# Programowanie proceduralne w języku C++
# Instrukcja warunkowa
# Instrukcja wielokrotnego wyboru

Mirosław Głowacki [1,2]

[1] Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Inżynierii Metali i Informatyki Stosowanej
Katedra Informatyki Stosowanej i Modelowania

[2] Uniwersytet im. Jana Kochanowskiego w Kielcach
Wydział Nauk Ścisłych i Przyrodniczych
Instytut Fizyki
Zakład Fizyki Komputerowej i Informatyki

Kwiecień 2020 / Kwiecień 2021

# Spis treści

# Spis treści

# Wprowadzenie do tematu

1. Stosowanie instrukcji warunkowej otwiera drzwi do pisania użytecznych aplikacji w zasadzie w każdym języku programowania proceduralnego.
2. Dlatego też zajmiemy się tą instrukcją sposób dokładny, a zarazem pragmatyczny.
3. Pokażemy jej użycie w różnych sytuacjach i odmianach.
4. Materiał zamieszczony poniżej pozwoli pełniej zrozumieć zasady stosowanie operatorów relacji i operatorów logicznych

# Podejmowanie decyzji

W życiu stale podejmujemy jakieś decyzje - zazwyczaj jednak intuicyjnie, co w przypadku człowieka jest naturalne.

Pomimo, iż komputer również stale podejmuje jakieś decyzje, to postępuje zgodnie z poleceniami.

Wszystko więc musi być precyzyjnie sformułowane - inaczej komputer stałby się nieprzewidywalny.

Sztuką programowania jest umiejętność zapisywania swoich rozbieganych myśli w sposób formalny i poprawny z punktu widzenia logiki za pomocą języka programowania.

# Instrukcja warunkowa if

Najprostsza wersja instrukcji warunkowej wygląda następująco:

**c++: if**

```
if (warunek) instrukcja;
```

- Warunek to wyrażenie logiczne.
- Wyrażenia logiczne zapisywaliśmy do tej pory do zmiennych i wyglądało to tak:

```
int wiek = 30;
bool jestSamodzielny = wiek > 18 && wiek < 100;
```

# Instrukcja warunkowa if

Wyrażenie logiczne możemy wstawić do instrukcji warunkowej na dwa sposoby:

- Pierwszym z nich jest użycie zmiennej, tj:

```
if(jestSamodzielny == true) instrukcja;
```

równoważny zapisowi

```
if(jestSamodzielny) instrukcja;
```

- Drugi ze sposobów, to umieszczenie wyrażenia logicznego wprost w nawiasach instrukcji warunkowej:

```
int wiek = 30;
if(wiek > 18 && wiek < 100) instrukcja;
```

# Instrukcja warunkowa if

Instrukcja występująca jako pierwsza po nawiasie zamykającym wyrażenie logiczne będzie tą, która zostanie wykonana jeśli warunek będzie spełniony.

Jeśli warunek nie będzie spełniony instrukcja ta zostanie pominięta. Przykład:

```
#include <iostream>
using namespace std;
int main(){
    int wiek;
    cout << "Podaj swoj wiek: "; cin >> wiek;
    if( wiek >= 18 ) cout << "Jestes pelnoletni. "
                          << endl;
        cout << "Koniec" << endl;
    return 0;}
```

# Instrukcja warunkowa if

A oto wynik użycia zaprezentowanego programu:

**Okno konsoli**

```
Podaj swoj wiek: 12
Koniec
Process returned 0 (0x0) execution time : 4.251 s
Press any key to continue.
```

i drugi przykład:

**Okno konsoli**

```
Podaj swoj wiek: 22
Jestes pelnoletni.
Koniec
Process returned 0 (0x0) execution time : 4.392 s
Press any key to continue.
```

# Instrukcja warunkowa if

Jeśli więcej niż jedna instrukcja ma zostać wykonana po spełnieniu warunku, należy użyć bloku instrukcji, który tworzymy za pomocą nawiasów klamrowych { }

```
#include <iostream>
using namespace std;
int main(){
    int wiek;
    cout << "Podaj swoj wiek: "; cin >> wiek;
    if( wiek >= 18 ){
        cout << "Masz juz " << wiek << " lat - ";
        cout << " jestes pelnoletni. " << endl; }
    cout << "Koniec" << endl;
    return 0; }
```

# Zagnieżdżanie warunków

Warunki możemy zagnieżdżać - zasady używania i działania warunków pozostają te same.

```
#include <iostream>
using namespace std;
int main(){
    int liczba;
    cout << "Podaj dwucyfrowa liczbe naturalna: ";
    cin >> liczba;
    if( liczba < 10 || liczba > 99) {
        cout << "Liczba nie jest dwucyfrowa" << endl;
        if( liczba < 0 ){
            cout << "oraz nie jest naturalna" << endl;
        }
    return 0; }
```

# Zagnieżdżanie warunków

Popatrzmy na przykład użycia kodu:

```
Podaj dwucyfrowa liczbe naturalna: 7
Liczba nie jest dwucyfrowa
Process returned 0 (0x0) execution time : 3.188 s
```

i drugi przykład:

```
Podaj dwucyfrowa liczbe naturalna: -3
Liczba nie jest dwucyfrowa
oraz nie jest naturalna
Process returned 0 (0x0) execution time : 2.922 s
```

a na koniec trzeci przykład:

```
Podaj dwucyfrowa liczbe naturalna: 11
Process returned 0 (0x0) execution time : 3.422 s
```

# Słowo kluczowe else

Bardziej złożona wersja instrukcji warunkowej jest następująca:

**c++: if ... else**

```
if (warunek) tInstrukcja; else fInstrukcja;
```

- Niewiadomą w tym zapisie jest "fInstrukcja", która oznacza instrukcję lub blok instrukcji wykonywany jeśli warunek nie zostanie spełniony. Posłużmy się fragmentem kodu:

```
int wiek; cout << "Podaj swoj wiek: "; cin >> wiek;
cout << "Masz " << wiek << " lat - ";
if( wiek >= 18 )
     cout << " jestes juz pelnoletni. " << endl;
else
   cout << " jestes niepelnoletni. " << endl;
```

# Słowo kluczowe else

A oto przykład użycia kodu:

**Okno konsoli**

```
Podaj swoj wiek: 12
Masz 12 lat - jestes niepelnoletni.
Process returned 0 (0x0) execution time : 3.282 s
Press any key to continue.
```

i drugi przykład:

**Okno konsoli**

```
Podaj swoj wiek: 35
Masz 35 lat - jestes juz pelnoletni.
Process returned 0 (0x0) execution time : 3.219 s
Press any key to continue.
```

# Spis treści

# Nasze pierwsze zadanie

## Problem do rozwiązania

- Robotnicy mają za zadanie wykonanie odcinka płotu ogrodzeniowego.
- Do dyspozycji mają pewną liczbę słupków i prętów.
- Każde z przęseł płotu powinno być wykonane z jednakowej liczby prętów.

## Nasze zadanie

Napisać program w języku C++, który pozwoli na:

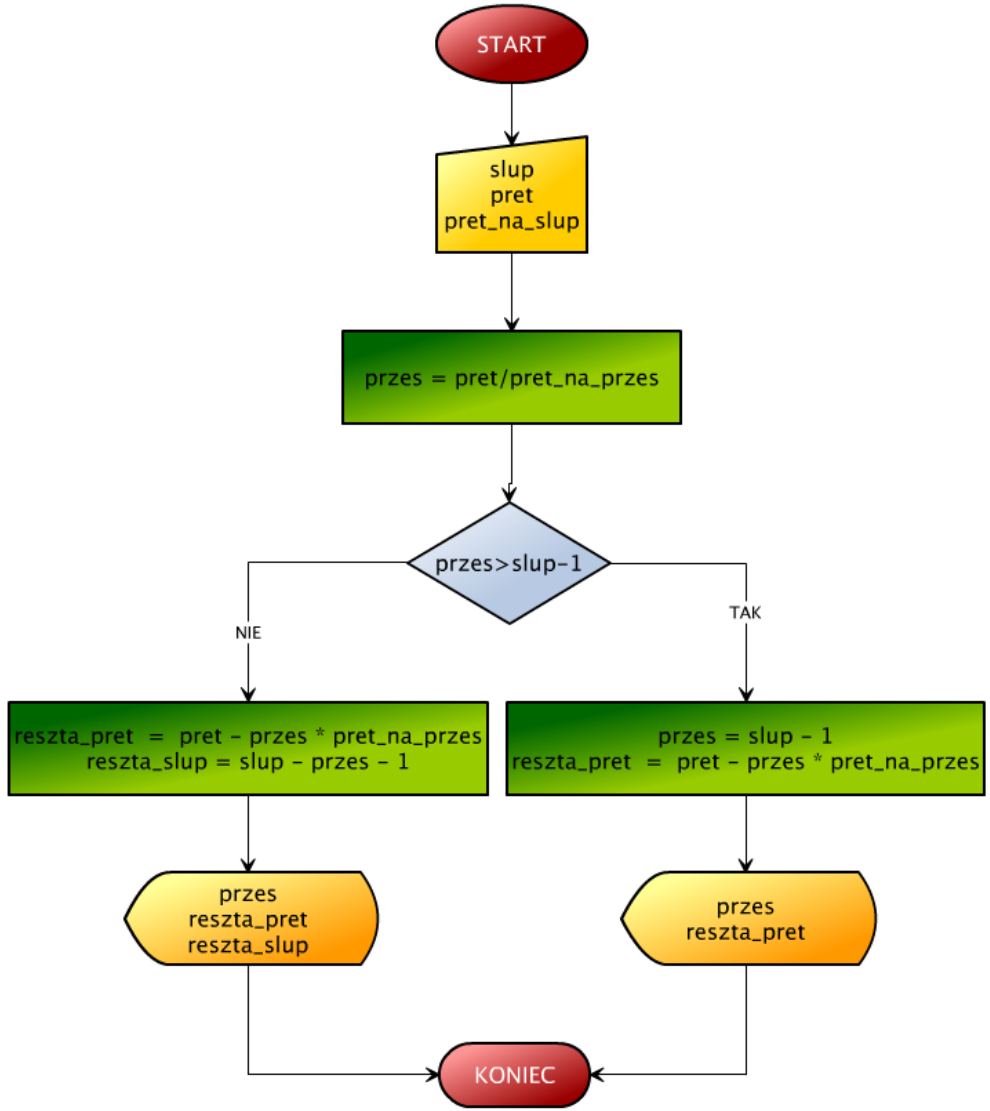
1. Zapamiętanie dostępnej liczby słupków i prętów oraz liczby prętów z jakich składa się przęsło.
2. Obliczenie i wypisanie maksymalnej liczby przęseł, jakie można wykonać z dostępnych materiałów.
3. Obliczenie i wypisanie liczby słupków i prętów jakie pozostaną do dyspozycji.

# Schemat blokowy algorytmu

Wymagana jest odpowiedź na pytanie:

Czy czynnikiem limitującym produkcję jest liczba słupków czy prętów?

Odpowiedź twierdząca bądź przecząca prowadzą do wypisania: liczby przęseł, oraz pozostałych prętów i słupków.



# Zapamiętanie dostępnej liczby słupków i prętów

```
#include <iostream>
using namespace std;
int main(){
    int slup, pret, pret_na_przes;
    cout << "Liczba slupkow: "; cin >> slup;
    cout << "Liczba pretow: "; cin >> pret;
    cout << "Liczba pretow w przesle: "; cin >> pret_na_przes;
    int przes = pret / pret_na_przes;
    if (przes > slup - 1){
        przes = slup - 1;
        int reszta_pret = pret - przes*pret_na_przes;
        cout << "Mozna zbudowac plot z " << slup - 1
             << " przesel" << endl;
        cout << "Pozostaje " << reszta_pret << " pretow"
             << endl;
    } else {
```

# Obliczenie i wypisanie liczby przęseł

```
        cout << "Mozna zbudowac plot z " << przes
             << " przesel" << endl;
        int reszta_pret = pret - przes*pret_na_przes;
        int reszta_slup = slup - przes - 1;
        cout << "Pozostaje " << reszta_slup
             << " slupkow oraz " << reszta_pret
             << " pretow" << endl;
    }
    return 0;
}
```

# Spis treści

# Operator warunkowy

Gdy kod instrukcji if-else jest prosty można zastąpić go jednym operatorem, tzw. operatorem warunkowym.

1. Operator warunkowy (ang. conditional operator) jest jedynym potrójnym operatorem języka c++.
2. Działa on z wykorzystaniem trzech wartości w przeciwieństwie do innych operatorów, które są w większości binarne (oddziałują na dwie wartości).
3. Format tego operatora jest następujący:

```
warunek ? instrukcja 1 : instrukcja 2;
```

4. Jeśli wyrażenie warunkowe jest prawdziwe, to wykonywana jest instrukcja 1, w przeciwnym razie, tzn. gdy wyrażenie warunkowe jest fałszywe wykonywana jest instrukcja 2.

# Operator warunkowy

1. O ile wyrażenie warunkowe jest tu typowym wrażeniem logicznym, to instrukcja 1 i instrukcja 2 są instrukcjami prostymi - żadna z nich nie może być instrukcją blokową.
2. Istnieje natomiast możliwość zagnieżdżania operatorów warunkowych.
3. Odpowiednikiem operatora:

```
warunek ? instrukcja 1 : instrukcja 2;
```

jest następująca instrukcja warunkowa

```
if(warunek)
    instrukcja 1;
else
    instrukcja 2;
```

# Przykład użycia operatora warunkowego

1. Należy jednak stwierdzić, że instrukcja warunkowa ma znacznie większe możliwości od operatora warunkowego i ten nieczęsto bywa stosowany.
2. Poniżej przedstawiono proste zastosowanie operatora warunkowego - instrukcję 1 operatora stanowi wywołanie funkcji sqrt, która oblicza pierwiastek z liczby rzeczywistej.

```
#include <iostream>
#include <cmath>
using namespace std;
int main(){
  double d;
  cout << "Podaj liczbę rzeczywistą: "; cin >> d;
  d >= 0 ? cout << sqrt(d) : cout << "liczba ujemna";
  return 0;
}
```

# Przykład użycia operatora warunkowego

Przykłady wykonania programu:

```
Podaj liczbę rzeczywistą: 7
2.64575
```

```
Podaj liczbę rzeczywistą: -7
liczba ujemna
```

Jako przykład zagnieżdżenia operatorów warunkowych może posłużyć fragment kodu:

```
double i1, i2;
cout << "Podaj dwie liczby całkowite: ";
cin >> i1 >> i2;
cout << "Większa liczba: ";
i1 < i2 ? cout << i2 : i1 > i2 ? cout << i1 :
                          cout << i1 << "=" << i2;
```

# Spis treści

# Instrukcja wielokrotnego wyboru

1. Za pomocą instrukcji warunkowej `if` możemy określić dokładnie co ma się wydarzyć w zależności od stanu jednej lub kilku zmiennych i instrukcja `if` daje pełną kontrolę nad przebiegiem programu.
2. W języku C++ jest jednak dostępna również instrukcja wielokrotnego wyboru `switch` umożliwiająca decyzje tylko i wyłącznie na podstawie wartości jednej zmiennej.
3. Możliwości instrukcji `switch` są nieporównywalnie mniejsze, jednak używanie jej w niektórych przypadkach jest znacznie korzystniejsze dla szybkości działania programu i estetyki kodu niż użycie instrukcji `if`.

# Instrukcja wielokrotnego wyboru switch

Zapoznajmy się ze składnią instrukcji wielokrotnego wyboru:

**c++: switch**

```
switch( expression ){
case firstValue:
    {...} break;
case secondValue:
    {...} break;
case nthValue:
    {...} break;
default:
    {...} // break;
}
```

gdzie {...} oznacza fragment kodu programu.

# Słowo kluczowe `switch`

Przeanalizujmy kolejno składnię instrukcji wielokrotnego wyboru. Na początek linia:

```
switch( expression )
```

- Wynik wyrażenia `expression` musi być typu podstawowego i to typu całkowitego lub dającego się odwzorować w typ całkowity (np. kod znaku ASCII).
- Wyrażenie w nawiasach okrągłych, musi się znaleźć się zaraz po wystąpieniu słowa kluczowego `switch`.
- Typ wyrażenia `expression` to jeden z typów: `char`, `short`, `int`, `long`, `long long` oraz typy wyliczeniowe definiowane słowem kluczowym `enum`

# Słowo kluczowe case

Teraz należy powiedzieć jak w zależności od wartości zmiennej `expression` wykonać jakiś fragment kodu.

Jak można się łatwo domyślić po składni instrukcji `switch` do tego celu służy słowo kluczowe `case`.

Ogólny zapis bloku `case` wygląda tak:

```
case firstValue:
    {...}
    break;
```

- Jeśli wynikiem wyrażenia `expression` jest wartość `firstValue` to sterowanie przechodzi do linii leżącej bezpośrednio po linii:

```
case firstValue:
```

# Instrukcja wielokrotnego wyboru `switch`

- Jeśli wynikiem wyrażenia `expression` jest wartość `secondValue` to sterowanie przechodzi do linii leżącej bezpośrednio po linii:

```
case secondValue:
```

  Podobnie jest dla wszystkich pozostałych linii zawierających słowo `case` z jednym wyjątkiem.
- Jeśli wynikiem wyrażenia `expression` jest wartość która nie występuje po żadnym słowie kluczowym `case` to sterowanie przechodzi do linii leżącej bezpośrednio sekwencji:

```
case default:
```

# Instrukcja wielokrotnego wyboru `switch`

- W każdym z w/w przypadków wykonywanie programu jest kontynuowane do pierwszego napotkania instrukcji `break;`, po czym następuje przejście poza blok `switch`
- Brak słowa `break` w ciągu wykonywanych instrukcji powoduje kontunuowanie wykonania aż do końca bloku `switch`.

# Przykład

Popatrzmy na kod programu:

```
#include <iostream>
using namespace std;
int main(){
    char ch; cout << "Podaj litere: "; cin >> ch;
    switch( ch ){
    case 'A':
        cout << "Podales A" << endl; break;
    case 'a':
        cout << "Podales a" << endl;
    default:
        cout << "Nie wiem co podales" << endl;
    }
    return 0;
}
```

# Przykład

I na trzy przypadki wykonania tego programu:

**Okno konsoli**

```
Podaj litere: A
Podales A
```

**Okno konsoli**

```
Podaj litere: x
Nie wiem co podales
```

**Okno konsoli**

```
Podaj litere: a
Podales a
Nie wiem co podales
```

# Instrukcja wielokrotnego wyboru switch

Pozostaje jeszcze problem inicjalizacji zmiennych w bloku instrukcji `switch`. Kod programu:

```
int main(){
    int i1; char ch;
    cout << "Podaj litere: "; cin >> ch;
    switch( ch ){
    case 'A':
        i1 = 7; break;
    default:
        i1 = 0;
    }
    cout << "i1 = " << i1 << endl;
    return 0;
}
```

jest poprawny

# Instrukcja wielokrotnego wyboru switch

Natomiast nie powiedzie się kompilacja kodu:

```
int main(){
    char ch;
    cout << "Podaj litere: "; cin >> ch;
    switch( ch ){
    case 'A':
        int i1 = 7;
        break;
    default:
        i1 = 0; }
    return 0; }
```

Przyczyną błędu kompilacji nie jest zakaz deklaracji zmiennych wewnątrz bloku switch, a zakaz inicjalizacji takich zmiennych.

# Instrukcja wielokrotnego wyboru switch

Tak więc program:

```
int main(){
    char ch;
    cout << "Podaj litere: "; cin >> ch;
    switch( ch ){
    case 'A':
        int i1;
        break;
    default:
        i1 = 0;
    }
    return 0;
}
```

skompiluje się poprawnie

# Instrukcja wielokrotnego wyboru switch

Dla zademonstrowania działania instrukcji switch rozważmy przykładowy kod programu:

```
#include <iostream>
using namespace std;
int main(){
    int i;
    cout << "Podaj liczbe calkowita: "; cin >> i;
    switch( 2*i-1 ){
    case 1:
        cout << "case 1" << endl; break;
    case 3:
        cout << "case 3" << endl; break;
    default:
        cout << "case default" << endl; }
    return 0; }
```

# Instrukcja wielokrotnego wyboru `switch`

A oto przykład użycia zaprezentowanego programu:

Okno konsoli

```
Podaj liczbe calkowita: 1
case 1
```

i drugi przykład:

Okno konsoli

```
Podaj liczbe calkowita: 2
case 3
```

oraz trzeci:

Okno konsoli

```
Podaj liczbe calkowita: 3
case default
```

# Spis treści

# Kalkulator z wykorzystaniem instrukcji switch

Kolejnym zadaniem wykorzystującym działanie instrukcji switch będzie bardzo prosty kalkulator. A oto kod programu:

```
#include <iostream>
using namespace std;
int main(){
    char ch;
    float varOne, varTwo;
    cout << "Podaj dwie liczby: ";
    cin >> varOne >> varTwo;
    cout << "Wybierz dzialanie: (+,-,* lub /): ";
    cin >> ch;
```

# Kalkulator z wykorzystaniem instrukcji switch

```
switch( ch ){
case '+':
    cout << varOne << " + " << varTwo << " = "
         << varOne + varTwo << endl;
    break;
case '-':
    cout << varOne << " - " << varTwo << " = "
         << varOne - varTwo << endl;
    break;
case '*':
    cout << varOne << " * " << varTwo << " = "
         << varOne * varTwo << endl;
    break;
```

# Kalkulator z wykorzystaniem instrukcji switch

```
    case '/':
        cout << varOne << " / " << varTwo << " = "
             << varOne / varTwo << endl;
        break;
    default:
        cout << "Bledne dzialanie!" << endl;
    }
    return 0;
}
```

# Kalkulator z wykorzystaniem instrukcji `switch`

I przykłady użycia kalkulatora:

**Okno konsoli**

```
Podaj dwie liczby: 1.25 3.14
Wybierz dzialanie: (+,-,* lub /): +
1.25 + 3.14 = 4.39
```

**Okno konsoli**

```
Podaj dwie liczby: 1.25 3.14
Wybierz dzialanie: (+,-,* lub /): *
1.25 * 3.14 = 3.925
```

**Okno konsoli**

```
Podaj dwie liczby: 17.25 5.55
Wybierz dzialanie: (+,-,* lub /): /
17.25 / 5.55 = 3.10811
```