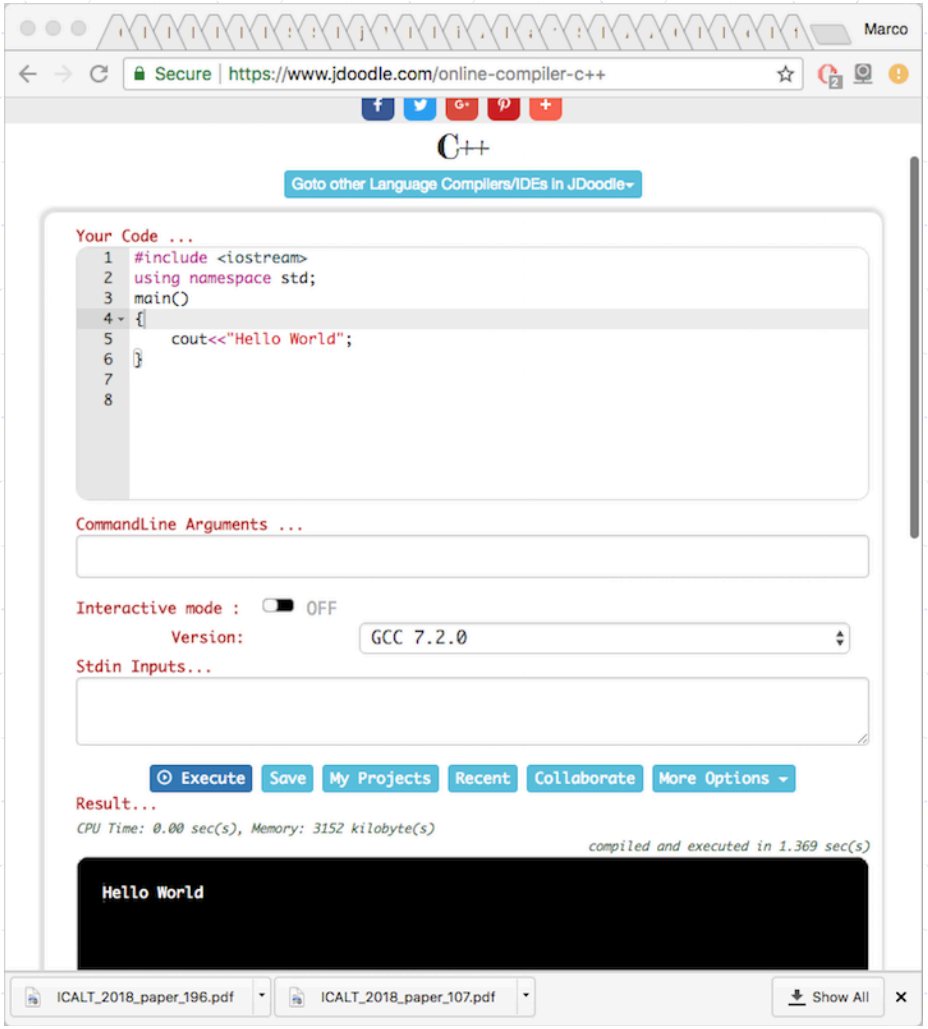
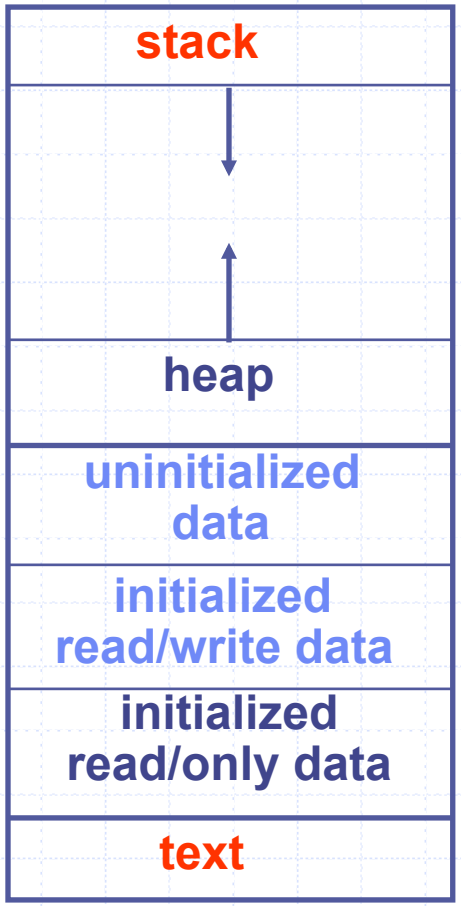
Richiami di C++ di base

Tool per fare esercizi in C++

- https://www.jdoodle.com/online-compiler-c++



Il modello di memoria



memoria allocata dalle funzioni (Variabili automatiche)

memoria allocata dinamicamente dal programmatore

Variabili globali e statiche

<- questo é supportato solo da alcuni hardware

Codice eseguibile

Modularizzazione: Funzioni

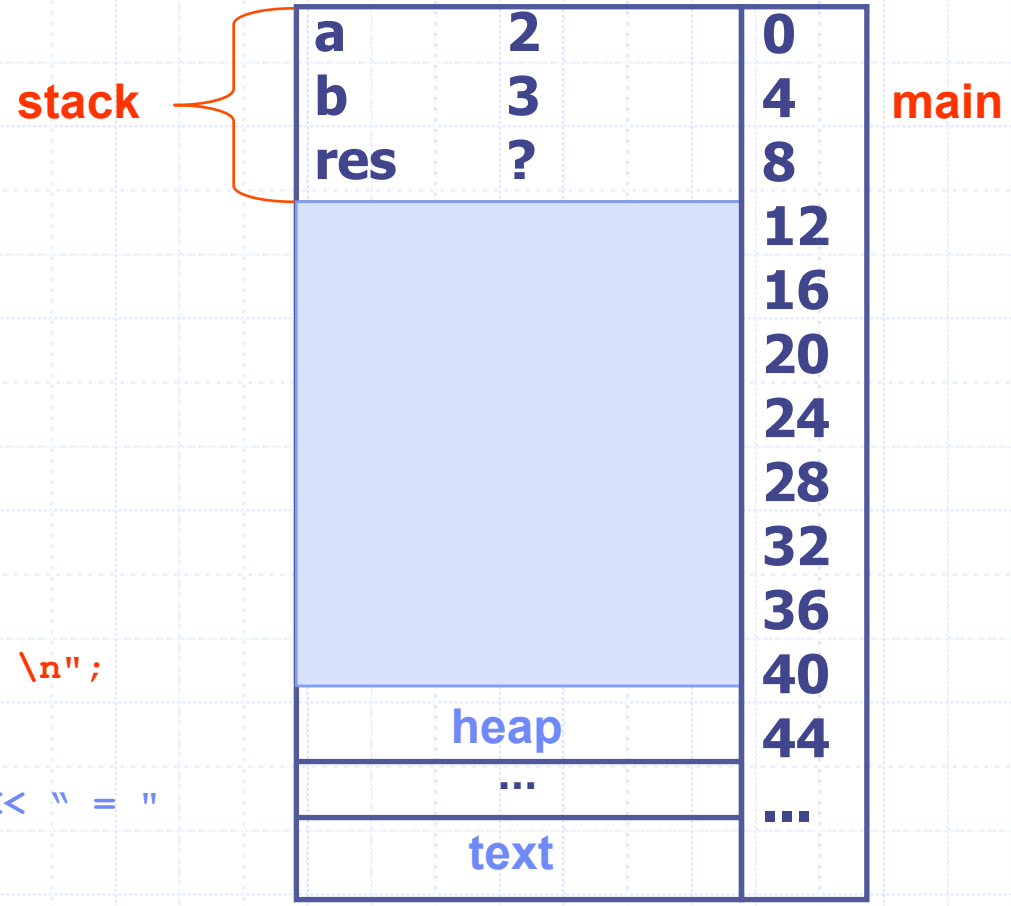
Funzioni come "procedure parametrizzate"

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

Modularizzazione: Funzioni

Esempio

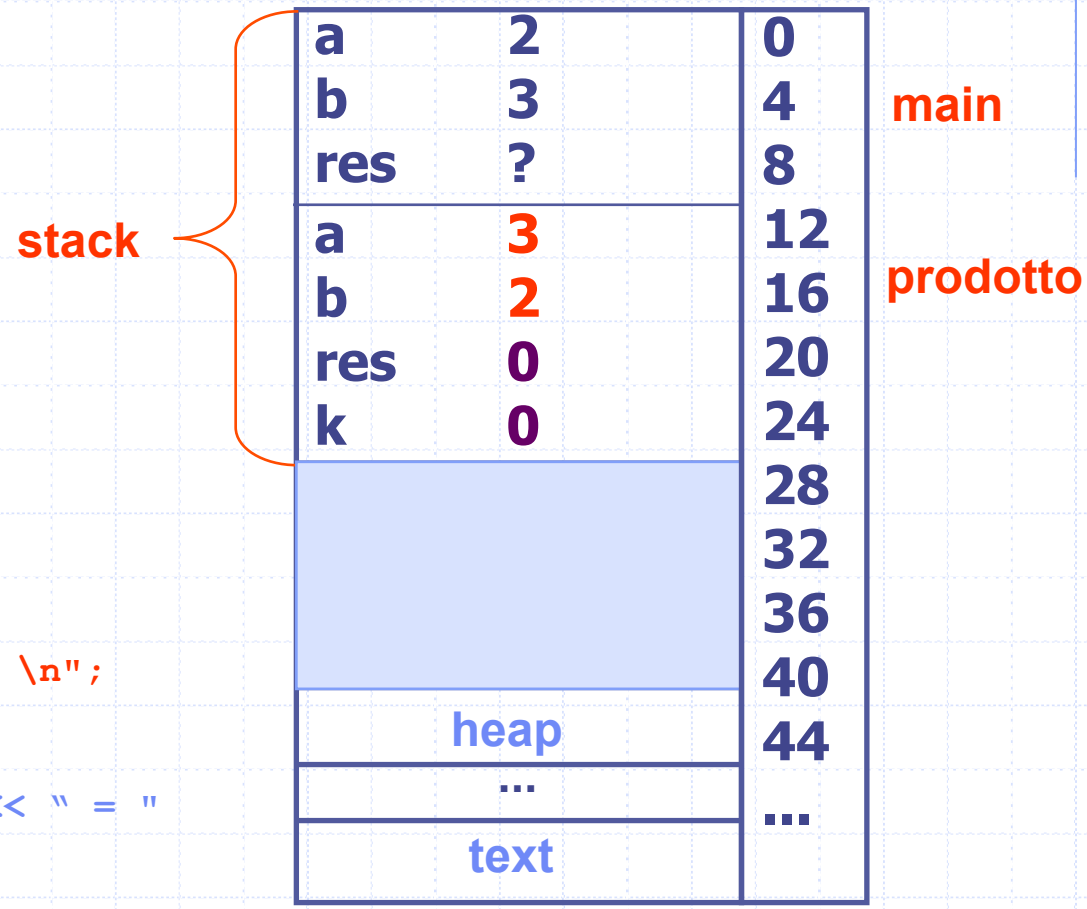
```
int somma(int a, imt b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
<< res << "\n";
}
```



Modularizzazione: Funzioni

Esempio

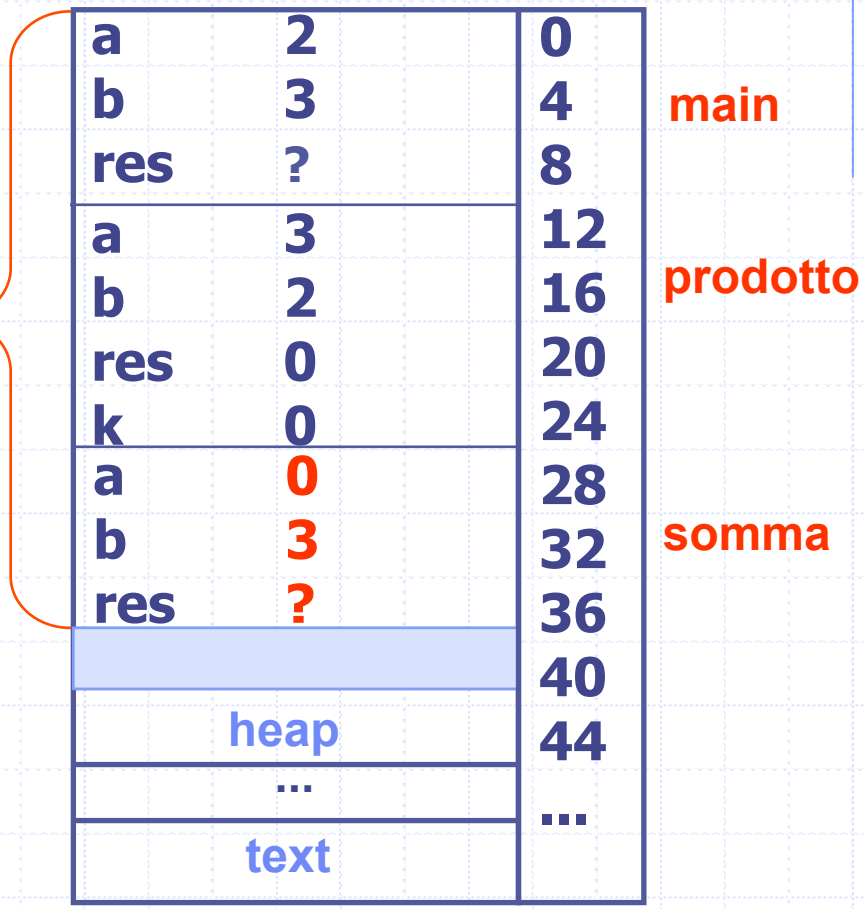
```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
<< res << "\n";
}
```



Modularizzazione: Funzioni

Esempio

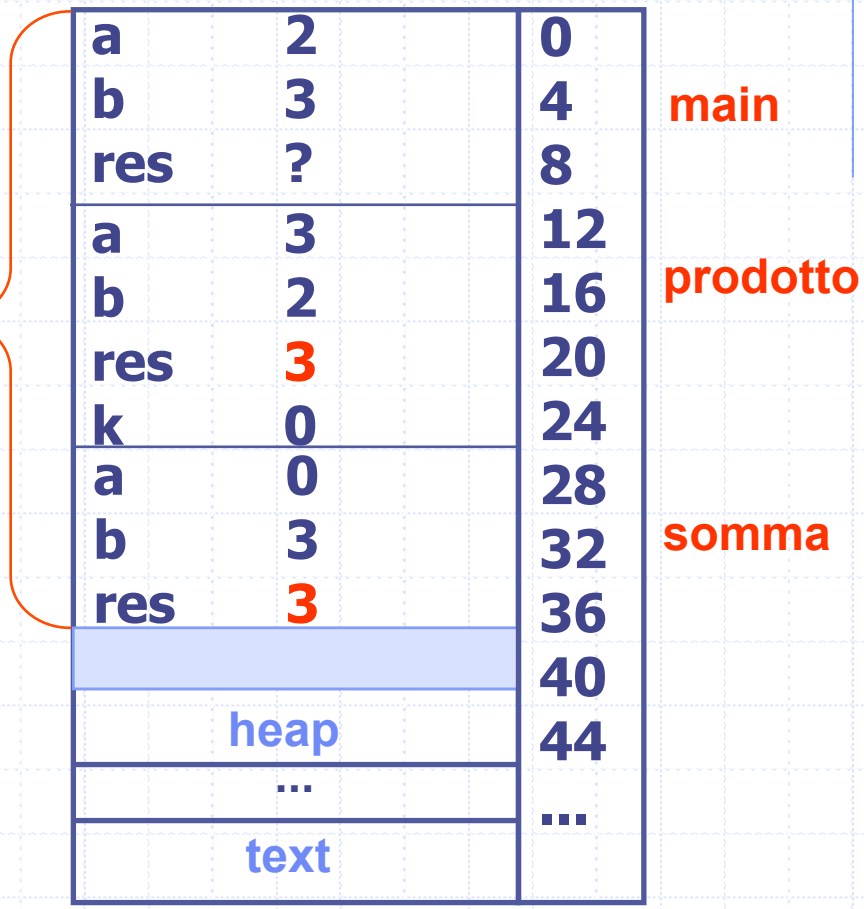
```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
<< res << "\n";
}
```



Modularizzazione: Funzioni

Esempio

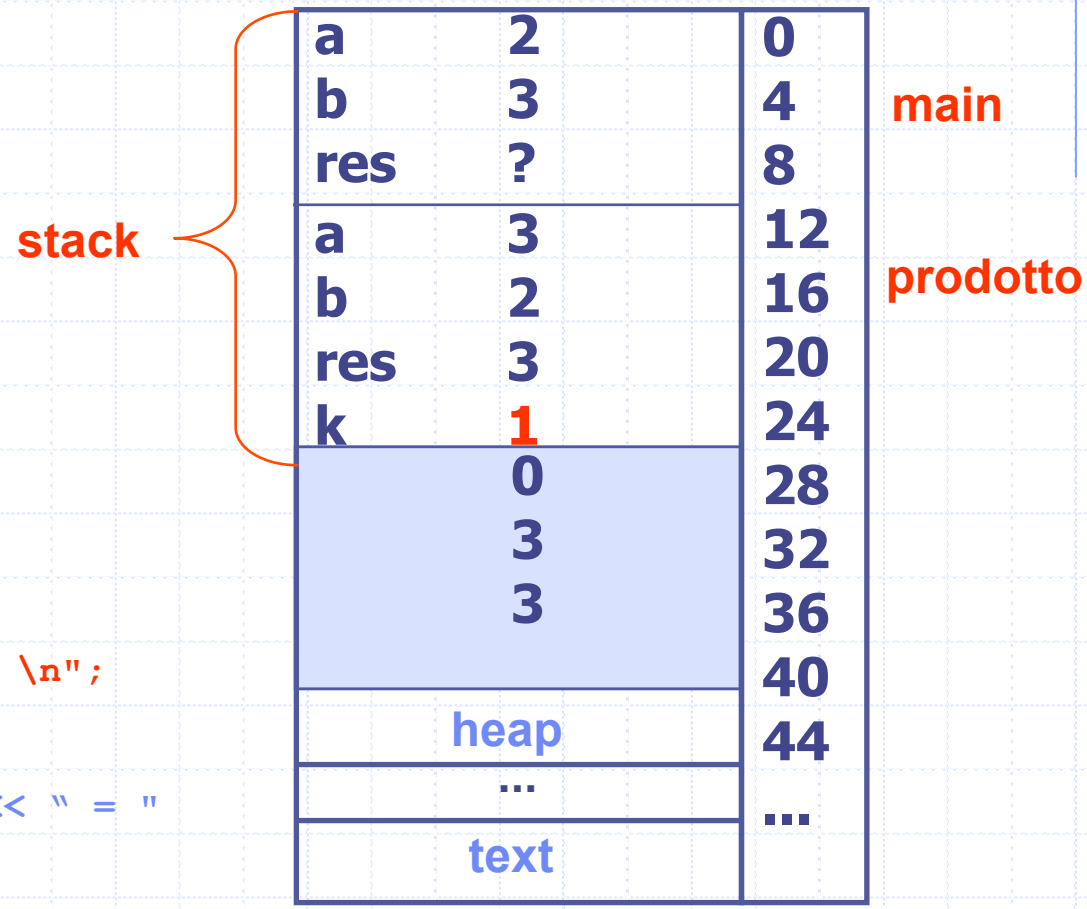
```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
<< res << "\n";
}
```



Modularizzazione: Funzioni

Esempio

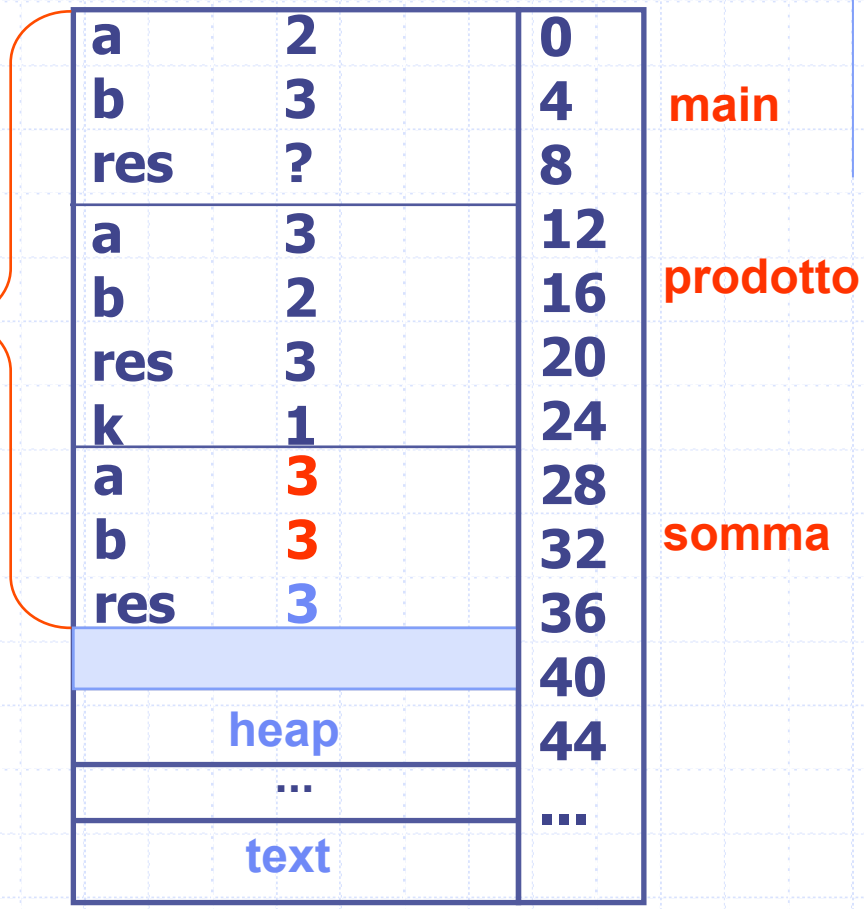
```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
<< res << "\n";
}
```



Modularizzazione: Funzioni

Esempio

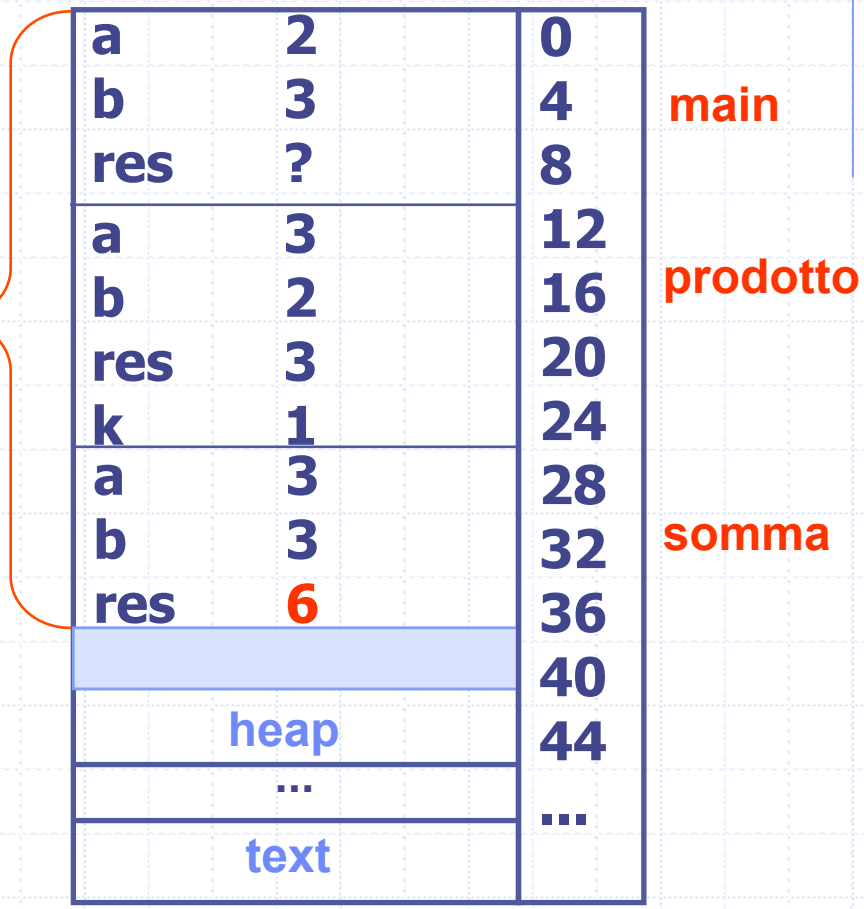
```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
<< res << "\n";
}
```



Modularizzazione: Funzioni

Esempio

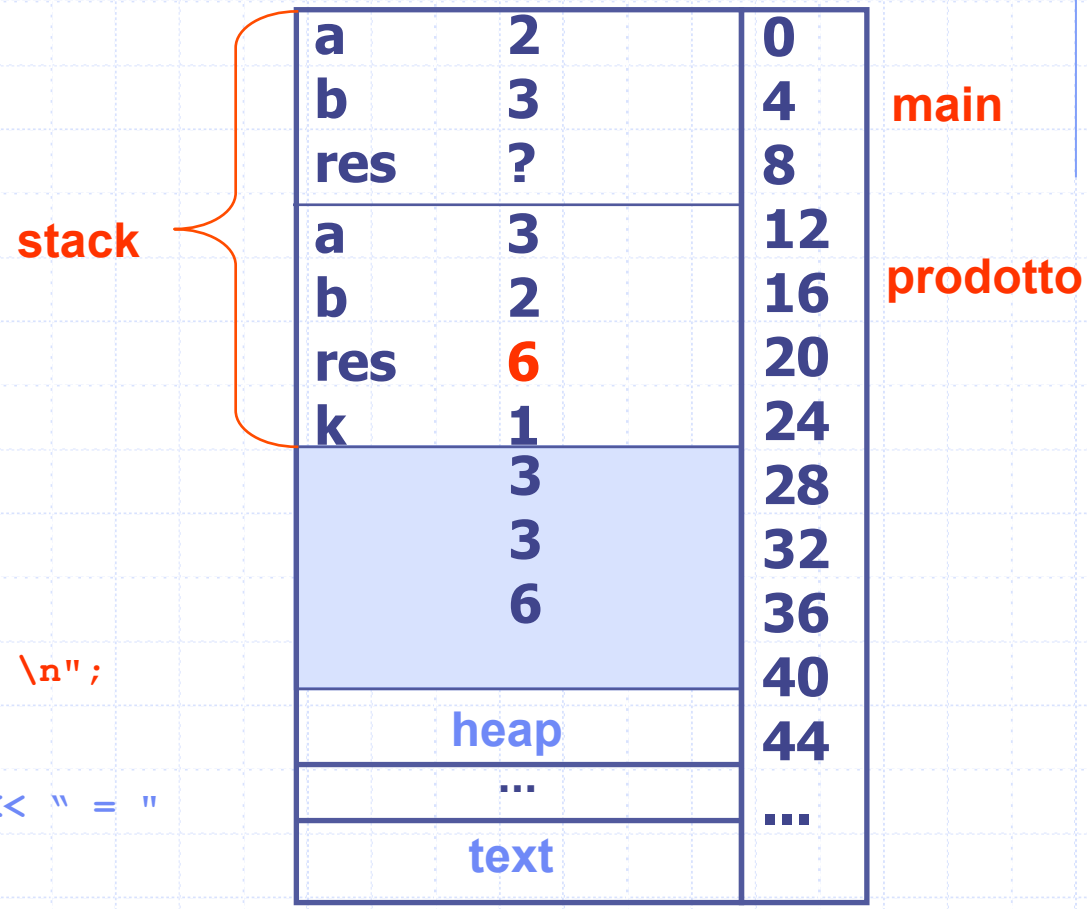
```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
<< res << "\n";
}
```



Modularizzazione: Funzioni

Esempio

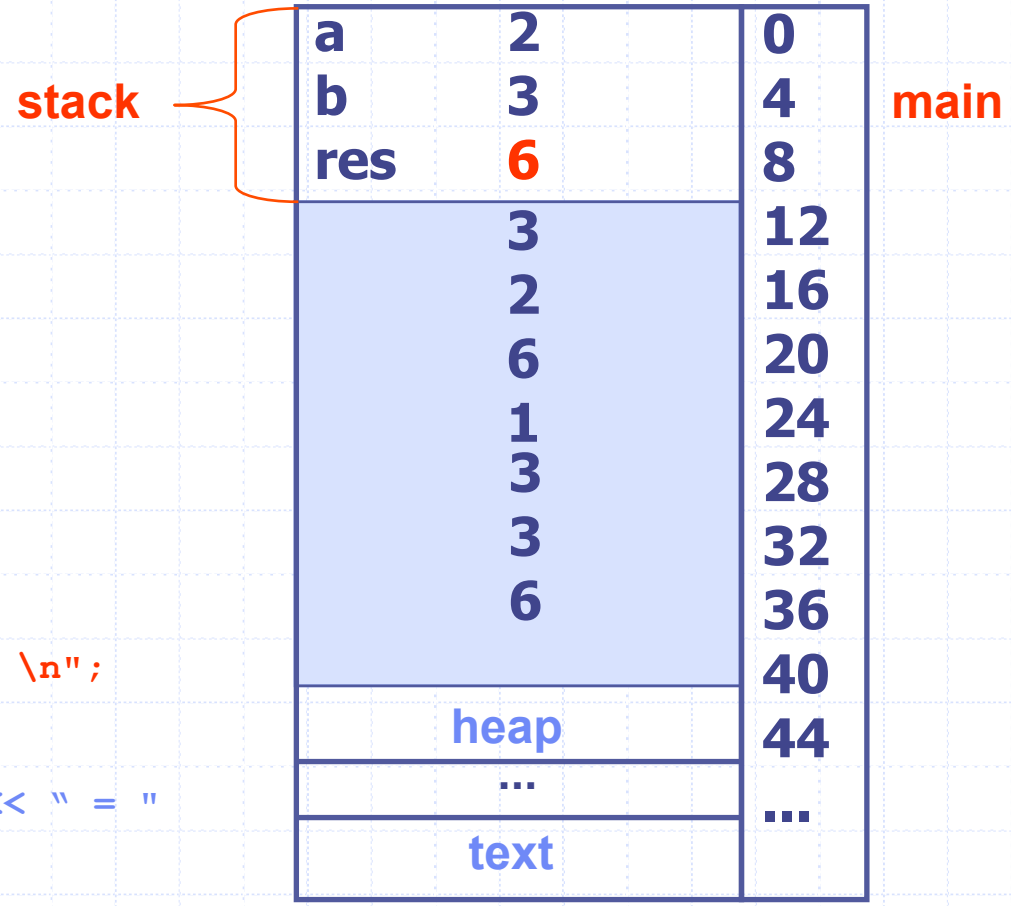
```
int somma(int a, int b) {
    int res;
    res=a+b;
    return res;
}
int prodotto(int b, int a) {
    int res=0;
    for (int k=0; k<b; k++)
        res=somma(res,a);
    return res;
}
main() {
    int a,b,res;
    cout << "dammi due numeri \n";
    cin >> a >> b;
    res=prodotto(a,b);
    cout << a << " * " << b << " = "
<< res << "\n";
}
```



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";
}
```



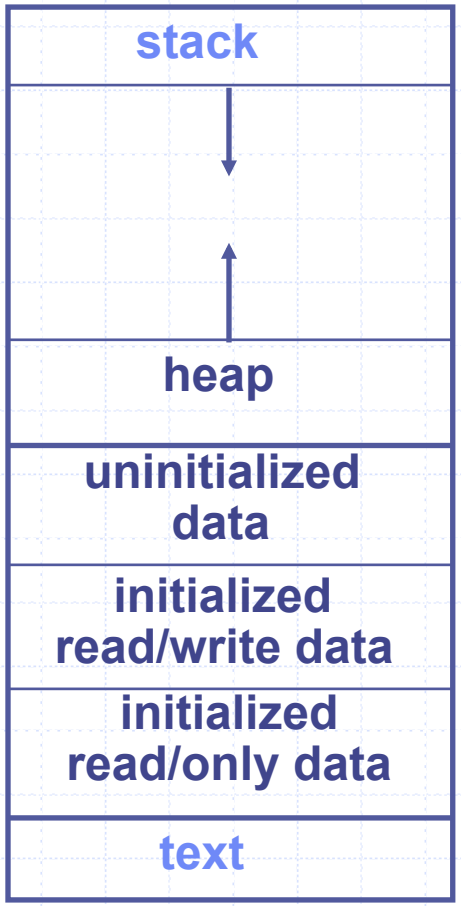
Funzioni ricorsive

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



memoria allocata dalle funzioni (Variabili automatiche)

memoria allocata dinamicamente dal programmatore

Variabili globali e statiche

<- questo é supportato solo da alcuni hardware

Codice eseguibile

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 svogere 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

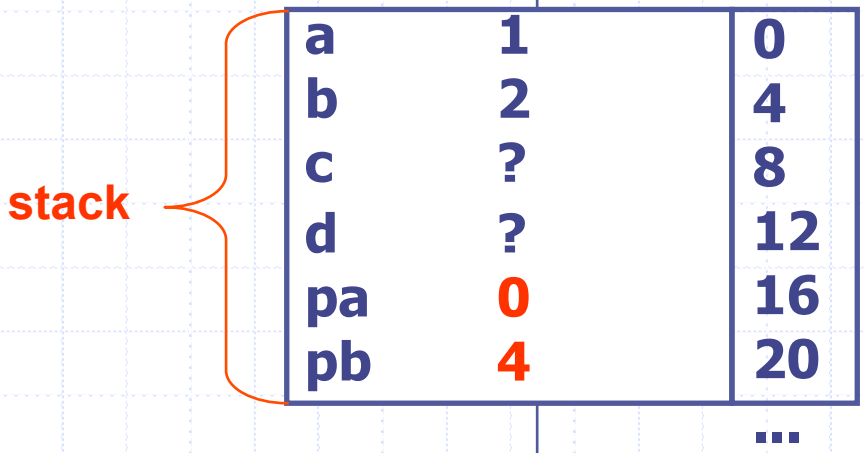
Operatore indirizzo: &

&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

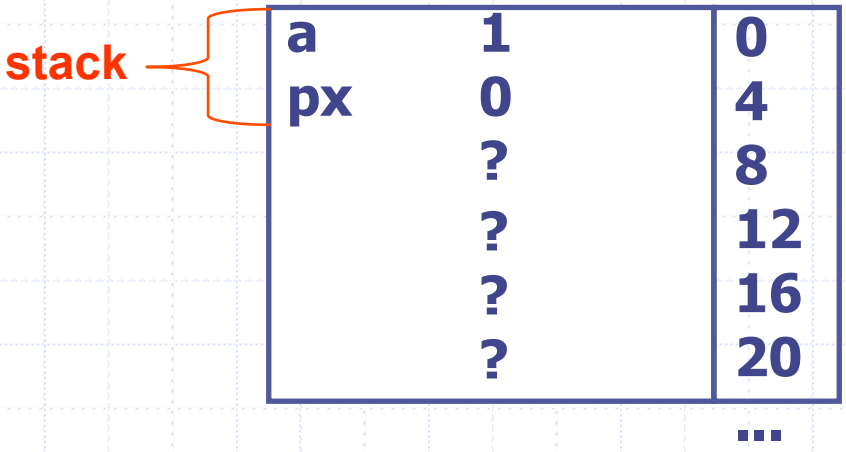
```
main() {
      int a,b,c,d;
      int * pa, * pb;
      pa=&a; pb=&b;
      a=1; b=2;
      c=a+b;
      d=*pa + *pb;
      cout << a<<"  "<<b<<"  "<< c <<endl;
      cout << a <<"  "<< *pb <<"  "<< d <<endl;
}
```



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;
}
```



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: ->

```
enum colors {BLACK, BLUE, RED, GREEN};
main()
{
   struct point {
        int x;
        int y;
        enum colors color;
    };
    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*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

```
#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;
}
```

```
Base:
0012FF74 0012FF74

0012FF74 0
0012FF78 1
0012FF7C 2
0012FF80 3
0012FF74 0
0012FF78 1
0012FF7C 2
0012FF80 3
```

Vettori

Vettore Uni-dimensionale di double

```
#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;
}
```

```
Base:
0012FF64 0012FF64

0012FF64 0
0012FF6C 1
0012FF74 2
0012FF7C 3
0012FF64 0
0012FF6C 1
0012FF74 2
0012FF7C 3
```

Vettori e funzioni

```
#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;
```

Esercizio di esame

```
void g(char x[], int y) {
   y--;
   x[y]--;  }
void f(char *x, int * y) {
   (*y)++;
   x[*y]++;  }
int main(){
  char x[2];
  int y;
  x[0]='R';  x[1]='R';   y=0;
  f(x,&y);
  g(x,y);
  cout<<x[0]<<" "<<x[1]<<" "<<y;
  return 0; }
```

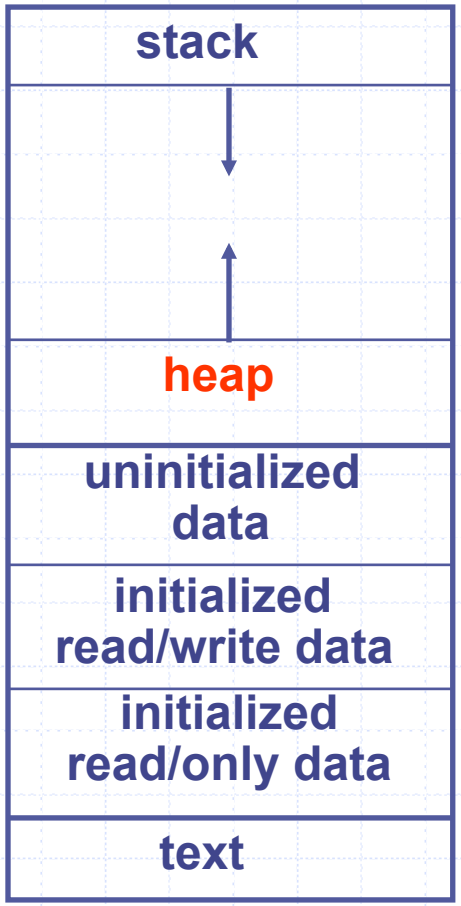
Esercizio di esame

```
int k=2;
  void f( int m ) { m=m*2; }
  void g( int *m ) { m++; }
  void h( int m[4] ) { m[0]--; }
  void p(){cout<<k;}
  int main(){
    int k=5;
    g(&k);h(&k);f(k);
    cout << k;
    p();
    return 0;
 }
```

Esercizio di esame

```
void f(char *x, int * y) {
  (*y)++;
  x[*y]++;  }
void g(char x[], int y) {
  y--;
  x[y]- -;  }
int main(){
 char x[2];
 int y;
 x[0]='N';  x[1]='C';   y=0;
 f(x,&y);
 g(x,y);
 cout<<x[0]<<" "<<x[1]<<" "<<y;
 return 0; }
```

Il modello di memoria



memoria allocata dalle funzioni (Variabili automatiche)

memoria allocata dinamicamente dal programmatore

Variabili globali e statiche

<- questo é supportato solo da alcuni hardware

Codice eseguibile

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'analoga funzione in C chiamata `free`).

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

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
```

Stringhe: vettori di caratteri

```
#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;
}
```

```
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
```

Esercizio di esame

```
void f(char x[2],int index,char value){
    x[index]=value;
}
int main(int argc, char** argv) {
    char a[]="ABCDEFGHIL";
    strcpy(&a[2],"000");
    f(&a[2],3,'$');
    f(&a[2],5,0);
    cout<<a;
    return 0;
}
```

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";
```