Punteros en c++

Definición: Un puntero es una variable que contiene la dirección de memoria de otra variable. Los punteros permiten código más compacto y eficiente; utilizándolos en forma ordenada dan gran flexibilidad a la programación.

La sintaxis de un puntero es:

```
int *ptrID, ID;
ID = 8;
ptrID = &ID; // puntero a ID
```

ptrID es un puntero a int, mientras que la variable ID es solo una variable del tipo int. Todo puntero debe ser precedido por un asterisco **(*)** en la declaración.

Se puede declarar más de un puntero en la misma sentencia

Los operadores utilizados en los punteros son

Existen dos operadores a tener en cuenta cuando trabajamos con punteros. El **operador de dirección (&)** que devuelve la dirección de memoria de su operando y el **operador de indirección (*)** que devuelve un alias para el objeto al cual apunta el operando del puntero.

```
int *ptrX;
ptrX = &X;
```

Por ejemplo en este se utiliza el operador de indirección para la variable *ptrX, asignándole un valor para la X

Tenemos a los vectores o arrays, estos son:

Los vectores son punteros constantes. Un vector sin subíndice es un puntero al primer elemento del vector. Una matriz es un vector de vectores. de manera que en cada elemento del primer vector "se cuelga" otro vector, pudiendo hacer así referencia a filas y columnas.

```
int X[15];
int *ptrX;
ptrX = X; // ptrX recibe la dirección del primer elemento ( 0 ) de X
```

Así como también podría escribirse

```
int X[15];
int *ptrX;
ptrX = &X[0]; // ptrX es igual a la dirección del primer elemento de X
```

Se pueden utilizar distintos elementos del vector teniendo en cuenta la sintaxis de punteros.

```
int X[15], Y, *ptrX;
ptrX = X;

Y = *( ptrX + 7 );
```

En este caso puede verse que Y toma el valor del elemento 7 del vector X, siendo 7 el desplazamiento dentro del vector. El operador de indirección queda fuera del paréntesis porque tiene una prioridad superior a la del operador +. De no existir los paréntesis, se sumaria 7 al elemento X[0]. Teniendo en cuenta que los vectores son punteros constantes, el nombre del vector puede tratarse como un puntero:

```
Y = *( X + 7 );
```

Utilizar punteros para aritmetica:

Al usar punteros a matrices, hay que tener en cuenta que la aritmética cambia sensiblemente.

Un ejemplo es:

```
#include <iostream>

using std::cout;
using std::endl;

void main()
{
   int X[6] = { 1, 2, 3, 4, 5, 6 };
   int *prtX;

   prtX = X; // incializo el valor del puntero.

   cout << endl << *prtX;
   prtX += 2;
   cout << endl << *prtX;
   prtX -= 2;
   cout << endl << *prtX;
   prtX++;
   cout << endl << *prtX;
}
```

Por ejemplo en este código, primero se crea un puntero a un arreglo de 6 elementos y si inicializa el puntero *prtX* al primer elemento del arreglo *X[0]*. Si tenemos en cuenta que el siguiente ejemplo se ejecuta en una computadora con enteros de 4 bytes, el segundo elemento de la matriz tendrá en memoria un desplazamiento de 4 bytes, el 2 de ocho y asi sucesivamente. La operación *prtX += 2;* produce un desplazamiento llevándolo al 3 elemento dentro del arreglo. Debe entenderse que prtX ahora apunta a una dirección de memoria y la instrucción cambia esta dirección de memoria sumándole 2 multiplicado por el tamaño del tipo de dato del arreglo que en este supuesto sería de 4. *(dir = (dir + 2 * 4))*, dando por resultado un desplazamiento de 8 bytes. Sería igual que ejecutar la operación *prtX = &X[2];*. La operación *prtX -= 2* obedece a la misma lógica estableciendo el puntero al primer elemento del array *X[0]* y el operador ++ modifica el puntero desplazándolo 4 bytes y asignándole el segundo elemento de la matriz.

Las matrices de punteros:

Para realizar una estructura de datos dinámica, se puede utilizar una matriz donde sus elementos sean punteros. Suponiendo que queramos hacer un calendario y lo dividamos por semanas. Podríamos utilizar una matriz con los días de la semana.

```
const char *dias[7] = { "Domingo", "Lunes", "Martes", "Miercoles", "Jueves", "Viernes", "Sabado" }
```

Cada día de la semana, no es un elemento de la matriz, sino que la expresión dias[7] crea una matriz de siete elementos como punteros a char.

Pasar un valor a referencia, esto se hace:

Un puntero constante a un dato no constante

```
#include <iostream>
void sumoeldoble( int * ); // prototipo

void main ()
{
   int X = 15;
    sumoeldoble( &X ); // Pasa la dirección de memoria de X .
   std::cout << X;

}

void sumoeldoble( int *ptrX )
{
   // Toma el valor de X mediante el operador de indirección
   // La funcion no devuelve nada porque modifica el valor por referencia.
   *ptrX = *ptrX + ( *ptrX * 2 );
}
```

Un puntero no constante a un dato constante

```
#include <iostream>

void imprimeChars( const char * ); // prototipo

void main ()
{
   char cFrase[] = "Hola Mundo";
   imprimeChars( cFrase );
}

void imprimeChars( const char *ptrStr )
{
   for ( ; *ptrStr != '\0'; ptrStr++ ) //Sin inicialización
      std::cout << *ptrStr;
}
```

Un puntero constante a un dato no constante

Un puntero es constante cuando apunta siempre a la misma dirección de memoria y si el dato no es constante entonces el valor puede ser modificado.

```
void main ()
{
   int foo, bar;
    int * const ptrFoo = &foo; // Apuntador constante a un entero en la dirección de memoria de foo

   *ptrFoo = 53;


   // Esto devuelve un error porque es un puntero constante.
   // No se puede alterar la dirección de memoria a la que apunta.
   ptrFoo = &bar;
}
```

Un puntero constante a un dato constante

El puntero constante apunta siempre a la misma dirección de memoria y el valor al que apunta dicho puntero no puede ser modificado mediante el puntero. Este es el metodo en que se debe pasar matrices a funciones que solo leen la matriz y no la modifican.

```
#include <iostream>

using namespace std;

int main ()
{
   int foo = 3, bar;
   const int * const ptrFoo = &foo;

            cout << foo;

            *ptrFoo = 53; //Error porque no puede alterarse el valor
   ptrFoo = &bar; // Error

            foo = 23;

            cout << foo;

}
```

De punteros a funciones

Teniendo en cuenta que el nombre de una función es en realidad la dirección de memoria donde comienza el código, los punteros a funciones contienen la dirección de memoria de la función a la que apunta, y se pueden pasar y retornar entre funciones.

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

bool funcionA( int, int, bool (*)( int ) ); //Prototipo
bool funcionB( int ); //Prototipo

void main()
{
   int x = 113, y = 226;
   if ( funcionA( x, y, funcionB ) )
                  cout << "\nEl resultado es verdadero";
            else
                  cout << "\nEl resultado es falso";
}

bool funcionA( int param1, int param2, bool (*verificar)( int ) )
{
            if ( ( (*verificar)( param1 ) ) && ( (*verificar)( param2 ) ) )
```

```
                return true;
}

bool funcionB( int param )
{
        if ( param > 100 )
                return true;
        else
                return false;
}
```

En el ejemplo anterior podrá ver que en la definición de los prototipos, la función funcionA recibe tres parámetros, siendo el tercer parámetro un puntero a la función funcionB, prácticamente copiando el prototipo sin el nombre. Cuando se ejecuta funcionA, se le pasa como parámetro el nombre de la función

Hacer un ordenamiento burbuja utilizando punteros

```
#include <iostream>
#include <iomanip>

using namespace std;

void orden( int *, const int ); // prototipo
void swap( int * const, int * const ); // prototipo

int main()
{
  const int nSize = 10;
  int a[ nSize ] = { 3, 9, 14, 27, 18, 154, 8, 6, 74, 33 };

  cout << "\nElementos a ordenar\n";

  for ( int j = 0; j < nSize; j++ )
    cout << setw( 5 ) << a[ j ];

  orden( a, nSize );  // ordena el arreglo

  cout << "\nElementos ordenados\n";

  for ( int j = 0; j < nSize; j++ )
    cout << setw( 5 ) << a[ j ];

  cout << endl;
```

```
   return 0;  // indica terminación exitosa

}

void orden( int *matriz, const int nSize )
{
   for ( int pasada = 0; pasada < nSize - 1; pasada++ )
   {
      for ( int k = 0; k < nSize - 1; k++ )
      {
         if ( matriz[ k ] > matriz[ k + 1 ] )
         {
            swap( &matriz[ k ], &matriz[ k + 1 ] );
         }
      }
   }
}

void swap( int * const ptrElemento1, int * const ptrElemento2 )
{
   int mantiene = *ptrElemento1;
   *ptrElemento1 = *ptrElemento2;
   *ptrElemento2 = mantiene;
}
```

Mi ejemplo de apuntador es:

```
//AIME MEDINA 26/11/2019

// Este programa sirve para buscar la direccion de memoria en la que esta guardado un numero, el programa te devuelve la dirección de memoria en hexadecimal

#include <iostream>

#include <conio.h>

using namespace std;


int main() {

   int num, *dir_num;

   num=20;

   dir_num= &num;
```

```
   cout<<"Numero"<<*dir_num<<endl;
   cout<<"Direccion de memoria"<<dir_num<<endl;
   getch();
         return 0;
}
```

```
//AIME MEDINA 26/11/2019
// Este programa sirve para buscar la direccion de memoria en la que esta guardado un numero, el programa te devuelve la dirección de memoria en hexadecimal
#include <iostream>
#include <conio.h>
using namespace std;

int main() {

    int num, *dir_num;
    num=20;
    dir_num= &num;
    cout<<"Numero"<<*dir_num<<endl;
    cout<<"Direccion de memoria"<<dir_num<<endl;
    getch();
    return 0;
}
```

```
Numero20
Direccion de memoria0x23fe44

Time limit exceeded
```