

Programowanie obiektowe w języku C++

Stanisław Gepner

sgepner@meil.pw.edu.pl

Literatura

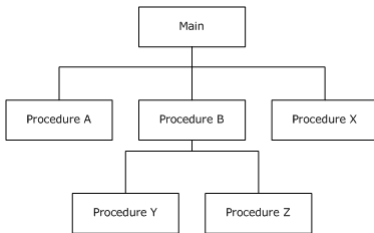
- Internet
- C++ programming tutorials
- Stack Overflow
- C++ reference - [cppreference.com](http://cplusplus.com)
- Kompilator online: <http://cpp.sh/>,
<http://coliru.stacked-crooked.com/>
- Google, Bing, Duck Duck Go
- Visual Studio 2015, Community or Visual Studio 2013, GCC
- Jak pompki - trzeba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

Proceduralnie, czyli jak?

np. C, Fortran

Czyli poprzez dzielenie zadania na procedury wykonujące określone operacje i przekazujące pomiędzy sobą przetwarzane dane.

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



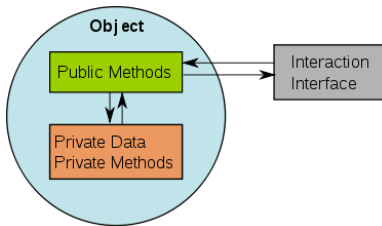
Public domain

Obiektowo, czyli jak?

np. C++, Java, C# ...

Zagadnienie dzielimy na obiekty łączące w sobie zarówno dane, jak i operujące na nich procedury (metody).

- Obiekty mogą ze sobą współdziałać
- Program można składać z różnych obiektów - zwiększa się modularność
- Obiekty (powinny) są od siebie niezależne (do pewnego stopnia)
- Obiekt to instancja klasy. Klasy są przepisem na zawartą w obiekcie strukturę danych i metody manipulacji



Obiektość:

Pozwala na wprowadzenie następujących koncepcji:

- Abstrakcyjność, czyli opis problemu na wyższym, oderwanym od poszczególnych zadań poziomie.
- Hermetyzacja (enkapsulacja), czyli kontrolę dostępu.
- Polimorfizm, czyli wielopostaciowość.

A to pozwala na:

- Hierarchię klas.
- Lepszą (być może) organizację kodu.
- Dostęp do różnych obiektów poprzez jednorodny interfejs.
- Potencjalną łatwość utrzymania, rozwoju i współdzielenia kodu.

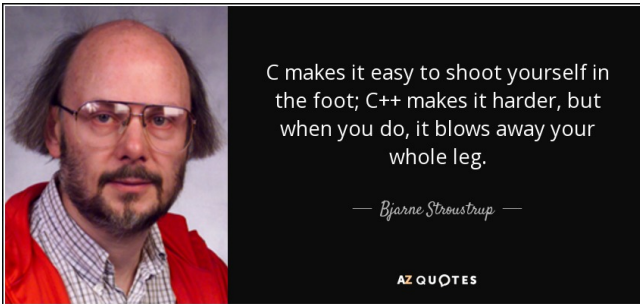
Trochę historii

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, ...

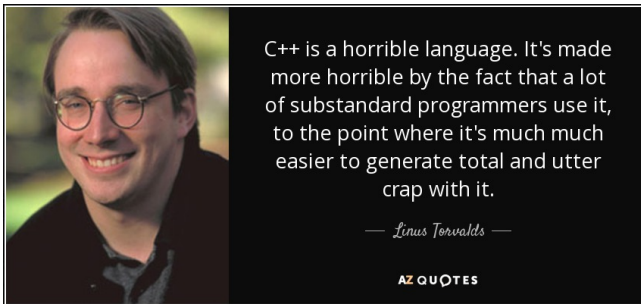
Trochę historii C++

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



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++

Język ogólnego przeznaczenia

- Lepszy C
- Umożliwia abstrakcję
- Pozwala na programowanie zarówno proceduralne jak i obiektowe
- 'Rozumie' C - pozwala na kompilację kodu napisanego w C
- Rozbudowuje kontrolę typów - type safety, w stosunku do C (np. funkcja printf nie kontroluje typu danych)

C vs 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;
}
```

- Zarówno scanf i printf muszą zostać poinformowane o typie poprzez formatowanie: %d, a konkretna implementacja wybierana jest w czasie działania programu.
- Implementacje cin i cout są wybierane na podstawie typu zmiennej, już w czasie kompilacji.
- Być może jest wygodniej :)

C++ - przestrzeń nazw - namespace

Pozwalają na organizację kodu poprzez zamykanie kodu w

- 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)`
- Ciągła w pamięci - dostęp przez []

new & delete

zamiast malloc i free

```
#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
- Modyfikatory dostępu: *public*, *private* i *protected*
- domyślnie wszystko *private*
- dostępne przez operator `.` lub `->`
- 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();
}
```

- Definicja w ciele klasy (pomiędzy {})
- albo poza - z deklaracją w ciele
- Metoda ma dostęp do wszystkich atrybutów klasy
- Zasada: 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
- Destraktor wywoływany przed zwolnieniem zasobów