Programowanie w C++

Kurs (średnio)zaawansowany

Zajęcia numer 19

Rafał Berdyga
rberdyga@gmail.com

Plan na dzisiaj

- Podstawy obsługi rozproszonego systemu kontroli wersji: Git
- Metoda specjalna - destruktor
- Dynamiczna alokacja pamięci – przypomnienie + ćwiczenie
- Zadanie przekrojowe wykorzystujące podstawowe techniki obiektowe

Metoda specjalna - destruktor

Metoda wywoływana automatycznie w momencie niszczenia obiektu.

- służy do wykonania niezbędnych czynności przed likwidacją obiektu,
- nazwa destruktora ma postać: nazwa klasy poprzedzony znakiem ~
- w przypadku tej metody nie podaje się typu zwracanego wyniku;
- umieszcza się ją w części publicznej definicji klasy,
- w klasie istnieje tylko jeden destruktor.

Destruktor - przykład użycia

Uzupełniamy przykład ze Studentem

Plik student.h – dodano:

```
~Student();
```

Plik student.cpp – dodano:

```
Student::~Student()
{
   cout << ”Destructor” << endl;
}
```

Przykład Gaduła

Automatyczna generacja konstruktorów i destruktorów

- Jeżeli w definicji klasy nie występują konstruktory to kompilator samodzielnie wygeneruje konstruktor domniemany (bezparametrowy).
- Jeżeli w definicji klasy brak konstruktora kopiującego (mogą istnieć inne wersje) to kompilator również go wygeneruje automatycznie.
- Jeżeli w definicji klasy brakuje destruktora to również zostanie on wygenerowany automatycznie

Dla klas które nie zawierają zasobów dynamicznie alokowanych, automatycznie wygenerowane metody konstruktorów i destruktora są wystarczające.

Automatyczna generacja - niebezpieczeństwo

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

class SomeClass
{
    char data;

public:
    SomeClass (char a) { data = a; }

};

int main()
{
   SomeClass object1;
   return 0;
}
```

Kompilator zaprotestuje

Candidate constructor not viable: requires single argument 'a', but no arguments were provided

Tablice dynamiczne

W języku C++ do alokowania pamięci służy operator **new**, a do zwalniania - **delete**.

```
int *tab;
tab = new int[10];

delete [] tab;
```

Alokujemy miejsce w pamięci na **10 elementową tablicę** typu **int**, a kiedy już nam nie będzie ona potrzebna – pamiętajmy aby to miejsce zwolnić!

Tablice dynamiczne cd.

Zapis w dwóch linijkach:

```
int *tab = new int[10];

delete [] tab;
```

Z tablic dynamicznych korzystamy tak, jak ze zwykłych tablic!

```
int main()
{
   int n;
   cin >> n;

   float *tab = new float[n];

//jakies operacje na tablicy

   delete [] tab;
   return 0;
}
```

Tablice dynamiczne pozwalają na tworzenie tablicy, o wymiarze zdefiniowanym przez użytkownika (**z góry nie musimy określać jak duża będzie tablica**).

Dynamiczne tworzenie obiektów

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

class SomeClass
{
public:
  SomeClass() { cout<<"1"<<endl; }
  SomeClass(char a){ cout<<"2"<<endl; }
  SomeClass(const SomeClass& o){cout<<"3"<<endl;}
};

int main()
{
  SomeClass *p1= new SomeClass;
  SomeClass *p2= new SomeClass('a');
  SomeClass *p3= new SomeClass(*p2);
  SomeClass *p4= new SomeClass [2];

  delete p1;
  delete p2;
  delete p3;
  delete [] p4;
}
```

Wynik:

1
2
3
1
1

Ćwiczenie 2

Korzystając z kodu z ćwiczenia 1 w funkcji *main()* umieść poniższy kod testowy:

```
Figura *f4 = new Figura();
Figura *f5 = new Figura(5,3);
Figura f6(*f4);

delete f4;
delete f5;
```

Jak „poszerzyć” tablicę dynamiczną?

Załóżmy, że mamy tablicę dynamiczną 10 elementową. Chcemy wrzucić do niej element nr 11. Co robimy?

1. Tworzymy tablicę pomocniczą (również dynamicznie),
2. Przepisujemy (w pętli) wartości z tablicy pierwotnej do tablicy pomocniczej,
3. Dodajemy element nr 11 na koniec tablicy pomocniczej,
4. Zamieniamy miejscami wskaźniki, tj. wskaźnikowi, który wskazywał na starą tablicę przypisujemy wartość wskaźnika tablicy pomocniczej, np.

```
int *stara = new int[10];
int *pom = new int[11];

// punkt 2. oraz 3.

stara = pom;
```

5. Zwalniamy miejsce w pamięci zajmowane przez tablicę pomocniczą.

Zadanie
18.1
czas: 1h – 1,5h