

Unità didattica: Funzioni del linguaggio C per la gestione dinamica della memoria [1-C]

Titolo: Funzioni del linguaggio C per l'allocazione dinamica della memoria

Argomenti trattati:

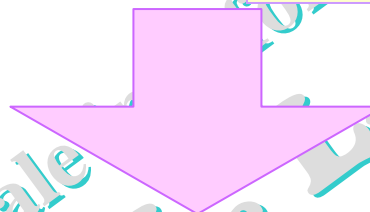
- ✓ Cos'è l'allocazione dinamica della memoria
- ✓ Funzioni C in `stdlib.h` per allocare e deallocare blocchi di memoria
- ✓ Differenza tra `malloc` e `calloc`
- ✓ Massimo size di un array C

Prerequisiti richiesti: programmazione C (array, puntatori)

**allocazione statica
della memoria**

=

allocazione di tutto lo spazio di memoria per i dati di un programma prima della sua esecuzione

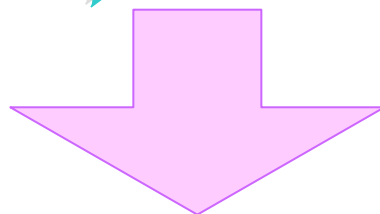


L'occupazione di memoria è fissa

**allocazione dinamica
della memoria**

=

possibilità di allocare/deallocare spazio di memoria in fase di esecuzione di un programma



L'occupazione può variare secondo necessità

Funzioni C

(#include <stdlib.h>)

<code>malloc(...)</code>	Alloca blocchi di memoria per un oggetto
<code>calloc(...)</code>	Alloca blocchi di memoria per un array di oggetti (elementi inizializzati a 0)
<code>realloc(...)</code>	Rialloca blocchi di memoria allocati prima con <code>malloc(...)</code> o <code>calloc(...)</code>
<code>free(...)</code>	Dealloca (libera) blocchi di memoria

Prototipo delle funzioni

```
void *malloc(size)
```

numero bytes
da allocare

Restituisce un *void pointer* (*puntatore generico*) al blocco di memoria allocato, oppure **NULL** se la memoria disponibile è insufficiente.

Usare il **type cast** sul valore restituito dalla funzione.

```
punt1=(char *)malloc(num_byte);  
if punt1 != NULL ...
```

Controllare sempre il valore restituito da `malloc`, anche se la quantità di memoria richiesta è piccola.

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

puntatore al blocco
da deallocare

Libera il blocco di memoria puntato da *p*.

```
free(punt1);
```

```
void *calloc(num, size)
```

Restituisce un ***void pointer*** ad un array (di dimensione ***num***, dove ciascuna componente ha ampiezza ***size***) oppure **NULL** se la memoria disponibile è insufficiente. Usare il **type cast** sul valore restituito dalla funzione. Le componenti dell'array sono inizializzate a 0.

```
b = (float *)calloc(n,sizeof(float));  
if b != NULL ...
```

Controllare sempre il valore restituito.

```
void *realloc(*p, size)
```

Restituisce un ***void pointer*** ad un nuovo blocco di memoria di ampiezza ***size*** e contenente il blocco puntato da ****p***, oppure **NULL** se la richiesta non può essere soddisfatta (in tal caso ****p*** rimane invariato). Se la nuova ampiezza è maggiore della vecchia, il nuovo spazio non è inizializzato. Usare il **type cast** sul valore restituito dalla funzione.

```
b = (float *)realloc(b,npn*sizeof(float));  
if b != NULL ...
```

Esempio 1

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

```
void main()
```

```
{char *punt1,*punt2; short j, num_byte=10;
```

```
punt1=(char *)malloc(num_byte);
```

alloca un blocco di byte e
non inizializza il contenuto

```
punt2=(char *)calloc(num_byte, sizeof(char));
```

```
if (punt1 != NULL)
```

alloca un blocco di byte ed
inizializza il contenuto

```
{puts("contenuto dei bytes allocati con malloc");
  for (j=0; j<num_byte; j++) printf("%hd\n",*(punt1+j));
}
```

```
if (punt2 != NULL)
```

```
{puts("contenuto dei bytes allocati con calloc");
  for (j=0; j<num_byte; j++) printf("%hd\n",*(punt2+j));
}
```

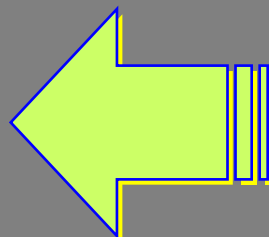
```
free(punt1); free(punt2);
```

```
}
```

Run

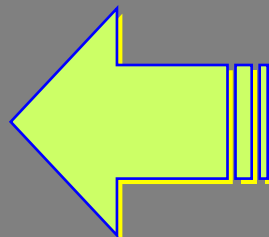
contenuto dei bytes allocati con malloc

8
3
61
0
8
3
61
0
0
0
0
0
0
0
0
0
0
0
0



non inizializzati

contenuto dei bytes allocati con calloc



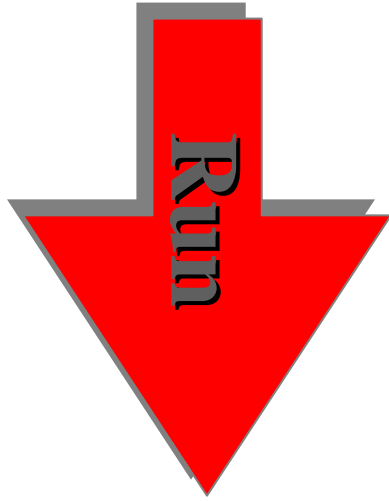
inizializzati a zero

Esempio 2

```
#include <stdio.h>
#include <stdlib.h>
void main()
{int i,n,npn; float *b;
 float a[]={0.f,1.f,2.f,3.f,4.f,5.f,6.f,7.f,8.f,9.f};
 n=10;
b=(float *)calloc(n,sizeof(float));
 ...
 for (i=0; i<n; i++) *(b+i)=a[i];
 puts("array prima di realloc");
 for (i=0; i<n; i++) printf("\tb[%2d] = %f\n",i,b[i]);
 npn=2*n;
b=(float *)realloc(b,npn*sizeof(float));
 ...
 for (i=0; i<n; i++) b[n+i]=a[n-1-i];
 puts("array dopo realloc");
 for (i=0; i<npn; i++) printf("\tb[%2d] = %f\n",i,b[i]);
}
```

alloca un array di **n** componenti reali

raddoppia il numero delle componenti



array prima di realloc

b[0]	=	0.000000
b[1]	=	1.000000
b[2]	=	2.000000
b[3]	=	3.000000
b[4]	=	4.000000
b[5]	=	5.000000
b[6]	=	6.000000
b[7]	=	7.000000
b[8]	=	8.000000
b[9]	=	9.000000

array dopo realloc

b[0]	=	0.000000
b[1]	=	1.000000
b[2]	=	2.000000
b[3]	=	3.000000
b[4]	=	4.000000
b[5]	=	5.000000
b[6]	=	6.000000
b[7]	=	7.000000
b[8]	=	8.000000
b[9]	=	9.000000
b[10]	=	9.000000
b[11]	=	8.000000
b[12]	=	7.000000
b[13]	=	6.000000
b[14]	=	5.000000
b[15]	=	4.000000
b[16]	=	3.000000
b[17]	=	2.000000
b[18]	=	1.000000
b[19]	=	0.000000

Esempio 3 - main

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

```
long *funzio(short , long *);
```

```
void main()
```

```
{short i, n=6; long *a;
```

è equivalente a

a=(long *)malloc(n*sizeof(long));

1ª realloc()

a=(long *)realloc(NULL, n*sizeof(long));

```
puts("\narray dopo 1a realloc");
```

```
for (i=0; i<n; i++) printf("a[%2d]=%d\n",i,a[i]);
```

```
for (i=0; i<n; i++) a[i]=i;
```

```
puts("\narray dopo definizione");
```

```
for (i=0; i<n; i++) printf("a[%2d]=%d\n",i,a[i]);
```

```
a=funzio(n, a); /* la function rialloca l'array a */
```

```
puts("\narray dopo 2a realloc");
```

```
for (i=0; i<2*n; i++) printf("a[%2d]=%d\n",i,a[i]);
```

```
for (i=n; i<2*n; i++) *(a+i)=10+i;
```

```
puts("array dopo modifica");
```


```
for (i=0; i<2*n; i++) printf("a[%2d]=%d\n",i,a[i]);
```

```
}
```

```
... ..
```

Esempio 3 - function

```
... ..  
long *funzio(short n, long a[])  
{short i; long temp;  
for (i=0; i<=n/2; i++)  
    {temp=a[i];  
      a[i]=a[n-1-i];  
      a[n-1-i]=temp;  
    }  
a=(long *)realloc((long *)a,(2*n)*sizeof(long));  
if (a == NULL) exit(1);  
return a;  
}
```



2ª realloc()

non inizializzati

array dopo 1a realloc

```
a[ 0]=3998400
a[ 1]=3998400
a[ 2]=0
a[ 3]=0
a[ 4]=0
a[ 5]=0
```

array dopo definizione

```
a[ 0]=0
a[ 1]=1
a[ 2]=2
a[ 3]=3
a[ 4]=4
a[ 5]=5
```

array dopo 2a realloc

```
a[ 0]=5
a[ 1]=4
a[ 2]=2
a[ 3]=3
a[ 4]=1
a[ 5]=0
```

```
a[ 6]=1766223201
a[ 7]=1030972780
a[ 8]=1348221507
a[ 9]=1919381362
a[10]=1768779105
a[11]=1818838620
```

array dopo modifica

```
a[ 0]=5
a[ 1]=4
a[ 2]=2
a[ 3]=3
a[ 4]=1
a[ 5]=0
a[ 6]=16
a[ 7]=17
a[ 8]=18
a[ 9]=19
a[10]=20
a[11]=21
```

È necessaria l'allocazione dinamica?

Qual è il massimo size di un array?

```
#include <stdio.h>
#define SIZE 262144
void main()
{unsigned long A[SIZE]; int k;
 printf("(OK!)\tSIZE = %d\n", SIZE);
 for (k=0; k<SIZE; k++)
 {
  A[k]=k%10;
  printf(((k%20 == 0) ? "%u, " : "%u, "), A[k]);
 }
}
```

array statico

Funziona* per SIZE=262144 (1.048.576 byte=1MB)

Non funziona* per SIZE=524288 (=2MB)

* Dipende dal PC

Qual è il massimo size di un array?

array dinamico

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    unsigned char *pA; unsigned long k, SIZE;
    SIZE=1024*1024;
    k=1; pA=(unsigned char *)malloc(k*SIZE);
    while (pA>0) finché l'allocazione viene eseguita...
    {
        printf("%d MB di memoria dinamica", k);
        free(pA);
        k++; pA=(unsigned char *)malloc(k*SIZE);
    }
} Funziona* fino a SIZE=1911 MB  $\approx 2^{31}$ =2GB
```

* Dipende dal PC

vers. 32bit: Il limite fisico è $2^{32}-1$ (lunghezza della voce intera long)

Gli array dinamici possono raggiungere un size molto maggiore di quelli statici