Memoria Dinámica

La función *malloc*

- La forma más habitual de C para obtener bloques de memoria es mediante la llamada a la función *malloc( )*.
- La función asigna un bloque de memoria que es el número de bytes pasados como argumento.
- *malloc( )* devuelve un puntero, que es la dirección del bloque asignado de memoria.
- El puntero se utiliza para referenciar el bloque de memoria y devuelve un puntero del tipo void*.

La función *malloc*

- La forma de llamar a la función *malloc( )* es:

```
puntero = malloc(tamaño en bytes);
```

- Generalmente se hará una conversión al tipo del puntero:

```
tipo *puntero;
puntero = (tipo *) malloc(tamaño en bytes);
```

- Por ejemplo:

```
long *p;
p = (long *) malloc(32);
```

La función *malloc*

- El operador unario **sizeof** se utiliza con mucha frecuencia en las funciones de asignación de memoria.

- El operador se aplica a un tipo de dato (o una variable), el valor resultante es el número de bytes que ocupa.

- Así, si se quiere reservar memoria para un buffer de 10 enteros:

```
int *r;
r = (int *) malloc( 10*sizeof(int) );
```

- Al llamar a la función *malloc( )* puede ocurrir que no haya memoria disponible, en ese caso *malloc ( )* devuelve NULL.

La función *malloc*

- El siguiente código utiliza *malloc( )* para asignar espacio para un valor entero:

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

- El siguiente código reserva memoria para un array de 100 números reales:

```
float *bloqueMem;
bloqueMem = (float *) malloc(100*sizeof(float));
```

La función *malloc*

- La reserva de n caracteres se puede declarar así:

```
int n;
char *s;

printf ("Numero de elementos: ");
scanf("%d", &n);

s = (char *) malloc(n * sizeof(char));
```

La función *malloc*

- La función *malloc*, definida en el archivo "**stdlib.h**", permite direccionar ***n*** bytes consecutivos de memoria siendo ***n*** un entero que le pasamos como argumento.

- La memoria que gestionamos con *malloc* permanece asignada durante toda la ejecución del programa y trasciende a la función que la invocó.

La función *malloc*

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

int main()
{
    double *ptr_lista;
    int i;

    ptr_lista = (double*) malloc(1000 * sizeof(double));
    if(ptr_lista == NULL)
    {
        puts ("Error en la asignación de memoria");
        return -1;    //intentar recuperar memoria
    }
    for (i = 0; i < 1000; i++)
        ptr_lista[i] = i;

    for (i = 0; i < 1000; i++)
        printf("%lf \n",ptr_lista[i]);
}
```

Supongamos, por ejemplo, que se desea asignar un array de 1,000 números reales en doble precisión:

```c
#include<stdio.h>
#include<stdlib.h>
struct complejo
{
    float x, y;
};
int main()
{
    int n, j;
    struct complejo *p;

    printf("Cuantos numeros complejos: ");
    scanf("%d",&n);

    p = (struct complejo*) malloc(n*sizeof(struct complejo));

    if (p == NULL) {
        puts("Error de asignacion de memoria.");
        exit (-1) ;
    }

    for (j = 0; j < n; j++, p++) {
        printf("Parte real e imaginaria del complejo %d: ",j);
        scanf ("%f %f", &p->x, &p->y);
    }
}
```

Asignación de memoria a estructuras

En este otro ejemplo se declara un tipo de dato complejo, se solicita cuántos números complejos se van a utilizar, se reserva memoria para ellos y se comprueba que existe memoria suficiente.

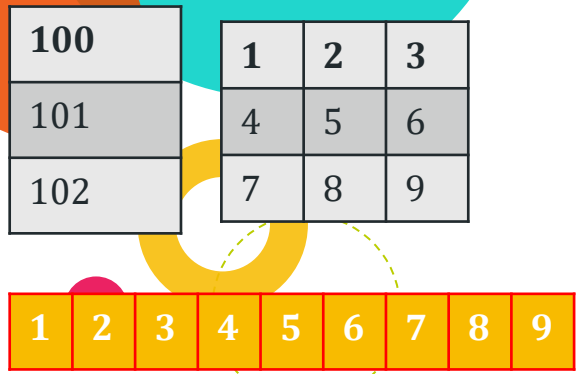
Al final, se leen los **n** números complejos.

Uso de malloc para arrays bidimensionales y rectangulares

1	2	3
4	5	6
7	8	9


```c
#include <stdio.h>
#include <stdlib.h>
#define filas 3
#define columnas 3
int main() {
    int i, j, contador;
    int *A;
    //Asignamos memoria dinamica con malloc
    A = (int *)malloc(filas * columnas * sizeof(int));
    //Inicialización de las matrices
    contador = 1;
    for (i=0 ; i<filas ; i++) {
        for(j=0 ; j<columnas ; j++) {
            A[i*columnas + j] = contador;
            contador++;
        }
    }
    //Impresion de la matriz
    for (i = 0; i<filas; i++) {
        for(j = 0; j<columnas; j++)
            printf("%d \t", A[i*columnas + j] ) ;
        printf("\n") ;
    }
    return 0 ;
}
```

Uso de malloc para arrays multidimensionales



- Un array bidimensional es, en realidad, un array cuyos elementos son arrays.
- Al ser el nombre de un array unidimensional un puntero constante, un array bidimensional será un puntero a puntero constante (tipo **).
- Para asignar memoria a un array multidimensional, se indica cada dimensión del array de igual forma que se declara un array unidimensional.

Uso de malloc para arrays multidimensionales

En el siguiente ejemplo se reserva memoria en tiempo de ejecución para una matriz de determinado número de **FILAS** y **COLUMNAS.**

1	2	3	4
5	6	7	8
9	10	11	12

```
#include <stdio.h>
#include <stdlib.h>
#define FILAS 3
#define COLUMNAS 4
int main()
{
    int **p, i, j, valor = 1;

    //Asignamos memoria para las filas
    p = (int**) malloc( FILAS*sizeof(int*) );

    for (i=0; i<FILAS; i++) {
        //Por cada fila asignamos memoria para las columnas
        p[i] = (int*) malloc( COLUMNAS*sizeof(int) );
        for (j=0; j<COLUMNAS; j++) {
            p[i][j] = valor;
            valor++;
        }
    }
    //Impresión de los valores asignados
    for (i=0; i<FILAS; i++) {
        for (j=0; j<COLUMNAS; j++)
            printf("%d \t",p[i][j]);
        printf("\n");
    }
    return 0;
}
```

Uso de malloc para arrays multidimensionales

- En el ejemplo anterior, la sentencia:

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

  reserva memoria para un array de **n** elementos, donde **n** es el número de FILAS, cada elemento es un puntero a entero (int *).

- Por cada iteración del bucle for externo se requiere reservar memoria para el número de elementos de la fila **(COLUMNAS)** con la sentencia:

  p[i] = (int *) malloc (COLUMNAS * sizeof(int)) ;

Liberación de memoria

- Cuando se ha terminado de utilizar un bloque de memoria previamente asignado por *malloc*, se puede liberar el espacio de memoria y dejarlo disponible para otros usos, mediante una llamada a la función ***free( )***.

- El bloque de memoria suprimido se devuelve al espacio de almacenamiento libre, de modo que habrá más memoria disponible para asignar otros bloques de memoria.

Liberación de memoria

- El formato de la llamada es:

```
free(puntero);
```

- Así, por ejemplo, para las declaraciones:

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

```
char *adc;
adc = (char*) malloc( 100*sizeof(char) );
```

- El espacio asignado se puede liberar con las sentencias:

```
free(ad);
```

```
free(adc);
```