jedynym operatorem który przyjmuje trzy argumenty. Oto prosty przykład:

```
7 > 4 ? 10 : 12;
```

co czytamy, jeżeli 7 jest większe od 4 to wynikiem jest 10, jeżeli nie jest równe to wynikiem jest 12. Oczywiście zamiast liczb możemy stosować własne wyrażenia. Spójrzmy na poniższy przykład:

```
#include <conio.h>
#include <iostream>
int main()
{
    using namespace std;
    int a = 34, liczba = 0;

    cout << "Podaj jakas liczbe\n";
    cin >> liczba;
    cout << ((liczba > 34)
? "Twoja liczba jest wieksza od mojej\n"
: "Twoja liczba jest mniejsza od mojej\n");

    cout << "\nOperator ?: dziala jak: \n"
<< "if (liczba > 34)\n cout "
"<< \"Twoja liczba jest wieksza od mojej\";\n"
"else \n"
" cout << \"Twoja liczba jest mniejsza od mojej\";";

    getch();
    return 0;
}
```

Jak widać, operator **?:** może być ciekawą alternatywą dla instrukcji **if else**. Dodatkowo ma jedną przewagę, iż może być częścią większego wyrażenia (lub zagnieżdżony).

11.6 Ćwiczenia

1. Zbuduj wyrażenia logiczne:

1. prędkość samochodu zapisana w zmiennej *km* jest od 0 do 220,
2. zmienna *znak* jest większy od „B” i mniejszy od „b” lub równy „Z”
3. zmienna *liczba* jest cyfrą szesnastkową
4. pojemność basenu *obj* jest większa lub równa 135, ale mniejsza o 300
5. zmienna *max* określa liczb mieszkańców, która jest w granicach 12234 a 13242, lub *pies* równo się -4

2. Napisz program „Kino”. Program ma zawierać ponumerowane menu, z dostępnymi seansami filmowymi. Gdy wybierzemy dany film program ma zapytać ile biletów chcemy kupić, dodatkowo ma sprawdzić czy na danych seans są jeszcze miejsca(załóż iż na każdy seans mieści się 20 osób). Co wiąże ze sobą to by program liczył ile już biletów kupiono na dany film.

Program ma się zakończyć po naciśnięciu litery „Z” lub „z”. Oto przykład wyglądu aplikacji:

Kino Domownik Zaprasza.

Dzisiejsza oferta to:

1. Bolek i Lolek
2. Seksmisja
3. Krzyżacy
4. Killer

Wybierz seans?

Wybrano film Killer (wolnych miejsc 3)

Podaj liczbę biletów, które chcesz nabyć.

Dodatkowo aplikacja powinna mieć zabezpieczenie, by podczas podania większej liczby biletów niż jest dostępna, poinformowała o tym. Jeżeli klient rezygnuje z zakupu biletów po naciśnięciu „p” aplikacja powinna wrócić do menu(najlepiej gdyby sprawdzała czy została naciśnięta cyfra lub literka „p”). Po zakupie biletów również powinna wrócić do menu.

3. Napisz grę koło fortuny, gra będzie składać się z dwóch części. Najpierw pierwszy gracz wpisze szukane słowo, a potem kolejny będzie miał za zadanie je odgadnąć. Skorzystaj z funkcji `strcmp()`(nagłówka `cstring`) do porównywania słów. Dodatkowo wprowadź licznik, który pokarze, po której próbie zawodnik uzyskał poprawne rozwiązanie.