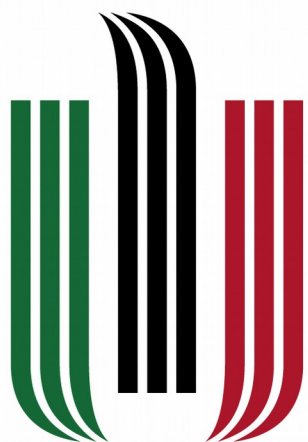
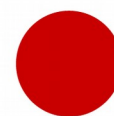




AGH



Podstawy Informatyki – *struktury, rzutowanie, dynamiczna alokacja pamięci, obiekty*

9.11.218

dr inż. Michał Rad



Plan wykładu:

- 1.Struktury
- 2.Rzutowanie, wskaźniki
- 3.Dynamiczna alokacja pamięci
- 4.Objekty

Struktury

- » Struktura to typ zmiennej który może przechowywać „w jednym miejscu” wiele wartości różnych typów.
- » Strukturę definiuje się samodzielnie i nadaje jej nazwę, która funkcjonuje potem dość podobnie do nazw typów podstawowych



Przykład

```
struct nazwa_struktury
{
    typ_pola1 nazwa_pola1;
    typ_pola2 nazwa_pola2;
    //...
}
```

Struktury

- » Struktury można wskazywać wskaźnikami oraz tworzyć tablice, które przechowują struktury

Przykład *(struct_w4a)*

```
struct nazwa_struktury
{
    typ_pola1 nazwa_pola1;
    typ_pola2 nazwa_pola2;
    //...
}

nazwa_struktury *wsk_do_struct;
nazwa_struktury tabl_el_str[5];
```

Dynamiczna alokacja pamięci

- » Dość często w chwili uruchomienia programu nie można określić z góry ilości potrzebnej pamięci albo jest to nieefektywne
- » W takich przypadkach należy korzystać z możliwości alokacji (zarezerwowania) pamięci w sposób dynamiczny – czyli na żądanie, w chwili gdy jest potrzebna.

Dynamiczna alokacja pamięci

» Do obsługi dynamicznej alokacji pamięci w języku C++ służą operatory **new** i **delete**

Przykład *(new_w4)*



```
int *wsk_i, *wsk_t;  
  
wsk_i = new int;  
wsk_t = new int[100];
```

Dealokacja pamięci



- » Pamięć przydzieloną operatorem **new** należy zawsze zwalniać gdy nie jest już potrzebna.
- » Służy do tego operator **delete** oraz **delete[]** (do tablic)
- » W języku C++ zarządzanie pamięcią jest zadaniem programisty

Przykład „wycieku pamięci”

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

int main()
{
    int *wsk_i;
    for(int k=1;k<10;k++)
    {
        wsk_i=new int;
        *wsk_i = k*k;
        cout<<"wsk: "<<wsk_i<<" wart:"<<*wsk_i<<endl;
    }
}
```

Rzutowanie

- » Mówiąc w pewnym uproszczeniu, zmiana typu zmiennej, a dokładniej zmiana typu wartości to rzutowanie
- » Potrzeba rzutowania w programach C++ zachodzi dość często
- » Jest co najmniej kilka sposobów rzutowania

Przykład (cast_w4)

» Rzutowanie typów podstawowych:

```
int i=11;  
float f;  
  
f= (float) i;  
f=float (i) ;
```

```
int n=1, k=2;  
  
cout<<n/k<<endl;  
cout<<float (n) / float (k) <<endl;
```

Przykład

» Rzutowanie w wywołaniu funkcji:

```
float funkcja(float x,float y)
{ ... }
int a=1,b=2;

funkcja((float) a,(float) b);
```

Przykład (void_ptr_w4.cpp)

- » Wskaźnik typu `*void`
- » Można powiedzieć że jest to wskaźnik uniwersalny
- » Używanie go jednak wiąże się z rzutowaniem



Programowanie obiektowe

- wstęp

```
int main()
{
    Osoba os1;
    os1={"Stefan","Wielki",24,610112113};

    os1.przedstaw_sie();
}
```

Programowanie obiektowe

- wstęp

```
struct Osoba
{
    string imie;
    string nazwisko;
    int wiek;
    long int numer;
    void przedstaw_sie()
    {
        cout<<imie<<" "<<nazwisko<<" ,wiek:"
            <<wiek<<endl;
    }
};
```



- » Nasza struktura zawiera teraz oprócz pól (danych) również funkcję
- » Takie połączenie funkcji i pól w jednej konstrukcji tworzy nam obiekt
- » Innymi słowy: obiekt posiada swoje cechy (pola, składowe, *members*) oraz posiada swoje metody (funkcje składowe, *methods*)

Programowanie obiektowe

- » Programowanie polegające na modelowaniu problemu za pomocą obiektów i ich wzajemnych interakcji nazywane jest programowaniem obiektowym, lub programowaniem zorientowanym obiektowo
- » W C++ obiekty można deklarować ze słowem kluczowym struct lub class



- » Bardzo często deklaracje klas umieszcza się w osobnym pliku, dla czytelności kodu szczególnie dla rozbudowanych klas
- » Definicje poszczególnych funkcji składowych można umieścić w pliku w którym znajduje się program główny
- » Aby określić której funkcji dotyczy definicja stosuje się operator zakresu ::

Przykład cd..



```
struct Osoba
{
    string imie;
    string nazwisko;           //
    int wiek;                  // deklaracja klasy
    long int numer;           //
    void przedstaw_sie();
    void dodaj_rok(); // dodajemy kolejną funkcję
};
```



```
void Osoba::dodaj_rok() // definicja
{
    wiek++;
};
```

```
int main()
{
    Osoba os1;
    os1++; //a może tak?
    os1.przedstaw_sie();
};
```



```
struct Osoba
{
    string imie;
    string nazwisko;           //
    int wiek;                  // deklaracja klasy
    long int numer;           //
    void przedstaw_sie();
    void operator++();
};
```



» Przykład z wektorami:

- Problem: zdefiniować klasę reprezentującą wektor w przestrzeni i zdefiniować operator dodawania wektorów.

Konstruktory

- » Do inicjalizacji obiektu, jego „konstrukcji” służy specjalna funkcja, która wywoływana jest automatycznie przy tworzeniu obiektu
- » Jeżeli programista nie zdefiniuje tej funkcji kompilator wygeneruje tę funkcję sam.

Przykład

```
struct Wektor
{
    float x,y,z;
    Wektor operator+(Wektor &v2);
    void wypisz();
    Wektor(); //konstruktor bez parametrów;
};
```

Konstruktory

- » Obiekt musi mieć co najmniej jeden konstruktor
- » Jeżeli programista zdefiniował swój konstruktor, kompilator nie generuje automatycznie konstruktora bezparametrycznego
- » Konstruktor nie zwraca żadnej wartości – nawet typ void nie może być użyty

Podsumowanie



» Quiz!