Allocazione dinamica della memoria

Memoria stack ed heap (1)

- **L'area di memoria stack é quella in cui viene allocato un *pacchetto* di dati non appena l'esecuzione passa dal programma chiamante a una funzione.**
- **Questo *pacchetto* (il quale contiene l'indirizzo di rientro nel programma chiamante, la lista degli argomenti passati alla funzione e tutte le variabili definite nella funzione) viene "impilato" sopra il *pacchetto* precedente (quello del programma chiamante) e poi automaticamente rimosso dalla memoria appena l'esecuzione della funzione é terminata.**
- **Le variabili automatiche definite nella funzione hanno *lifetime* limitato all'esecuzione della funzione stessa proprio perché, quando la funzione termina, il corrispondente *pacchetto* allocato nell'area stack viene rimosso.**

Memoria stack ed heap (2)

- **Esiste un'altra area di memoria che il programma può utilizzare. Questa area, detta heap, è soggetta alle seguenti regole:**
- **non é allocata automaticamente, ma può essere allocata o rimossa solo su esplicita richiesta del programma (allocazione dinamica della memoria);**
- **l'area allocata non é identificata da un nome, ma é accessibile esclusivamente tramite dereferenziazione di un puntatore;**
- **il suo *scope* coincide con quello del puntatore che contiene il suo indirizzo;**
- **La sua *lifetime* coincide con l'intera durata del programma, a meno che non venga esplicitamente deallocata; se il puntatore va *out of scope*, l'area non é più accessibile, ma continua a occupare memoria inutilmente: si verifica l'errore di *memory leak*.**

Operatore new

- **In C++, l'operatore new costruisce uno o più oggetti nell'area heap e ne restituisce l'indirizzo. In caso di errore (memoria non disponibile) restituisce NULL.**
- **Gli operandi di new (tutti alla sua destra) sono tre, di cui solo il primo é obbligatorio:**

  new tipo [dimensione] (valore iniziale)

- **tipo é il tipo (anche *astratto*) dell'oggetto (o degli oggetti) da creare;**
- **dimensione é il numero degli oggetti, che vengono sistemati nella memoria heap consecutivamente (come gli elementi di un array); se questo operando é omesso, viene costruito un solo oggetto; se é presente, l'indirizzo restituito da new punta al primo oggetto;**
- **valore iniziale é il valore con cui l'area allocata viene inizializzata (deve essere dello stesso tipo di tipo); se é omesso l'area non é inizializzata.**

Operatore delete

- **In C++, l'operatore delete (con un operando *opzionale* e l'altro obbligatorio) dealloca la memoria dell'area heap puntata dall'operando (obbligatorio). Non restituisce alcun valore e quindi deve essere usato da solo in un'istruzione.**
- **Contrariamente all'apparenza l'operatore delete non cancella il puntatore né altera il suo contenuto: l'unico effetto é di liberare la memoria puntata rendendola disponibile per ulteriori allocazioni.**
- **Se l'operando punta a un'area in cui è stato allocato un array di oggetti, bisogna inserire dopo delete l'operando *opzionale*, che consiste in una coppia di parentesi quadre (senza la dimensione dell'array, che il C++ é in grado di riconoscere automaticamente).**

- **L'operatore delete costituisce l'unico mezzo per deallocare memoria heap, che, altrimenti, sopravvive fino alla fine del programma, anche quando non é più raggiungibile.**

Esempi

*int *IDpt = new int;*	alloca un intero
*double *forza = new double(3.5);*	alloca un double e gli assegna il valore 3.5
*char *letter = new char;*	alloca un char
*int *pt = new int[1024];*	alloca un array di 1024 interi
delete IDpt;	libera la memoria
delete forza;	libera la memoria
delete letter;	libera la memoria
delete pt[];	libera la memoria occupata dall'array

Notate la differenza:

*int *pt = new int[1024];* // **crea un array di 1024 interi**

*int *pt = new int(1024);* // **crea un singolo intero che vale 1024**

Per inizializzare un array allocato dinamicamente:

```
int *buff = new int[1024];
for (int i = 0; i < 1024; i++)
{
    *buff = 52;          //Assigns 52 to each element;
    buff++;
}
```

oppure:

```
int *buff = new int[1024];
for (int i = 0; i < 1024; i++)
    buff[i] = 52;        //Assigns 52 to each element;
```