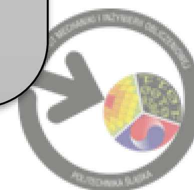
JĘZYKI PROGRAMOWANIA
Z
PROGRAMOWANIEM OBIEKTOWYM

Wykład 3

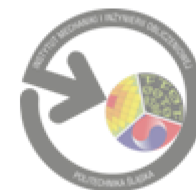


Dlaczego C++?

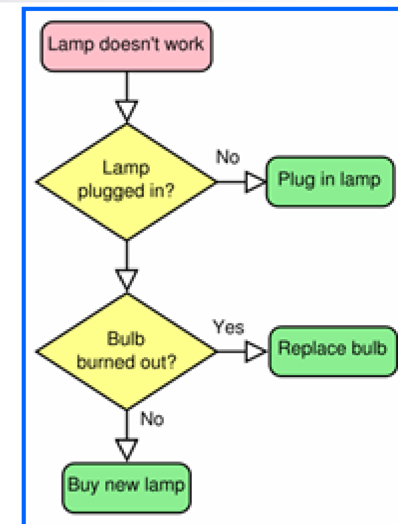
- na coś trzeba się zdecydować ...;
- bardzo duże możliwości;
- szybkość działania;
- zwięzła składnia;
- przenośność (dostępne kompilatory na różne systemy operacyjne);
- język hybrydowy (patrz: następne slajdy).

Skąd taka (dziwna) nazwa?

- było **C** (Dennis Ritchie, 1972)
- mogło być **D** (istnieje zresztą taki język, 2007)
- `x++` oznacza `x+1`
- zatem: **C++** (Bjarne Stroustrup, 1983)
  to nawiązanie do C i jego rozszerzenie
  (przy zachowaniu maksymalnej zgodności z C)



Programowanie strukturalne



- „zrób najpierw to, a potem tamto”;
- hierarchiczne dzielenie kodu na bloki, z 1 punktem wejścia i jednym (lub wieloma) punktami wyjścia;
- nieużywanie (lub ograniczenie) instrukcji skoku (**goto**).
- „dobre struktury” to np. instrukcje warunkowe (**if**, **if**...**else**), pętle (**while**, **do...while**);
- „złe struktury” (zakłócające strukturalność) to np. **break**, **continue**, **switch** (w C itp.);
- programowanie strukturalne tworzy programy bardziej zrozumiałe niż niestrukturalne, łatwiejsze do sprawdzania i usuwania błędów oraz do modyfikacji.

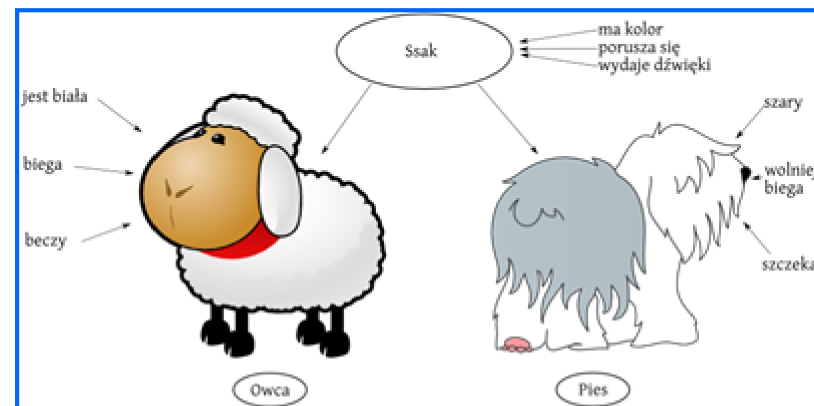
Programowanie strukturalne

Każdy program da się napisać (bez użycia **goto**) w oparciu o trzy struktury sterujące:

- sekwencja – wykonanie instrukcji w określonej kolejności (np. operator sekwencji w C i C++ to średnik).
- wybór – wykonanie jednej z kilku instrukcji zależnie od stanu programu (**if...else**, **switch...case**).
- iteracja – powtarzanie instrukcji tak długo, jak długo spełniony (lub niespełniony) jest dany warunek (np. pętle **while**, **do...while**, **for**).

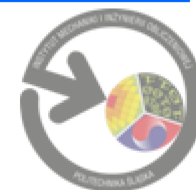
Programowanie orientowane obiektowo

- bardziej (wbrew pozorom ☺?) intuicyjne pisanie programów;
- program jest zbiorem komunikujących się ze sobą *obiektów* (jednostek zawierających określone dane i potrafiących wykonywać na nich określone operacje);
- powiązanie danych (*zmienne*, *pola*) z operacjami na nich (*funkcje*, *metody*) w całość, stanowiącą odrębną jednostkę – obiekt;
- obiekty są elementami do ponownego użycia, tworzonymi z „planów" zwanych *klasami*;
- mechanizm dziedziczenia – możliwość definiowania nowych, bardziej złożonych obiektów, na podstawie obiektów już istniejących;



C++ jest językiem **hybrydowym**:

- umożliwia zarówno programowanie strukturalne, jak i programowanie orientowane obiektowo.



Elementy języka programowania

Składnia (syntaktyka) języka programowania:

Opisuje rodzaje dostępnych symboli i zasady ich łączenia w większe struktury.
Kolorowanie składni - często wykorzystywane w edytorach kodu, ułatwia czytanie kodu:

```
int main()
{
    unsigned int Li;
     cout<<"Ciag Fibonacciego rekurencyjnie..."<<endl;
    do {
        cout << "Podaj liczbe naturalna " << endl;
        cin>>Li;
        cout<<"Wynik to: "<<fibo(Li)<<endl;
        } while (getch()!='k');
return 0;
}
```

Semantyka języka programowania:

- Definiuje precyzyjnie znaczenie poszczególnych symboli oraz ich funkcję w programie;
- Semantykę najczęściej definiuje się słownie.
- Błędem semantycznym jest np. odwołanie się do nieistniejącej funkcji.

Elementy języka programowania

Typy danych:

Każdy język programowania operuje na pewnym zestawie danych. Niezbędne jest podzielenie danych na odpowiednie typy, zdefiniowane ich właściwości oraz operacji na nich.

Większość języków posiada różne typy danych do reprezentowania:

- liczb całkowitych (różne zakresy);
- liczb zmiennoprzecinkowych (różne stopnie dokładności);
- ciągów tekstowych.

Biblioteka standardowa:

Dla większości języków zawiera podstawowy zestaw funkcji pozwalających realizować wszystkie najważniejsze operacje, np.:

- obsługę WE-WY;
- obsługę plików;
- zarządzanie pamięcią;
- podstawowe typy danych i funkcje do zarządzania nimi;
- operacje na ciągach tekstowych.

ETAPY TWORZENIA PROGRAMU

Tworzenie programu – etapy:

1. Opracowywanie kodu (tekstu) źródłowego programu;
2. Kompilacja;
3. „Linkowanie" (łączenie).

Ad. 1. Zapis kodu programu za pomocą instrukcji danego języka (np. C++) w postaci pliku tekstowego (zwykle z rozszerzeniem .cpp).

C++ (również C) jest językiem o tzw. wolnym formacie. Tzw. „białe znaki" (spacje, tabulatory, znaki nowego wiersza) są prawie zawsze ignorowane. Wstawianie tych znaków służy programiście do zwiększenia czytelności kodu.

Do pisania tekstów programów służą edytory, zwykle dostępne wraz z kompilatorem. Zazwyczaj oferują one tzw. kolorowanie składni, zwiększające czytelność programu.

Należy korzystać z edytorów, które nie dodają do tekstu innych informacji (np. o jego formatowaniu).

ETAPY TWORZENIA PROGRAMU

Tworzenie programu – etapy:

1. Opracowywanie kodu (tekstu) źródłowego programu;
2. Kompilacja;
3. „Linkowanie” (łączenie).

Ad. 2. Tłumaczenie kodu źródłowego programu na język maszynowy. Kompilacja zwykle odbywa się automatycznie po wybraniu stosownej opcji w środowisku.

W wyniku kompilacji otrzymuje się skompilowaną wersję programu (jeszcze niepełną).

Ad. 3. Jest to proces łączenia programu z bibliotekami wykonywany przez program zwany „linkerem”. Łączy on skompilowany program z bibliotekami, o których nagłówkach poinformowaliśmy kompilator dyrektywami `#include` (by kompilator mógł sprawdzić poprawność ich użycia).

Polska nazwa „linkowania” w postaci: „konsolidacji” nie przyjęła się...

Operacja linkowania jest zwykle wykonywana automatycznie tuż po kompilacji.

W efekcie uzyskiwany jest program w postaci pliku wykonywalnego (z rozszerzeniem .exe)

PRZYKŁADOWY PROGRAM


```
#include <iostream>

using namespace std;

int main()
{
    int a;
    cout<<"No to zaczynamy!"<<endl;
    cout<<"\nPodaj jakas liczbe calkowita: ";
    cin>>a;
    cout<<"Podales<<a;
return 0;
}
```

Wrócimy do tego programu…

BIBLIOTEKA STANDARDOWA C++

Programy w C++ składają się z części zwanych *klasami* i *funkcjami*.

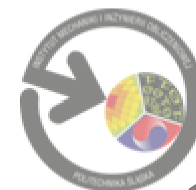
Biblioteka standardowa C++ zawiera bogaty zbiór:

- funkcji do przeprowadzania operacji matematycznych,
- funkcji do manipulacji napisami i znakami,
- funkcji sterujących operacjami wejścia/wyjścia,
- funkcji sprawdzania błędów i wielu innych użytecznych operacji…

Korzystanie z bibliotek bardzo ułatwia pracę programisty (brak konieczności „powtórnego wymyślania koła”). Programy w C++ są zazwyczaj tworzone przez łączenie nowych funkcji i klas, napisanych przez programistę, z funkcjami i klasami z biblioteki standardowej C++ oraz z innych niestandardowych bibliotek klas („wielokrotne użycie oprogramowania”).

Funkcje i klasy biblioteki standardowej są udostępniane jako część środowiska programistycznego C++ (zwykle dostarczane przez dostawców kompilatorów).

Wiele bibliotek klas specjalnego przeznaczenia jest dostarczanych przez niezależnych dostawców oprogramowania.



BIBLIOTEKA STANDARDOWA C++

Standardowe pliki nagłówkowe C++:

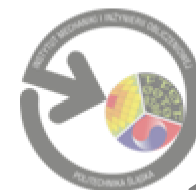
<algorithm> <cstdarg> <ios> <ostream> <bitset> <cstddef> <iosfwd>
<queue> <cassert> <cstdio> <iostream> <set> <cctype> <cstdlib> <istream>
<sstream> <cerrno> <cstring> <iterator> <stack> <cfloat> <ctime> <limits>
<stdexcept> <ciso646> <cwchar> <list> <streambuf> <climits> <cwctype>
<locale> <string> <clocale> <deque> <map> <typeinfo> <cmath> <exception>
<memory> <utility> <complex> <fstream> <new> <valarray> <csetjmp>
<functional> <numeric> <vector> <csignal> <iomanip>

Można tworzyć własne pliki nagłówkowe, które powinny się kończyć rozszerzeniem .h. Plik nagłówkowy zdefiniowany przez programistę jest dołączany – podobnie jak pliki nagłówkowe bibliotek standardowych – przez użycie stosownej dyrektywy preprocesora, np.

```
#include ”moja_biblioteka.h”
```

PRZYKŁADOWY PROGRAM

```
#include <iostream>

using namespace std;

int main()
{
     int a;
     cout<<"No to zaczynamy!"<<endl;
     cout<<"\nPodaj jakas liczbe calkowita: ";
     cin>>a;
     cout<<"Podales<<a;
return 0;
}
```

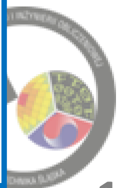


PRZYKŁADOWY PROGRAM

```
#include <iostream>
/* dyrektywa preprocesora włączająca plik nagłówkowy
   biblioteki iostream do programu
*/

using namespace std; //użycie nazw z obszaru bibliotek standardowej

int main() //funkcja główna programu
{
     int a; //definicja (i deklaracja) zmiennej typu całkowitego
    cout << "No to zaczynamy!" << endl; //wyświetlenie tekstu na ekran
    // cout<<"No to zaczynamy!\n;"    //przejście do nowej linii inaczej
    // std::cout<<"No to zaczynamy!"<<endl;   //gdyby nie było "using..."

    cout<<"\nPodaj jakas liczbe calkowita: ";
    cin>>a; // Wczytanie danych z klawiatury
    cout<<"Podales "<<a;

return 0; //funkcja main() zwraca 0 (prawidłowe zakończenie programu)
}
```

Fragment pliku nagłówkowego iostream

```
/** @file iostream
 *  This is a Standard C++ Library header.
 */

//
// ISO C++ 14882: 27.3  Standard iostream objects
//

#ifndef _GLIBCXX_IOSTREAM
#define _GLIBCXX_IOSTREAM 1

#pragma GCC system_header

#include <bits/c++config.h>
#include <ostream>
#include <istream>

_GLIBCXX_BEGIN_NAMESPACE(std)

  /**
   *  @name Standard Stream Objects
   *
   *  The &lt;iostream&gt; header declares the eight <em>standard stream
   *  objects</em>.  For other declarations, see
   *  http://gcc.gnu.org/onlinedocs/libstdc++/manual/bk01pt11ch24.html
   *  and the @link iosfwd I/O forward declarations @endlink
   *
   *  They are required by default to cooperate with the global C
   *  library's @c FILE streams, and to be available during program
   *  startup and termination. For more information, see the HOWTO
   *  linked to above.
  */
  //@{
  extern istream cin;        ///< Linked to standard input
  extern ostream cout;       ///< Linked to standard output
  extern ostream cerr;       ///< Linked to standard error (unbuffered)
  extern ostream clog;       ///< Linked to standard error (buffered)

#ifdef _GLIBCXX_USE_WCHAR_T
  extern wistream wcin;      ///< Linked to standard input
  extern wostream wcout;     ///< Linked to standard output
  extern wostream wcerr;     ///< Linked to standard error (unbuffered)
  extern wostream wclog;     ///< Linked to standard error (buffered)
#endif
  //@}
```



WAŻNE!

- Programy w C++ rozpoczynają wykonywanie od funkcji głównej programu `main()`.
- Wszystkie zmienne w C++ muszą być zadeklarowane przed ich użyciem.
- Każda instrukcja w języku C++ MUSI kończyć się średnikiem; **;**
- Nazwa zmiennej jest dowolnym dozwolonym identyfikatorem, który jest serią znaków składającą się z liter, cyfr i znaków podkreślenia, która nie rozpoczyna się cyfrą.
- C++ rozróżnia wielkość znaków (małe/wielkie litery).
- Język C++ jest językiem o tzw. wolnym formacie, tzn. kod programu może się znaleźć w każdym miejscu linii, lub może być rozpisany na wiele linii.
  Poza nielicznymi sytuacjami, w dowolnym miejscu instrukcji można przejść do nowej linii i kontynuować pisanie (dlatego, że każda instrukcja kończy się średnikiem…)

NAZWY ZAREZERWOWANE

Słowa kluczowe w C++ (będziemy je stopniowo poznawać…):

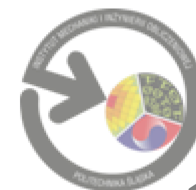
asm	auto	bool	break	case
catch	char	class	const_cast	continue
default	delete	do	double	else
enum	dynamic_cast	extern	false	float
for	union	unsigned	using	friend
goto	if	inline	int	long
mutable	virtual	namespace	new	operator
private	protected	public	register	void
reinterpret_cast	return	short	signed	sizeof
static	static_cast	volatile	struct	switch
template	this	throw	true	try
typedef	typeid	unsigned	wchar_t	while

Tzw. zamienniki operatorów:

and	and_eq	bitand	bitor	compl	not
not_eq	or	or_eq	xor	xor_eq	

CZĘSTE BŁĘDY:

- Próba skorzystania z jakiejś funkcji bibliotecznej bez dołączenia odpowiedniego nagłówka (dyrektywą preprocesora `#include`). Kompilator wygeneruje komunikat o błędzie.
- Brak średnika na końcu instrukcji jest błędem składni. Kompilator nie może rozpoznać instrukcji i wygeneruje komunikat o błędzie.
- Błędem składni jest rozdzielanie identyfikatorów przez wstawianie znaków odstępu (spacji) w ich nazwach, np. pisanie **ma in** zamiast **main**.



DOBRY STYL PROGRAMISTY

- Pisz swoje programy w prosty i bezpośredni sposób.
- Każdy program powinien rozpoczynać się komentarzem opisującym swoje przeznaczenie. Stosuj komentarze w tekście programu – ułatwią jego późniejszą analizę (np. po latach).
- Nadawaj zmiennym nazwy w taki sposób, aby oddawały ich przeznaczenie. Pomaga to programowi być „samodokumentującym”, przez co łatwiejsze jest zrozumienie programu przez jego samo czytanie.
- Unikaj nazw zmiennych (identyfikatorów), które rozpoczynają się znakiem pojedynczego lub podwójnego podkreślenia – kompilator może używać takich nazw dla swoich celów.

TYPY DANYCH

Zmienna – obiekt, który może się zmieniać podczas wykonywania programu.
Zmienne mają określone nazwy i są określonych typów.

W C++ każda nazwa musi zostać zadeklarowana przed użyciem.

Deklaracja informuje kompilator, że dana nazwa reprezentuje obiekt danego typu (co może być w niej przechowywane). Sama deklaracja nie rezerwuje miejsca w pamięci.

Definicja dodatkowo rezerwuje miejsce w pamięci – powołuje obiekt do życia.

Definicja jest również deklaracją (ale nie odwrotnie...)

Przykłady:

```
float Cisnienie      // deklaracja i definicja w jednym
extern float Cisnienie // tylko deklaracja (definicja np.
                       // w innym pliku
```

Do deklaracji i definicji wrócimy np. przy omawianiu funkcji...

Podział typów (1):

- wbudowane – składniki języka C++
- zdefiniowane przez użytkownika

Podział typów (2):

- fundamentalne – podstawowe
- złożone – wykorzystujące w swej budowie typy fundamentalne

To, ile zmienna danego typu zajmuje pamięci zależy od typu komputera oraz od kompilatora.

Sprawdzenie rozmiaru typu (wynik w bajtach):

```
cout<<sizeof(int);
```

TYPY FUNDAMENTALNE

- Typ wbudowany reprezentujący znaki alfanumeryczne:

```
char Znak='a';                    //znaki w kodzie ASCII, a ma kod 97
```

- Typy wbudowane reprezentujące liczby całkowite:

```
short Mala_calkowita=5; //inna nazwa: short int
int Jakas_calkowita=32700;
long Duza_calkowita=13e6; //inna nazwa: long int
```

- Typy wbudowane reprezentujące liczby zmiennoprzecinkowe:

```
float Srednica=13.76;
double Odleglosc=32.7e+8;
long double Masa=13.236e+23;
```

Wszystkie powyższe typy mogą być w dwóch wariantach:

- ze znakiem (**signed**)
- bez znaku (**unsigned**).

Przez domniemanie typ występuje ze znakiem, np. **int** a oznacza **signed int** a (czyli może przechowywać wartości dodatnie i ujemne).

TYPY FUNDAMENTALNE

- Typ wbudowany reprezentujący rozszerzony zestaw znaków alfanumerycznych:

```
wchar_t
```

- Typ wbudowany reprezentujący obiekty logiczne:

```
bool czy_prawda=true; // true=1, false=0
```

Typ	Szerokość (bajty)	Typowy przedział wartości
`(signed) char`	1	-128…127
`unsigned char`	1	0…255
`wchar_t`	2	0…65535
`(signed) short int`	2	-32768…32767
`unsigned short int`	2	0…65535
`(signed) int`	4	-2147483648… 2147483647
`unsigned int`	4	0…4294967295
`(signed) long int`	4	-2147483648… 2147483647
`unsigned long int`	4	0…4294967295
`float`	4	3.4E-38…3.4E+38
`double`	8	1.7E-208…1.7E+308
`long double`	10	3.4E-4932.. 1.1E+4932

Co się dzieje po przekroczeniu zakresu?

Zadanie domowe (proszę uruchomić):

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

int main()
{
    int a=2147483647;  //kraniec zakresu int

    cout<<a<<endl;
    ++a;                  //dodajemy 1...
    cout<<a<<endl;  //niespodzianka!

    return 0;
}
```

Są to błędy trudne do wykrycia...

- Typ wyliczeniowy **enum**

- osobny typ dla liczb całkowitych. Przydaje się, gdy w obiekcie typu całkowitego chcemy przechowywać pewien rodzaj informacji.

Przykład:

```
enum KIERUNEK1 {gora, dol, lewo, prawo};
enum KIERUNEK2 {wschod=0, zachod, polnoc=5, poludnie};
```

Definicja zmiennych typu wyliczeniowego KIERUNEK1 oraz KIERUNEK2:

```
KIERUNEK1 ruch_typu_1;
KIERUNEK2 ruch_typu_2;
```

Do zmiennych ruch_typu_1 i ruch_typu_1 można podstawić tylko jedną z wartości na liście wyliczeniowej. Czyli: dozwolone są następujące operacje:

```
ruch_typu_1=gora; ruch_typu_2=zachod;
```

Niedozwolone są następujące operacje:

```
ruch_typu_1=0; ruch_typu_2=5;
```

Przez domniemanie lista wyliczeniowa zaczyna się od 0 i dalej co 1. Programista może to wyliczanie dowolnie określić.

TYPY DANYCH

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

int main()
{
    enum  GDZIE {gora, dol, lewo, prawo=5};    // definiujemy typ wyliczeniowy
    cout<<gora<<" "<<dol<<" "<<lewo<<" "<<prawo<<endl; // wyświetlamy wartości z listy

    GDZIE ruch=dol;        // definiujemy zmienną typu GDZIE i przypisujemy...
                                                  // ...wartość z listy
     cout<<endl<<ruch;
      if (ruch==5) cout<<"\n ruch w prawo";    // w zależności czy ruch==5
      else cout<<"\n ruch nie w prawo..."<<endl;

    ruch=prawo;            // przypisujemy inną wartość z listy wyliczeniowej
    cout<<endl<<ruch;
     if (ruch-5) cout<<"\n ruch nie w prawo"<<endl; //inaczej (wrócimy do tego...)
     else cout<<"\n ruch w prawo..."<<endl; //nieco inaczej...

    //ruch=0;                                     //BLAD! Tak nie wolno!

     return 0;
}
```

```
0 1 2 5
1
 ruch nie w prawo...
5
 ruch w prawo...
```

TYPY POCHODNE

Typy pochodne oznacza się stosując nazwę typu, od którego pochodzą, oraz operator deklaracji typu pochodnego.

Operatory do tworzenia obiektów typów pochodnych:

`[]` – tablica obiektów danego typu;

`()` – funkcja zwracająca wartość danego typu;

`*` – wskaźnik do pokazywania na obiekty danego typu;

`&` – referencja (przezwisko) obiektu danego typu.

W deklaracjach typów złożonych może się pojawiać słowo **`void`** (pusty) w miejscu, gdzie stawia się nazwę typu.

Np.:

```
void wypisz();
```

oznacza, że funkcja `wypisz()` nie zwraca żadnej wartości.

Więcej o typach pochodnych – na kolejnych wykładach...