Apuntes de operador new

Modelos Y Algoritmos para videojuegos I

Este apunte es una introducción al uso del operador new

Emmanuel Rojas Fredini & Sebastián Rojas Fredini

Espacios de memoria:

Existen 2 espacios de memoria en un programa donde se almacenan todas las variables, estas son:

1. **Stack o Pila:** Es donde se almacenan todas las variables estáticas, es decir que es donde se almacenaran aquellas variables que sepamos en el momento de compilar el programa cuanto ocuparan. Esto quiere decir que variables del tipo:

```
double b;
int a = 2;
char texto[] = "estoy escribiendo";
bool Tabla_de_Verdad[256];
```

2. **Heap o Montón:** Es donde se almacenaran todas las variables(especialmente arreglos) variables, es decir que su tamaño no lo sabemos en el momento de compilar. Es decir que los arreglos dinámicos irán en el heap, ya que podemos crear un arreglo dinámico del tamaño de una variable. Entonces aquí tendremos variables del tipo:

```
double b = (int*)malloc( sizeof(double));
int* a = (int*)malloc( sizeof(int));

*a= 300;
//creamos un arreglo del tamaño que tiene la variable // *a, en
nuestro caso será 300
//pero podríamos no saber con certeza
char* Arreglo = (char*)malloc( sizeof(char)*a);
```

La idea es que en el stack se almacene un cantidad no muy grande de datos, y en el heap los grandes volúmenes de datos y aquellas cantidades que queramos dimensionar de forma dinámica. Esto es debido a que al ejecutarse un programa el stack recibe una cantidad fija de memoria (1), la cual no puede crecer arbitrariamente, sino que esta fija. En cambio el heap puede crecer básicamente tanto como memoria RAM tenga la PC(o tanto como el sistema operativo quiera darnos memoria). Con esto queremos decir:

En el stack:

```
char Texto_de_Libro[10485760]; // es una locura
```

En el heap:

```
char* Texto_de_Libro = (char*)malloc(sizeof(char)* 10485760);
// en general esta ok
```

Esta discriminación de la memoria nos permite ver un punto importante: si usamos malloc, realloc, calloc estamos pidiendo memoria, es decir guardando nuestras variables en el heap.
Estas funciones son propias del lenguaje C, como sabemos el lenguaje que usamos(C++) es una extensión de C, o como dijo su propio creador, Bjarne Stroustrup**,** C con clases ya que lo que agrega C++ es la orientación a objetos. No obstante C++ al ser una extensión soporta toda la biblioteca y sintaxis (y semántica claro) de C, es decir por eso es que podemos usar por ejemplo malloc sin problemas. Pero con la llegada de C++ también vino una nueva forma de manejar la memoria, que dicho sea de paso encaja a la perfección con la orientación a objetos. Esta forma es mediante el operador new.

Operador **new**:

Este operador es el remplazo de malloc, es decir que le podremos "pedir" que nos dé una porción de memoria. Al igual que malloc también pedirá memoria del heap, y podemos almacenar arreglos dinámicos.

El operador new devuelve un puntero a algún tipo de dato, y necesita saber qué tipo de dato se desea guardar, y si es un arreglo lo que queremos el tamaño de este. La sintaxis es:

```
//para almacenar un solo valor
Tipo_dato* nombre_dato = new Tipo_dato;


//para almacenar un arreglo
Tipo_dato* nombre_dato = new Tipo_dato[ tamaño ];
```

Aquí:

- Tipo_dato es como dice un tipo de dato incluyendo clases por suepuesto, es decir: `int`, `char`, `double`, `float`, Caballero
- Nombre_dato es el nombre que le dimos a la variable
- Tamaño es la dimensión que deseamos que tenga el arreglo

Ejemplos:

```
int* num = new int;
*Num = 3;

Class Caballero:
{
Private:
      int Vidas;
Public:
Caballero();
Caballero(int nuevaVidas);
      void Ajusticiar();
};

//aquí se utilizara el constructor vacio o por defecto de la clase
Caballero
Caballero* Gawain = new Caballero;


//notar que abajo se usa el constructor al cual se le pasa las vidas
del Caballero
Caballero* Gawain = new Caballero(15);



// O si queremos arreglos

char* Texto_de_Libro = new char[10485760];
int* Numero_Telefono = new int[30];
Caballero* Mesa_Redonda = new Caballero[30];
```

Como vemos el operador new nos devuelve un puntero del tipo deseado, es decir no es necesario hacer el cast que debíamos hacer con malloc. Además algo fundamental: si creamos una clase con la función malloc de la forma:

```
Caballero* Gawain = (Caballero*)malloc(sizeof(Caballero));
```

Si prestamos atención vemos que en realidad malloc nos dará la memoria que ocuparía el objeto de clase Caballero, pero no construiría el objeto, es decir no llamara al constructor de Caballero. En cambio el operador new pedirá la memoria y además nos construirá la clase Caballero llamando al constructor. Para no confundir recordemos que si se lo crea en el stack se llamara al constructor y se creara de la forma debida(es decir: Caballero Gawain; funciona perfectamente).

Un detalle de suma importancia es que una vez que se termina de usar memoria solicitada a la función malloc, realloc, calloc o al operador new siempre hay que eliminar esa memoria. De no ser

asi dejaríamos de usar esa memoria pero no la liberaríamos, a esto se lo llama memory leak o escape de memoria.

Las funciones de C, malloc, calloc y realloc se liberan con la función free como sigue:

```
char* Arreglo = (char*)malloc( sizeof(char)*115);
free(Arreglo);
```

Y el operador new se libera con el operador delete de la siguente forma:

```
// si la variable es un solo valor
int* num = new int;
delete num;


// si la variable es un arreglo
Caballero* Mesa_Redonda = new Caballero[30];
delete[] Mesa_Redonda;
```

Es decir que si solicitamos un solo valor usamos delete sin corchetes, y si cuando pedimos memoria solicitamos un arreglo liberamos la memoria con delete[], los corchetes siempre van vacios, el lenguaje sabra cuanta memoria corresponde librerar y nosotros no podemos imponerle cuanta liberar.

Para mayor profundidad en el tema y especialmente la sobrecarga de operadores especiales como este se recomienda encarecidamente “The C++ Programming Language” de Bjarne Stroustrup.

Notas:

1) (1)En realidad los sistemas de manejo de memoria son más complejos que eso, la explicación fue una simplificación o si queremos usar un poco de lenguaje de POO (Programación Orientada a Objetos) una abstracción de la realidad. Por ejemplo en Windows maneja el stack al igual que el resto de la memoria con un sistema de memoria reservada y commiteada. Si se quiere expandir mas en el tema recomiendo las referencias de MSDN (http://msdn.microsoft.com/es-es/default.aspx).