Laboratorium Podstaw Informatyki

1. Przebieg ćwiczenia laboratoryjnego

Konstruktor

Konstruktor jest specyficzna funkcja, która jest wywoływana zawsze gdy tworzony jest obiekt. Jeśli programista nie utworzy konstruktora dla klasy, kompilator automatycznie utworzy konstruktor, który nic nie będzie robił. Konstruktor nie pojawi się nigdzie w kodzie, jednak będzie on istniał w skompilowanej wersji programu i będzie wywoływany za każdym razem, gdy będzie tworzony obiekt klasy. Jeśli chcemy zmienić domyślne własności konstruktora jaki jest tworzony przez kompilator C++ wystarczy, że utworzymy własny konstruktor dla klasy.

```
classMyClass
{
public:
MyClass();            // To jest deklaracja konstruktora bez parametrów
MyClass(int a);       // To jest deklaracja konstruktora z parametrami
};
MyClass::MyClass()   // To jest definicja konstruktora bez parametrów
{
      // Tu inicjujemy wartości zmiennych klasy
}
MyClass::MyClass(int a)    // To jest definicja konstruktora z
parametrami
{
      // Tu inicjujemy wartości zmiennych klasy
}
```

Pierwsze co rzuca się w oczy w przypadku konstruktora, to brak zwracanego typu danych. Druga istotna własność konstruktora to jego nazwa. Konstruktor musi nazywać się tak samo jak nazwa klasy. Konstruktorom można przekazywać parametry tak samo jak funkcjom. C++ umożliwia również tworzenie kilku konstruktorów dla jednej klasy (musza

się one jednak różnic parametrami wejściowymi tak jak w przypadku funkcji).

```
class NazwaKlasy
{
public:                          //kwantyfikator sekcji publicznej
        double liczba;           //prawo dostępu: publiczne
        char tablica[20];        //prawo dostępu: publiczne
private:                         //kwantyfikator sekcji prywatnej
        int abc;                 //prawo dostępu: prywatne
        char znak;               //prawo dostępu: prywatne
        string napis;            //prawo dostępu: prywatne
 };
int main()
 {
        NazwaKlasy NazwaZmiennej;
        return(0);
}
```

Destruktor

Często jest tak, ze podczas życia klasy rezerwujemy pamięć, którą chcielibyśmy zwalniać zawsze przed usunięciem klasy. Pierwszym wariantem jest pamiętanie o wywołaniu funkcji, która będzie za to odpowiedzialna. Takie podejście jest jednak ryzykowne, ponieważ bardzo łatwo zapomnieć o wywoływaniu funkcji, która będzie zwalniała ewentualną zarezerwowaną dynamicznie pamięć. Lepszym rozwiązaniem tego problemu jest wykorzystanie destruktorów. Destruktor jest specjalna funkcja, która jest wywoływana zawsze tuz przed zniszczeniem(usunięciem) klasy z pamięci. Budowa destruktora wygląda następująco:

```
classMyClass
{
public:
~MyClass();              //To jest deklaracja destruktora
};
MyClass::~MyClass() // To jest definicja destruktora
{
        // Tu wykonujemy wszystkie operacje jakie maja się wykonać
        automatycznie tuz przed zwolnieniem pamięci zajmowanej
        przez obiekt klasy
}
```

Destruktor, tak samo jak konstruktor nie posiada zwracanego typu. Drugą ważną cechą jest to, że destruktor musi być zawsze bezparametrowy. Trzecią, a zarazem ostatnią ważną cechą jest możliwość zdefiniowania tylko i wyłącznie jednego destruktora dla danej klasy. Przykład działania konstruktora i destruktora został pokazany na listingu:

```
#include<iostream>
#include<cstdio>
using namespace std;
class MyClass
{
public:
MyClass();
~MyClass();
};
int main()
{
cout<< "TworzeobiektklasyMyClass" <<endl;
MyClass f1;
cout<< "Wchodzę do bloku {" <<endl;
{
MyClass f2;
}
cout<< "Wyszedłem z bloku }" <<endl;
cout<< "Koncze program" <<endl;
return 0;
}
MyClass::MyClass()
{
cout<< " =>Konstruktorwywolany!" <<endl;
}
MyClass::~MyClass()
{
cout<< " =>Destruktorwywolany!" <<endl;
}
```

Składniki statyczne klasy

Zmienna statyczna jest to zmienna która istnieje w pamięci programu od jego rozpoczęcia do zakończenia. W odróżnieniu od zwykłych zmiennych lokalnych nie jest usuwana z pamięci po zakończeniu lokalnego bloku programu. Przykład wykorzystania zmiennej statycznej.

```
void foo(void)
{
int a = 0;
static int b = 0;
a++;
b++;
cout<< "a = " << a << " b = " << b <<endl;
}
```

Wartość zmiennej b po każdym wywołaniu funkcji foo() jest zapamiętywana, tzn. zmienna ta nie jest usuwana z pamięci. Dodatkowo zmienna statyczna zawsze inicjalizowana jest wartością 0 (zero), podczas gdy zwykła zmienna nie jest. Można to sprawdzić poprzez usuniecie inicjalizacji zmiennych w listingu. Można zatem powiedzieć ze zmienne statyczne

zachowują się podobnie jak zmienne globalne, przy czym mogą mieć inny zakres ważności (np. lokalnie w funkcji).

Analogicznie do zmiennych statycznych, składniki statyczne klasy są to składniki, które są wspólne dla całej klasy (globalne dla całej klasy). Zwykłe pola składowe są definiowane dla każdego obiektu danej klasy, natomiast składniki statyczne są definiowane dla całej klasy. Deklaracja oraz definicja składnika statycznego klasy przedstawiona została poniżej.

```
classMyClass
{
public:
staticint licznik; //deklaracja składnika statycznego
};
intMyClass::licznik; //definicja składnika statycznego
```

Definicja składnika statycznego następuje poza deklaracja klasy, ale w pliku źródłowym (*.cpp) w którym widoczna jest deklaracja klasy. Najczęściej jest to plik źródłowy w którym zawarte są definicje wszystkich metod klasy. Klasa może również zawierać statyczne funkcje składowe. Funkcje takie mogą operować tylko i wyłącznie na składnikach statycznych klasy. Ważne jest aby zapamiętać, ze do składników statycznych (pól oraz metod) można odnieść się nawet wtedy kiedy nie istnieje żądne obiekt klasy, poprzez użycie nazwy klasy i operatora zakresu np. MyClass::licznik =4;.

Zadanie do realizacji:

1. Utwórz klasę osoba, która będzie przechowywała elementarne dane osobowe (imię, nazwisko, data urodzenia, płeć, miasto) oraz pole identyfikatora typu int o nazwie id.
2. Identyfikator danej osoby powinien być unikalny dla każdego obiektu i automatycznie nadawany w momencie tworzenia obiektu klasy osoba. Ponadto identyfikator powinien być unikalny bez względu na ilość uruchomień programu. W jaki sposób to zrealizujesz?
3. Utwórz funkcje składowa klasy, która zwróci numer ostatnio nadanego identyfikatora w klasie osoba. Funkcja powinna być możliwa do wywołania nawet gdy nie istnieje żaden obiekt klasy osoba.
4. Zdefiniuj wewnątrz klasy osoba konstruktory które umożliwią tworzenie obiektów tej klasy na okoliczność podania różnych danych nowo tworzonej osoby.
5. Zdefiniuj w klasie osoba funkcję pozwalającą na przedstawienie obiektów z danej klasy..
6. Dodaj do klasy osoba pole które zwróci ilość aktualnie istniejących obiektów klasy osoba.