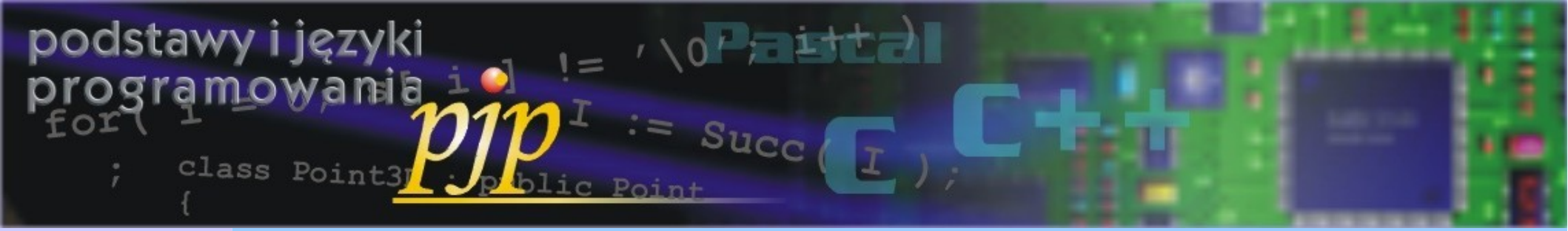


Podstawy programowania

Część trzecia

Instrukcje sterujące wykonaniem programu

Autor

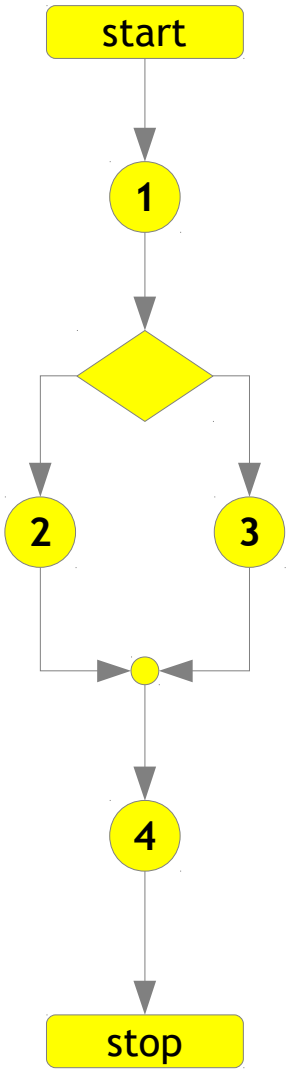
Roman Simiński

Kontakt

roman.siminski@us.edu.pl

www.programowanie.siminskionline.pl

Średnie spalanie — krótkie przypomnienie




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

int main()
{
  float dystans, paliwo;

  cout << endl << "Obliczam ile Twoj pojazd spala paliwa na 100 km" << endl;

  cout << "Dystans: " << flush;
  cin >> dystans;
  dystans = fabs( dystans );

  cout << "Paliwo: " << flush;
  cin >> paliwo;
  paliwo = fabs( paliwo );

  if( dystans == 0 )
    cout << "Nie dokonam obliczen dla zerowego dystansu" << endl;
  else
    cout << "Spalanie " << (paliwo*100)/dystans << "l na 100 km" << endl;

  cout << "Nacisnij Enter by zakonczyc program..." << endl;
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

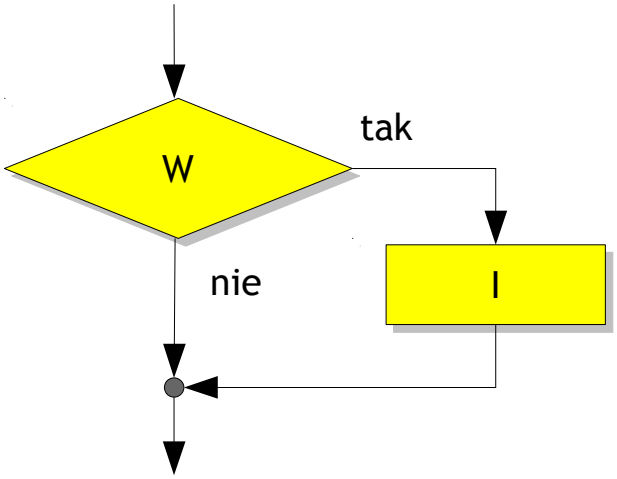
1 2 3 4

Instrukcja warunkowa i alternatywy

- Instrukcja *alternatywy* i *warunkowa* należą go grupy *instrukcji sterujących wykonaniem programu*. Wspólnie są nazywane *instrukcjami warunkowymi*.

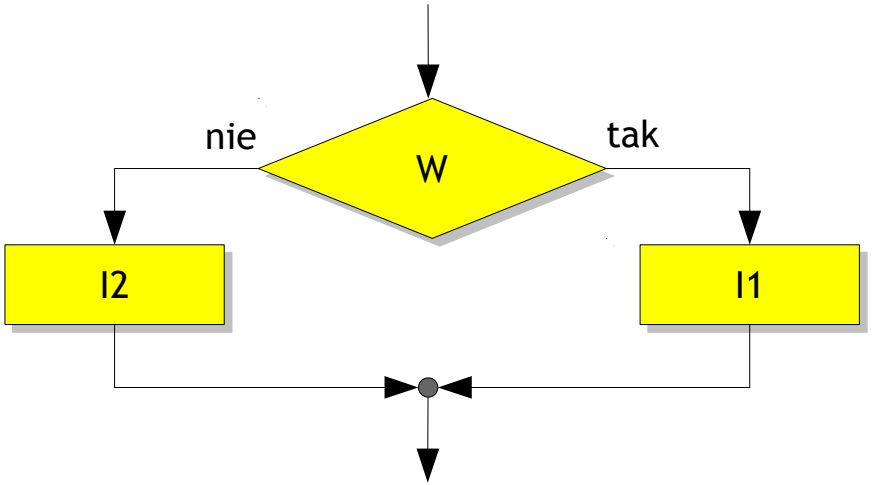
Instrukcja warunkowa

```
if( W )
  I
```



Instrukcja alternatywy

```
if( W )
  I1
else
  I2
```

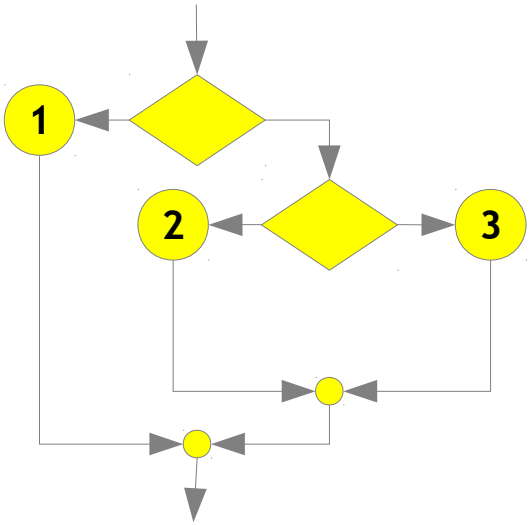


Instrukcja `if` oraz `if-else` obejmują swoim zasięgiem *jedną instrukcję* zapisaną dalej! Aby obejmowały *większą liczbę instrukcji*, trzeba je połączyć w instrukcję *grupującą*, tworzącą *blok instrukcji*.

Instrukcja alternatywy a instrukcje warunkowe

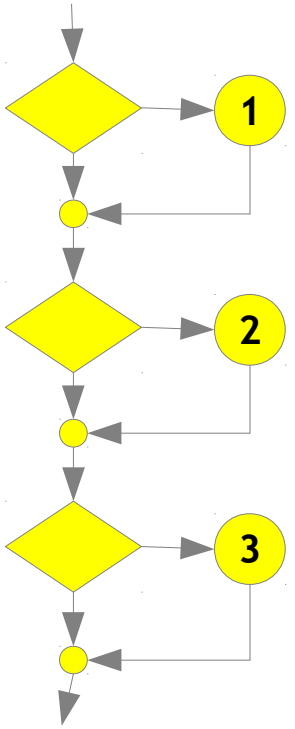
Złożenie instrukcji alternatywy:

```
if( delta < 0 )
  cout << "Brak pierwiastków rzeczywistych" << endl;   1
else
  if( delta == 0  )
    cout << "Jeden pierwiastek rzeczywisty" << endl;   2
  else
    cout << "Dwa pierwiastki rzeczywiste" << endl;     3
```



Kolejne instrukcje warunkowe:

```
if( delta < 0 )
  cout << "Brak pierwiastków rzeczywistych" << endl;   1

if( delta == 0  )
  cout << "Jeden pierwiastek rzeczywisty" << endl;     2

if( delta > 0  )
  cout << "Dwa pierwiastki rzeczywiste" << endl;       3
```



Operatory relacji:

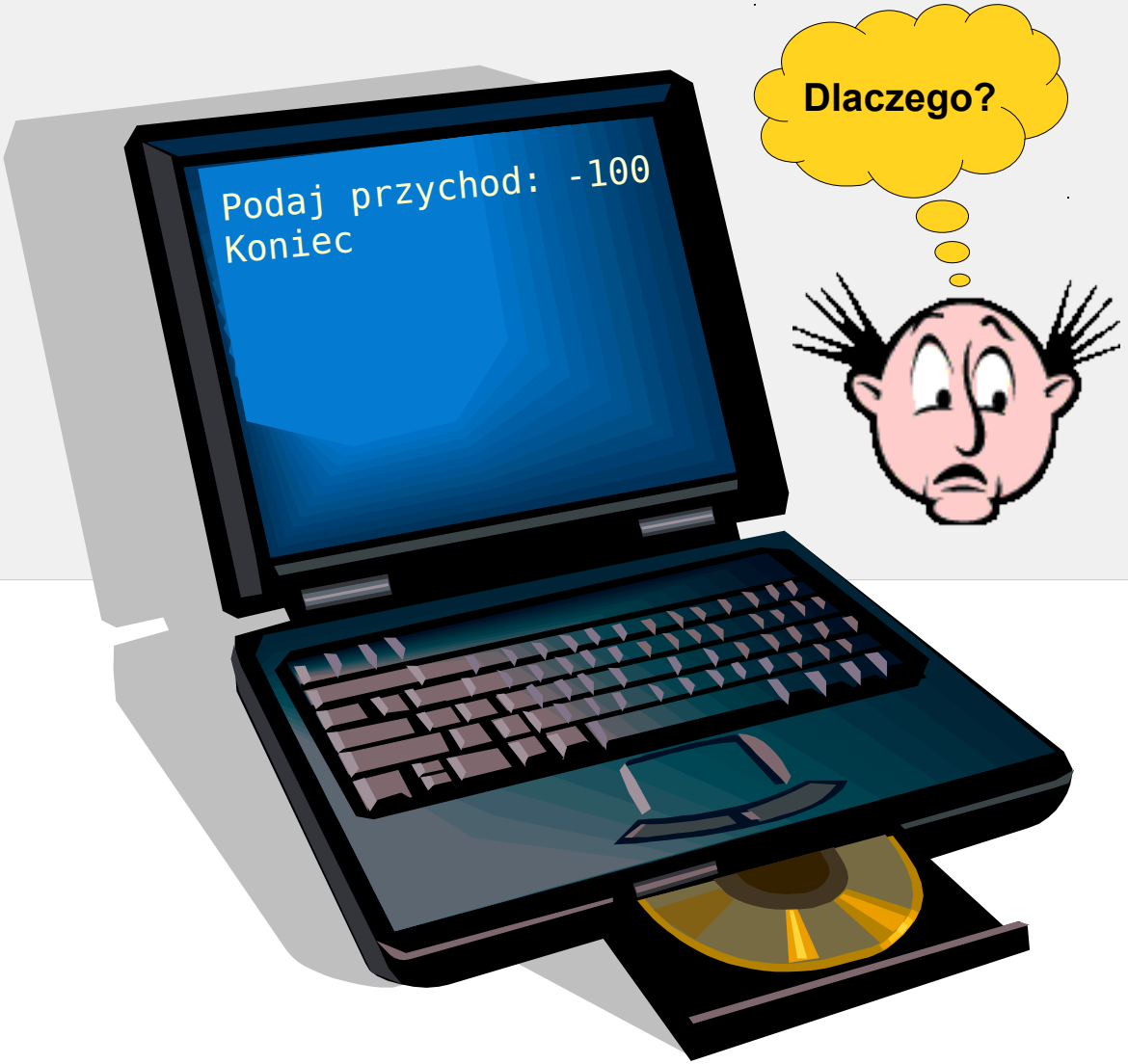
>	większy	>=	większy lub równy
<	mniejszy	<=	mniejszy lub równy

Uwaga na zagnieżdżone instrukcje warunkowe!

```
. . .
double kwota;

cout << "Podaj przychod: ";
cin >> kwota;

if( kwota >= 0 )
  if( kwota > 0 )
    cout << "Dochod";
else
  cout << "Strata";

cout << "Koniec”;
. . .
```

Uwaga na zagnieżdżone instrukcje warunkowe!

```
. . .
double kwota;

cout << "Podaj przychod: ";
cin >> kwota;

if( kwota >= 0 )
  if( kwota > 0 )
    cout << "Dochod";
else
  cout << "Strata";

cout << "Koniec”;
. . .
```



Uwaga na zagnieżdżone instrukcje warunkowe!

„Ify" z pułapką, wydaje się, że jest tak jak sugerują wcięcia:

```
if( kwota >= 0 )
  if( kwota > 0 )
    cout << "Dochod";
else
  cout << "Strata";
```

A jest tak:

```
if( kwota >= 0 )
  if( kwota > 0 )
    cout << "Dochod";
  else
    cout << "Strata";
```

Trzeba użyć instrukcji złożonej lub „sparować" *if* z *else*:

```
if( kwota >= 0 )
{
  if( kwota > 0 )
    cout << "Dochod";
}
else
  cout << "Strata";
```

```
if( kwota >= 0 )
  if( kwota > 0 )
    cout << "Dochod";
  else
    cout << "Zero!";
else
  cout << "Strata";
```

Instrukcja grupująca (złożona)

- Wykonanie instrukcji złożonej polega na *sekwencyjnym wykonaniu jej instrukcji wewnętrznych*.
- Instrukcja złożona tworzy z ciągu instrukcji jedną instrukcję, poprzez ujęcie tego ciągu w nawiasy **{** i **}**.
- Instrukcje wewnętrzne zakończone są znakiem średnika.

```
{
  cout << "Podaj promien kola: ";
  cin >> promien;
  cout << "Obwod: " << 2 * M_PI * promien;
}
```

Instrukcja złożona ma takie same uprawnienia jak pojedyncza instrukcja

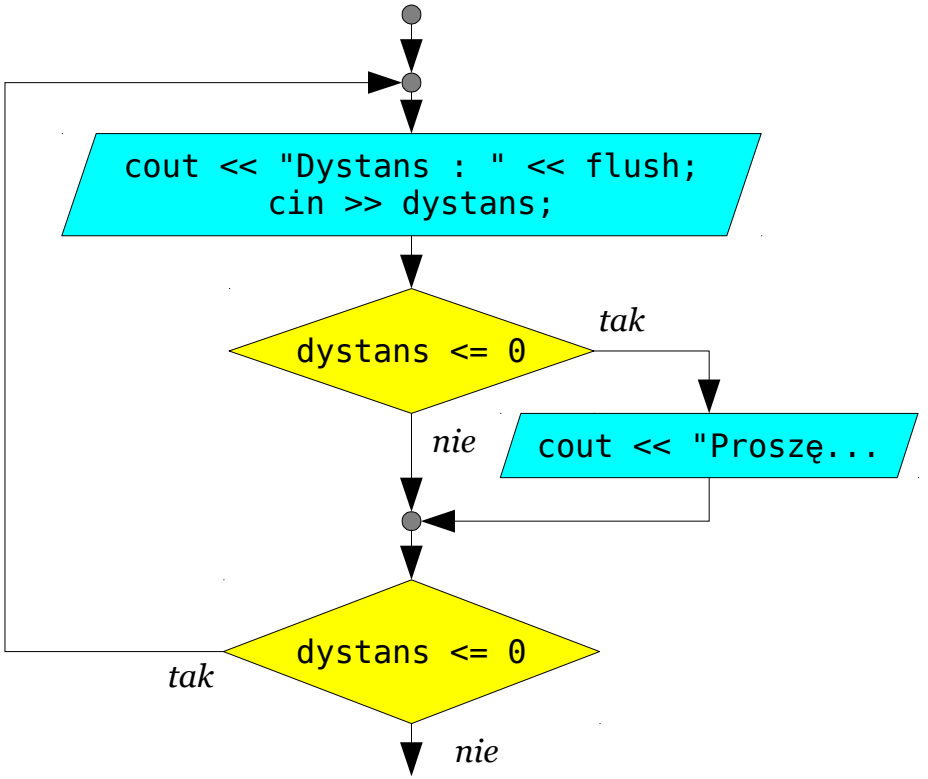
Średnie spalanie — nie pozwól na wprowadzenie błędnego dystansu

Program kontroluje wprowadzany dystans i nie pozwala na wprowadzenie liczby ujemnej i zera.

```
Obliczam ile Twoj pojazd spala paliwa na 100 km.
Wprowadz dystans - liczbe przejechanych kilometrow
oraz liczbe litrow paliwa na tym dystansie zuzytych.
Dystans : 0
Prosze wprowadzic prawidlowy dystans
Dystans : -200
Prosze wprowadzic prawidlowy dystans
Dystans : -100
Prosze wprowadzic prawidlowy dystans
Dystans : -1
Prosze wprowadzic prawidlowy dystans
Dystans : 0
Prosze wprowadzic prawidlowy dystans
Dystans : 500
Paliwo  : 37
Spalanie 7.40 l/100 km
Nacisnij Enter by zakonczyc program...
```

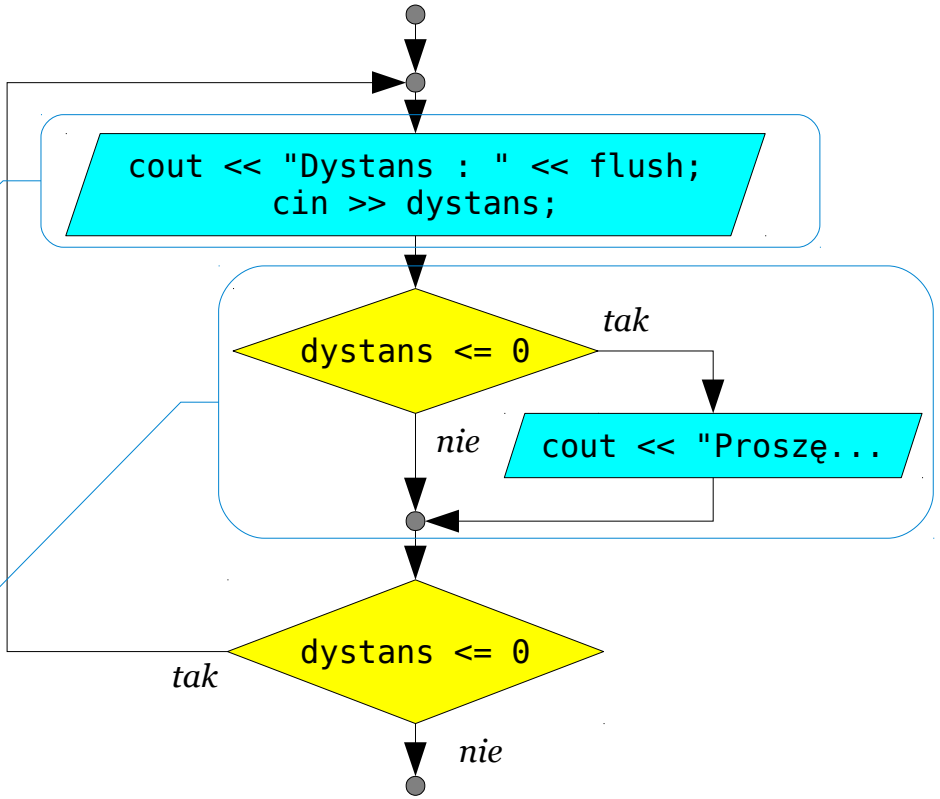
Wczytywanie dystansu — algorytm

```
Obliczam ile Twoj pojazd spala paliwa
Wprowadz dystans - liczbe przejechanyc
oraz liczbe litrow paliwa na tym dysta
Dystans : 0
Prosze wprowadzic prawidlowy dystans
Dystans : -500
Prosze wprowadzic prawidlowy dystans
Dystans : 500
Paliwo  : 35
Spalanie    7.00 l/100 km
Nacisnij enter by zakonczyc program...
_
```



Wczytywanie dystansu — jak to działa?

```
Obliczam ile Twoj pojazd spala paliwa
Wprowadz dystans - liczbe przejechanyc
oraz liczbe litrow paliwa na tym dysta
Dystans : 0
Prosze wprowadzic prawidlowy dystans
Dystans : -500
Prosze wprowadzic prawidlowy dystans
Dystans : 500
Paliwo  : 35
Spalanie   7.00 l/100 km
Nacisnij enter by zakonczyc program...
_
```

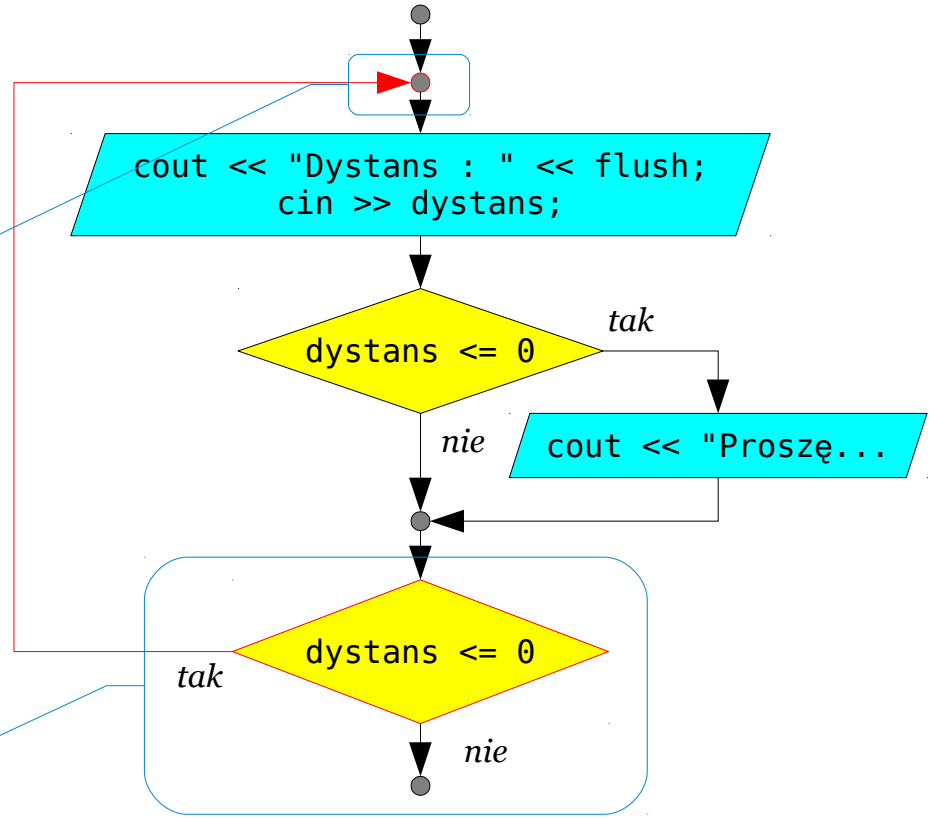



```
cout << "Dystans: " << flush;
cin >> dystans;
```

```
if( dystans <= 0 )
  cout << "Prosze wprowadzic prawidlowy dystans" << endl;
```

Wczytywanie dystansu — jak to działa?

```
Obliczam ile Twoj pojazd spala paliwa
Wprowadz dystans - liczbe przejechanyc
oraz liczbe litrow paliwa na tym dysta
Dystans : 0
Prosze wprowadzic prawidlowy dystans
Dystans : -500
Prosze wprowadzic prawidlowy dystans
Dystans : 500
Paliwo  : 35
Spalanie    7.00 l/100 km
Nacisnij enter by zakonczyc program...
_
```



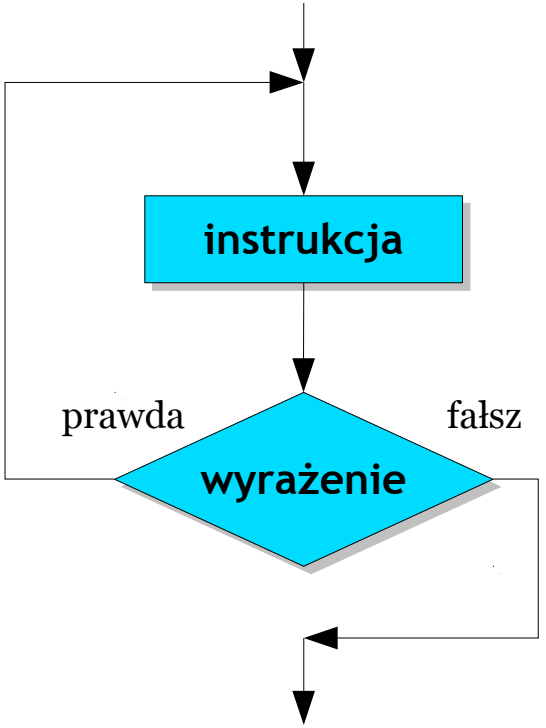

```
do
{
  cout << "Dystans: " << flush;
  cin >> dystans;

  if( dystans <= 0 )
    cout << "Prosze wprowadzic prawidlowy dystans" << endl;

}
while( dystans <= 0 );
```

Instrukcja iteracyjna do-while

Instrukcja iteracyjna pozwala na wielokrotne wykonywanie pewnego fragmentu kodu. Iteracja pozwala na „wracanie się” w trakcie wykonania programu.

Gdy iterowana jest jedna instrukcja:

```
do
  instrukcja
while( wyrażenie );
```



Gdy iterowany jest ciąg instrukcji:

```
do
{
  ciąg_instrukcji
}
while( wyrażenie );
```

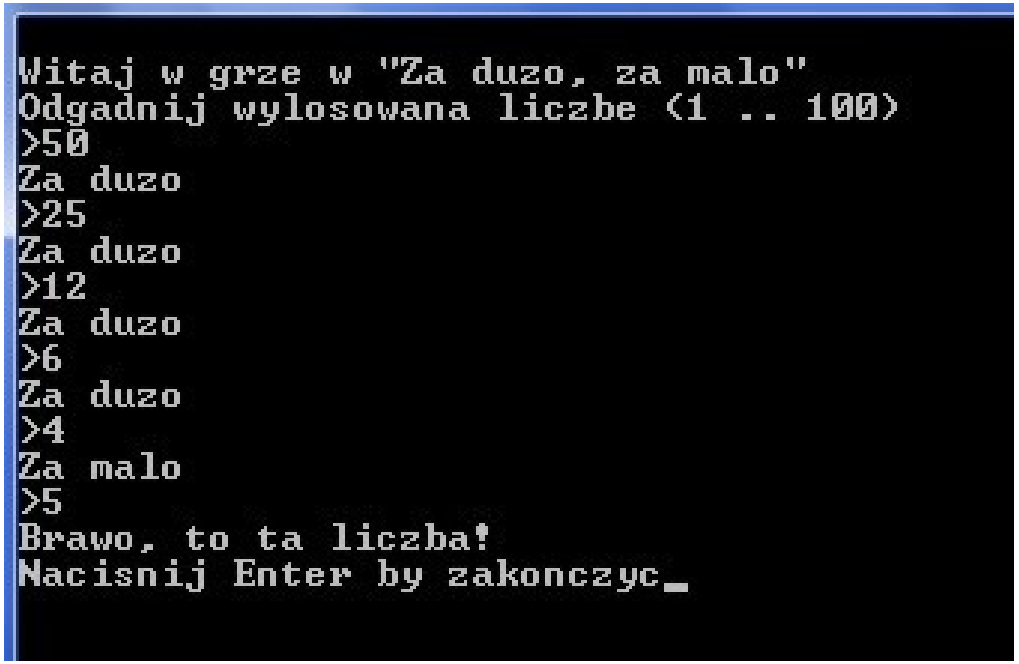
- Instrukcja stanowiąca ciało iteracji *do-while* wykona się przynajmniej raz.
- Wyrażenie występujące w nawiasach określa *warunek kontynuacji*, zatem iteracja *kończy się* gdy wartość wyrażenia będzie *zerowa*.

Zastosowanie instrukcji — gra w „za dużo, za mało”

Problem

Program losuje liczbę z przedziału od 1 do 100. Zadaniem użytkownika jest odgadnięcie wylosowanej liczby — wpisuje swoją propozycję a program stwierdza, czy proponowana liczba jest równa, mniejsza lub większa od wylosowanej.

Scenariusz działania programu:



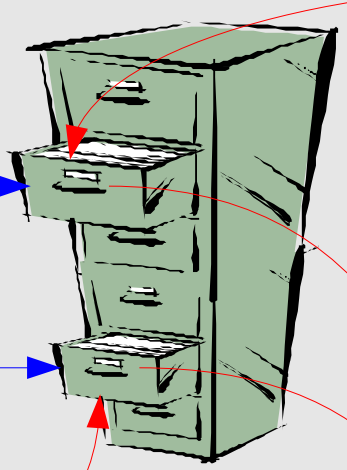
Zastosowanie instrukcji — gra w „za dużo, za mało”

Świat programisty

Zmienna `wczytana`

Zmienna `wylosowana`

Wylosuj liczbę z przedziału 1 .. 100



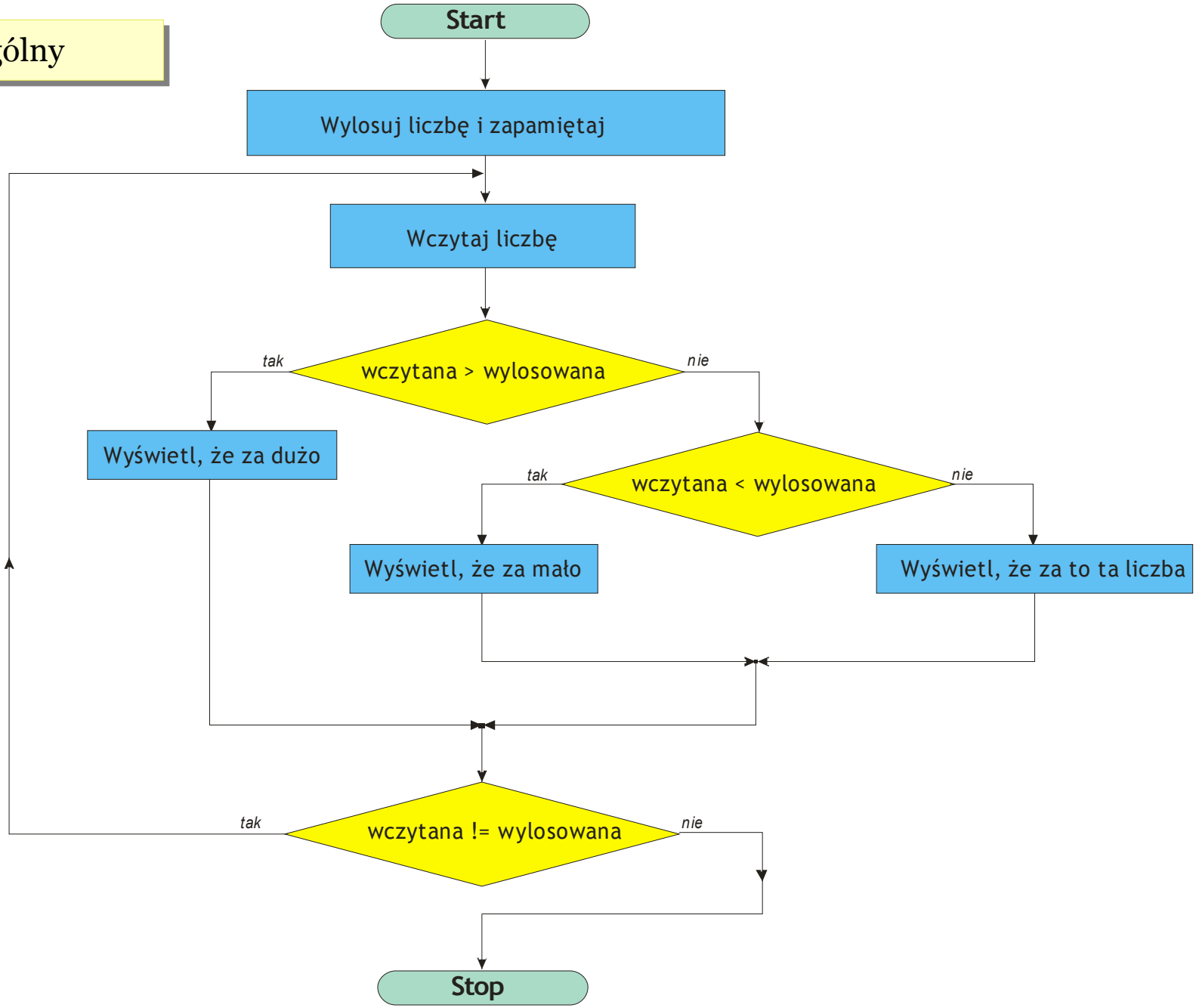
Porównaj wartości zmiennych `wczytana` i `wylosowana` Wyświetl odpowiedni komunikat

Odczytaj liczbę z klawiatury

Świat użytkownika

Zastosowanie instrukcji — gra w „za dużo, za mało”

Algorytm ogólny



Pierwsza przymiarka — funkcja main i deklaracja zmiennych

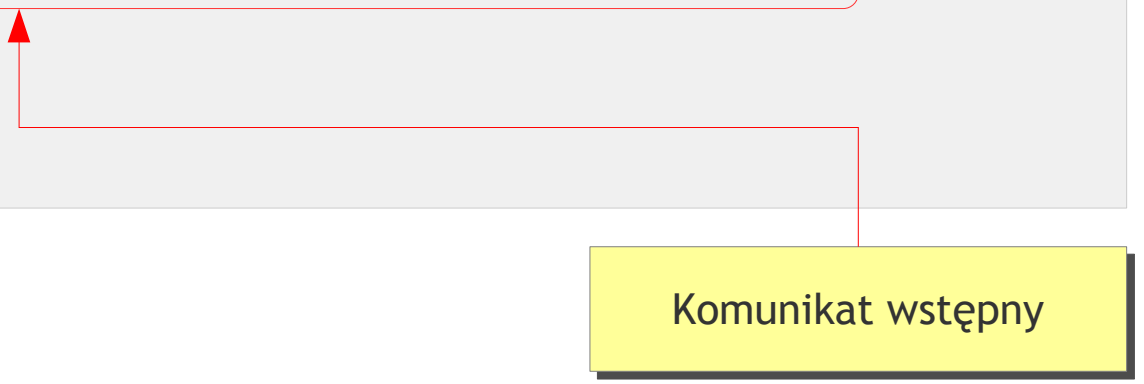
```
#include <iostream>
using namespace std;

int main()
{
  int wczytana, wylosowana;

  return EXIT_SUCCESS;
}
```

Będziemy korzystać z strumieni wejścia-wyjścia

W zmiennej *wylosowana* zapamiętamy liczbę wylosowaną przez komputer. W zmiennej *wczytana* zapamiętamy liczbę wprowadzoną przez gracza.

Pierwsza przymiarka — komunikat wstępny

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

int main()
{
  int wczytana, wylosowana;

  cout << endl << "Witaj w grze w \"Za duzo, za malo\"";
  cout << endl << "Odgadnij wylosowana liczbe (1 .. 100)" << endl;


  return EXIT_SUCCESS;
}
```



Pierwsza przymiarka — komunikat wstępny

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

int main()
{
  int wczytana, wylosowana;

  cout << endl << "Witaj w grze w \"Za duzo, za malo\"";
  cout << endl << "Odgadnij wylosowana liczbe (1 .. 100)" << endl;


  return EXIT_SUCCESS;
}
```

Znak ” jest ogranicznikiem napisu.
Aby uzyskać ten znak w napisie, trzeba użyć symbolu specjalnego \”

Pierwsza przymiarka — losowanie liczby

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

int main()
{
  int wczytana, wylosowana;

  cout << endl << "Witaj w grze w \"Za duzo, za malo\"";
  cout << endl << "Odgadnij wylosowana liczbe (1 .. 100)" << endl;

  srand( 1000 );

  wylosowana = rand() % 100 + 1;

  return EXIT_SUCCESS;
}
```

Losowanie zalążka generatora liczb pseudolosowych. Stały zalążek nie jest dobry... :-\

Losowanie liczby z wykorzystaniem generatora liczb pseudolosowych.

Pierwsza przymiarka — losowanie liczby, komentarz

Inicjalizacja generatora liczb pseudolosowych (stały zalążek o wartości 1000):

```
srand( 1000 );
```

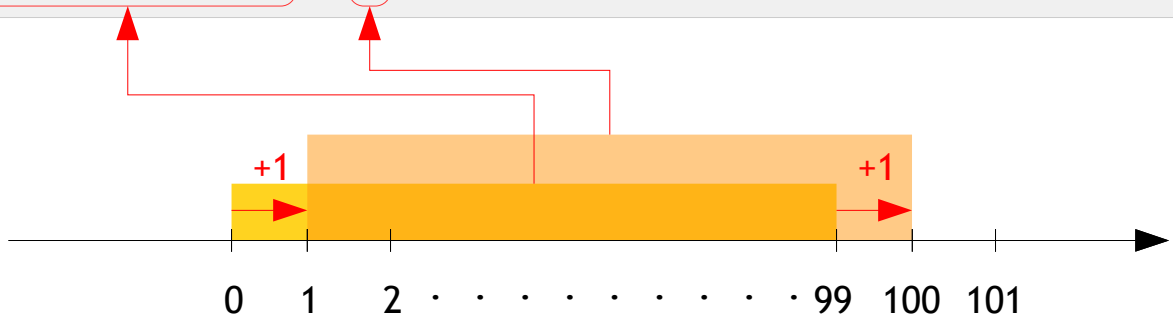
Tak zainicjowany generator będzie działał powtarzalnie. Należy uzmiennić zalążek, przykładowo uzależniając go od bieżącego czasu:

```
srand( ( unsigned )time( NULL ) );
```

Funkcja *rand*() generuje liczby pseudolosowe z przedziału 0..*RAND_MAX*. My potrzebujemy liczby od 1..100.

Ograniczamy zakres używając operatora *modulo* (reszta z dzielenia) oznaczonego w C/C++ symbolem % oraz przesuwamy przedział o jeden w lewo dodając +1.

```
wylosowana = rand() % 100 + 1;
```



Pierwsza przymiarka — lepszy dobór zalążka generatora pseudolosowego

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

int main()
{
  int wczytana, wylosowana;

  cout << endl << "Witaj w grze w \"za duzo, za malo\"";
  cout << endl << "Odgadnij wylosowana liczbe (1 .. 100)" << endl;

  srand( ( unsigned )time( NULL ) );

  wylosowana = rand() % 100 + 1;

  return EXIT_SUCCESS;
}
```

Plik nagłówkowy zwykle potrzebny dla fun. *time*

Losowanie zalążka generatora liczb pseudolosowych zależnego ob aktualnej wartości timera

Główna iteracja programu

```
int main()
{
  . . .
  wylosowana = rand() % 100 + 1;

  do
  {
    cout << '>' << flush;
    cin >> wczytana;

    if( wczytana > wylosowana )
      cout << "Za duzo" << endl;
    else
      if( wczytana < wylosowana )
        cout << "Za malo" << endl;
      else
        cout << "Brawo, to ta liczba!";
  }
  while( wylosowana != wczytana );

  cout << "Nacisnij Enter by zakonczyc";
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

Wykonuj, dopóki liczba nie jest odgadnięta

Kontakt z graczem

```
int main()
{
  . . .
  wylosowana = rand() % 100 + 1;

  do
  {
    cout << '>' << flush;
    cin >> wczytana;

    if( wczytana > wylosowana )
      cout << "Za duzo" << endl;
    else
      if( wczytana < wylosowana )
        cout << "Za malo" << endl;
      else
        cout << "Brawo, to ta liczba!";
  }
  while( wylosowana != wczytana );

  cout << "Nacisnij Enter by zakonczyc";
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

Wczytaj liczbę typowaną przez gracza

Oceń wprowadzoną liczbę

```
int main()
{
  . . .
  wylosowana = rand() % 100 + 1;

  do
  {
    cout << '>' << flush;
    cin >> wczytana;

    if( wczytana > wylosowana )
      cout << "Za duzo" << endl;
    else
      if( wczytana < wylosowana )
        cout << "Za malo" << endl;
      else
        cout << "Brawo, to ta liczba!";
  }
  while( wylosowana != wczytana );

  cout << "Nacisnij Enter by zakonczyc";
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

Porównaj wczytaną liczbę z wylosowaną i wyprowadź odpowiedni komunikat

Od instrukcji warunkowej do instrukcji przełączającej

Problem

Napisać program realizujący funkcje prostego kalkulatora, pozwalającego na wykonywanie operacji dodawania, odejmowania, mnożenia i dzielenia na dwóch liczbach rzeczywistych.

Program ma identyfikować sytuację wprowadzenia błędnego symbolu działania oraz próbę dzielenia przez zero.

Scenariusz działania programu

```
Wykonuje operacje arytmetyczne na dwoch liczbach.
Podaj pierwsza liczbe: 100
Podaj dzialanie [+ - * /]: +
Podaj druga liczbe: 100
Wynik: 200.0000

Nacisnij Enter by zakonczyc...
```

```
Wykonuje operacje arytmetyczne na dwoch liczbach.
Podaj pierwsza liczbe: 100
Podaj dzialanie [+ - * /]: /
Podaj druga liczbe: 0
Dzielenie przez zero nie jest dozwolone!

Nacisnij Enter by zakonczyc...
```

```
Wykonuje operacje arytmetyczne na dwoch liczbach.
Podaj pierwsza liczbe: 100
Podaj dzialanie [+ - * /]: &
Podaj druga liczbe: 100
Niedozwolone dzialanie

Nacisnij Enter by zakonczyc...
```

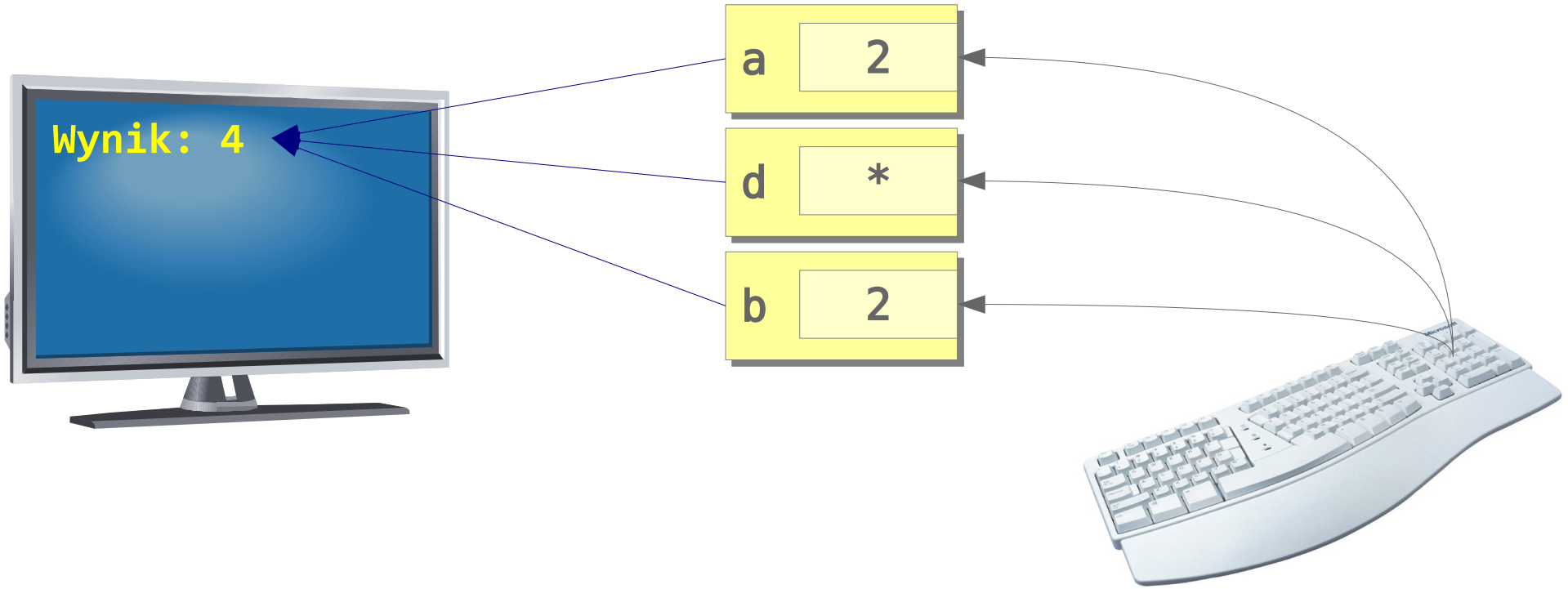
```
Wykonuje operacje arytmetyczne na dwoch liczbach.
Podaj pierwsza liczbe: 1
Podaj dzialanie [+ - * /]: +
Podaj druga liczbe: 1
Wynik: 2.0000
Nacisnij P by powtorzyc obliczenia, inny klawisz by zakonczyc...p
Podaj pierwsza liczbe: 2
Podaj dzialanie [+ - * /]: *
Podaj druga liczbe: 2
Wynik: 4.0000
Nacisnij P by powtorzyc obliczenia, inny klawisz by zakonczyc...
```

Czego potrzebujemy?

Do realizacji programu potrzebne są:

- dwie zmienne numeryczne (zapamiętane liczb-argumentów),
- zmienna znakowa (zapamiętanie znaku oznaczającego działanie).

Niech zmienne nazywają się: *a* — liczba pierwsza, *b* — liczba druga, *d* — działanie.



Dobór typów dla zmiennych a i b

Zmienne *a* i *b* powinny być jednego z podstawowych typów rzeczywistych:

- *float* — typ rzeczywisty pojedynczej prowizji,
- *double* — typ rzeczywisty podwójnej precyzji,
- long double — typ rzeczywisty podwójnej pracyzji o powiększonym zakresie.

Typ *double* zapewni odpowiedni zakres dla obliczeń.

```
double a, b;
```

Zakres typu *float* to zwykle od 3.4×10^{38} do 3.4×10^{-38} (7 cyfr).

Zakres typu *double* to zwykle od 1.7×10^{-308} do 1.7×10^{308} (15 cyfr).

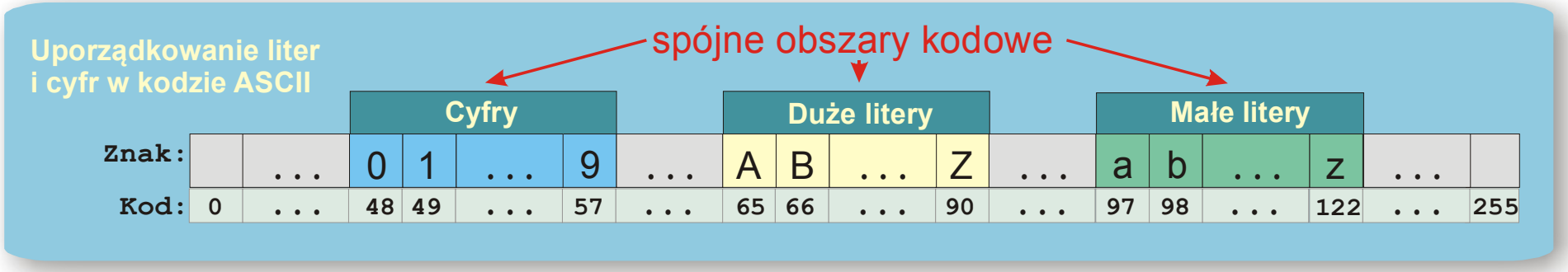
Zakres typu *long double* jest czasem identyczny jak typu *double*, niektóre implementacje języka C++ oferują zakres od 3.4×10^{-4932} do 1.1×10^{4932} (18 cyfr).

Dobór typów dla zmiennej d

Zmienne *d* ma przechowywać informacje o wprowadzonym z klawiatury symbolu działania arytmetycznego. Symbol ten jest *znakiem*. Do reprezentacji znaków służy typ *char* (skrót od ang. *character*).

char — typ znakowy, obejmujący zbiór znaków używanych do komunikacji z człowiekiem (monitor, klawiatura, drukarka, tekstowe transfery sieciowe).

- Zakres wartości : konkretny wykaz znaków oraz sposób ich uporządkowania zależy od języka programowania i specyfiki platformy sprzętowej i systemowej.
- Jednak najpopularniejsze jest kodowanie znaków według ASCII (*American Standard Code for Information Interchange*) — wykorzystywane w C i C++.



Literały znakowe a literały łańcuchowe

Literał — to dana, której wartość wynika z jej zapisu w kodzie programu.

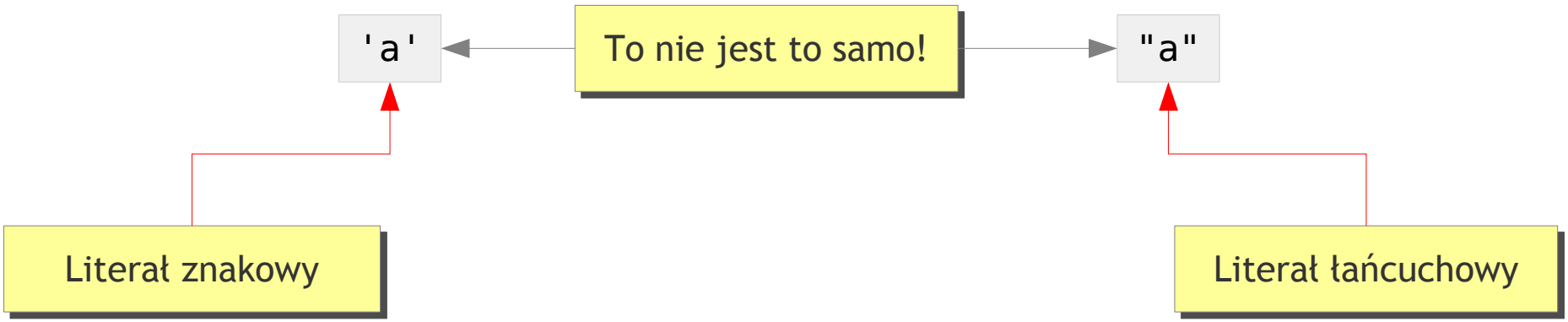
- Literały znakowe:

```
'a'    '1'    '%'    '&'    'A'    '.'    '*'    ' '    'R'
```

- Literały łańcuchowe:

```
"C++"      "Liczba"   "12345"    "programowanie"    "program"
```

Nie można mylić *literałów znakowych* z *literałami łańcuchowymi*!



Kalkulator — wersja pierwsza

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

int main()
{
  double a, b;
  char   d;

  cout << endl << "Wykonuje dzialania na dwoch liczbach.";
  cout << endl << "Wczytam pierwsza liczbe, symbol dzialania";
  cout << endl << "potem druga liczbe i wyswietle wynik.";
  cout << endl << "Dozwolone dzialania: + - * /" << endl;

  cout << "Podaj pierwsza liczbe: ";
  cin >> a;

  cout << "Podaj dzialanie [* - * /]: ";
  cin >> d;

  cout << "Podaj druga liczbe: ";
  cin >> b;
```

Deklaracje zmiennych

1/2

Kalkulator — wersja pierwsza

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

int main()
{
  double a, b;
  char   d;

  cout << endl << "Wykonuje dzialania na dwoch liczbach.";
  cout << endl << "Wczytam pierwsza liczbe, symbol dzialania";
  cout << endl << "potem druga liczbe i wyswietle wynik.";
  cout << endl << "Dozwolone dzialania: + - * /" << endl;

  cout << "Podaj pierwsza liczbe: ";
  cin >> a;

  cout << "Podaj dzialanie [* - * /]: ";
  cin >> d;

  cout << "Podaj druga liczbe: ";
  cin >> b;
```

Komunikat wstępny

1/2

Kalkulator — wersja pierwsza

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

int main()
{
  double a, b;
  char   d;

  cout << endl << "Wykonuje dzialania na dwoch liczbach.";
  cout << endl << "Wczytam pierwsza liczbe, symbol dzialania";
  cout << endl << "potem druga liczbe i wyswietle wynik.";
  cout << endl << "Dozwolone dzialania: + - * /" << endl;

  cout << "Podaj pierwsza liczbe: ";
  cin >> a;

  cout << "Podaj dzialanie [* - * /]: ";
  cin >> d;

  cout << "Podaj druga liczbe: ";
  cin >> b;
```

Wczytanie danych

1/2

Kalkulator — wersja pierwsza


```
if( d == '+' || d == '-' || d == '*' || d == '/' )
{
  if( d == '+' )
    cout << "Wynik: " << a + b << endl;
  if( d == '-' )
    cout << "Wynik: " << a - b << endl;
  if( d == '*' )
    cout << "Wynik: " << a * b << endl;
  if( d == '/' )
    if( b != 0 )
      cout << "Wynik: " << a / b << endl;
    else
      cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
}
else
  cout << "Niedozwolone dzialanie!" << endl;

cout << endl << "Nacisnij Enter by zakonczyc..." << endl;
cin.ignore();
cin.get();

return EXIT_SUCCESS;
}
```

Operatory logicze

`||`	**or**	alternatywa
`&&`	**and**	koniunkcja
`!`	**not**	negacja

Czy wprowadzono prawidłowy symbol działania?

Kalkulator — wersja pierwsza

2/2

```
  if( d == '+' || d == '-' || d == '*' || d == '/' )
  {
    if( d == '+' )
      cout << "Wynik: " << a + b << endl;
    if( d == '-' )
      cout << "Wynik: " << a - b << endl;
    if( d == '*' )
      cout << "Wynik: " << a * b << endl;
    if( d == '/' )
      if( b != 0 )
        cout << "Wynik: " << a / b << endl;
      else
        cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
  }
  else
    cout << "Niedozwolone dzialanie!" << endl;

  cout << endl << "Nacisnij Enter by zakonczyc..." << endl;
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

Identyfikacja działania, wyznaczenie wartości i wyprowadzenie do strumienia wyjściowego

Kalkulator — wersja pierwsza


```
  if( d == '+' || d == '-' || d == '*' || d == '/' )
  {
    if( d == '+' )
      cout << "Wynik: " << a + b << endl;
    if( d == '-' )
      cout << "Wynik: " << a - b << endl;
    if( d == '*' )
      cout << "Wynik: " << a * b << endl;
    if( d == '/' )
      if( b != 0 )
        cout << "Wynik: " << a / b << endl;
      else
        cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
  }
  else
    cout << "Niedozwolone dzialanie!" << endl;

  cout << endl << "Nacisnij Enter by zakonczyc..." << endl;
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

Dzielenie o ile dzielnik jest niezerowy

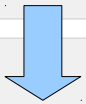
Kalkulator — wersja pierwsza, wada


```
  if( d == '+' || d == '-' || d == '*' || d == '/' )
  {
    if( d == '+' )
      cout << "Wynik: " << a + b << endl;
    if( d == '-' )
      cout << "Wynik: " << a - b << endl;
    if( d == '*' )
      cout << "Wynik: " << a * b << endl;
    if( d == '/' )
      if( b != 0 )
        cout << "Wynik: " << a / b << endl;
      else
        cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
  }
  else
    cout << "Niedozwolone dzialanie!" << endl;

  cout << endl << "Nacisnij Enter by zakonczyc..." << endl;
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

Dużo instrukcji **if** i **if-else**

Wprowadzamy instrukcję przełączającą

```
if( d == '+' )
  cout << "Wynik: " << a + b << endl;
if( d == '-' )
  cout << "Wynik: " << a - b << endl;
if( d == '*' )
  cout << "Wynik: " << a * b << endl;
if( d == '/' )
  if( b != 0 )
    cout << "Wynik: " << a / b << endl;
  else
    cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
```



```
switch( d )
{
  case '+' : cout << "Wynik: " << a + b << endl;
             break;
  case '-' : cout << "Wynik: " << a - b << endl;
             break;
  case '*' : cout << "Wynik: " << a * b << endl;
             break;
  case '/' : if( b != 0 )
               cout << "Wynik: " << a / b << endl;
             else
               cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
             break;
}
```

Jak działa instrukcja przełączająca

```
switch( d )
{
  case '+' : cout << "Wynik: " << a + b << endl;
             break;

  case '-' : cout << "Wynik: " << a - b << endl;
             break;

  case '*' : cout << "Wynik: " << a * b << endl;
             break;

  case '/' : if( b != 0 )
               cout << "Wynik: " << a / b << endl;
             else
               cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
             break;
}
```

Wyznaczenie selektora – wartości wyrażenia zapisanego w nawiasach. W tym przypadku sprawdzenie zawartości zmiennej **d**

Jak działa instrukcja przełączająca

```
switch( d )
{
  case '+' : cout << "Wynik: " << a + b << endl;
             break;

  case '-' : cout << "Wynik: " << a - b << endl;
             break;

  case '*' : cout << "Wynik: " << a * b << endl;
             break;

  case '/' : if( b != 0 )
               cout << "Wynik: " << a / b << endl;
             else
               cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
             break;
}
```

Poszukiwanie frazy **case** po którym występuje literał równy wartości selektora instrukcji **switch**

Jak działa instrukcja przełączająca

```
switch( d )
{
  case '+' : cout << "Wynik: " << a + b << endl;
             break;

  case '-' : cout << "Wynik: " << a - b << endl;
             break;

  case '*' : cout << "Wynik: " << a * b << endl;
             break;

  case '/' : if( b != 0 )
               cout << "Wynik: " << a / b << endl;
             else
               cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
             break;
}
```

Załóżmy, że zmienna **d** == '—'. Instrukcja **switch** przełącza sterowanie do odpowiedniego przypadku.

Jak działa instrukcja przełączająca

```
switch( d )
{
  case '+' : cout << "Wynik: " << a + b << endl;
             break;

  case '-' : cout << "Wynik: " << a - b << endl;
             break;

  case '*' : cout << "Wynik: " << a * b << endl;
             break;

  case '/' : if( b != 0 )
               cout << "Wynik: " << a / b << endl;
             else
               cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
             break;
}
```

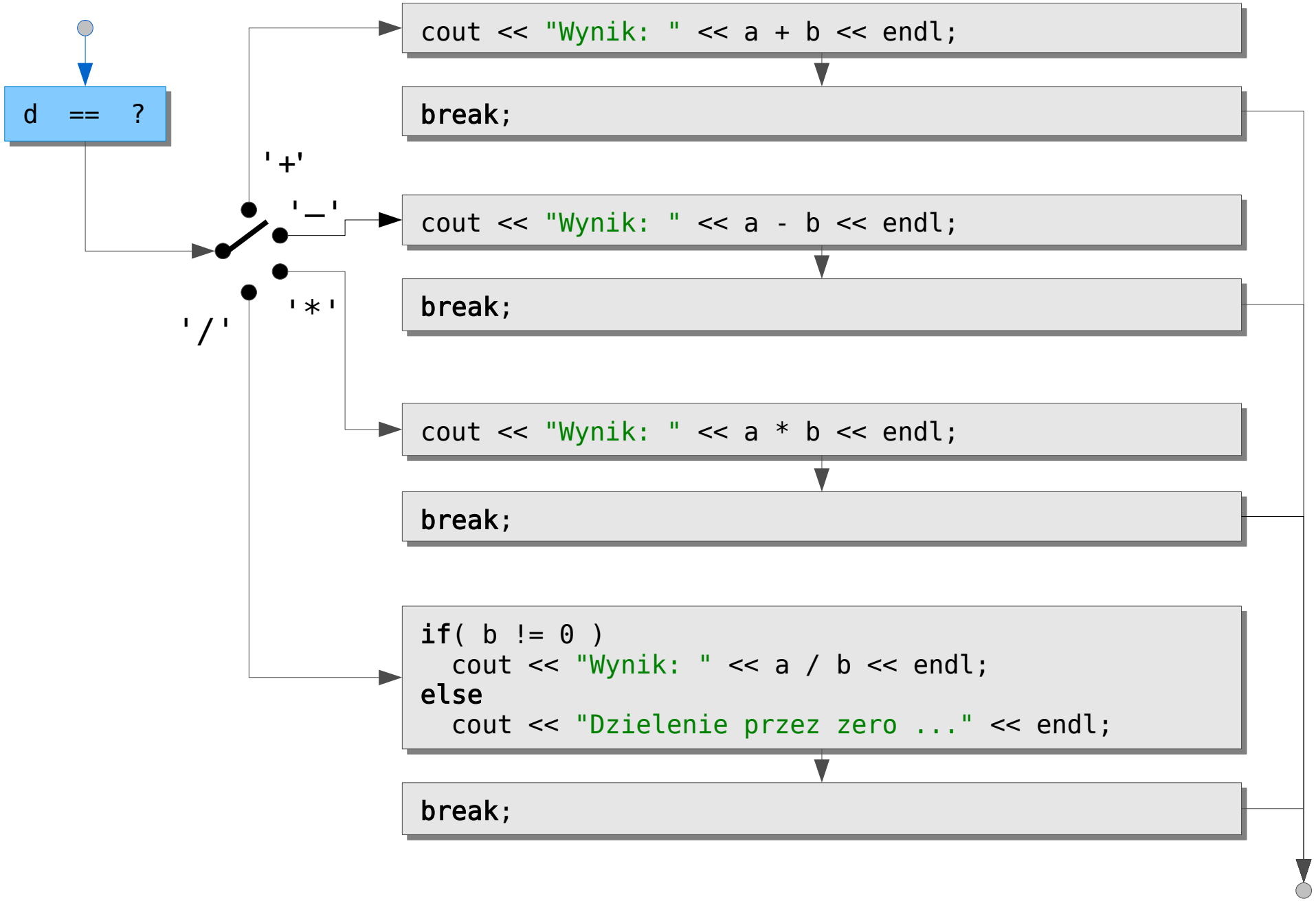
Wykonywane są kolejne instrukcje, począwszy od pierwszej instrukcji przypadku zgodnego z selektorem.

Jak działa instrukcja przełączająca

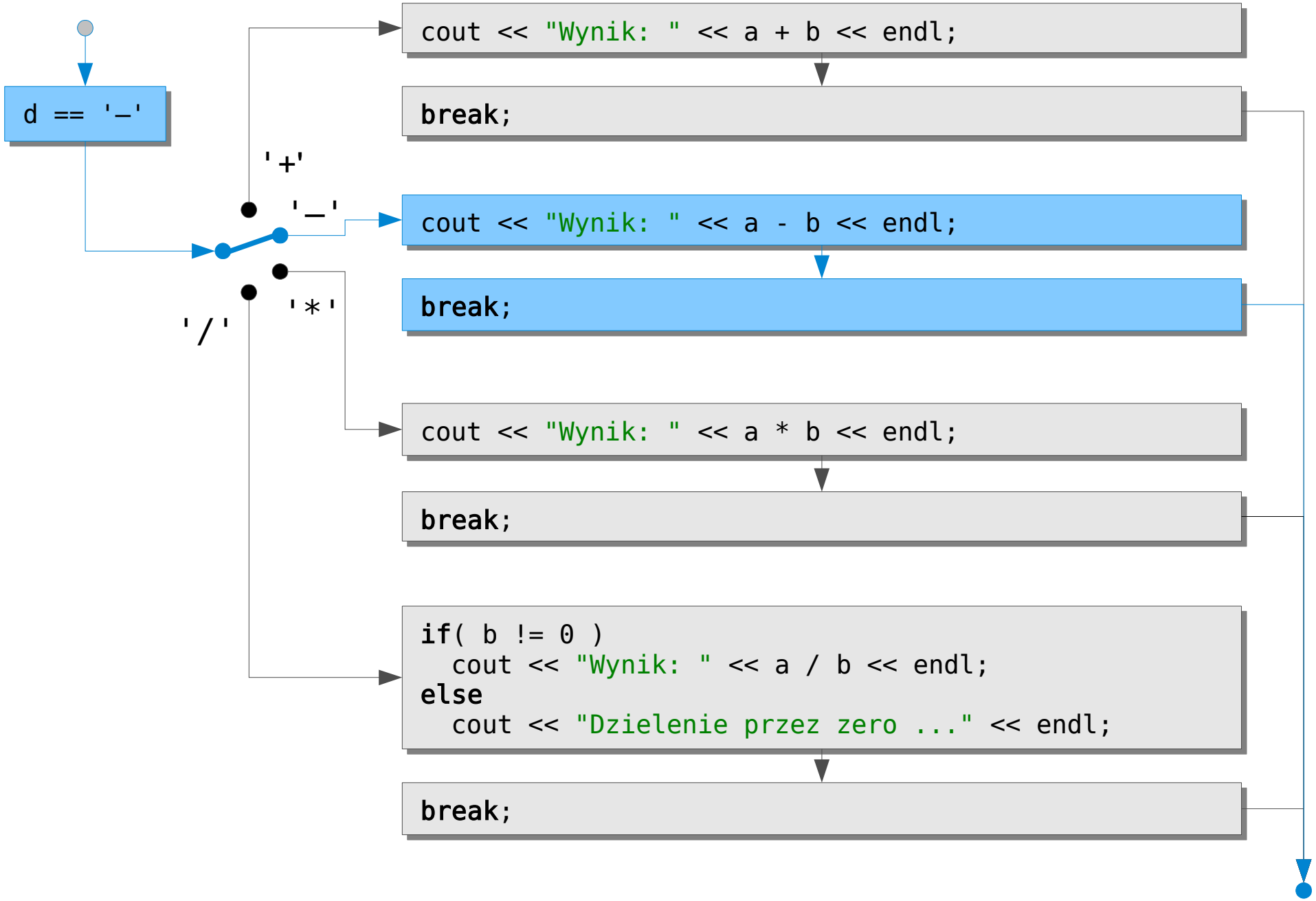
```
switch( d )
{
  case '+' : cout << "Wynik: " << a + b << endl;
             break;

  case '-' : cout << "Wynik: " << a - b << endl;
             break;

  case '*' : cout << "Wynik: " << a * b << endl;
             break;

  case '/' : if( b != 0 )
               cout << "Wynik: " << a / b << endl;
             else
               cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
             break;
}
```

Instrukcja break powoduje wyjście (zakończenie wykonania) z najbliższej instrukcji iteracyjnej lub instrukcji **switch**.

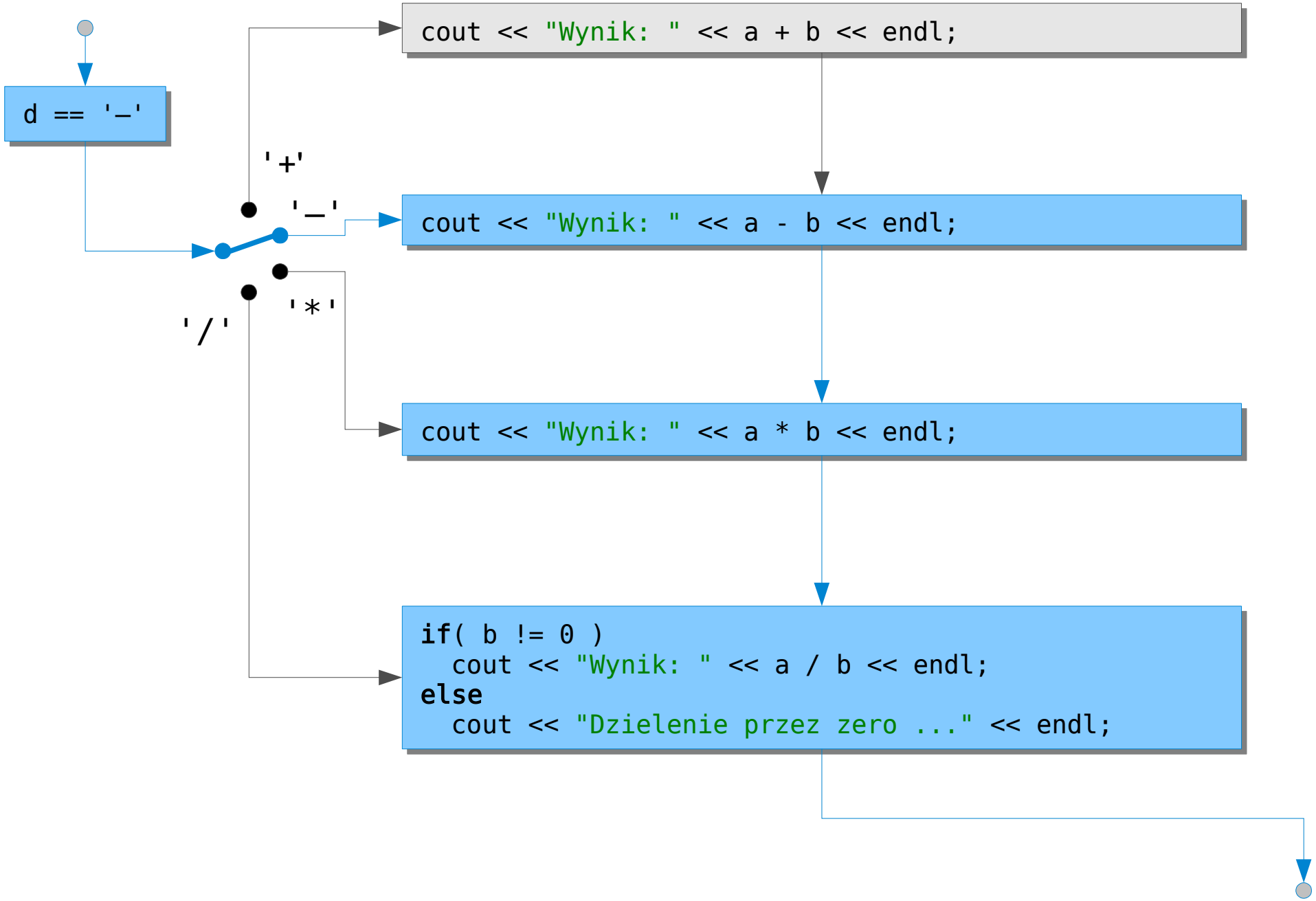
Jak działa instrukcja przełączająca



Jak działa instrukcja przełączająca



Gdyby nie było instrukcji break



Co się stanie gdy selektor nie pasuje do żadnego z przypadków?

```
switch( d )
{
  case '+' : cout << "Wynik: " << a + b << endl;
             break;

  case '-' : cout << "Wynik: " << a - b << endl;
             break;

  case '*' : cout << "Wynik: " << a * b << endl;
             break;

  case '/' : if( b != 0 )
               cout << "Wynik: " << a / b << endl;
             else
               cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
             break;
}
```

Załóżmy, że d == '$' – co się wtedy stanie?

Co się stanie gdy selektor nie pasuje do żadnego z przypadków?

```
switch( d )
{
  case '+' : cout << "Wynik: " << a + b << endl;
             break;

  case '-' : cout << "Wynik: " << a - b << endl;
             break;

  case '*' : cout << "Wynik: " << a * b << endl;
             break;

  case '/' : if( b != 0 )
               cout << "Wynik: " << a / b << endl;
             else
               cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
             break;
}
```

Gdy selektor nie pasuje do żadnego przypadków, żadna instrukcja nie zostanie wykonana.

Co się stanie gdy selektor nie pasuje do żadnego z przypadków?

```
switch( d )
{
  case '+' : cout << "Wynik: " << a + b << endl;
             break;

  case '-' : cout << "Wynik: " << a - b << endl;
             break;

  case '*' : cout << "Wynik: " << a * b << endl;
             break;

  case '/' : if( b != 0 )
               cout << "Wynik: " << a / b << endl;
             else
               cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
             break;
  default  : . . .
}
```

Można wprowadzić przypadek domyślny – tutaj zostanie skierowane sterowanie gdy selektor nie pasuje do żadnego z przypadków

Wykorzystanie przypadku domyślnego

```
switch( d )
{
  case '+' : cout << "Wynik: " << a + b << endl;
             break;

  case '-' : cout << "Wynik: " << a - b << endl;
             break;

  case '*' : cout << "Wynik: " << a * b << endl;
             break;

  case '/' : if( b != 0 )
               cout << "Wynik: " << a / b << endl;
             else
               cout << "Dzielenie przez zero nie jest dozwolone!" << endl;
             break;
  default  : cout << "Niedozwolone dzialanie!" << endl;
             break;
}
```

Przypadek domyślny można wykorzystać do identyfikacji niedozwolonego symbolu działania

Instrukcja switch może zastąpić wiele instrukcji warunkowych

```
if( d == '+' || d == '-' || d == '*' || d == '/' )
{
  if( d == '+' )
    cout << "Wynik: " << a + b << endl;
  if( d == '-' )
    cout << "Wynik: " << a - b << endl;
  if( d == '*' )
    cout << "Wynik: " << a * b << endl;
  if( d == '/' )
    if( b != 0 )
      cout << "Wynik: " << a / b << endl;
    else
      cout <<
}
else
  cout << "Ni
```

```
switch( d )
{
  case '+' : cout << "Wynik: " << a + b << endl;
             break;
  case '-' : cout << "Wynik: " << a - b << endl;
             break;
  case '*' : cout << "Wynik: " << a * b << endl;
             break;
  case '/' : if( b != 0 )
               cout << "Wynik: " << a / b << endl;
             else
               cout << "Dzielenie przez zero nie jest dozwolone!" << endl;

             break;
  default  : cout << "Niedozwolone dzialanie!" << endl;
             break;
}
```

Suplement — znaki specjalne

- Sekwencje specjalne pozwalają na reprezentowanie znaków nie posiadających swoich legalnych symboli graficznych.
- Dodatkowo sekwencje specjalne są wykorzystywane do zapisu pewnych „niewygodnych” stałych znakowych.

Sekwencja	Wartość	Znak	Znaczenie
\a	0x07	BEL	Audible bell
\b	0x08	BS	Backspace
\f	0x0C	FF	Formfeed
\n	0x0A	LF	Newline (linefeed)
\r	0x0D	CR	Carriage return
\t	0x09	HT	Tab (horizontal)
\v	0x0B	VT	Vertical tab
\\	0x5c	\	Backslash
\'	0x27	'	Apostrof
\"	0x22	"	Cudzysłów
\?	0x3F	?	Pytajnik
\O		any	O = łańcuch ósemkowych cyfr
\xH		any	H = łańcuch szesnastkowych cyfr
\XH		any	H = łańcuch szesnastkowych cyfr

Instrukcja przełączająca switch + iteracja do-while = proste menu

Połączenie instrukcji iteracyjnej *do-while* oraz instrukcji przełączającej *switch* pozwala na zorganizowanie prostego, ale użytecznego, menu konsolowego.

```
Formatowanie dysku, wybierz opcje:
1. Format
2. Szybki format
3. Diagnostyka
4. Koniec
>2

Wybrales szybki format

Formatowanie dysku, wybierz opcje:
1. Format
2. Szybki format
3. Diagnostyka
4. Koniec
>3

Wybrales diagnostyke

Formatowanie dysku, wybierz opcje:
1. Format
2. Szybki format
3. Diagnostyka
4. Koniec
>
```

Instrukcja przełączająca switch + iteracja do-while = proste menu

```
. . .
char klawisz;

do
{
  cout << "\nFormatowanie dysku, wybierz opcje:\n1. Format";
  cout << "\n2. Szybki format\n3. Diagnostyka\n4. Koniec\n>" << flush;

  cin >> klawisz;
  switch( klawisz )
  {
     case '1' : cout << "\nWybrales formatowanie\n";
                . . . tu formatowanie . . .
                break;
     case '2' : cout << "\nWybrales szybki format\n";
                . . . tu szybki format . . .
                break;
     case '3' : cout << "\nWybrales diagnostyke\n";
                . . . tu diagnostyka . . .

                break;
  }
}
while( klawisz != '4' );
. . .
```

Główna iteracja sterująca wykonaniem programu

Instrukcja przełączająca switch + iteracja do-while = proste menu

```
. . .
char klawisz;

do
{
  cout << "\nFormatowanie dysku, wybierz opcje:\n1. Format";
  cout << "\n2. Szybki format\n3. Diagnostyka\n4. Koniec\n>" << flush;

  cin >> klawisz;
  switch( klawisz )
  {
     case '1' : cout << "\nWybrales formatowanie\n";
                . . . tu formatowanie . . .
                break;
     case '2' : cout << "\nWybrales szybki format\n";
                . . . tu szybki format . . .
                break;
     case '3' : cout << "\nWybrales diagnostyke\n";
                . . . tu diagnostyka . . .

                break;
  }
}
while( klawisz != '4' );
. . .
```

Wczytanie znaku identyfikującego wybraną przez użytkownika opcję

Instrukcja przełączająca switch + iteracja do-while = proste menu

```
. . .
char klawisz;

do
{
  cout << "\nFormatowanie dysku, wybierz opcje:\n1. Format";
  cout << "\n2. Szybki format\n3. Diagnostyka\n4. Koniec\n>" << flush;

  cin >> klawisz;
  switch( klawisz )
  {
     case '1' : cout << "\nWybrales formatowanie\n";
                . . . tu formatowanie . . .
                break;
     case '2' : cout << "\nWybrales szybki format\n";
                . . . tu szybki format . . .
                break;
     case '3' : cout << "\nWybrales diagnostyke\n";
                . . . tu diagnostyka . . .

                break;
  }
}
while( klawisz != '4' );
. . .
```

Identyfikacja znaku i wykonanie odpowiedniej akcji

Typ zmiennej selektora a typ wartości przypadku

```
char klawisz;
. . .
cin >> klawisz;
switch( klawisz )
{
   case '1' : cout << "\nWybrales formatowanie\n";
              break;
   case '2' : cout << "\nWybrales szybki format\n";
              break;
   case '3' : cout << "\nWybrales diagnostyke\n";
              break;
}
```

```
int klawisz;
. . .
cin >> klawisz;
switch( klawisz )
{
   case 1 : cout << "\nWybrales formatowanie\n";
            break;
   case 2 : cout << "\nWybrales szybki format\n";
            break;
   case 3 : cout << "\nWybrales diagnostyke\n";
            break;
}
```

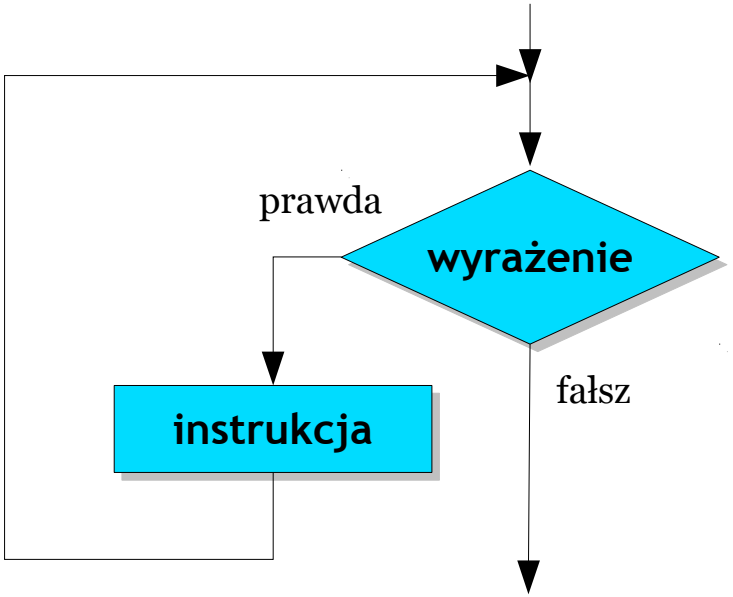
Instrukcja iteracyjna while

Gdy iterowana jest jedna instrukcja:

```
while( wyrażenie )
  instrukcja
```

Gdy iterowany jest ciąg instrukcji:

```
while( wyrażenie )
{
  ciąg instrukcji
}
```



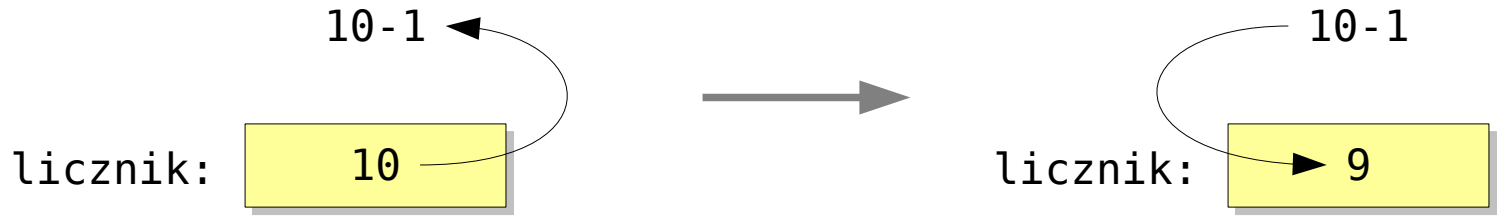
- Instrukcja stanowiąca ciało iteracji *while* może nie *wykonać się wcale*.
- Wyrażenie występujące w nawiasach określa warunek kontynuacji, zatem iteracja kończy się gdy wartość wyrażenia będzie *zerowa*.

Instrukcja iteracyjna while — przykład I

```
int licznik = 10;

while( licznik > 0 )
{
  cout << endl << licznik << "...";
  licznik = licznik - 1;
}
cout << endl << "Nowy Rok!!!" << endl;
```

```
10...
9...
8...
7...
6...
5...
4...
3...
2...
1...
Nowy Rok !!!
```

- Iteracja wykona się 10 razy, liczbą wykonań steruje zmienna *licznik*.
- W każdym przebiegu wartość zmiennej jest zmniejszana o 1.



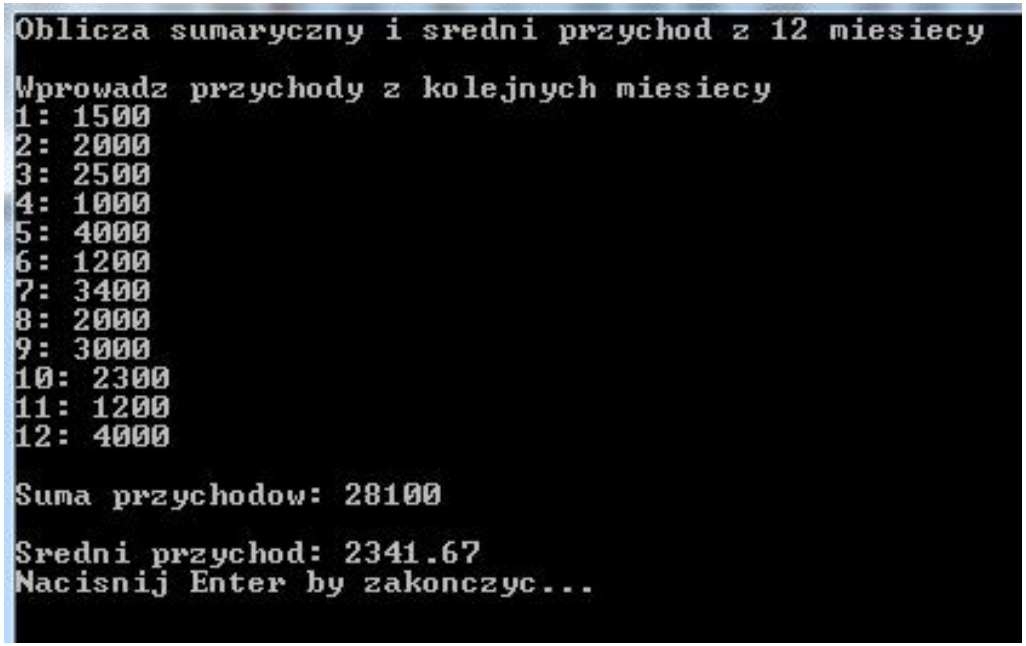
- W C/C++ zamiast `licznik = licznik – 1` napiszemy `--licznik` lub `licznik--`
- Podobnie zamiast `licznik = licznik + 1` napiszemy `++licznik` lub `licznik++`

Instrukcja iteracyjna while — przykład II

Problem

Podatnik osiąga w każdym miesiącu roku podatkowego przychód. Należy napisać program wczytujący przychody z kolejnych 12-tu miesięcy i wyznaczający przychód sumaryczny oraz średni.

Scenariusz działania programu



Instrukcja iteracyjna while — przykład II

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

int main()
{
  double przychod, suma = 0;
  int nr_miesiaca;

  cout << "\nObliczam sumaryczny i sredni przychod z 12 miesiecy\n";
  cout << "\nWprowadz przychody z kolejnych miesiecy\n";
```

Zmienne robocze programu
Wyzerowanie zmiennej suma jest bardzo ważne

Komunikat wstępny

Instrukcja iteracyjna while — przykład II

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

int main()
{
  double przychod, suma = 0;
  int nr_miesiaca;

  cout << "\nObliczam sumaryczny i sredni przychod z 12 miesiecy\n";
  cout << "\nWprowadz przychody z kolejnych miesiecy\n";

  nr_miesiaca = 1;


     cout << nr_miesiaca << ": ";
     cin >> przychod;
```

Wczytanie przychodu styczniowego

Instrukcja iteracyjna while — przykład II

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

int main()
{
  double przychod, suma = 0;
  int nr_miesiaca;

  cout << "\nObliczam sumaryczny i sredni przychod z 12 miesiecy\n";
  cout << "\nWprowadz przychody z kolejnych miesiecy\n";

  nr_miesiaca = 1;


     cout << nr_miesiaca << ": ";
     cin >> przychod;
     suma = suma + przychod;
```

Dodanie wczytanego przychodu do sumy przychodów

Instrukcja iteracyjna while — przykład II

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

int main()
{
  double przychod, suma = 0;
  int nr_miesiaca;

  cout << "\nObliczam sumaryczny i sredni przychod z 12 miesiecy\n";
  cout << "\nWprowadz przychody z kolejnych miesiecy\n";

  nr_miesiaca = 1;
  while( nr_miesiaca <= 12 )
  {
     cout << nr_miesiaca << ": ";
     cin >> przychod;
     suma = suma + przychod;
     ++nr_miesiaca;
  }
```

Iteracja wczytująca przychody z 12-tu miesięcy

Instrukcja iteracyjna while — przykład II

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

int main()
{
  double przychod, suma = 0;
  int nr_miesiaca;

  cout << "\nObliczam sumaryczny i sredni przychod z 12 miesiecy\n";
  cout << "\nWprowadz przychody z kolejnych miesiecy\n";

  nr_miesiaca = 1;
  while( nr_miesiaca <= 12 )
  {
     cout << nr_miesiaca << ": ";
     cin >> przychod;
     suma = suma + przychod;
     ++nr_miesiaca;
  }
  cout << "\nSuma przychodow: " << suma;
  cout << "\nSredni przychod: " << suma / 12;

  cout << endl << "Nacisnij Enter by zakonczyc..." << endl;
  cin.ignore(); cin.get();
  return EXIT_SUCCESS;
}
```

Wyświetlenie wyników

Instrukcja while a do-while

- Zazwyczaj można iteracje **while** i **do-while** stosować zamiennie. W tym przypadku nie można jednoznacznie wskazać, która wersja jest lepsza.

```
nr_miesiaca = 1;
while( nr_miesiaca <= 12 )
{
  cout << nr_miesiaca << ": ";
  cin >> przychod;

  suma = suma + przychod;

  ++nr_miesiaca;
}
```

```
nr_miesiaca = 1;
do
{
   cout << nr_miesiaca << ": ";
   cin >> przychod;

   suma = suma + przychod;

   ++nr_miesiaca;
}
while( nr_miesiaca <= 12 );
```

- W C/C++ zamiast `suma = suma + przychod` napiszemy `suma += przychod`

Warto pewne wartości parametryzować

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

int main()
{
  double przychod, suma = 0;
  int nr_miesiaca;

  cout << "\nObliczam sumaryczny i sredni przychod z 12 miesiecy\n";
  cout << "\nWprowadz przychody z kolejnych miesiecy\n";

  nr_miesiaca = 1;
  while( nr_miesiaca <= 12 )
  {
     cout << nr_miesiaca << ": ";
     cin >> przychod;
     suma += przychod;
     ++nr_miesiaca;
  }
  cout << "\nSuma przychodow: " << suma;
  cout << "\nSredni przychod: " << suma / 12 ;

  cout << endl << "Nacisnij Enter by zakonczyc..." << endl;
  cin.ignore(); cin.get();
  return EXIT_SUCCESS;
}
```

Te wartości warto sparametryzować

Warto pewne wartości parametryzować

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

int main()
{
  const int LB_MIESIECY = 12;
  double przychod, suma = 0;
  int nr_miesiaca;

  cout << "\nObliczam sumaryczny i s
  cout << "\nWprowadz przychody z kolejnych miesiecy\n";

  nr_miesiaca = 1;
  while( nr_miesiaca <= LB_MIESIECY )
  {
     cout << nr_miesiaca << ": ";
     cin >> przychod;
     suma += przychod;
     ++nr_miesiaca;
  }
  cout << "\nSuma przychodow: " << suma;
  cout << "\nSredni przychod: " << suma / LB_MIESIECY;
  cout << endl << "Nacisnij Enter by zakonczyc..." << endl;
  cin.ignore(); cin.get();
  return EXIT_SUCCESS;
}
```

Deklaracja dziwnej zmiennej — zmiennej o wartości ustalonej. Możemy powiedzieć, że LB_MIESIECY to *stała*.

Zwyczajowo nazwy stałych zapisuje się inaczej, aby odróżnić je od zwykłych zmiennych.

Odwołanie do wartości stałej

Od szczegółu do ogółu

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

int main()
{
  const int LB_MIESIECY = 12;
  double przychod, suma = 0;
  int
  cou
  cou
  nr_
  whi
  {
     cin >> przychod;
     suma += przychod;
     ++nr_miesiaca;
  }
  cout << "\nSuma przychodow: " << suma;
  cout << "\nSredni przychod: " << suma / LB_MIESIECY ;
  cout << endl << "Nacisnij Enter by zakonczyc..." << endl;
  cin.ignore(); cin.get();
  return EXIT_SUCCESS;
}
```

Program wyznaczający sumaryczny i średni przychód jest szczególnym przypadkiem programu typu:

Napisz program wyznaczający sumę i wartość średnią ciągu N liczb, gdzie N jest pewną stałą o określonej wartości, np. 20.

Od szczegółu do ogółu

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

int main()
{
  const int N = 20;
  double liczba, suma = 0;
  int licznik;

  cout << "\nObliczam sume i srednia ciagu N=20 liczb\n";
  cout << "\nWprowadz kolejne liczby\n";

  licznik = 1;
  while( licznik <= N )
  {
     cout << licznik << ": ";
     cin >> liczba;
     suma += liczba;
     ++licznik;
  }
  cout << "\nSuma liczb: " << suma;
  cout << "\nSrednia : " << suma / N;
  cout << endl << "Nacisnij Enter by zakonczyc..." << endl;
  cin.ignore(); cin.get();
  return EXIT_SUCCESS;
}
```

Od szczegółu do ogółu

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

int main()
{
  const int N = 20;
  double liczba, suma = 0;
  int

  cou
  cou

  lic
  whi
  {
     cin >> liczba;
     suma += liczba;
     ++licznik;
  }
  cout << "\nSuma liczb: " << suma;
  cout << "\nSrednia : " << suma / N;
  cout << endl << "Nacisnij Enter by zakonczyc..." << endl;
  cin.ignore(); cin.get();
  return EXIT_SUCCESS;
}
```

A gdyby zadanie zostało zmienione:

Napisz program wyznaczający sumę i wartość średnią ciągu N liczb, gdzie N jest nie jest z góry znane i program powinien je wczytać tuż po uruchomieniu.

Od szczegółu do ogółu

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

int main()
{
  const int n = 20;
  double liczba, suma = 0;
  int licznik;

  cout << "\nObliczam sume i srednia ciagu N liczb\n";
  cout << "\nWprowadz ile ma byc liczb: ";
  cin >> n;

  licznik = 1;
  while( licznik <= n )
  {
     cout << licznik << ": ";
     cin >> liczba;
     suma += liczba;
     ++licznik;
  }
  cout << "\nSuma liczb: " << suma;
  cout << "\nSrednia : " << suma / n;
  . . .;
}
```

N powinno być teraz zwykłą zmienną
Można też zmienić jej nazwę, żeby nie sugerowała, że jest stałą.

Wprowadzenie liczby liczb ;)

To jest niebezpieczne miejsce!
Dlaczego?

Uwaga na pozornie proste zmiany w programie!

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

int main()
{
  int n;
  double liczba, suma = 0;
  int licznik;

  cout << "\nObliczam sume i srednia ciagu N liczb\n";
  cout << "\nWprowadz ile ma byc liczb: ";
  cin >> n;

  licznik = 1;
  while( licznik <= n )
  {
     cout << licznik << ": ";
     cin >> liczba;
     suma += liczba;
     ++licznik;
  }
  cout << "\nSuma liczb: " << suma;
  cout << "\nSrednia : " << suma / n;
  . . .;
}
```

Użytkownik powinien wprowadzić wartość dodatnią.
Ale może np. wprowadzić 0.

Uwaga na pozornie proste zmiany w programie!

Iteracja while się obroni – ma wartownika, kontrolującego warunek przed pierwszym wejściem do wnętrza.

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

int main()
{
  int n;
  double liczba, suma = 0;
  int licznik;

  cout << "\nObliczam sume i srednia ciagu N liczb\n";
  cout << "\nWprowadz ile ma byc liczb: ";
  cin >> n;

  licznik = 1;
  while( licznik <= n )
  {
     cout << licznik << ": ";
     cin >> liczba;
     suma += liczba;
     ++licznik;
  }
  cout << "\nSuma liczb: " << suma;
  cout << "\nSrednia : " << suma / n;
  . . .;
}
```

Uwaga na pozornie proste zmiany w programie!

W przypadku n==0 wystąpi błąd dzielenia przez zero

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

int main()
{
  int n;
  double liczba, suma = 0;
  int licznik;

  cout << "\nObliczam sume i srednia ciagu N liczb\n";
  cout << "\nWprowadz ile ma byc liczb: ";
  cin >> n;

  licznik = 1;
  while( licznik <= n )
  {
     cout << licznik << ": ";
     cin >> liczba;
     suma += liczba;
     ++licznik;
  }
  cout << "\nSuma liczb: " << suma;
  cout << "\nSrednia : " << suma / n;
  . . .;
}
```

Obrona przed nieprawidłowymi danymi

```
int main()
{
  . . .
  cout << "\nObliczam sume i srednia ciagu N liczb\n";
  cout << "\nWprowadz ile ma byc liczb: ";
  cin >> n;

  licznik = 1;
  while( licznik <= n )
  {
     cout << licznik << ": ";
     cin >> liczba;
     suma += liczba;
     ++licznik;
  }
  if( n > 0 )
  {
    cout << "\nSuma liczb: " << suma;
    cout << "\nSrednia : " << suma / n;
  }
  else
    cout << "\nNie wprowadzono danych";
  . . .
}
```

Instrukcja alternatywy chroni przed dzielenie przez zero

Obrona przed nieprawidłowymi danymi

```
int main()
{
  . . .
  cout << "\nObliczam sume i srednia ciagu N liczb\n";
  cout << "\nWprowadz ile ma byc liczb: ";
  cin >> n;

  li
  wh
  {



  }
  if
  {
    cout << "\nSuma liczb: " << suma;
    cout << "\nSrednia : " << suma / n;
  }
  else
    cout << "\nNie wprowadzono danych";
  . . .
}
```

Program broni się przed błędem, będącym wynikiem wprowadzenia nieprawidłowych danych

A może nie pozwolimy na wprowadzenie nieprawidłowych danych?

Obrona przed nieprawidłowymi danymi

```
int main()
{
  . . .
  cout << "\nObliczam sume i srednia ciagu N liczb\n";

  do
  {
    cout << "\nWprowadz ile ma byc liczb: ";
    cin >> n;
    if( n <= 0 )
      cout << "\nWartosc powinna byc dodatnia!";
  }
  while( n <= 0 );

  licznik = 1;
  while( licznik <= n )
  {
     cout << licznik << ": ";
     cin >> liczba;
     suma += liczba;
     ++licznik;
  }
  cout << "\nSuma liczb: " << suma;
  cout << "\nSrednia : " << suma / n;
  . . .
}
```

Ta iteracja nie pozwala na wprowadzenie nieprawidłowej wartości dla *n*.

Nie trzeba zatem bronić się przed dzieleniem przez 0.

Instrukcja iteracyjna while — przykład III

Problem

Rowerzysta notuje dystanse przejechane w ramach każdego wypadu rowerowego. Po zakończeniu sezonu chce obliczyć, ile w sumie przejechał kilometrów oraz jaki był średni dystans wycieczki. Liczba dystansów nie jest z góry ustalona, wprowadzenie zerowej wartości dystansu kończy wczytywanie danych.

Scenariusz działania programu

```
Obliczam sumaryczny i sredni dystans.
Podaj kolejne dystanse, 0 konczy wprowadzanie:
>25
>33
>89
>14
>42
>0
Dystans sumaryczny: 203.00
    Dystans sredni:  40.60
```

Instrukcja iteracyjna while — przykład III

Analiza

- Program powinien wczytać kolejno przejechane dystanse, na bieżąco dodawać je do dystansu sumarycznego.
- Ponieważ nie wiadomo ile będzie dystansów, zakładamy, że wprowadzenie dystansu zerowego jest sygnałem końca wprowadzania danych.
- Po tym następuje wyświetlenie dystansu sumarycznego i średniego.
- Wprowadzenie wartości ujemnej zostanie potraktowane jako mimowolny błąd, znak zostanie zignorowany.

Instrukcja iteracyjna while — przykład III

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

int main()
{
  double dystans, suma;
  int licznik;

  cout << "\nObliczam sumaryczny i sredni dystans.";
  cout << "\nPodaj kolejne dystanse, 0 konczy wprowadzanie:\n";

  suma = 0;
  licznik = 0;

  cout << '>';
  cin >> dystans;
  while( dystans != 0 )
  {
    suma += fabs( dystans );
    ++licznik;

    cout << '>';
    cin >> dystans;
  }
```

Wyzerowanie zmiennej suma i licznika — to jest bardzo ważne!

1/2

Instrukcja iteracyjna while — przykład III

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

int main()
{
  double dystans, suma;
  int licznik;

  cout << "\nObliczam sumaryczny i sredni dystans.";
  cout << "\nPodaj kolejne dystanse, 0 konczy wprowadzanie:\n";

  suma = 0;
  licznik = 0;

  cout << '>';
  cin >> dystans;
  while( dystans != 0 )
  {
    suma += fabs( dystans );
    ++licznik;

    cout << '>';
    cin >> dystans;
  }
```

Wczytanie pierwszego dystansu

1/2

Instrukcja iteracyjna while — przykład III

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

int main()
{
  double dystans, suma;
  int licznik;

  cout << "\nObliczam sumaryczny i sredni dystans.";
  cout << "\nPodaj kolejne dystanse, 0 konczy wprowadzanie:\n";

  suma = 0;
  licznik = 0;

  cout << '>';
  cin >> dystans;
  while( dystans != 0 )
  {
    suma += fabs( dystans );
    ++licznik;

    cout << '>';
    cin >> dystans;
  }
```

Sprawdzenie czy czasem nie jest zerowy

1/2

Instrukcja iteracyjna while — przykład III

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

int main()
{
  double dystans, suma;
  int licznik;

  cout << "\nObliczam sumaryczny i sredni dystans.";
  cout << "\nPodaj kolejne dystanse, 0 konczy wprowadzanie:\n";

  suma = 0;
  licznik = 0;

  cout << '>';
  cin >> dystans;
  while( dystans != 0 )
  {
    suma += fabs( dystans );
    ++licznik;

    cout << '>';
    cin >> dystans;
  }
```

Dodanie dystansu do sumy, zwiększenie licznika dystansów

1/2

Instrukcja iteracyjna while — przykład III

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

int main()
{
  double dystans, suma;
  int licznik;

  cout << "\nObliczam sumaryczny i sredni dystans.";
  cout << "\nPodaj kolejne dystanse, 0 konczy wprowadzanie:\n";

  suma = 0;
  licznik = 0;

  cout << '>';
  cin >> dystans;
  while( dystans != 0 )
  {
    suma += fabs( dystans );
    ++licznik;

    cout << '>';
    cin >> dystans;
  }
```

Wczytanie kolejnego dystansu

Instrukcja iteracyjna while — przykład III

```
  if( licznik > 0  )
  {
    cout << "\nSuma: " << suma;
    cout << "\nSrednia: " << suma / licznik;
  }
  else
    cout << "\nNie mam nic do roboty";

  cout << endl << "Nacisnij Enter by zakonczyc..." << endl;
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

2/2

Wyprowadzenie wyników gdy wczytano przynajmniej jeden dystans

Przykład III a iteracja do-while

Kontrola wprowadzonego dystansu

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

int main()
{
  double dystans, suma;
  int licznik;

  cout << "\nObliczam sumaryczny i sredni dystans.";
  cout << "\nPodaj kolejne dystanse, 0 konczy wprowadzanie:\n";

  suma = 0;
  licznik = 0;

  do
  {
    cout << '>';
    cin >> dystans;
    if( dystans != 0 )
    {
      suma += fabs( dystans );
      ++licznik;
    }
  }
  while( dystans != 0 );
```

1/2

Przykład III a iteracja do-while

```
  if( licznik > 0  )
  {
    cout << "\nSuma: " << suma;
    cout << "\nSrednia: " << suma / licznik;
  }
  else
    cout << "\nNie mam nic do roboty";

  cout << endl << "Nacisnij Enter by zakonczyc..." << endl;
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

2/2

Wyprowadzenie wyników gdy wczytano przynajmniej jeden dystans

Przykład III — Iteracja while a iteracja do-while

```
suma = 0;
licznik = 0;

cout << '>';
cin >> dystans;
while( dystans != 0 )
{
  suma += fabs( dystans );
  ++licznik;

  cout << '>';
  cin >> dystans;
}
```

```
suma = 0;
licznik = 0;

do
{
  cout << '>';
  cin >> dystans;
  if( dystans != 0 )
  {
    suma += fabs( dystans );
    ++licznik;
  }
}
while( dystans != 0 );
```

Od instrukcji while do instrukcji for

```
int licznik;
licznik = 10;
while( licznik > 0 )
{
  cout << endl << licznik << "..." ;
  --licznik;
}
```

Inicjalizacja

Warunek

Ciało iteracji

Modyfikacja

- Iteracja *for* w C/C++ nie ma nic wspólnego — poza nazwą — z instrukcją iteracyjną *for* z języka Pascal.
- Instrukcja *for* w języku C/C++ stanowi uogólnienie schematu iteracji *while*.

```
int licznik;
for( licznik = 10 ; licznik > 0 ; --licznik )
  cout << endl << licznik << "..."
```

Inicjalizacja

Warunek

Modyfikacja

Ciało iteracji

Instrukcja iteracyjna for — ogólny schemat

```
inicjalizacja;
while( warunek )
{
  ciało_iteracji
  modyfikacja
}
```

```
for( inicjalizacja; warunek; modyfikacja )
  ciało_iteracji
```

Przykład wykorzystania iteracji for

Problem 1

- Napisać program wyznaczający średni, dobowy kurs waluty EURO na podstawie kursów notowanych na początku każdej godziny.
- Pod koniec doby analityk wprowadza zanotowane liczby — program ma wyznaczyć na tej podstawie średnie kurs dobowy.
- Liczba wprowadzanych kursów jest znana, jest to zawsze 24.

Iteracyjne przetwarzanie ciągów liczbowych

W części inicjalizacyjnej może być więcej instrukcji rozdzielonych przecinkami.

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

int main()
{
  const int MAKS_LB_KURSOW = 24;
  float kurs, sumaryczny;
  int lb_kursow;

  cout << "\nWyznaczam dobowy sredn kurs waluty EURO.";
  cout << "\nWprowadz 24 dodatnie liczby -- kursy EURO";
  cout << "\nzanotowane na poczatku kazdej godziny.\n" << flush;

  for( sumaryczny = 0, lb_kursow = 1 ; lb_kursow <= MAKS_LB_KURSOW; lb_kursow++ )
  {
    cout << '>';
    cin >> kurs;

    sumaryczny += kurs;
  }//for

  cout << "\nKurs sredni: " << sumaryczny / MAX_LB_KURSOW;

  cout << endl << "Nacisnij Enter by zakonczyc...";
  cin.ignore(); cin.get();
  return EXIT_SUCCESS;
}
```

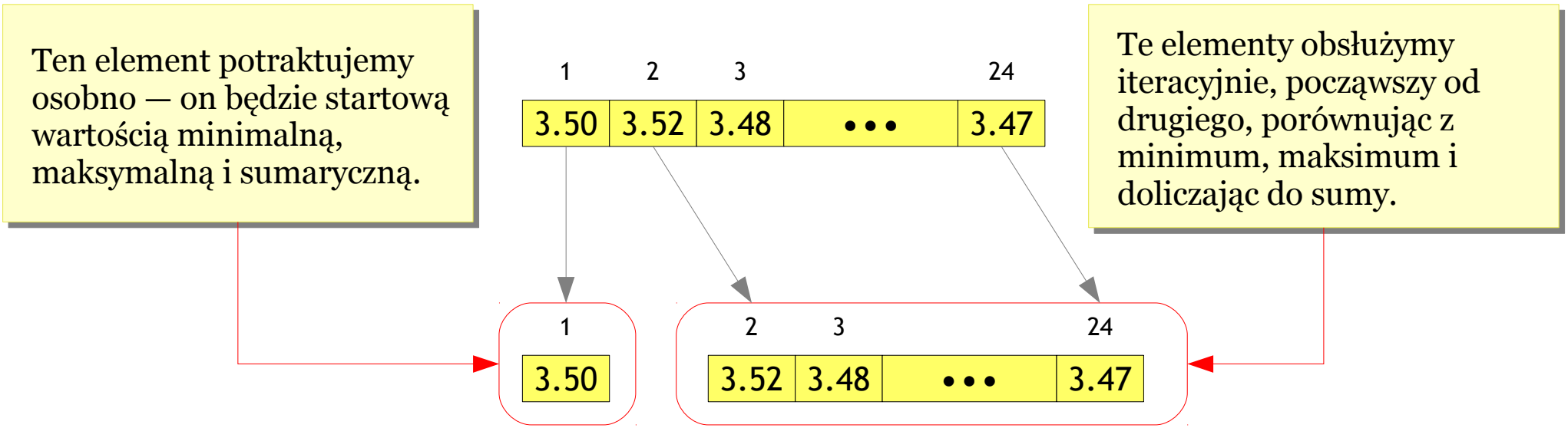
Iteracyjne przetwarzanie ciągów liczbowych

Problem 2

- Uzupełnić poprzedni program o wyznaczanie kursu minimalnego i maksymalnego.

Wyznaczanie minimum i maksimum

- Jeżeli *wczytany kurs* jest *mniejszy od minimalnego*, to niech on się stanie *minimalnym*. Jeżeli *wczytany kurs* jest *większy od maksymalnego*, to niech on się stanie *maksymalnym*. Jak ustawić wartość startową minimum i maksimum?



Iteracyjne przetwarzanie ciągów liczbowych

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

const int MAKS_LB_KURSOW = 24;

int main()
{
  float kurs, sumaryczny, maksymalny, minimalny;
  int lb_kursow;

  cout << "\nWyznaczam dobowy, sredni, minimalny i maksymalny kurs";
  cout << "\nwaluty EURO. Wprowadz 24 dodatnie liczby -- kursy EURO";
  cout << "\nzanotowane na poczatku kazdej godziny.\n>" << flush;

  // Wczytanie pierwszego kursu
  cin >> kurs;

  // Pierwszy i jedyny na razie kurs to kurs minimalny, maksymalny i sumaryczny
  maksymalny = minimalny = sumaryczny = kurs;
```

1/2

Iteracyjne przetwarzanie ciągów liczbowych

```
  for( lb_kursow = 2; lb_kursow <= MAKS_LB_KURSOW; lb_kursow++ )
  {

    cout << '>' << flush;
    cin >> kurs;

    sumaryczny = sumaryczny + kurs;

    if( kurs < minimalny )
        minimalny = kurs;

    if( kurs > maksymalny )
        maksymalny = kurs;
  }

  cout << "\nKurs najwyzszy: " << maksymalny;
  cout << "\nKurs najnizszy: " << minimalny;
  cout << "\nKurs sredni: " << sumaryczny / MAKS_LB_KURSOW;

  cout << endl << "Nacisnij Enter by zakonczyc...";
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

2/2