Allocazione dinamica della memoria in C++

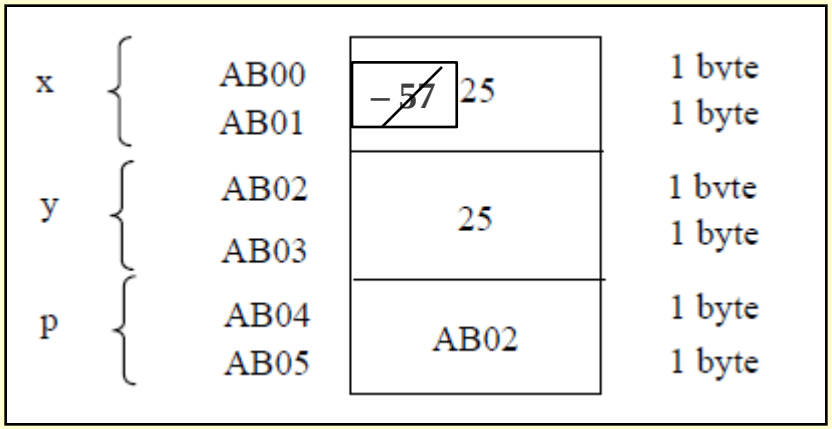
Allocazione statica della memoria

Cosa succede nella memoria del computer quando il un programma vengono **dichiarate** delle variabili?

Ad esempio:

```
int  main()
{  int x= – 57 , y=25;
   int *p;
   p=&y;
   x=*p;
 }
```

memoria



Le dichiarazioni determinano la presenza, in questo caso, di **tre locazioni di memoria x, y** e **p** allocate nell'area dati, come mostrato nella seguente figura.

Questo modalità di **allocazione** della memoria viene detta **statica.**

Le variabili dichiarate in questo modo **possono variare il loro contenuto**, ma **non possono variare le loro caratteristiche** a tempo di esecuzione.

Ad esempio, se dichiariamo: int V[5];

non possiamo modificare l'array a tempo di esecuzione in modo che questo contenga un numero di elementi che sia superiore a 5.

Allocazione *statica* della memoria

- La quantita di **memoria** da allocare è **determinata** e **fissata a tempo di compilazione** automaticamente dal sistema.
- Ogni variabile statica ha un ***nome***, attraverso il quale la si può riferire.
- Il **programmatore non ha la possibilità di influire sul tempo di vita** di queste variabili che verranno deallocate alla terminazione del programma o alla chiusura di un blocco di istruzioni (es.: {… } , for(int i=….)).

Allocazione *dinamica* della memoria

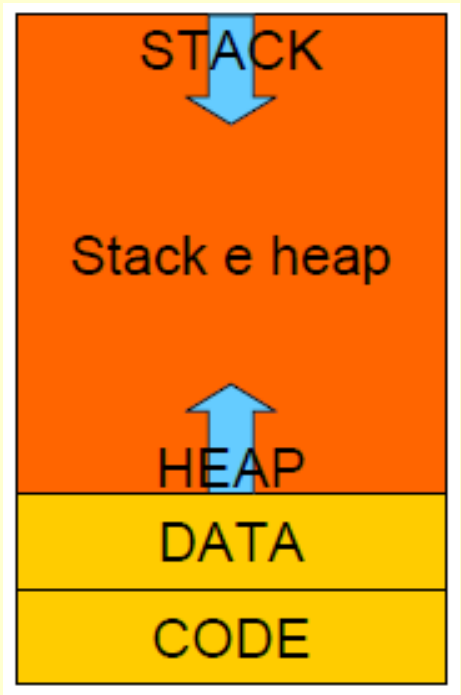
- Permette di **definire variabili che possono essere create o accresciute in fase di esecuzione**.
- Le variabili dinamiche devono essere **allocate** e **deallocate *esplicitamente dal programmatore.***
- L'area di **memoria** in cui vengono allocate le variabili dinamiche si chiama ***heap*** **(in C)** o ***memoria libera*** **(in C++).**
- Le variabili dinamiche non hanno un ***identificatore (nome),*** **ma possono essere riferite soltanto** attraverso il loro **indirizzo** (mediante i ***puntatori).***
- **Il tempo di vita** delle variabili dinamiche è l'intervallo di **tempo** che intercorre tra l'allocazione e la deallocazione (che sono impartite esplicitamente dal programmatore).

Allocazione della memoria

Il programma compilato è costituito da due parti distinte:

- Code Segment
- Data Segment

Quando il programma viene eseguito, il Sistema Operativo riserva uno spazio di memoria per allocarvi **stack** e **heap.**



Se lo spazio per stack e heap ha dimensione fissa (dipende dal S.O.), questo può esaurirsi per effetto di ripetute chiamate a funzione (stack) non chiuse o allocazioni dinamiche (heap o memoria libera).

Esiste infatti una proprietà che lega heap e stack:

allocando memoria in una delle due aree, questa aumenta mentre l'altra diminuisce, rientrando ovviamente nei limiti imposti dalla quantità di memoria disponibile.

Allocazione dinamica e rilascio della memoria dinamica in C

Riassumendo

I prototipi delle funzioni **malloc** , **calloc**, **realloc** e **free** sono i seguenti:

void * **malloc** (dim_totale); **void** * **calloc** (int num_elementi, int dim_elemento);	Restituiscono un **puntatore void** che indica che si tratta di un puntatore ad una regione di dati di tipo sconosciuto. Restituiscono **NULL** se non riescono ad allocare la quantità di memoria richiesta.
void * **realloc** (void *ptr, int dim_totale);	Essa modifica la dimensione di un blocco di memoria puntato da ptr a dim_totale bytes. Se ptr è NULL, l'invocazione è equivalente ad una malloc(dim_totale). Se dim_totale è 0, l'invocazione è equivalente ad una free(ptr).
void **free** (void *ptr);	Essa libera lo spazio di memoria puntato da ptr, il cui valore proviene da una precedente malloc() o calloc() o realloc() e lo restituisce al S.O. Se ptr è NULL non viene eseguita nessuna operazione.

Allocazione dinamica e rilascio della memoria in C++

Le funzioni che il programmatore può usare per accedere alla ***memoria libera*** in C++ sono:

- **new** **per allocare memoria in modo dinamico.**

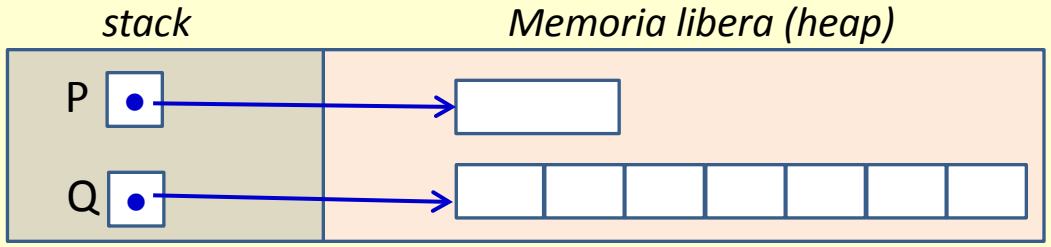
  L'operatore **new** **riconosce il tipo e anche la dimensione in byte** dell'area di memoria da allocare.

  Restituisce **un puntatore**.

  Il compilatore controlla che il tipo del puntatore all'oggetto corrisponda al tipo del puntatore a cui viene assegnato il valore e genera un errore se sono di tipo diverso.

Allocazione dinamica e rilascio della memoria in C++

Esempi:

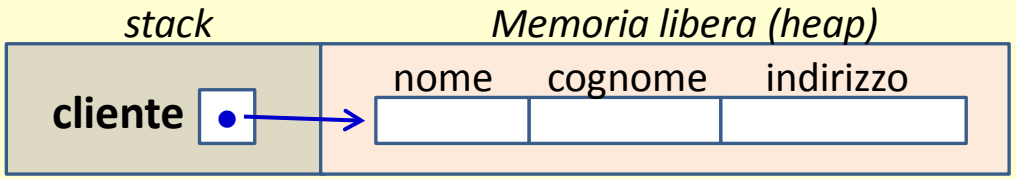


int *P;
float *Q;
P = new int; // P punta ad una variabile di tipo intero
Q= new float[7]; // Q punta ad un vettore di 7 elementi di tipo reale

Cliente * ***cliente***;
cliente = **new Cliente**( ); // ***cliente*** punta ad un **oggetto** della classe Cliente che viene allocato dinamicamente

***cliente*->setNome**("Mario"); // poiché cliente è un puntatore dobbiamo usare **->** per richiamare un metodo

```
class Cliente {
  private:
    char nome[20];
    char cognome[20];
    char indirizzo[30];
  public:
    void setNome(char _nome[20])
          {
             strcpy(nome, _nome);
          }
   };
```



Allocazione dinamica e rilascio della memoria in C++

- **delete per liberare memoria allocata in modo dinamico**

  Per ogni istruzione new, deve essere presente la corrispondente istruzione delete.
  Dopo un delete inoltre, è buona norma assegnare al puntatore il valore **NULL** (cioè rimuovere ogni indirizzo di memoria), questo perché **delete non cancella** il **puntatore né altera il suo contenuto.**
  L'unico effetto é di liberare la memoria puntata rendendola disponibile per ulteriori **allocazioni.**

Esempi:

```
int *P;
float *Q;
P = new  int;            // P punta ad una variabile di tipo intero
Q= new float[10];        // Q punta ad un vettore di 10 elementi di tipo
                            reale
.....
.....
delete P;        // libera la memoria (variabile semplice) puntata da P
delete [ ] Q;    // libera la memoria allocata per un array il cui
                    puntatore è Q
P = NULL;
Q = NULL;
```

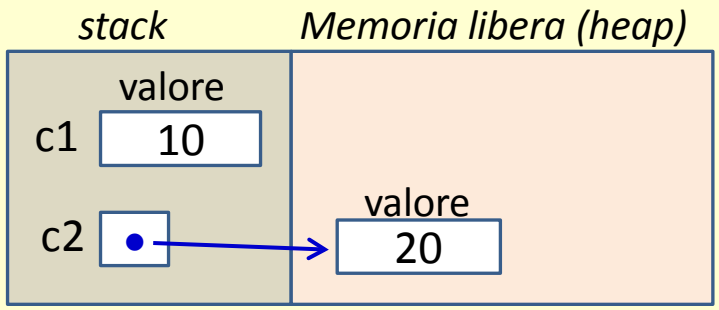
Allocazione dinamica e rilascio della memoria in C++

Esempio: ***oggetti*** *allocati staticamente e dinamicamente*

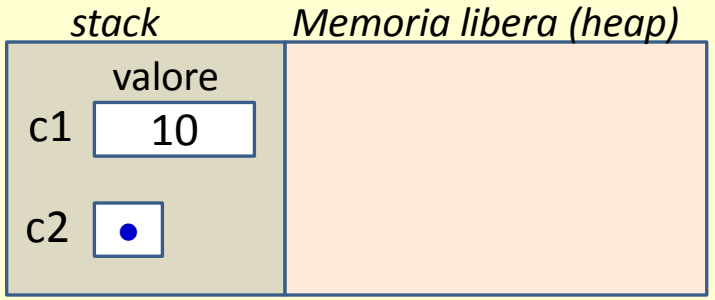
Calcola.cpp

```
#include "Counter.h"
#include <iostream>
using namespace std;

int main()
{
   Counter  c1(10);   // statica

   Counter *c2 = new Counter(20);  // dinamica

   cout << c1.getValore() << endl;

   cout << c2->getValore() << endl;

   delete c2;
}
```

```
class Counter {
  private: int valore;
  public:
     Counter():  valore(0) { }
     Counter(int n): valore(n) { }
     int getValore() { return valore; }
     void incrementa(int n)  {valore += n;}
};
```



Dopo l'istruzione delete:



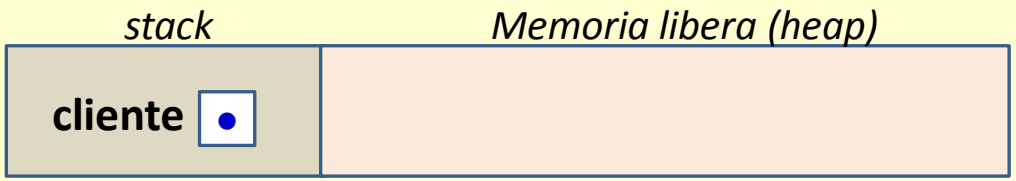
Allocazione dinamica e rilascio della memoria in C++

Esempio: ***oggetto*** *allocato dinamicamente*

```
int main()
{
    Cliente * cliente;     // puntatore
    cliente = new Cliente( );    // oggetto

    cliente->setNome("Mario");
    cliente->setCognome("Rossi");
    cliente->setIndirizzo("Via Roma,34");

    cout << "Il nome del cliente inserito è: " << cliente->getNome ( ) << endl;
    cout << "Il cognome del cliente inserito è: " << cliente->getCognome( ) << endl;
    cout << "L' indirizzo del cliente inserito è: " << cliente->getIndirizzo( ) << endl;

    delete cliente;
}
```

```
class Cliente {
  private:
     char nome[20];
     char cognome[20];
     char indirizzo[30];
  public:
     void setNome(char _nome[20])
              {
               strcpy(nome, _nome);
              }
  .....
  .....
};
```

Dopo l'istruzione delete:

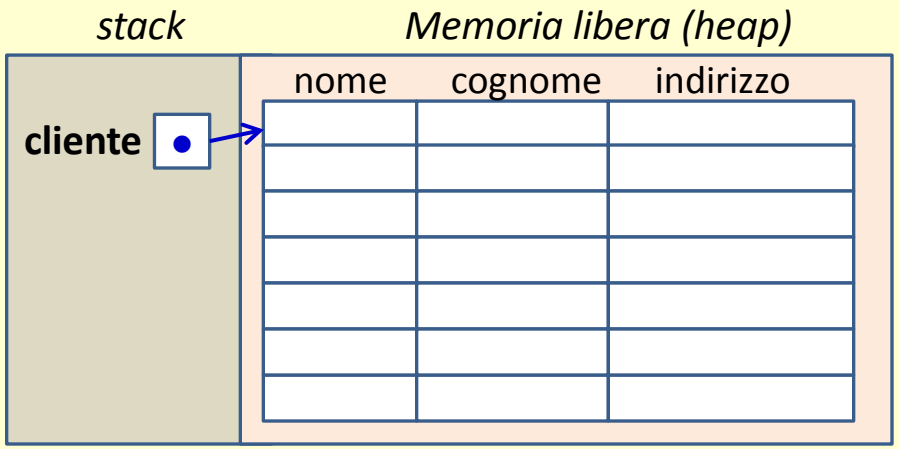


Allocazione dinamica e rilascio della memoria in C++

Esempio: ***array di oggetti*** *allocato dinamicamente*

```
int main()
{
    Cliente * cliente;    // puntatore
    int nClienti;
    char nome[20], cognome[20], indirizzo [30];
    cout<< "Inserisci il numero di clienti da gestire:  ";
    cin >> nClienti;
    cliente = new Cliente[nClienti];    // array di oggetti

    // caricamento clienti
    for (int i=0; i<nClienti; i++)
    {  cout<<"Inserisci nome, cognome e indirizzo: ";
       cin>> nome>>cognome>>indirizzo;
       cliente[i].setNome(nome);
       cliente[i].setCognome(cognome);
       cliente[i].setIndirizzo(indirizzo);
     }
     ......
     ......
     delete [ ] cliente;
}
```

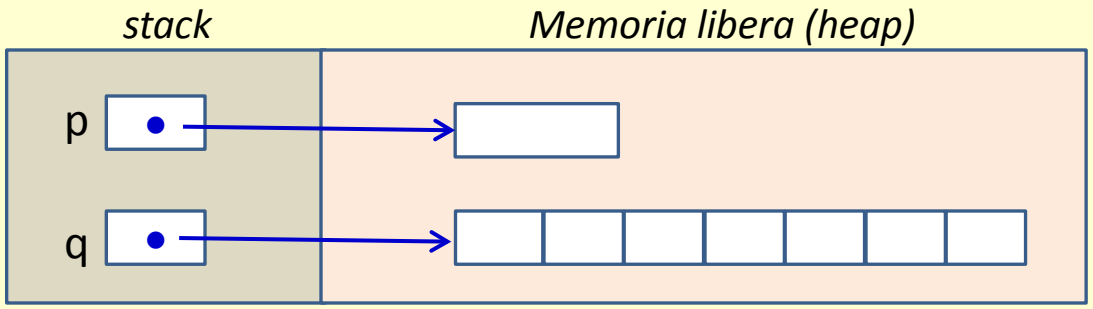
```
class Cliente {
   private:
      char nome[20];
      char cognome[20];
      char indirizzo[30];
   public:
      void setNome(char _nome[20])
                {
                strcpy(nome, _nome);
              }
      .....
      .....
 };
```



Allocazione dinamica e rilascio della memoria in C++

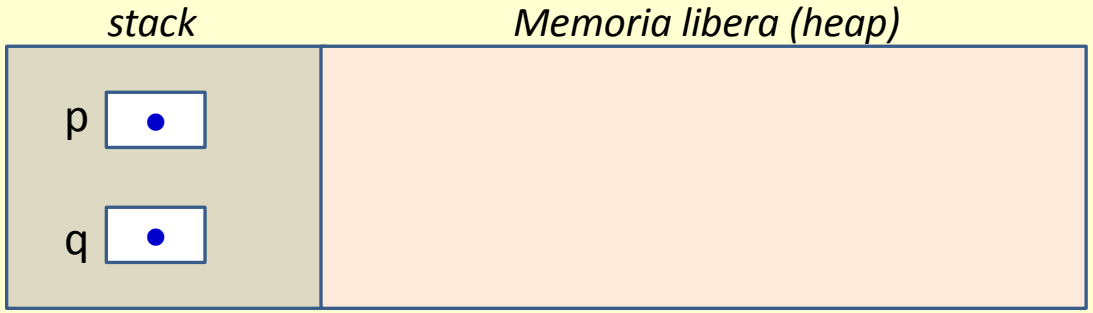
La sintassi è: tipodato *p = **new** *tipodato*;

tipodato *q = **new** *tipodato [numero_elementi]*;



La sintassi è: **delete** *p;*

delete [ ] *q;* // per liberare memoria allocata per un array



Allocazione dinamica e rilascio della memoria in C++

Attenzione al seguente esempio:

```
int  x=5;
int  *p = new int ;
p=&x;
```

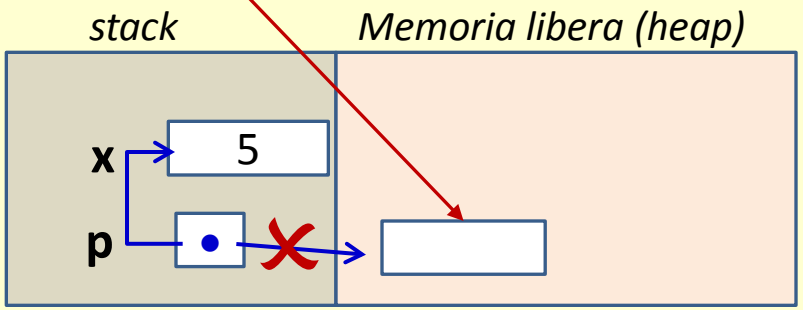
Quale significato hanno queste istruzioni ?

Allocazione dinamica e rilascio della memoria in C++

Attenzione al seguente esempio:

```
int  x=5;
int  *p = new int ;
p=&x;
```

// Memory leak!!



È stata allocata una locazione di tipo intero nella memoria dinamica.
Il **puntatore a tale memoria e andato perso**: memory leak e quindi **non sarà possibile deallocarla** all'interno del programma.

Tale memoria verra deallocata solo al termine del programma.

Questo potrebbe essere un problema, ad esempio: se il programma è scritto per restare in attesa e non terminare e si presentano molte situazioni di questo tipo la memoria potrebbe esaurirsi.