# SEZIONE 2
# Introduzione al copy constructor e ad operator overloading

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

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

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

```
eseguo il costruttore 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]
==================
entro in distruggi
w==================
size = 5
defaultGrowthSize = 5
marker = 1
[8126648]
==================
entro in distruggi
```

Usa il default Copy-constructor

Output di f()

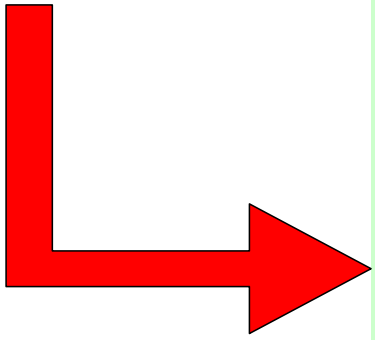
```
[main] E:\cpp2\Pila6.exe 1037 (0) handle_exceptions: Exception: STATUS VIOLATION
[main] Pila6 1037 (0) handle_exceptions: Dumping Stack trace to Pila6
```

Cosa è successo?
Il sistema ha cercato di fare una copia di Pila…
ma non ha usato il nostro metodo "copia!"

(Shallow copy vs. deep copy)

```
Pila * Pila::copia(){
  Pila * to=new Pila(size);
  to->defaultGrowthSize=defaultGrowthSize;
  for (int k=0; k<=marker;k++) {
    to->contenuto[k]=contenuto[k];
  }
  to->marker=marker;
  return to;}
```

Re-implementazione di copia – fase 2



```
Pila::Pila(Pila & from) {
#ifdef DEBUG
  cout<<"entro in copy constructor"<<endl;
#endif
  size=from.size;
  defaultGrowthSize=from.defaultGrowthSize;
  contenuto=new int[from.size];
  for (int k=0; k<= from.marker;k++) {
    contenuto[k]=from.contenuto[k];
  }
  marker=from.marker;
}
```

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

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

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

```
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()

# Attenzione alle chiamate implicite al constructor, copy constructor e al destructor

# 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.

# Operatore =

Copia una struttura sull'altra. ATTENZIONE!

```
int main() {
   Pila s(5);
   for (int k=1; k<10;k++)
           s.inserisci(k);
   cout<<"s";s.stampaStato();

   Pila w(15);
   cout<<"w";w.stampaStato();
   w=s;
   cout<<"w";w.stampaStato();
   for (int k=1; k<5;k++)
           cout<<s.estrai()<<endl;
    for (int k=1; k<4;k++)
           s.inserisci(10+k);
  cout<<"s"; s.stampaStato();
   cout<<"w"; w.stampaStato();
}
```

```
entro in constructor
...
s==================
size = 10
defaultGrowthSize = 5
marker = 9
[1][2][3][4][5][6][7][8][9]
==================
entro in constructor
w==================
size = 15
defaultGrowthSize = 15
marker = 0

==================
w==================
size = 10
defaultGrowthSize = 5
marker = 9
[1][2][3][4][5][6][7][8][9]
==================
...
```

```
...
s==================
size = 10
defaultGrowthSize = 5
marker = 8
[1][2][3][4][5][11][12][13]
==================
w==================
size = 10
defaultGrowthSize = 5
marker = 9
[1][2][3][4][5][11][12][13][9]
==================
entro in destroy
entro in destroy
```

# Operatore =

ECCEZIONE!

In fase di DEFINIZIONE

chiama il copy constructor

```
int main() {
   Pila s(5);
   for (int k=1; k<10;k++)
           s.inserisci(k);
   cout<<"s";s.stampaStato();

   Pila w=s;
   cout<<"w";w.stampaStato();
   for (int k=1; k<5;k++)
           cout<<s.estrai()<<endl;
    for (int k=1; k<4;k++)
           s.inserisci(10+k);
  cout<<"s"; s.stampaStato();
   cout<<"w"; w.stampaStato();
}
```

```
entro in constructor
...
s==================
size = 10
defaultGrowthSize = 5
marker = 9
[1][2][3][4][5][6][7][8][9]
==================
entro in copy constructor
w==================
size = 10
defaultGrowthSize = 5
marker = 9
[1][2][3][4][5][6][7][8][9]
==================
...
```

```
...
s==================
size = 10
defaultGrowthSize = 5
marker = 8
[1][2][3][4][5][11][12][13]
==================
w==================
size = 10
defaultGrowthSize = 5
marker = 9
[1][2][3][4][5][6][7][8][9]
==================
entro in destroy
entro in destroy
```

# Operatore =

Facciamolo comportare bene!

```
Pila & Pila::operator=(Pila & from){
  if (this = = &from) return *this;
#ifdef DEBUG
  cout<<"entro in operator="<<endl;
#endif
  size=from.size;
  defaultGrowthSize=from.defaultGrowthSize;
  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;
}
```

# Re-imple-mentazione di copy – fase 3

# Overloading dell'operatore =

```
int main() {
   Pila s(5);
   for (int k=1; k<10;k++)
          s.inserisci(k);
   cout<<"s";s.stampaStato();

   Pila w(15);
   cout<<"w";w.stampaStato();
   w=s;
   cout<<"w";w.stampaStato();
   for (int k=1; k<5;k++)
          cout<<s.estrai()<<endl;
    for (int k=1; k<4;k++)
          s.inserisci(10+k);
  cout<<"s"; s.stampaStato();
  cout<<"w"; w.stampaStato();
}
```

```
entro in constructor
…
s==================
size = 10
defaultGrowthSize = 5
marker = 9
[1][2][3][4][5][6][7][8][9]
==================
entro in constructor
w==================
size = 15
defaultGrowthSize = 15
marker = 0

==================
entro in operator=
w==================
size = 10
defaultGrowthSize = 5
marker = 9
[1][2][3][4][5][6][7][8][9]
==================
…
```

```
…
s==================
size = 10
defaultGrowthSize = 5
marker = 8
[1][2][3][4][5][11][12][13]
==================
w==================
size = 10
defaultGrowthSize = 5
marker = 9
[1][2][3][4][5][6][7][8][9]
==================
entro in destroy
entro in destroy
```

# Pila.h versione 7

```
class Pila {
   private:
          int size;
          int defaultGrowthSize;
          int marker;
          int * contenuto;
          void cresci(int increment);
    public:
          Pila(int initialSize) ;
          Pila(Pila & to) ;
          Pila & Pila::operator=(Pila & from){
          ~Pila() ;
          void inserisci(int k) ;
          int estrai() ;
          void stampaStato() ;
} ;
```

# Concetto di friend

# Operatore <<

Accesso al canale di output

# Re-implementazione di stampaStato
## Overloading dell'operatore <<

```
ostream & operator<<(ostream &sx, const Pila &dx) {
#ifdef DEBUG
  cout<<"entro in operator<<"<<endl;
#endif
        sx <<"====================="<< endl;
        sx << "size = "<<dx.size<<endl;
        sx << "defaultGrowthSize = "<<dx.defaultGrowthSize<<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.h versione 8

```
class Pila {
  private:
        int size;
        int defaultGrowthSize;
        int marker;
        int * contenuto;
        void cresci(int increment);
   public:
        Pila(int initialSize) ;
        Pila(Pila & to) ;
        Pila & Pila::operator=(Pila & from){
        ~Pila() ;
        void inserisci(int k) ;
        int estrai() ;
        friend ostream & operator<<(ostream &sx, const Pila &dx);
} ;
```

# uguaglianza
# e identita'

Due variabili sono UGUALI se hanno lo stesso valore

Due variabili sono IDENTICHE se coincidono

```
int k=3; int j; j=k;
int * pk1=&k;  int * pk2;  pk2=pk1;
```

k e j sono UGUALI

*pk1 e *pk2 sono IDENTICHE

```
bool  Pila::equals(Pila & s) {
#ifdef DEBUG
  cout<<"entro in equals"<<endl;
#endif
  if (size!=s.size) return false;
  if (marker!=s.marker) return false;
  if (defaultGrowthSize!=s.defaultGrowthSize) return false;
  for (int k=0;k<marker;k++)
          if (contenuto[k]!=s.contenuto[k]) return false;
  return true;
}

bool Pila::operator==(Pila & from){
   return equals(from);
}
```

```
int main() {
   Pila * s= new Pila(5);
   for (int k=1; k<10;k++) s->inserisci(k);
   cout<<"==s=="<<endl<<*s;
 //cout<<"==s=="<<endl<<s; NO!
   Pila *w= new Pila(15);
   cout<<"==w=="<<endl<<*w;
   //w=s; NO!!
   *w=*s;
   cout<<"==w=="<<endl<<*w;
   cout<<"w->equals(s): "<<w->equals(*s)<<endl;
   cout<<"w==s: "<<(w==s)<<endl;
   cout<<"*w==*s: "<<(*w==*s)<<endl;
}
```

```
entro in constructor
entro in inserisci
…
entro in inserisci
==s==
==================
size = 10
defaultGrowthSize = 5
marker = 9
[1][2][3][4][5][6][7][8][9]
==================

…
entro in equals
w->equals(s): 1
w==s: 0
entro in equals
*w==*s: 1
```

# Pila.h versione 9

```
class Pila {
            int size;
            int defaultGrowthSize;
            int marker;
            int * contenuto;
            void cresci(int increment);
   public:
            Pila(int initialSize) ;
            Pila(Pila & to) ;
            Pila & Pila::operator=(Pila & from){
           ~Pila() ;
            void inserisci(int k) ;
            int estrai() ;
            bool equals(Pila &s);
            bool Pila::operator= =(Pila & from);
            friend ostream & operator<<(ostream &sx, const Pila &dx);
} ;
```