Fundamentos de Programación 101

By Ernie

Ernesto José Canales Guillén

Círculos de estudio UCA

Ciclo Virtual 01/2021

using namespaces

namespace

- Permiten agrupar entidades como clases, objetos y funciones bajo un nombre. De esta forma el ámbito global se puede dividir en "sub-ámbitos", cada uno con su propio nombre.
  - Los namespaces aparecen solo en el ámbito global.
  - Los namespaces se pueden anidar dentro de otro namespace.
  - Los namespaces no tienen especificadores de acceso. (Público o privado).
  - No es necesario poner punto y coma después de la llave de cierre de la definición de un namespaces.
  - Podemos dividir la definición de un namespaces.

```
1 namespace namespace_name {
2    // code declarations
3 }
```

```cpp
#include <iostream>
using namespace std;

// first name space
namespace first_space {
   void func() {
      cout << "Inside first_space" << endl;
   }
}

// second name space
namespace second_space {
   void func() {
      cout << "Inside second_space" << endl;
   }
}

int main () {
   // Calls function from first name space.
   first_space::func(); //Output Inside first_space

   // Calls function from second name space.
   second_space::func(); //Output Inside second_space

   return 0;
}
```

Operadores new y delete

Memoria dinámica

Memoria dinámica

- La asignación de memoria dinámica en C/C++ se refiere a realizar la asignación de memoria manualmente por parte del programador.

- La memoria asignada dinámicamente se asigna en “Salto” y, las variables locales y no estáticas se asigna en la pila.

  - Un uso, es asignar memoria de tamaño variable, lo que no es posible con la memoria asignada por el compilador, excepto las matrices de longitud variable.

  - El uso más importante es la flexibilidad proporcionada a los programadores. Somos libres de asignar y desasignar memoria cuando lo necesitemos y cuando ya no lo necesitemos.
    - Ejemplos de tales casos son Lista enlazada, Árbol, etc…

¿Cómo se asigna/desasigna la memoria?

- C usa la función **malloc()** y **calloc()** para asignar memoria dinámicamente en tiempo de ejecución y usa la función **free()** para liberar memoria asignada dinámicamente.

- C++ soporta estas funciones y además cuenta con dos operadores **new** y **delete** que realizan la tarea de asignar y liberar la memoria de una forma mejor y más sencilla.

new

- El operador **new** denota una solicitud de asignación de memoria en Free Store.
- Si hay suficiente memoria disponible, el nuevo operador inicializa la memoria y devuelve la dirección de la memoria recién asignada e inicializada a la variable de puntero.

```
1 //To allocate memory of any data type
2 pointer-variable = new data-type;
3
4 //To allocate block of memory
5 pointer-variable = new data-type[size];
```

delete

- Dado que es responsabilidad del programador desasignar la memoria asignada dinámicamente, C++ proporciona a los programadores el operador **delete**.
- Si el programador no desasigna la memoria, provoca una pérdida de memoria (la memoria no se desasigna hasta que el programa termina).

```
1 //To release memory pointed by pointer-variable
2 delete pointer-variable;
3
4 //To release block of memory pointed by pointer-variable
5 delete[] pointer-variable;
```

```
#include <fstream>
```

Bibliografía

- L. J. Aguilar, Programación en C++. Algoritmos, estructuras de datos y objetos, Aravaca (Madrid): McGRAW-HILL, 2006.
- D. Malik, C++ Programming: From Problem Analysis to Program Design, Boston, MA: Cengage Learning, 2003.
- Microsoft Official Documentation – C++