VIII Programowanie obiektowe

Konstruktory i destruktory(C++). Rodzaje konstruktorów w C++

I. Konstruktory

Konstruktor to specjalna funkcja składowa, która nazywa się tak samo jak klasa. W ciele tej funkcji możemy zamieścić intrukcje nadające wartości początkowe atrybutom definiowanego w danej chwili obiektu. W trakcie definiowania obiektu, przydziela mu się miejsce w pamięci, a następnie uruchamiany jest konstuktor.

UWAGA!
Kontruktor sam nie przydzila pamięci na obiekt. On może ja tylko zainicjalizować

Najważniejszym aspektem kontruktora jest to, ze jest uruchamiany **automatycznie**, przy definiowaniu każdego obiektu tej klasy.

Cechy konstruktora:

- ✔ Konstruktor może być przeładowywany
- ✔ Nie ma wyspecyfikowanego żadnego typu wartości zwracanej. Nie zwraca nic , nawet typu *void*! Jeśli więc w ciele konstruktora jest instrukcja *return* to nie może przy niej stać żadna wartość. Tylko średnik.
- ✔ Konstruktor może być wywoływany dla tworzenia obiektów z przydomkami *const* i *volatile*, ale sam nie może być funkcją typu *const* i *volatile*.
- ✔ Konstruktor nie może być typu *static*, gdyż ma z założenia pracować na niestatycznych składnikach klasy, jako *static* nie miałby takiej możliwości(nie miałby wskaźnika *this*).
- ✔ Konstruktor nie jest obowiązkowy, nie ma przymusu jego tworzenia jeśli nie czuje się takiej potrzeby.
- ✔ Konstruktor może być również prywatny

Przykład:

```
class human
{
  int age;
  char name[15];
  char surname[15];

  public:
  //konstruktor
  human(int a, const char *n, const char *s)
  {
    age=a;
    strcpy(name, n);
```

```
    strcpy(surname, s);
  }

}
  void hello()
  {
    cout<<age<<"name:"<<name<<" surname:"<<surname;
  }
}

int main()
{
  //definiujemy obiekt
  human john(10,"John","Smith");
  john.hello();
}
```

Kiedy i jak wywoływany jest konstruktor?
Podobnie jak w przypadku obiektów typów wbudowanych, tak i w przypadku obiektów typów zdefiniowanych przez użytkownika, możemy mieć kilka rodzajów obiektów, zależnie od tego gdzie i w jaki sposób te obiekty zdefiniujemy.

- ✔ *Obiekty lokalne autmatyczne*, powstają w momencie, gdy program napotyka ich definicję, a przestają istnieć, gdy gdy program wychodzi poza blok, w którym zostały zdefiniowane. Konstruktor w tym przypadku uruchamiany jest gdy program napotka definicję takiego obiektu.
- ✔ *Obiekty lokalne statyczne*, czyli takie, gdzie przy definicji stoi słowo *static*, istnieją od samego początku programu, aż do momenu jego zakończenia. Konstruktor rusza do pracy jeszcze zanim wykonana zostanie funkcja *main*.
- ✔ *Obiekty globalne*, czyli te, które definiowane są poza wszystkimi funkcjami i czas ich życia równy jest czasowi wykonywania programu. Konstruktor rusza do pracy przed wykonaniem funkcji *main*.
- ✔ *Obiekty tworzone za pomocą operatora new*, gdzie czas życia takiego obiektu trwa od momentu wywolania operatora *new* do momentu zlikwidowania za pomoca operatora *delete*. Konstruktor w takim przypadku jest wywoływany w momencie wywołania operatora *new*, zaraz po przydzieleniu pamięci.
- ✔ *Jawne wywołanie konstruktora*, ma miejsce gdy użyjemy składni nazwa_klasy(argmenty_konstruktora), tworzony jest wówczas obiekt nienazwany a czas życia ogranicza się tylko do wyrażenia w którym go użyto.

Typy konstruktorów w C++:

- ✔ *Konstruktor domniemany,* to taki konstruktor, który można wywołać nie podając żadnego argumentu, np.:

```
class test
{
 //....
 public:
 test(int);
```

```
 test(void); //konstruktor domniemany!
 test(char *);
}
```

Uwaga! Konstruktorem domniemanym może być również konstruktor ze wszystkimi argumentami domniemanymi.

✔ *Konstruktor kopiujący*, to taki konstruktor, który można wywołać z jednym argumentem, *klasa::klasa(klasa &);* Mówiąc krótko, konstruktor kopiujący służy do skonstruowania obiektu, który jest kopią innego, już istniejącego obiektu tej klasy. np.:

```
class human
{
//...
  public:
  human(human &template)
  {
    //...
  }
}
```

Uwaga! Argumentem musi być koniecznie referencja. Dlaczego? Załóżmy, ze jako argument dla konstruktora human, podajemy jakis obiekt klasy human przez wartosc, wówczas musi zostać zrobiona kopia, więc do pracy rusza konstruktor kopiujący, kontruktor jako argument otrzymuje wiadomo.. obiekt klasy human i znow rusza konstruktor kopiujacy, gdyż kolejny raz potrzebna jest kopia, itd....

✔ *Konstruktor konwertujący,* to taki konstruktor w klasie *K*, przyjmujący jeden argument, np. typu *T*, *K::K(T)* (nie może być *explicit*!), określa konwersję: od typu tego argumentu do typu swojej klasy, *T*---->*K*, np.:

```
class zespolona
{
   double rzeczywista;
   double urojona;
   public:
   //konstruktor konwertujący
   zespolona(double r)
   {
      rzeczywista=r;
      urojona=0;
    }
   //...
}
```

Dlaczego bez *explicit*?
Ponieważ specyficator *explicit* oznacza, ze zabraniamy kompilatorowi użyc tego konstruktora niejawnie!

Co to daje?
Mając funkcje *dodaj(zespolona, zespolona)*, gdy użyjemy *wynik=dodaj(pierwsza, 7.5)*, kompilator zrozumie to jako *wynik=dodaj(pierwsza, zespolona(7.5))* i wówczas

mamy przykład niejawnego wywołania konstruktora.

II. Destuktory

Destruktorem klasy *K* jest jej funkcja składowa o nazwie ~*K*. Funkcja ta jest wykonywana automatycznie, gdy obiekt jest likwidowany(np. za pomocą operatora *delete*). Klasa nie musi mieć obowiązkowo destruktora, tak samo jak w przypadku konstruktora.

UWAGA!
Destruktor nie likwiduje obiektu, ani nie zwalnia obszaru pamięci, którą obiekt zajmował, przydaje się wtedy, gdy przed zlikwidowaniem obiektu należy coś posprzątać.

Destruktor jest przydatny, gdy obiekt dokonał na swój użytek rezerwacji dodatkowej pamięci(za pomocą operatora *new*) jak np. zaalokowal dodatkową pamięć na dużą tablicę. W takiej sytuacji znajduje właśnie zastosowanie destruktor, gdzie w jego ciele umieszczamy instrukcję *delete* zwalniając zaalokowany wczesniej obszar pamięci na tą tablicę. Innym dość częstym zastosowaniem destruktora może być sytuacja, gdy zliczamy ilość instancji danej klasy, wówczas w destruktorze umieszczamy instrukcje dekramentacji.

Cechy destruktora:

- ✔ Destruktor tak samo jak konstruktor, nie może zwracać żadnej wartości, nawet typu *void*.
- ✔ Destruktor nie może być wywoływany z argumentami, czyli nie może być przeładowywany.
- ✔ Destruktor nie jest wywoływany, gdy wskaźnik do jakiegoś obiektu wychodzi ze swojego zakresu. To, że wskaźnik przestaje istnieć nie oznacza, że obiekt również kończy swój żywot.
- ✔ Destruktor może wywoływać jakąś funkcje składową
- ✔ Destruktor nie może być typu *const* ani *voltaile*, ale może pracować na obiektach klasy z takim przydomkiem.
- ✔ Destruktor może być wirtualną metodą.

Przykład:

```
class foobar
{
  int *tab;
  public:
  //konstruktor
  foobar
 {
    tab=new int[50];
 }
```

```
//destruktor
~foobar
{
    delete [] tab;

  }
}
```

Destruktor może zostać wywołany jawnie, używa się wówczas składni: *obiekt.~klasa();* badź też *wskaznik->~klasa();*