INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERIA MECANICA Y ELECTRICA

INGENIERIA EN COMUNICACIONES Y ELECTRONICA

FUNDAMENTOS DE PROGRAMACION

PROF. OSCAR CRUZ

Actividad 8

TRABAJO DE INVESTIGACION

(APUNTADORES)

Alumno:

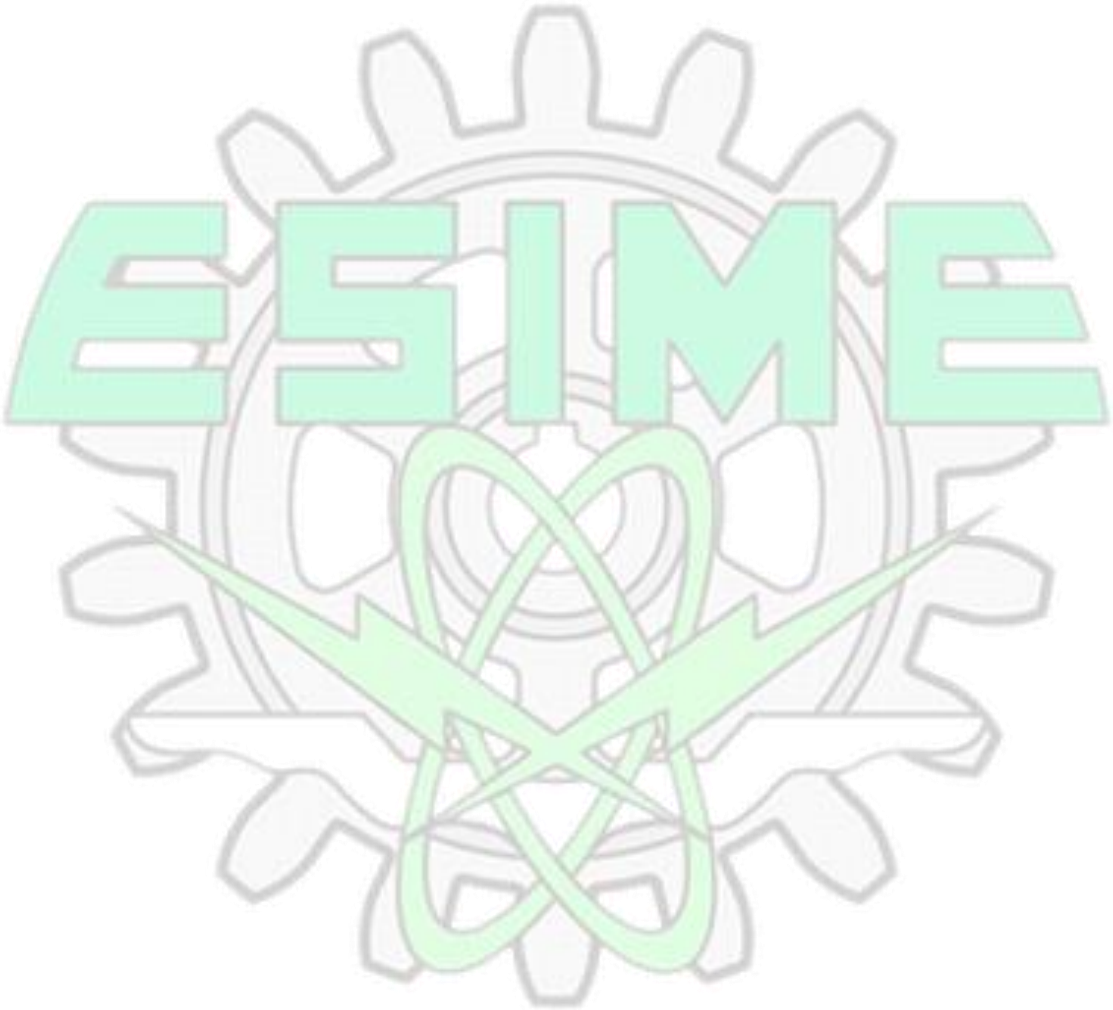
DIAZ ANAYA EDUARDO

Boleta:

2020300206

Desarrollar los temas de investigación para:

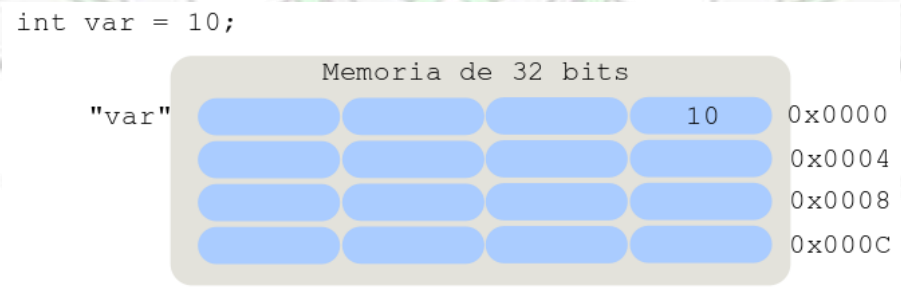
1. Que son los apuntadores.
2. Que es un mapa de memoria.
3. Relación entre vectores y apuntadores.



1.- ¿Qué es un Apuntador?

Los punteros (o apuntadores) son variables que se utilizan para almacenar direcciones de memoria, puntualmente las direcciones de memoria que fueron asignadas a variables convencionales en las que se almacenan datos de distinto tipo. Vale la pena entonces recordar que a todas las variables en C++ se les asigna un espacio de memoria en el cual se va almacenar el valor que se le asigne en algún punto de la aplicación a esa variable, el tamaño de dicho espacio va depender del tipo de dato que se pretende almacenar en la variable, del compilador y de la arquitectura del procesador. Cada uno de los espacios de memoria cuenta con una dirección para identificarlo, esta dirección es por lo general un número en representación hexadecimal. Es precisamente ese número correspondiente a la dirección lo que se almacena en un puntero.

Observe la siguiente imagen de ejemplo, se declara una variable var y se inicializa directamente en la declaración, dicha variable recibe un espacio en memoria para almacenar el valor que se le asigna en la inicialización. Dicho espacio en memoria tiene su propia dirección para poder ser referenciado.



Se puede declarar un puntero para almacenar la dirección de memoria correspondiente a la variable var, es decir, se puede "apuntar" un puntero a la variable var. Para declarar un puntero se utiliza la sintaxis para declaración de variables: calificadores opcionales, modificadores opcionales, tipo de dato obligatorio y un identifador para el puntero que también es obligatorio. El tipo de dato del puntero debe ser obligatoriamente el mismo tipo de dato de la variable a la que se pretende apuntar, es decir, si se requiere almacenar la dirección en memoria de una variable de tipo int, entonces el tipo de dato del puntero también debe ser int. Un puntero se distingue de otras variables porque en su declaración se utiliza el operador *

luego del tipo de dato y antes del identificador del puntero. Observe a continuación la declaración de varios punteros:

```
int *puntero_a_int;
float *puntero_a_float;
ClaseA *puntero_a_objeto_claseA;
```

Para apuntar un puntero a una variable se utilizan el operador de asignación =, el operador & y la variable a la que se quiere apuntar. Con el operador & se obtiene la dirección de la variable y se le asigna al puntero mediante el operador de asignación =. Observe, ejecute y analice el ejemplo a continuación:

```
1  #include<iostream>
2  using namespace std;
3
4  class MiClase
5  {
6      int x;
7      public:
8      MiClase():x(0){}
9  };
10
11 int main()
12 {
13     int var = 250;  //Una variable cualquiera
14     int *ptr_var = &var;  //Apuntando un puntero a la variable var
15
16     MiClase obj;  //Un objeto cualquiera
17     MiClase *ptr_obj = &obj;  //Apuntando un puntero al objeto obj
18
19     cout<<"Valor de la variable var: "<<var<<endl;
20     cout<<"Direccion de la variable var: "<<&var<<endl;
21     cout<<"Direccion almacenada en el puntero ptr_var: "<<ptr_var<<endl;
22     cout<<"-------------------------------------------------------"<<endl;
23     cout<<"Direccion del objeto obj: "<<&obj<<endl;
24     cout<<"Direccion almacenada en el puntero ptr_obj: "<<ptr_obj<<endl;
25     return 0;
26 }
27
```

Acceso a miembros de clase mediante puntero

Para acceder a los miembros de clase de un objeto a través de un puntero se utiliza el operador flecha -> en lugar del operador punto ., obviamente el acceso con operador flecha sigue respetando los niveles de acceso establecidos en la definición de la clase.

Aritmética de punteros

Los punteros almacenan un valor que corresponde a una dirección de memoria y el lenguaje de programación C++ permite que un puntero pueda recibir una nueva dirección de memoria, es decir que sea apuntado a otra variable. Es claro que esto último solo se prodra lograr si la nueva posición de memoria almacena un dato del mismo tipo del puntero o sí se hace una conversión explícita del tipo de dato del puntero. Para poder desplazar un puntero por la memoria C++ permite ejecutar los operadores de adición y sustracción en los punteros, por tanto la dirección que almacena el puntero se puede incrementar o decrementar de acuerdo a la operación que se involucre al puntero. Está permitido el uso de los siguientes operadores: +, -, ++ y -- para ejecutar operaciones de aritmética de punteros.

2.- MAPA DE MEMORIA

Un mapa de memoria (del inglés memory map) es una estructura de datos (tablas) que indica cómo está distribuida la memoria. Contiene información sobre el tamaño total de memoria y las relaciones que existen entre direcciones lógicas y físicas, además de poder proveer otros detalles específicos sobre la arquitectura del computador. Es capaz de direccionar un microprocesador. Distribucion de la misma, es decir que direcciones ocupan los diferentes dispositivos destinados a funciones determinadas

La especificacion del mapa de la memoria se puede realizer como:

Funcional: ubicacion (direcciones) de los elementos (hardware o software) del Sistema digital, atendiendo a la function de los mismos. Asi se describiran la ubicacion de: sectores del programa, posicion de datos generales y tablas, registros de interfaz,etc.

Fisico: correspondencia entre las direcciones del mapa y el dispositivo fisico en el que se plasman. De acuerdo a el se realizara la conexion entre los diferentes dispositivos, teniendo en cuenta la estructura del bus de direcciones y el bus de datos, la forma de seleccion de dispositivos, etc.

JERARQUIA DE LA MEMORIA

Se conoce como jerarquía de memoria a la organización piramidal de la memoria en niveles que tienen los ordenadores. Su objetivo es conseguir el rendimiento de una memoria de gran velocidad al coste de una memoria de baja velocidad, basándose en el principio de cercanía de referencias.

Los puntos básicos relacionados con la memoria pueden resumirse en:

- Capacidad
- Velocidad
- Coste

La cuestión de la capacidad es simple, cuanto más memoria haya disponible, más podrá utilizarse. La velocidad óptima para la memoria es la velocidad a la que el procesador puede trabajar, de modo que no haya tiempos de espera entre cálculo y cálculo, utilizados para traer operandos o guardar resultados. En suma, el coste de la memoria no debe ser excesivo, para que sea factible construir un equipo accesible.

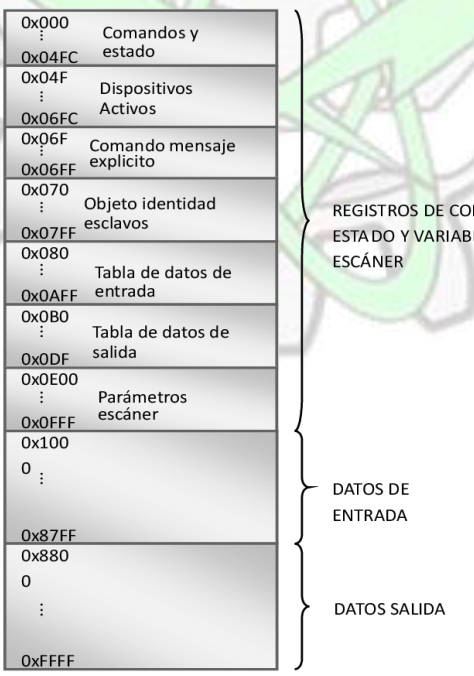
Como puede esperarse los tres factores compiten entre sí, por lo que hay que encontrar un equilibrio. Las siguientes afirmaciones son válidas:

- A menor tiempo de acceso mayor coste.
- A mayor capacidad menor coste por bit.
- A mayor capacidad menor velocidad.

Se busca entonces contar con capacidad suficiente de memoria, con una velocidad que sirva para satisfacer la demanda de rendimiento y con un coste que no sea excesivo. Gracias a un principio llamado cercanía de referencias, es factible utilizar una mezcla de los distintos tipos y lograr un rendimiento cercano al de la memoria más rápida.

Los niveles que componen la jerarquía de memoria habitualmente son:

- Nivel 0: Registros
- Nivel 1: Memoria caché
- Nivel 2: Memoria principal
- Nivel 3: Memorias flash
- Nivel 4: Disco duro (con el mecanismo de memoria virtual)
- Nivel 5: Cintas magnéticas Consideradas las más lentas, con mayor capacidad.
- Nivel 6: Redes (Actualmente se considera un nivel más de la jerarquía de memorias)



3.- RELACION ENTRE VECTORES Y APUNTADORES

Existe una diferencia entre un nombre de arreglo y un apuntador. que debe tenerse en mente. Un apuntador es una variable. Por esto pa = a y pa++ son legales. Pero un nombre de arreglo no es una variable; construcciones como a = pa y a++ son ilegales.

Por ejemplo, el siguiente fragmento de código no tiene sentido:

```
1. int v1[] = {1,0,1};
2. int v2[] = {2,0,2};
3. v2 = v1; // no se puede copiar un vector en otro, v1 y v2 son
   nombres no variables, no guardan nada, sólo designan espacios
4. v2++ // lo mismo, esta expresión es como escribir v2 = v2 + 1
```

Existe una importante diferencia entre estas definiciones:

char amessage[] = "ya es el tiempo"; /* arreglo */
char *pmessage = "ya es el tiempo"; /* apuntador */

amessage es un arreglo, suficientemente grande como para contener la secuencia de caracteres y el que lo inicializa. Se pueden modificar caracteres individuales dentro del arreglo, pero amessage siempre se referira a la misma localidad de almacenamiento. Por otro lado, pmessage es un apuntador inicializado para apuntar a una cadena constante; el apuntador puede modificarse posteriormente para que apunte a algun otro lado, pero el resultado es indefinido si trata de modificar el contenido de la cadena.
Precisamente del hecho de que los vectores no se puedan reasignar y que los punteros sí, se deriva el que los vectores no sean punteros. No obstante el identificador de un vector sí que es, por definición, la dirección del primer elemento del vector.

En síntesis podríamos decir que:

- ✓ El identificador de un vector no puede aparecer a la izquierda de una asignación, un puntero sí.
- ✓ El identificador del vector es, por definición, la dirección al primer elemento.
- ✓ Podemos acceder a los elementos del vector bien con la notación de punteros, bien con la notación de los corchetes. Son equivalentes.

BIBLIOGRAFIAS:

https://unoyunodiez.wordpress.com/2011/02/09/la-verdad-sobre-vectores-y-punteros-y-const/

http://uin14131.blogspot.com/p/mapa-de-memoria-un-mapa-de-memoria-del.html

https://www.codingame.com/playgrounds/51214/manejo-dinamico-de-memoria-y-polimorfismo-practica-4/punteros-en-c

https://aprende.olimpiada-informatica.org/cpp-vector