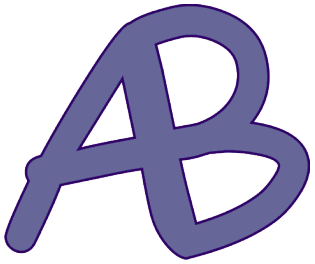


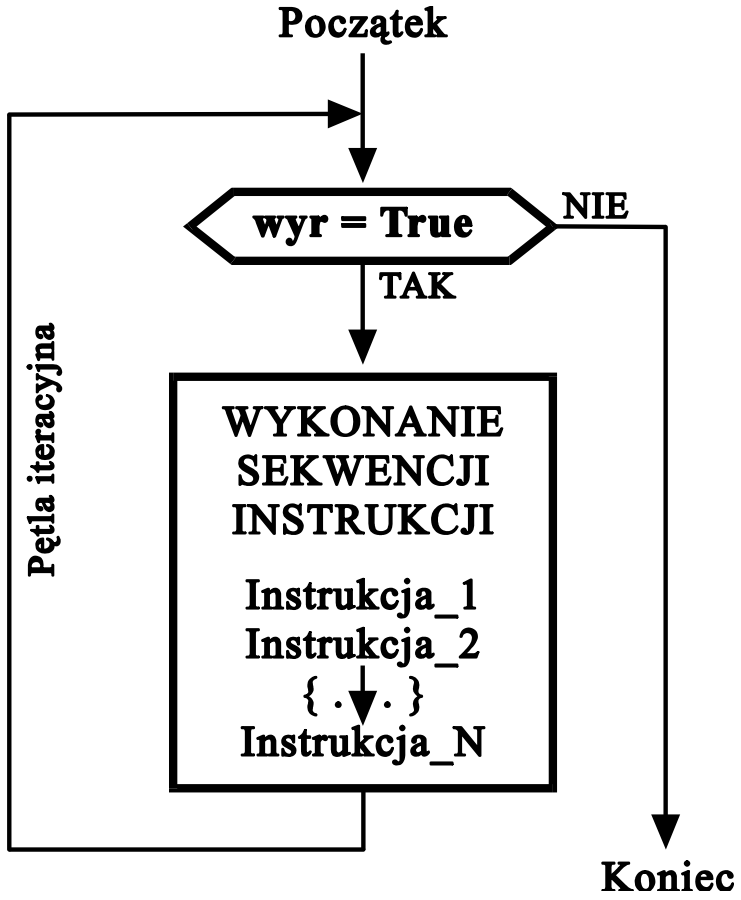
Wykład: 5

Instrukcje sterujące c.d.
Stałe,
Typy zmiennych c.d.

Instrukcje sterujące

Pętla while

```
while (wyrazenie)
instrukcja;
.........................
while (wyrazenie)
   {
     instrukcja_1;
     instrukcja_2;
   }
```




```
// pętla wyświetlająca liczby 1, 2, 3 ...

      int i = 1;
      while( i <=10 ) cout << i++ << „, „;
```

Pętla while

Przykład:

pętla wyświetlająca liczby 1, 2, 3 ...

```
int i = 1;
while( i <=10 ) cout << i++ << „, „;
```

Pętla while

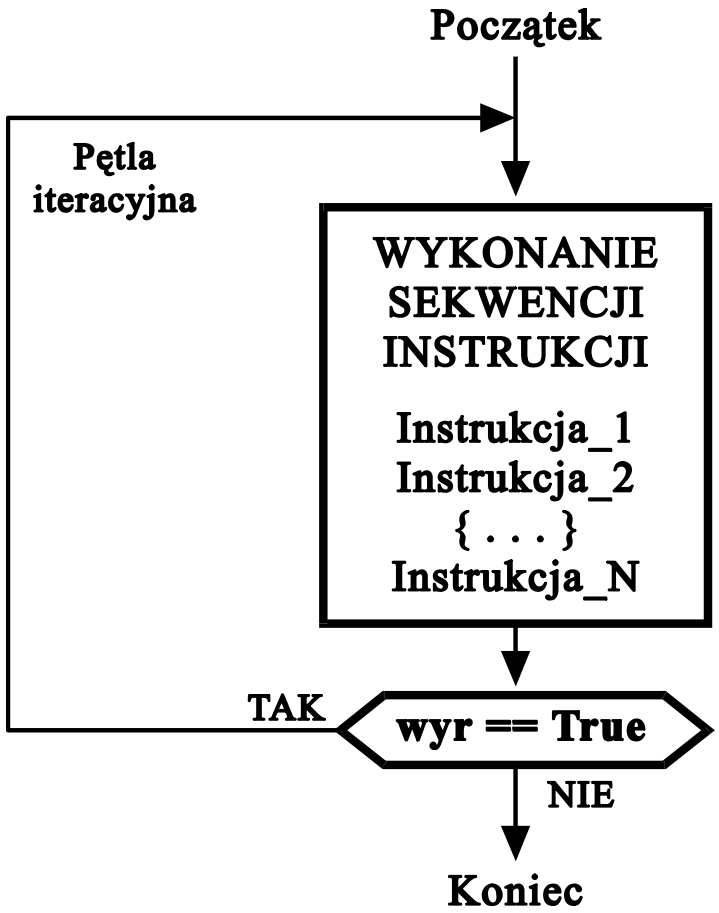
Program sumuje podane liczby, aż do chwili, gdy podamy liczbę 0.

```
1  #include <iostream>
2
3  using namespace std;
4  int main()
5  {
6    int liczba,suma;
7    suma=0;
8    liczba=0;
9    cout<<"Podaj kolejna liczbe: ";
10   cin>>liczba;
11   suma=suma+liczba;
12   while (liczba!=0)//wykonuje się jeżeli liczba jest różna od 0
13   {//poczatek petli
14     cout<<"Podaj kolejna liczbe: ";
15     cin>>liczba;
16     suma=suma+liczba;
17   }//koniec petli
18   cout<<"Suma wynosi: "<<suma;
19   return 0;;
20 }
```

Pętla do-while

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

```
do
   {
      instrukcja_1;
      instrukcja_2;
   }
while ( wyrażenie );
```



Pętla do-while

Przykład:

pętla wyświetlająca liczby 1, 2, 3 ...

```
int i = 1;
do
   {
     cout << i << „, „;
     i = i + 1;
   }
while( i<=10);
```

Pętla do-while

Program sumuje podane liczby, aż do chwili, gdy podamy liczbę 0.

```
1  #include <iostream.h>
2  #include <conio.h>
3  using namespace std;
4
5  int main()
6  {
7    int liczba,suma;
8    suma=0;
9    liczba=0;
10   do
11   {//poczatek petli
12     cout<<"Podaj kolejna liczbe: ";
13     cin>>liczba;
14     suma=suma+liczba;
15   }//koniec petli
16   while (liczba!=0);//wykonuje sie jezeli liczba jest rozna od 0
17   cout<<"Suma wynosi: "<<suma;
18   return 0;
19 }
```

Pętla do-while

Przykład: program zlicza naciśnięte klawisze (aż do chwili gdy naciśniemy klawisz ESC

```
1   #include <conio.h>
2   #include <iostream>
3
4   using namespace std;
5
6   int main()
7   {
8       int licznik = 0, znak;
9       cout << "Program zliczajacy naciskane klawisze (ESC - koniec)" ;
10      do
11          {
12              znak = getch();
13              licznik++;
14          }
15      while( znak != 27 ); // 27 = kod znaku Escape
16      cout << endl << "Nacisnieych klawiszy: " << licznik ;
17
18      return 0;
19  }
```

Instrukcje break i continue

break - kończy wykonywanie najbliższej otaczającej pętli lub instrukcji warunkowej, w której występuje. Jeśli po końcu przerwanej instrukcji występuje kolejna, sterowanie przechodzi do niej.

Przykład:

Pętla kończy się po wypisaniu i = 4

```
for (int i = 1; i < 10; i++)
{
    cout << i << '\n';
    if (i == 4)
        break;
}
```

Wyjście z pętli

Instrukcje break i continue

Instrukcja **break** jest używana z instrukcją warunkową **switch**, a także z instrukcjami pętli **do**, **for** i **while**.

W instrukcji **switch**, instrukcja **break** powoduje, że program wykonuje kolejną instrukcję, która występuje po instrukcji **switch**. Bez instrukcji **break**, wykonywane są wszystkie instrukcje od dopasowanej etykiety **case** do końca instrukcji **switch**, łącznie z klauzulą **default**.

W pętlach, instrukcja **break** kończy wykonywanie najbliższej otaczającej instrukcji **do**, **for** lub **while**. Sterowanie przechodzi do instrukcji następującej po zakończonej, jeśli taka istnieje.

Instrukcja break i co

continue - wymusza przekazanie kontroli do wyrażenia kontrolującego najmniejszej pętli do, for, lub while.

Przykład:

Pętla wypisze liczby od 1 do 10 pomijając liczbę 5

```
for (int i = 1; i <= 10; i++)
{
    if (i == 5)  continue;
    cout << i << '\n';
}
```

Powrót na górę pętli (z pomięciem instrukcji poniżej continue)

Pętla while

Przykład:

Algorytm Euklidesa

```
 1  #include <iostream>
 2  using namespace std;
 3  int main()
 4  {
 5      unsigned int a, b;
 6      cout << "a = ";    cin >> a;
 7      cout << "b = ";    cin >> b;
 8      while (a!=b)
 9          if (a>b) a-=b; else b-=a;
10      cout << "NWP(a,b) = " << a << endl;
11      return 0;
12  }
```

Instrukcje break i continue

Przykład:

Algorytm Euklidesa – wersja 2

```
1   #include <iostream>
2   using namespace std;
3   int main()
4   {
5       unsigned int a, b;
6       cout << "a = ";    cin >> a;
7       cout << "b = ";    cin >> b;
8       while (1)
9           if (a>b) a-=b;
10            else if (a<b) b-=a;
11                else break;
12      cout << "NWP(a,b) = " << a << endl;
13      return 0;
14  }
```

Pętla nieskończona

Wyjście z pętli

Instrukcja return

return - kończy wykonywanie funkcji i zwraca sterowanie do funkcji wywołującej (lub do systemu operacyjnego, jeśli kontrola zostanie przeniesiona z funkcji main). Wykonanie wznawia działanie w funkcji wywołującej w punkcie bezpośrednio po wywołaniu.

```
int normalizuj (int a)
{
    if (a>0)return(a);
      else if (a<0) return (-a);
        else return (0);
}
```

Typy c.d.

Typy zmiennych

Systematyka typów w języku C++

Typy zmiennych dzielą się na:

- typy fundamentalne
- typy pochodne (zmodyfikowane typy fundamentalne)

Oraz:

- typy wbudowane
- typy zdefiniowanie przez użytkownika

Typy fundamentalne omówione zostały w poprzednim wykładzie

Typy pochodne

Typy pochodne to modyfikacje typów podstawowych (ich rozwinięcia lub dopełnienia).r

[ ] – tablica obiektów danego typu
* - wskaźnik do typu
( ) - funkcja zwracająca wartość danego typu
& - referencja obiektu

Typ void

Typ **void** - typ nieokreślony.

- **void *p** -> p jest wskaźnikiem, ale jeszcze nie wiemy na jaki obiekt
- **void jakas_funkcja( )** -> funkcja nie zwraca wartości

Zakresy widoczności zmiennych

Zakres ważności nazwy zmiennej – obszar w programie, w którym jest ona rozpoznawana przez kompilator (niedokładnie to samo, co czas życia zmiennej)

Wyróżniamy zakresy:

- **Lokalny** - gdy świadomie ograniczamy go do jakiegoś bloku.

```
{
   int x;
}
```

- **Blok funkcji** – wewnątrz całej funkcji


```
int main()
{
    int x;
}
```

Zakresy widoczności zmiennych

- **Obszar pliku (globalnie)** - zmienną taką definiujemy poza obszarem funkcji i jest widziana przez wszystkie funkcje programu (w obrębie pliku)

```
int x;
int main()
{
   ........
}
```

- **Obszar klasy**

  // jeszcze do tego wrócimy

Modyfikatory zmiennych

- register - informujemy kompilator, że zmienna będzie często wykorzystywana. Kompilator może (nie musi) przyspieszyć program przechowując zmienną w rejestrze zamiast w pamięci operacyjnej. Do zmiennej typu register nie można odwołać się poprzez adres (wskaźnik).

```
register int i;
```

- voletile – zmienna „ulotna”. Może być zmieniona w sposób ukryty przed kompilatorem. Np. odczytujemy wartość z układu we/wy.

```
voletile int stan_ukladu;
```

Stałe

Stałe dosłowne:

- Sam zapis liczb, a nie obiekt, któremu nadajemy jakąś wartość.

```
X = 10;
cout << 3.14;
cout << „Ala ma kota”;
```

Stałe

Stałe dosłowne liczbowe można zapisać nie tylko w systemie dziesiętnym.

- Rozpoczynając zapis od zera informujemy kompilator, że używamy systemu ósemkowego.

  011 -> to liczba 9

- Prefiks 0x oznacza system szesnastkowy

  0x10 -> 16

Stałe

Stałe dosłowne zmiennoprzecinkowe

- 10.0
- 15.5
- -10.
- 8E3 -> $8*10^{3} = 8000$
- 5.2E-3 -> $5.2*10^{-3} = 0,0052$

Stałe

Stałe znakowe i łańcuchowe

```
'a'
'7'
"Ala ma kota"
"7.1"
```

Stałe

Stałe dosłowne przechowywane są jako typy zmiennych – kompilator dobiera jakie.

Można wymuszać zmianę typu np.:

10 -> zapisana jako int
10L -> zapisana jako long int
10uL -> unsignet long int

Stałe

Modyfikator **const**

float pi = 3.14;

- zmienna (może być zainicjalizowana)

const float pi = 3.14;

- stała (musi być zainicjalizowana)

W klasycznym C używana była dyrektywa preprocesora #define. Powodowała ona wstawienie liczby w miejsce każdego wystąpienia stałej.

#define pi 3.14;

Typy zmiennych

Instrukcja typedef - pozwala na nadanie dodatkowej nazwy już istniejącemu typowi.

Np.:

```
typedef unsigned int wiek;
wiek Ania, Tomek;
      // to samo co: unsigned int Ania, Tomek;
```

Albo:

```
typedef int * wsk_na_int;
```

Instrukcja typedef nie wprowadza nowego typu, a jedynie synonim do typu już istniejącego

Typy wyliczeniowe enum

Jest to osobny typ dla liczb całkowitych, a właściwie lista wartości stałych całkowitych Const Int.

Deklaracja typu wyliczeniowego składa się ze słowa kluczowego **enum** następnie nazwy (opcjonalnie), po której znajduje się w nawiasach klamrowych lista elementów oddzielonych przecinkami.

```
enum dzien {PON, WT, SR, CZW, PT, SOB, NIE};
```

Pierwszy element na liście ma domyślnie wartość 0, następny - wartość 1, 2, 3 itd..

```
dzien dzis = CZW;
dzis++;
if (dzis == 4) cout << "piąteczek!";
```

Typy wyliczeniowe enum

Można zdefiniować wartość początkową lub też wszystkie wartości stałych.

```
enum dzien {PON = 1, WT, SR, CZW, PT, SOB, NIE};
 /* element PON ma wartość 1, WT ma wartość 2, z
  * kolei NIE ma wartość 7 */

 enum marka {VOLKSWAGEN, AUDI, SEAT, SKODA = 9,
             BENTLEY, BUGATTI = 32, LAMBORGHINI};
 /* powyższa deklaracja przypisuje elementom
  * następujące wartości:
  * VOLKSWAGEN = 0, AUDI = 1, SEAT = 2,
  * SKODA = 9, BENTLEY = 10,
  * BUGATTI = 32, LAMBORGHINI = 33 */
```

Liczby losowe

Funkcja:

int rand(); // #include <cstdlib>

zwraca pseudolosową liczbę całkowitą, która zawiera się w zakresie od 0 do stałej RAND_MAX.

```
#include <cstdlib>
#include <cstdio>
#include <ctime>

int main()
{
    srand( time( NULL ) );

    for( int i = 0; i < 10; i++ )
        printf( "Wylosowano %d\n", rand() );

}
```

Liczby losowe

Funkcja **srand( )** ustawia punkt startowy generowania serii pseudolosowych liczb całkowitych.

W celu reinicjalizowania generatora, ustaw wartość 1 dla argumentu seed. Każda inna wartość przekazana jako seed ustawia losowy punkt startowy generatora. Funkcja pobiera pseudolosowe wartości, które są generowane.

```
#include <cstdlib>

void srand( unsigned int seed );
```

Losowanie liczb z określonego zakresu

int losowa_liczba =
(rand() % ile_liczb_w_przedziale) + startowa_liczba;

Losowanie 100 liczb z zakresu <-10; 10>

```
1  #include <iostream>
2  #include <stdlib.h>
3  #include <ctime>
4
5  using namespace std;
6
7  int main()
8  {
9      srand (time(NULL));
10     for (int i=0;i<100;i++)
11       cout << rand()%21 -10 << "  ";
12     return 0;
13 }
```

Literatura:

W prezentacji wykorzystano przykłady i fragmenty:

- Gręboszj. : ***Symfonia C++, Programowanie w języku C++ orientowane obiektowo***, Wydawnictwo Edition 2000.
- Jakubczyk K.: *Turbo Pascal i Borland C++ Przykłady*, Helion.

Warto zajrzeć także do:

- Sokół R. : ***Microsoft Visual Studio 2012 Programowanie w Ci C++,*** Helion.
- Kerninghan B. W., Ritchie D. M.: ***język ANSI C***, Wydawnictwo Naukowo Techniczne.

Dla bardziej zaawansowanych:

- Gręboszj. : ***Pasja C++,*** Wydawnictwo Edition 2000.
- Meyers S.: ***język C++ bardziej efektywnie***, Wydawnictwo Naukowo Techniczne