Appunti pre-esame orale da ricordare

Allocazione dinamica della memoria

`tipo *nome_puntatore = new tipo;` →NON ARRAY
es.
`int *ptr = new int;`
es.
`Studente *ptr = new Studente;`

`tipo *nome_puntatore = new tipo[dimensione];` → ARRAY
es.
`int *array = new int[10];`
es.
`Studente *array = new Studente[10];`

Deallocazione memoria dinamica
`delete nome_puntatore;` → NON ARRAY

`delete[] nome_puntatore;` →ARRAY

Classi

`class nome_classe{` →definizione di una classe
`public:`
`void print();`
`private:`
`int dato;`
`string nome`
`};`

una funzione può essere definita dentro la classe oppure fuori in questo modo, a patto che il prototipo si trovi nella classe
`void nome_classe::print(){`
`...`
`}`

Costruttore
non si può inizializzare un membro dentro la classe, va fatto nel costruttore che ha lo stesso nome della classe e può aspettarsi dei parametri. Si definisce all'interno della

classe in questo modo:
`nome_classe(){}` →se devo inizializzare delle variabili dopo le parentesi tonde metto i 2 punti :
es.
`nome_classe():dato(10){}` dentro le parentesi graffe posso effettuare delle operazioni
(la parte con i 2 punti si chiama vettore di inizializzazione dei membri ed è il modo preferito per inizializzare uno o più membri)

se devo chiamare una funzione di una classe ho bisogno di fare l'istanza a quella classe e usare l'operatore punto "."
es.
```
nome_classe dato;
dato.print();
```
(oppure l'operatore freccia "→" se l'istanza fatta è un puntatore)
es.
```
nome_classe *dato;
dato->print();
```

se ho un costruttore che si aspetta dei parametri questi vanno forniti alla definizione dell'istanza
es.
`nome_classe dato("Harry Potter",675,50);`
altrimenti posso creare un costruttore che si aspetta parametri e in aggiunta un "costruttore di default" che non aspetta nulla, in questo modo posso comunque creare l'istanza

(In sintesi:

- **Costruttore di default**: nessun parametro.
- **Costruttore con parametri**: riceve valori.
- **Costruttore di copia**: riceve un oggetto della stessa classe.)

Distruttore
E' un metodo speciale che serve a deallocare la memoria assegnata ad un oggetto dal costruttore, se non se ne crea uno C++ ne creerà uno di default.
Non accetta parametri e ce ne può essere solo 1 per classe.
ha lo stesso nome della classe come il costruttore ma è preceduto dalla tilde ~
es.
`~nome_classe(){}`
all'interno delle parentesi graffe posso deallocare memoria:

es.
`~nome_classe(){ delete [] array;}`

Classi derivate (ereditarietà e polimorfismo)

la classe derivata eredita membri e metodi della classe base, e si dichiara cosi:
`class classe_derivata : specificatore_accesso nome_classe_base{};`

per specificatore si intende public private o protected (spesso si usa public)

bisogna che il costruttore della classe base venga chiamato per creare un oggetto (della classe base) prima che si attivi il costruttore della classe derivata (l'oggetto della classe base deve esistere prima di diventare un oggetto della classe derivata)

```
class Libro : public Libreria{

    public: //Libro prende 2 stringhe titolo e autore e le passa al costruttore di Libreria dove vengono inizializzate
    Libro(string title, string autor, int n):Libreria(title,autor),nPagine(n){}
```

Il distruttore non si eredita.
si gestiscono come i costruttori, ma tutto va fatto in ordine inverso, vale a dire che s'inizia ad eseguire il distruttore dell'ultima classe derivata e si sale fino a quella base (il costruttore al contrario)

Funzioni virtuali
`virtual` anteposto alla dichiarazione di una funzione indica al compilatore che essa può essere definita in una classe derivata (deve avere però lo stesso tipo e gli stessi parametri in input)

Quindi se io voglio usare un metodo della classe base con un oggetto della classe derivata non posso farlo, devo dichiarare la funzione virtual nella classe base e ridefinirla uguale (tranne per il corpo che può cambiare) nella classe derivata

Template

I template sono un metodo per istanziare classi o funzioni a tempo di compilazione
in breve: permette di definire funzioni o classi di tipo generico in modo da poterlo cambiare senza modificare la funzione o classe (una volta int poi float poi double ecc...)

I programmatore devo quindi solo specificare i tipi con cui essi devono lavorare

- definizione

  ```
  template <typename T> class nome_classe
  {...};
  ```

  - `T` nome del tipo (qualunque) utilizzato dal template

- istanziazione

  ```
  nome_classe <tipo> oggetto;
  ```