INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERIA MECANICA Y ELECTRICA

INGENIERIA EN COMUNICACIONES Y ELECTRONICA

FUNDAMENTOS DE PROGRAMACION

PROF. OSCAR CRUZ

Actividad 7

TRABAJO DE INVESTIGACION

Alumno:

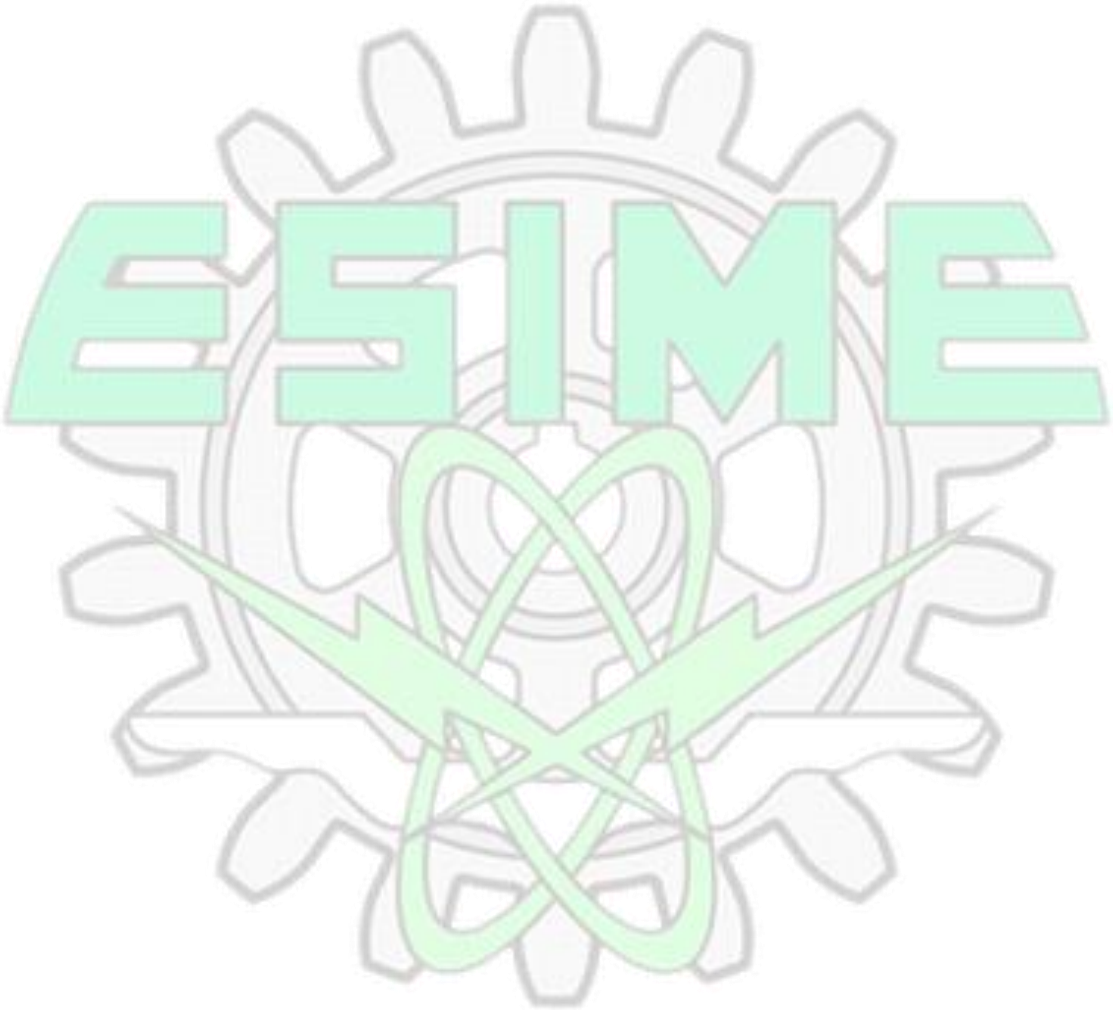
DIAZ ANAYA EDUARDO

Boleta:

2020300206

Desarrollar los temas de investigación para:

1. Que son las estructuras de datos en C++
2. Que son los Vectores
3. Que son los Matices



1.- ¿Qué son las estructuras de datos en C++?

Las estructuras de datos en C++ se pueden entender como un tipo de dato compuesto (no complejo). Las estructuras de datos permiten almacenar de manera ordenada una serie de valores dados en una misma variable. Las estructuras de datos más comunes son los vectores o arreglos y las matrices, aunque hay otras un poco más diferentes como son el struct y las enumeraciones.

Las estructuras de datos han sido creadas para solucionar una gran variedad de problemáticas que no eran solucionables (o al menos no muy fácilmente) con los tipos de datos primitivos.

Las estructuras de datos se pueden ver como una agrupación o estructuración para una serie de tipos de datos primitivos (aunque también pueden poseer tipos de datos complejos) que pueden ser fácilmente utilizadas e identificadas. Sin la existencia de las estructuras de datos sería imposible o bastante complicado por ejemplo conocer y manejar todos los números de identificación, nombres y direcciones de todos los usuarios de un sistema (que normalmente serían muchísimos) pero ahora con las estructuras de datos es muy simple definir una serie de posiciones en memoria para cada valor que deseamos guardar o definir un orden o valores específicos para cada campo y accediendo a ellos generalmente por medio de una única variable, todo esto es sencillo hacerlo con el uso de las estructuras de datos y sin desperdiciar recursos.

Las estructuras de datos se emplean con el objetivo principal de organizar los datos contenidos dentro de la memoria del ordenador. Así, nuestra primera experiencia con estructuras comienza desde el momento mismo en que usamos en nuestros programas variables de tipos primitivos (char, short, int, float, etc). A la memoria del ordenador se le puede considerar como un gran bloque compuesto por una serie de BYTES dispuestos secuencialmente uno detrás de otro. por ejemplo, si un ordenador posee una memoria de 128MB (128 megas) entonces se le puede leer o escribir desde el BYTE 0 hasta el BYTE 128MB - 1 (0000000H .. 7FFFFFFH)

La idea de ver la memoria como un serie de bytes es buena, sin embargo no es suficiente ya que en la misma podemos guardar números, cadenas de caracteres, funciones, objetos, etc. de tal manera que surge la necesidad de establecer los mecanismos adecuados para dar cuenta de la forma, tamaño y objetivo de los datos almacenados. Según el tipo de microprocesador,

estos tienen la capacidad para manipular o direccionar estructuras compuestas por uno, dos, cuatro, etc, bytes; de donde se derivan los tipos que comunmente se conocen como: BYTE, WORD, DWORD, QWORD y TWORD.

La estructura mínima de información manipulable en un sistema de computación es el BIT el cual se agrupa normalmente en bloques de 8 para formar un BYTE. Cabe mencionar que los BITS no son direccionables directamente, sino a través de compuertas AND, OR, NOT, XOR, las cuales en C y C++ se escriben como &, |, ~ y ^, conocidos como "Bitwise operators" u "Operadores de manipulación de bits".

En C,C++ existe una serie de estructuras básicas o tipos primitivos, los cuales pueden ser usados por el programador para declarar variables, y también son el fundamento sobre el cual se crean estructuras complejas. El tamaño de los tipos primitivos no es estándar ya que los mismos dependen de factores tales como:

Tipo del microprocesador

El compilador

Sin embargo, en la actualidad, la mayoria de compiladores de C y C++ soportan los siguientes tipos con la longitud indicada:

Tipos primitivos

Nombre común	Nombre C	Longitud	Procesador 64 bits
BYTE	char	8 bits	8 bits
WORD	short	16 bits	16 bits
DWORD	int	32 bits	32 bits
DWORD	long	32 bits	64 bits
DWORD	float	32 bits	32 bits
QWORD	double	64 bits	64 bits
TWORD	long double	80 bits	128 bits

Nota: en el lenguaje C,C++ existe el operador **sizeof()**, con el cual se puede obtener el tamaño (número de bytes) ocupados por un tipo específico. Por ejemplo, **sizeof(int)** regresa el número de bytes ocupados por los datos de tipo **int**.

2.- VECTORES

El tipo de variable vector permite almacenar varios valores bajo una sola variable y realizar operaciones complejas con esta. Todos los valores dentro del vector han de ser del mismo tipo (todas int o todas string o todas char o todas bool, ...).

Lo primero que se ha de hacer para poder utilizar vectores es añadir al principio del código esta línea:

```
#include <vector>
```

Una variable de tipo vector se declara así:

```
vector < tipo_de_los_valores > nombre_de_la_variable( tamanio );
```

Por ejemplo:

```
vector<int> puntos_equipos(15); // Esto declara un vector de 15 valores
```

Podemos dar un valor inicial a todas las posiciones del vector. Por ejemplo, inicialmente queremos que todos los equipos tengan 0 puntos porque la competición todavía no ha comenzado:

```
vector<int> puntos_equipos(15, 0);
```

Los valores se pueden leer y asignar igual que con los arrays: puntos_equipos[0], puntos_equipos[1], ...

Los dos parámetros pueden ser variables:

```
int numEquipos = 15;
int puntosIniciales = 0;
vector<int> puntos_equipos(numEquipos, puntosIniciales);
```

Podemos mirar cuantos valores tiene este *vector* con su subsintrucción *.size()*:

```
cout << puntos_equipos.size();
```

Esto mostrará por pantalla "*15*". No importa si todas las 15 posiciones del vector tienen valor o no, te dirá el tamaño total del *vector*.

Podemos añadir un elemento nuevo a un *vector* con su subinstrucción *.push_back(valor)*. Por ejemplo si ahora hacemos *puntos_equipos.push_back(10)* el vector pasaría a ser de 16 posiciones. *push_back()* siempre añade el valor al final del vector (al final de su tamaño, aunque tenga posiciones sin valor).

Por lo tanto incluso podríamos crear un vector sin tamaño (tamaño *0*) e ir añadiendo con *push_back()* uno a uno los elementos que leemos del *cin*:

```
vector<int> puntos_equipos;
int numEquipos;
cin >> numEquipos;
while(numEquipos > 0) {
    int puntos;
    cin >> puntos;
    punts_equipos.push_back(puntos);
    numEquipos = numEquipos - 1;
}
```

Vector ofrece muchas más subinstruccions como *pop_back()*, *insert()*, *erase()*, *sort()*, que pueden ser útiles en muchos momentos.

Cuando quieres enviar como parámetro un vector a una función, a la función receptora se le debe poner un "*&*" antes del nombre del parámetro:

```
#include <iostream>
#include <vector>
using namespace std;
void muestraPosicion(vector<int> &parametro, int index) {
    cout << parametro[index];
}
int main() {
    vector<int> lista;
    lista = { 6, 8, 7, 3, 2, 8, 9, 3 };
    int posicion;
    cin >> posicion;
    muestraPosicion(lista, posicion);
}
```

Esto es porque normalmente cuando pasamos un parámetro lo que hace C++ realmente es hacer una copia del valor y pasar esta copia a la función. Pero con los vectores casi siempre lo que queremos es que la función receptora modifique el vector y nos lo devuelva con las modificaciones, por lo tanto no queremos que reciba una copia sino que queremos que reciba el original y trabaje directamente sobre nuestra variable.

Esto, evidentemente, también se puede hacer con cualquier otro tipo de variable como *int*, *string, char,* etc:

```
#include <iostream>
using namespace std;
void dobla(int &n) {
    n = n * 2;
}
int main() {
    int num;
    cin >> num;
    dobla(num);
    cout << num;
}
```

3.-MATICES

Una Matriz (en inglés, array, también denominado arreglo) es una estructura usada para agrupar bajo un mismo nombre

listas de datos de un mismo tipo.

El tipo de matriz puede ser cualquiera, sin embargo cada componente tiene que ser del mismo tipo. En C estándar solamente da soporte para matrices estáticas, mientras que con C++ se pueden crear matrices dinámicas pudiendo usar la librería estándar de plantillas (STL).

Matrices estáticas

Una matriz estática es una estructura cuyo tamaño es determinado en tiempo de compilación, es decir, una vez establecido el tamaño de la matriz ésta no podrá cambiarse durante el tiempo de ejecución. En C, C++ para declarar un arreglo estático de datos se emplea la sintaxis:

```
tipo identificador[ [tamaño] ] [ = { lista de inicialización } ] ;
```

donde:

- tipo se refiere al tipo de datos que contendrá la matriz. El tipo puede ser cualquiera de los tipos estándar (char, int, float, etc.) o un tipo definido por el usuario. Es más, el tipo de la matriz puede ser de una estructura creada con: struct, union y class.
- identificador se refiere al nombre que se le dará a la matriz.
- tamaño es opcional e indica el número de elementos que contendrá la matriz. Si una matriz se declara sin tamaño, la misma no podrá contener elemento alguno a menos que en la declaración se emplee una lista de inicialización.
- lista de inicialización es opcional y se usa para establecer valores para cada uno de los componentes de la matriz. Si la matriz es declarada con un tamaño especifíco, el número de valores inicializados no podrá ser mayor a dicho tamaño.

Ejemplos:

```
int intA[5];
long longA[5] = { 1, 2, 3, 4, 5 };
char charA[] = { 'a', 'b', 'c' };
```

Acceso a los miembros de una matriz de datos:

En orden de acceder a los miembros de una matriz se debe indicar el nombre de la matriz seguido de dos corchetes, dentro de los cuales se debe especificar el índice del elemento deseado. Se debe aclarar que los índices son números o expresiones enteras y que en C, C++ estos tienen un rango permitido de 0 a T-1 (T = tamaño de la matriz).

Ejemplos: dadas las matrices intA, charA, longA (ejemplo anterior)

```
intA[0] = 100; // establece el valor del elemento 0 de intA a 100.

charA[3] = 'O'; // establece el valor del elemento 3 de charA a 'O'.

cout << longA[0]; // muestra por pantalla el elemento 0 de longA, que
es longA[0].
```

Matrices dinámicas

Una matriz dinámica es una estructura compleja y, ya que C estándar no da el soporte para operar con estos tipos de estructuras, le corresponde al programador crear los algoritmos necesarios para su implementación. Crear lista dinámicas de datos en C estándar no es una tarea para programadores inexpertos, ya que para lograr tal objetivo se necesita tener conocimentos solidos acerca de los punteros y el comportamiento de los mismos. Los usuarios de C++ pueden auxiliarse de la librería estándar de plantillas, conocidas por sus siglas en ingles como STL.

BIBLIOGRAFIAS:

https://www.programarya.com/Cursos/C++/Estructuras-de-Datos

https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Estructuras

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

https://docs.microsoft.com/es-es/cpp/cpp/arrays-cpp?view=vs-2019