1

Operator overloading

2

Operator overloading

Tipo 1: funzione esterna

[return-type] operatorXX([return-type] param_sx, [return-type] param_dx);

Esempio:

Integer & operator+(Integer sx,Integer dx);

Uso:

```
Integer a,b,c;
a=b+c;
```

Operator overloading

```
#include <iostream.h>
class Integer{
    int x;
    public:
    void setValue(int v) {x=v;}
    int getValue() {return x;}
};
Integer & operator+(Integer sx,Integer dx) {
   Integer *r=new Integer;
   int a=  sx.getValue();
   int b=  dx.getValue();
   r->setValue(a+b);
   return *r;
}
```

```
int main()
{
     Integer a;
     Integer b;
     a.setValue(1);
     b.setValue(2);
     Integer c=a+b;
     cout<<c.getValue();
}
```

```
3
```

Operator overloading

```
#include <iostream.h>
class Integer{
    int x;
    public:
    void setValue(int v) {x=v;}
    int getValue() {return x;}
};
Integer & operator+(Integer sx,Integer dx) {
   Integer & r= *(new Integer);
   int a=  sx.getValue();
   int b=  dx.getValue();
   r.setValue(a+b);
   return r;
}
```

```
int main()
{
     Integer a;
     Integer b;
     a.setValue(1);
     b.setValue(2);
     Integer c=a+b;
     cout<<c.getValue();
}
```

```
3
```

5

Operator overloading

Tipo 2: member function

[return-type] operatorXX([return-type] param_dx);

NOTA: Param sx è implicito, coincide con "this"

Esempio:

Integer & Integer::operator+(Integer dx);

Uso:

Integer a,b,c;

a=b+c;

Operator overloading

```
#include <iostream.h>
class Integer{
    int x;
    public:
    void setValue(int v) {x=v;}
    int getValue() {return x;}
    Integer & operator+(Integer dx);
};
Integer & Integer::operator+(Integer dx) {
   Integer *r=new Integer;
   int a=  this->getValue();
   int b=  dx.getValue();
   r->setValue(a+b);
   return *r;
}
```

```
int main()
{
    Integer a;
    Integer b;
    a.setValue(1);
    b.setValue(2);
    Integer c=a+b;
    cout<<c.getValue();
}
```

```
3
```

Operator overloading

```
#include <iostream.h>
class Integer{
    int x;
    public:
    void setValue(int v) {x=v;}
    int getValue() {return x;}
    Integer & operator+(Integer dx);
};
Integer & Integer::operator+(Integer dx) {
   Integer & r=*(new Integer);
   int a=  this->getValue();
   int b=  dx.getValue();
   r.setValue(a+b);
   return r;
}
```

```
int main()
{
    Integer a;
    Integer b;
    a.setValue(1);
    b.setValue(2);
    Integer c=a+b;
    cout<<c.getValue();
}
```

```
3
```

8

Self-printing

9

Self-printing

Pila * p=new Pila;
cout<<*p;

Pila p;
cout<<p;

```
ostream & operator<<(ostream &sx, const Pila &dx) {
    sx <<"=================="<< endl;
    sx << "size = "<<dx.size<<endl;
    sx << "defGsize = "<<dx.defGsize<<endl;
    sx << "marker = "<<dx.marker <<endl;
    for (int k=0;k<dx.marker;k++) sx << "["
       <<(dx.contenuto[k])<<"]";
    sx << endl;
    sx <<"=================="<< endl;
    return sx;
}
```

dove va definita la funzione?

10

Self-printing

Ma se è esterna, come può avere accesso
alle variabili di istanza (es. sx.size)?

```
class Pila {
  ...
  friend ostream & operator<<(ostream &sx, const Pila &dx);
}
```

- "friend" somiglia alla visibilità di tipo package
- String toString() è la soluzione per il self-printing in Java

11

Self-printing

Permette la concatenazione

```
ostream & operator<<(ostream &sx, const Pila &dx) {
    sx <<"=================="<< endl;
    sx << "size = "<<dx.size<<endl;
    sx << "defGsize = "<<dx.defGsize<<endl;
    sx << "marker = "<<dx.marker <<endl;
    for (int k=0;k<dx.marker;k++) sx << "["
       <<(dx.contenuto[k])<<"]";
    sx << endl;
    sx <<"=================="<< endl;
    return sx;
}
```

Pila p,q;
cout<<p<<q;

12

Confronto

Pila p;
Pila q;
if (q==p) ...

```
bool Pila::operator==(Pila & cmp){
  #ifdef DEBUG
  cout<<"Pila::operator==";
  #endif
  if (this == &cmp) return true;
  if (marker!=cmp.marker) return false;
  for (int k=0; k<marker;k++)
      if (contenuto[k]!=cmp.contenuto[k]) return false;
  return true;
}
```

- in Java il problema di == non si pone!
- boolean equals(Object k)

Confronto

```
int main() {
  Pila c;
  for (int k=0;k<5;k++)
    c.inserisci(k);
  Pila d
  for (int s=0;s<5;s++)
    d.inserisci(s);
  cout<<(d==c ? "true" :"false")<<endl;
  d.inserisci(1+d.estrai());
  cout<<(d==c ? "true" :"false")<<endl;
  return 0;
}
```

```
Pila()
Pila()
Pila::operator==
true
Pila::operator==
false
~Pila()
~Pila()
```

Confronto

```
int main() {
  Pila * c= new Pila();
  for (int k=0;k<5;k++)
    c->inserisci(k);
  Pila * d=new Pila();
  for (int s=0;s<5;s++)
    d->inserisci(s);
  if (c==d) cout<<"identiche!";
  if (*c==*d) cout<<"uguali!";
}
```

Qual è l'output?

15

String

```
class String {
  private:
  // VARIABILI DI ISTANZA
        char * base;
        int length;
  ...
```

16

String: Esercizio 1

Scrivere e testare i metodi di base:

a. i costruttori String(), String(char * s),
b. il distruttore.
c. l'operatore di output (self-print)
d. l'operatore di confronto ==

Semantica:
due String sono uguali se e' uguale il contenuto della stringa (char*) che esse possiedono.

17

String: Esercizio 2

Scrivere e testare i metodi di "business"

1. int lenght();
2. bool compare(String & w);
3. String & append(String & w);
4. int charAt(char c);