Programación Avanzada

Pontificia Universidad
JAVERIANA
Colombia

Ing. Orlando Vasquez

Memoria dinámica y estática

La asignación de memoria en lenguaje C++ se puede realizar de tres formas:

Asignación estática: Es la memoria que se asigna cuando se inicia la ejecución de un programa y aplica para:

- Variables globales
- Variables de archivos
- Variables denominada como **static**

Asignación automática: Se asigna a las variables declaradas dentro de las funciones y su información se destruye una vez finaliza la ejecución de la función.

Asignación dinámica: Controla el tamaño exacto y la vida útil de las ubicaciones de memoria. Si la memoria asignada dinámicamente no se libera, se corre el riesgo de generar fugas y uso ineficiente de memoria lo que puede causar bloqueos en la ejecución del programa.

Puntero tipo void

Tenemos claro que en C++ un apuntador solo puede apuntar al tipo de dato del apuntador, de lo contrario se presenta error:

```
int *ptr;
double d = 5.0;
ptr = &d; // Error, no se puede asignar double* a int*
```

Solución: En C++ existe el apuntador vacio.

```
void *ptr; // puntero vacío
int i = 1; //
int double d = .0; // double
ptr = &i; // funciona para un entero
ptr = &d; // funciona para un double
```

apuntador_vacio.cpp

Memoria dinámica y estática

Ventajas de la memoria estatica	**Ventajas de la memoria dinámica**
Es mas fácil de manejar	Se puede incrementar durante la ejecución del programa.
No necesita ser destruida de forma explicita	Es eficiente para el manejo de listas, pilas, arboles, etc.
Se conoce desde el tiempo de compilacion del programa	Bien manejada, puede evitar el desperdicio de memoria

Memoria dinámica y estática

Funciones estandar C	Estandar C++
malloc: reserva un bloque de memoria y devuelve un apuntador void*. Posterior a este uso se debe validar si devuelve NULL. **void *malloc(size_t size)**	**Operador new :** Se encarga de reservar memoria para el tipo de datos indicado. El resultado de **new** es un apuntador al tipo indicado. void* operator new (std::size_t size) throw (std::bad_alloc);
calloc: Similar a malloc, pero adicional a reservar la memoria la inicializa a '0' **void *calloc(size_t nmemb, size_t size)**	**Operador delete:** Libera el bloque de memoria señalado por ptr (si no es nulo), liberando el espacio de almacenamiento previamente asignado por una llamada al operador **new** y haciendo que la ubicación del puntero sea inválida. Este operador reemplaza el garbage collector que existe en otros lenguajes como por ejemplo Java. void operator delete (void* ptr) throw();
realloc: redimensiona el espacio asignado de forma dinámica anteriormente a un apuntador. void *realloc(void *ptr, size_t size)	A diferencia de la funciones de C, los operadores en C++ asignan memoria en función de los elementos necesitados y no en base de los bytes necesarios a asignar a la memoria
free: Se utiliza para liberar la memoria asignada dinámicamente. void free(void *ptr)	

Memoria dinámica y estática

Ejemplos del operador **new** y **delete**
void* operator new (std::size_t size) throw (std::bad_alloc);

```
int *i ;
i= new int; /*Asigna una dirección de memoria que no pertenece a una variable*/

*i=20; /*Asigna un valor a la dirección de memoria reservada por el operador new*/

delete i; /*Liberar el espacio de memoria de i */

int *v = new int[10]; /*Declaracion dinamica de un arreglo*/
tipoDato *variable_arreglo =  new tipoDato[tamaño];

for (int k=0; k<10;k++)
    v[k]=k+3;

delete []v; /*Liberar la memoria dinamica de un arreglo*/

delete v; /*Solo libera la primera posición de memoria del arreglo, no se recomienda*/
```

Memoria dinámica y estática

Ejemplos del operador **new** y **delete**

Uso de memoria dinámica con structs

```
typedef struct estudiante{
        char nombre [10];
        char carrera[20];
};
estudiante *est = new estudiante ;
strcpy(est->nombre,"ramon");  /*Para accéder a los miembros de la estructura se debe utilizar ->
strcpy(est->carrera,"ingenieria");
cout <<"\nnombre :"<<est->nombre;
cout <<"\ncarrera :"<<est->carrera<<endl;
```

Memoria dinámica y estática

Ejemplos del operador **new** y **delete**

Uso de memoria dinámica con structs y arreglos

Ejercicio1.cpp

```
int registros=3;
estudiante *ests =  new estudiante[3];
for (int i=0;i<registros;i++){
        cout << "Estudiante "<<i+1<<endl;
        cout <<" Ingrese nombre :";
        cin.getline(ests[i].nombre,10);
        cout <<" Ingrese carrera :";
        cin.getline(ests[i].carrera,20);
}

for (int i=0;i<registros;i++){
        cout << "Estudiante "<<i+1<<endl;
        cout << "Nombre: "<< ests[i].nombre<<endl;
        cout << "Carrera: "<< ests[i].carrera <<endl;
}
```

Memoria dinámica y estática

Ejemplos del operador **new** y **delete** con arreglos

Ejercicio2.cpp

tipoDato **variable //Declara arreglo de dos dimensiones
variable = new tipoDato* [filas]; // M3:

Para determinar el numero de columnas, lo haremos dentro un ciclo for

Ejemplo:

```
int fil = 3;                    //Número de filas
int col = 5;                    //Número de columnas
double** data;
data = new double* [fil]; //Establecer filas
for (int j = 0; j < fil; j++)    //
        data[j] = new double [col]; // Establecer columnas
```

Pontificia Universidad JAVERIANA Colombia

Memoria dinámica y estática

Paso de apuntadores por valor

Cuando enviamos a una función un apuntador, este por defecto se envia por valor, es decir se puede modificar el valor de la dirección de memoria a la cual apunta, pero no se puede cambiar la dirección de memoria; si esta se cambia, este cambio tiene efecto solo dentro de la función.

Ej:

Ejercicio3.cpp

```
#include <iostream>
int global=42;
using namespace std;

int numero (int *p){ //La variable apuntador se pasa por valor
        *p=8;
        p=&global;
        cout <<" p:"<<*p<<endl;
}
using namespace std;
int main (){
        int j=10;
        int *pj = &j;
        cout <<" pj:"<<*pj<<endl;
        numero (pj);
        cout <<" pj:"<<*pj<<endl;
        cout <<" j:"<<j<<endl;
}
```

Memoria dinámica y estática

Paso de apuntadores por referencia
Para pasar un apuntador por referencia se debe indicar con *&

Ejercicio4.cpp

```
#include <iostream>
int global=42;
using namespace std;

int numero (int *&p){ //La variable apuntador se pasa por referencia

		*p=8;
		p=&global;
		cout <<" p:"<<*p<<endl;
}
using namespace std;
int main (){
		int j=10;
		int *pj = &j;
		cout <<" pj:"<<*pj<<endl;
		numero (pj);
		cout <<" pj:"<<*pj<<endl;
		cout <<" j:"<<j<<endl;
}
```

Pontificia Universidad JAVERIANA Colombia

Memoria dinámica y estática

Retorno de arreglos creados con memoria dinámica

Ejercicio5.cpp

ConcatenarCadenas.cpp

Uso de apuntadores y memoria dinámica para crear una lista simple

Listas1.cpp

Pontificia Universidad
JAVERIANA
Colombia