AGGIORNAMENTI RECENTI IN C++

Programmazione ad Oggetti

Corso di Laurea in Informatica

Università della Calabria

C++0x/C++11/C++14/C++17 : Cosa sono?

- C++ ha subito molti miglioramenti da quando è stato creato
- Diverse revisioni al linguaggio sono state apportate negli anni
- Nel 2011 è stata ufficialmente rilasciata un'importante revisione del linguaggio, chiamata C++11 o anche C++0x
- C++11 rappresenta un punto di svolta per C++
- Aggiunge molte nuove funzionalità "salva-tempo", richieste da anni dai programmatori C++
- Ulteriori revisioni sono state fatte più recentemente con C++14 e C++17

Cosa vedremo?

- Alcune funzionalità aggiunta a partire da C++11
  - **For each**
  - La parola chiave **auto**
  - La parola chiave **override**
  - Inizializzazione dei dati membri nella classe
- Per ulteriori dettagli:
  - http://www.cplusplus.com/reference/
  - Il libro «**C++ Without Fear**» Terza Edizione - Brian Overland - Prentice Hall
  - Il libro «**C++. Linguaggio, libreria standard, principi di programmazione**» Quarta Edizione - Bjarne Stroustrup - Pearson

- Per usare queste nuove caratteristiche occorre compilare usando l'opzione:
  - **--std=c++11** (oppure **--std=c++0x**)
- La **versione** del compilatore **g++** deve essere **4.6 o superiore**
- Per verificare la versione di g++ installata:
  - **g++ -v**

- Ad esempio:
  - **g++ --std=c++11 main.cpp**
- Oppure:
  - **g++ --std=c++0x main.cpp**

FOR EACH E AUTO

Esempio

```
#include <iostream>
#include <list>
using namespace std;

int main(){
 list<int> l;
 for(unsigned i=0;i<10;i++)
  l.push_back(i);

 for(list<int>::const_iterator it=l.begin();it!=l.end();it++)
  cout<<(*it)<<" ";
 cout<<endl;

 for(list<int>::iterator it=l.begin();it!=l.end();it++){
  (*it)++;
  cout<<(*it)<<" ";
 }
 cout<<endl;
```

Esempio (continua…)

```
float array[4]={1.2, 2.3, 3, 4};
for(float f: array)
 cout<<f<<" ";
cout<<endl;

cout<<"Iterazione con for each c++11 per valore:"<<endl;
for(int n: l)
 cout<<n<<" ";
cout<<endl;

cout<<"Iterazione ed assegnamento con for each c++11
 per riferimento:"<<endl;
for(int& n: l){
 n++;
 cout<<n<<" ";
}
cout<<endl;
```

Esempio (continua…)

```
 auto stringa= "Ciao";
 auto j = 5;

 cout<<"Iterazione con for each c++11 e auto per valore:"<<endl;
 for(auto n:l)
  cout<<n<<" ";
 cout<<endl;

 cout<<"Iterazione ed assegnamento con for each c++11 e auto
  per riferimento:"<<endl;
 for(auto& n:l){
  n++;
  cout<<n<<" ";
 }
 cout<<endl;
 return 0;
}
```

Riassumendo

- Il «for each» rappresenta una nuova tipologia di for più compatta che permette di iterare su array, liste, vettori, altre strutture dati

```
for (tipo_base& variabile: struttura_dati){ // Per riferimento
   // istruzioni
}
for (tipo_base variabile: struttura_dati){ // Per valore
   // istruzioni
}
```

- **`auto`**
  - Indica al compilatore di determinare «automaticamente» il tipo di una variabile in base al contesto
  - Il tipo viene scelto al momento dell'inizializzazione
  - Può essere utilizzato in un «for each»

OVERRIDE

Esempio

```
#ifndef POINT2D_H
#define POINT2D_H
#include <iostream>
using namespace std;

class Point2D {
private:
 int x;
 int y;
public:
 Point2D():x(0),y(0){}
 Point2D(int a, int b):
  x(a),y(b){}
 virtual void stampa() const
  {cout<<"X: "<<x<<" Y: "<<y;}
};
#endif
```

```
#ifndef POINT3D_H
#define POINT3D_H
#include "Point2D.h"

class Point3D : public Point2D {
private:
 int z;
public:
 Point3D():Point2D(),z(0){}
 Point3D(int a, int b, int c):
  Point2D(a,b),z(c){}
 virtual void stampa() const override
  {Point2D::stampa();
    cout<<" Z: "<<z;}
};
#endif
```

Esempio (continua…)

```
#include "Point3D.h"

int main(){
        Point2D* p = new Point2D;
        p->stampa();
        cout<<endl;

        delete p;
        p = new Point3D;
        p->stampa();
        cout<<endl;

        return 0;
}
```

- **`override`** si può aggiungere alla fine della dichiarazione di una **funzione virtuale**
- Permette di verificare la correttezza dell'**overriding**
- Se ciò non accadesse (ad esempio, perché il nome della funzione è errato) il compilatore segnalerebbe un **errore**

INIZIALIZZAZIONE DEI DATI MEMBRO NELLA CLASSE

Esempio

```
#ifndef LIBRO_H
#define LIBRO_H

#include <iostream>
using namespace std;

class Libro {
private:
 int isbn = 0;
 string autore = "";
 string titolo = "";
public:
 void stampa(){cout<<"ISBN:"<<isbn<<
  " Autore: "<<autore<<" Titolo: "
  <<titolo<<endl;};
 void setIsbn(int c){isbn=c;}
 void setAutore(const string& s)
  {autore=s;}
 void setTitolo(const string& s)
  {titolo=s;}
};
#endif
```

```
#include "Libro.h"

int main(){
        Libro l;
        l.stampa();

        l.setIsbn(123);
        l.stampa();

        return 0;
}
```

Riassumendo

- C++11 consente di specificare i valori di default a cui inizializzare i dati membro di una classe
- **Questa funzionalità non sostituisce i costruttori**
- Nel caso in cui un costruttore non inizializza un dato membro, a cui è assegnato un valore di default, esso assumerà tale valore
- Ad esempio, se la classe `Libro` dell'esempio fosse dotata del costruttore:
  ```
  Libro(string a, string t): autore(a), titolo(t) {}
  ```
  ogni oggetto creato con questo costruttore avrebbe `isbn` pari a `0`
- Se un costruttore inizializza un dato membro, a cui è assegnato un valore di default, esso assumerà il valore assegnato dal costruttore
- Ad esempio, se la classe `Libro` dell'esempio fosse dotata del costruttore:
  ```
  Libro(string a, string t): isbn(1), autore(a), titolo(t) {}
  ```
  ogni oggetto creato con questo costruttore avrebbe `isbn` pari a `1`