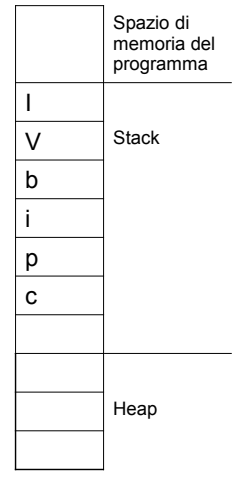
Allocazione dinamica della memoria

Allocazione statica della variabili

Dato il seguente programma C

```
#include <stdio.h>
void f1(int i,double p);
void main(void)
{ int a,V[10];
  double b;
 …
 fa(a,b);
 …
}
Void f1(int i, double p)
{ int c;
..
}
```

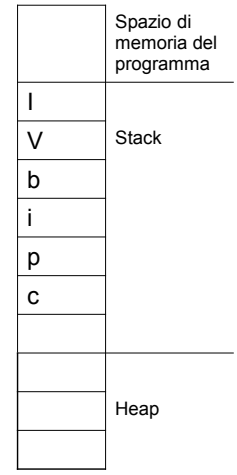


le variabili dichiarate nella sezione dichiarativa del main vengono allocate staticamente nel momento in cui il programma viene eseguito.

- L'area di memoria in cui vengono allocate viene chiamata ***stack***.
- Nella stessa area vengono allocati i parametri della funzione f1 al momento della sua attivazione e le variabili locali di f2

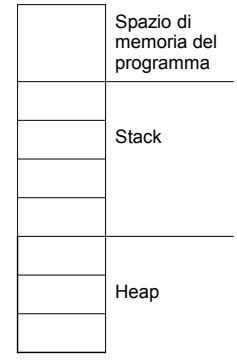
Allocazione statica della variabili

- Al termine dell'esecuzione delle funzioni le variabili allocate nello stack vengono liberate in ordine inverso rispetto a quello di allocazione.
- Durante l'esecuzione del programma non è possibile variare il numero e la dimensione delle variabili, la cui dimensione (in termini di area di memoria allocata) è definita già in fase di compilazione, prima dell'esecuzione dei programmi.
- Es. Il vettore int V[10], allocato staticamente, manterrà sempre la stessa dimensione. Non è possibile modificarne il numero di elementi.
- In questi casi è necessario sovradimensionare la dimensione del vettore per porsi nelle condizioni peggiori, con un conseguente spreco di memoria.
- Per accedere all'area di memoria delle variabili allocate staticamente è sufficiente utilizzare il nome della stessa variabile (Es. a=10, V[i]=20).



Allocazione dinamica della memoria

- Un modo alternativo per ottenere l'allocazione di aree di memoria durante l'esecuzione di un programma è l'allocazione dinamica.
- In questo modo è possibile richiedere l'allocazione di un'area di memoria la cui dimensione è nota solo in fase di esecuzione.
- Ad esempio volendo copiare il contenuto di un file in un vettore di interi, poiché il numero di elementi è noto solo durante l'esecuzione del programma (accedendo al file) il vettore non può essere allocato staticamente, ma deve essere allocato dinamicamente.
- La funzione che permette l'allocazione dinamica della memoria è la `malloc();`
- Quella che libera la memoria precedentemente allocata dinamicamente è: `free();`
- Entrambe le funzioni sono definite nel file di header ***stdlib.h***
- La memoria allocata dinamicamente è ottenuta dalla regione detta ***heap***.



malloc()

- La funzione malloc() permette l'allocazione di un'area di memoria in precedenza libera.
- Il prototipo della funzione malloc è il seguente:

```
void * malloc(size_t  numero_di_byte);
```

  - *numero_di_byte* indica il numero di byte che si intende allocare.
- Il valore restituito dalla malloc è il puntatore al primo byte dell'area allocata.
- La malloc restituisce un puntatore di tipo void; questo significa che può essere assegnato a ogni tipo di puntatore.
- Se non è possibile allocare l'area di memoria richiesta, la malloc restituisce un puntatore NULL.

Esempio malloc()

```
main()
{ int *pa,*V;
  double *pd;

  pa=(int *) malloc(sizeof(int); /* Allocazione di
  un                          intero */
  V=(int *) malloc(sizeof(int)*10); /*Allocazione di
  un                          vettore di interi*/
  pd=(double *) malloc(sizeof(double));
  /*Allocazione di un double*/
…
}
```

I puntatori `pa`, `V` e `pd` sono allocati staticamente.

malloc()

- A differenza delle variabili allocate staticamente, l'accesso alle aree di memoria allocate dinamicamente può avvenire solo mediante il puntatore.

```
Es.
main()
{ int *pa;
  double *pd;

  pa=(int *) malloc(sizeof(int); /* Allocazione di un
   intero */
  pd=(double) malloc(sizeof(double)); /*Allocazione di
   un double*/
  *pa=10;
  scanf("%lf",pd);
  printf("%d %lf",*pa,*pd);
}
```

- Nel caso di vettori allocati dinamicamente, l'accesso agli elementi avviene con le stesse modalità dei vettori allocati staticamente (poiché il nome del vettore è il puntatore al primo elemento del vettore).

free()

- Il prototipo della free è il seguente:

```
void free( void *p);
```

- Questa funzione ha il compito di liberare l'area di memoria in precedenza allocata mediante la malloc() il cui indirizzo iniziale è stato assegnato al puntatore p.

Allocazione di un vettore di 10 interi

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

main()
{ int *V;
  int i;
 /*Allocazione dinamica del vettore il cui indirizzo iniziale
   viene assegnato al puntatore V*/
 V=(int *) malloc(10*sizeof(int));

 if(V) /* Se l'allocazione ha avuto successo*/
 { for(i=0;i<10;i++)
    {   printf("Numero: ");
        fscanf("%d",&V[i]);
    }
   for(i=0; i<10 ;i++)
      printf("Matricola: %d ",V[i].matricola);

 /* Viene liberata l'area di memoria in precedenza allocata il cui
   indirizzo iniziale è conservato nel puntatore V */
 free(V);
 }
 else printf("Memoria esaurita");
}
```

Lettura e visualizzazione di un vettore di 10 struct allocato dinamicamente

```
#include <stdio.h>
#include <stdlib.h>
struct studente {
   long matricola;
   int esami;
};
main()
{ struct studente *V;
   int i;
 V=(struct studente *) malloc(10*sizeof(struct studente));
 if(V==NULL)
/* L'esecuzione del programma viene interrotta restituendo un
   valore (1) che indica la presenza di un errore */
 exit(1);
 for(i=0;i<10;i++)
    {printf("Matricola: ");  scanf("%ld",&V[i].matricola);
      printf("Num.esami: "); scanf("%d",&V[i].esami);
    }
 for(i=0; i<10 ;i++)
    {   printf("Matricola: %d ",V[i].matricola);
        printf("Num.Esami: %d \n",V[i].esami);
     }
 free(V);
}
```

Copia da file ad un vettore allocato dinamicamente (1)

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

void visualizza_vettore(int *V, int n);
long dim_file(FILE *pf);

main()
{int *V;
 FILE *pf;
 long numero;

 /* Inizializzazione vettore */
 pf=fopen("dati.dat","r");
 if(pf) { numero=dim_file(pf)/sizeof(int);
          V=( int *) malloc(numero*sizeof(int));
         if(V==NULL) {printf("Memoria esaurita");
                      exit(1);
                     }
         fread(V,sizeof(int),numero,pf);
         visualizza_vettore(V,numero);
         fclose(pf);
        }
 else printf("Errore lettura da file");

 free(V);
}
```

Copia da file ad un vettore allocato dinamicamente (2)

```
void visualizza_vettore(int *V, int n)
{
  int i;
  for(i=0;i<n;i++)
   printf("%d\n",V[i]);
}

long dim_file(FILE *pf)
{ long corr,ultimo;

  corr=ftell(pf);
  fseek(pf,0,SEEK_END);
  ultimo=ftell(pf);
  fseek(pf,corr,SEEK_SET);
 return ultimo;
}
```

realloc()

- Mediante la malloc() è possibile allocare un'area di memoria, ma non è possibile modificarne la dimensione.
- Per ottenere il cambiamento della dimensione di un'area precedentemente allocata (il cui puntatore è punt_vecchio) è necessario ricorrere ad un'altra funzione, la realloc() il cui prototipo è il seguente:

```
void * realloc(void *punt_vecchio, size_t nuova_dim);
```

  - Punt_vecchio è il puntatore all'area di memoria la cui dimensione deve essere modificata;
  - Nuova_dim è la nuova dimensione che dovrà avere l'area di memoria prededentemente allocata. Il suo valore può essere maggiore o minore.

Es.

```
int *V,*Vaux,i;

V=(int *) malloc(sizeof(int)*10);
if(V)
  for(i=0;i<10;i++)
      V[i]=i;
Vaux=(int *)realloc(V,sizeof(int)*20);
```

- La realloc() potrebbe dover spostare il blocco di memoria originale per poterne aumentare la dimensione, se subito dopo il blocco originale non è presente spazio sufficiente per aumentarne la dimensione. In questo caso il valore restituito sarà diverso da *punt_vecchio*. Inoltre, il contenuto del vecchio blocco viene copiato nel nuovo blocco e quindi non vengono perse informazioni.

realloc()

- Se c'è spazio libero sufficiente dopo il blocco originale per aumentarne la dimensione o nel caso di diminuzione della dimensione il valore restituito dalla reallo() coincide con quello di *punt_vecchio*.
- Se non è possibile incrementare la dimensione di un'area precedentemente allocata, la realloc() restituisce un puntatore NULL. Pertanto è opportuno assegnare il valore restituito dalla realloc() ad un puntatore diverso rispetto a quello del blocco originario.
- Scrivendo:

```
V=realloc(V,sizeof(int)*nuova_dim);
```

nel caso in cui la realloc() restituisse NULL, non potremmo più accedere all'area di memoria originariamente allocata e comunque l'area di memoria resterebbe allocata con un conseguente spreco di memoria.

- La soluzione è quella che prevede di assegnare il valore restituito sempre a un puntatore ausiliario e assegnare al puntatore relativo al blocco originario il valore restituito dalla realloc() solo se non NULL
- Es.

```
Vaux=realloc(V,sizeof(int)*20);
if(Vaux) V=Vaux;
```

Copia da file a vettore allocato dinamicamente e successiva incremento della dimensione del vettore (1)

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

long num_file(FILE *pf);
void visualizza(int *V,long num);
void aggiungi(int *V,long num,int nag);
void salva(int *V,long num,int nag, FILE *pf);

main()
{ FILE *pf;
  int *V=NULL,*Vaux=NULL, naggiunti;
  char nomefile[30];
  long numero; /*Numero di elementi letti da file*/

  printf("Inserire il nome del file");
  scanf("%s",nomefile);
  pf=fopen(nomefile,"r");
  if(pf) { numero=num_file(pf);
           /* Allocazione dinamica del vettore*/
          V=malloc(numero*sizeof(int));
          if(V) {fread(V,sizeof(int),numero,pf);
                  visualizza(V,numero);
                }
          else  { printf("Memoria esaurita\n");
                   exit(1);
                }
         fclose(pf);
        }
```

Copia da file a vettore allocato dinamicamente e successiva incremento della dimensione del vettore (2)

```
  else   /* Poiché il file non esiste il vettore non viene allocato */
         {numero=0;
          printf("File vuoto");
          }
  printf("Quanti elementi vuoi inserire ?\n");
  scanf("%d",&naggiunti);
  /*Incremento della dimensione del vettore mediante realloc */
  Vaux=realloc(V,(numero+naggiunti)*sizeof(int));
  if(Vaux)
       {V=Vaux;
        aggiungi(V,numero,naggiunti);
        visualizza(V,numero+naggiunti);
       }
  else {printf("Memoria esaurita\n");
        exit(1);
       }
  /* Salvataggio su file dei soli elementi aggiunti*/
  pf=fopen(nomefile,"a+");
  if(pf)
   { salva(V,numero,naggiunti,pf);
     fclose(pf);
   }
  else printf("Errore in scrittura");

  free(V);
}
```

Allocazione dinamica e funzioni

Con il seguente programma si vorrebbe allocare dinamicamente un vettore all'interno di una funzione (leggi_vettore)

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

void leggi_vettore(int *Vf, int *num);

main ()
{  int *V,i,numero=0;

  V=NULL;
  leggi_vettore(V,&numero);
   for(i=0;i<numero;i++)
  printf("%d ",V[i]);
}

void leggi_vettore(int *Vf, int *num)
{ int i;

  printf("Quanti numeri vuoi leggere ?");
  scanf("%d",num);
  Vf=malloc((*num)*sizeof(int));
  for(i=0;i<*num;i++)
{ printf("Nuovo numero ");
      scanf("%d",&Vf[i]);
    }
}
```

Allocazione dinamica e funzioni

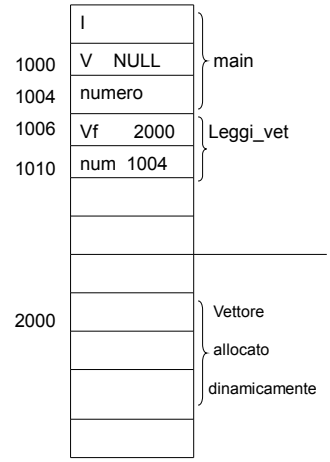
Tuttavia il precedente programma si comporta in modo diverso rispetto a quanto si vorrebbe ottenere.

Come si può vedere nella rappresentazione della memoria, quando viene invocata la funzione leggi_vettore, vengono allocate le locazioni per il parametro formale Vf, in cui viene copiato il valore del parametro attuale V, e per il parametro num, in cui viene copiato l'indirizzo del parametro attuale numero.

Durante l'esecuzione della funzione viene allocato un vettore il cui indirizzo iniziale viene assegnato al parametro Vf e vengono letti i valori degli elementi del vettore.

Al termine della funzione l'area associata a Vf viene liberata e il valore iniziale del vettore allocato viene perso.

La ragione è che la funzione non modifica il valore del parametro attuale che è stato passato per valore.



Passaggio per indirizzo del puntatore

E' necessario passare il puntatore V per indirizzo.

Il prototipo della funzione leggi_vettore è il seguente:

```
void leggi_vettore(int **Vf, int *num);
```

In questo modo, all'attivazione della funzione leggi_vettore, nel parametro formale Vf viene copiato l'indirizzo del parametro attuale e ogni modifica attuata alla locazione *Vf si ripercuote in una modifica del parametro attuale V.

Versione corretta del programma con passaggio per indirizzo del puntatore

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

void leggi_vettore(int **Vf, int *num);

main ()
{  int *V,i,numero=0;

  V=NULL;
  leggi_vettore(&V,&numero);
   for(i=0;i<numero;i++)
     printf("%d ",V[i]);
}

void leggi_vettore(int **Vf, int *num)
{ int i;

  printf("Quanti numeri vuoi leggere ?");
  scanf("%d",num);
  *Vf=malloc((*num)*sizeof(int));
  for(i=0;i<*num;i++)
   { printf("Nuovo numero ");
      scanf("%d",&(*Vf)[i]);
    }
}
```

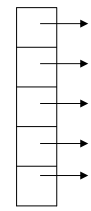
Allocazione dinamica di una tabella con un numero fisso di righe e variabile di colonne

Per poter allocare una tabella di cui è noto a priori il numero delle righe, ma non è noto il numero di colonne è necessario dichiarare un vettore i cui elementi sono dei puntatori agli elementi della riga.

Es.

Se si vuole allocare una tabella di interi con 5 righe, di cui non si conosce a priori il numero di colonne, è necessario fare la seguente dichiarazione:

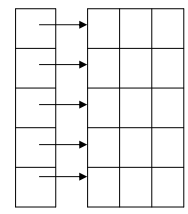
```
int *T[5];
```



Il vettore T viene allocato staticamente quando il programma viene mandato in esecuzione.

Il passo successivo è l'allocazione dinamica di ciascuna riga mediante la malloc().

```
for(i=0;i<5;i++)
  T[i]=malloc(sizeof(int)* dim_riga);
```



Allocazione dinamica di una tabella con un numero fisso di righe e variabile di colonne

- Al termine dell'esecuzione, per liberare lo spazio allocato precedentemente per la tabella è necessario liberare lo spazio allocato per ciascuna riga.

  Es.

```
for(i=0;i<5;i++)
   free(T[i]);
```

Allocare una tabella con 5 righe e un numero di colonne letto da tastiera

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

main ()
{
 int *T[5];
 int riga,colonna,dim_riga;

 printf("Inserire il numero di colonne\n");
 scanf("%d",&dim_riga);
 /* Allocazione delle righe */
 for(riga=0;riga<5;riga++)
    T[riga]=malloc(sizeof(int)*dim_riga);
 /* Assegnamento degli elementi */
 for(riga=0;riga<5;riga++)
  for(colonna=0;colonna<dim_riga;colonna++)
      T[riga][colonna]=riga*colonna;

 for(riga=0;riga<5;riga++)
   for(colonna=0;colonna<dim_riga;colonna++)
     printf("%d\t",T[riga][colonna]);
 /* Deallocazione della memoria */
 for(riga=0;riga<5;riga++)
  free(T[riga]);
}
```

Allocare dinamicamente una matrice con 11 righe, pari al numero massimo di esami, e un numero variabile di colonne (1)

```
/* Questo programma legge da file coppie di variabili (matricola,
   esami) e inserisce in una tabella di 11 righe (pari al numero
   massimo di esami) il numero di matricola nella riga di indice pari
   al valore della variabile esami.
   Per effettuare l'inserimento in una riga è necessario incrementarne
   la dimensione mediante la funzione realloc.
   Per tenere traccia della dimensione di ogni riga si fa uso di un
   vettore D i cui elementi D[i] indicano la dimensione della riga
   S[i]. */

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

main ()
{
 FILE *pf;
 long *S[11],*Vaux,matricola;
 int riga,colonna,D[11],esami;

/* Inizializzazione dei vettore */
 for(riga=0;riga<11;riga++)
    { S[riga]=NULL;
      D[riga]=0;
    }
```

Allocare dinamicamente una matrice con 11 righe, pari al numero massimo di esami, e un numero variabile di colonne (2)

```
pf=fopen("studenti.txt","r");
 if(pf)
  { while(!feof(pf))
    { /*Lettura da file */
      fscanf(pf,"Matricola: %ld\n",&matricola);
      fscanf(pf,"Esami: %d\n",&esami);
      /* Incremento della dimensione della riga S[esami] */
      Vaux=realloc(S[esami],(D[esami]+1)*sizeof(long));
      if(Vaux) /* L'allocazione ha avuto successo */
      { S[esami]=Vaux;
            S[esami][D[esami]]=matricola;
        D[esami]++;
      }
      else {printf("Memoria esaurita\n");
            break;
            }
    }
   fclose(pf);
  }
```

Allocare dinamicamente una matrice con 11 righe, pari al numero massimo di esami, e un numero variabile di colonne (3)

```
for(riga=0;riga<11;riga++)
    { printf("\nElenco studenti con %d esami\n",riga);
       for(colonna=0;colonna<D[riga];colonna++)
    printf("%ld\t",S[riga][colonna]);
     }

 for(riga=0;riga<11;riga++)
  free(S[riga]);

}
```

Allocare dinamicamente una matrice con 11 righe (massimo numero di esami) e un numero variabile di colonne (V2) (1)

```
/* Questo programma è la versione con le funzioni del programma
   precedente*/

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

void leggi_da_file(FILE *pf, int **Vf,int *Df);
void visualizza(int **Vf,int *Df);

void main (void)
{ FILE *pf;
  long *S[11];
  int riga,colonna,D[11];

for(riga=0;riga<11;riga++)
    { S[riga]=NULL;
       D[riga]=0;
    }
  pf=fopen("studenti.txt","r");
  if(pf)
  { leggi_da_file(pf,S,D);
    fclose(pf);
  }
  else printf("Errore in lettura");
  visualizza(S,D);
  for(riga=0;riga<11;riga++)
    free(S[riga]);
}
```

Allocare dinamicamente una matrice con 11 righe (massimo numero di esami) e un numero variabile di colonne (V2) (2)

```
void leggi_da_file(FILE *pf, int **Vf,int *Df)
{ long *Vaux,matricola;
  int esami;

   while(!feof(pf))
    { fscanf(pf,"Matricola: %ld\n",&matricola);
      fscanf(pf,"Esami: %d\n",&esami);
      /* Incremento della dimensione della riga S[esami] */
      Vaux=realloc(Vf[esami],(Df[esami]+1)*sizeof(long));
      if(Vaux) /* L'allocazione ha avuto successo */
      { Vf[esami]=Vaux;
             Vf[esami][Df[esami]]=matricola;
        Df[esami]++;
      }
      else {printf("Memoria esaurita\n"); break;
            }
    }
}

void visualizza(int **Vf,int *Df)
{ int riga,colonna;

  for(riga=0;riga<11;riga++)
    { printf("\nElenco studenti con %d esami\n",riga);
       for(colonna=0;colonna<Df[riga];colonna++)
    printf("%ld\t",Vf[riga][colonna]);
     }
}
```

Leggere da tastiera corso di laurea, matricola ed esami e inserire in una tabella con un numero di righe pari al numero di corsi di laurea (1)

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

struct studente {
   long matricola;
   int esami;    };

void inserisci(struct studente **VC,int *DC);
void visualizza(struct studente **VC,int *DC);

main (void)
{
 FILE *pf;
 struct studente *CL[2];
 int corso,D[2];

 for(corso=0;corso<2;corso++)
    { CL[corso]=NULL;
       D[corso]=0;
    }
  inserisci(CL,D);
  visualizza(CL,D);

 for(corso=0;corso<2;corso++)
  free(CL[corso]);

}
```

Leggere da tastiera corso di laurea, matricola ed esami e inserire in una tabella con un numero di righe pari al numero di corsi di laurea (2)

```
void inserisci(struct studente *VC[],int *D)
{  struct studente *Vaux,A;
   int esami;

/* Vengono inseriri nuovi dati fino a quando la matricola è non nulla */
  do
    { printf("Matricola: ");
      scanf("%ld",&A.matricola);
      if(A.matricola)
       { printf("Esami: ");
         scanf("%d",&A.esami);
     printf("Corso di Laurea (0=Civile, 1=Industriale)\n");
     scanf("%d",&esami);
     Vaux=realloc(VC[esami],(D[esami]+1)*sizeof(struct studente));
           if(Vaux)
        { VC[esami]=Vaux;
           VC[esami][D[esami]]=A;
      D[esami]++;
       }
          else {printf("Memoria esaurita\n");
                break;
                }
         }
       else printf("Fine inserimento\n");
   }
  while(A.matricola);
}
```

Leggere da tastiera corso di laurea, matricola ed esami e inserire in una tabella con un numero di righe pari al numero di corsi di laurea (3)

```
void visualizza(struct studente *VC[],int *DC)
{
 int r,c;

 for(r=0;r<2;r++)
    { if(r==0)
         printf("\nElenco studenti di Ingegneria Civile\n");
      else printf("\nElenco studenti di Ingegneria Industriale\n");
      for(c=0;c<DC[r];c++)
        printf(" Matr: %ld Esami: %d\n",VC[r][c].matricola,VC[r][c].esami);
     }
}
```