Programmazione a oggetti

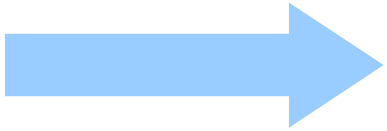
overloading di operatori particolari, uso di const classe string

A.A. 2020/2021
Francesco Fontanella

Overloading di [ ]

```
int main()
{
  Tipo x;
  MyArray a;
     .
     .
  x = a[3];
     .
     .
}
```

```
class MyArray{
    Tipo *v;
      .
  public:
      .
      .
    Tipo& operator[](int i){return v[i];}
  };
```



```
x = a.operator[](3)
```

NOTA
Se **operator**[]() restituisce by reference, l'operatore potrà poi essere utilizzato sia a destra che a sinistra nelle istruzioni di assegnazione

Esempio

```
int main()
{
  MyArray a;

  cout << a[2];  // visualizza 2
  cout << " ";
  // [] a sinistra di = (è un l-value)
  a[0] = 5;
  a[2] = 10;

  // [] a destra di = (è un r-value)
  cout << a[0];  // visualizza 5
  cout << a[2];  // visualizza 10

  return 0;
}
```

```
// controllo di limite
int &MyArray::operator[](int i)
{
  if(i<0 || i > SIZE-1) {
    cout <<endl<< "ERRORE!: indice";
    cout << i << " è fuori limite.\n";
    exit(1);
  }
  return v[i];
}
```

NOTA

L'operatore [ ] DEVE avere un unico parametro

Overloading di ()

- Alla funzione **operator**() è possibile passare un numero arbitrario di parametri, che possono essere di qualsiasi tipo.
- L'operatore () può restituire qualsiasi tipo
- **Esempio**

```
class Matrice {
   public:
        .
        .
     int& operator()(int i, int j){return m[i][j];}

    private:
       int rows, cols;
       int **m;
};
```

return by reference


```
int main() {
  Matrice m;
  int a, b;
      .
      .

  a = m(1,2); // a DESTRA dell'assegnazione (r-value)
      .
      .
  m(1,2) = b; // a SINISTRA dell'assegnazione (l-value)
      .
      .
  return 0;
}
```


UNIVERSITÀ DEGLI STUDI DI CASSINO E DEL LAZIO MERIDIONALE
SOL PER NOCTEM
1979

Uso di const

Il qualificatore const

- Il qualificatore **const**, nel suo uso più semplice, rende non modificabile il valore assegnato ad una variabile in fase di dichiarazione:

  const int MAXSTRING = 100;

- In pratica, il compilatore segnala come errore tutte le espressioni in cui si tenta di modifcare MAXSTRING.

- In C++ le costanti possono essere definite anche per mezzo della direttiva al preprocessore #define. **Esempio**

  #define MAXSTR 100

- In questo però, è il preprocessore a sostituire tutte le occorrenze di MAXSTR con il valore 100

- È preferibile l'uso di **const** poichè:
  - Il compilatore effettua type-checking, il preprocessore no.
  - Il preprocessore non impedisce eventuali, erronee, ridefinizioni
  - Non è possibile limitare lo scopo di MAXSTR

const per le variabili puntatore

- Nel caso dei puntatori il significato di **const** dipende dalla sua posizione:
  const int * const_ptr1;
  int const * const_ptr1;
  - In entrambi i casi, const_ptr1 è un puntatore ad una costante di tipo **int**

  int * **const** const_ptr2;
  - in questo caso, const_ptr2 è un puntatore costante ad una variabile di tipo **int**

  int const * **const** const_ptr3;
  - In quest ultimo caso, const_ptr3 è un puntatore costante ad una costante di tipo **int**

REGOLA GENERALE
In generale possiamo dire che il qualificatore **const** si applica:
– a quello che appare immediatamente alla sua sinistra
– a quello che appare immediatamente a sua destra, se a sinistra non c'è nulla

Variabili membro const

- Le variabili definite **`const`** all'interno di una classe hanno un valore costante per tutta la vita dell'oggetto
- Per ogni ogni oggetto, il valore della variabile può essere diverso
- All'interno del corpo del costruttore i membri const devono essere già inizializzati
- La loro inizializzazione avviene nella lista di inizializzazione del costruttore

NOTA
Quando si crea un **`const`** ordinario (non-static) all'interno
di una classe, non è possibile assegnare un valore iniziale nella specifica.

```
class Myclass {
   public:
     Myclass()
        .
        .
   private:
     const int num = 10;
        .
        .
  };
```

ERRORE!

```
class Myclass {
   public:
      Myclass(int n): num(n):
         .
         .
   private:
      const int num;
         .
         .
   };

int main ()
{
   Myclass m1(10), m2(20), m3(30);
         .
         .
         .
   return 0;
}
```

OK

NOTA

Possiamo in generale scrivere:

```
int i(0), a(1);
float f(3.14159);
```

Variabili static const

- Solo per le variabili const di tipo static è possibile l'inizializzazione nella dichiarazione della classe

```
class Myclass {
   public:
     Myclass()
        .
        .
   private:
     static const int num = 10; // OK
         .
         .
  };
```

```
int main ()
{
      .
      .
   MyClass::num;
      .
      .
}
```

Funzioni membro const

- Significato: *questa funzione non modifica lo stato dell'oggetto sul quale è chiamata*
- Sono le uniche funzioni che possono essere chiamate su oggetti dichiarati costanti
- **Esempio**

```
class Myclass {
   public:
     Myclass()
       .
       .
     void funz() const;
     int get_i() const{return i;}
       .
       .
   private:
     int i;
  };
```

Deve essere ripetuta anche qui

```
void Myclass::funz() const{
    i = 10; // ERRORE!
    return;
  }
```

Restituzione per valore

```
class Myclass {
   public:
     Myclass()
        .
        .
     C funz(); // può essere anche l-value

        .
        .
  };
```

```
Myclass m;
C c, c2;
    .
    .
c2 = m.funz(); //OK
m.funz()= c; // OK
    .
    .
```

Restituzione di const

```
class Myclass {
   public:
     Myclass()
        .
        .
     const C funz(); // NO l-value

        .
        .
        .
  };
```

```
Myclass m;
C c, c2;
    .
    .
c2 = m.funz(); // OK
m.funz()= c; // ERRORE
    .
    .
```

Restituzione di reference const


```
class Myclass {
   public:
     Myclass()
        .
        .
     const C& funz(); // NO l-value

        .
        .
        .
  };
```

```
Myclass m;
C c, c2;
    .
    .
c2 = m.funz(); //OK
m.funz()= c; // ERRORE
    .
    .
```

Uso di const: esempio

```
class Myclass {
   public:
     Myclass()
        .
        .
     const C& funz(const C& c) const;

        .
        .
        .
  };
```

La classe string

La classe string

- Per la gestione delle stringhe, il C++ mette a disposizione la classe `string`
- La lunghezza della stringa non deve essere dichiarata a priori, poichè è gestita in maniera automatica dalla classe
- Le operazioni sulle stringhe vengono fatte per mezzo degli overloading dei seguenti operatori:

=, <, >, ==, <=, >=, !=, <<, >>, +, +=

Esempi

```
#include <iostream>
#include <string>

using namespace std;
int main()
{

  string s1("alfa"), s2("beta"), s3("omega"), s4, s5;
  char *C_str = "ciao";  // stringa in stile C

  s4 = s1;       // assegnazione tra stringhe

  s4 = s1 + s2; // assegna a s4 il concatenamento di
                   s1 e s2
  s4+= s3;      // concatena s4 a s3

  s4 = s1 + "-" + s2; // quanto vale ora s4 ???
}
```

assegnazione e concatenazione

```
s4 = s1;          // assegnazione tra stringhe

s4 = s1 + s2    // assegna a s4 il concatenamento di s1 e s2

s4+= s3;        //concatena s4 a s3

s4 = s1 + "-" + s2;

s1 = "questa è una stringa chiusa dal carattere nullo";

s3 = s1 + "pippo";

s3 = "pippo" + s1;

s4 = C_str;  // si possono usare stringhe classiche.
```

Operazioni di confronto e I/O

```
   // Operazioni di confronto:
  if (s1 > s2)
    cout<<endl<<"s2 precede s1";

  if (s1 == s2)
    cout<<endl<<”s1 è uguale a s2”;

  // Operazioni di I/O:
  cout<<endl<<”introdurre una stringa: ”;
  cin>> s4;
  cout<<endl<<s5;
}
```

Osservazioni

- La classe `string` dimensiona automaticamente l'array di char che memorizza la stringa:
  - quando si assegnano o concatenano due o più stringhe, le dimensioni della stringa destinazione cresceranno automaticamente per contenere la nuova stringa.
- La gestione automatica e controllata delle dimensioni elimina tutti gli errori delle gestione delle stringhe del C dovuti allo sfondamento degli array.

Stringhe: algoritmo di confronto

- n =1 (0 in C++)
- si confrontano i simboli nella posizione n-esima della stringa:
  - se i simboli sono uguali, si passa alla posizione successiva della stringa (n → n+1)
  - se questi sono diversi, il loro ordine è l'ordine delle stringhe
  - se una delle due stringhe non possiede l'elemento n-esimo, allora è minore dell'altra e l'algoritmo termina
  - se entrambe le stringhe non possiedono l'elemento n-esimo, allora sono uguali e l'algoritmo termina