MR18116

GUIA 04

sizeof

El operador sizeof es el operador más común en C. Es un operador unario en tiempo de compilación y se usa para calcular el tamaño de su operando. Devuelve el tamaño de una variable. Se puede aplicar a cualquier tipo de datos, tipo flotante, variables de tipo puntero.

Cuando sizeof () se usa con los tipos de datos, simplemente devuelve la cantidad de memoria asignada a ese tipo de datos. La salida puede ser diferente en diferentes máquinas, como un sistema de 32 bits puede mostrar una salida diferente, mientras que un sistema de 64 bits puede mostrar diferentes tipos de datos.

Aquí hay un ejemplo en lenguaje C,

Ejemplo

```
#include <stdio.h>int main() {

    char a = 'S';

    double b = 4.65;

    printf("Size of variable a : %d\n",sizeof(a));

    printf("Size of an expression : %d\n",sizeof(a+b));

    int s = (int)(a+b);

    printf("Size of explicitly converted expression : %d\n",sizeof(s));

    return 0;}
```

Salida

```
Size of variable a : 4

Size of int data type : 4

Size of char data type : 1
```

Size of float data type : 4

Size of double data type : 8

Cuando se usa sizeof () con una expresión, devuelve el tamaño de la expresión. Aquí hay un ejemplo.

Ejemplo

```
#include <stdio.h>int main() {

    char a = 'S';

    double b = 4.65;

    printf("Size of variable a : %d\n",sizeof(a));

    printf("Size of an expression : %d\n",sizeof(a+b));

    int s = (int)(a+b);

    printf("Size of explicitly converted expression : %d\n",sizeof(s));

    return 0;}
```

Salida

Size of variable a : 1

Size of an expression : 8

Size of explicitly converted expression : 4

MALLOC

Asigna determinados bytes de tamaño de almacenamiento no inicializado.

Si la asignación tiene éxito, devuelve un puntero al byte más bajo (el primero) en el bloque de memoria asignado que está alineado adecuadamente para cualquier tipo de objeto con alineación fundamental.

Si el tamaño es cero, el comportamiento está definido en la implementación (el puntero nulo puede ser devuelto, o algún puntero no nulo puede ser devuelto que no puede ser usado para acceder al almacenamiento, pero tiene que ser pasado a free).

malloc es seguro para hilos: se comporta como si sólo accediera a las posiciones de memoria visibles a través de su argumento, y no a cualquier almacenamiento estático.

Una llamada previa a free o realloc que desasigna una región de memoria *se sincroniza con* una llamada a un malloc que asigna la misma o parte de la misma región de memoria. Esta sincronización se produce después de cualquier acceso a la memoria por parte de la función de desasignación y antes de cualquier acceso a la memoria por parte de malloc. Hay un solo orden total de todas las funciones de asignación y desasignación que operan en cada región particular de la memoria.

```
#include <stdio.h>     #include <stdlib.h>
 int main(void) {
      int *p1 = malloc(4*sizeof(int));    // asigna suficiente para un arreglo de 4 int
      int *p2 = malloc(sizeof(int[4])); // lo mismo, nombrando el tipo directamente
      int *p3 = malloc(4*sizeof *p3);      // lo mismo, sin repetir el nombre del tipo

      if(p1) {
            for(int n=0; n<4; ++n) // rellena el arreglo
                  p1[n] = n*n;
            for(int n=0; n<4; ++n) // lo imprime fuera
                  printf("p1[%d] == %d\n", n, p1[n]);
      }

      free(p1);
      free(p2);
      free(p3);}
```

Salida:
p1[0] == 0
p1[1] == 1
p1[2] == 4
p1[3] == 9

Free

La función de biblioteca C **void free (void * ptr)** desasigna la memoria previamente asignada por una llamada a calloc, malloc o realloc.

Declaración

La siguiente es la declaración de la función free ().

```
void free(void *ptr)
```

Parámetros

ptr : este es el puntero a un bloque de memoria previamente asignado con malloc, calloc o realloc para ser desasignado. Si se pasa un puntero nulo como argumento, no se produce ninguna acción.

Valor de retorno

Esta función no devuelve ningún valor.

Ejemplo

El siguiente ejemplo muestra el uso de la función free ().

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

int main () {

   char *str;

   /* Initial memory allocation */

   str = (char *) malloc(15);

   strcpy(str, "tutorialspoint");

   printf("String = %s,  Address = %u\n", str, str);

   /* Reallocating memory */

   str = (char *) realloc(str, 25);

   strcat(str, ".com");

   printf("String = %s,  Address = %u\n", str, str);

   /* Deallocate allocated memory */
```

```
   free(str);


   return(0);}
```

Vamos a compilar y ejecutar el programa anterior que producirá el siguiente resultado:

```
String = tutorialspoint, Address = 355090448

String = tutorialspoint.com, Address = 355090448
```