Lekcja 2
Sprawdzanie danych – instrukcja warunkowa

instrukcja warunkowa – formatowanie – typy zmiennych – operacje arytmetyczne i logiczne – warunki zagnieżdżone

Proste programy działają „liniowo” – instrukcja po instrukcji, w kolejności, w jakiej zostały wpisane. Działanie bardziej skomplikowanych programów zależne jest (między innymi) od spełnienia (bądź nie) określonych warunków. Wszystkie języki programowania wykorzystują w tym celu instrukcję warunkową.

Schemat instrukcji warunkowej IF

if (warunek) instrukcja;
if (warunek) instrukcja; else instrukcja;
if (warunek) { instrukcje;… }
if (warunek) { instrukcje;… } else { instrukcje;… }

Zadanie - parzystość

Wczytaj liczbę z klawiatury i sprawdź czy jest parzysta.

```
int liczba;
cout << "SPRAWDZAMY PARZYSTOŚĆ" << endl;
cout << "Wpisz liczbę: ";
cin >> liczba;
if (liczba % 2 == 0)
  cout << liczba << " - parzysta";
else
  cout << liczba << " - nieparzysta";
```

- wczytujemy liczbę do zmiennej liczba
- **jeżeli liczba podzielona całkowicie przez 2 (modulo) jest równa 0, to…**
- wypisujemy komunikat o parzystości
- w przeciwnym razie komunikat o nieparzystości

Zadanie - prostokąt

Wczytaj dwie liczby z klawiatury – boki prostokąta. Jeśli są większe od zera to oblicz pole o obwód tego prostokąta.

```
cout << "wpisz boki: ";
double BOK1,BOK2;
cin >> BOK1 >> BOK2;
if (BOK1>0 && BOK2>0){
  double POL=BOK1*BOK2;
  double OBW=2*BOK1+2*BOK2;
  cout << " POLE=" << POL << endl;
  cout << "OBWÓD=" << OBW << endl;
}
else cout << "wpisz boki > 0";
```

- po każdej liczbie ENTER lub rozdzielone SPACJĄ i ENTER na końcu
- **jeżeli BOK1 jest większy od zera i BOK2 jest większy od zera, to…**
- obliczamy pole i obwód
- w instrukcji warunkowej wykonujemy kilka instrukcji, więc stosujemy instrukcję blokową {}

Operacje logiczne

== != < <= > >=

|| **alternatywa** - suma logiczna && **koniunkcja** – iloczyn logiczny ! **negacja** - zaprzeczenie

Przypisanie - porównanie

Wszystkie języki programowania mają problem z dokładnym określeniem **różnicy pomiędzy przypisaniem (nadawaniem wartości zmiennej), a porównywaniem** dwóch wartości. W C++ do przypisywania służy operator „=”, a do porównywania „==”. W Pascalu na przykład przypisujemy za pomocą operatora „:=” a porównujemy operatorem „=”. Istnieją również języki, które starają się same domyślić co programista miał na myśli i obie operacje wykonujemy operatorem „=”.

Warunki zagnieżdżone

Zadanie - egzamin

Wczytaj z klawiatury punkty za egzamin (0..60) i przypisz je do jednej z grup: (0..20) – podstawowa, (21..40) – średnia (41..60) – zaawansowana.

```
int punkty;
cout << "STOPIEŃ ZAAWANSOWANIA" << endl;
cout << "Wpisz liczbę punktów: ";
cin >> punkty;
if (punkty>=0 && punkty <=20)
  cout << "grupa podstawowa";
if (punkty>20 && punkty <=40)
  cout << "grupa średnia";
if (punkty>40 && punkty <=60)
  cout << "grupa zaawansowana";
cout << endl;
```

Ponieważ instrukcja warunkowa porównuje tylko dwie wartości, musimy porównywać kilka razy stosując złożone operatory logiczne.

Możemy też zagnieździć instrukcje warunkowe. Nie musimy tworzyć osobnego warunku logicznego dla ostatniej zależności – skoro punkty nie należą do poprzednich zakresów, to muszą należeć do ostatniego zakresu.

```
if (punkty>=0 && punkty <=20)
  cout << "grupa podstawowa";
else
  if (punkty>20 && punkty <=40)
    cout << "grupa średnia";
  else
    cout << "grupa zaawansowana";
```

Instrukcja wyboru

Instrukcja wielokrotnego wyboru pełni podobną rolę jak zagnieżdżona instrukcja wyboru – decydujemy o wykonywaniu instrukcji na podstawie wartości jednej zmiennej. W C++ ta zmienna decydująca o wyborze musi być typu całkowitego (lub znakowego).

Zadanie – kalkulator

Napisz prosty kalkulator dla dwóch liczb, obsługujący cztery działania matematyczne +, -, *, /.

Zwróć uwagę, że teksty wpisujemy w cudzysłowie, a zmienne znakowe **char** w apostrofach.

Wczytujemy kilka elementów: po każdym ENTER lub spacje pomiędzy i na końcu ENTER.

Jeżeli druga liczba jest zerem, to nie mamy możliwości dzielenia, co jest sprawdzane i sygnalizowane komunikatem o błędzie.

Jeżeli nie wybierzemy poprawnie działania, to podejmowane jest działanie po słowie **default** – wypisujemy komunikat o błędzie.

Wpisywanie danych z klawiatury kończy się naciśnięciem klawisza ENTER, który też zostanie zapamiętany przez komputer i mogą pojawić się problemy podczas wczytywania kolejnych danych (np. wczytywanie w pętli). Dzięki użyciu polecenia **cin.ignore();** system nie powinien sprawiać problemów.

```
#include <stdlib.h>
#include <iostream>
using namespace std;
int main(){
  setlocale(LC_ALL, "");
  //kalkulator
  double L1,L2;
  char Z;
  cout<< "KALKULATOR" << endl << endl;
  cout<< "Wpisz liczbę, znak i liczbę"<<endl;
  cout<< "ENTER" << endl;
  //spacje pomiędzy i ENTER lub 3xENTER
  cin >> L1 >> Z >> L2;
  cin.ignore();
  cout << "=" << endl;
  switch (Z){
  case '+':
    cout << L1+L2 << endl;
    break;
  case '-':
    cout << L1-L2 << endl;
    break;
  case '*':
    cout << L1*L2 << endl;
    break;
  case '/':
    //sprawdzamy podzielność przez zero
    if (L2!=0) cout << L1/L2 << endl;
    else cout << "ERROR /0" << endl;
    break;
  default:
    //gdy nie wpisano poprawnie działania
    cout << "ERROR +-*/" << endl;
    break;
  }
  system("pause");
}
```

Sprawdzanie poprawnego wczytywania danych w konsoli

Co się stanie, jeśli napisany przez nas program oczekiwać będzie na wpisanie z klawiatury liczby (np. długości boku), a użytkownik wstawi tekst? Prawdopodobnie program się zawiesi lub zakończy swoje działanie z wyliczeniem złego wyniku.

Czy mamy możliwość sprawdzania, jaki rodzaj danych został wpisany z klawiatury: liczba całkowita, rzeczywista, pojedynczy znak czy tekst? Wszystkie języki programowania mają z tym problem i prawdopodobnie najwięcej czasu należy poświęcić na uodpornienie naszego programu na różne „dziwne" zachowania potencjalnego użytkownika.

W C++ panuje w tym względzie jeszcze większy bałagan. Część funkcji mamy w bibliotekach, ale dla większości przypadków trzeba posługiwać się funkcjami pisanymi samodzielnie.

Sposób formatowania kodu źródłowego

Formatowanie tekstu programu (kod źródłowy) ma bardzo ważne znaczenie, bo pozwala szybko i bezboleśnie zrozumieć to, co kiedyś napisaliśmy (lub ktoś inny napisał). Każdy porządny programista powinien przyswoić sobie podstawowe techniki edycji i stosować je wszędzie, gdzie to tylko możliwe: akapity, wcięcia, odstępy i właściwe nazewnictwo. Przyznacie sami, że istnieje zdecydowana różnica w próbach zrozumienia „co autor miał na myśli" pisząc drugi z programów – oba wykonują takie same operacje.

```
int suma = 0;
int iloczyn = 1;
for (int i = 1; i <= 10; i=i+1)
{
  suma = suma + i;
  iloczyn = iloczyn * i;
}
```

```
int s=0;int l=1;for(int i=1;i<=10;i++){s+=i;l*=i;}
```

Elementarz poprawnego formatowania

- dziel na akapity każdą odrębną instrukcję
- stosuj wcięcia bloków programu
- nawiasy klamrowe {} wyrównuj do odpowiednich instrukcji
- nadawaj opisowe (ale niezbyt długie) nazwy zmiennym (zamiast **l** utwórz zmienną **liczba**)
- stosuj odstępy pomiędzy operatorami (zamiast **l%2==0** napisz **liczba % 2 == 0**)

Zadania – wyniki na ekranie, jak na załączonych obrazkach

1) *Sprawdzamy, czy liczba wpisana z klawiatury jest podzielna przez 3, 5, 7, 10.*

```
PODZIELNOŚĆ 3,5,7,10
Wpisz liczbę: 20
20 podzielna przez 5
20 podzielna przez 10
```

2) *Sprawdzamy czy wpisane z klawiatury boki trójkąta spełniają warunek istnienia trójkąta.*

```
TRÓJKĄT
Wpisz boki trójkąta
12
6
13
Trójkąt OK
```

3) *Wyliczamy deltę na podstawie wpisanych z klawiatury parametrów równania kwadratowego i wypisujemy liczbę rozwiązań.*

```
RÓWNANIE KWADRATOWE
Wpisz parametry a, b, c: 2 4 2
jedno rozwiązanie
X=-1
```

```
RÓWNANIE KWADRATOWE
Wpisz parametry a, b, c: 1 4 3
dwa rozwiązania
X1=-3
X2=-1
```

```
RÓWNANIE KWADRATOWE
Wpisz parametry a, b, c: 4 1 2
brak rozwiązań
```

4) *Napisz program, który wystawi ocenę z testu. Użyj zagnieżdżonej instrukcji if - else*

0 - 39 pkt - ocena niedostateczna
40 - 54 pkt - ocena dopuszczająca
55 - 69 pkt - ocena dostateczna
70 - 84 pkt - ocena dobra
85 - 98 pkt - ocena bardzo dobra
99 - 100 pkt - ocena celująca

5) *Napisz program który sprawdza, która z podanych 2 liczb przez użytkownika jest najmniejsza.*

6) *Napisz program, który dla danego punktu na płaszczyźnie (X, Y) sprawdzi, w której ćwiartce układu współrzędnych się on znajduje. Jeżeli punkt leży w środku układu współrzędnych, program powinien wypisać liczbę 0. Jeżeli na osi X program powinien wypisać OX, jeśli na osi Y – wypisać OY.*