Programowanie obiektowe w języku C++

Stanisław Gepner

sgepner@meil.pw.edu.pl

Regulamin przedmiotu i zasady zaliczenia

1. Przedmiot składa się z:
   a. 7 wykładów i 7 obowiązkowych ćwiczeń,
   b. kolokwium zaliczeniowego,
   c. obowiązkowego projektu zaliczeniowego.
2. Postępy studenta opisane są skalą punktową w zakresie 0-100 punktów. Punkty otrzymuje się za:
   a. Zaliczenie wejściówek przeprowadzonych na 5 ćwiczeniach (max 5pkt każda). Łącznie 25 pkt..
   b. Projekt zaliczeniowy, 35 punktów.
      i. W razie stwierdzenia niesamodzielności pracy (zapożyczenia kodu bez jego zrozumienia) student otrzymuje 0 punktów bez możliwości poprawy.
      ii. Ostateczny termin zaliczenia projektu ustala prowadzący ćwiczenia, jednak nie może być to termin późniejszy niż ostatni dzień semestru.
   c. Kolokwium zaliczeniowe oceniane do 40 punktów.
      i. Kolokwium przeprowadzane jest bez wykorzystania materiałów dodatkowych, tj. notatek, książek, pamięci zewnętrznych, itp..
      ii. W razie stwierdzenia niesamodzielności pracy (zapożyczenia kodu bez jego zrozumienia) student otrzymuje 0 punktów.
3. Zaliczenie przedmiotu jest możliwe jedynie w przypadku uzyskania minimalnej liczby punktów z każdej ocenianej części, tj.:
   a. 13 punktów z ćwiczeń,
   b. 18 z projektu,
   c. 21 punktów z kolokwium

   W przypadku nieuzyskania minimalnej liczby punktów student otrzymuje ocenę niedostateczną (2.0).
4. W przypadku uzyskania minimalnej liczby punktów, podanej w punkcie 3, ostateczna ocena obliczana jest w następujący sposób:
   a. 0-50: 2.0
   b. 51-60: 3.0
   c. 61-70: 3.5
   d. 71-80: 4.0
   e. 81-90: 4.5
   f. 91-100: 5.0

   Oceny wystawione w ostatnim dnu semestru są ostateczne. Nie będzie popraw w sesji ani semestrze letnim. Nie istnieją oceny „N”.
5. Obecność na ćwiczeniach jest obowiązkowa. Każda nieusprawiedliwiona i nieodrobiona nieobecność na ćwiczeniach powoduje odjęcie 3 punktów.
   a. W przypadku nieobecności w wyniku zdarzeń losowych odrobienie ćwiczeń możliwe jest po przedstawieniu zwolnienia.
   b. W przypadku nieobecności planowanej (np. wyjazd) odrobienie jest możliwe jedynie w przypadku wcześniejszej zgody prowadzącego ćwiczenia.
   c. Zasady, oraz termin odrobienia nieobecności ustala prowadzący ćwiczenia.
6. Prowadzący ćwiczenia może zorganizować poprawę kolokwium zaliczeniowego. Do kolokwium poprawkowego może przystąpić student:
   a. Obecny na kolokwium zaliczeniowym lub z usprawiedliwioną na nim nieobecnością. Obowiązują zasady jak w 5a,b.
   b. Z zaliczonym projektem zaliczeniowym. Brak pozytywnej oceny z projektu uniemożliwia poprawę kolokwium.
   c. Obecny na 5 z 7 wykładów.
7. Zapisy do grup laboratoryjnych odbywają się na pierwszym wykładzie. Kolejność zapisu wg. list rankingowych dostępnych w wirtualnym dziekanacie w dniu zapisu.
   a. Studenci niezapisanie na przedmiot nie mają prawa uczestniczyć w laboratoriach.
   b. Studenci zapisani, zapisani po terminie i nie wymienieni na listach dziekańskich zapisują się jako ostatni.

Zapisy na laboratoria

Proponowane terminy laboraorium:

- 2 x Środa 8:15-10:00
- 1 x Czwartek 10:15-12:00
- 2 x Czwartek 12:15-14:00

Literatura

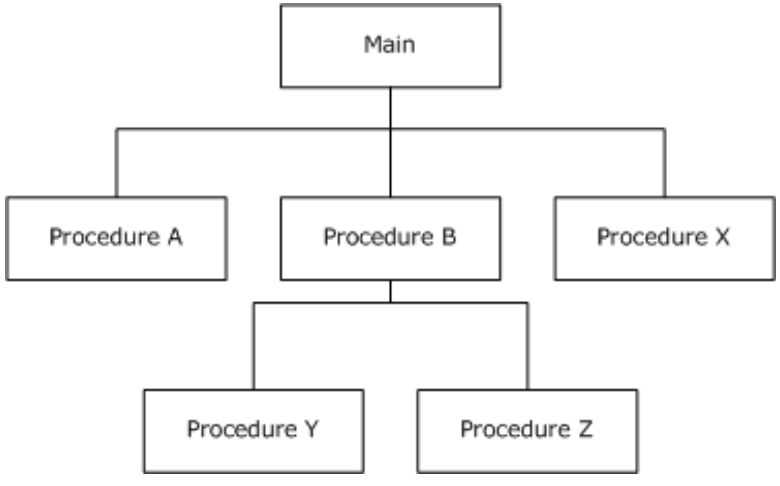
- Internet
- C++ programming tutorials
- Stack Overflow
- C++ reference - **cppreference.com**
- Kompilator online: http://cpp.sh/, **http://coliru.stacked-crooked.com/**
- Google, Bing, Duck Duck Go
- Visual Studio 2015, Community or Visual Studio 2013, GCC
- Jak pompki - trzeba ćwiczyć, nie czytać!

Obiektowo?

Proceduralnie np. C, Fortran

Czyli poprzez dzielenie zadania na procedury wykonujące określone operacje.

- Zmienne,
- Dane,
- Procedury
- Wywoływanie procedur, przekazywanie danych przez argumenty

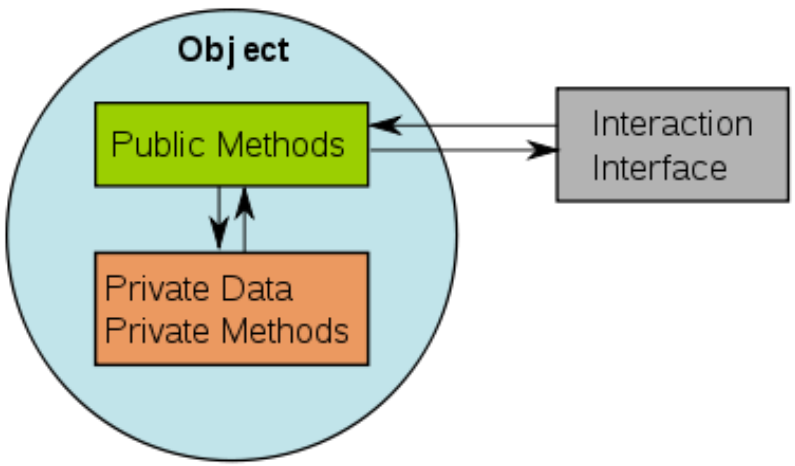


Public domain

Obiektowo?

Obiekt posiada zarówno dane jak i procedury operujące na nich.

- Obiekty mogą ze sobą współdziałać
- Program można składać z różnych obiektów - modularny
- Obiekty (powinny) są od siebie niezależne (do pewnego stopnia)
- Obiekt to instancja klasy. Posiada strukturę danych i metody jej manipulacji



Public domain

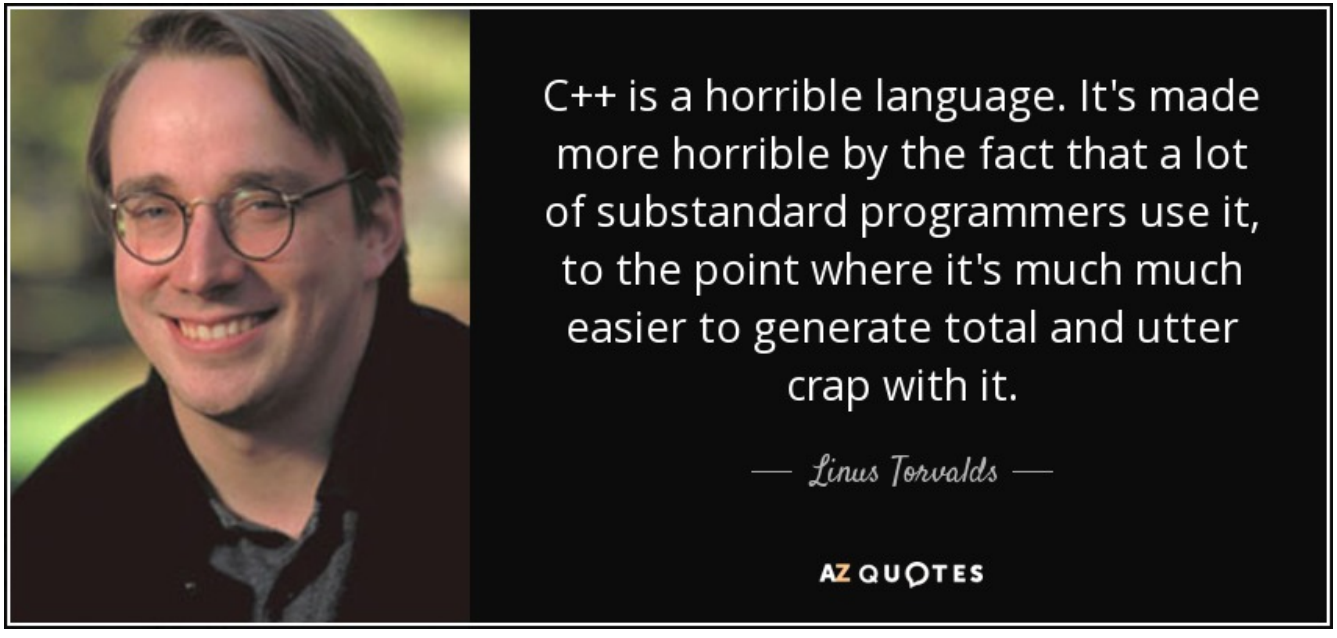
Obiektowo

- Abstrakcyjność
- Hermetyzacja (enkapsulacja)
- Dziedziczenie i polimorfizm

- Hierarchia klas
- Organizacja kodu
- Dostęp do różnych obiektów poprzez jednorodny interfejs
- Łatwość utrzymania i rozwoju

Linus Torvalds

Kernel Linuxa, git, bóstwo pomniejsze ...



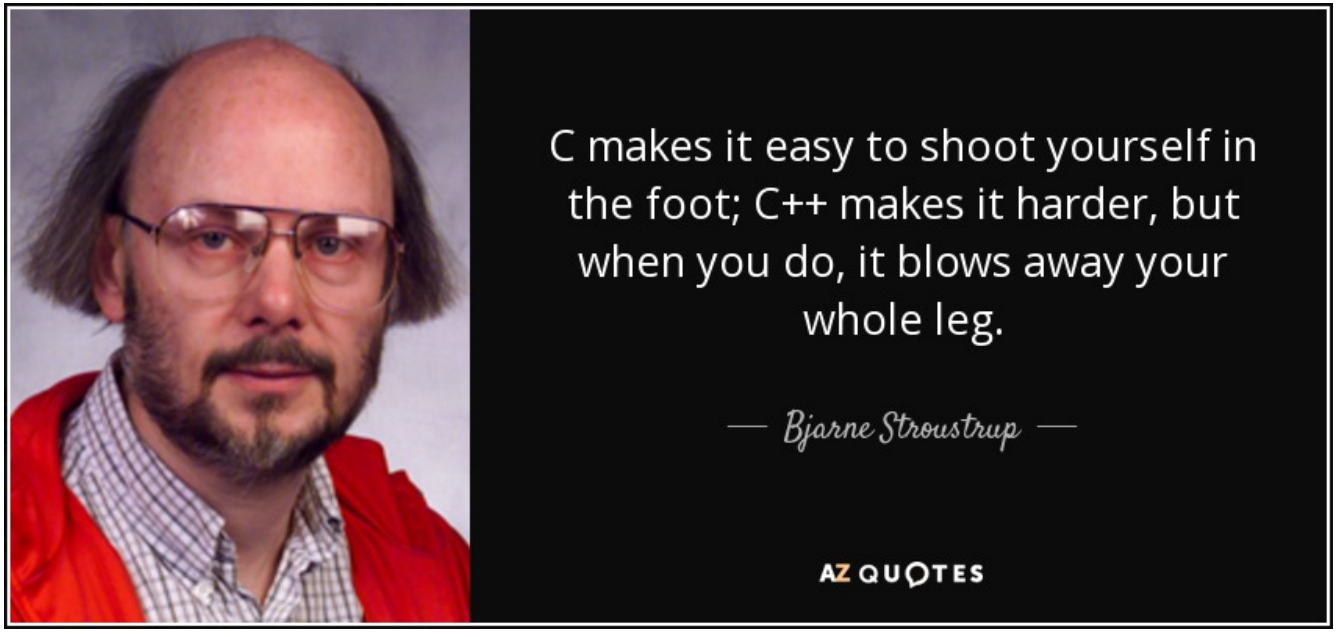
(...)In other words, the only way to do good, efficient, and system-level and portable C++ ends up to limit yourself to all the things that are basically available in C. And limiting your project to C means that people don't screw that up, and also means that you get a lot of programmers that do actually understand low-level issues and don't screw things up with any idiotic 'object model' crap.(...)

C language

- Dennis Ritchie - AT&T Bell Laboratories - 1972
- The C Programming Language - first specification - 1978
- 1989: ANSI C89, 1990: ISO C90
- 1999: C99 standard
- Still in use, and here to stay for a while
  - Wide range of applications. OS, microcontrollers, ATM systems ...
  - Efficiency and performance
  - Provides low level access
  - Influenced C++, Obj. C, C#, Java, ...

C++

- Bjarne Stroustrup - AT&T Bell Laboratories - 1979 'C with classes'
- C++ używany przez AT&T - 1983
- Pierwsza specyfikacja - 1985



C++

Język ogolnego przeznaczenia

- Lepszy C
- Umożliwia abstrakcję
- Pozwala na programowanie proceduralne i obiektowe
- 'Rozumie' C
- Kontrola typów - type safty (printf)

C++

```
#include <stdio.h>
int main()
{
    int a;
    scanf("%d", &a);
    printf("Hello! a=%d\n ", a);
}
```

```
#include <iostream>
int main()
{
    int a;
    std::cin >> a ;
    std::cout << "Hello! a=" << a
          << std::endl;
}
```

- Type resolution at compilation
- bez formatowania
- Wygodniej?

namespace

- namespace nazwa { deklaracje } - ustala przestrzeń nazw
- namespace A { namespace B { namespace C {
- nazwa1::nazwa2 - :: operator zasięgu - użyj nazwa2 z nazwa1
- using namespace nazwa; - ostrożnie!
- using nazwa1::nazwa2; - nazwa2 jest teraz w zasięgu
- namespace nazwa1 = nazwa2; - alias przestrzeni

namespace

```
#include <iostream>
namespace A{int fun();}

namespace B {namespace C {
     namespace D{
    int fun() {return 3;}
} } }

int main()
{
    int a = A::fun();
    int b = B::C::D::fun();
    std::cout << "Hello␣!!␣a=" <<
          a << "␣b=" << b << std
         ::endl;

    using namespace B::C;
    a=D::fun();
    std::cout << "Hello␣!!␣a=" <<
          a << "␣b=" << b << std
         ::endl;

    using D::fun;
    int c = fun();
    std::cout << "Hello␣!!␣a=" <<
          a << "␣b=" << b << "␣c=
         " << c << std::endl;
    namespace AA=B::C::D;
    namespace BB=A;
    a = AA::fun();
    b = BB::fun();
    std::cout << "Hello␣!!␣a=" <<
          a << "␣b=" << b << std
         ::endl;

    using namespace AA;
    using namespace BB;
    a = AA::fun();
    b = BB::fun();
    std::cout << "Hello␣!!␣a=" <<
          a << "␣b=" << b << std
         ::endl;

    a = fun();
    b = fun();
    std::cout << "Hello␣!!␣a=" <<
          a << "␣b=" << b << std
         ::endl;
}

int A::fun()
{
    return 1;
}
```

vector

```
#include <iostream>
#include <vector>

using namespace std;

int main(){
    vector<int> a(5);
    a[0] = 10;
    for(int i=1; i<a.size(); ++i){
        a[i] = 10*(i+1);
        cout << a[i] << endl;
    }
    cout << endl;
    a.pop_back();
    a.push_back(1);
    for(int i=0; i<a.size(); ++i)
        cout << a[i] << endl;
}
```

- Dynamiczna - resize(20)
- Ciągla w pamięci - dostęp przez [ ]

new & delete

```
#include <iostream>
#include <vector>

using namespace std;

struct D{
    int a;
};

int main(){
    int n=10;
    D * p = new D;
    D * tab = new D[50];
    tab[8].a = 1;
    cout << tab[8].a << endl;
    delete p;
    delete []tab;
}
```

- new zamiast malloc
- delete zamiast free
- kontrola typu, malloc zwracał *void

referencja &

```
#include <iostream>

using namespace std;

void fun1(int a){
  a=1;
}
void fun2(int *pa){
  *pa=2;
}
void fun3(int & a){
  a=3;
}

int main(){
  int a = 6;
  cout << a << endl;
  fun1(a);
  cout << a << endl;
  fun2(&a);
  cout << a << endl;
  fun3(a);
  cout << a << endl;
  int & b; // this will not
       compile
}
```

- Przypomina wskaźnik
- Można przypisać tylko raz
- Nie może istnieć niezainicjalizowana
- Dalej jak zmienna

const

```
#include <iostream>

using namespace std;

#define sin cos
#define true false
#define fabs abs
#define PI 3.141592
const double pi=3.141592

pi = 3; //compiler error
```

- #define to zło
- const jest sprawdzane w czasie kompilacji

Klasa

```
struct A{
  int a;
};
class B{
  public:
    int a;
  private:
    A b;
}
class C{
  public:
    A a;
    B b;
    int c[5];
};

int main(){
  C c;
  c.a.a=0;
  c.b.a=3;
  c.c[2]=9;
  c.b.b.z=2; //Will not work
}
```

- Podobna do struct ale z hermetyzacją
- i kilkoma innymi dodatkami
- Atrybutami mogą być typy proste, inne klasy, kolekcje, wskaźniki i referencje
- public, private i protected
- domyślnie wszystko private
- dostępne przez operator .
- Przykład ...

Funkcje w klasie? Czyli metody!

```
#include <iostream>

using namespace std;

class person{
  public:
    void setAge(int a){ mAge=a; }
    int getAge(){ return mAge; }
    void printtS(){cout << mS <<
         endl;}
    void calcS();
  private:
    int mAge;
    int mS;
};

void person::calcS(){
  mS = 2 * mAge;
}

int main(){
  person p;
  p.setAge(3);
  p.calcS();
  cout << p.getAge() << endl;
  p.printtS();
}
```

- Mogą być w ciele
- albo poza - z deklaracją
- Metoda ma dostęp do wszystkich atrybutów klasy
- Ukrywaj atrybuty, wystawiaj interfejs

Konstruktor i destruktor

```
#include <iostream>
#include <stdlib.h>

using namespace std;

class collection{
public:
  collection(){size=0; tab=NULL;}
  collection(int s) : size(s) {
       allocate();}
  collection(collection& c){
    size=getSize();
    //tab = ????
    }
  ~collection(){
    cout << "The cleaning service
         !" << endl;
    delete []tab;
  }
  void setSize(int a){ size=a; }
  int getSize(){ return size; }
  void allocate();
  int& rTab(int i)
  { return tab[i];}
private:
  int size;
  int * tab;
};
```

```
void collection::allocate()
{
  tab = new int[size];
}

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

- Metody specjalne, tworzone domyslnie
- Definicja w lub poza ciałem
- Destruktor wywoływany przed zwolnieniem zasobów