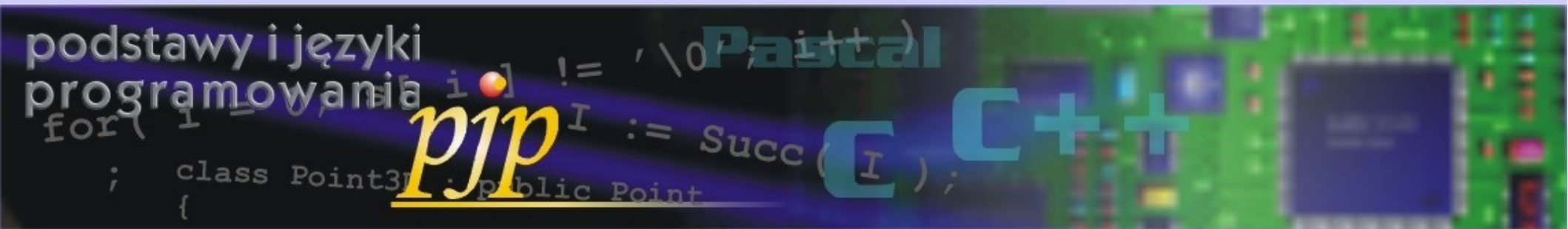
Języki programowania obiektowego
Nieobiektowe elementy języka C++

Roman Simiński
roman.siminski@us.edu.pl
www.programowanie.siminskionline.pl

Instrukcje sterujące wykonaniem programu,
przykłady wykorzystania



Prosty program przykładowy

Zadaniem programu jest obliczanie, ile średnio litrów paliwa zużywa pojazd na trasie 100 km.

```
Obliczam ile Twoj pojazd spala paliwa na 100 km
Dystans: 500
Paliwo: 35
Spalanie 7.00 l na 100 km
Nacisnij Enter by zakonczyc program...
```

$$\frac{\text{przejechany dystans [km]} \;-\; \text{ilość paliwa [litry]}}{100\ \text{[km]} \;-\; x\ \text{[litry]}}$$

$$\text{spalanie} = (100 * \text{ilość paliwa}) \;/\; \text{przejechany dystans}$$

Prosty program przykładowy

```
#include <iostream>
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;

  cout << "Paliwo: " << flush;
  cin >> paliwo;

  cout << "Spalanie " << ( paliwo*100 ) / dystans << " l na 100 km" << endl;

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

  return EXIT_SUCCESS;
}
```

Potencjalne dzielenie przez zero!

Prosty program przykładowy

```
#include <iostream>
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;
  cout << "Paliwo: " << flush;
  cin >> paliwo;

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

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

  return EXIT_SUCCESS;
}
```

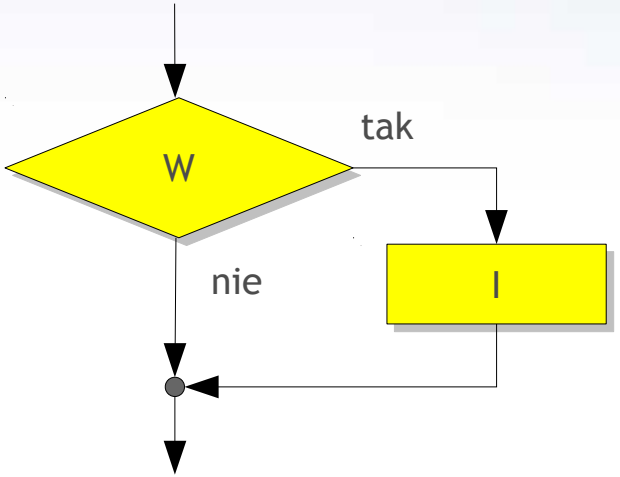
Instrukcja warunkowa

Instrukcja warunkowa

jeżeli (warunek W jest prawdziwy)
  wykonaj instrukcję I

```
if( W ) I
```

```
if( W )
  I
```



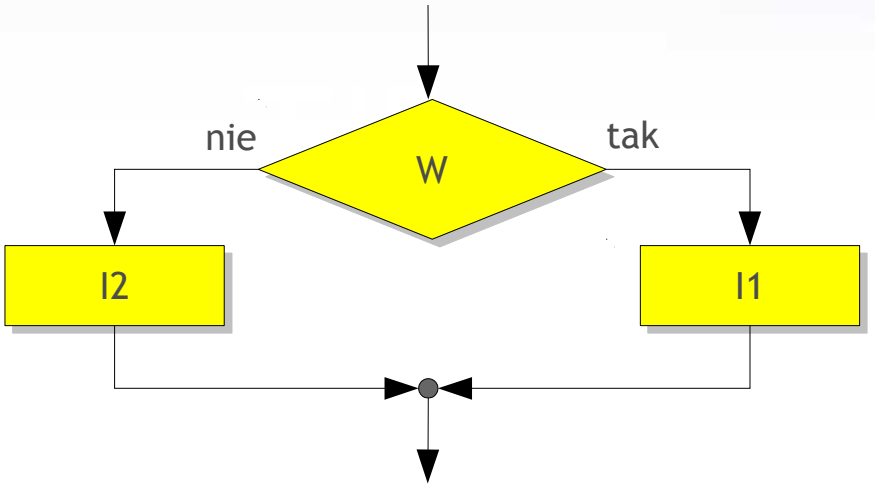
- Instrukcja warunkowa obejmuje swoim zasięgiem jedną, następną instrukcję.
- Jeżeli instrukcja warunkowa ma obejmować więcej niż jedną instrukcję, należy ująć je w {}, tworząc *instrukcję złożoną*.

Instrukcja alternatywy

jeżeli (warunek W jest prawdziwy)
  wykonaj instrukcję I1
w przeciwnym przypadku
  wykonaj instrukcję I2

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

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



- Instrukcja alternatywy obejmuje swoim zasięgiem jedną, następną instrukcję, zarówno w części *if* jak i *else*.
- Jeżeli instrukcja alternatywy ma obejmować więcej niż jedną instrukcję, należy ująć je w {}, tworząc instrukcję złożoną.

Kiedy wyrażenie jest fałszywe a kiedy prawdziwe?

- W C89 nie ma predefiniowanego typu Boolean — używa się liczb całkowitych.
- W języku C++ zdefiniowany jest typ **bool** i wartości **false** i **true**.
- Jednak w C i C++ *każde wyrażenie* dające w wyniku wartość całkowitą *różną od zera* jest traktowane jako *prawdziwe*, a dające w wyniku *wartość zerową* jest traktowane jako *fałszywe*.
- W wyrażeniach *relacyjnych* i *porównania* prawda to 1, fałsz to 0 całkowite (int).

Przydatne operatory

Operator	Znaczenie
==	równe
!=	różne
!	logiczna negacja
&&	logiczny and
||	logiczny or

Instrukcja alternatywy a instrukcje warunkowe

```
float a, b, c, delta;
. . .
delta = b * b – 4 * a * c;
```

Złożenie instrukcji alternatywy:

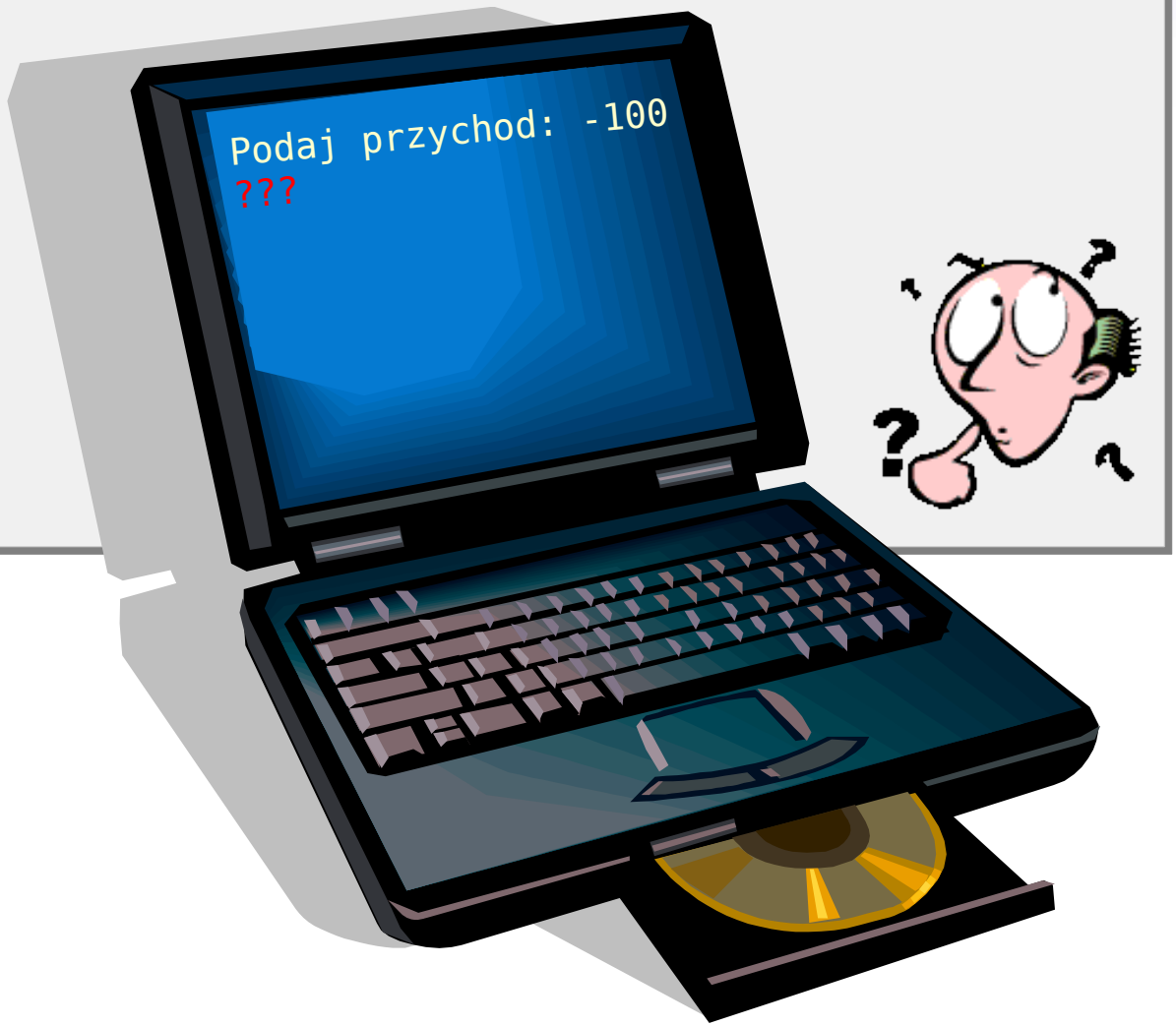
```
if( delta < 0 )
  cout << "Brak pierwiastków rzeczywistych" << endl;
else
  if( delta == 0  )
    cout << "Jeden pierwiastek rzeczywisty" << endl;
  else
    cout << "Dwa pierwiastki rzeczywiste" << endl;
```

Kolejne instrukcje warunkowe:

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

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

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

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";
```

Na marginesie — instrukcja wyrażeniowa

- *Instrukcja wyrażeniowa* — to każde poprawne wyrażenie w języku C++ (również wyrażenie puste) zakończone znakiem średnika.
- *Wykonanie* takiej instrukcji polega na *wyznaczeniu wartości* danego wyrażenia.

```
x = 0;
```

```
a + b;
```

Oczywiście bez większego sensu, lecz legalne

```
x = a + b;
```

```
;
```

Niepozorna, lecz bardzo użyteczna *instrukcja pusta*

- W języku C/C++ średnik *kończy* instrukcję.

Na marginesie — instrukcja grupująca

- *Instrukcja grupująca* (*złożona*) — zwana inaczej *blokiem*, to lista instrukcji ujęta w nawiasy klamrowe {}.
- *Blok* traktowany jest jako *pojedyncza instrukcja*.
- Identyfikator zadeklarowany w obrębie bloku ma jego zakres.
- Bloki mogą być zagnieżdżone do dowolnej głębokości.
- W obrębie *zagnieżdżonych* bloków następuje *przesłanianie nazw*.

```
{
  int i = 0, j = 1, k;
  k = i + j;
  {
    float k = 10.2;
    cout << "k = " << k;
  }
  cout << "k = " << k;
}
```

```
k = 10.2
k = 1
```

Od instrukcji warunkowych do instrukcji switch

```
int main()
{
  int nadwozie;

  cout << "\nJaki typ nadwozia lubisz?";
  cout << "\n1. Sedan\n2. SUV\n3. Coupe";
  cout << "\nWpisz 1, 2 lub 3: ";
  cin >> nadwozie;

  if( nadwozie == 1 )
    cout << "\nChyba lubisz eleganckie limuzyny!";
  if( nadwozie == 2 )
    cout << "\nWidze, ze ciagnie Cie w teren!";
  if( nadwozie == 3 )
    cout << "\nTy to pewnie lubisz szybka jazde!";

  cout << "\n\nEnter=Koniec";
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

Od instrukcji warunkowych do instrukcji switch

```
int main()
{
  int nadwozie;

  cout << "\nJaki typ nadwozia lubisz?";
  cout << "\n1. Sedan\n2. SUV\n3. Coupe";
  cout << "\nWpisz 1, 2 lub 3: ";
  cin >> nadwozie;

  switch( nadwozie )
  {
    case 1 : cout << "\nChyba lubisz eleganckie limuzyny!";
                     break;
    case 2 : cout << "\nWidze, ze ciagnie Cie w teren!";

                     break;
    case 3 : cout << "\nTy to pewnie lubisz szybka jazde!";
                     break;
  }

  cout << "\n\nEnter=Koniec";
  cin.ignore();
  cin.get();
  return EXIT_SUCCESS;
}
```

Instrukcja *switch*: jedna zmienna i porównanie z wartościami znanymi na etapie kompilacji

Instrukcja przełączająca switch

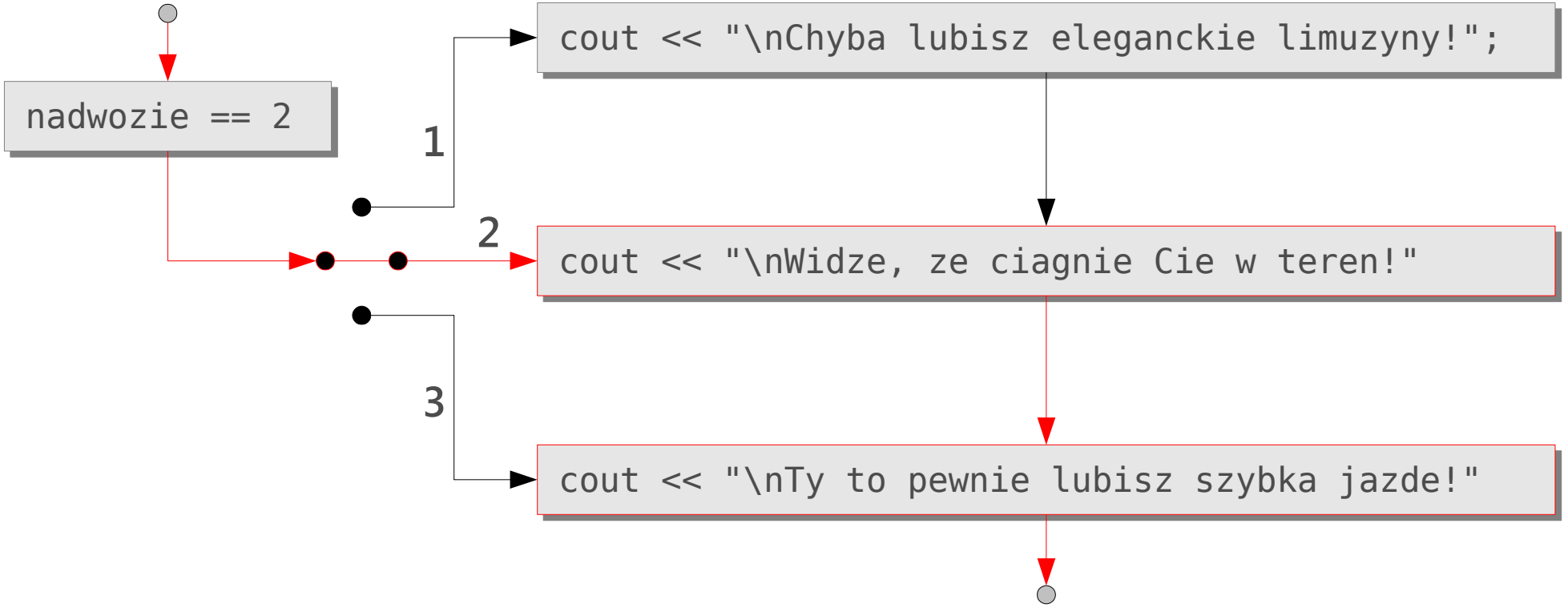
```
switch( wyrażenie )
{
  case wyrażenie_stałe : instrukcje
  case wyrażenie_stałe : instrukcje
  . . .
  default : instrukcje
}
```

- Instrukcja *switch*:
  - *sprawdza* czy wartość wyrażenia jest równa jednemu z kilku *przypadków* określonych wyrażeniem stałym,
  - *wykonuje* skok do instrukcji zapisanych za owym wyrażeniem stałym.
- Jeżeli nie znaleziono przypadku pasującego do wartości wyrażenia, wykonywane są instrukcje zapisane po frazie *default*.
- Instrukcja służy do podejmowania decyzji wielowariantowych.
- Wszystkie wyrażenia są typu *całkowitego*.

Gdyby brakowało instrukcji break

```
. . .
switch( nadwozie )
{
  case 1 : cout << "\nChyba lubisz eleganckie limuzyny!";
  case 2 : cout << "\nWidze, ze ciagnie Cie w teren!";
  case 3 : cout << "\nTy to pewnie lubisz szybka jazde!";
}
. . .
```

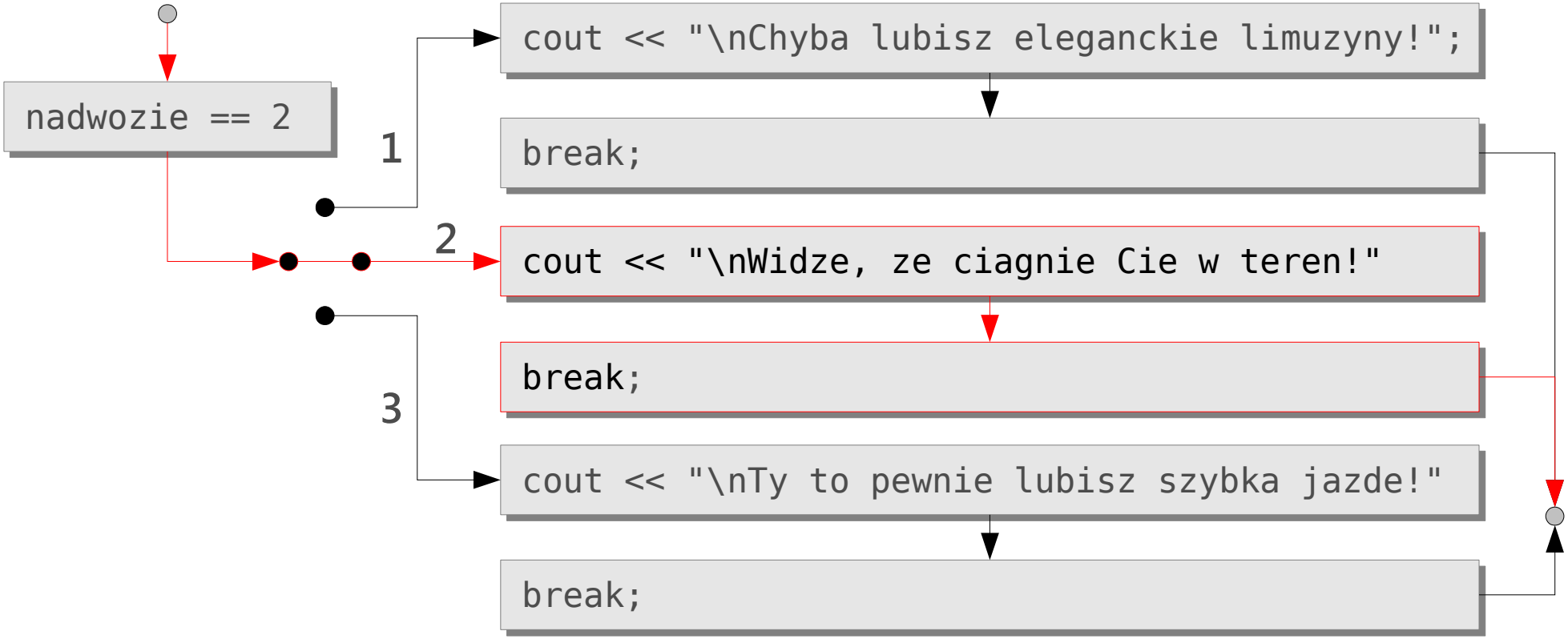
nadwozie: 2



Rola instrukcji break

```
switch( nadwozie )
{
  case 1 : cout << "\nChyba lubisz eleganckie limuzyny!";
           break;
  case 2 : cout << "\nWidze, ze ciagnie Cie w teren!";
           break;
  case 3 : cout << "\nTy to pewnie lubisz szybka jazde!";
           break;
}
```

nadwozie: 2



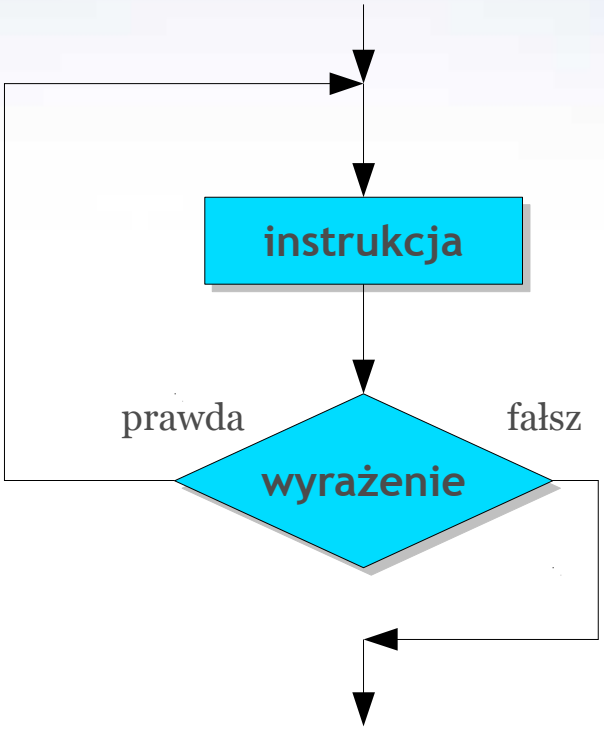
Instrukcja iteracyjna do-while

Gdy iterowana jest jedna instrukcja:

```
do
  instrukcja
while( wyrazenie );
```

Gdy iterowany jest ciąg instrukcji:

```
do
{
  ciag instrukcji
}
while( wyrazenie );
```



- 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 do-while

```
#include <iostream>
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;
  cout << "Paliwo: " << flush;
  cin >> paliwo;

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

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

  return EXIT_SUCCESS;
}
```

W tym miejscu użytkownik może wprowadzić nieprawidłowe dane. Nie pozwólmy mu na to!

Zastosowanie instrukcji do-while — weryfikacja danych

```
do
{
  cout << "Dystans: " << flush;
  cin >> dystans;
}
while( dystans <= 0 );

do
{
  cout << "Paliwo: " << flush;
  cin >> paliwo;
}
while( paliwo <= 0 );
```

Wykonuj wczytywanie dystansu, dopóki jest on nieprawidłowy.

Wykonuj wczytywanie il. paliwa, dopóki jest ona nieprawidłowa.

Ta wersja nie jest zbyt dobra, bo użytkownik nie jest informowany o wprowadzeniu niepoprawnej wartości.

Zastosowanie instrukcji do-while — weryfikacja danych

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

  if( dystans <= 0 )
    cout << "Dystans musi byc wiekszy od zera!" << endl;

}
while( dystans <= 0 );

do
{
  cout << "Paliwo: " << flush;
  cin >> paliwo;

  if( paliwo <= 0 )
    cout << "Ilosc paliwa musi byc wieksza od zera!" << endl;

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

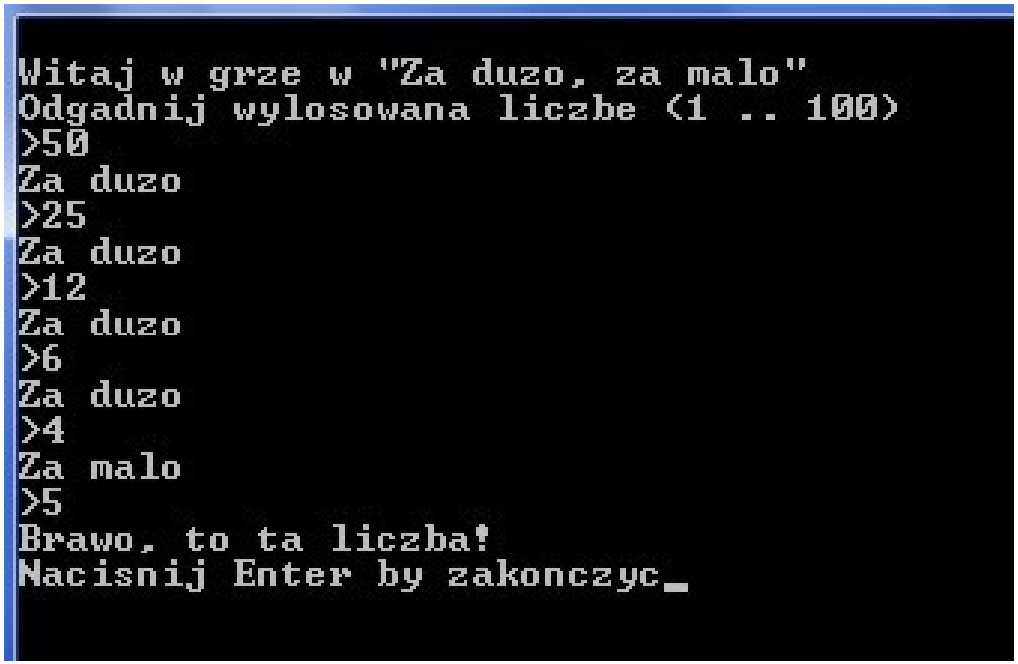
Wykonuj wczytywanie dystansu, dopóki jest on nieprawidłowy, poinformuj o tym użytkownika.

Wykonuj wczytywanie il. paliwa, dopóki jest ona nieprawidłowa, poinformuj o tym użytkownika.

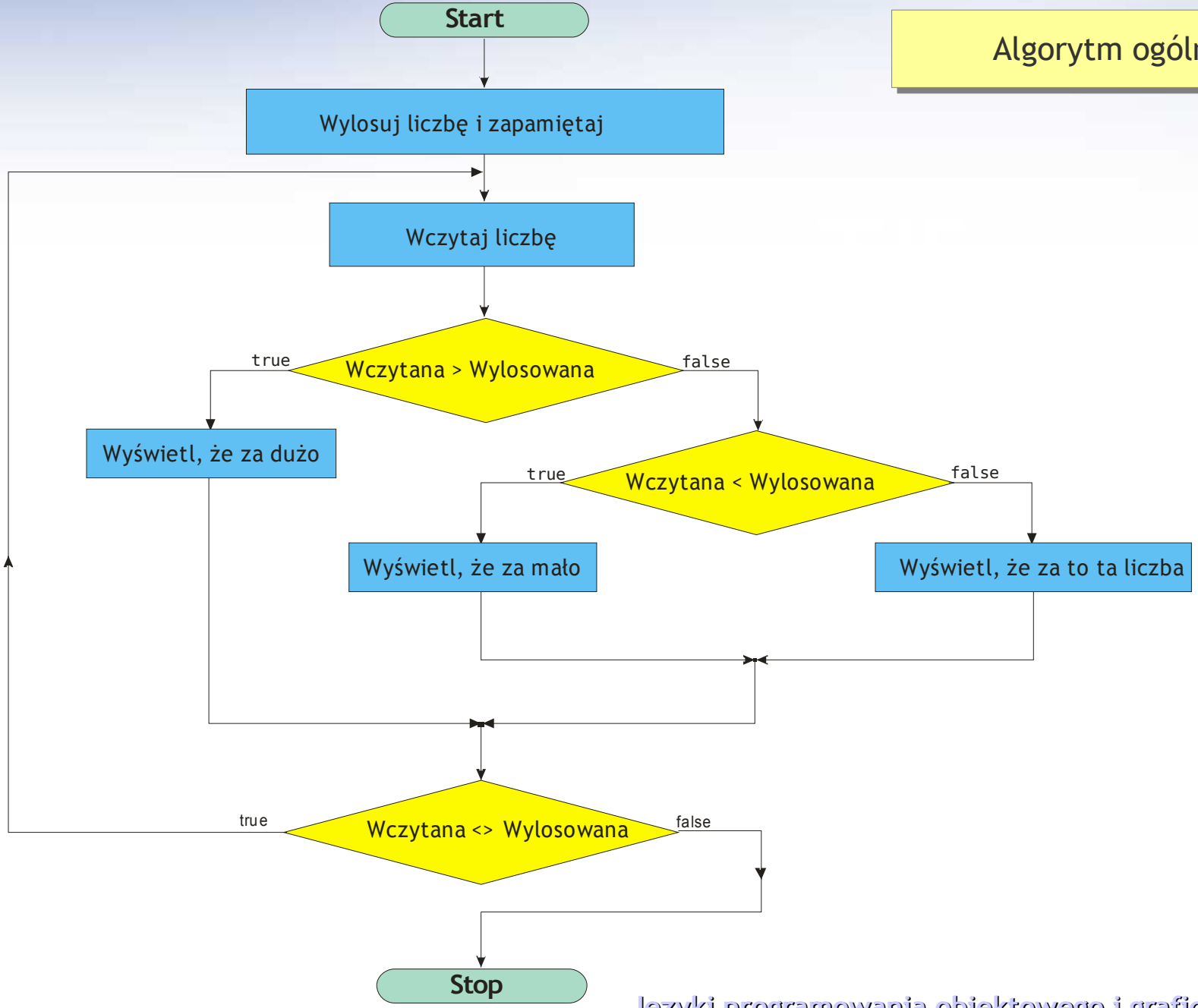
Ta wersja jest wyraźnie lepsza.

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

Program losuje liczbę z przedziału od 1 do 100. Zadaniem użytkownika jest odgadnięcie wylosowanej liczby.



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



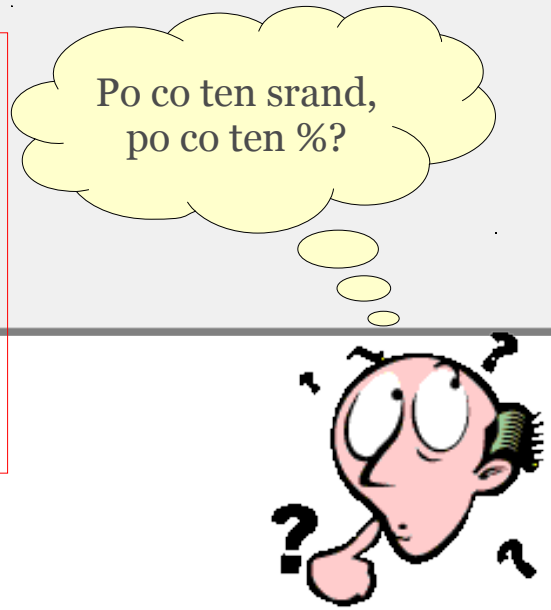
Zastosowanie instrukcji — gra w „za dużo, za mało”

```
#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 funkcji *time*

Losowanie zalążka generatora liczb pseudolosowych zależnego ob aktualnej wartości timera

Losowanie liczby pseudolosowej, operator % to **modulo**



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

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

```
srand( 1000 );
```

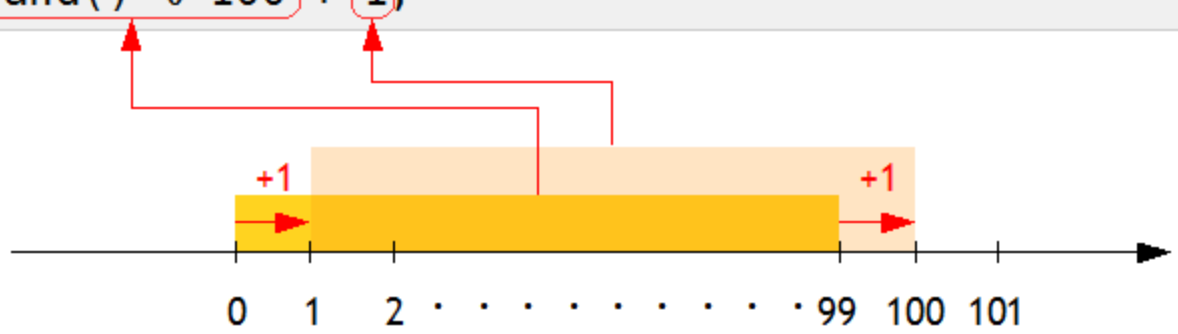
Tak zainicjowany generator będzie działał powtarzalnie. Należy uzmiennić zalążek, przykładowo uzależniając go od bieżącego czasu:

```
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;
```



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

```
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!" << endl << "Nacisnij Enter by zakonczyc";
  }
  while( wylosowana != wczytana );

  cin.ignore();
  cin.get();
  return EXIT_SUCCESS;
}
```

Wczytaj liczbę typowaną przez gracza

Porównaj wczytaną liczbę z wylosowaną
i wyprowadź odpowiedni komunikat

Wykonuj, dopóki liczba nie została odgadnięta

Instrukcja switch + 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 switch + 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";
                break;
     case '2' : cout << "\nWybrales szybki format\n";
                break;
     case '3' : cout << "\nWybrales diagnostyke\n";
                break;
  }
}
while( klawisz != '4' );

cout << "\n\nWybrales Koniec, nacisnij Enter by potwierdzic...";
cin.ignore();
cin.get();
. . .
```

Typ zmiennej sterującej a typ wartości przypadku

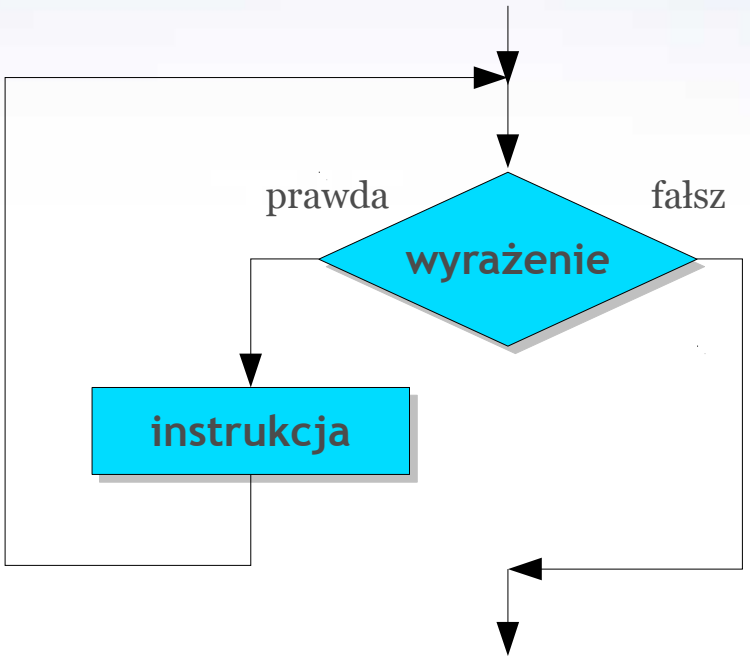
```
char klawisz;
. . .
cin >> klawisz;
switch( klawisz )
{
   case '1' : cout << "\nWybrales formatowanie\n";
              break;
   case '2' : cout << "\nWybrales szybki format\n";
              break;
   case '3' : cout << "\nWybrales diagnostyke\n";
              break;
}
```

```
int nadwozie;
. . .
cin >> nadwozie;
switch( nadwozie )
{
  case 1 : cout << "\nChyba lubisz eleganckie limuzyny!";
           break;
  case 2 : cout << "\nWidze, ze ciagnie Cie w teren!";
           break;
  case 3 : cout << "\nTy to pewnie lubisz szybka jazde!";
           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 — sylwestrowe odliczanie ;-)

```
int licznik = 10;

while( licznik > 0 )
{
  cout << endl << licznik << "...";
  licznik--;
}

cout << endl << "Nowy Rok!!!" << endl;
```

`licznik--`

odpowiada

`licznik = licznik - 1`

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

Odpowiednik wykorzystujący iterację **do-while**:

```
int licznik = 10;

do
{
  cout << endl << licznik << "...";
  licznik--;
}
while( licznik > 0 )

cout << endl << "Nowy Rok!!!" << endl;
```

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 stanowi uogólnienie schematu iteracji *while*.

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

Inicjalizacja

Warunek

Modyfikacja

Ciało iteracji

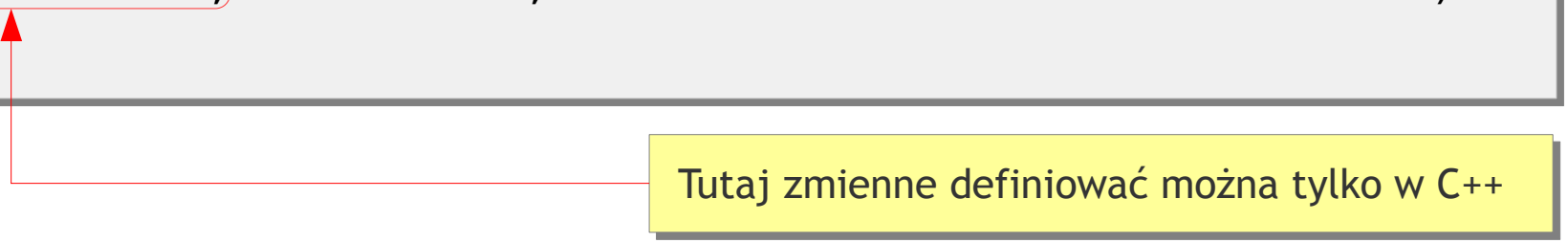
Instrukcja iteracyjna for — ogólny schemat

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

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

- Uwaga — poszczególne sekcje iteracji for mogą być dowolnymi legalnymi, być może złożonymi, wyrażeniami w języku C++.
- W części inicjalizacyjnej w C++ wolno deklarować zmienne, w C nie wolno.

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



Instrukcje break i continue

- Instrukcja *break* pozwala na natychmiastowy „wyskok” z najbardziej zagnieżdżonej, dowolnej *instrukcji iteracyjnej* lub *instrukcji switch*.
- Instrukcja *continue* pozwala na przerwanie bieżącego przebiegu *dowolnej iteracji* i przejście do wykonania *następnego jej przebiegu od jego początku*.

Dla iteracji **while** i **do-while** instrukcja **continue** powoduje przeniesienie sterowania do fazy testowania warunku kontynuacji. W przypadku iteracji for sterowanie przenoszone jest do wyrażenia modyfikującego a potem do testowania warunku.

```
switch( klawisz )
{
  case '1' : full_disk_format();
             break;
  case '2' : quick_disk_format();
             break;
  case '3' : check_disk();
             break;
}
```

```
cin >> klawisz;

while( klawisz != 'k' )
{
  if( klawisz == 'q' )
    break;
  else
    . . .
}
```

Przykład zastosowania instrukcji break i continue w iteracji

```
. . .
for( ; ; )  // Iteracja nieskończona czyli pętla
{
  cout << "\nFormatowanie dysku, wybierz opcje:\n1. Format";
  cout << "\n2. Szybki format\n3. Diagnostyka\n4. Koniec\n>" << flush;

  cin >> klawisz;

  if( klawisz == '4' )    // Czy wybrano klawisz końca?
    break; // Tak, wychodzimy z iteracji

  if( !( klawisz >= '0'&& klawisz <= '9') ) // Czy klawisz jest cyfrą?
     continue; // Klawisz nie jest cyfrą, nie ma co dalej sprawdzać

  switch( klawisz )
  {
     case '1' : full_disk_format();
                break;
     case '2' : quick_disk_format();
                break;
     case '3' : check_disk();
                break;
  }// switch
}
. . .
```

Suplement — wykorzystanie instrukcji, przykład 1

Opis problemu:

- 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.

Suplement — wykorzystanie instrukcji, przykład 1

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

const int MAKS_LB_KURSOW = 24;

int main()
{
  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 << '>' << flush;
    cin >> kurs;

    sumaryczny = sumaryczny + kurs;
  }//for

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

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

Umieszczenie **const** przed zmienna sprawia, że jej wartość w trakcie wykonania programu nie może się zmieniać (interpretuj jak **readonly**).

Java — istnieje specyfikacja **final** pozwalające na uzyskanie podobnego efektu, to nie to samo co **const** z języka C++. W języku C# występuje słowo **const** interpretowane podobnie jak w języku C++.

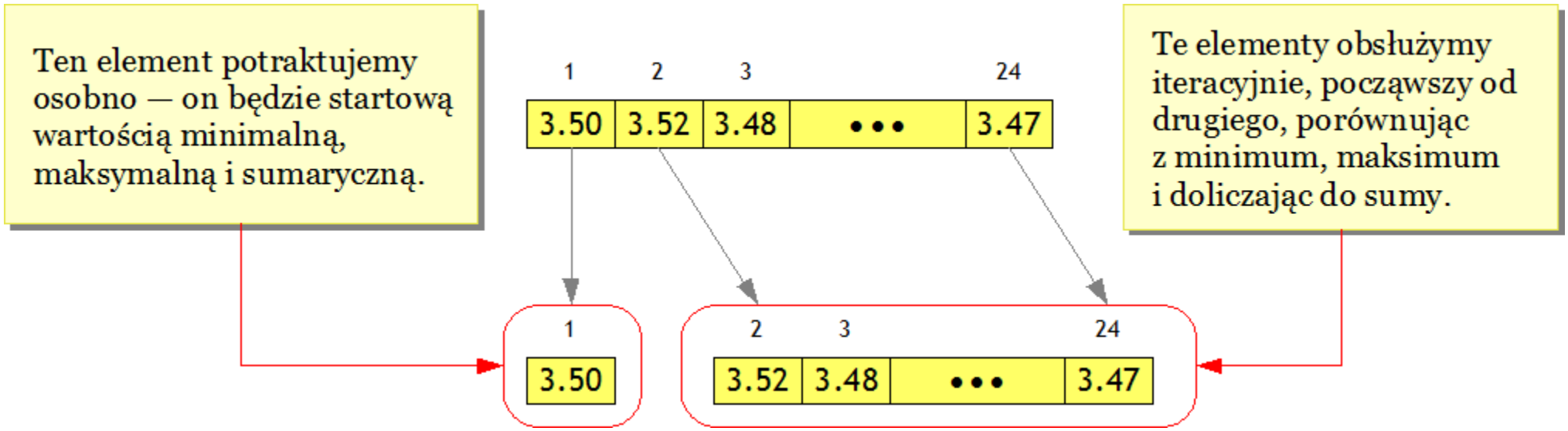
Suplement — wykorzystanie instrukcji, przykład 2

Opis problemu:

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?



Suplement — wykorzystanie instrukcji, przykład 2

```
#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

Suplement — wykorzystanie instrukcji, przykład 2

```
  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;
  }//for

  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

Suplement — wykorzystanie instrukcji, przykład 3

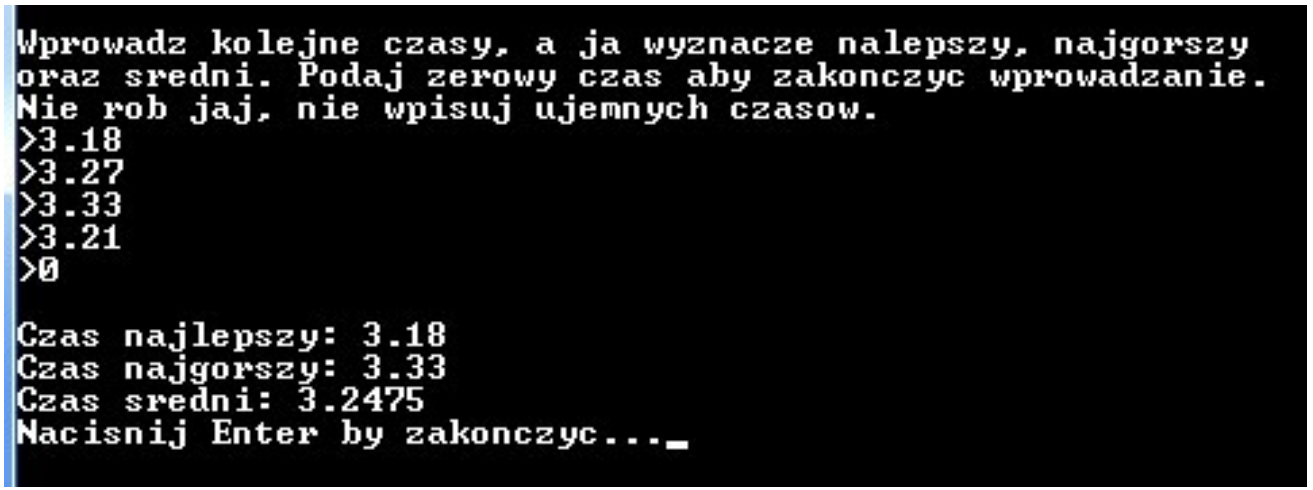
Opis problemu:

- Napisać program przyśpieszający ocenę wyników sprintera. Sprinter wielokrotnie pokonuje jednakowy dystans, notując kolejno osiągane czasy.
- Po treningu zawodnik siada do komputera, wpisuje kolejne czasy i oczekuje, że program wyznaczy mu czas najlepszy, najgorszy oraz średni.
- Liczba wprowadzanych czasów jest bliżej nieznana. Może ich być np. kilka, kilkanaście, kilkadziesiąt.

Suplement — wykorzystanie instrukcji, przykład 3

Scenariusz działania programu:

- Program wyświetla informację o jego przeznaczeniu.
- Program wczytuje kolejno czasy, przy czym ich liczba nie jest z góry ograniczona ani wcześniej znana.
- Wprowadzenie czasów kończy wpisanie wartości 0, po zakończeniu wprowadzania czasów program wyznacza i wyprowadza czas najlepszy, najgorszy i średni.
- Program kończy swoje działanie po naciśnięciu przez użytkownika klawisza Enter.



Suplement — wykorzystanie instrukcji, przykład 3

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

int main()
{
  float czas, sumaryczny, najlepszy, najgorszy;
  int licznik;

  cout << "\nWprowadz kolejne czasy, a ja wyznacze nalepszy, najgorszy";
  cout << "\noraz sredni. Podaj zerowy czas aby zakonczyc wprowadzanie.";
  cout << "\nNie rob jaj, nie wpisuj ujemnych czasow.\n>" << flush;

  cin >> czas;
```

1/3

Suplement — wykorzystanie instrukcji, przykład 3

```
if( czas == 0 )
  cout << "Nic do roboty";
else
{
  sumaryczny = najgorszy = najlepszy = czas;
  licznik = 1;

  cout << ">" << flush;
  cin >> czas;

  while( czas != 0 )
  {
    sumaryczny = sumaryczny + czas; // lepiej: sumaryczny += czas;
    if( czas > najgorszy )
      najgorszy = czas;
    if( czas < najlepszy )
      najlepszy = czas;
    licznik = licznik + 1; // lepiej: licznik++;

    cout << ">" << flush;
    cin >> czas;
  }//while

  cout << "\nCzas najlepszy: " << najlepszy;
  cout << "\nCzas najgorszy: " << najgorszy;
  cout << "\nCzas sredni: " << sumaryczny / licznik;
}//if-else
```

2/3

Suplement — wykorzystanie instrukcji, przykład 3

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

  return EXIT_SUCCESS;
}
```

2/3

Dziękuję za uwagę

Pytania? Polemiki?
Teraz, albo:
roman.siminski@us.edu.pl