1

Assegnazione

2

```
Pila()
==================
size = 6
defGsize = 3
marker = 5
[0][1][2][3][4]
==================
Pila()
==================
size = 6
defGsize = 3
marker = 5
[0][1][2][3][5]
==================
==================
size = 6
defGsize = 3
marker = 5
[0][1][2][3][5]
==================
~Pila()
~Pila()
```

Assegnazione

```
int main() {
  Pila c;
  for (int k=0;k<5;k++)
    c.inserisci(k);
  cout<<c;
  Pila d;
  d=c;
  d.inserisci(1+d.estrai());
  cout<<d;
  cout<<c;
  return 0;
}
```

Shallow copy...

3

Assegnazione

```
Pila p;
Pila q;
q=p;
```

supporto alla assegnazione multipla

```
Pila & Pila::operator=(Pila & from){
  if (this == &from) return *this;
  size=from.size;
  defGsize=from.defGsize;
  delete []contenuto;
  contenuto=new int[from.size];
  for (int k=0; k<=from.marker;k++)
      contenuto[k]=from.contenuto[k];
  marker=from.marker;
  return *this;
}
```

Deep copy...

• in Java il problema non si pone!

4

Assegnazione

```
Pila c;
Pila d;
Pila e;
…
//assegnazione multipla
e=d=c;
…
//autoassegnazione
c=c;
…
//autoassegnazione
Pila *cp=&c;
c=*cp;
```

5

```
Pila()
==================
size = 6
defGsize = 3
marker = 5
[0][1][2][3][4]
==================
Pila()
==================
size = 6
defGsize = 3
marker = 5
[0][1][2][3][5]
==================
==================
size = 6
defGsize = 3
marker = 5
[0][1][2][3][4]
==================
~Pila()
~Pila()
```

Assegnazione

```
int main() {
  Pila c;
  for (int k=0;k<5;k++)
    c.inserisci(k);
  cout<<c;
  Pila d;
  d=c;
  d.inserisci(1+d.estrai());
  cout<<d;
  cout<<c;
  return 0;
}
```

6

```
Pila()
==================
size = 6
defGsize = 3
marker = 5
[0][1][2][3][4]
==================
Pila()
==================
size = 6
defGsize = 3
marker = 5
[0][1][2][3][5]
==================
==================
size = 6
defGsize = 3
marker = 5
[0][1][2][3][5]
==================
~Pila()
~Pila()
```

Pila d=c; è diverso da Pila d; d=c;

```
int main() {
  Pila c;
  for (int k=0;k<5;k++)
    c.inserisci(k);
  cout<<c;
  Pila d=c;
  d.inserisci(1+d.estrai());
  cout<<d;
  cout<<c;
  return 0;
}
```

7

Copy Constructor

```
Pila (Pila & from){
  size=from.size;
  defGsize=from.defGsize;
  contenuto=new int[from.size];
  for (int k=0; k<=from.marker;k++)
      contenuto[k]=from.contenuto[k];
  marker=from.marker;
}
```

- in Java il problema non si pone!

8

```
Pila()
==================
size = 6
defGsize = 3
marker = 5
[0][1][2][3][4]
==================
Pila()
==================
size = 6
defGsize = 3
marker = 5
[0][1][2][3][5]
==================
==================
size = 6
defGsize = 3
marker = 5
[0][1][2][3][4]
==================
~Pila()
~Pila()
```

Copy Constructor

```
int main() {
  Pila c;
  for (int k=0;k<5;k++)
    c.inserisci(k);
  cout<<c;
  Pila d=c;
  d.inserisci(1+d.estrai());
  cout<<d;
  cout<<c;
  return 0;
}
```

9

Copy constructor

- Ma a parte Pila d=c; a che serve?

- Pila c; Pila * s=new Pila(c);
- Pila *c=new Pila; Pila *s=new Pila(*c);
- Pila *c=new Pila; Pila s=*c;

Ma soprattutto....

Pila c; f(c);

10

```
void f(Pila s) {
   s.estrai();
   s.inserisci(3);
   cout<<"s"<<w;
}

int main() {
   Pila w(5);
   w.inserisci(1);
   cout<<"w"<<w;

   f(w);
   cout<<"w"<<w;
}
```

```
Pila(int)
entro in inserisci
w==================
size = 5
defaultGrowthSize = 5
marker = 1
[1]
==================
entro in estrai
entro in inserisci
s==================
size = 5
defaultGrowthSize = 5
marker = 1
[3]
==================
~Pila
w==================
size = 5
defaultGrowthSize = 5
marker = 1
[8126648]
==================
~Pila
```

Usa il default Copy-constructor

Output di f()

Senza Copy Constructor

```
[main] E:\cpp2\Pila6.exe 1037 (0) handle_exceptions: Exception: STATUS VIOLATION
[main] Pila6 1037 (0) handle_exceptions: Dumping Stack trace to Pila6
```

11

```
void f(Pila s) {
    s.estrai();
    s.inserisci(3);
    cout<<"s"<<w;
}

int main() {
    Pila w(5);
    w.inserisci(1);
    cout<<"w"<<w;

    f(w);
    cout<<"w"<<w;
}
```

```
eseguo il costruttore Pila(int)
entro in inserisci
w==================
size = 5
defaultGrowthSize = 5
marker = 1
[1]
==================
Entro nel copy constructor
entro in estrai
entro in inserisci
s==================
size = 5
defaultGrowthSize = 5
marker = 1
[3]
==================
entro in distruggi
w==================
size = 5
defaultGrowthSize = 5
marker = 1
[1]
==================
entro in distruggi
```

Usa il nostro Copy-constructor

Output di f()

Con Copy Constructor

12

Attenzione alle chiamate implicite

constructor,
copy constructor
e destructor

13

E se non volessi permettere il passaggio per copia?

```
class Pila {
        …
private:
        Pila(Pila & to) ;
        …
} ;
```

```
int main() {
  Pila w(5);
  w.inserisci(1);
  cout<<"w";w.stampaStato();
  f(w);
  cout<<"w";w.stampaStato();
}
```

Alla riga f(w); il compilatore segnala:
[C++ Error] Pila2.cpp(114): E2247 'Pila::Pila(Pila &)' is not accessible.

14

E se volessi fare il passaggio per copia in Java?


```
Pila w=new Pila();
…
f(w.clone());
```

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 assegnazione =
e. 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);

String: Esercizio 3

Visto che abbiamo scritto metodi collegato con gli operatori di:

assegnazione =

uguaglianza ==

output <<

provare (in maniera analoga) a definire gli operatori di

input >>

concatenazione +