AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

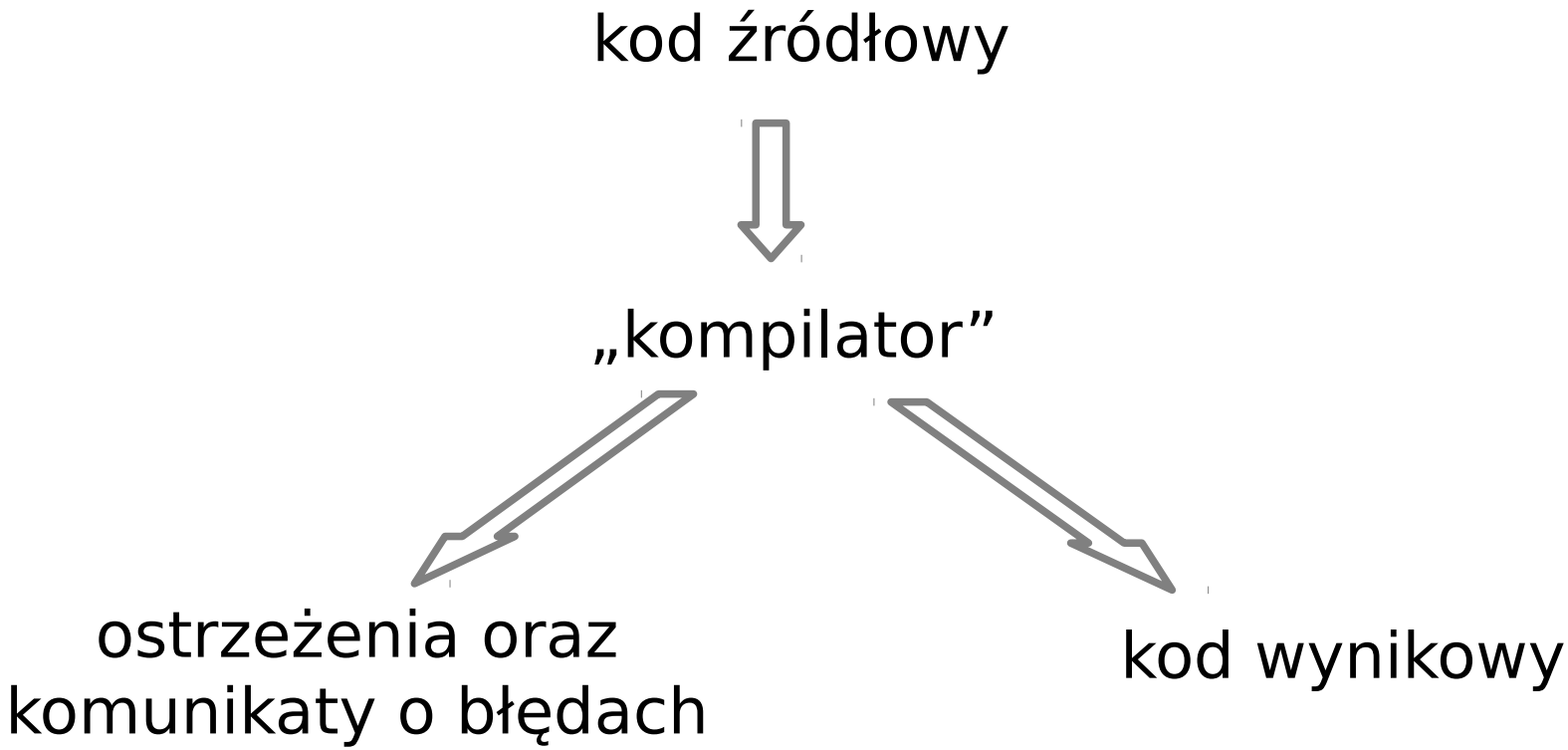
Programowanie w C/C++

Inżynieria Oprogramowania

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej
Katedra Informatyki

Wojciech Szmuc

Podstawowe pojęcia



Zagadnienia podstawowe

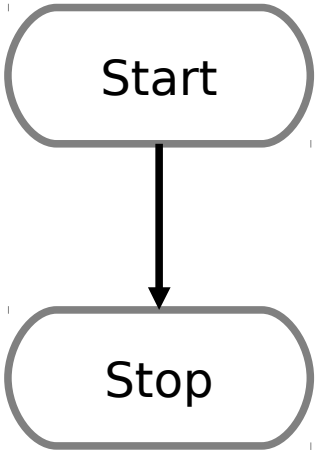
- Program musi zawierać dokładnie jedną funkcję główną (`main`)
- Komentarze (`/* */`, `//`)są ignorowane przez kompilator
- Blok instrukcji ograniczony znakiem { oraz } zawiera kod wykonywany w ramach jednej funkcjonalności
- Na końcu polecenia należy umieścić znak ;
- Elementy wyrażeń można oddzielać znakiem ,

Najprostszy program

kod źródłowy

```
/*najprostszy program*/

int main() //glowna funkcja programu
{
}
```

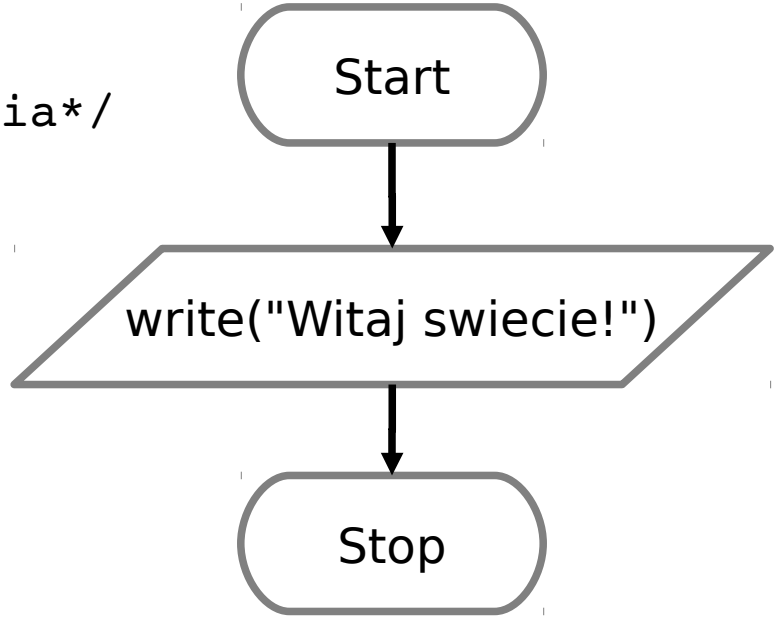
algorytm



Witaj świecie w C

- Aby móc odwoływać się do konsoli należy dołączyć bibliotekę stdio

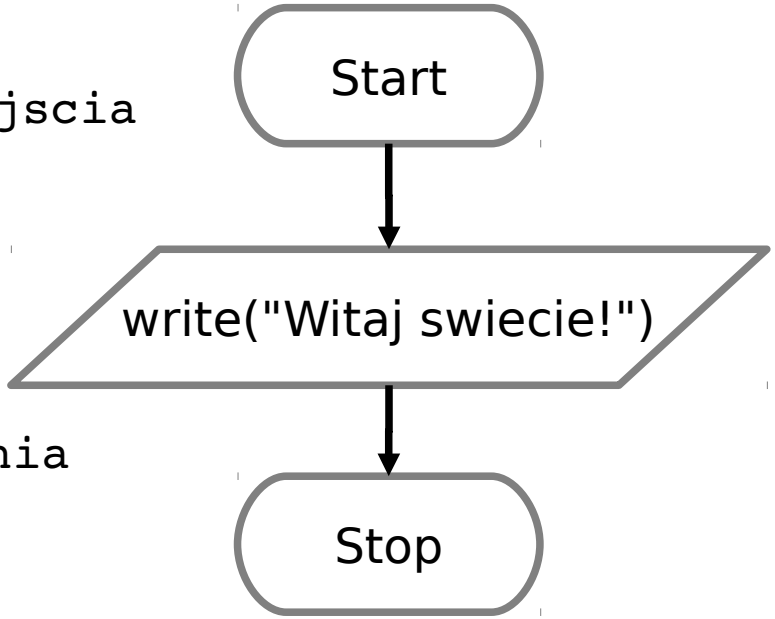
```
#include <stdio.h> /*biblioteka
                     wejscia/wyjscia*/
int main()
{
 printf("Witaj swiecie!");
 /* wypisanie tekstu */

 printf("\n"); /* nowa linia */
 return 0;
}
```



Witaj świecie w C++

- Aby móc odwoływać się do konsoli należy dołączyć bibliotekę iostream

```
#include <iostream> //biblioteka
                    //wejscia/wyjscia
int main()
{
 std::cout<<"Witaj swiecie!";
//wypisanie tekstu

 std::cout<<std::endl; //nowa linia
 return 0;
}
```



Przestrzeń nazwy

- Określa zbiór definicji
- Nazwy definicji nie mogą się powtarzać wewnątrz przestrzeni nazw
- Nazwy definicji mogą się powtarzać dla różnych przestrzeni nazw

```
#include <iostream>

using namespace std; //wskazanie domyslnej przestrzeni
                                //nazwy
int main()
{
 cout<<"Witaj swiecie!"<<endl;
 return 0;
}
```

Typy danych

- Określają jaki rodzaj informacji jest przechowywany w zmiennej
- Wyznaczają zakres oraz dokładność reprezentacji
- Typy wbudowane:
  bool – logiczny – true/false
  char – znakowy – pojedynczy znak graficzny lub sterujący
  int – całkowity – liczby całkowite
  float, double – zmiennoprzecinkowy – liczby rzeczywiste
  void – pusty – brak wartości

Definiowanie zmiennych

- Pierwszym elementem definicji jest nazwa typu
- Jako kolejny element może wystąpić identyfikator zmiennej
- Identyfikator jest ciągiem mogącym zawierać litery, cyfry oraz znaki podkreślenia
- Identyfikator nie może zaczynać się od cyfry oraz być słowem kluczowym
- Litery małe i wielkie nie są równoważne

Przykładowe definicje zmiennych

```
bool decyzja; //definicja zmiennej logicznej

char znak; //definicja zmiennej znakowej

int nrDnia; //definicja zmiennej calkowitoliczbowej

float pi; //definicja zmiennej rzeczywistej

//przypisanie przykładowych wartosci

decyzja=true;
znak='a';
nrDnia=13;
pi=3.14;
```

Cechy poszczególnych typów danych

- Zakresy wartości typów liczbowych:

short int: -32768 ÷ 32767
int: -2147483648 ÷ 2147483647
long long int: -9223372036854775808 ÷ 9223372036854775807
float: 3.4e +/- 38
double: 1.7e +/- 308

- Typy całkowite po dodaniu modyfikatora `unsigned` mogą przechowywać jedynie wartości nieujemne

Reprezentacja danych

- Typ całkowity zapisywany jest w postaci bezpośredniej liczby binarnej
  00000000 00000001

  Liczba ujemna jest negacją liczby całkowitej zmniejszonej o 1
  11111111 11111111

- Typ zmiennoprzecinkowy jest zapisywany w postaci notacji wykładniczej: bit znaku, bity wykładnika (8), bity mantysy (23)
  0 01111111 00000000000000000000000

Właściwości zmiennych

- Zmienne globalne (zdefiniowane poza jakimkolwiek blokiem instrukcji) są inicjalizowane wartością 0
- Wartość początkowa zmiennych lokalnych wynika z zawartości pamięci przydzielonej tej zmiennej
- Zmiennej można nadać wartość w miejscu, gdzie jest definiowana
- Stałe definiuje się przy pomocy modyfikatora `const`
- Wartości zmiennych mogą być konwertowane na inny typ przy pomocy operatora rzutowania

Przykłady właściwości zmiennych

```
#include <iostream>

using namespace std;

int licznik; //zmienna globalna

int main()
{
 int l1; //zmienna lokalna
 double ulamek=0.3; //inicjalizacja z definicja
 const float pi=3.14; //stala

 cout<<"licznik="<<licznik<<endl; //wypisanie wartosci
 cout<<"l1="<<l1<<endl;
 cout<<"ulamek="<<ulamek<<endl;

 return 0;
}
```

Wypisywanie wartości w C

- Aby wypisać wartości należy umieścić w przekazywanym funkcji `printf` łańcuchu znaków informacje o sposobie interpretacji
- Wartości dopisuje się po przecinku za łańcuchem formatującym (pierwszy argument)
- Informacje o formacie umieszcza się po znaku `%`
- Przykładowe oznaczenia formatu:

  d – liczba całkowita

  f – liczba zmiennoprzecinkowa

  c – znak

Przykład wypisywania wartości w C

```
#include <stdio.h>

int licznik;

int main()
{
 int l1;
 double ulamek=0.3;
 const float pi=3.14;

 printf("licznik=%d\n", licznik);
 printf("l1=%d\n", l1);
 printf("ulamek=%f\n", ulamek);

 return 0;
}
```

Definiowane typy danych

- Typ wyliczeniowy umożliwia zdefiniowanie jego elementów przez ich wymienienie
- Poszczególnym elementom typu wyliczeniowego przypisywane są wartości liczbowe
- Możliwe jest utworzenie synonimu dla dowolnego typu przy pomocy instrukcji `typedef`:

```
typedef int ilosc;
```

Przykłady typu wyliczeniowego

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

enum dniTygodnia
{
 poniedzialek=1,
 wtorek,
 sroda,
 czwartek,
 piatek,
 sobota,
 niedziela
};
int main()
{
 enum dniTygodnia dzien=niedziela;
 cout<<"Numer dnia tygodnia "<<dzien<<endl;
 return 0;
}
```

Operatory

- Arytmetyczne: +, -, /, *, %
- (pre/post)(in/de)krementacji: ++, --
- Relacyjne: <, <=, >, >=, !=, ==
- Bitowe: ~ (not), | (or), & (and), ^ (exor), << (przesunięcie w lewo), >> (przesunięcie w prawo)
- Przypisania (z prawej strony na lewą): =, +=, -=, /=, *=, %=, |=, &=, ^=, <<=, >>=
- Porządkujący: ()

Przykłady wykorzystania operatorów

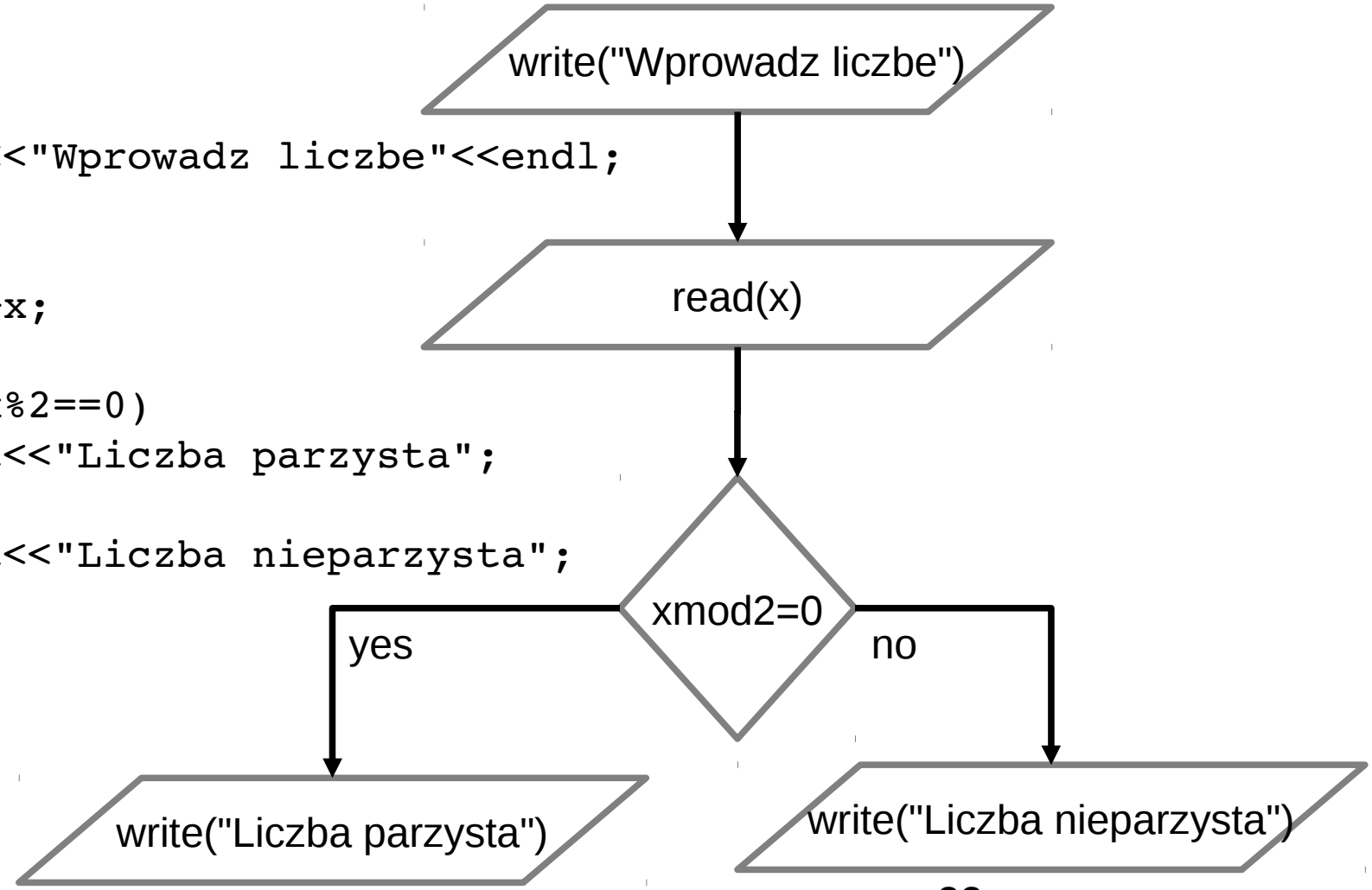
```
int a=1, b=2, c, d;

cout<<"Wprowadz wartosc d"<<endl;
cin>>d; //pobieranie danych; C: scanf("%d", &d);
c=a+b*d;
cout<<"c="<<c<<endl;

float f;
f=a/(float)b; //rzutowanie
cout<<"f="<<f<<endl;

c=a++;
d=++b;
cout<<"a="<<a<<", c="<<c<<endl;
cout<<"b="<<b<<", d="<<d<<endl;

c+=a;
cout<<"c="<<c<<endl;
```

Instrukcja warunkowa

- Pozwala na wykonywanie fragmentów programu w zależności od spełnienia określonych warunków
- Ma możliwe postacie:
  if (wyrażenie) instrukcja;
  if (wyrażenie) instrukcja1; else instrukcja2;
- Jeżeli ma być wykonana więcej niż jedna instrukcja należy je umieścić w bloku
- Wyrażenie może się składać z wielu warunków powiązanych operatorami logicznymi: ! (nie), || (lub), && (i)
- Wyrażenie jest prawdziwe, gdy ma wartość różną od 0

Przykład instrukcji warunkowej

```
cout<<"Wprowadz liczbe"<<endl;


cin>>x;

if (x%2==0)
 cout<<"Liczba parzysta";
else
 cout<<"Liczba nieparzysta";
```



Przykłady użycia instrukcji warunkowej

```
if (x>-10 && x<10)
 cout<<"Liczba jednocyfrowa"<<endl;

if (x) cout<<"Liczba rozna od 0"<<endl;

if (x>0)
 cout<<"Liczba dodatnia"<<endl;
else
 if (x<0)
  cout<<"Liczba ujemna"<<endl;

if (x%6==0)
{
 int y=x/6;
 cout<<y<<endl;
}
else
 cout<<"Nie mozna podzielic x przez 6"<<endl;
```

Operator wyrażenia warunkowego

Składnia:

wyrażenie ? wartość1 : wartość2;

```
/*wyznaczanie wartosci malsymalnej przy pomocy
instrukcji warunkowej*/

if (x>y) max=x;
else max=y;


/*wyznaczanie wartosci malsymalnej przy pomocy
operatora wyrazenia warunkowego*/

max=x>y ? x: y;
```

Instrukcja wyboru

- Pozwala zastąpić wiele instrukcji warunkowych sprawdzających wartość wyrażenia całkowitego
- Wykonywany jest fragment od miejsca zgodności wartości z wyrażeniem
- Aby przerwać wykonywanie dalszych instrukcji należy użyć polecenia break
- `default` zawiera instrukcje wykonywane, gdy żadna z wcześniejszych wartości nie pasowała do wyrażenia

Instrukcja wyboru - składnia

```
switch (wyrażenie)
{
 case wartosc1: instrukcje1;
 case wartosc2: instrukcje2; break;
 case wartosc3: instrukcje3; break;
 ...
 default: instrukcje;
}
```

Przykład użycia instrukcji wyboru

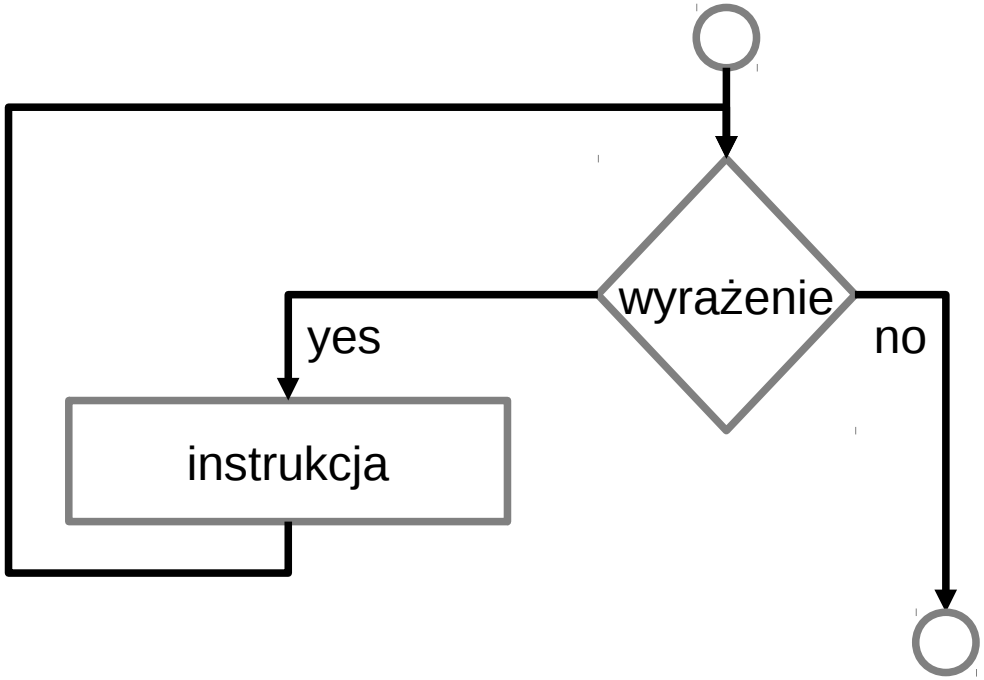
```
int x;

cout<<"Wprowadz numer miesiaca"<<endl;
cin>>x;

switch (x)
{
 case 4:
 case 6:
 case 9:
 case 11: cout<<"Miesiac ma 30 dni"<<endl; break;
 case 2: cout<<"Miesiac moze miec 28 lub 29 dni"<<endl;
          break;
 default: cout<<"Miesiac ma 31 dni"<<endl;
}
```

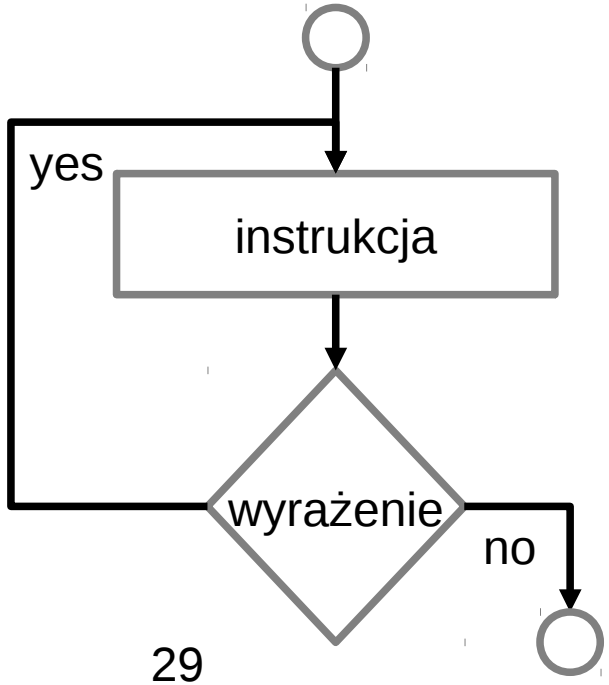
Pętle `while`

- Pozwalają na wielokrotne wykonywanie tych samych operacji bez konieczności wielokrotnego kopiowania kodu
- Pętla `while` sprawdza wyrażenie przed wykonaniem instrukcji
- Pętla `do while` sprawdza wyrażenie po wykonaniu instrukcji
- Polecenie `continue` przerywa wykonywanie instrukcji i przechodzi do sprawdzania wyrażenia
- Polecenie `break` przerywa wykonywanie pętli

Pętle `while` składnia

- Pętla `while`:
  while (wyrażenie) instrukcja;
- Pętla `do while`:
  do instrukcja; while (wyrażenie)





Pętle `while` przykłady

```
int n;

do
{
 cout<<"Wprowadz wartosc nieujemna"<<endl;
 cin>>n;
}while (n<0);

int s=1;

while (n)
{
 s*=n;
 n--;
}

cout<<"Silnia wynosi: "<<s<<endl;
```

Pętla `for`

- Pozwala na inicjalizowanie zmiennych, na początku wykonywania pętli
- Wyrażenie sprawdzane jest przed każdym obiegiem pętli
- Zmienne związane z kolejnym krokiem modyfikowane są po każdym obiegu pętli
- Składnia:
  for (inicjalizacja; wyrażenie; modyfikacja)
- Brak wyrażenia definiuje pętlę nieskończoną

Pętla for przykład

```
int n, s=1;

cout<<"Wprowadz wartosc n"<<endl;
cin>>n;

for (int i=1; i<=n; i++) s*=i;

cout<<"Silnia "<<n<<" wynosi: "<<s<<endl;
```

Wskaźniki

- Zmienna wskaźnikowa przechowuje informację o adresie
- Wskaźnik ma stały rozmiar, niezależnie od wskazywanego typu
- Aby zdefiniować zmienną wskaźnikową należy umieścić * przed jej nazwą
- Przed użyciem zmiennej wskaźnikowej należy nadać jej wartość - dla bezpieczeństwa można ustawić wartość NULL

Wskaźniki cd

- Adres elementu można pobrać przy pomocy operatora &
- Wartość znajdującą się pod wskazywanym adresem możemy pobrać przy pomocy operatora * (wyłuskania)

Wskaźniki - przykład

```
int a=1;
int *wa; //definicja zmienne wskaznikowej

wa=&a; //od tego miejsca wa wskazuje na a

cout<<"Wartosc zmiennej a "<<a<<endl;

cout<<"Wartosc zmiennej wa (adres) "<<wa<<endl;

cout<<"Wartosc zmiennej wskazywanej przez wa "<<*wa<<endl;
```

Tablice

- Służą do przechowywania większej ilości wartości danego typu
- Dostęp do poszczególnych elementów możliwy jest przez podanie jego indeksu
- Jeżeli tablica ma przechowywać n wartości jej pierwszy element ma indeks 0 a ostatni n-1
- Nazwa tablicy jest wskaźnikiem do jej pierwszego elementu
- Ponieważ elementy tablicy są ułożone w pamięci obok siebie można przy pomocy wskaźnika odwoływać się do kolejnych elementów

Tablice - przykłady

```
int n=5;
int tab[n]; //n jest rozmiarem tablicy

cout<<"Wprowadz wartosci poszczegolnych elementow"
for (int i=0; i<n; i++) cin>>tab[i];

cout<<"Pierwszy element tablicy "<<tab[0]<<endl;

cout<<"Ostatni element tablicy "<<tab[n-1]<<endl;

int *wTab=tab; //wTab wskazuje na poczatek tablicy

(*wTab)++; //zwiekszenie o jeden wartosci przechowywanej
           //w tablicy pod wskazywanym adresem

wTab++; //przesuniecie wskaznika na kolejny element
       //tablicy
```

Tablice cd

- Wartości w tablicy można umieszczać w momencie jej tworzenia:
  int tab[]={1, 3, 5, 7};
- Szczególnym rodzajem tablicy jest łańcuch znaków
- Inicjalizacji tablicy znaków możemy dokonać przy pomocy stałej łańcuchowej
- Tablica znaków powinna być dłuższa o jeden element od ilości znaków przechowywanego tekstu

Tablice znaków – przykłady

```
char lancuch[]="Witaj swiecie!";

cout<<"Pierwsza litera lancucha: "<<lancuch[0]<<endl;

cout<<lancuch<<endl;                //%s dla printf
cout<<sizeof(lancuch)<<endl;

cout<<"Wprowadz lancuch bez spacji"<<endl;
cin>>lancuch;
cout<<"Wpisany lancuch"<<endl;
cout<<lancuch<<endl;
```

Tablice wielowymiarowe

- Tablice, których elementami są tablice
- Przy definiowaniu tablicy można pominąć rozmiar jedynie pierwszego wymiaru
- Ponieważ pamięć jest jednowymiarowa tablica wielowymiarowa jest przechowywana w formie „rozpłaszczonej”
- Przy posługiwaniu się wskaźnikiem do elementów tablicy wielowymiarowej należy uwzględnić jej reprezentację w pamięci

Tablice wielowymiarowe - przykład

```
int tab1[2][2];  //przyklad definicji tablicy
                 //dwuwymiarowej

int tab2[][2]={1, 2, 3, 4};

cout<<tab2[1][0]<<endl; //przyklad wypisania wartosci

int *wsk=tab2[0]; //utworzenie wskaznika do wiersza
                  //tablicy

cout<<*wsk<<endl;

float tab3[4][3][4]; //przyklad tablicy trojwymiarowej
```

Dynamiczna alokacja pamięci

- Pozwala na przydzielanie oraz zwalnianie pamięci umożliwiając optymalizację zajętości
- Umożliwia tworzenie większych tablic niż w przypadku statycznym
- Pamięć przydzielamy przy pomocy operatora `new` (funkcje malloc, calloc w C)
- Operator ten zwraca wskaźnik do zarezerwowanego obszaru pamięci lub NULL
- Do zwalniania pamięci używamy operatora: `delete` dla pojedynczych elementów `delete[]` dla tablic (funkcja free w C)

Dynamiczna alokacja pamięci w C++ - przykłady

```
double *d=new double; //zaalokowanie obszaru pamieci dla
                      //dla zmiennej typu double


*d=1/3.;

cout<<*d<<endl;

delete d; //zwolnienie pamieci

int *tab=new int[5]; //alokowanie pamieci dla tablicy

tab[0]=7;

delete[] tab;
```