

Décimosexta Sesión

Metodologías y Técnicas de Programación II

Programación Orientada a Objeto (POO) C++

Creación Dinámica de Objetos

16.1 Creación Dinámica de Objetos

Repaso de Sobrecarga de Operadores +, ++, [], (), int()

Podemos sobrecargar la suma de Complejo de forma general a través de una función o en particular como un método.

Operadores binarios: Uno de los argumentos soy “Yo mismo”.

Operadores unitarios: El argumento soy “Yo mismo”.

Acordaos de la diferencia entre prefijo y sufijo (++i i++)

La completa razon para la existencia de la sobrecarga de operadores es para **aquellas situaciones en que nos simplifica la vida.**

Los operadores sobrecargados son solo **funciones con nombres divertidos**, y las llamadas a funcion son llamadas por el compilador cuando se satisface el patrón adecuado.

Si la sobrecarga de operadores no proporciona un **beneficio significativo para el creador de la clase o para el usuario de la clase**, no compliquemos el asunto anadiéndolo.

16.2 Creación Dinámica de Objetos

Creación de Objetos

La creación de un objeto en C++ tiene lugar en dos pasos:

- 1.- Asignación de Memoria para el objeto.
- 2.- Llamada al constructor.

Aceptemos por ahora que este segundo paso ocurre siempre.

C++ lo fuerza, debido a que el uso de objetos no inicializados es una de las causas más frecuentes de errores de programación. Siempre se invoca al constructor, sin importar cómo ni dónde se crea el objeto.

16.2 Creación Dinámica de Objetos

Creación de Objetos: Asignación de Memoria

Puede ocurrir de varios modos y en diferente momento:

1.- Asignación de memoria en la zona de almacenamiento estático, que tiene lugar durante la carga del programa. El espacio de memoria asignado al objeto existe hasta que el programa termina.

2.- Asignación de memoria en la pila, cuando se alcanza algún punto determinado durante la ejecución del programa (la llave de apertura de un bloque). La memoria asignada se vuelve a liberar en el punto de ejecución complementario (la llave de cierre). Es necesario saber cuantas variables se necesitan mientras se escribe el programa de modo que el compilador pueda generar el código correspondiente.

3 Asignación dinámica, en una zona de memoria libre llamada heap o free store. Se reserva espacio para un objeto en esta zona mediante la llamada a una función durante la ejecución del programa. Se puede decidir en cualquier momento que se necesita cierta cantidad de memoria. Está fuera de las reglas de ámbito. Hay que decidir cuándo se libera.

16.2 Creación Dinámica de Objetos

Operador new

Cuando se crea un objeto mediante el operador **new**, éste se encarga de obtener el espacio necesario para el objeto y de llamar a su constructor.

```
TipoNuestro *ptn = new TipoNuestro(1,2);
```

Primero asigna espacio, comprueba que se haya recuperado correctamente y utilizando el puntero **this**, hace la llamada al constructor adecuado.

En las expresiones con **new** se puede usar cualquiera de los constructores disponibles para una clase.

Si éste no tiene argumentos, se escribe la expresión sin lista de argumentos

```
TipoNuestro *ptn = new TipoNuestro;
```

La creación dinámica de objetos es muy simple: una única expresión realiza todo el trabajo de cálculo de tamaño, asignación, comprobaciones de seguridad y conversión de tipo. La creación dinámica de objetos es casi tan sencilla como la creación en la pila.

16.2 Creación Dinámica de Objetos

Operador delete

Delete primero llama al destructor y después libera la memoria. EL argumento para delete debe ser una dirección: Puntero a objeto creado con new:

```
delete ptn;
```

El uso del operador delete debe reservarse solo para los objetos que hayan sido creados mediante new.

No ocurre nada si el puntero que se le pasa a delete es nulo. Por esa razón, a menudo se recomienda asignar cero al puntero inmediatamente después de usar delete; se evita así que pueda ser usado de nuevo como argumento para delete.

Tratar de destruir un objeto más de una vez es un error de consecuencias imprevisibles.

16.2 Creación Dinámica de Objetos

Gestión de Memoria

El compilador conoce con exactitud el tipo, la cantidad y el ámbito de los objetos automáticos que se crean en la pila.

El espacio asignado a los objetos y su duración queda bien definido en el código que genera.

Se requiere mayor gasto tanto en tiempo como en espacio para la creación dinámica de objetos.

Hay que buscar un hueco en la memoria que esté disponible para el objeto que estamos creando.

16.2 Creación Dinámica de Objetos

Delete void* : Mal asunto

```
class Objeto
{
void* datos_;
const int tam_;
const char id_;
public:
Objeto(int a, char c): tam_(a),
    id_(c)
{
    datos_ = new char[tam_];
    cout<<"Constructor de"<< id_;
}
~Objeto()
{
    cout<<"DEstructor de"<< id_;
    delete [] datos_; //OK Simple
}
};
```

```
int main()
{
    Objeto* a = new Objeto(40, 'a');
    delete a;
    void* b = new Objeto(40, 'b');
    delete b;
}
```

INICIO SALIDA POR PANTALLA

Constructor de a

DEStrcutor de a

Constructor de b

FIN

¡¡Como resultado obtenemos un programa con una silenciosa fuga de memoria!!

16.2 Creación Dinámica de Objetos

Operadores new y delete con arrays

En C++ se pueden crear vectores de objetos ya sea en la pila o en el heap, con la certeza de que se producirá la llamada al constructor para cada uno de los objetos del vector.

Hay una restricción: debe existir un constructor por defecto, o sea, sin argumentos, que es el que será llamado para cada objeto.

```
TipoNuestro *ptn = new TipoNuestro;
```

```
TipoNuestro *pa = new TipoNuestro[100];
```

Se asigna espacio suficiente para 100 objetos del tipo adecuado y se llama uno a uno a sus constructores.

Pero ptn y pa son iguales para alguien que no haya visto estas sentencias ¿no?. Nosotros sabemos que pa[3] tiene sentido y que ptn[40] no lo tiene.

16.2 Creación Dinámica de Objetos

Operadores new y delete con arrays

```
delete ptn; // Correcto  
delete pa; // No obtenemos el resultado deseado
```

En el segundo caso ocurre lo mismo que en el primero: Se llama al destructor del elemento apuntado.

En el segundo caso nos estamos olvidando de llamar a los destructores de 99 objetos.

Sin embargo, sí se liberará toda la memoria asignada al vector, ya que fue obtenida como un único gran bloque cuyo tamaño quedó anotado en alguna parte por las rutinas de asignación.

```
delete ptn; // Correcto  
delete [] pa; // Correcto
```

Los corchetes indican al compilador la necesidad de generar el código para obtener el número de objetos en el vector, que fue guardado en alguna parte cuando se creó, y llamar al destructor para cada uno de dichos elementos.

16.2 Creación Dinámica de Objetos

¿Cómo hacer que un puntero se comporte como un array?

```
TipoNuestro *pa = new TipoNuestro[100];
```

Quiero que “**pa**” sea como si hubiera creado un array de forma estática. Es decir no quiero que se utilice para apuntar o otras cosas.

Tampoco quiero que se utilice con, por ejemplo, ++.

```
const TipoNuestro *pa = new TipoNuestro[100];
```

```
TipoNuestro * const pa = new TipoNuestro[100];
```

¿Qué opción es la “buena”?

16.2 Creación Dinámica de Objetos

Sobrecarga de los operadores new y delete

Cuando se ejecuta una expresión con new, ocurren dos cosas. Primero se asigna la memoria al ejecutar el código del operador new() y después se realiza la llamada al constructor.

En el caso de una expresión con delete, se llama primero al destructor y después se libera la memoria con el operador operator delete().

Las llamadas al constructor y destructor no están bajo el control del programador pero se pueden cambiar las funciones operator new() y operator delete().

Frecuentemente la razón para cambiar el asignador es la eficiencia; puede ser que se necesite crear y destruir tantos objetos de la misma clase que lo haga ineficaz en términos de velocidad: un cuello de botella.

En C++ es posible sobrecargar new y delete de forma global y de forma local a una clase.