C: Oggetti dinamici

Allocazione memoria

Processo utilizzato per assegnare la memoria ad un programma.

Allocazione **statica** (in fase di compilazione - stack).

Allocazione **dinamica** (in fase di esecuzione - heap).

Lo stack è un'area della memoria (*di dimensione fissa*) che conserva le variabili locali, i parametri delle funzioni e le chiamate alla stesse funzioni.
Le variabili locali sono attive solo durante la chiamata ad una specifica funzione, dopo vengono "rilasciate".

Allocazione memoria: statica

Stack:

- Allocata / deallocata *automaticamente* dal compilatore
- La dimensione è fissa e non può "crescere" (problema dello stack overflow)
- LIFO
- Accesso più rapido

Allocazione memoria: dinamica

Heap:

- Allocata / deallocata *manualmente* tramite l'uso di specifiche funzioni
- La dimensione può cambiare ed è meno limitata (problema della frammentazione)
- Nessun ordine specifico
- Accesso meno rapido

Oggetti dinamici

I *puntatori* possono essere usati per la creazione e la gestione di oggetti dinamici

Gli oggetti dinamici sono creati durante l'esecuzione del programma. Non è quindi necessaria una dichiarazione esplicita: *non serve il nome e non serve conoscerne il numero* a priori.

La memoria viene allocata attraverso l'utilizzo di specifiche funzioni: per esempio, `malloc()` (restituisce un puntatore)

Allocazione dinamica della memoria

Utilizzo della funzione `malloc()` - serve `stdlib.h`

```
p = (int *) malloc (sizeof(int));
```

Questa istruzione permette di allocare (dinamicamente) la memoria necessaria per un intero che sarà accessibile utilizzando il puntatore p.

`(int *)` è un *cast* al tipo specifico.

- Da questo momento possiamo accedere (tramite il puntatore p) a quella locazione di memoria e quindi all'intero senza l'utilizzo di una dichiarazione esplicita.

Allocazione dinamica della memoria

```
p = (int *) malloc (sizeof(int));
```

Possiamo modificare il valore puntato da `p` utilizzando la già nota rappresentazione:

```
*p=10;
```

E' necessario deallocare lo spazio per evitare che diventi successivamente inutilizzabile!

```
free(p);
```

Esempio: stack vs heap

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *p; //p è nello stack

    p=(int *)malloc(sizeof(int));
    //malloc restituisce un indirizzo di heap
    *p=10; //modifica di heap

    printf("%p %p %d",&p,p,*p);
    }
```

Esempio: step 1

```c
#include <stdio.h>

int main() {
    int n1,n2;

    printf("Inserisci un numero: ");
    scanf("%d",&n1);
    printf("Inserisci un altro numero: ");
    scanf("%d",&n2);

    printf("La somma vale %d", n1 + n2);
    return 0;
}
```

Esempio: step 2

```c
#include <stdio.h>

int main() {
    int n1,n2;
    int *p1,*p2;

    printf("Inserisci un numero: ");
    scanf("%d",&n1);
    printf("Inserisci un altro numero: ");
    scanf("%d",&n2);

    p1=&n1;
    p2=&n2;
    printf("La somma vale %d", *p1 + *p2);
    return 0;
}
```

Esempio: step 3

```c
#include <stdio.h>

int main() {
    int n1,n2;
    int *p1,*p2;

    p1=&n1;
    p2=&n2;

    printf("Inserisci un numero: ");
    scanf("%d",p1);
    printf("Inserisci un altro numero: ");
    scanf("%d",p2);

    printf("La somma vale %d", *p1 + *p2);
    return 0;
}
```

Esempio: step 4

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *p1,*p2;

    p1=(int *) malloc(sizeof(int));
    p2=(int *) malloc(sizeof(int));

    printf("Inserisci un numero: ");
    scanf("%d",p1);
    printf("Inserisci un altro numero: ");
    scanf("%d",p2);

    printf("La somma vale %d", *p1 + *p2);
    free(p1);
    free(p2);
    return 0;
}
```

Array e liste concatenate

- Array:
  - Occupazione memoria (sovrastima)
  - Velocità (inserimento/eliminazione e spostamento)
- Liste lineari:
  - Insieme di elementi omogenei memorizzati in una posizione qualsiasi (semplicità di inserimento/eliminazione e spostamento)

Array dinamico: malloc()

```c
#include <stdio.h>
#include <stdlib.h>

int main () {
    int i,n;
    int *ptr;

    printf("Inserisce dimensione array: ");
    scanf("%d",&n);

    ptr=(int*) malloc(n * sizeof(int));
     for(i=0;i<n;i++) {
         printf("\nInserisce valore: ");
         scanf("%d",ptr+i);
     }

    for(i=0;i<n;i++)
        printf("%d ",*(ptr+i));

    free(ptr);
}
```

Array dinamico: calloc()

```c
#include <stdio.h>
#include <stdlib.h>

int main () {
    int i,n;
    int *ptr;

    printf("Inserisce dimensione array: ");
    scanf("%d",&n);

    ptr=(int*) calloc(n, sizeof(int)); //init to 0!
     for(i=0;i<n;i++) {
         printf("\nInserisce valore: ");
         scanf("%d",ptr+i);
     }

    for(i=0;i<n;i++)
        printf("%d ",*(ptr+i));

    free(ptr);
}
```

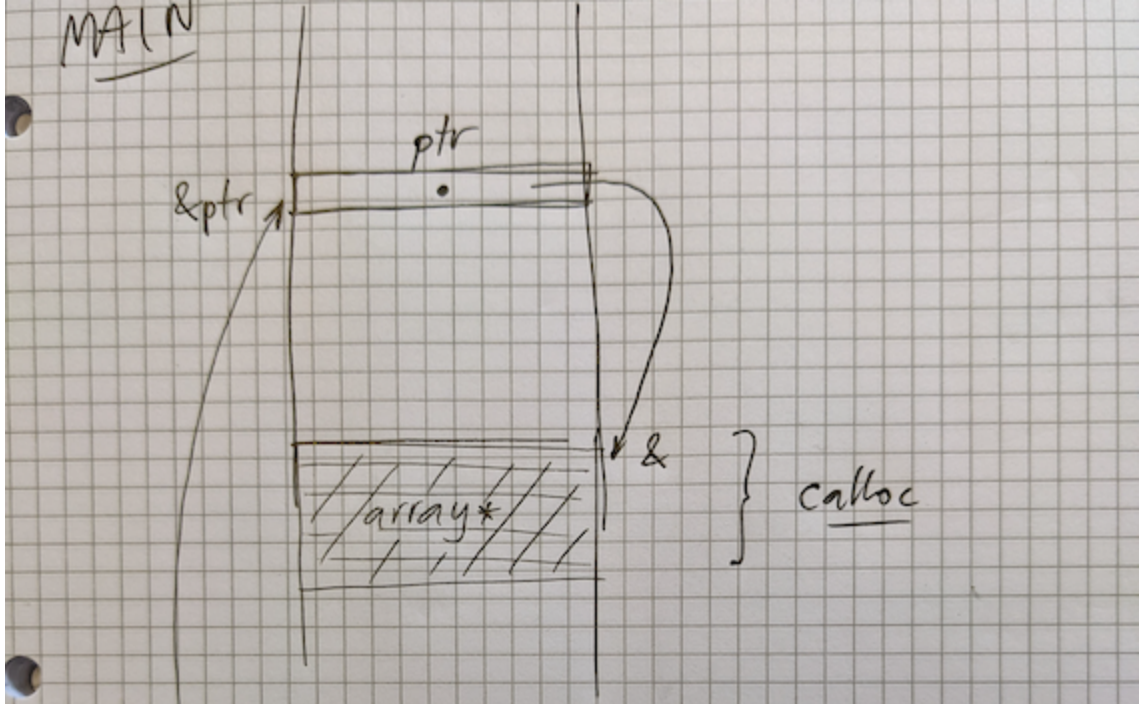
Array dinamico: con funzioni (sol. 1)

```c
#include <stdio.h>
#include <stdlib.h>
int * allocate_array(int size);

int main () {
    int i,n, *ptr;
    printf("\nInserisce dimensione array: ");
    scanf("%d",&n);
    ptr=allocate_array(n);
    printf("ptr: %p\n",ptr);
    for(i=0;i<n;i++) printf("%d ",*(ptr+i));
    free(ptr);
}
int * allocate_array(int size) {
    int i,*p;
    p=(int*) calloc(size, sizeof(int));
    printf("p: %p\n",p);
    for(i=0;i<size;i++)
        scanf("%d",p+i);
    return p;
}
```

Array dinamico: con funzioni (sol. 2)

```c
#include <stdio.h>
#include <stdlib.h>
void allocate_array(int size, int **p);
int main () {
    int i,n, *ptr;
    printf("Inserisce dimensione array: ");
    scanf("%d",&n);
    printf("Indirizzo ptr: %p\n",&ptr);
    allocate_array(n,&ptr);
    printf("Valore di ptr in main: %p\n",ptr);
    for(i=0;i<n;i++) printf("%d ",*(ptr+i));
    free(ptr);
}
void allocate_array(int size, int **p) { //p=&ptr;
    int i;
    printf("Indirizzo ptr nella f: %p\n",p);
    *p=(int*) calloc(size, sizeof(int));
    //*p equivale al valore di ptr del main -> ptr = calloc...
    printf("Contenuto di ptr in f: %p \n",*p);
    //indirizzo puntato da ptr
    for(i=0;i<size;i++) scanf("%d",*p+i); //carico in array dinamico
}
```



* array allocato
con calloc

F

```
p=&ptr;
```

*p = calloc ⟺ ptr = calloc

Array dinamico: esempio di realloc()

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *p,*q, n=5, m,i;

    p=(int *) malloc(n * sizeof(int));
    for(i=0;i<n;i++) scanf("%d",p+i);
    printf("\n");
    for(i=0;i<n;i++) printf("%d ",*(p+i));
    m=10;
    q=realloc(p,m * sizeof(int));
    for(i=n;i<m;i++) scanf("%d",q+i);
    for(i=0;i<m;i++) printf("%d ",*(q+i));

    return 0;
}
```

Primo esempio, crea lista

Creare una lista di N interi e visualizzarla

Primo esempio, crea lista

```c
#include <stdio.h>
#include <stdlib.h>

struct lista {
    int num;
    struct lista *next;
};

struct lista *crealista();
void visualizza_lista(struct lista *);
void libera_lista(struct lista *);

int main() {
    struct lista *punt_lista;

    punt_lista=crealista();
    if(punt_lista!=NULL) visualizza_lista(punt_lista);
    if(punt_lista!=NULL) libera_lista(punt_lista);
}
```

```c
struct lista *crealista() {
  int n,i;
  struct lista *p,*paux;

  printf("Quanti elementi vuoi inserire? ");
  scanf("%d",&n);
  if(n==0) p=NULL;
  else {
    p=(struct lista *)malloc(sizeof(struct lista));
    printf("Inserisci valore: ");
    scanf("%d",&p->num);
    paux=p;
    for(i=2;i<=n;i++) {
      paux->next=(struct lista *)malloc(sizeof(struct lista));
      paux=paux->next;
      printf("Inserisci valore: ");
      scanf("%d",&paux->num);
    }
    paux->next=NULL;
  }
  return p;
}
```

```c
void visualizza_lista(struct lista *p) {
  while(p!=NULL) {
    printf("%d ",p->num);
    p=p->next;
  }
}
```

```c
void libera_lista(struct lista *p) {
    struct lista *paux;

    while(p!=NULL) {
        paux=p;
        p=p->next;
        free(paux);
    }
}
```

Esempio: calcola minimo

```c
#include <stdio.h>
#include <stdlib.h>

struct lista {
    int num;
    struct lista *next;
};

struct lista *crealista();
void visualizza_lista(struct lista *);
int minimo(struct lista *);

int main() {
    struct lista *punt_lista;
    int min;

    punt_lista=crealista();
    if(punt_lista!=NULL) visualizza_lista(punt_lista);
    if(punt_lista!=NULL) printf("\nIl min vale: %d\n",
    min=minimo(punt_lista));
}
```

```c
int minimo(struct lista *p) {
  int min=p->num;

  while(p!=NULL) {
    if(p->num<min) min=p->num;
    p=p->next;
  }
  return min;
}
```

Crea lista, in testa

```
struct lista *crea_intesta() {
  int n,i;
  struct lista *p,*paux;

  printf("Quanti elementi vuoi inserire? ");
  scanf("%d",&n);
  if(n==0) p=NULL;
  else {
    p=(struct lista *)malloc(sizeof(struct lista));
    printf("Inserisci valore: ");
    scanf("%d",&p->num);
    p->next=NULL;
    for(i=2;i<=n;i++) {
      paux=(struct lista *)malloc(sizeof(struct lista));
      printf("Inserisci valore: ");
      scanf("%d",&paux->num);
      paux->next=p;
      p=paux;
    }
  }
  return p;
}
```

Inserisci elemento in una lista

La funzione deve **aggiungere** un elemento alla lista ad ogni chiamata.
L'inserimento può essere eseguito:

- in coda
- in testa

Inserisci elemento in coda

```c
#include <stdio.h>
#include <stdlib.h>

struct lista {
    int num;
    struct lista *next;
};

struct lista *ins_coda(struct lista *);
void visualizza_lista(struct lista *);
```

```c
int main() {
    struct lista *punt_lista=NULL;
    int sel=-1;

    while(sel!=0) {
      printf("\n1- Inserisci elemento\n");
      printf("\n2- Visualizza elementi\n");
      printf("\n0- Esci\n");
      scanf("%d",&sel);
      switch (sel) {
        case 1:
          punt_lista=ins_coda(punt_lista);
          break;
        case 2:
          if(punt_lista!=NULL) {
            printf("\nLista elementi: ");
            visualizza_lista(punt_lista);
          }
          else printf("\nLista vuota\n");
          break;
        case 0:
          printf("\nProgramma terminato\n");
          break;
        default:
          printf("\nScelta non valida\n");
```

```c
struct lista *ins_coda(struct lista *p) {
  struct lista *p1,*paux;

  p1=(struct lista *)malloc(sizeof(struct lista));
  printf("Inserisci valore: ");
  scanf("%d",&p1->num);
  if(p==NULL) {
    p=p1;
    p->next=NULL;
  }
  else {
    p1->next=NULL;
    paux=p;
    while(paux->next!=NULL) paux=paux->next;
    paux->next=p1;
  }
  return p;
}
```

Inserisci elemento in testa

```
struct lista *ins_testa(struct lista *p) {
  struct lista *p1;

  p1=(struct lista *)malloc(sizeof(struct lista));
  printf("Inserisci valore: ");
  scanf("%d",&p1->num);
  if(p==NULL) {
    p=p1;
    p->next=NULL;
  }
  else {
    p1->next=p;
    p=p1;
  }
  return p;
}
```

Elimina elemento da una lista

```
struct lista *elimina(struct lista *p) {
    struct lista *paux,*p2;
    int el;
    int trovato=0;

    printf("Inserisci elemento da eliminare: \n");
    scanf("%d",&el);
    if(p!=NULL) {
        if(p->num == el) {
            p2=p;
            p=p->next;
            free(p2);
            return p;
        }
```

Elimina elemento da una lista

```
        else {
            paux=p;
            while(paux->next!=NULL && trovato!=1) {
              if(paux->next->num != el) paux=paux->next;
              else {
                  trovato=1;
                  p2=paux->next;
                  paux->next=paux->next->next;
                  free(p2);
                  return p;
                }
            }
        }
        if(!trovato) printf("Elemento non presente!\n");
    }
    else printf("La lista e' vuota!\n");
    return p;
}
```