POO. Observații

Document editat de Raluca Tudor

Martie 2020

1 Sizeof(clasa)

```
class A {
public:
};

int main() {
    cout << sizeof(A); // 1- POINTER CATRE VOID
}
```

Când adaug o variabilă, automat, sizeof(clasă) = sizeof(datelor pe care le am)

Când se creează o clasă (deci un anumit tip de date) și vreau să instanțiez tipul respectiv, trebuie să îi aloc o zonă de memorie. Dar dacă în acea clasă nu avem nici date și nici funcții, obiectul în continuare există, deci trebuie să am un pointer către (ceva), și atunci se utilizează un pointer către void.

```
class A {
public:
    int x;
};

int main() {
    cout << sizeof(A); // 4
}
```

```
class A {
public:
    int x;
    char y; // nu am acelasi tip de data
};

int main() {
    cout << sizeof(A); // 8 - se lucreaza cu grupuri de catre 4 octeti/ se face o
    ALINIERE si se completeaza pana la multiplu de 4
}
```

```
class A {
public:
    char x, z;
    char y; // acelasi tip de data
};

int main() {
    cout << sizeof(A); // 3 - suma sizeof(datelor)
}
```

```
class A {
public:
    long long x;
    char y;
};
```

```
6
7  int main() {
8      cout << sizeof(A); // 16
9  }
```

```
1  class A {
2  public:
3      long long x;
4      char y;
5      int z;
6  };
7
8  int main() {
9      cout << sizeof(A); // 24
10 }
```

sizeof(A) nu se modifică dacă adaug funcții! Funcțiile sunt create într-o altă zonă de memorie, eu doar am acces la ele. Compilatorul face legătura dintre clasă și zona de memorie pentru funcții.

```
1  class A
2  {
3  public:
4      int z;
5      virtual void f(){}
6      virtual void g(){}
7      virtual void h(){}
8  };
9
10 int main()
11 {
12     cout<<sizeof(A); // 8 -> 1 int + vptr
13 }
```

```
1  class A
2  {
3  public:
4      //int z;
5      virtual void f(){}
6      virtual void g(){}
7      virtual void h(){}
8  };
9
10 int main()
11 {
12     cout<<sizeof(A); // 4 -> vptr
13 }
```

2 Keyword-uri

2.1 inline

Funcțiile inline se comportă ca macrodefiniții - se copiază codul cu valorile înlocuite și nu se mai face apel de funcție! Funcțiile implementate (definite complet) în clase sunt implicit (by default) inline (sau, mai degrabă, este o sugestie către compilator - se încearcă să se pună bucata de cod în cache-ul procesorului).

2.2 static

Variabilele statice se pun în zona variabilelor globale, nu pe stivă! O variabilă statică este unică la nivelul clasei! Funcțiile statice se pot executa de sine-stătătoare, nu trebuie să instanțieze clasa.

In funcțiile statice putem folosi variabilele statice, parametrii trimiși sau putem instanția obiecte locale de clasa respectivă, caz în care este ok să folosim membrul nestatic al obiectului nou creat. Însă nu putem folosi membrii nestatici ai lui this!!!

2.3 const

Când declarăm o variabilă const, trebuie și să îi dăm o valoare!

Obiectele const pot apela doar metodele const!!!

2.4 explicit

explicit evită conversiile implicite pe care le face compilatorul (C++ iubește conversiile :))

3 Când se apelează copy constructorul?

1. Când construim un obiect pe baza unui alt obiect.
2. Când pasez un obiect ca paramentru într-o funcție.
3. Când returnez un obiect prin valoare.

4 Alte observații

Operatorii de citire și de afișare nu pot fi metode deoarece fluxul este primul. (sigur, putem să punem invers și așa să fie metodă, dar e contraintuitiv).

Niciun operator nu acceptă parametrii default, cu excepția lui ()!