Laboratorio 05

Punteros

1. Competencias

1.1. Competencias del curso

Conoce, comprende e implementa programas usando punteros en el lenguaje de programación C++.

1.2. Competencia del laboratorio

Conoce, comprende e implementa programas usando punteros en el lenguaje de programación C++.

2. Equipos y Materiales

- Un computador.
- IDE para C++.
- Compilador para C++.

3. Marco Teórico

3.1. Los punteros y elementos dinámicos en C++

Los punteros en C++ (o apuntadores) son quizá uno de los temas que más confusión causan al momento de aprender a programar en C++, sin embargo, verás que no es para tanto y que todo depende de dos elementos: el signo & (ampersand) y el * (asterisco). Verás que no es para nada difícil hacer y usar punteros y que además son de gran ayuda al momento de necesitar valores y estructuras dinámicas, por ejemplo, para crear un array dinámico, con dinámico me refiero a que su tamaño puede ser establecido en tiempo de ejecución y lo mismo se puede hacer con las matrices (que en realidad son un array multidimensional).

3.1.1. Ejemplo de punteros

```
int variable; //Creamos un entero
int * apuntador = &variable; //Creamos un puntero a la posición en memoria de "variable"
*apuntador = 20; //Le asignamos un valor a esa posición de memoria.
```

Muy bien, ya hemos creado y usado nuestro primer puntero ¿Notaste el uso del asterisco y del ampersand? espero que sí y además de eso hay otros detalles que debemos considerar:

Detalles al crear y usar punteros en C++

El tipo de dato del apuntador debe coincidir con el de la variable cuya posición en memoria apuntan. En el ejemplo vemos que tanto variable como apuntador son enteros.
Siempre que queremos usar el apuntador debemos anteponer el asterisco (*) para indicar que usaremos el valor en la posición de memoria apuntada.
De no usar el asterisco el comportamiento sería impredecible. Estaremos haciendo uso de la dirección de memoria más no del valor almacenado en ésta.
Un puntero o apuntador puede ser de cualquier tipo de dato, inclusive los podemos usar con tipos complejos.
Ya que sabemos algunos trucos y detalles sobre los apuntadores en C++, vamos a definir formalmente la utilidad del operador & y del asterisco.

Los punteros y el ampersand

El **ampersand** es un operador de C++ y es comúnmente utilizado para los punteros. Este operador nos permite obtener la dirección de memoria de una variable cualquiera y es justo esto (la dirección en memoria) lo que utilizan los punteros para referenciar valores.

El **asterisco** es, por decirlo de alguna forma, el operador por excelencia de los punteros. Su utilidad radica en que, si el valor de dicho apuntador corresponde a una dirección de memoria, el asterisco nos permite resolverla y acceder al valor almacenado allí. Viéndolo desde otro enfoque, un apuntador es únicamente una dirección de memoria (un número) y el asterisco es el que hace la magia de obtener el valor referenciado por dicha dirección.

```
cout << variable; // 20
cout << &variable; // Posición en memoria
cout << apuntador; // Posición en memoria
cout << *apuntador; // 20
```

Veamos otro ejemplo con cada elemento detallado paso a paso

3.1.2. Otro ejemplo de punteros

```
char *apuntador = NULL; //Declaramos un puntero
//Es recomendable inicializar un puntero en null, para detectar errores fácilmente
char letra; //Declaramos una variable primitiva
apuntador = &letra; //Asignamos al apuntador la dirección de memoria de la variable primitiva
*apuntador = 'x'; //Modificamos la variable a través del apuntador
cout << letra; //Muestra x por pantalla
```

En este ejemplo vemos que podemos usar cualquier tipo de dato, que un puntero se puede inicializar independientemente y luego se le puede asignar su referencia correspondiente. Nótese que al asignar (línea 6) no utilizamos el asterisco, pues estamos definiendo la dirección de memoria y no el valor en dicha dirección (recuerda que el * resuelve la dirección de memoria y no es lo que requerimos en esa línea).

Ahora que hemos visto los ejemplos y tenemos claro el uso del ampersand y el asterisco podemos entonces realizar algunos ejercicios interesantes.

3.1.3. Parámetros por referencia

Usualmente al enviar un parámetro a una función todo lo que se haga con dicho parámetro allí adentro NO tiene efecto por fuera. Por ejemplo, si a una función la se le envía una variable cuyo valor es diez y al interior de la función le sumamos un cinco, después de la ejecución de la función el valor de la variable seguirá siendo diez en vez de quince. Lo que pasó al interior de la función se quedó allí. Para solucionar esto, si queremos que el valor cambie definitivamente, usamos punteros para pasar no el valor del parámetro sino una referencia a éste (paso por referencia). Veamos:

```
#include "iostream"
#include "stdio.h"

using namespace std;

int funcion(int valor)
{
   valor = valor + 10; //Se le suma 10 -> valor = 20
   return valor;
}
```

```
int funcionPunteros(int* valor)
{
   *valor = *valor + 10; //Se le suma 10 a la posición en memoria
   return *valor;
}

int main()
{
   int numero = 10;
   cout << "Antes de función " << numero << "\n"; //10
   funcion(numero); //Se pasa por valor
   cout << "Después de función " << numero << "\n"; //10
   cout << "Antes de funcionPunteros " << numero << "\n"; //10
   funcionPunteros(&numero); //Se envía la dirección de memoria y la función resuelve la referencia
   cout << "Después de funcionPunteros " << numero << "\n"; //20 (10+10)
   system("pause");
   return 0;
}
```

Como podrás comprobar si ejecutas el código del ejercicio al llamar a "funcion" sólo enviamos el valor y por ende éste no es modificado por fuera de la función, con "funcionPunteros" estamos manipulando la posición en memoria del parámetro recibido (por eso usamos el *) y por ende al ejecutarla el valor de la variable se modifica. De ese modo ya hicimos el primer ejercicio con punteros en C++ y ya comprendemos el paso por referencia.

3.1.4. Array dinámico

Por medio de punteros podemos crear arreglos o vectores dinámicos, es decir, un array al cual se le define su tamaño o capacidad durante la ejecución del código y no antes, lo cual nos permite definirle el tamaño deseado por el usuario.

Para este ejercicio retomaré el ejemplo de arreglos o vectores: Queremos crear un programa con el cual podamos guardar los títulos y los autores de diferentes libros sin perder ninguno de ellos. El usuario es el encargado de suministrar la información de cada libro. En esta ocasión ya sabemos usar punteros, así que será también el usuario quien nos diga cuántos libros desea ingresar, ya no necesitamos suponer que sólo ingresará 5 libros.

```
#include "stdafx.h"
#include "iostream"
#include "stdio.h"
#include "string"

using namespace std;

int main()
{
   string* titulos = NULL; //Se inicializa el puntero (inicia en null)
   string* autores = NULL; //Se inicializa el puntero (inicia en null)

   int tamanio; //Se inicializa la variable
   cout << "Cuantos libros desea ingresar?";
   string entrada;
   getline(cin, entrada); //Se asigna el valor ingresado
   tamanio = stoi(entrada); //Se transforma la entrada en número
   //Declaramos un arreglo del tamaño ingresado para los títulos
   titulos = new string[tamanio];
   //Declaramos un arreglo del tamaño ingresado para los autores
   autores = new string[tamanio];
   cout << "Por favor ingrese la siguiente información de los Libros: \n";
   for(int i = 0; i < tamanio; i++)
   {
      cout << "\n******* Libro " << i + 1 << "********:\n";
      cout << "Titulo: ";
      //cin >> titulos[i]; //No funciona con espacios
      getline(cin, titulos[i]);
      cout << "Autor: ";
      //cin >> autores[i]; //No funciona con espacios
      getline(cin, autores[i]);
   }
   //Liberamos la memoria de ambos punteros
   delete [] titulos;
   delete [] autores;
   titulos = NULL;
   autores = NULL;

   system("pause");
   return 0;
}
```

Así entonces tuvimos dos punteros, uno para todos los autores y otro para todos los títulos. Haciendo uso de ellos pudimos definir la cantidad de libros a ingresar por medio del usuario, es decir lo hicimos de manera dinámica, en tiempo de ejecución.

3.1.5. Matrices dinámicas

Así como lo hicimos con los arreglos, también podemos tener matrices dinámicas y definir su tamaño, número de filas o número de columnas (o las dos) según sea necesario.

Para esto tomaré el mismo ejemplo de los libros, pero usando una matriz, en vez de dos vectores, tal y como se solucionó en la sección de matrices veamos:

```
#include "stdafx.h"
#include "iostream"
#include "stdio.h"
#include "string"

using namespace std;

int main()
{
   int cols = 2; //El número de columnas es fijo (sólo título y autor)
   string** libros; //Si inicializa la matriz (punteros de punteros)
   int tamanio ; //Se inicializa la variable
   cout << "Cuantos libros desea ingresar?";
   string entrada;
   getline(cin, entrada); //Se asigna el valor ingresado
   tamanio = stoi(entrada); //Se transforma la entrada en número
   libros = new string*[tamanio]; //Se asigna el número de filas según el usuario
   cout << "Por favor ingrese la siguiente información de los Libros: \n";
   string titulo ,autor;
   for(int i = 0; i < tamanio; i++)
   {
      libros[i] = new string[cols]; //Cada fila contendrá dos columnas
      //Notar que cols pudo haber sido ingresada por el usuario también

      cout << "\n******* Libro " << i + 1 << "********:\n";
      cout << "Titulo: ";
      getline(cin,titulo);
      cout << "Autor: ";
      getline(cin,autor);
      libros[i][0] = titulo;
      libros[i][1] = autor;
   }
```

```
    //Para liberar la memoria debemos recorrer fila por fila primero.
    for (int i = 0; i < tamanio; ++i)
    {
       delete [] libros[i]; //Cada fila de libros es otro array de punteros
       //Por eso son punteros a punteros
    }

    //Luego de limpiar las columnas, quitamos la fila única que quedó
    delete [] libros;

    system("pause");
    return 0;
}
```

Este ejercicio es el perfecto para aclarar dudas o darse cuenta si realmente comprendes el concepto de apuntadores y su aplicación para arrays dinámicos. Debido a que la cantidad de columnas es fija, no se lo pedimos al usuario, simplemente lo declaramos con valor dos. Luego tenemos el puntero, pero no es un puntero cualquiera, al ser una matriz, será un puntero que tendrá otros punteros adentro, por eso se usa doble asterisco, luego se obtiene el tamaño del usuario (cantidad de libros) y al momento de inicializar la fila estamos indicando que es un arreglo de punteros, por eso se usa el *. Luego al interior del ciclo, cuando estamos llenando la matriz, debemos indicar que cada fila estará compuesta por un array de punteros de tamaño dos (dos columnas) y así construimos nuestra matriz dinámica.

Debes notar también que la liberación de la memoria es un poco más trabajosa, pues debemos ir fila por fila liberando la memoria de las columnas creadas y luego liberar la fila completa. Ahí podrás notar la diferencia en eficiencia y uso de memoria al usar arreglos o usar matrices.

Nota: La instrucción new utilizada con los strings es necesaria para asignar el espacio en memoria para esa colección de elementos. No te preocupes por ello de momento, solo asegúrate de usarla y ya el lenguaje se hará cargo del resto.

4. Ejercicios

Resolver los siguientes ejercicios planteados:

1. Asignar valores a dos variables enteras, intercambie estos valores almacenados usando solo punteros a enteros.
2. Cree dos vectores con valores flotantes y asígnele valores aleatorios, para esto deberá de asignar memoria a cada vector. Calcule el producto punto de vectores y muestre por pantalla. Una vez finalizado este proceso, retire la memoria asignada a los

punteros. Repita este proceso de asignación y retiro de memoria dentro de un for de 1 000 000 veces.

3. Construya una lista enlazada simple utilizando solo punteros. Añada las funciones de insertar y eliminar un elemento. En la función insertar se debe asegurar que los números insertados estén en orden creciente. Simule el proceso con 10000 números aleatorios sin que el programa falle.
4. Construya una lista enlazada que almacene tanto números como cadenas de texto utilizando punteros void. Incluya una función de búsqueda de muestre un dato almacenado además del tipo de dato que se encuentra almacenado (int, float, char, …)
5. Implemente su propia función de concatenación de cadenas de texto especial (¡no es la función ordinaria de concatenación!), recibirá como parámetro dos punteros de caracteres y tendrá que asignar el contenido del segundo puntero al INICIO del primer puntero. La función no retorna ningún valor.
6. Utilizando punteros a funciones, implemente las funciones:
    a. void sumar (int a, int b);
    b. void restar (int a, int b);
    c. void multiplicar (int a, int b);
    d. void dividir (int a, int b);

    Cree un vector de punteros a funciones e implemente un programa que permita la invocación de cada función, pero a través del puntero.

5. Entregables

Al final estudiante deberá:

1. Compactar el código elaborado y subirlo al aula virtual de trabajo. Agregue sus datos personales como comentario en cada archivo de código elaborado.
2. Elaborar un documento que incluya tanto el código como capturas de pantalla de la ejecución del programa. Este documento debe de estar en formato PDF.
3. El nombre del archivo (comprimido como el documento PDF), será su LAB05_GRUPO_A/B/C_CUI_1erNOMBRE_1erAPELLIDO.

   (Ejemplo: LAB05_GRUPO_A _2022123_PEDRO_VASQUEZ).
4. Debe remitir el documento ejecutable con el siguiente formato:

   LAB04_GRUPO_A/B/C_CUI_ EJECUTABLE_1erNOMBRE_1erAPELLIDO

   (Ejemplo: LAB05_GRUPO_A_EJECUTABLE_2022123_PEDRO_VASQUEZ).

En caso de encontrarse trabajos similares, los alumnos involucrados no tendrán evaluación y serán sujetos a sanción.