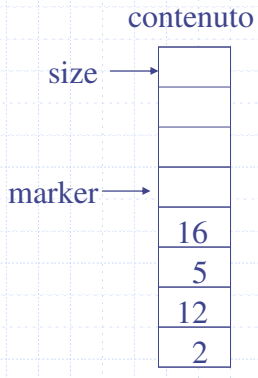
1

Il nostro esempio guida: La costruzione di uno stack

2

```
#include <iostream.h>
#include <cassert>
#define DEBUG

struct Pila {
  int size;
  int defaultGrowthSize;
  int marker;
  int * contenuto;
} ;
```



3

Pila.h

```
struct Pila {
  int size;
  int defaultGrowthSize;
  int marker;
  int * contenuto;
} ;

Pila  * crea(int initialSize) ;
void distruggi(Pila * s) ;
Pila * copia(Pila * from) ;
void cresci(Pila *s, int increment);
void inserisci(Pila *s, int k) ;
int estrai(Pila *s) ;
void stampaStato(Pila *s) ;
```

4

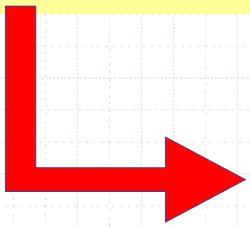
Pila.h versione 2

```
struct Pila {
  int size;
  int defaultGrowthSize;
  int marker;
  int * contenuto;
  int estrai() ;
} ;
Pila  * crea(int initialSize) ;
void distruggi(Pila * s) ;
Pila * copia(Pila * from) ;
void cresci(Pila *s, int increment);
void inserisci(Pila *s, int k) ;
// int estrai(Pila *s) ; vecchia versione
void stampaStato(Pila *s) ;
```

5

```
int estrai(Pila *s) {
//estrai l'ultimo valore
#ifdef DEBUG
  cout<<"entro in estrai"<<endl;
#endif
  assert(s->marker>0);
  return s->contenuto[--(s->marker)];
}
```

Re-implementazione di estrai



```
int estrai() {
//estrai l'ultimo valore
#ifdef DEBUG
  cout<<"entro in estrai"<<endl;
#endif
  assert(this->marker>0);
  return this-> contenuto[--(this-> marker)];
}
```

6

Re-implementazione del main

```
int main() {
  Pila  * s=crea(5);
    cout<<"s";stampaStato(s);
    for (int k=1; k<10;k++) inserisci(s,k);
    cout<<"s"; stampaStato(s);
    Pila * w = copia(s);
    cout<<"w"; stampaStato(w);
    for (int k=1; k<8;k++)
      //cout<<estrai(s)<<endl;
      cout<<s->estrai()<<endl;
    …
}
```

7

Re-implementazione
di estrai: dove scrivo il codice?

```
struct Pila {
   int size;
   int defaultGrowthSize;
   int marker;
   int * contenuto;
   int estrai() {
        //estrai l'ultimo valore
#ifdef DEBUG
        cout<<"entro in estrai"<<endl;
#endif
        assert(this->marker>0);
        return this->contenuto[--(this->marker)];
    }
};
```

8

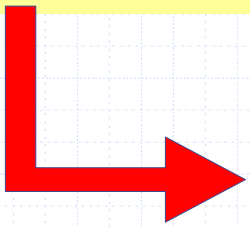
Re-implementazione
di estrai: dove scrivo il codice?

```
struct Pila {
   int size;
   int defaultGrowthSize;
   int marker;
   int * contenuto;
   int estrai();
};
int Pila::estrai() {
        //estrai l'ultimo valore
#ifdef DEBUG
        cout<<"entro in estrai"<<endl;
#endif
        assert(this->marker>0);
        return this->contenuto[--(this->marker)];
}
```

9

```
int estrai(Pila *s) {
//estrai l'ultimo valore
#ifdef DEBUG
  cout<<"entro in estrai"<<endl;
#endif
  assert(s->marker>0);
  return s->contenuto[--(s->marker)];
}
```

Re-implementazione di estrai con this implicito



```
int estrai() {
//estrai l'ultimo valore
#ifdef DEBUG
  cout<<"entro in estrai"<<endl;
#endif
  assert(marker>0);
  return contenuto[--(marker)];
}
```

10

This implicito

```
struct Pila {
   int  marker;
   ...
   int f();
   void setMarker(int m);
};
int Pila::f() {
   int marker;    //variabile locale
   this->marker; //variabile di istanza
}

void Pila::setMarker(int marker) {
   this->marker=marker;
}
```

11

Terminologia

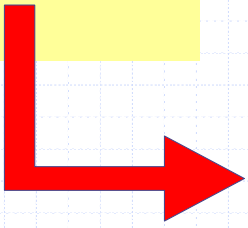
Funzioni:
- funzioni membro (member functions)
- metodi (methods)

Variabili:
- dati membro (member data)
- variabili di istanza (instance variables)

12

Re-implementazione di crea

```
Pila  * crea(int size) {
  Pila  * s= new Pila ;
  s->size=size;
  s->defaultGrowthSize=size;
  s->marker=0;
  s-> contenuto=new int[size];
 return s;
}
```



```
Pila::Pila(int size) {
  this->size=size;
  defaultGrowthSize=size;
  marker=0;
  contenuto=new int[size];
}
```

"Il costruttore"

13

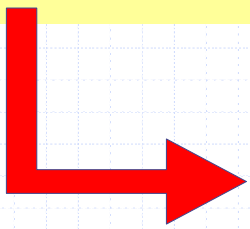
Re-implementazione del main

```
int main() {
   Pila  * s=new Pila(5); // OLD: =crea(5)
    cout<<"s";s->stampaStato();
    for (int k=1; k<10;k++) s->inserisci(k);
    cout<<"s"; s->stampaStato();
    Pila * w = s->copia();
    cout<<"w"; w->stampaStato();
    for (int k=1; k<8;k++)
      cout<< s->estrai()<<endl;
    cout<<"s"; s->stampaStato();
    delete s; // OLD: s->distruggi();
    cout<<"s"; s->stampaStato();
    for (int k=1; k<15;k++)
       cout<< w->estrai()<<endl;
    cout<<"w"; w->stampaStato();
}
```

14

Re-implementazione di distruggi

```
void Pila:: distruggi () {
//distruggi lo Pila
#ifdef DEBUG
  cout<<"entro in distruggi"<<endl;
#endif
  delete []contenuto;
  delete this;
}
```



```
Pila::~Pila() {
//distruggi lo Pila
#ifdef DEBUG
  cout<<"entro nel distruttore"<<endl;
#endif
  delete []contenuto;
  // NO! delete this;
}
```

"Il distruttore"

15

Re-implementazione del main

```
int main() {
   Pila  * s=new Pila(5); // OLD: =crea(5)
    cout<<"s";s->stampaStato();
    for (int k=1; k<10;k++) s->inserisci(k);
    cout<<"s"; s->stampaStato();
    Pila * w = s->copia();
    cout<<"w"; w->stampaStato();
    for (int k=1; k<8;k++)
      cout<< s->estrai()<<endl;
    cout<<"s"; s->stampaStato();
    delete s; // OLD: s->distruggi();
    cout<<"s"; s->stampaStato();
    for (int k=1; k<15;k++)
       cout<< w->estrai()<<endl;
    cout<<"w"; w->stampaStato();
}
```

16

Ruolo di new e delete

```
new:
      alloca memoria
      chiama il costruttore (inizializzazione)

delete:
      chiama il distruttore (finalizzazione)
      dealloca la memoria
```

17

Pila.h versione 3

```
struct Pila {
   int size;
   int defaultGrowthSize;
   int marker;
   int * contenuto;
   Pila(int initialSize) ;
   ~Pila() ;
   Pila * copia() ;
   void cresci(int increment);
   void inserisci(int k) ;
   int estrai() ;
   void stampaStato() ;
} ;
```

Variabili di istanza, Dati membro

Metodi, Funzioni membro

18

Overloading dei metodi

Possono esistere piu' metodi con uguale nome!

La FIRMA (signature) di un metodo e' data da

NOME + tipo degli argomenti passati

(Il tipo del valore di ritorno non conta!)

Vale anche per i costruttori

19 Overloading del costruttore

```
Pila::Pila() { //crea una Pila
   cout<<"eseguo il costruttore Pila()"<<endl;
   int initialSize=10;
   this->size=initialSize;
   this->defaultGrowthSize=initialSize;
   this->marker=0;
   this->contenuto=new int[initialSize];}
//----------------------------------------
Pila::Pila(int initialSize) { //crea una Pila
    cout<<"eseguo il costruttore Pila(int)"
    <<endl;
    this->size=initialSize;
    this->defaultGrowthSize=initialSize;
    this->marker=0;
    this->contenuto=new int[initialSize];
}
```

20 Overloading del costruttore

```
void Pila::inizializza(int initialSize) {
   this->size=initialSize;
   this->defaultGrowthSize=initialSize;
   this->marker=0;
   this->contenuto=new int[initialSize];
}
//----------------------------------------
Pila::Pila() { //crea una Pila
   cout<<"eseguo il costruttore Pila()"<<endl;
   inizializza(15); //  valore di default
}
//----------------------------------------
Pila::Pila(int initialSize) { //crea una Pila
    cout<<"eseguo il costruttore Pila(int)"<<endl;
    inizializza(initialSize);
}
```

21 Overloading del costruttore – metodi inline

```
inline void Pila::inizializza(int initialSize) {
   this->size=initialSize;
   this->defaultGrowthSize=initialSize;
   this->marker=0;
   this->contenuto=new int[initialSize];
}
//-----------------------------------------
Pila::Pila() { //crea una Pila
   cout<<"eseguo il costruttore Pila()"<<endl;
   inizializza(15); //  valore di default
}
//-----------------------------------------
Pila::Pila(int initialSize) { //crea una Pila
    cout<<"eseguo il costruttore Pila(int)"<<endl;
    inizializza(initialSize);
}
```

22 Overloading del costruttore

Parametri con valori di default

```
Pila::Pila(int initialSize=10) { //crea una Pila
   cout<<"eseguo il costruttore Pila(int)"<<endl;
   this->size=initialSize;
   this->defaultGrowthSize=initialSize;
   this->marker=0;
   this->contenuto=new int[initialSize];
}
```

23

Pila.h versione 4

```
struct Pila {
        Pila(int initialSize) ;
        Pila();
        ~Pila() ;
         Pila * copia() ;
        void inserisci(int k) ;
        int estrai() ;
        void stampaStato() ;
    private:
        int size;
        int defaultGrowthSize;
        int marker;
        int * contenuto;
        void cresci(int increment);
} ;
```

24

Pila.h versione 5

```
class Pila {
        int size;
        int defaultGrowthSize;
        int marker;
        int * contenuto;
        void cresci(int increment);
    public:
        Pila(int initialSize) ;
        Pila();
        ~Pila() ;
        Pila * copy() ;
        void inserisci(int k) ;
        int estrai() ;
        void stampaStato() ;
} ;
```

25

Pila.h versione 6

```
struct Pila {
  private:
        int size;
        int defaultGrowthSize;
        int marker;
        int * contenuto;
        void cresci(int increment);
   public:
        Pila(int initialSize) ;
        Pila();
        ~Pila() ;
         Pila * copy() ;
        void inserisci(int k) ;
        int estrai() ;
        void stampaStato() ;
} ;
```

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