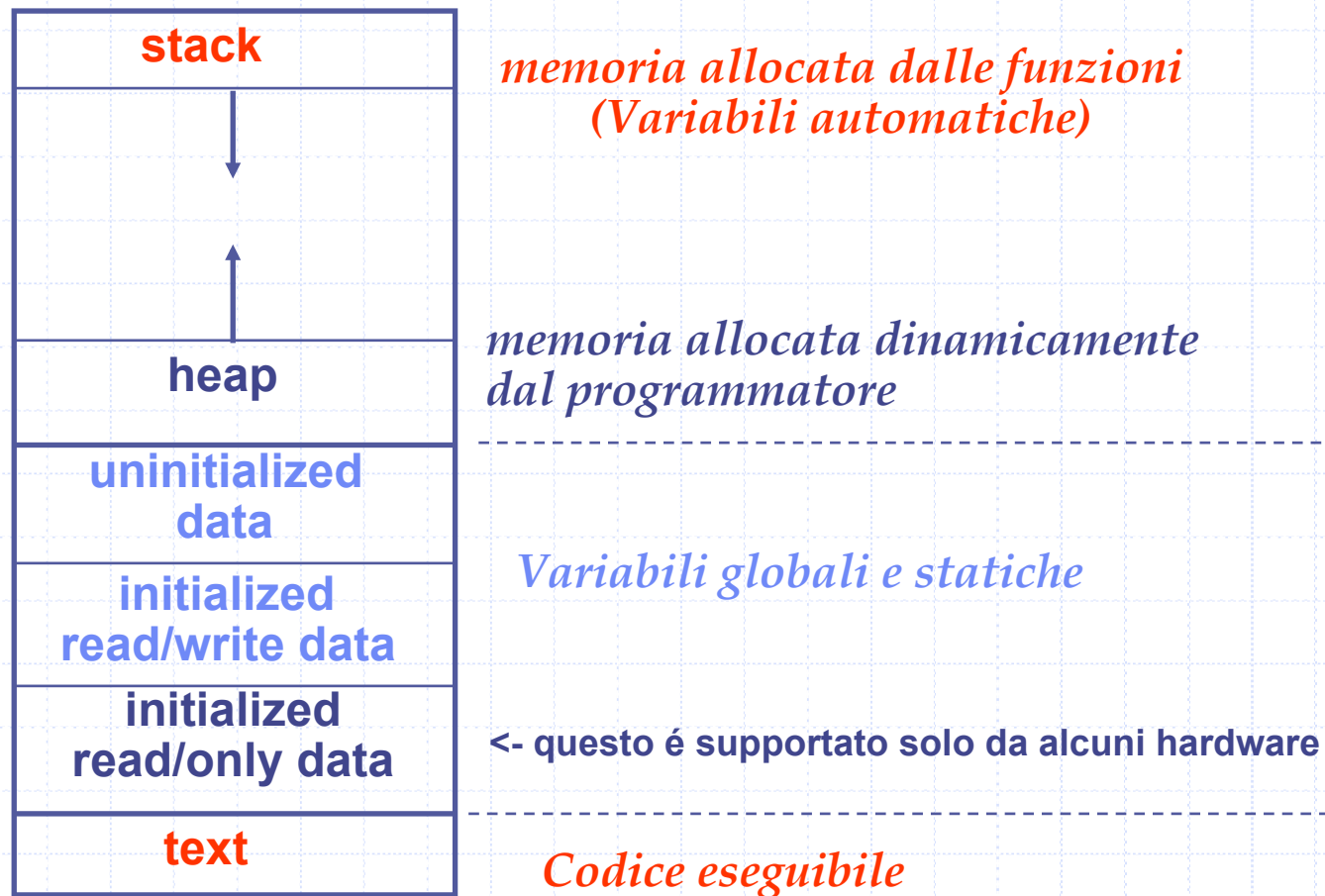


1

# Richiami di C++ di base



# Il modello di memoria



# Modularizzazione: Funzioni

Funzioni come "procedure parametrizzate"

```
tipo funzione(tipo argom1,...,tipo argomN)
{
    corpo della funzione
    return var;
}
```

# Modularizzazione: Funzioni

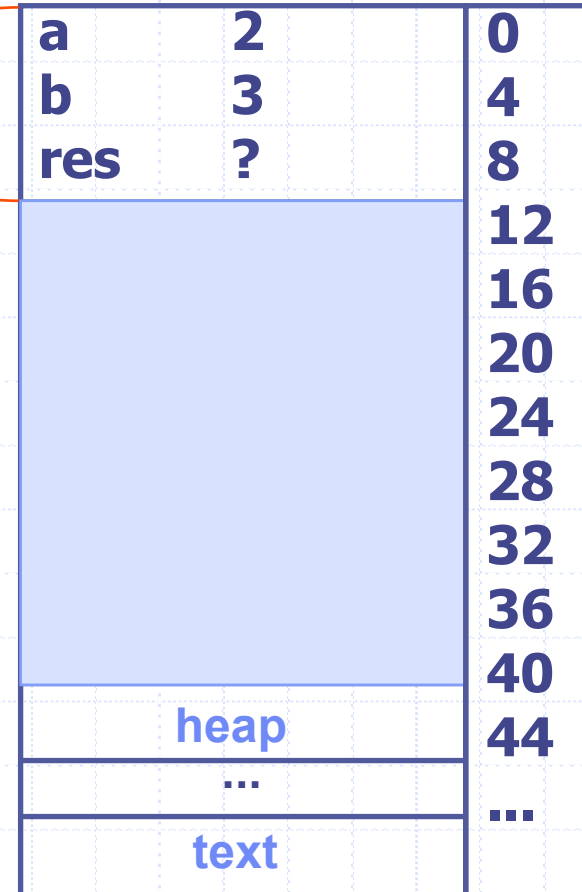
Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}

int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}

main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
    << res << "\n";
}
```

stack



main

# Modularizzazione: Funzioni

Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}

int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}

main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
    << res << "\n";
}
```

stack

a	2	0
b	3	4
res	?	8
a	3	12
b	2	16
res	0	20
k	0	24
		28
		32
		36
		40
		44
		...
		...
		...

main

prodotto

heap

...

text

# Modularizzazione: Funzioni

Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}

int prodotto(int b, int a) { stack
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}

main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
    << res << "\n";
}
```

a	2	0	<b>main</b>
b	3	4	
res	?	8	
a	3	12	<b>prodotto</b>
b	2	16	
res	0	20	
k	0	24	<b>somma</b>
a	0	28	
b	3	32	
res	?	36	
heap		40	
...		44	
text		...	

# Modularizzazione: Funzioni

## Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}

int prodotto(int b, int a) { stack
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}

main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
    << res << "\n";
}
```

a	2	0	<b>main</b>
b	3	4	
res	?	8	
a	3	12	<b>prodotto</b>
b	2	16	
res	<b>3</b>	20	
k	0	24	<b>somma</b>
a	0	28	
b	3	32	
res	<b>3</b>	36	
heap		40	
...		44	
text		...	

# Modularizzazione: Funzioni

Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}

int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}

main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
    << res << "\n";
}
```

stack

a	2	0
b	3	4
res	?	8
a	3	12
b	2	16
res	3	20
k	1	24
	0	28
	3	32
	3	36
		40
		44
	heap	
	...	
	text	

main

prodotto

# Modularizzazione: Funzioni

Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}

int prodotto(int b, int a) { stack
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}

main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
    << res << "\n";
}
```

a	2	0	<b>main</b>
b	3	4	
res	?	8	
a	3	12	<b>prodotto</b>
b	2	16	
res	3	20	
k	1	24	<b>somma</b>
a	<b>3</b>	28	
b	<b>3</b>	32	
res	3	36	
heap		40	
...		44	
text		...	

# Modularizzazione: Funzioni

## Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}

int prodotto(int b, int a) { stack
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}

main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
    << res << "\n";
}
```

a	2	0	<b>main</b>
b	3	4	
res	?	8	
a	3	12	<b>prodotto</b>
b	2	16	
res	3	20	
k	1	24	<b>somma</b>
a	3	28	
b	3	32	
res	<b>6</b>	36	
		40	
heap		44	
...		...	
text			

# Modularizzazione: Funzioni

## Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}

int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}

main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
    << res << "\n";
}
```

stack

a	2	0
b	3	4
res	?	8
a	3	12
b	2	16
res	6	20
k	1	24
	3	28
	3	32
	6	36
		40
		44
	heap	
	...	
	text	

main

prodotto

# Modularizzazione: Funzioni

## Esempio

```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}

int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}

main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
    << res << "\n";
}
```

stack

a	2	0
b	3	4
res	6	8
	3	12
	2	16
	6	20
	1	24
	3	28
	3	32
	6	36
		40
heap		44
...		...
text		

main

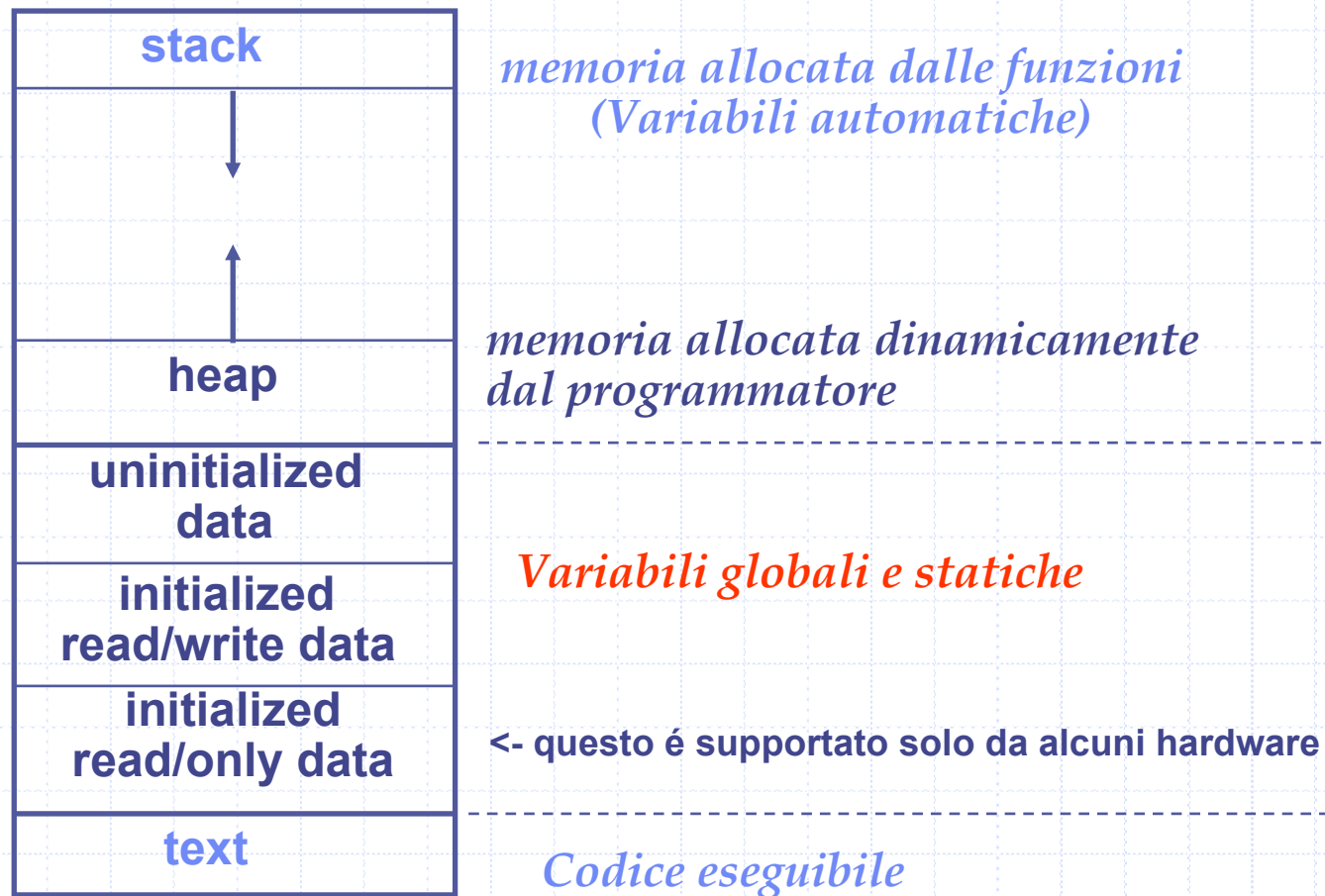
# Funzioni ricorsive

Una funzione può richiamare se stessa.

```
int fact(int n) {  
    if (n==0) return 1;  
    else return n*fact(n-1);  
}  
  
main(void) {  
    int n;  
    cout<<"dammi un numero\n";  
    cin >> n;  
    cout << "Il suo fattoriale vale "<<fact(n)<<"\n";  
}
```

Cosa avviene nello stack?

# Il modello di memoria



# Scope delle variabili

## Variabili globali

Nel seguente esempio a e' una variabile globale.

Il suo valore è visibile a tutte le funzioni.

**ATTENZIONE!** Le variabili globali vanno EVITATE a causa dei side-effects.

```
int a=5;
void f() {
    a=a+1;
    cout << "a in f: " << a << " - ";
    return;
}
main() {
    cout << "a in main:" << a << " - ";
    f();
    cout << "a in main: " << a << endl);
}
```

Output:

```
a in main: 5 - a in f: 6 - a in main: 6
```

# Scope delle variabili

## Variabili automatiche

Nel seguente esempio a e' una variabile automatica per la funzione f.  
Il suo valore è locale ad f.

```
int a=5;
void f() {
    int a=2, b=4;
    printf("(a,b) in f: (%d,%d) -",a,b);
    return;
}
main() {
    int b=6;
    printf("(a,b) in main: (%d,%d) -",a,b);
    f();
    printf("(a,b) in main: (%d,%d)\n",a,b);
}
```

### Output:

(a,b) in main: (5,6) - (a,b) in f: (2,4) - (a,b) in main: (5,6)

**ATTENZIONE!** Le variabili automatiche SCHERMANO le variabili globali.

# Quanto vale s?

```
void modifica(int s) {  
    s++;  
}  
main(void) {  
    int s=1;  
    modifica(s);  
    cout << "s=" << s << "\n";  
}
```

← A) "locale"

"globale" (B→

```
int s;  
int modifica() {  
    s++;  
    return s;  
}  
main(void) {  
    s=1;  
    modifica();  
    cout << "s=" << s << "\n";  
}
```

# Variabili globali

Le variabili globali sono "cattive"  
(almeno quanto il GOTO)!

perchè violano il principio della località della  
informazione (Principio di "Information hiding")

E' impossibile gestire correttamente progetti  
"grossi" nei quali si faccia uso di variabili globali.

Principio del **NEED TO KNOW**:

Ciascuno deve avere **TUTTE** e **SOLO** le  
informazioni che servono a svolgere il compito  
affidato

# Principi di Parna

Il committente di una funzione deve dare all'implementatore tutte le informazioni necessarie a realizzare la funzione, e NULLA PIÙ

L'implementatore di una funzione deve dare all'utente tutte le informazioni necessarie ad usare la funzione, e NULLA PIÙ

# Funzioni: problema #1

Come faccio a scrivere una funzione che modifichi le variabili del chiamante?

```
void incrementa(int x) {  
    x=x+1;  
}  
main(void) {  
    int a=1;  
    incrementa(a);  
    cout << "a=" a << "\n";  
}
```

Quanto vale a quando viene stampata?

I parametri sono passati per valore (copia)!

## Funzioni: problema #2

Come faccio a farmi restituire  
più di un valore da una funzione?

# Puntatori

## Operatore indirizzo: &

&a fornisce l'indirizzo della variabile a

## Operat. di dereferenziazione: \*

\*p interpreta la variabile p come un puntatore (indirizzo) e fornisce il valore contenuto nella cella di memoria puntata

```
main() {
    int a,b,c,d;
    int * pa, * pb;
    pa=&a; pb=&b;
    a=1; b=2;
    c=a+b;
    d=*pa + *pb;
    cout << a<<" "<<b<<" "<< c <<endl;
    cout << a <<" "<< *pb <<" "<< d <<endl;
}
```

stack

a	1	0
b	2	4
c	?	8
d	?	12
pa	0	16
pb	4	20

...

# Funzioni e puntatori

TRUCCO: per passare un parametro **per indirizzo**,  
passiamo per valore un puntatore ad esso!

```
void incrementa(int *px) {  
    *px=*px+1;  
}  
  
main(void) {  
    int a=1;  
    incrementa(&a);  
    cout<<a<<endl;  
}
```

stack

a	1	0
px	0	4
	?	8
	?	12
	?	16
	?	20

...

OUTPUT: 2

# Funzioni e puntatori

TRUCCO: per ottenere più valori di ritorno,  
passiamo degli indirizzi!

```
void minimax(int a1, int a2, int a3, int *pmin, int *pmax) {  
    *pmin=a1; *pmax=a1;  
    if (*pmin>a2) *pmin=a2; else if (*pmax<a2) *pmax=a2;  
    if (*pmin>a3) *pmin=a3; else if (*pmax<a3) *pmax=a3;  
}  
  
main(void) {  
    int a,b,c,d,min,max;  
    cout << "Dammi 3 numeri\n";  
    cin >> a >> b >> c;  
    minimax(a,b,c,&min,&max);  
    cout << "Il min vale "<<min<<" Il max vale "<<max<<"\n";  
}
```

# Tipi di dati

## Tipi di dati personalizzati

```
typedef float coordinata;  
coordinata z,t;
```

## Tipi di dati composti

```
struct punto {  
    coordinata x;  
    coordinata y;  
}
```

```
punto origine;  
origine.x=0.0;  
origine.y=0.0;
```

# Tipi di dati

## Tipi di dati personalizzati

```
typedef float coordinata;  
coordinata z,t;
```

## Tipi di dati composti

```
struct punto {  
    coordinata x;  
    coordinata y;  
}
```

```
punto origine;  
origine.x=0.0;  
origine.y=0.0;
```

# Puntatori a strutture

Operat. di dereferenziazione di struttura: ->

```
main()
{
    struct point {
        int x;
        int y;
        enum color {BLACK, BLUE, RED, GREEN};
    };
    struct point a, *pa;
    a.x = 3; a.y = 5; a.color=GREEN;
    pa = &a;
    cout<<a.x<<" "<<pa->y <<" "<<(*pa).color;
}
```

# Elementi di C++ di base

## Arrays (vettori)

# Array

Gli array sono collezioni di elementi omogenei

```
int valori[10];  
char v[200], coll[4000];
```

Un array di  $k$  elementi di tipo  $T$  in è un blocco di memoria contiguo di grandezza  
 $(k * \text{sizeof}(T))$

## Array - 2

Ogni singolo elemento di un array può essere utilizzato esattamente come una variabile con la notazione:

`valori[indice]`

dove indice stabilisce quale posizione considerare all'interno dell'array

# Limitazioni

- ◆ Gli indici spaziano sempre tra  $0$  e  $k-1$
- ◆ Il numero di elementi è fisso (deciso a livello di compilazione - *compile time*): non può variare durante l'esecuzione (a *run time*)
- ◆ Non c'è nessun controllo sugli indici durante l'esecuzione

# Catastrofe (potenziale)

```
....  
int a[10];  
a[256]=40;  
a[-12]=50;  
....
```

# Vettori

## Vettore Uni- dimensionale di interi

Base:

0012FF74 0012FF74

0012FF74 0

0012FF78 1

0012FF7C 2

0012FF80 3

0012FF74 0

0012FF78 1

0012FF7C 2

0012FF80 3

```
#include <iostream.h>
int main() {
    int v[4];
    int i,k;
    k=0;
    cout<<"Base:"<<endl
        <<&(v[0])<<" "<<v<<endl<<endl;
    for (i=0;i<4;i++) {
        v[i]=k++;
        cout<<&v[i]<<" "<<v[i]<<endl;
    }
    for (i=0;i<4;i++)
        cout<<(v+i)<<" "<< *(v+i)<<endl;
    return 0;
}
```

# Vettori

## Vettore Uni- dimensionale di double

Base :

0012FF64 0012FF64

0012FF64 0

0012FF6C 1

0012FF74 2

0012FF7C 3

0012FF64 0

0012FF6C 1

0012FF74 2

0012FF7C 3

```
#include <iostream.h>
int main() {
    double v[4];
    int i,k;
    k=0;
    cout<<"Base:"<<endl
        <<&(v[0])<<" "<<v<<endl<<endl;
    for (i=0;i<4;i++) {
        v[i]=k++;
        cout<<&v[i]<<" "<<v[i]<<endl;
    }
    for (i=0;i<4;i++)
        cout<<(v+i)<<" "<< *(v+i)<<endl;
    return 0;
}
```

# Vettori e funzioni

35



```
#include <iostream.h>
const int N=4;
void printLargest(int v[]) {
    // void printLargest(int *v){ è equivalente
    // void printLargest(int v[2]){ è equivalente
        int largest=v[0];
        for(int i=1;i<N;i++)
            if (largest<v[i]) largest=v[i];
        cout<< "Il massimo e': "<<largest<<"\n";
    }
    main() {
        int v[N];
        cout << "Introduci "<<N<<" numeri: ";
        for (int i=0;i<N;i++) cin>>v[i];
        printLargest(v);
        return 0;
    }
}
```

```
Introduci 4 numeri: 3 9 5 1
Il massimo e': 9
```

# Vettori e funzioni 2

```
main() {
    int v[N];
    cout<<"dammi " <<N<<
        " numeri:"<<endl;
    for (int i=0;i<N;i++) {
        cin >> v[i];
    } printVector(N,v);
    invertMax(N,v);
    printVector(N,&v[0]);
}
```

```
dammi 4 numeri : 3 5 8 1
3      5      8      1
3      5     -8      1
```

```
#include <iostream.h>
const int N=4;
void invertMax(int n,v[]) {
    int max,indexOfMax;
    indexOfMax=0;
    max=v[0];
    for (int i=1;i<n;i++)
        if (max<v[i]) {
            max=v[i];
            indexOfMax=i;
        }
    v[indexOfMax]=
        -v[indexOfMax];
}
void printVector(int n,*v) {
    for (int i=0;i<n;i++)
        cout << v[i] <<" ";
    cout << endl;
}
```

# Costrutti idiomatici 1

## Inizializzazione di un vettore

```
int i, n=100, a[100], *p; ...  
for (p=a; p<a+n; p++) *p=0;
```

o in alternativa

```
int i, n=100, a[100], *p; ...  
for (p=&a[0]; p<&a[n]; p++) *p=0;
```

equivale a scrivere:

```
for (i=0; i<n; i++) a[i]=0;
```

# Operatori *new* e *delete*

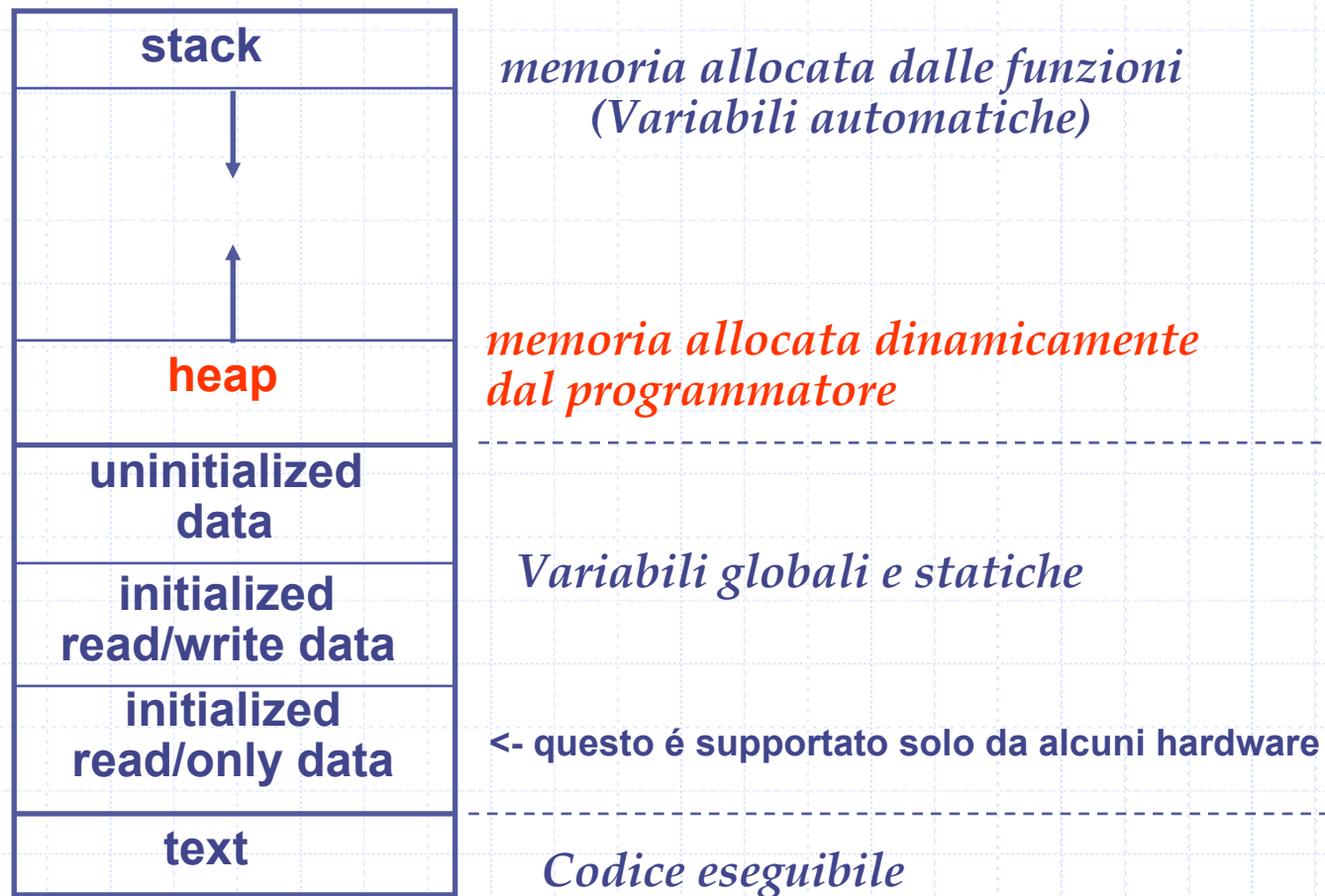


**new type** alloca **sizeof(type)** bytes in memoria (heap) e restituisce un puntatore alla base della memoria allocata. (esiste una funzione simile usata in C e chiamata **malloc**)

**delete (\* p)** dealloca la memoria puntata dal puntatore p. (Funziona solo con memoria dinamica allocata tramite new. Esiste un'analogia funzione in C chiamata **free**).

Il mancato uso della **delete** provoca un insidioso tipo di errore: il **memory leak**.

# Il modello di memoria



# Allocazione della memoria

Allocazione statica  
di memoria  
(at compile time)

```
main() {  
    int a;  
    cout<<a<<endl; //NO!  
    a=3;  
    cout<<a<<endl;  
}
```

OUTPUT: 1  
3

Allocazione  
dinamica  
di memoria  
(at run time)

```
main() {  
    int *pa;  
    pa=new int;  
    cout<<*pa<<endl; //NO!  
    *pa=3;  
    cout<<*pa<<endl;  
    delete(pa);  
    cout<<*pa<<endl; //NO!  
}
```

OUTPUT: 4322472  
3  
8126664

# Vettori rivistati

Dichiarare un vettore è in un certo senso come dichiarare un puntatore.

`v[0]` è equivalente a `*v`

Attenzione però alla differenza!

```
int v[100];    è "equivalente" a:  
               int *v; v=new int[100];
```

ATTENZIONE!

la prima versione alloca spazio STATICAMENTE (Stack)

la seconda versione alloca spazio DINAMICAMENTE (Heap)

# Elementi di C++ di base

## Stringhe

# Stringhe

In C e C++ non esiste il tipo di dato primitivo "stringa".  
**Tuttavia le funzioni di libreria di I/O trattano in modo speciale le regioni di memoria contenenti dei "char" (arrays di caratteri)**

Sono considerate "stringhe" i vettori di caratteri terminati da un elemento contenente il carattere '\0', indicato anche come NULL.

Un array di lunghezza N può contenere una stringa di lunghezza massima N-1! **(l'N-esimo carattere serve per il NULL)**

## Stringhe: vettori di caratteri

```
#include <iostream.h>
#define DIM 8
main() {
    char parola[DIM];
    cout<<"dammi una stringa :";
    cin>>parola;
    cout<<"La stringa e' "<<parola<<endl;
    for (int i=0;i<DIM;i++)
        cout<<parola[i]<<" "<<(int)parola[i]<<endl;
    return 0;
}
```

```
dammi una stringa :pippo
La stringa e' pippo
p 112
i 105
p 112
p 112
o 111
  0
B 66
☺ 1
```

```
dammi una stringa : pi po
La stringa e' pi
p 112
i 105
  0
  0
X 88
B 66
☺ 1
  0
```

```
#include <iostream.h>
```

```
main() {
```

```
    const int DIM=8;
```

```
    char parola[DIM];
```

```
    cout<<"dammi una stringa :";
```

```
    cin.getline(parola,DIM);
```

```
    cout<<"La stringa e' "<<parola<<endl;
```

```
    for (int i=0;i<DIM;i++)
```

```
        cout<<parola[i]<<" "<<(int)parola[i]<<endl;
```

```
    return 0;
```

```
}
```

## Stringhe: vettori di caratteri

dammi una stringa :pippo

La stringa e' pippo

p 112

i 105

p 112

p 112

o 111

0

B 66

☺ 1

dammi una stringa : pi po

La stringa e' pi po

32

p 112

i 105

32

p 112

o 111

0

B 66

## Stringhe: vettori di caratteri

```
#include <iostream.h>
main() {
    const int DIM=8;
    char parola[DIM];
    cout<<"dammi una stringa :";
    cin<<ws;
    cin.getline(parola,DIM);
    cout<<"La stringa e' "<<parola<<endl;
    for (int i=0;i<DIM;i++)
        cout<<parola[i]<<" "<<(int)parola[i]<<endl;
    return 0;
}
```

```
dammi una stringa :pippo
La stringa e' pippo
p 112
i 105
p 112
p 112
o 111
0
B 66
☺ 1
```

```
dammi una stringa : pi po
La stringa e' pi po
p 112
i 105
32
p 112
o 111
0
B 66
☺ 1
```

# Operatori su stringhe

**Nella libreria string.h sono predefinite una serie di funzioni operanti su stringhe.**

La libreria va inclusa con il comando `#include <string.h>`

Le funzioni di uso più frequente sono:

```
char *strcpy(a,b); /*copia b su a*/  
int strcmp(a,b);   // restituisce<0 se a<b,0 se a=b,>0 se a>b  
char *strcat(a,b); /* appende b in coda ad a*/  
size_t strlen(a);  //restituisce la lunghezza della stringa a
```

(Abbiamo assunto la definizione: `char *a, *b;`)

## Esercizio:

Implementare queste funzioni (ricordando la definizione di stringa)

# Stringhe e funzioni

```
#include <iostream.h>
#include <string.h>
#define DIM 10
main() {
    char parola[DIM];
    //char * altraparola; NO!
    char altraParola[DIM];
    cout<<"dammi una stringa :";
    cin>>ws;
    cin.getline(parola,DIM);
    cout<<"La stringa inserita e' \""
        <<parola<<"\"\\n";
    strcpy(altraParola,parola);
    cout<<"Il contenuto di altraParola e' \""
        <<altraParola<<"\"\\n";
    return 0;
}
```

# Stringhe

## Attenzione alle sottigliezze!

```
char *mystring="Pippo";           è "equivalente" a:  
    char *mystring;  
    mystring=new char[6];  
    mystring[0]='P';mystring[1]='i';...;  
    ;...; mystring[4]='o';mystring[5]='\0';
```

(in realtà c'è una differenza riguardo a dove  
in memoria viene riservato il posto: stack o heap)

```
char *frase;  
    definisce il puntatore ma NON alloca spazio in  
    memoria!
```

# Costrutti idiomatici 1

## Assegnazioni

Le due righe seguenti hanno SIGNIFICATI  
DIVERSI  
(ATTENZIONE! sorgente di MOLTI errori)

```
if (a==b) cout << "OK\n";
```

```
if (a=b) cout << "OK\n";
```

La seconda è equivalente a:

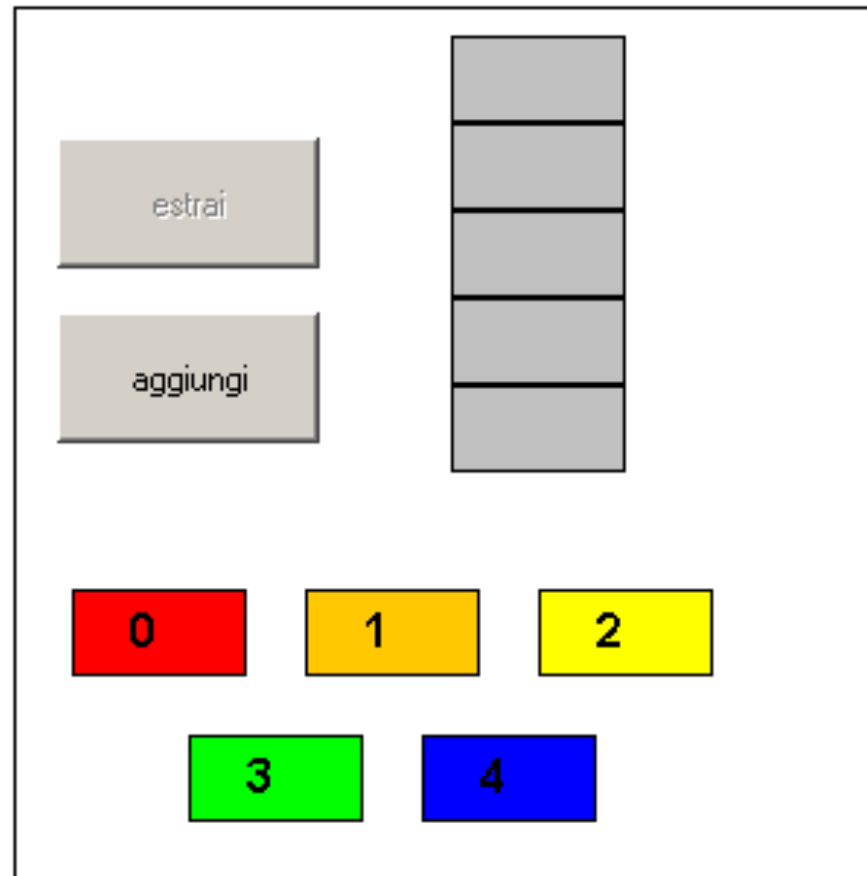
```
a=b;
```

```
if (a!=0) cout << "OK\n";
```

A decorative blue line is positioned on the left side of the slide. It starts with a small circle at the top, then extends horizontally to the right, and finally extends vertically downwards.

## Il nostro esempio guida: La costruzione di uno stack

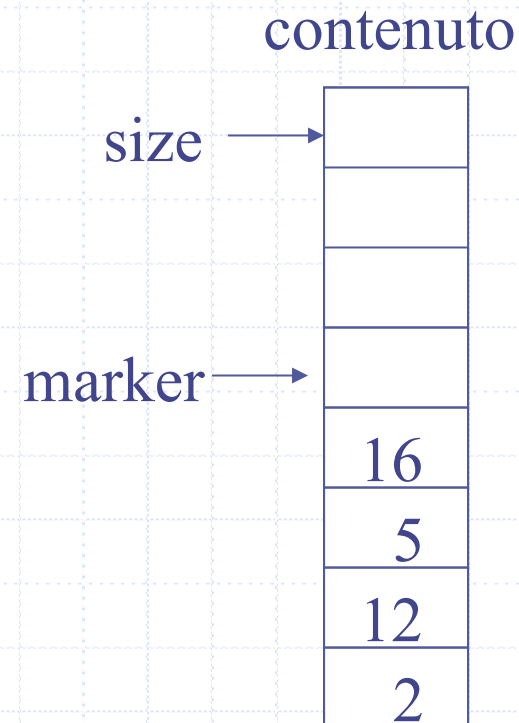
# Costruiamo uno stack



[stackapplet.html](http://stackapplet.html)

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

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



```
Pila * crea(int initialSize) {  
    //crea una Pila  
    #ifdef DEBUG  
        cout<<"entro in crea"<<endl;  
    #endif  
    Pila * s= new Pila ;  
    s->size=initialSize;  
    s->defaultGrowthSize=initialSize;  
    s->marker=0;  
    s-> contenuto=new int[initialSize];  
    return s;  
}
```

```
void distruggi(Pila * s) {  
    //distruggi lo Pila  
    #ifdef DEBUG  
        cout<<"entro in destroy"<<endl;  
    #endif  
    delete [] (s->contenuto);  
    delete s;  
}
```

# Il preprocessore

# Il preprocessore

Agisce come un Editor prima che la compilazione inizi.

## Inclusione di files

`#include <filename>` (cerca in /usr/lib)

`#include "filename"` (specifica la pathname)

## Definizione di costanti

```
#define N 100
```

```
    for (k=1; k<N ; k++) cout<< k << endl;
```

SCONSIGLIATO! E' meglio:

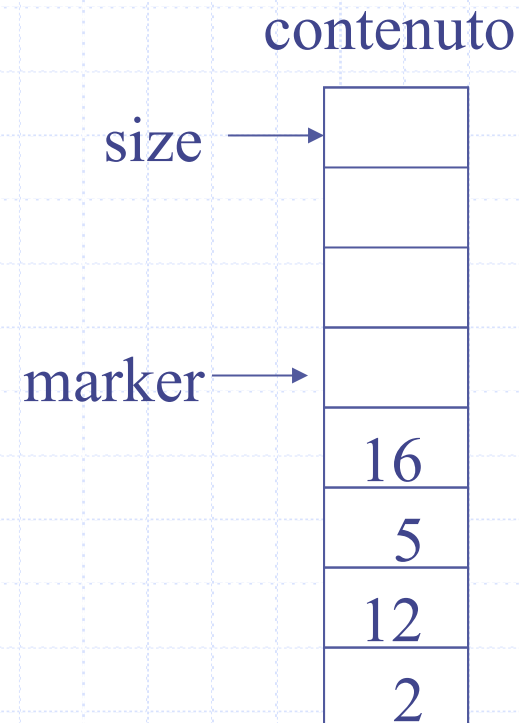
```
const int N=100
```

A decorative blue line is positioned on the left side of the slide. It starts with a small circle at the top, then extends horizontally to the right, and finally extends vertically downwards.

## Il nostro esempio guida: La costruzione di uno stack

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

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



```
Pila * crea(int initialSize) {  
    //crea una Pila  
    cout<<"entro in crea"<<endl;  
    Pila * s= new Pila ;  
    s->size=initialSize;  
    s->defaultGrowthSize=initialSize;  
    s->marker=0;  
    s-> contenuto=new int[initialSize];  
    return s;  
}
```

```
void distruggi(Pila * s) {  
    //distruggi lo Pila  
    cout<<"entro in destroy"<<endl;  
    delete [] (s->contenuto);  
    delete s;  
}
```

```
void cresci(Pila *s, int increment){  
    //aumenta la dimensione dello Pila  
    cout<<"entro in cresci"<<endl;  
    s->size+=increment;  
    int * temp=new int[s->size];  
    for (int k=0; k<s->marker;k++) {  
        temp[k]=s->contenuto[k];  
    }  
    delete [] (s->contenuto);  
    s->contenuto=temp;  
}
```

```
void inserisci(Pila *s, int k) {  
    //inserisci un nuovo valore  
    cout<<"entro in inserisci"<<endl;  
    if (s->size==s->marker)  
        cresci(s,s->defaultGrowthSize);  
    s->contenuto[s->marker]=k;  
    s->marker++;  
}
```

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

```
void stampaStato(Pila *s) {  
    //stampa lo stato dello Pila  
    cout <<"===== "<< endl;  
    cout << "size = "<<s->size<<endl;  
    cout << "defaultGrowthSize = "  
        <<s->defaultGrowthSize<<endl;  
    cout << "marker = "<<s->marker <<endl;  
    for (int k=0;k<s->marker;k++)  
        cout << "["<<(s-&br/>>contenuto[k])<<"]";  
    cout << endl;  
    cout <<"===== "<< endl;  
}
```

```
Pila * copia(Pila * from) {  
    cout<<"entro in copia"<<endl;  
    Pila * to=crea(from->size);  
    to->defaultGrowthSize=from->defaultGrowthSize;  
    for (int k=0; k<from->marker;k++) {  
        to->contenuto[k]=from->contenuto[k];  
    }  
    to->marker=from->marker;  
    return to;  
}
```

```
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"; stampaStato(s);  
    distruggi(s);  
    cout<<"s"; stampaStato(s);  
    for (int k=1; k<15;k++)  
        cout<<estrai(w)<<endl;  
    cout<<"w"; stampaStato(w);  
}
```

```
bash-2.02$ gcc Pila.cc -o Pila.exe
```

```
bash-2.02$ Pila.exe
```

```
entro in crea
```

```
s=====
```

```
size = 5
```

```
defaultGrowthSize = 5
```

```
marker = 0
```

```
=====
```

```
entro in inserisci
```

```
entro in inserisci
```

```
entro in inserisci
```

```
entro in inserisci
```

```
entro in inserisci
```

```
entro in inserisci
```

```
entro in cresci
```

```
entro in inserisci
```

```
entro in inserisci
```

```
entro in inserisci
```

```
s=====
```

```
size = 10
```

```
defaultGrowthSize = 5
```

```
marker = 9
```

```
[1][2][3][4][5][6][7][8][9]
```

```
=====
```

```
entro in copia
```

```
w=====
```

```
size = 10
```

```
defaultGrowthSize = 10
```

```
marker = 9
```

```
[1][2][3][4][5][6][7][8][9]
```

```
=====
```



entro in estrai

9

entro in estrai

8

entro in estrai

4

entro in estrai

3

$$S =$$

```
size = 10
```

defaultGrowthSize = 5

marker = 2

[1][2]

entro in distruggi

S \_\_\_\_\_

size = 1627775824

```
defaultGrowthSize = 1627775824
```

marker = 2

[1627775848][1627775848]

Handwriting practice lines featuring dashed letters and numbers for tracing on a four-line grid. The first row contains the letters 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z' and the numbers '1', '2', '3', '4', '5', '6', '7', '8', '9', '0'. The second row contains the letters 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' and the numbers '1', '2', '3', '4', '5', '6', '7', '8', '9', '0'. The third row contains the letters 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z' and the numbers '1', '2', '3', '4', '5', '6', '7', '8', '9', '0'. The fourth row contains the letters 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' and the numbers '1', '2', '3', '4', '5', '6', '7', '8', '9', '0'.

entro in estrai

9

entro in estrai

8

entro in estrai

2

entro in estrai

# 1

entro in estrai

assertion "s->marker>0" failed: file "Pila.cc", line 72

```
bash-2.02$
```

```
#include <Pila.h>
```

```
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=s;
```

```
    cout<<"w"; stampaStato(w);
```

```
    for (int k=1; k<8;k++)
```

```
        cout<< estrai(s)<<endl;
```

```
    cout<<"s"; stampaStato(s);
```

```
    cout<<"w"; stampaStato(w);
```

```
}
```

Perchè abbiamo scritto  
il metodo copia?

Una copia  
(troppo)  
sbrigativa...

```

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

```

```

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

```

```

entro in estrai
9
entro in estrai
8
...

```

```

...
entro in estrai
4
entro in estrai
3

```

```

s=====
size = 10
defaultGrowthSize = 5
marker = 2
[1][2]
=====

```

```

w=====
size = 10
defaultGrowthSize = 5
marker = 2
[1][2]
=====

```

## Pila.h

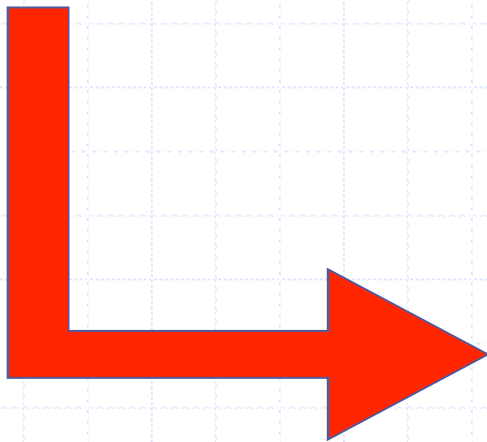
```
struct Pila {  
    int size;  
    int defaultGrowthSize;  
    int marker;  
    int * contenuto;  
} ;
```

```
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) ;
```

```
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) ;
```

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

## Re-implementazione di estrai



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

# 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;  
    ...  
}
```

Re-implementazione  
di estrai: dove scrivo il codice?

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

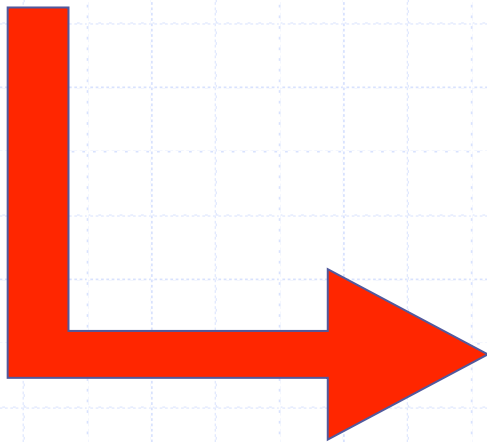
## Re-implementazione di estrai: dove scrivo il codice?



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

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

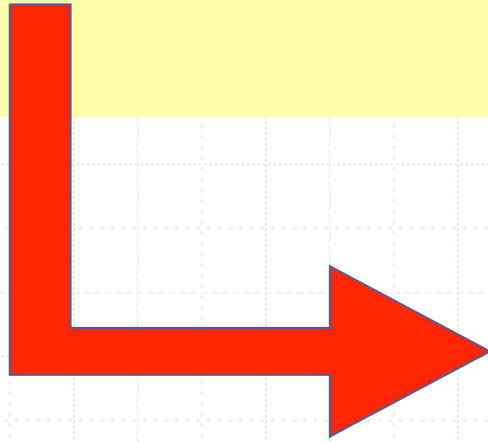
Re-implementazione  
di estrai con this  
implicito



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

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

## Re-implementazione di crea



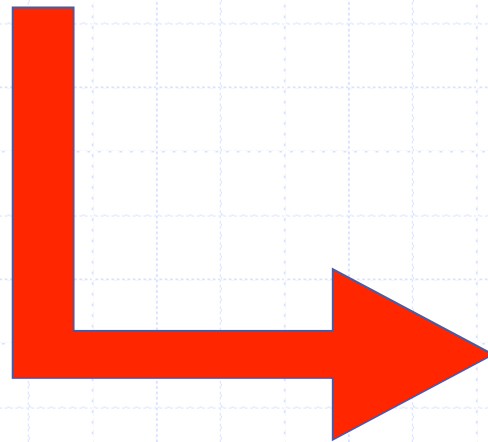
```
Pila::Pila(int initialSize) {  
    size=initialSize;  
    defaultGrowthSize=initialSize;  
    marker=0;  
    contenuto=new int[initialSize];  
}
```

**"Il costruttore"**

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

## Re-implementazione di distruggi

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



**"Il distruttore"**

# 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->distrogi();  
    cout<<"s"; s->stampaStato();  
    for (int k=1; k<15;k++)  
        cout<< w->estrai()<<endl;  
    cout<<"w"; w->stampaStato();  
}
```

## 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

## Pila.h

## versione 4

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

# Pila.h

## versione 5

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

# 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() ;  
        void copy(Pila * to) ;  
        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() ;  
        void copy(Pila * to) ;  
        void inserisci(int k) ;  
        int estrai() ;  
        void stampaStato() ;  
};
```