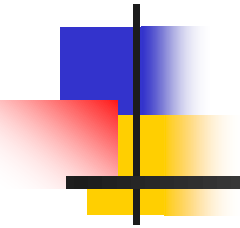


# Puntatori e Heap in C

Prof.Ing.S.Cavalieri

# Tipi Puntatori

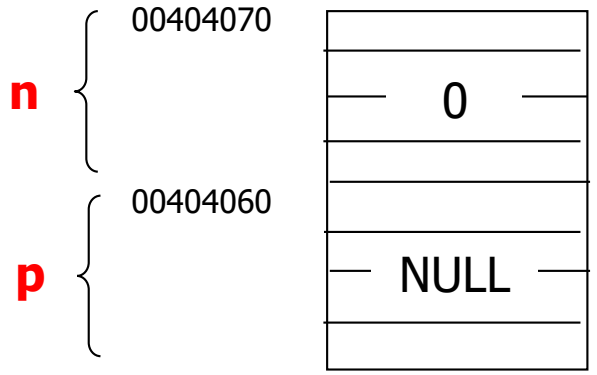
```
#include <stdio.h>

int n;
int * p;

int main(void) {
    n = 15;
    p = &n;

    printf("\nIl Numero n e' uguale a :  %d  ",n);
    *p = 25;
    printf("\nIl Numero n e' uguale a :  %d   ", *p);
}
```

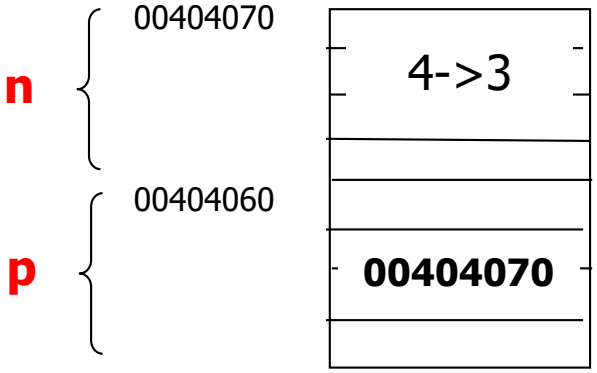
NULL

int n, *p;

# Tipi Puntatori

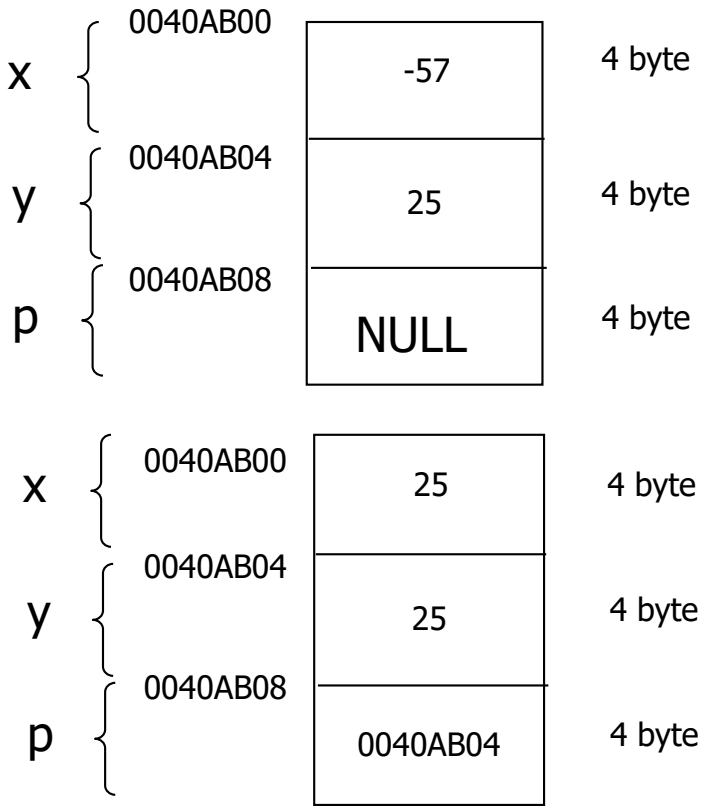
```
#include<stdio.h>

int n, *p;

int main(void) {
   n=4;
   printf("\nContenuto di n = %d ",n);
   printf("\nIndirizzo di n = %p ",&n);
   p=&n;
   printf("\nContenuto di p = %p ",p);
   printf("\nIndirizzo di p = %p ",&p);
   *p=3;
   printf("\nContenuto di n = %d ",n);
}
```

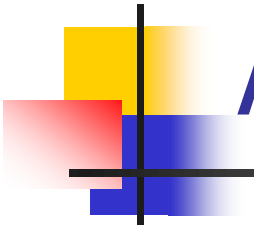


# Tipi Puntatori

```
int x=-57, y=25;
int *p;
int main(void)
{
   p=&y;
   x=*p;
}
```



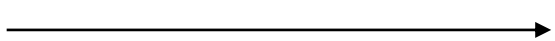


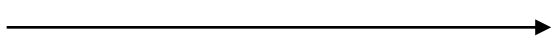
# Aritmetica dei Puntatori

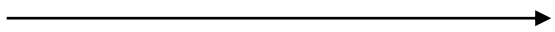
- Aritmetica dei Puntatori:
  - Incremento/Decremento di una variabile puntatore è legato alla dimensione della variabile puntata
  - p++, p si incrementa di un numero di byte pari alla dimensione della variabile a cui punta
  - p--, p si decrementa di un numero di byte pari alla dimensione della variabile a cui punta

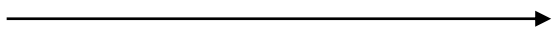
# Aritmetica dei Puntatori

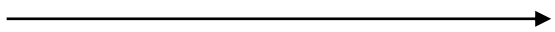
```
#include<stdio.h>
int x=-57, y=25, *p;

int main(void)
{
   printf("\nIndirizzo di x=%p ",&x);
   printf("\nIndirizzo di y=%p ",&y);
   printf("\nIndirizzo di p=%p ",&p);
   p=&x;
   printf("\nContenuto di p=%p ",p);
   p++;
   printf("\nContenuto di p=%p ",p);
}
```

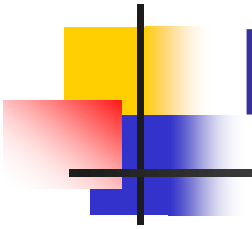




# Puntatori e Vettore

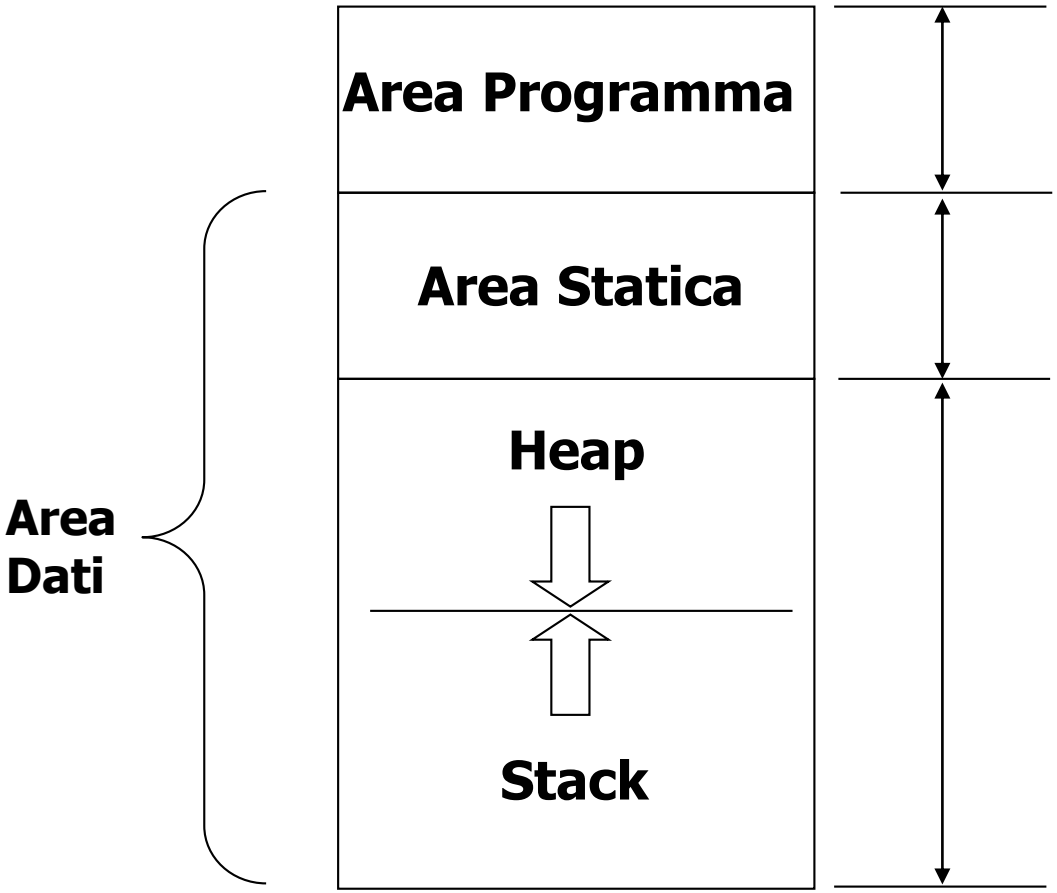
- Il nome della variabile vettore è l'indirizzo del primo elemento del vettore:
  - int v[10];
  - v  corrisponde a  &v[0]
  - v+i corrisponde a &v[i] (aritmetica dei puntatori!)
  - v[i] corrisponde a *(v+i)
  - scanf("%d",&v[i]) corrisponde a scanf("%d",v+i);
  - printf("%d",v[i]) corrisponde a printf("%d",*(v+i));

# Puntatori e Vettore

```
#include<stdio.h>
#define N 10

int v[N], i;

int main(void)
{
   for (i=0; i<N; i++)
       scanf("%d",&v[i]);

   for (i=0; i<N; i++)
       scanf("%d”,v+i);

}
```

Equivalente!

# Allocazione di Memoria in C



# Allocazione di Memoria in C

*#include<stdlib.h>*

*variabile puntatore= (cast tipo puntatore) malloc (dimensione in byte);*

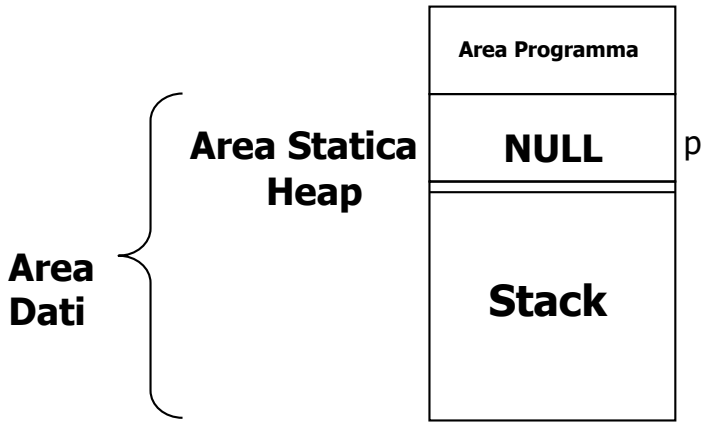
*free (variabile puntatore);*

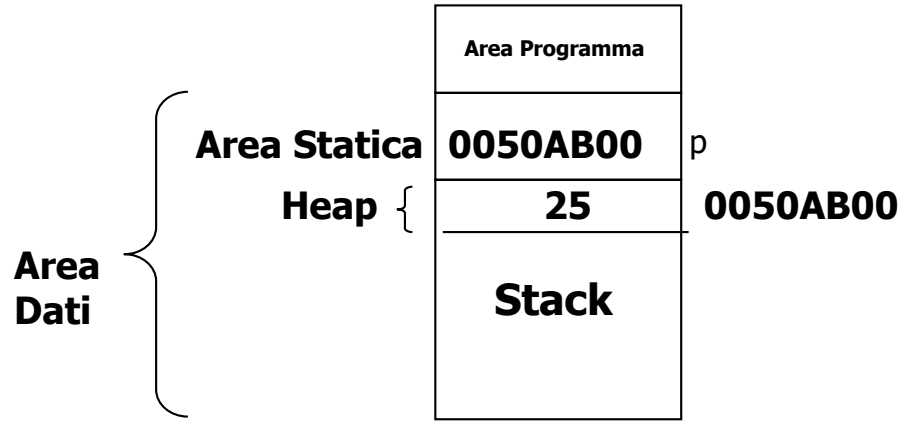
**Da non dimenticare !!!!!!**

Si noti che la funzione *free()*, de-alloca l'area puntata dal puntatore, ma non annulla il puntatore, ossia esso non viene posto a NULL, ma viene lasciato al valore attuale.

# Allocazione di Memoria in C

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

int *p;
int main(void)
{
    p = (int *) malloc(sizeof(int));
   *p = 25;

   free (p);
}
```

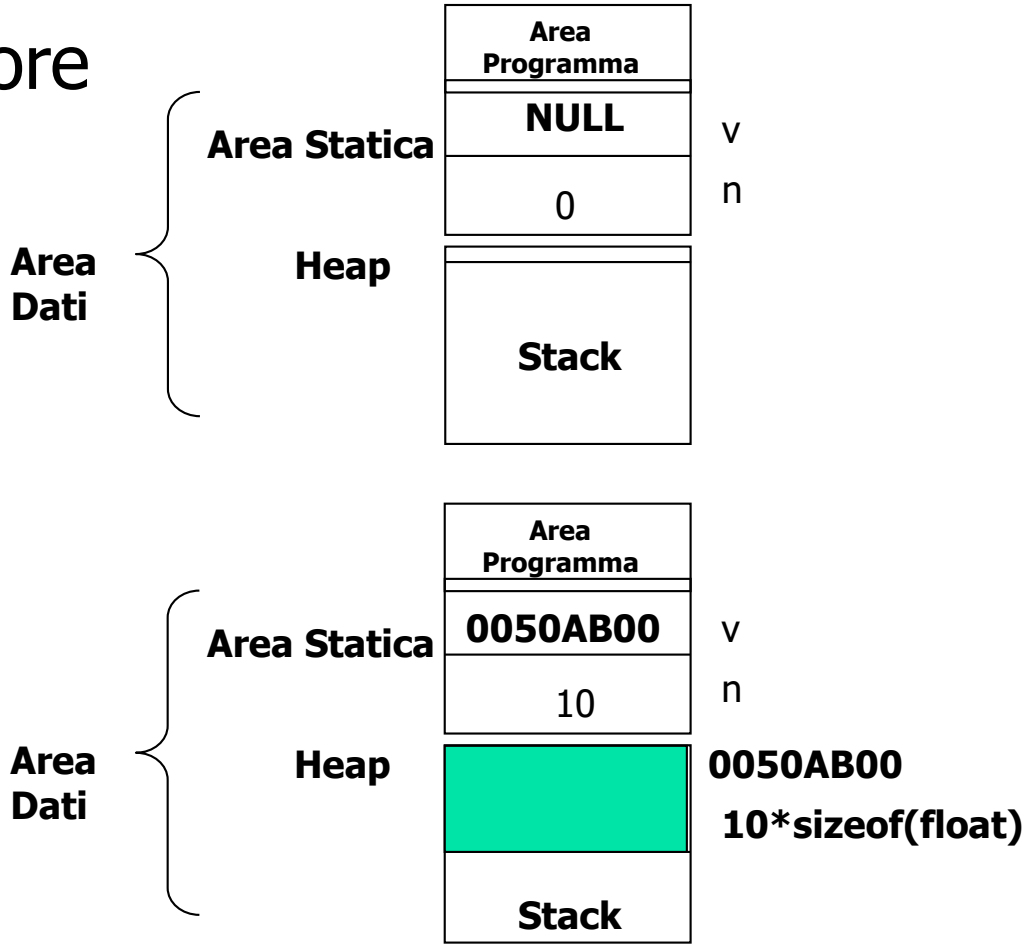


# Allocazione di Vettore in C

- Allocazione di un vettore

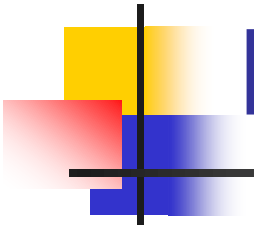
```
#include<stdio.h>
#include<stdlib.h>
float *v;
unsigned int n;

int main(void)
{
  printf ("Inserisci il valore di n  ");
  scanf("%u",&n);

  v=(float *)malloc(n*sizeof(float));
  free(v);
}
```



# Esercizio

- Allocazione di un vettore
- Riempimento Vettore
- Stampa Vettore
- De-Allocazione Vettore

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

float *v;
unsigned int i, n;

int main(void)
{
    do {
        printf("\nInserisci il valore di n > 1  ");
        scanf("%u", &n);
    } while (n<2);

    v = (float *)malloc(n*sizeof(float));

    for (i = 0; i<n; i++) {
        printf("Inserisci l'elemento del vettore di indice %u ",i);
        scanf("%f", v + i); /*oppure scanf("%f",&v[i]);*/
    }
    for (i = 0; i<n; i++)
        printf("\nv[%u] = %f  ", i, v[i]);

    free(v);
    return 0;
}
```



# Ri-Allocazione di Vettore in C

- realloc(*punt_vecchio*, *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.
- La realloc() restituisce un puntatore:
  - Uguale a punt_vechio se il vettore può essere "allungato"
  - Diverso da punt_vechio, se il vettore viene spostato in un'altra zona di memoria
  - Il valore di puntatore restituito è NULL se non riesce a spostare il blocco, e il vecchio blocco non viene modificato

# Ri-Allocazione di Vettore in C

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

float * vettore, *tmp;
unsigned int i, dim1 = 4, dim2 = 8;

int main(void)
{
    vettore = (float *)malloc(dim1*sizeof(float));
    for (i = 0; i<dim1; i++) {
            printf("\nInserisci Elemento Indice %u ", i);
            scanf("%f", vettore+i);
    }
    printf("\nPuntatore Vettore = %p", vettore);

    tmp = (float *)realloc(vettore, dim2*sizeof(float));
    if (tmp) {
          vettore = tmp;
          printf("\nPuntatore Vettore Riallocato = %p", vettore);
          for (i = 0; i<dim2; i++)
              printf("\nElemento di indice %u = %f", i, vettore[i]);
    } else printf("\nIl vettore non puo' essere riallocato ");
}
```

# Tipi Puntatori e Struct

```
struct elemento {
   char cognome[15];
   unsigned short eta;
} x, *p;
```

- p=&x
- *p  corrisponde a x, all'intero struct
- (*p).cognome corrisponde a x.cognome
- (*p).eta corrisponde a x.eta
- la parentesi è necessaria a causa della priorità
- **l'operatore -> può essere usato in alternativa**
- (*p).cognome equivale a **p->cognome**
- (*p).eta equivale a **p->eta**

# Tipi Puntatori e Struct

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

#define S 30
#define T 15

typedef struct {
    char cognome[S], nome[S], telefono[T];
    unsigned short eta;
} persona;

persona * p;
```

```
int main(void) {

        printf("\nIndirizzo p = %p ", p);
        p = (persona *)malloc(sizeof(persona));
        printf("\nIndirizzo p = %p ", p);

        printf("\nIserisci il Cognome ");
        fgets(p->cognome,S, stdin);

        printf("\nIserisci il Nome ");
        fgets(p->nome, S, stdin);

        printf("\nIserisci il Telefono ");
        fgets(p->telefono, T, stdin);

        printf("\nIserisci l'eta' ");
        scanf("%hu", &p->eta);

        printf("\nCognome = %s ", p->cognome);
        printf("\nNome = %s ", p->nome);
        printf("\nTelefono = %s ", p->telefono);
        printf("\nEta = %u ", p->eta);
        free(p);
}
```

# Struct, Vettori e Puntatori

```
typedef struct {
   char cognome[30], nome[30], telefono[15];
} persona;

persona vettore[N];

persona * vettore;
vettore=(persona *) malloc(n*sizeof(persona));
```

**Vettore Statico**

**Vettore Dinamico**

# Struct, Vettori e Puntatori

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

#define FFLUSH while(getchar()!='\n')

#define S 30
#define T 15

typedef struct {
    char cognome[S], nome[S], telefono[T];
    unsigned short eta;
} persona;

persona * v;
unsigned int i,n;
```

```
int main(void)
{
    do {
        printf("\nInserisci la dimensione ");
        scanf("%u", &n);
        FFLUSH;
    } while (n<2);

    v = (persona *)malloc(n*sizeof(persona));

    for (i = 0; i<n; i++) {
        printf("\nInserisci il Cognome ");
        fgets(v[i].cognome,S,stdin);
        printf("\nInserisci il Nome ");
        fgets(v[i].nome, S, stdin);
        printf("\nInserisci il Telefono ");
        fgets(v[i].telefono, T, stdin);
        printf("\nInserisci l'eta' ");
        scanf("%hu", &v[i].eta);
        FFLUSH;
    }

    for (i = 0; i<n; i++) {
        printf("\nCognome = %s ", v[i].cognome);
        printf("\nNome = %s ", v[i].nome);
        printf("\nTelefono = %s ", v[i].telefono);
        printf("\nEta' = %u ", v[i].eta);
    }

    free(v);
}
```