Guía de prácticas de Algoritmos en Sistemas Electrónicos
Programa de Ingeniería Electrónica
Escuela Colombiana de Ingeniería Julio Garavito

Ing. Alexander Pérez Ruiz MSc. PhD.

2019 – 1

1. Práctica 1

1.1. Objetivo

El objetivo de esta práctica, es desarrollar la habilidad de utilización de las herramientas de desarrollo en entorno linux de forma práctica con un proyecto que involucra lectura de parámetros enviados desde la consola.

1.2. Procedimiento

Se proporciona una máquina virtual basada en VirtualBox que tiene instalada todas las herramientas que se utilizarán durante el desarrollo del curso.

1. Verifique que la BIOS del computador tenga habilitado el soporte de virtualización.
2. Abrir el VirtualBox.
3. Crear una máquina nueva en linux Ubuntu de 64bits.
4. Seleccionar RAM de 4Gb.
5. Seleccionar disco duro proporcionado.
6. Ejecutar la máquina virtual.

1.3. Problema

Abrir la máquina virtual y ejecutar desde consola la llamada a las herramientas de compilación gcc y cmake para verificar su versión. Abrir el Qtcreator y cargar el proyecto que se encuentra en **/Documentos/Proyectos/ALSE_labs/Lab1**. Ejecutarlo desde el IDE y desde la consola.

2. Práctica 2

2.1. Objetivo

Adquirir destreza en el manejo de las herramientas de desarrollo tanto para crear proyectos como para editar código a través de QtCreator y reforzar las buenas prácticas de documentación con Doxygen.

2.2. Conceptos a utilizar

Se deben utilizar los conceptos de:

1. Estructuras.
2. Operadores.
3. Parámetros de una función (valor y referencia).
4. Parámetros de *main*.
5. Entrada y salida estándar *cin* y *cout*.
6. Caraster\u00edsticas de las directivas de preprocesador.

2.3. Problema

Crear una estructura de datos llamada complejo con una parte real y la otra imaginaria de tipo flotante. Crear un número complejo con los valores pasados como parámetros del *main* y calcular su norma y ángulo a través de dos funciones independientes con parámetros por referencia. Mostrar en la pantalla el número complejo creado en el formato $a + bj$ y su norma y ángulo utilizando la salida estándar.

2.4. Solución

3. Práctica 3

3.1. Objetivo

Desarrollar las habilidades lectura de archivos de texto plano como medio de entrada de datos y de utilización de vectores de la STL (*Standard Template Library*).

3.2. Conceptos a utilizar

Se deben utilizar los conceptos de:

1. Vector estandar.
2. Sobrecarga de operadores.
3. Iteradores.

3.3. Problema

Se debe desarrollar un programa de computador que sea capaz de leer un archivo de texto que contiene una columnas de números reales que corresponden de forma alternada a la parte real e imaginaria de números complejos. Estos números complejos deberán quedar almacenados en un vector de la *STL* para su utilización. Se debe calcular la norma de cada uno de ellos y almacenarlo en otro vector. Se debe calcular el complejo promedio de la lista. Como salida el programa deberá mostrar en la pantalla la norma de cada uno de los números complejos y el número complejo promedio. Se debe sobrecargar el operador de suma, el operador resta, el operador de comparación $<$ y el operador de flujo de salida $<<$ para mostrar los números en la forma $a + bj$.

Crear una función que reciba como parámetro el vector de complejos por referencia y lo ordene de menor a mayor de acuerdo al ángulo de cada complejo.

Bonus: Hacer el procesamiento del vector utilizando iteradores del vector.

3.4. Solución

ksjdflkajsldfk

4. Práctica 4

4.1. Objetivo

Conocer las de la OOP (*Object Oriented Programming*) y sus principios de encapsulamiento, herencia y polimorfismo. Conocer las herramientas de diseño en UML (*Unified Modeling Language*) y adquirir habilidad en el diseño e implementación de clases.

4.2. Conceptos a utilizar

Se deben utilizar los conceptos de:

1. Clases.
2. Encapsulamiento.
3. Objetos.
4. Constructores.
5. Destructores.
6. Funciones miembro y no miembro.
7. Sobrecarga de operadores.

4.3. Problema

Implementar la clase **Complejo** con atributos *float*. Esta debe tener constructor por omisión, constructor con parámetros r e i, constructor por copia y el destructor. Se debe sobrecargar el operador de suma, el operador de resta, el operador de comparación $<$ y el de igualdad $==$ como funciones miembro. Desarrollar un programa que muestre el uso de los diferentes constructores y los operadores implementados. *Bonus:* Dibujar el diagrama de la clase **Complejo**.

4.4. Solución

```
int main(int argc, char** argv){
    cout << a.b;
}
```

5. Práctica 5

5.1. Objetivo

Entender y aplicar los principios de encapsulamiento de la OOP. Conocer las generalidades del lenguaje de modelado unificado **UML** para la creación de diagramas de clases y utilizar una herramienta para realizarlas.

5.2. Conceptos a utilizar

Se deben utilizar los conceptos de:

1. Diagramas de clases.
2. Tipo de acceso.
3. Atributos.
4. Métodos. Sobrecarga de funciones.
5. Coherencia Diagrama – Código.

5.3. Problema

Utilizando el ciclo de desarrollo de programas con las herramientas descritas, se debe crear el diagrama con las clases **Circulo**, **Cuadrado**, **Triangulo** y **Pentagono** en UML usando StarUML y posteriormente generar el código fuente desde él. Cada uno de las clases debe permitir cambiar el centro de la figura y sus características básicas como el radio en el círculo o el lado en el cuadrado sin violar el pricipio de encapsulamiento. Se debe implementar una función *area* y una función *perimetro* para cada una de ellas.

Con estas clases, crear un objeto de cada una de las clases, aumentar o disminuir el valor de sus características básicas (p.e. el radio en un círculo) y mostrar en la salida estándar de qué tipo es, sus características básicas, su perímetro y su área.

5.4. Solución

6. Práctica 6

6.1. Objetivo

Entender y aplicar el principio de herencia de la OOP. Desarrollar la habilidad de establecer relaciones de espacialización entre las clases que componen un programa a través del uso de la herencia.

6.2. Conceptos a utilizar

Se deben utilizar los conceptos de:

1. Diagramas de clases.
2. Herencia.
3. Punteros.
4. *std::vector<>*.
5. Funciones virtuales.
6. Memoria dinámica (*new* y *delete*).

6.3. Problema

A partir del análisis de las clases generadas para el problema anterior, crear una jerarquía de clases a partir de una clase padre **Geometrica** de la cual se deben heredar las clases **Circulo**, **Cuadrado**, **Triangulo** y **Pentagono**. Realizar este diseño en UML usando StarUML y posteriormente generar el código fuente desde él. Cada uno de las clases debe permitir cambiar el centro de la figura y sus características básicas como el radio en el círculo o el lado en el cuadrado sin violar el pricipio de encapsulamiento. Se debe implementar una función *area* y una función *perimetro* para cada una de ellas.

Se debe crear con estas clases un programa que solicite al usuario por consola qué figura geométrica desea crear y solicitarle los valores para las variables necesarias (p.e. el radio en un círculo) y repetir esta acción hasta que el usuario decida no crear más. Estas figuras deben quedar almacenadas en un *std::vector*.

Al terminar la creación de figuras, se debe mostrar en la salida estándar un listado de las figuras ingresadas al vector y mostrar sus características básicas, su perímetro y su área.

6.4. Solución

7. Práctica 7

7.1. Objetivo

Recordar el procesamiento de cadenas de caracteres e incorporar el tipo de dato *string*.

7.2. Conceptos a utilizar

Se deben utilizar los conceptos de:

1. Enumeraciones.
2. *std::string*.
3. *substring*.
4. *find*.

7.3. Problema

A partir de la jerarquía de clases del problema anterior, crear un programa que permita la lectura de un archivo de texto plano que contiene en cada línea el identificador de la figura geométrica que se debe crear y la lista de parámetros de configuración de acuerdo a lo definido para cada una de ellas. Por ejemplo para un círculo o un triángulo se tendría:

```
1 rd xc yx angulo
3 base altura xc yc angulo
```

Se debe definir una enumeración para evitar números mágicos dentro del programa. El programa leerá todas las líneas del archivo y creará las figuras geométricas de acuerdo a lo allí indicado. Posteriormente se debe mostrar en la salida estándar un listado de las figuras creadas y mostrar sus características básicas, su perímetro y su área. *Bonus:* Procesar el texto en una función *setParametros* y utilizarla para crear un constructor con parámetro *string*.

7.4. Solución