Corporacion Univeritaria Minuto de Dios

UNIMINUTO

Asignatura:

Estructura de Datos

NRC :7233

Tema

Arreglos en C++

Presentado:

Michael Daniel Murillo López
ID:534830

DOCENTE:

Segundo Fidel Puerto Garavito

Colombia Ciudad De Bogotá D.C septiembre 27 de 2017

Crear un programa en C++ donde implemente los siguientes requisitos

Una empresa solicita un programa donde busquen por el número de torre y apartamento. Utilizando Arreglos.

Los arrays, arreglos o vectores forman parte de la amplia variedad de estructuras de datos que nos ofrece C++, siendo además una de las principales y más útiles estructuras que podremos tener como herramienta de programación

```
#include <iostream> // librerias
#include <string>
#include <list> //manejos de listas
#include <fstream> //manejo de archivos
#include <cctype>
```

se incluyen ciertos archivos llamados bibliotecas más comúnmente librerías. Las bibliotecas contienen el código objeto de muchos programas que permiten hacer cosas comunes, como leer el teclado, escribir en la pantalla, manejar números, realizar funciones matemáticas, etc.

```
using namespace std;

//clase que representa un apartamento
class Apartamento {
public:
	int piso;
	int apto;
};
//lista de pisos
list<Apartamento> apartamentos[100];
int n = 0;
```

Creamos las clases enteresa de piso y apartemento en el cual llegan las funciones enteras.

```
// compara dos apartamentos para ordenarlos ascendentemente
bool compararAsc(const Apartamento& first, const Apartamento& second) {
	return (first.apto < second.apto);
}
// compara dos apartamentos para ordenarlos descendentemente

bool compararDesc(const Apartamento& first, const Apartamento& second) {
	return (first.apto > second.apto);
}
```

Analizar dos o más objetos para luego poder establecer las diferencias y las semejanzas que mantienen entre sí.

```
// ordena  orden ascendente

void ordenarAsc() {
	for (int i = 0; i < 26; i++) {
		apartamentos[i].sort(compararAsc);
	}
}
// ordena en orden descendente

void ordenarDesc() {
	for (int i = 0; i < 26; i++) {
		apartamentos[i].sort(compararDesc);
	}
}
```

Se pueden ordenar filas completas o solamente las celdas seleccionadas, basándose en la primera columna de la selección. Si sus datos consisten en filas de hechos relacionados entre sí, ordenar por filas completas es más seguir; de otra manera, sus registros pueden mezclarse fácilmente! Usar el diálogo Ordenar es más flexible que hacerlo mediante los botones de la barra de herramientas. El diálogo le permitirá seleccionar cuales columna(s) usar como base del ordenamiento.

```
// lista los apartamentos ascendentemente
void ListarAsc() {
	//llama al metodo ordenar ascendentemente
	ordenarAsc();
	//crea un iterador de la lista
	list<Apartamento>::iterator it;
	//imprime mensajes
	cout << "Listar Ascendentemente" << endl;

	//itera en las 26 listas
	for (int i = 0;i < n;i++) {
		cout << "Piso:"<<(i+1)<<endl;
		//imprime la lista de apartamentos de la letra i
		for (it = apartamentos[i].begin(); it != apartamentos[i].end();
++it) {
			std::cout << it->apto << "\t";
		}
		cout << endl;
	}
}
//similar a ordenar ascendentemente solo que lo hace al contrario
void ListarDesc() {
	ordenarDesc();
	list<Apartamento>::iterator it;
	cout << "Listar Descendentemente" << endl;
	for (int i = n-1;i >= 0;i--) {
		cout << "Piso:" << (i + 1) << endl;
		for (it = apartamentos[i].begin(); it != apartamentos[i].end();
++it) {
			std::cout << it->apto << "\t";
		}
		cout << endl;
```

```
	}

}
```

puede llegar a ser casi tan sencillo como el manejo de la entrada y salida estándar (pantalla y teclado), con la diferencia de que abrimos el fichero (*open*) antes de trabajar con él y lo cerramos (*close*) al terminar. Por ejemplo, para escribir una frase en un fichero de texto (que se crearía automáticamente), podríamos usar un fichero de salida (*ofstream*)

```
//lee el archivo de datos apartamentos.txt
void leerArchivo() {
	//abre el archivo
	std::fstream f("apartamentos.txt", std::ios_base::in);

	string in;
	//lee de a 6 lineas que representan un apartamento, cuando llega a la
ultima linea no encuentra otro y finalixa
		//crea el apartamento
	f >> n;

	for (int i = 0;i < n;i++) {
		int apts, num;
		f >> apts;
		for (int j = 0;j < apts;j++) {
			Apartamento c;
			c.piso = i + 1;
			f >> c.apto;
			apartamentos[i].push_back(c);
		}
	}

}
```

Si lo que queremos es leer una línea de un fichero, sería muy similar, pero usaríamos *ifstream* en vez de *ofstream*, y, si la línea que leemos puede contener espacios (es lo habitual), usaremos *getline* en vez de >>, al igual que hacíamos con la entrada desde teclado:

```
//imprime el resument
void resumen() {
	cout << "Resumen" << endl;
	//variable para calcular el total
	int total = 0;
	for (int i = 0;i < n;i++) {
		cout << "Piso " << i + 1 <<" = " << apartamentos[i].size() <<endl;

		total += apartamentos[i].size();
	}
	cout << "total: " << total;

}
```

Podemos querer añadir al final de un fichero que ya existe, o modificar cualquier posición intermedia del fichero, o abrir un fichero de forma que podamos tanto leer de él como escribir en él. Para esas cosas, en vez de

usar *ofstream* o *ifstream* usaremos un tipo de fichero más genérico, el *fstream*, que nos permite indicar el modo de apertura (lectura o escritura, texto o binario, etc).

```
//imprime menu
void menu() {
	system("cls");
	cout << "1. Listar Ordenado del primer piso al ultimo " << endl;
	cout << "2. Listar Ordenado del ultimo al primero" << endl;
	cout << "3. Resumen Edificio" << endl;
	cout << "4. Salir" << endl;


}
```

son una estructura de control condicional, que permite definir múltiples casos que puede llegar a cumplir una variable o cualquiera, y qué acción tomar en cualquiera de estas situaciones. Incluso es posible determinar qué acción llevar a cabo en caso de no cumplir ninguna de las condiciones dadas

```
//metodo principal
int main() {
	//lee el archivo
	leerArchivo();

	int opt;
	//por siempre...
	while (true) {
		//imprime menu
		menu();
		cout << "Digite opcion: ";
		cin >> opt;
		//dependiendo de la opcion ingresada se accede a uno de los
siguientes casos, si el usuario digita una opcion invalida se termina el
programa
		switch (opt)
		{
		case 1:
			ListarAsc();
			break;
		case 2:
			ListarDesc();
			break;
		case 3:
			resumen();
			break;
		default:
			exit(0);
			break;
		}
		cout << endl;
		system("pause");
	}
	system("pause");
```

```
        return 0;
}
```

Los condicionales switch de hecho todos los condicionales en sí, son extremadamente útiles pues permiten definirle a nuestro software múltiples vías de ejecución contemplando así todas las posibilidades durante la ejecución. Me gustaría hacer una leve aclaración, el condicional switch encuentra su utilidad al momento de tener más de una posibilidad de valores para una variable cualquiera, evidentemente si nuestra variable solo puede adquirir un valor útil para nosotros, nuestra alternativa inmediata debería ser un if o un if-else, no un switch que resulta un poco más engorroso de escribir, sin embargo cuando tenemos varias posibilidades es mejor un switch que tener condicionales anidados o un condicional después de otro.