

Corso di Programmazione e strutture dati

Docente di Laboratorio: Marco Romano

Email: marromano@unisa.it

PUNTATORI E ALLOCAZIONE DINAMICA DELLA MEMORIA


ARRAY: ALCUNE OSSERVAZIONI

1. **Cardinalità:** dimensione massima
2. **Riempimento:** dimensione utilizzata in un certo momento
 - può variare a causa di operazioni di inserimento/rimozione

Dobbiamo prevedere però una cardinalità molto grande, sufficiente per tutte le esecuzioni del programma

Anche se non si prevede di inserire/rimuovere dinamicamente elementi ...

... con evidente spreco di memoria



	0.	1.	2.	3.	4.	5.	6.	7.
V	3	5	1	7	11			

Cardinalità/dimensione fisica: 8
Riempimento/dimensione logica: 5

Come facciamo ad allocare solo la memoria che effettivamente ci serve per un array ?

ALLOCAZIONE DINAMICA DELLA MEMORIA

Allocazione di memoria al **run-time**:

- Permette di creare strutture dati la cui dimensione varia (aumenta o diminuisce) durante l'esecuzione, in funzione delle necessità
- Viene usata spesso per stringhe, array e strutture
- Per allocare dinamicamente la memoria esistono delle funzioni specifiche ...

ALLOCAZIONE DINAMICA DELLA MEMORIA: LE FUNZIONI

- La libreria stdlib mette a disposizione tre funzioni per l'allocazione dinamica della memoria:

1. void ***malloc**(size_t size);

- Alloca un blocco di memoria di size bytes senza iniziarlo
- size_t è un tipo intero senza segno definito nella libreria del C
- Restituisce il puntatore al blocco

2. void ***calloc**(size_t nelements, size_t elementSize);

- Alloca un blocco di memoria di di nelements * elementSize bytes e lo inizializza a 0 (clear) e restituisce il puntatore al blocco

3. void ***realloc**(void *pointer, size_t size);

- Cambia la dimensione del blocco di memoria precedentemente allocato puntato da pointer
- Restituisce il puntatore ad una zona di memoria di dimensione size, che contiene gli stessi dati della vecchia regione indirizzata da pointer troncata alla fine nel caso la nuova dimensione sia minore di quella precedente).

```
1 //allocare un vettore di n elementi di tipo intero
2 int *a, n=10;
3 a=(int *) malloc(n*sizeof(int));
```

```
1 //allocare un vettore di n elementi di tipo intero
2 int *a, n=10;
3 a=(int *) calloc(n,sizeof(int));
```

ESEMPIO

Il programma prende in input un vettore di n interi e ne calcola la media

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main () {
5     int *a, i, n;
6     float media;
7
8
9     printf("Quanti valori?" );
10    scanf("%d", &n);
11    a=(int *) calloc(n,sizeof(int));
12
13    for(i=0;i<n; i++)    {
14        printf("Elemento %d: ", i+1);
15        scanf("%d", &a[i]) ;
16    }
17
18    media=0;
19    for(i=0;i<n;i++)
20        media=media+a[i];
21
22    media/=n;
23    printf ("La media e': %.2f" , media );
24
25    return 0;
26 }
```

PUNTATORI

- Un **puntatore** è una variabile che contiene l'indirizzo di un'altra variabile
- I puntatori sono “**type bound**” cioè ad ogni puntatore è associato il tipo a cui il puntatore si riferisce
- Nella dichiarazione di un puntatore bisogna specificare un asterisco (*) prima del nome della variabile pointer: $T *p$

```
1  int *pointer;    // puntatore a intero
2  char *pun_car;   // puntatore a carattere
3  float *flt_pnt;  // puntatore a float
```


DEREFERENZIAZIONE

- L'accesso all'oggetto puntato avviene attraverso l'operatore di dereferenziazione *
- Prima di poter usare un **pointer** questo deve essere inizializzato, ovvero deve contenere l'indirizzo di un oggetto

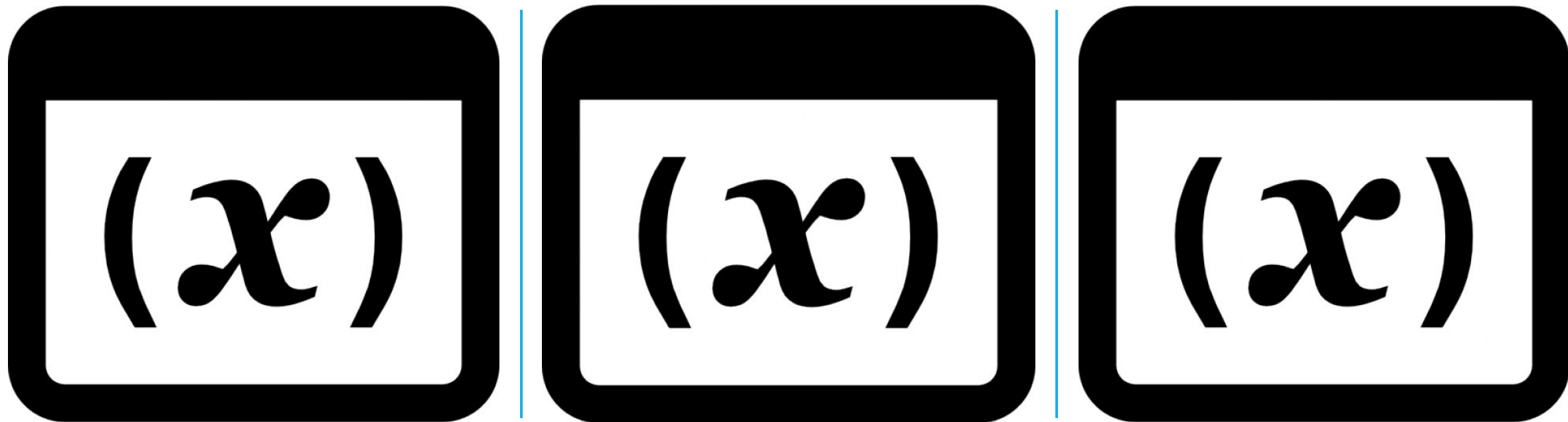
```
1  int j=2, x=1;  
2  int *a=j;  
3  
4  a // contiene l'indirizzo di j  
5  *a // contiene il contenuto di j  
6  *a=3; //modifica il contenuto di j  
7  a= &x; //muove il puntatore alla variabile x
```

OPERATORE DI INDIRIZZO

Per ottenere l'indirizzo di un oggetto si usa l'operatore unario **&**.

```
int volume, *vol_ptr;  
vol_ptr = &volume;
```

```
1  int i = 10, *p1;  
2  p1 = &i;  
3  printf("%d \n", *p1);
```



GESTIONE DELLE VARIABILI IN C |

TIPI DI VARIABILI IN C

1. **Globali:**

- Dichiarate esternamente alle funzioni
- Visibili a tutte le funzioni la cui definizione segue la dichiarazione della variabile nel file sorgente
- Sono dette **statiche**, perché la loro allocazione in memoria avviene all'atto del caricamento del programma (e la loro deallocazione al termine del programma)

2. **Locali:** dichiarate e visibili solo all'interno di una funzione

3. **Automatiche:**

- Dichiarate in blocchi interni alle funzioni
- Vengono allocate in memoria a tempo di esecuzione (dell'istruzione dichiarativa) e deallocate al termine del blocco



Scope: parte del programma in cui è attiva una dichiarazione (dice quando può essere usato un identificatore)



Visibilità: parte del programma in cui è accessibile una variabile (non sempre coincide con lo scope ...)



Durata: periodo durante il quale una variabile è allocata in memoria

SCOPE, VISIBILITÀ E DURATA

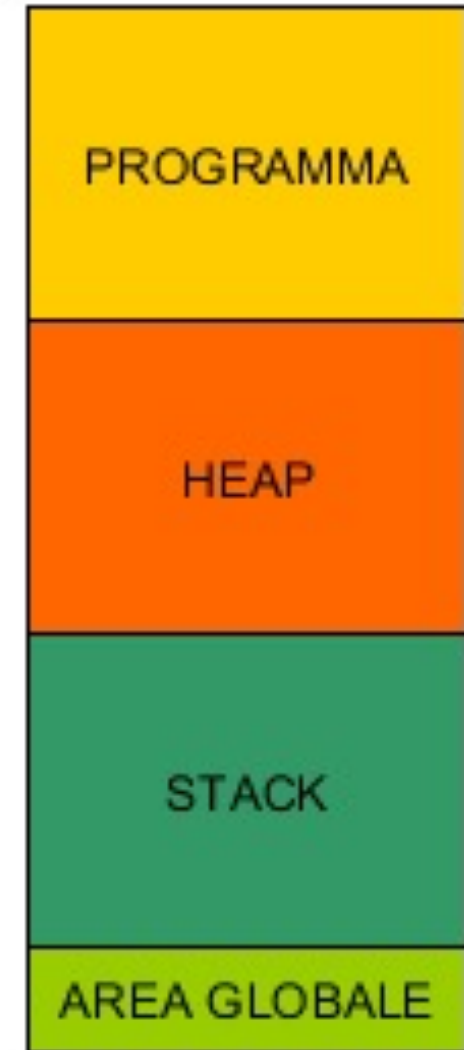
SCOPE E VISIBILITÀ: ESEMPIO

- ➡ `int n` (globale)
 - ➡ **scope:** intero file
 - ➡ **visibilità:** non è visibile nel `main`
- ➡ `long n` (locale)
 - ➡ **scope:** `main`
 - ➡ **visibilità:** non è visibile nel blocco interno
- ➡ `double n` (automatica)
 - ➡ **scope:** blocco interno
 - ➡ **visibilità:** coincide con lo scope

```
1  int n;  
2  ...  
3  int main()  
4  {  
5      long n;  
6      ...  
7      {  
8          double n;  
9          ...  
10     }  
11     ...  
12 }
```

DURATA: TRE AREE DI ALLOCAZIONE

- Il Sistema Operativo riserva ad un processo (un programma in esecuzione), un segmento di memoria RAM
- Questo, in generale è suddiviso in quattro distinte aree di memoria:
 1. L'**area del programma**, che contiene le istruzioni macchina del programma;
 2. L'**area globale**, che contiene le costanti e le variabili globali;
 3. Lo **stack**, che contiene la pila dei *record di attivazione* creati durante ciascuna chiamata delle funzioni;
 4. L'**heap**, che contiene le variabili allocate dinamicamente.





THE GNU C REFERENCE MANUAL

<https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html>

ESERCITAZIONE

- Modificare il modulo `vettore.c` sviluppando la funzione `input_array_dyn` descritta dalla seguente firma:

```
int *input_array_dyn(int *size, char* line);
```

- La funzione prende in input un buffer di caratteri (`line`) riempito nel programma principale e una variabile di tipo intero (`size`) passata per riferimento. Restituisce tramite return un puntatore a un vettore di interi estratti da `line` e la dimensione del nuovo vettore tramite la variabile passata per riferimento.
- Provare il modulo così modificato con il main qui proposto

```
1 #include <stdio.h>
2 #include "vettore.h"
3
4 int main()
5 {
6     int n;
7     char line[100];
8     int *a;
9
10    printf("Inserisci il vettore: ");
11    scanf("%[^\n]", line);
12    //legge un'intera riga fino al newline "\n"
13    a = input_array_dyn(&n, line);
14    bubble_sort(a, n);
15    output_array(a, n);
16
17    return 0;
18 }
```