



UNIVERSITÀ
degli STUDI
di CATANIA

Allocazione dinamica di memoria in C++

Corso di programmazione I AA 2019/20

Corso di Laurea Triennale in Informatica

Prof. Giovanni Maria Farinella

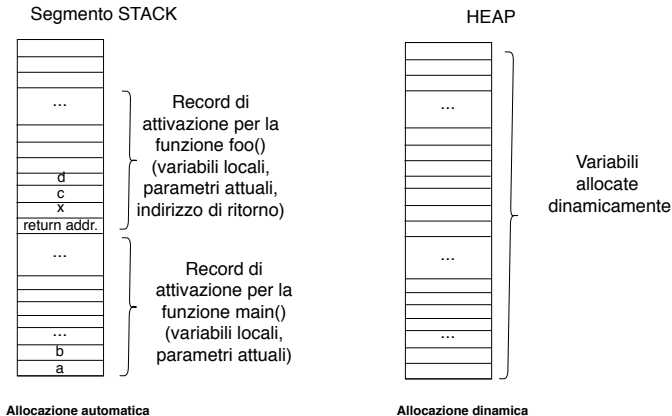
Web: <http://www.dmi.unict.it/farinella>

Email: gfarinella@dmf.unict.it

Dipartimento di Matematica e Informatica

Allocazione dinamica: Heap o Free Store

Lo heap è un altro segmento di memoria adibito ai dati che vengono allocati **dinamicamente**.



Allocazione: operatore `new` del C++

```
1  int *a = new int ;  
2  float *f = new float (0.9);  
3  *a = 10;  
4  double *arr = new double [10];
```

L'operatore `new` permette di operare allocazione dinamica.

Il segmento di memoria per allocazione dinamica è denominato **Heap** o **free store**.

Per istanziare un oggetto, per ospitare un array o una singola variabile di un determinato tipo.

Deallocazione: operatore delete

A differenza della allocazione automatica (stack), **la memoria allocata dinamicamente va successivamente liberata mediante l'operatore delete**

```
1  int *a = new int;  
2  //...  
3  delete a; //deallocazione
```

deallocazione di un blocco di memoria

```
1  double *arr = new double[10];  
2  //..  
3  delete [] arr;
```

Deallocazione: operatore delete

Il valore del puntatore (indirizzo di memoria) rimane invariato!

Si provi:

```
1  int *a = new int ;  
2  double *arr = new double [10];  
3  cout << arr << endl; // 0xaabb1244  
4  //...  
5  delete a; //deallocazione  
6  //..  
7  delete [] arr;  
8  cout << arr << endl; //0xaabb1244
```

Memory leak

```
1 double *arr = new double[10];  
2 //..  
3 double *v = new double[15];  
4 //..  
5 v = arr;
```



A seguito della **copia** di un differente indirizzo di memoria nella variabile `v` (“effetto **aliasing**”), si **perde il riferimento** (indirizzo) al blocco di memoria di 15 elementi double.

Deallocare il blocco di memoria con l'operatore **delete** ? Non più possibile!

Può essere un problema, ad esempio un Web server (centinaia di gg di uptime!)

Double Deletion

```
1  double *arr = new double [10];  
2  //..  
3  delete [] arr;  
4  //..  
5  delete [] arr;
```

Può capitare, soprattutto in un programma lungo e complesso di operare, per errore, una **doppia delete**.

Cosa succede alla riga 5? Come già detto in precedenza, il valore del puntatore, dopo la delete, rimane invariato.

Il tentativo di deallocazione avrà un comportamento **indefinito**, ovvero non predicibile e possibilmente **disastroso**.

Double Deletion

```
1  double *arr = new double [10];  
2  //..  
3  if(arr){  
4      delete [] arr;  
5      arr = nullptr;  
6  }  
7  //.. altri tentativi di deallocazione
```

Buona norma, ad ogni tentativo di deallocazione

- inserire un controllo sul valore del puntatore
- “azzerare” il puntatore dopo la delete

Premature deletion

```
1  double *arr = new double [10];  
2  //..  
3  double *v = arr;  
4  //..  
5  if(arr){  
6      delete [] arr;  
7      arr = nullptr;  
8  }  
9  //..  
10 v[5] = 4.56789; //!!!
```

Dopo la delete di arr, operata per errore, esso sarà `nullptr`, ma v conserva il vecchio valore di arr!!

Funzioni che restituiscono un puntatore

```
1  int *func(int k){  
2      int arr[k];  
3      for(int i=0; i<k; i++)  
4          arr[k] = 2*k;  
5      return arr;  
6  }  
7  
8  int *array = func(10);
```

Corretto? .. NO!

Infatti `arr` allocato automaticamente nello stack. Dobbiamo ricordare cosa accade dopo l'istruzione `return` al record di attivazione dello stack della funzione..

Funzioni che restituiscono un puntatore

```
1  int *func(int k){  
2      int *arr = new int[k];  
3      for(int i=0; i<k; i++)  
4          arr[i] = 2*k;  
5      return arr;  
6  }  
7  
8  int *array = func(10);
```

Corretto? .. SI!

arr allocato dinamicamente nel free store.

Memoria allocata per arr nel free store non sarà liberata fino a chiamata delete [].

Allocazione dinamica in C: malloc() e free()

```
1 #include <stdlib.h>
2 double *arr = (double *) malloc(sizeof(double) * 10);
3 //..
4 free(arr); // deallocazione
```

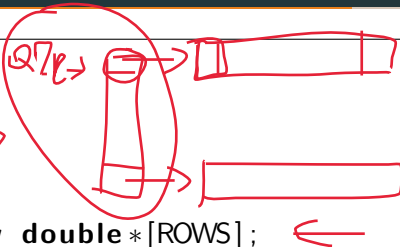
malloc() alloca dinamicamente un blocco di memoria:

- argomento è **dimensione in byte** (importante l'uso di sizeof: portabilità!)
- **restituisce un puntatore generico**, ovvero di tipo void *, per questo bisogna operare un **type casting** al tipo desiderato.

La funzione free() **libera la memoria precedentemente allocata**, come delete in C++.

Allocazione dinamica di un array multidimensionale

```
1 #define COLS 10
2 #define ROWS 5
3 double *arr[ROWS];
4 //oppure
5 double **arr = new double*[ROWS];
6 for(int i=0; i<ROWS; i++)
7     arr[i]=new double[COLS];
```



1. Allocazione automatica (nello stack, linea 3) o dinamica (free store/heap, linea 5) di un **vettore di (ROWS) puntatori a tipo double**.
2. Allocazione dinamica di ROWS **vettori di COLS celle di tipo double**.

Allocazione dinamica di un array multidimensionale



Allocazione dinamica di un array multidimensionale

```
1  double *arr [ROWS];  
2  //oppure  
3  double **arr = new double*[ROWS];  
4  for(int i=0; i<ROWS; i++)  
5      arr[i]=new double[COLS];
```

Accesso agli elementi (NB linea 3)

```
1  arr[i][j];  
2  (*(arr+i))[j];  
3  *(arr+i)[j]; //NO: ==*(arr+i+j)==arr[i+j][0]  
4  *(arr[i] + j);  
5  *(* (arr+i) + j);
```

FINE