http://kfd2.phys.uni.lodz.pl/~malinow/

Programowanie I
C++

Wykład dla studentów
Informatyki Stosowanej oraz Fizyki
Wydz. Fizyki i Informatyki Stosowanej UŁ

Prowadzący – dr Jan Malinowski

Tytuł:	**Języki Programowania I (1500-ISI1JP / 1500-ZZI1JP / 1500-FKL1PP)**
Wykładowca:	dr J. Malinowski
Termin:	rok I, semestr zimowy
Liczba godzin zajęć dydaktycznych:	14 (9) godz. wykład, 42 (18) godz. laboratorium
Punkty ECTS:	0 lub 2 **lub 4**
Język:	Polski
Przedmioty wprowadzające, wymagania wstępne:	Podstawy obsługi komputera; Narzędzia programistyczne: Code Blocks, DevCpp, MS Visual

Forma i warunki zaliczenia przedmiotu:

Na wszystkich zajęciach obecność obowiązkowa

Wymagania dla zaliczenia wykładu:

Obecności na wykładzie (40 – 70 %; ostatnio 40 %).

Test pisemny lub ustny.

- Znajomość materiału z zakresu wykładu;
- znajomość elementów programowania z ćwiczeń laboratoryjnych

  (np. napisać fragment programu, zaprojektować funkcję, lub przeanalizować fragment programu).

Wymagania minimalne do zaliczenia ćwiczeń laboratoryjnych:

Umiejętność zaprojektowania i napisania funkcji; użycie wskaźników i referencji jako parametrów funkcji.

Obsługa plików tekstowych i binarnych – I/O.

Umiejętność algorytmizowania prostych algorytmów z użyciem aparatu matematycznego lub fizycznego.

Zaliczone sprawdziany i kolokwia podczas ćwiczeń laboratoryjnych.

Treści programowe:

1. Zasady układania algorytmu i pisania programu; podstawowe instrukcje.
2. Pojęcie funkcji, argumenty funkcji.
3. Parametry domyślne funkcji.
4. Przeciążanie funkcji.
5. Rekurencja.
6. Obsługa plików w trybie tekstowym i binarnym (zapis na dysk / odczyt z dysku).
7. Obiekt, wskaźnik, referencja; (m.in. argumenty funkcji).
8. Tablice; operacje na wskaźnikach.
9. Tablice wielowymiarowe.
10. Struktury dynamiczne; operatory new i delete.
11. Informacje wprowadzające do tematu ‘Generatory liczb losowych’.

Literatura:

1. **B.Stroustrup** ***Język C++, WN-T Warszawa 1997;***
   The C++ programming language
2. J. Grębosz *Symfonia C++, wyd. Oficyna Kallimach, 1994*
   (i późniejsze wyd.)
3. T. Swan *Mastering Borland C++, SAMS Indiana, USA*
   (lub inne pozycje tego autora)
4. S. B. Lipman *C++ Primer, wyd. AT&T Bell Lab., 1989*
5. *Microsoft Visual C++ on-line help*
6. K. Kuczmarski *Kurs C++ (wersja elektroniczna)*
7. Scott Meyers *Effective C++ (50 new ways*
   to improve your programs and designs)
8. Scott Meyers *Effective C++ (35 new ways*
   to improve your programs and designs)

Literatura dodatkowa:

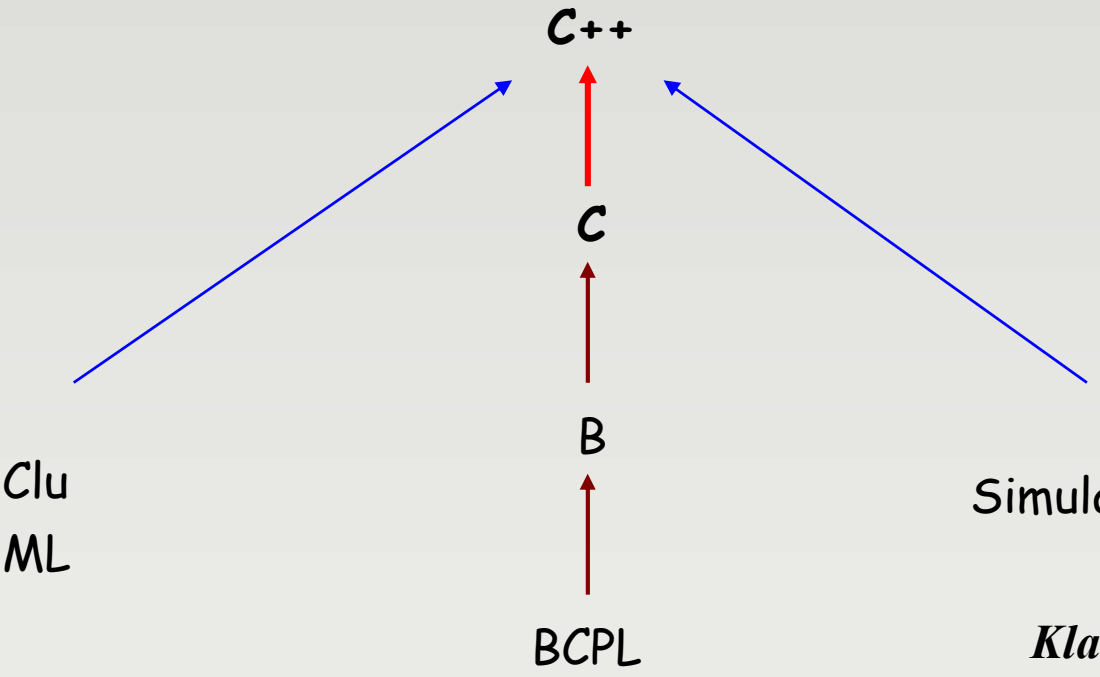
1. *P. Demidowicz, I. A. Maron - Metody Numeryczne, PWN Warszawa 1965*
2. *R. Zieliński - Generatory liczb losowych.*

Język ...

1. Co powiedzieć.
2. Jak powiedzieć.
3. Jakiego języka użyć.

1. Zrozumieć zadanie.
2. Ułożyć algorytm;
   (algorytmy,
   metody numeryczne,
   metody statystyczne).
3. Jakiego języka użyć.

1. Wiedzieć co mamy powiedzieć.
2. Ułożyć ciąg zdań.
3. Jakiego języka użyć.

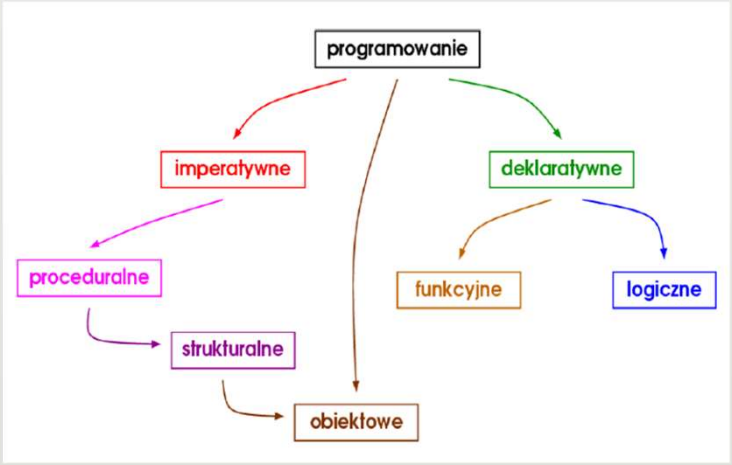


Obsługa wyjątków

Klasy

Dziedziczenie

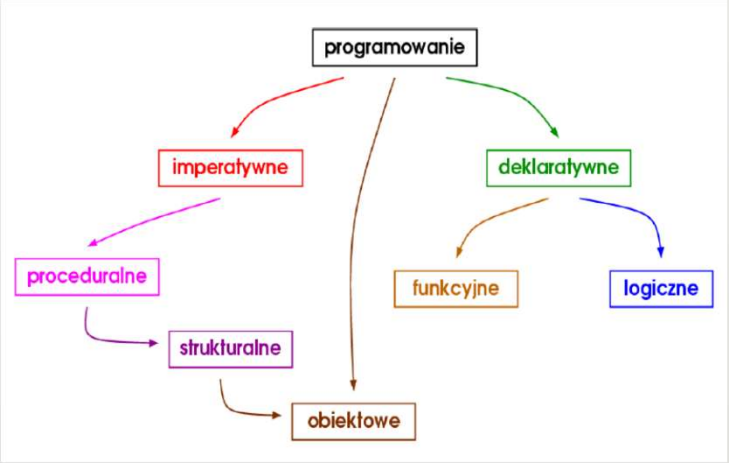
Funkcje virtualne



Techniki programowania:

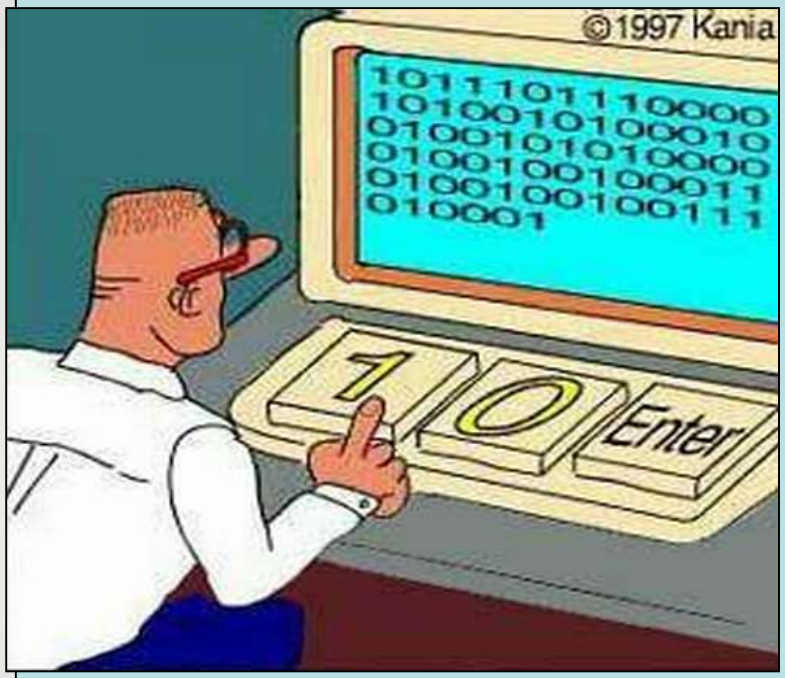
(wspomagane przez język programowania)

Liniowe	Assembler
Strukturalne (?)	
Proceduralne	Fortran Algol C
Obiektowe	Simula 67 C++ Java C#



Programowanie niskopoziomowe

```
mov     dx,AS_BUS
out     dx,al
mov     dx,port_c_8255
test    al,0001h
mov     dx,port_b_8255
test    al,01h
jz      no_ready_a
mov     al,abyte
out     dx,al
mov     cl,7
shl     al,cl
shr     al,cl
```



Real Programmers code in binary

- język programowania zbliżony do **języka maszynowego**

tak, by umożliwić maksymalnie efektywną kompilację i pełną kontrolę programisty nad postacią kodu wynikowego.

Programowanie proceduralne

```
open file();
read_file();
calculate();
```

```
program()
{
   ...
   open_file();
   ...
   read_file();
   ...
   calculate();
   ...
   display_result();
}
```

```
open_file()
{
   ...
}

read_file()
{
   ...
}

calculate()
{
   ...
}
```

Programowanie obiektowe

```
class Result { ... };
```

```
class Display { ... };
```

```
class File { ... };
```

```
class Data { ... };
```

```
class Calculator { ... };
```

```
program()
{
  Display display;
  Calculator calculator;
  Result result;
  result =
calculator.Calculate();
  display.Show(result);
```

```
Display::Show(Result& result)
{
}
```

```
File::Open()
{
}
```

```
Data::Read(File& file)
{
}
```

```
Result
Calculator::Calculate()
{
}
```

wykorzystuje się koncepcję **obiektu**,
zawierającego zarówno dane jak i przetwarzające je procedury.

Algorytm

**ściśle określony sposób postępowania (ciąg instrukcji),
prowadzący do rozwiązania pewnej klasy zadań.**

1. Warunek określoności:
nie dopuszcza żadnych niejasności w algorytmie.

2. Warunek kompletności:
uwzględnia wszystkie, również nietypowe sytuacje w trakcie obliczeń.

3. Warunek uniwersalności:
algorytm przeznaczony do rozwiązania całej klasy zadań.

4. Warunek efektywności:
po skończonej liczbie operacji otrzymujemy rozwiązanie.

5. Warunek ścisłości:
ciąg operacji arytmetyczno-logicznych realizowanych na zbiorze danych pierwotnych.

Architektura komputera – sposób organizacji elementów tworzących komputer.
Zazwyczaj pod pojęciem architektury rozumie się
organizację połączeń pomiędzy pamięcią, procesorem i urządzeniami wejścia-wyjścia.

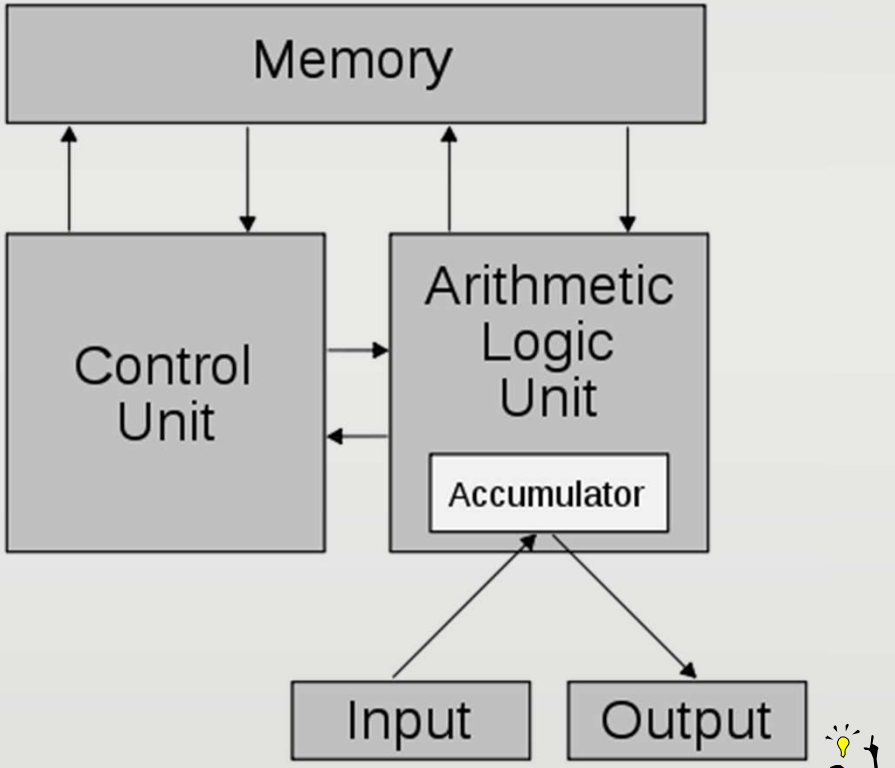
Architektura von Neumanna
- rodzaj architektury komputera, przedstawionej
po raz pierwszy w 1945 roku
przez **Johna von Neumanna** stworzonej wspólnie
z **Johnem W. Mauchly'ym**
i **Johnem Presper Eckertem**.

Polega na ścisłym podziale komputera na trzy podstawowe części:

* procesor (w ramach którego wydzielona bywa część sterująca oraz część arytmetyczno-logiczna)

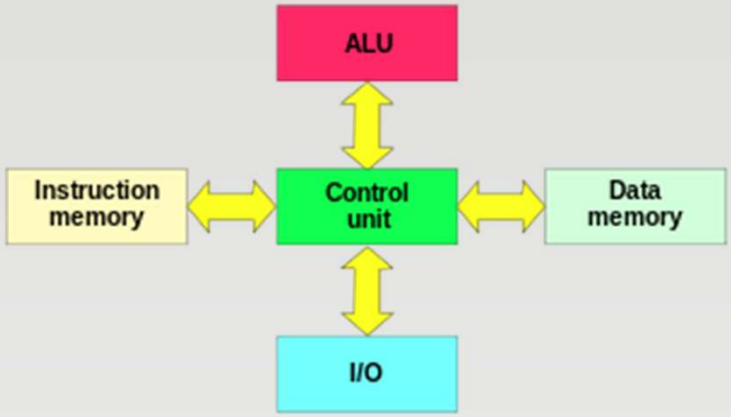
* pamięć komputera (zawierająca dane i sam program)

* urządzenia wejścia/wyjścia



Architektura harwardzka
– rodzaj architektury komputera.

Pamięć danych programu
jest oddzielona od pamięci rozkazów.



Podstawowa architektura komputerów zerowej generacji i początkowa komputerów pierwszej generacji.

Prostsza (w stosunku do architektury von Neumanna) budowa przekłada się na większą szybkość działania. Ten typ architektury jest często wykorzystywany w procesorach sygnałowych oraz przy dostępie procesora do pamięci cache.

Separacja pamięci danych od pamięci rozkazów sprawia, że architektura harwardzka jest obecnie powszechnie stosowana w mikrokomputerach jednoukładowych, w których dane programu są najczęściej zapisane w nieulotnej pamięci ROM (EPROM/EEPROM), natomiast dla danych tymczasowych wykorzystana jest pamięć RAM (wewnętrzna lub zewnętrzna).

Typy całkowite:

typ danych	wielkość w bajtach	wielkość w bitach	wartość minimalna	wartość maksymalna
char	1	8	-128	127
short	2	16	-32 768	32 767
int	(decyduje kompilator) np. 2	16	-32 768	32 767
long	4	32	-2 147 483 648	2 147 483 647

(UWAGA: każdy typ może mieć deklarację **'unsigned'**

np.

unsigned char

zajmuje również 1 Bajt, ale o wartościach ***od 0 do 255***)

Typy zmiennoprzecinkowe:

typ danych	wielkość w bajtach	wielkość w bitach	wartość minimalna	wartość maksymalna
float	4	32	3.4 E -38	3.4 E +38
double	8	64	1.7 E -308	1.7 E +308
long double	10	80	3.4 E -4932	3.4 E +4932

Typ logiczny:

bool wartość: false - gdy == 0
true - gdy != 0 (różne od 0)

Typy proste:

```
typ nazwa;                  np.  int a;          // deklaracja zmiennej typu całkowitego;
                                 double xn12;    // deklaracja zmiennej typu rzeczywistego;
```

Typy pochodne:

```
typ* nazwa;                 np.  int* pa;        // wskaźnik – deklaracja zmiennej typu wskaźnikowego;
typ& nazwa = zmienna;       np.  int& ra = a;    // referencja;
typ nazwa[ stała ];         np.  int ta[25];     // tablica;
typ nazwa( parametry );     np.  int fa( );      // funkcja;
```

Typy użytkownika:

struktura:

```
struct  Nazwa {
        // pola na dane;
}nazwa2;
```

klasa:

```
class  Nazwa {
        // pola na dane i funkcje;
}nazwa2;
```