C++: Klasy i obiekty II

Składniki statyczne

Składnik statyczny, w przeciwieństwie do pozostałych składników jest jeden wspólny dla wszystkich obiektów danej klasy.

```
class samochod{
....
    static int liczba_samochodow;
....
}
```

- Do składnika statycznego odnosimy się tak samo jak do zwykłego składnika.
- Dodatkowo możemy się do niego odnieść za pomocą nazwy klasy i operatora zakresu

```
samochod::liczba_samochodow=0;
```

Kompilator w trakcie czytania definicji klasy nie generuje żadnego kodu. W szczególności nie zarezerwuje nam pamięci dla składnika statycznego. Nie stanie się to też w momencie rezerwowania pamięci dla jakiegoś obiektu, bo składnik statyczny ma być zawsze jeden. Dlatego każdy składnik statyczny, oprócz deklaracji umieszczonej w klasie musi też mieć umieszczoną poza klasą definicję, dzięki której kompilator zarezerwuje dla niego stosowną pamięć. Robi się to wykorzystując operator zakresu:

```
int samochod::liczba_samochodow=0;
```

W powyższym przykładzie obok definicji dokonano też inicjalizacji składnika statycznego.

- Składnika statycznego nie można inicjalizować wewnątrz klasy, z wyjątkiem składników statycznych stałych.
- Składnika statycznego nie można inicjalizować na liście inicjalizacyjnej konstruktora.

Statyczne funkcje składowe

Statyczne mogą być również funkcje składowe, pod warunkiem, że operują wyłącznie na składnikach statycznych.

```
class samochod {
   .........
   static int ileSamochodow(void){
     return liczba_samochodow;
   };
};
```

Funkcje statyczne można wywołać nawet wtedy, gdy nie powstał jeszcze żaden obiekt danej klasy

Stałe składniki

- Słowa kluczowego `const` można użyć w odniesieniu do danej składowej klasy.

```
class Pojemnik{
  const int pojemnosc;
  int zawartosc;
  ...
}
```

- Składnik taki inicjalizowany jest raz w momencie tworzenia obiektu klasy i nie może być potem zmieniany
- Inicjalizacja stałego składnika odbywa się na liście inicjalizacyjnej konstruktora
- `Pojemnik(int p, int z):pojemnosc(p),zawartosc(z){}`

Składnik typu `const` można incjalizować tylko za pomocą listy inicjalizacyjnej

Stałe składniki statyczne

- W klasie można definiować stałe składniki statyczne
- Składniki takie, w przeciwieństwie do zwykłych składników statycznych definiuje się i inicjalizuje wraz z definicją klasy, a nie poza nią

```
class Wiadro{
  static const int pojemnosc=20;
  int zawartosc;
  ...
}
```

- Stałe składniki statyczne są wspólne dla wszystkich obiektów klasy
- Wartość stałych składników statycznych dostępna jest kompilatorowi w trakcie kompilacji

Obiekty stałe i stałe funkcje składowe

Obiekty, podobnie jak zwykłe zmienne, mogą być zdefiniowane jako stałe.

```
const samochod kra0089("Trabant",78825,4.5);
```

Dla takich obiektów wolno wywoływać tylko te funkcje składowe, które zadeklarowane są jako stałe

```
void prezentuj(void) const{
    cout << "To jest samochod marki " <<
            model << " o przebiegu " <<
            przebieg << "km i " <<
            paliwo << "l paliwa w baku\n";
};
```

Niestałe składniki w stałych obiektach

- Czasami gotowi jesteśmy uznać obiekt klasy za stały, pomimo faktu, że jakiś jego składnik się zmienia.
- Najczęściej dotyczy to składników, które nie są cechą opisującą obiekt, ale jakimś pomocniczym licznikiem
- Na przykład możemy chcieć zliczać ile razy jakaś funkcja została na rzecz danego obiektu wywołana
- Wtedy funkcja taka nie może być zadeklarowana jako `const`, nawet jeśli nie zmienia żadnych pozostałych składników klasy
- Rozwiązaniem jest poprzedzenie deklaracji składnika klasy, który może być modyfikowany nawet przez funkcje `const` słowa kluczowego `mutable`, na przykład

```
mutable int licznik_prezentacji;
```

Stałe obiekty chwilowe

- Przypomnijmy, że słowo kluczowe `const` można też wykorzystać w typie obiektu zwracanego przez funkcję

```
class K{
   ...
}
const K g(){
```

```
  ...
}
```

- Takie użycie `const` oznacza, że zwrócony, chwilowy obiekt, nie może być zmodyfikowany
- W szczególności taki obiekt nie może być wysłany do żadnej funkcji, która mogłaby go zmodyfikować

Tablice obiektów

Podobnie jak tablice typów wbudowanych można tworzyć tablice obiektów danej klasy.

```
samochod rent_a_car[20];
```

W momencie definiowania takiej tablicy można też przeprowadzić inicjalizację

```
samochod rent_a_car[20] = {
   samochod("Toyota Avensis",1200,20.7),
   samochod("Honda Civic",44000,39.2),
   ....
};
```

Funkcje zaprzyjaźnione

Funkcja zaprzyjaźniona to funkcja, która ma dostęp do prywatnych składników klasy choć nie jest funkcją składową klasy.

```
class samochod{
  .......
  friend void mechanik(samochod &jakis_samochod);
};
.........
void mechanik(samochod &jakis_samochod){
  jakis_samochod.przebieg+=5;
}
```

Jedynie sama klasa może zadeklarować, że jakaś funkcja jest z nią zaprzyjaźniona. Nie może zrobić tego zainteresowana funkcja

Pożytki z funkcji zaprzyjaźnionych

- Funkcja może być przyjacielem więcej niż jednej klasy. Czyli może mieć dostęp do prywatnych składników kilku klas.
- Funkcja zaprzyjaźniona nie musi być wywoływana na rzecz jakiegoś obiektu danej klasy

- Dzięki funkcjom zaprzyjaźnionym możemy nadać dostęp do prywatnych składników klasy funkcjom napisanym w innych językach programowania.

Wskaźnik do składników klasy

- Wszystkie obiekty danej klasy są tak samo zorganizowane w pamięci, to znaczy relatywne położenie składnika obiektu względem adresu obiektu w pamięci nie zależy od wyboru obiektu klasy
- Z tego powodu do niestatycznego składnika klasy można zastosować operator adresu.
- Operator ten zwraca adres relatywny, to znaczy adres składnika w obiekcie zmniejszony o adres samego obiektu
- Adres taki można przechowywać we wskaźniku do składników klasy

```
struct K{
   int a,b,c;
}
int K::* p; // wskaźnik do składników typu int w K
p=&K::a;    // p wskazuje na składnik a
```

Wskaźnika do składników klasy nie można ustawiać na składniki niedostępne w danym zakresie, ani stosować do niego arytmetyki wskaźników.

- Aby skorzystać z takiego wskaźnika potrzebny jest obiekt klasy `K` lub wskaźnik do pokazywania na obiekty klasy `K`.
- Jeśli `k` jest obiektem klasy `K`, do składnika wskazywanego przez wskaźnik `p` dobieramy się używając operatora .*

```
K k;
cout << k.*p;
```

- Jeśli `q` jest wskaźnikiem do pokazywania na obiekty klasy `K`, to do składnika wskazywanego przez wskaźnik `p` obiektu wskazywanego przez wskaźnik `q` dobieramy się używając operatora ->*

```
K k;
K* q=&k;
      cout << q->*p;
```