28. Alcune definizioni (in C++!)

Corso di Informatica

Corso di Laurea in Matematica (D.M. 270/04) - A.A. 2020/2021
Angelo Cardellicchio
angelo.cardellicchio@uniba.it
14/12/2020

Outline

- Namespace
- I/O (su riga di comando)
- Variabili reference
- Concetti avanzati su funzioni

Namespace (1)

- Abbiamo già visto che non è possibile associare ad una variabile una keyword
- Sappiamo anche che non è possibile (o è quantomeno **sconsigliato**) associare lo stesso nome a più variabili
- In C++ questo problema è amplificato dalla possibilità di definire nuovi tipi mediante le classi
- Queste problematiche sono risolte introducendo il concetto di **namespace**

Namespace (2)

- Immaginiamo di voler creare due funzioni che restituiscano rispettivamente le coordinate cartesiane e polari di un punto
- Una possibile soluzione è la seguente:

```
double* restituisci_coordinate_cartesiane(double x, double y)
{
    double coordinate[2] = {x, y};
    return coordinate;
}

double* restituisci_coordiante_polari(double r, double theta)
{
    double coordinate[2] = {r, theta};
    return coordinate;
}
```

Namespace (3)

- All’aumentare delle funzioni, però, aumenterebbero le possibili ambiguità
- Utilizzando i **namespace**:

```
namespace cartesiano {
    double* restituisci_coordinate(double x, double y) {
        // ...
    }
}

namespace polare {
    double* restituisci_coordinate(double r, double theta) {
        // ...
    }
}
```

Namespace (4)

- Per usare i namespace, possiamo usare la clausola **`using namespace`**:

```
using namespace cartesiano;
```

- In alternativa, anteponiamo (sempre) il namespace alla funzione/variabile/classe che useremo, seguito dall'**operatore di risoluzione di ambito**:

```
cartesiano::restituisci_coordinate(2.0, 3.0);
```

- Il suggerimento è, ove possibile, usare la prima soluzione per ottenere codice meno prolisso
- Il namespace maggiormente utilizzato nei programmi C++ è probabilmente il namespace **`std`** (crasi di **standard**)

I/O in C++ (da riga di comando) (1)

- La gestione dell'I/O in C++ è delegata alla libreria **`iostream`**
  - *Volendo, possiamo comunque usare le tecniche standard del C!*
  - *Permangono i meccanismi di interazione mediante stream*
- Per quello che riguarda l'output, esiste una classe base, definita all'interno del namespace std, e chiamata **`ostream`**
- Questa ci permette di definire tre tipi di canale:
  - **`cout`**: *usato come canale di default*
  - **`clog`**: *usato per il log*
  - **`cerr`**: *usato per gli errori*

I/O in C++ (da riga di comando) (2)

- Ecco un esempio:

```
#include <iostream>

using namespace std;

int main() {
    cout << "Questo è un messaggio in output" << endl;
    cout << "Concateno il numero: " << 2 << endl;
    cerr << "Questo è un messaggio di errore" << endl;
    clog << "Questo è un messaggio di log" << endl;
}
```

- L'operatore **<<** è detto di *inserimento*
  - *È possibile concatenare più operazioni di output*
  - **endl** *rappresenta l'escape character* **\n**
  - **ostream** *gestisce in automatico i diversi tipi base*

I/O in C++ (da riga di comando) (3)

- L'input da riga di comando avviene mediante la classe **`istream`**, che definisce un canale **`cin`**
- In questo caso, al posto dell'operatore di inserimento, viene utilizzato quello di estrazione **`>>`**
- L'utilizzo di **`cin`** è comunque duale rispetto a quello di **`cout`**
- Ad esempio:

```
int a;
cin >> a;
```

I/O in C++ (da riga di comando) (4)

- Possiamo richiedere più input separandoli con uno spazio:

```
int a;
string s;
cin >> a >> s;
```

- Per verificare la correttezza dei tipi usati si usa la funzione **`good()`**:

```
if (cin.good()) { cout << "Ok!"; }
else { cerr << "Errore!" }
```

- Per inserire stringhe composte da più parole si usa la funzione **`getline()`** inclusa nella libreria **`string`**:

```
#include <string>
string composed;
getline(cin, composed);
```

Variabili reference

- Sono degli **alias** per variabili già esistenti
- Sono logicamente equivalenti all'applicazione di un'operazione di dereferenziazione su un puntatore
- Sono precedute nella dichiarazione dal simbolo **&**

```
int a = 2;
int* pa = &a;
int& ref_a = a;
```

- Utilizzarle è una scelta stilistica (sono meno prolisse)
- Vanno necessariamente **inizializzate**, e cambiare la variabile di cui sono alias a runtime non è possibile (o comunque **sconsigliato**)

Concetti avanzati sulle funzioni (1)

- C++ offre alcune potenzialità aggiuntive rispetto al C a riguardo delle funzioni
- Un primo esempio è la possibilità di utilizzare dei parametri **opzionali**, ovvero parametri che possono essere omessi quando si chiama una funzione
- Ad esempio:

```
int calcola_voto_esame(int* voto_esoneri, int numero_esoneri = 4)
{
    int sum = 0;
    for (int i = 0; i < numero_esoneri; i++) {
        sum += voto_esoneri[i];
    }
    return int(round(sum / numero_esoneri));
}
```

Concetti avanzati sulle funzioni (2)

- Altro esempio è l'**overloading**, che permette di usare lo stesso nome per due funzioni differenti, **differenziandole in base ai parametri passati**
- Il tipo di ritorno **non** è importante ai fini della differenziazione delle funzioni in overloading; quello dei parametri passati invece lo è

```
int calcola_voto_esame(int* voto_esoneri,
            int numero_esoneri = 4) {
    int sum = 0;
    for (int i = 0; i < numero_esoneri; i++) {
        sum += voto_esoneri[i];
    }
    return int(round(sum / numero_esoneri));
}

double calcola_voto_esame(double* voto_esoneri,
            int numero_esoneri = 4) {
    double sum = 0;
    for (int i = 0; i < numero_esoneri; i++) {
        sum += voto_esoneri[i];
    }
    return (sum / numero_esoneri);
}
```

Domande?

42