Tutoriat 8

Cheat sheet

Acest material reprezintă chestiuni de teorie de care aveți nevoie în rezolvarea exercițiilor de la examen

Acestea NU sunt copiuțe pentru examen!

1. **Noțiuni introductive**
   Struct (C++) – accesul default este **public**
   Class (C++) – accesul default este **private**

   Funcțiile friend pot accesa și membrii private sau protected ai unei clase.

2. **Moștenire**
   - ✓ Presupune clasa de bază și clase derivate din aceasta. (una sau mai multe)
   - ✓ **Private** în clasa de bază => **inaccesibil** în clasa derivată.
   - ✓ **Protected** în clasa de bază => **accesibil** în clasa derivată.
   - ✓ Tipurile de moștenire(public, private, protected) influențează accesul datelor și metodelor moștenite în clasa derivată.
   - ✓ Constructorii sunt apelați din clasa de bază spre derivată. Destructorii sunt apelați invers constructorilor.

3. **Compunere**
   Instanțiere de obiecte de tipul unei clase în altă clasă.

4. **Keyword-ul const**
   - ○ Orice încercare de modificare a unui obiect constant => eroare de compilare.
   - ○ Variabilele constante trebuie să fie mereu inițializate.
   - ○ Pointer constant către un tip oarecare (adresa nu se modifică):
     *tip * const nume_pointer;*
   - ○ Pointer către un tip oarecare constant (zona de memorie nu se modifică):
     *const tip * nume_pointer;*
   - ○ Datele membre constante pot fi inițializate/modifică valoarea doar la declarare sau prin lista de inițializare.

- Metodele constante nu au voie să schimbe nimic la datele pointerului *this*.

5. **Keyword-ul static**
   - ✓ Variabila statică este inițializată o singură dată la pornirea programului.
   - ✓ Are nevoie de linia de inițializare din afara clasei, altfel => eroare de compilare.
   - ✓ Pot fi accesate prin instanțe sau numele clasei cu ::
   - ✓ Aceeași valoare pentru toate instanțele clasei.
   - ✓ Metodele statice nu au *this* => lucrează doar cu date și metode statice.

6. **Supraîncărcarea (overloading)**
   - Metode cu același nume dar parametrii diferiți (nu contează tipul returnat).
   - Operatorii **, ::, . , ?:, sizeof, typeid* **nu** pot fi supraîncărcați.
   - **Hiding**
     - Supraîncărcare la moștenire.
     - Printr-o instanță de tipul clasei derivate accesăm metoda supraîncărcată din clasa derivată.
     - Printr-o instanță de tipul clasei derivate folosim numele clasei de bază și *operatorul ::* pentru a accesa metoda supraîncărcată din clasa de bază.

7. **Suprascriere (overriding)**
   - ✓ Se realizează numai la moștenirea claselor.
   - ✓ Metode cu același nume și aceiași parametrii.
   - ✓ Printr-o instanță de tipul clasei derivate accesăm metoda suprascrisă din clasa derivată.
   - ✓ Printr-o instanță de tipul clasei derivate folosim numele clasei de bază și *operatorul ::* pentru a accesa metoda suprascrisă din clasa de bază.

8. **Upcasting**
   - ✓ Upcasting dinamic:
     clasă_de_bază pointer = new clasă_derivată();*
   - ✓ Moștenirea trebuie să fie de tip public, altfel => eroare de compilare.

- ✓ Suprascriere/supraîncărcare + upcasting => este accesibilă doar metoda din clasa de bază.
- ✓ Upcasting static (nerecomandat):
  clasă_de_bază ob = (clasă_de_bază) ob_clasă_derivată;

9. Metode virtuale

- ❖ Se folosesc în clasele de bază cu metode care urmează a fi suprascrise în clasele derivate.
- ❖ Nu produce erori dacă este folosit și la metode din clase care nu urmează a fi moștenite.
- ❖ Virtual + suprascriere + upcasting => apel metodă din clasa derivată.

10. Moștenirea diamant

Keyword-ul **virtual** la moștenirile de la mijlocul diamantului rezolvă problema => nu se mai moștenesc duplicate în clasa cea mai de jos din ierarhie

11. Downcasting

*clasă_derivată * pointer=dynamic_cast<clasă_derivată*>(obiect_clasă_de_bază);*

- Operatorul **dynamic_cast** retrunează NULL dacă nu se poate face conversia.
- În clasa de bază trebuie să existe **cel puțin** o metodă/ constructor/ destructor virtual, altfel => eroare de compilare.
- Pointerul pe care se face conversia trebuie să reprezinte un upcasting, altfel **dynamic_cast** retrunează NULL
- Downcasting fără operatorul **dynamic_cast**, fără metode virtuale și fără upcasting – este posibil sub forma:
  *clasă_derivată * pointer = (clasă_derivată*) pointer_clasă_de_bază;*

12. Abstractizare

- ❖ Clase care au cel puțin o metodă pur virtuală:
  virtual void nume_metodă() = 0;
- ❖ **NU** pot fi instanțiate.
- ❖ Clasele care le moștenesc trebuie să implementeze toate metodele pur virtuale, altfel nici ele nu pot fi instanțiate.

13. Template

- ❖ Typename este același lucru cu class (standarde C++ diferite).

- La apelul de funcții nu este obligatorie specificarea *<tip>*, dar la clase este.
- Ordinea de potrivire la compilare:
    1. Funcții normale
    2. Funcții specializate
    3. Funcții template