C++/Konstruktor i destruktor

Teoria

Wstęp

Pisząc klasy każdy kiedyś dotrze do momentu, w którym będzie odczuwał potrzebę napisania funkcji wykonującej jakieś niezbędne instrukcje na początku lub na końcu istnienia obiektu. W takim momencie programista powinien sięgnąć po dwa niezwykle przydatne narzędzia: **konstruktory** i **destruktory**.

Konstruktor

Konstruktor jest to funkcja w klasie, wywoływana w trakcie tworzenia każdego obiektu danej klasy. Funkcja może stać się konstruktorem gdy spełni poniższe warunki

- Ma identyczną nazwę jak nazwa klasy
- Nie zwraca żadnej wartości (nawet **void**)

Należy dodać że każda klasa ma swój konstruktor. Nawet jeżeli nie zadeklarujemy go jawnie zrobi to za nas kompilator (stworzy wtedy konstruktor bezparametrowy i pusty).

Mamy na przykład klasę *Miesiac*. Chcielibyśmy, aby każdy obiekt tej klasy tuż po utworzeniu wygenerował tablicę z nazwami dni tygodnia w zależności od miesiąca i roku. A może dało by się to zrobić *w trakcie* tworzenia klasy?
Przyjrzyj się poniższej klasie, oraz funkcji konstruktora:

```
class Miesiac
{
     public:
          int dni[31];
          int liczbaDni;
          string nazwa;
          Miesiac();//deklaracja konstruktora
};

Miesiac::Miesiac()//definicja konstruktora
{
     // instrukcje tworzące
}
```

Konstruktor może też przyjmować argumenty. Jak?
To zależy od sposobu w jaki tworzymy obiekt:

- jako obiekt

```
MojaKlasa obiekt(argumenty);
```

- jako wskaźnik do obiektu:

```
MojaKlasa* wsk = new MojaKlasa(argumenty);
```

Teraz powyższa klasa miesiąca może być stworzona z uwzględnieniem numeru miesiąca i roku:

```
class Miesiac
{
     public:
          int dni[31];
          int liczbaDni;
          string nazwa;
          Miesiac(int numer,int rok);
};

Miesiac::Miesiac(int numer,int rok)
{
     /* instrukcje tworzące */
}
```

Aby utworzyć nowy obiekt tej klasy trzeba będzie napisać:

```
Miesiac styczen2000(1,2000);
```

lub jako wskaźnik do obiektu:

```
Miesiac* styczen2000 = new Miesiac(1,2000);
```

otrzymawszy w ten sposób kalendarz na styczeń.

Najczęstszą funkcją konstruktora jest inicjalizacja obiektu oraz alokacja pamięci dla dodatkowych zmiennych (w tym celu lepiej jest użyć instrukcji inicjujących, które poznasz już za chwilę).

Instrukcje inicjujące

Instrukcje inicjujące to instrukcje konstruktora spełniające specyficzne zadanie. Mianowicie mogą one zostać wywołane przez kompilator zaraz po utworzeniu klasy. Służą do inicjowania pól klasy, w tym stałych i referencji.

Jeśli nie zaimplementujemy instrukcji inicjujących, niczego nie będą one robiły.

Jeżeli chcemy zaimplementować **instrukcje inicjujące**, musimy po liście argumentów konstruktora, użyć dwukropka, podać nazwę pola, które chcemy zainicjować i jego wartość ujętą w nawiasy okrągłe.

```
Rok()
: miesiace(new Miesiac[12])
, liczbaDni(7)
/*
zamiast średników stosuje się przecinki
przy ostatniej instrukcji przecinka nie stosuje się
*/
{}
```

Działa to podobnie jak użycie inicjowania w konstruktorze, jednak w przypadku instrukcji inicjujących pola będą zainicjowane w trakcie tworzenia klasy, a nie po utworzeniu jej obiektu.

Konstruktor kopiujący

Konstruktor kopiujący to konstruktor spełniający specyficzne zadanie. Mianowicie może on zostać wywoływany przez kompilator niejawnie jeżeli zachodzi potrzeba stworzenia drugiej instancji obiektu (np. podczas przekazywania obiektu do funkcji przez wartość).

Jeżeli nie zaimplementujemy konstruktora kopiującego, kompilator zrobi to automatycznie. Konstruktor taki będzie po prostu tworzył drugą instancję wszystkich pól obiektu. Możemy go jawnie wywołać np. tak:

```
Miesiac miesiac(12,2005);
Miesiac kopia(miesiac); //tu zostanie wywołany konstruktor kopiujący
/* obiekt kopia będzie miał taką samą zawartość jak obiekt miesiąc */
```

Jeżeli chcemy sami zaimplementować **konstruktor kopiujący** musimy zadeklarować go jako **konstruktor** o jednym parametrze będącym referencją na obiekt tej samej klasy.

```
class Miesiac
{
     public:
          int numer;
          int rok;
          Miesiac(const Miesiac &miesiac)
          {
               numer=miesiac.numer;
               rok=miesiac.rok;
          }
};
```

Porada

Jeżeli dokonujemy w instrukcjach inicjujących alokacji pamięci, np:

```
class Rok
{
      protected:
            Miesiac *miesiace;
      public:
            Rok()
            : miesiace(new Miesiac[12]) {}
            virtual ~Rok() {delete[] miesiace;}
};
```

to nie możemy się zdać na **konstruktor kopiujący** tworzony niejawnie. Jeżeli tak zrobimy, to w obiekcie stworzonym przez **konstruktor kopiujący** pole *miesiace* będzie wskazywać na ten sam fragment pamięci, co w obiekcie wzorcowym. Jeżeli nie jest to zamierzony efekt (a zwykle nie jest) musimy sami zaimplementować konstruktor kopiujący, np. tak:

```
class Rok
{
      protected:
            Miesiac *miesiace;
      public:
            Rok() : miesiace(new Miesiac[12]) {}
            Rok(const Rok &rok)
            {
                  //musimy sami zaalokować pamięć na pole ''miesiace''
                  miesiace=new Miesiac[12];
                  //oraz przypisać temu polu odpowiednie wartości
                  for (int i=0; i<12; ++i)
                        miesiace[i]=Miesiac(rok.miesiace[i]);
            }
            virtual ~Rok() {delete[] miesiace;}
};
```

Destruktor

Destruktor jest natomiast funkcją, którą wykonuje się w celu zwolnienia pamięci przydzielonej dodatkowym obiektom lub innych zasobów.

Zasady "przemiany" zwykłej funkcji do destruktora, są podobne do tych tyczących się konstruktora. Jedyna zmiana tyczy się nazwy funkcji: Musi się ona zaczynać od znaku **tyldy** - **~**.

```
class MojaKlasa
{
      MojaKlasa();//to oczywiście jest konstruktor
      ~MojaKlasa();//a to - destruktor
};
```

Najczęstszą funkcją destruktora jest zwolnienie pamięci (zwykle poprzez zniszczenie wszystkich pól używanych przez ten obiekt).

Porada

Należy pamiętać, że jeżeli zamierzamy implementować dziedziczenie po klasie dla której piszemy destruktor to powinniśmy stworzyć destruktor **wirtualny**!

```
class MojaKlasa
{
      MojaKlasa();
      virtual ~MojaKlasa();//to jest destruktor wirtualny
};
```

Początkujący programiści często o tym zapominają, doprowadzając w ten sposób czasami do tzw. wycieków pamięci. Dobrą praktyką jest tworzenie tylko **destruktorów wirtualnych** (patrz Funkcje wirtualne).

Ćwiczenia

Ćwiczenie 1

Napisz definicje instrukcji inicjujących do poniższej klasy:

```
class Vector
{
      public:
            double x;
            double y;
      public:
            Vector();
            Vector(double, double);
};
```

Klasa ma reprezentować wektor w przestrzeni dwuwymiarowej, a instrukcje inicjujące mają realizować inicjalizację tego wektora. Pierwsze instrukcje inicjujące powinny ustawiać wektor na wartość domyślną (0,0).

Ćwiczenie 2

Dopisz do kodu z poprzedniego ćwiczenia konstruktor kopiujący.

```
Vector(const Vector&);
```

Po wykonaniu tego ćwiczenia zastanów się, czy napisanie konstruktora kopiującego było konieczne. Jeżeli nie jesteś pewien - napisz program który testuje działanie Twojego konstruktora kopiującego i sprawdź jak program działa bez niego. Wyjaśnij dlaczego konstruktor kopiujący nie jest potrzebny.

Ćwiczenie 3

Poniższa klasa miała implementować dowolnej wielkości tablicę obiektów klasy Vector z poprzednich ćwiczeń. Niestety okazało się że powoduje wycieki pamięci - programista zapomniał o napisaniu destruktora:

```
class VectorsArray
{
      public:
            Vector* vectors;

            VectorsArray(size_t);
            Vector GetVector(size_t);
```

```
            size_t GetSize();
            size_t size;
};

VectorsArray::VectorsArray(size_t argSize)
: size(argSize)
, vectors(new Vector[argSize])
{
}
Vector VectorsArray::GetVector(size_t i)
{
     return vectors[i];
}
size_t VectorsArray::GetSize()
{
   return size;
}
```

Do powyższej klasy dopisz definicję destruktora. Nie zapomnij o dealokacji pamięci!

Źródła i autorzy artykułu

C++/Konstruktor i destruktor *Źródło*: http://pl.wikibooks.org/w/index.php?oldid=171555 *Autorzy*: Algorytmik, Delimata, Derbeth, Felix, Gang65, Kj, Lethern, Magalia, Matekm, Michał Głamowski, Piotr, Uniq, 43 anonimowych edycji

Źródła, licencje i autorzy grafik

Grafika:Fairytale messagebox info.png *Źródło*: http://pl.wikibooks.org/w/index.php?title=Plik:Fairytale_messagebox_info.png *Licencja*: GNU Lesser General Public License *Autorzy*: Amada44, Anime Addict AA, Bayo, Dake, Jon Harald Søby, Rocket000, ZooFari

Licencja