UNIVERSITÀ
degli STUDI
di CATANIA

Allocazione dinamica di memoria in C

Corso di programmazione I (A-E / O-Z) AA 2025/26

Corso di Laurea Triennale in Informatica

Fabrizio Messina
fabrizio.messina@unict.it

Dipartimento di Matematica e Informatica

Allocazione dinamica: Heap o Free Store

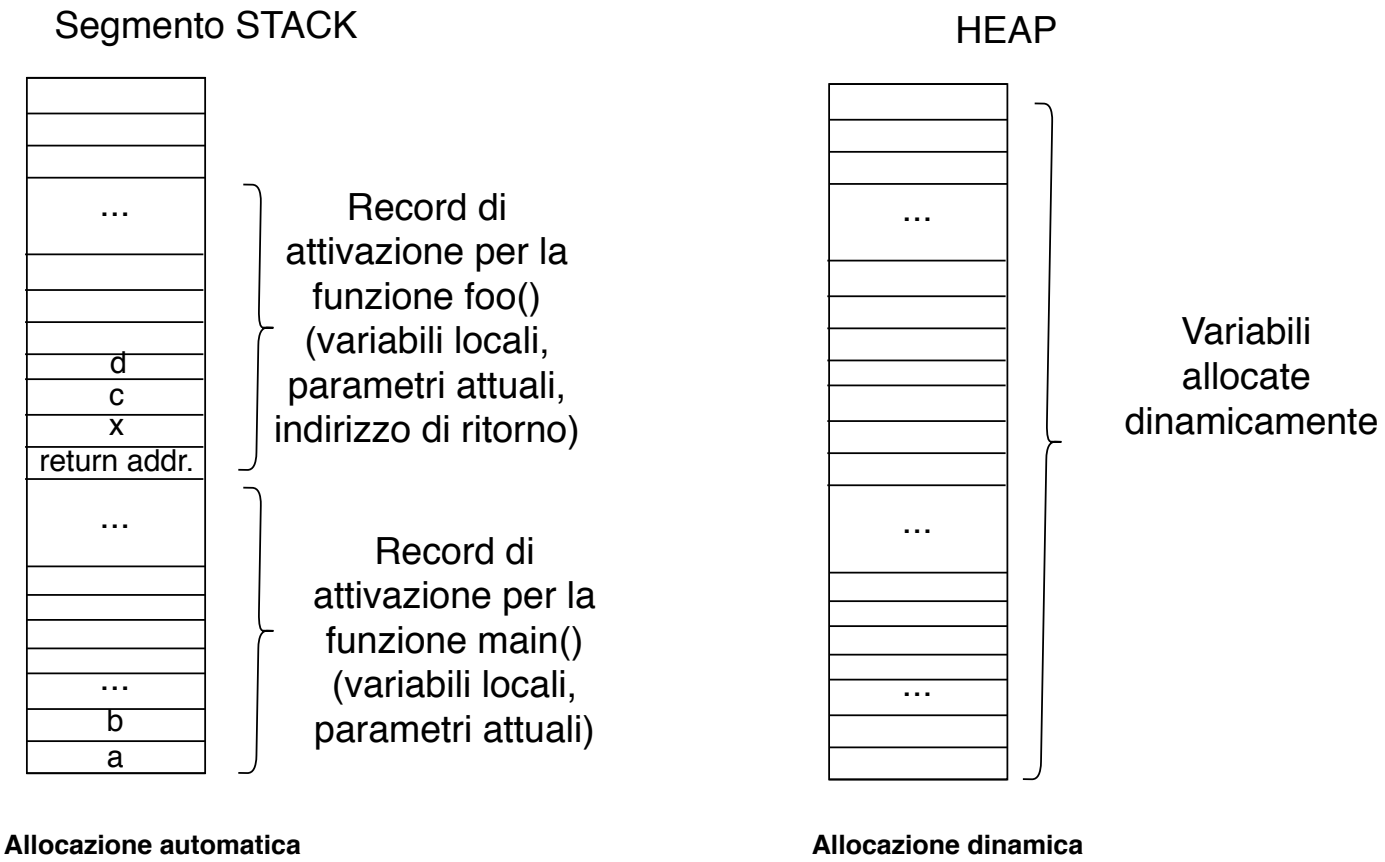
Lo HEAP è un segmento di memoria che permette alle applicazioni di memorizzare dati in modo **dinamico.**

In questo caso l'allocazione a la deallocazione avvengono mediante apposite chiamate a funzioni:

funzioni `malloc()` e `free` della libreria standard in linguaggio C;

Header `stdlib.h`

Allocazione dinamica: Heap o Free Store



Allocazione dinamica: Heap o Free Store

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

`malloc()` alloca dinamicamente un blocco di memoria nello HEAP (o *Free Store*):

- 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**.

Deallocazione: funzione `free`

1) **Allocazione automatica** (stack): memoria automaticamente deallocata dopo ultima istruzione del blocco in cui la variabile è stata dichiarata.

2) Viceversa: la memoria **allocata dinamicamente** va successivamente liberata mediante la funzione `free`.

```
1   double *arr = malloc(sizeof(double) * 10);
2   //..
3   free(arr);
```

Deallocazione: funzione `free`

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

Si provi:

```
1    double *arr = malloc(sizeof(double) * 10);
2    printf("%p", arr);
3    //...
4    free(arr); //deallocazione
5    //..
6    printf("%p", arr);
```

Memory leak / aliasing

```
1    double *arr = malloc(sizeof(double) * 10);
2    //..
3    double *v = malloc(sizeof(double) * 10);
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 degli elementi double.

Deallocare il blocco di memoria referenziato da v?? Non più possibile (manca il puntatore..) $\rightarrow$ memory leak!!

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

Double free

```
1   double *arr = malloc(sizeof(double)*10);
2   //..
3   free(arr);
4   //..
5   free(arr); // !!! comportamento indefinito!
```

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

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

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

Double free

```
1    double *arr = malloc(sizeof(double) * 10);
2    //..
3    if(arr){
4       free(arr);
5       arr = NULL;
6    }
7    //..altri tentativi di deallocazione
```

Buona norma, ad ogni tentativo di deallocazione

- inserire un controllo sul valore del puntatore
- "azzerare" il puntatore dopo invocazione a funzione free()

Premature free

```
 1    double *arr = malloc(sizeof(double) * 10);
 2    //..
 3    double *v = arr;  // v e' alias di arr
 4    //..
 5    if(arr){
 6       free(arr);
 7       arr = NULL;
 8    }
 9    //..
10    v[5] = 4.56789; //!!!
```

Dopo la `free` di `arr`, operata per errore, esso sarà `NULL`, 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[i] = 2*i;
5     return arr; // NO!!
6   }
7
8   int *array = func(10);
```

Corretto? .. NO!

Infatti `arr` rappresenta blocco di memoria allocato nello stack.

$\rightarrow$ dopo istruzione `return` record di attivazione per la funzione `func()` distrutto..

Funzioni che restituiscono un puntatore

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

`arr` allocato dinamicamente nel free store.

Memoria allocata per `arr` nel free store non sarà liberata fino a chiamata `free(arr)`.

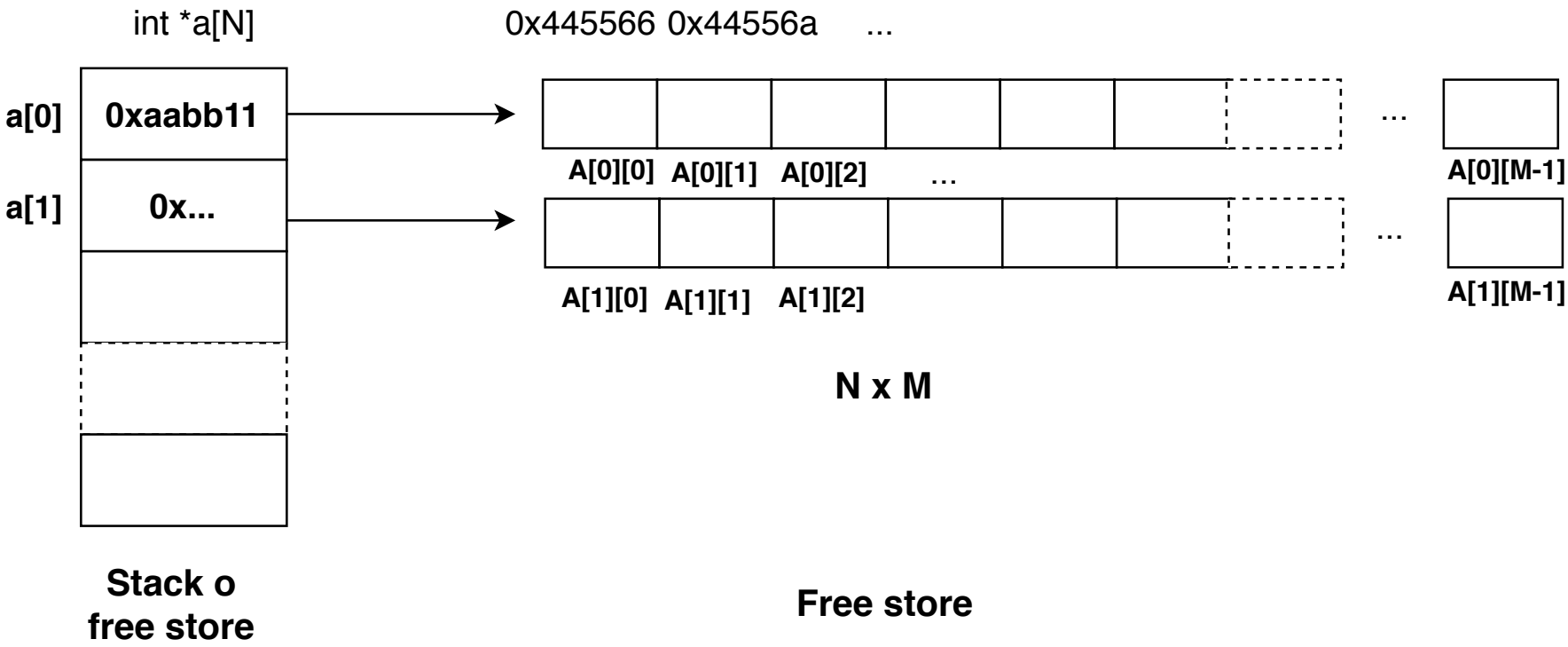
Allocazione dinamica di un array multidimensionale

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

Accesso agli elementi **sintatticamente equivalente** a quello relativo ad array a due dimensioni allocati nello stack.

```
1    arr[i][j];
2    (*(arr+i))[j];
3    *(arr[i] + j);
4    *(*(arr+i) + j);
```

Passaggio di array a funzioni (allocazione dinamica).

Passaggio di array ad una dimensione

Il passaggio di un array come parametro di una funzione avviene sempre per indirizzo. (il nome di uno array è un puntatore costante al primo elemento dello array..).

```
1   void init(int *v, int n){
2     //...
3     for(int j=0; j<n; j++){
4       v[j] = 0;
5     }
6   }
7   int main(){
8     // ...
9     int *x = malloc(sizeof(int) * 10); // free store
10    init(x, 10);
11  }
```

Passaggio di array ad una dimensione

Forma equivalente..

```
1   void init(int v[], int n){ //equivalente a int *v, ...
2     //...
3     for(int j=0; j<n; j++){
4       v[j] = 0;
5     }
6   }
7   int main(){
8     // ...
9     int *x = malloc(sizeof(int)*10);
10    init(x, 10);
11  }
```

Passaggio di array multidimensionali

Per il passaggio di **array multidimensionali allocati nel free store (HEAP)**, il prototipo della funzione che riceve il dato deve specificare un array di puntatori di un certo tipo che dipende dal numero di dimensioni dell'array.

```
1   void init(int **v, int n, int m);  // OK
2   void init(int *v[], int n, int m);  // OK
3   void main(){
4     int **v = malloc(sizeof(int *) * 5 );
5     for(unsigned short i = 0; i<5; i++)
6       v[i] = malloc(sizeof(int) * 10);
7     init(v, 5, 10);
8   }
```

Passaggio di array multidimensionali

Array a tre dimensioni..

```
1   void init(int ***w, int n, int m, int l);  //OK
2   void init(int **w[], int n, int m, int l);  //OK
3   void main(){
4     int ***v = malloc(sizeof(int **) * 5);
5     for(unsigned short i = 0; i<5; i++){
6       v[i] = malloc(sizeof(int *) * 7);
7       for(unsigned short j = 0; j<7; j++)
8         v[i][j] = malloc(sizeof(int)*9);
9     }
10    init(v, 5, 7, 9);
11  }
```

Funzioni calloc e realloc

La funzione `calloc` appartiene alla stessa famiglia della funzione `malloc` (allocazione dinamica nello HEAP o free store). Due argomenti:

- dimensione del singolo elemento da allocare
- numero di elementi totali da allocare

A differenza della funzione `malloc`, in questo caso tutti i byte del blocco saranno inizializzati a zero.

```
1 double *ptr = (double *) calloc(SIZE, sizeof(double));
```

Funzioni calloc e realloc

La funzione `realloc`:

- consente di ridimensionare un blocco di memoria precedentemente allocato nello HEAP
- se la nuova dimensione è maggiore della dimensione attuale, i dati già presenti nel blocco non subiranno cambiamenti.

```
1 double *ptr = (double *) malloc(sizeof(double) * SIZE);
2 //...
3 ptr = (double *) realloc(ptr, SIZE*2);
```

Esempi svolti

Esempi

16_01.c

16_02.c

16_03.c

16_04.c

16_05.c

16_06.c

16_07.c

Homework H16.1

Codificare una funzione C che prenda in input un parametro formale matrice M di dimensioni $n \times m$ di stringhe (puntatori a caratteri), e che restituisca il valore true (1) se esiste almeno una colonna in M che abbia un egual numero di stringhe palindrome di una delle righe di M.

Homework H16.2

Codificare una funzione in C che prenda in input tre parametri formali: una matrice di stringhe S di dimensioni $n \times m$, un array di caratteri C che contiene per ipotesi elementi distinti, ed un float w. Il metodo restituisca true se esiste almeno una riga o una colonna della matrice tale che la percentuale di caratteri di C presenti in essa è maggiore di w.

Homework H16.3

Codificare una funzione in C che prenda in input due parametri formali: una matrice A di dimensioni $n \times m$ ed una matrice B di dimensioni $k \times n$ entrambe di interi positivi. Il metodo restituisca un array C di double di dimensione n nel quale lo i-esimo elemento dello array C sia uguale al rapporto tra la somma degli elementi della riga i-esima di A e il prodotto degli elementi della colonna i-esima di B.

Homework H16.4

Codificare una funzione in C che prenda in input un parametro formale array A di interi di dimensioni $n \times m$ di elementi distinti ed un array B di double di dimensioni $k \times n$, e restituisca un array di n interi nel quale lo i-esimo elemento sia uguale alla media aritmetica degli elementi presenti sia nella riga i-esima di A che nella colonna i-esima di B. NB: per decidere se un elemento int della matrice A è uguale ad un elemento double della matrice B si calcoli l'approssimazione all'intero più vicino di quest'ultimo.

Homework H16.5

Codificare una funzione in C che prenda in input un parametro formale matrice S di puntatori a carattere di dimensione $n \times m$ (S quindi contiene stringhe), uno short w ed uno short k, e restituisca il valore booleano `true` (1) se in S sono presenti almeno una riga ed almeno una colonna che presentano ciascuna almeno w stringhe di lunghezza minore di k.

Homework H16.6

Codificare una funzione in C che prenda in input un parametro formale matrice A di interi di dimensioni $n \times m$, uno short k ed uno short w, e restituisca il valore booleano true (1) se la matrice contiene almeno w colonne che contengono almeno una sequenza di interi monotona crescente di lunghezza maggiore o uguale a k.

Homework H16.7

Codificare una funzione in C che prenda in input un parametro formale A matrice di stringhe di dimensione $n \times m$, ed una stringa s. Il metodo restituisca true se esistono almeno due stringhe in A che contengono la stringa s e che abbiano differenti indici di riga e di colonna.

Homework H16.8

Codificare una funzione C che prenda in input un parametro formale matrice di stringhe A di dimensioni $n \times m$, una matrice di caratteri C di dimensioni $k \times z$, uno short w e restituisca true se esiste almeno una riga o una colonna di A che contiene almeno una sequenza contigua di w o piu' caratteri che si trovano in una riga o in una colonna di C.

Homework H16.9

Codificare una funzione C che prenda in input una matrice di caratteri di dimensione $n \times m$ ed un ulteriore parametro intero a, e restituisca il valore booleano *true* (1) se esiste almeno una riga o una colonna che contiene almeno a caratteri che siano tutti minuscoli o tutti maiuscoli e che abbiano posizioni adiacenti. NB: Si assuma che la matrice contenga solo caratteri corrispondenti alle lettere dell'alfabeto.

Homework H16.10

Codificare una funzione C che prenda un parametro formale matrice quadrata di stringhe (puntatori a caratteri) di dimensioni $n \times n$, uno short k ed una ulteriore stringa s, e restituisca un array di n stringhe (puntatori a caratteri) il cui generico elemento di indice i contenga la i-esima stringa della diagonale secondaria se questa stringa ha lunghezza maggiore o uguale a k e inizia con la stringa s, altrimenti la corrispondente stringa (i-esima) della diagonale principale.

Homework H16.11

Codificare una funzione in C che prenda in input due matrici di interi, A e B di dimensioni $k \times n$ ed $n \times k$ rispettivamente, e restituisca uno array monodimensionale di k elementi double in cui lo i-esimo elemento sia uguale alla differenza tra la media aritmetica degli elementi della riga i-esima di A ed il minimo valore degli elementi della colonna i-esima di B.

Homework H16.12

Codificare una funzione C che prenda in input un array monodimensionale S di n puntatori a caratteri (stringhe), ed inoltre una matrice C di caratteri distinti (char) ed uno short k, e restituisca un array di n puntatori a caratteri (strighe) in cui il generico elemento di indice i punta alla i-esima stringa in S solo se questa contiene almeno k caratteri in C, altrimenti `NULL`.

Homework H16.13

Codificare una funzione in C che prenda in input due parametri formali: una matrice A di stringhe (puntatori a caratteri) ed una matrice B di short, entrambe di dimensioni $n \times m$. Il metodo restituisca un array di n stringhe nel quale il generico elemento di indice i sarà uguale alla concatenazione delle sole stringhe della riga i-esima di A che hanno lunghezza pari o maggiore del corrispondente numero in B (cioè bisogna confrontare la lunghezza della stringa a_{ij} con il numero b_{ij}.

Homework H16.14

Codificare una funzione in C che prenda in input una matrice S di stringhe (puntattori a caratteri) di dimensioni $n \times m$ ed un array B di short di dimensione m e che restituisca uno short che rappresenti l'indice della riga in S con il maggior numero di stringhe aventi lunghezza minore o uguale del corrispondente numero nello array B (NB: in pratica la lunghezza della stringa di indici (i,j) va confrontata con il numero di indice j in B).

Homework H16.15

Codificare una funzione in C che prenda in input una matrice di stringhe S (puntatori a caratteri) di dimensione $n \times m$ e due array di caratteri C e D di dimensione n e restituisca un array di short nel quale il generico elemento di posto i contiene il numero di stringhe che iniziano con il carattere di C di indice i e finiscono con il carattere in D di ndice i.

Homework H16.16

Codificare una funzione in C che prenda un parametro formale matrice A di double di dimensioni $n \times k$ ed un array B di interi di dimensione n, e restituisca l'indice della colonna in A che contiene il maggior numero di elementi tali che l'approssimazione dell'elemento stesso al numero intero più vicino sia uguale al corrispondente elemento intero in B (NB: in pratica il generico elemento di indice (i,j) va confrontato con l'elemento in B di indice i).