Instructivo: compilación, pruebas y depuración

1. Objetivos

1. Realizar un acercamiento inicial al proceso de compilación y depuración de programas en consola o línea de comandos.
2. Definir un plan de pruebas simple para los desarrollos a realizar.

2. Compilación g++

La compilación de un programa en **C++** a través de la línea de comandos se realiza con el compilador estándar **g++** (que será usado en este curso para evaluar y calificar todas las entregas). Para realizar el proceso de compilación, se abre una consola (terminal, línea de comandos), se ubica en la carpeta que contiene el código fuente y se siguen los siguientes pasos:

1. **Compilación:** de todo el código fuente compilable (ÚNICAMENTE LOS ARCHIVOS CON EXTENSIONES *.c, *.cpp, *.cxx)

   ```
   $g++ -std=c++11 -c *.c *.cxx *.cpp
   ```

   Las extensiones se incluyen de acuerdo a los archivos que se desea compilar. El * sirve para agrupar varios archivos con la misma extensión (y así no tener que listarlos uno por uno).

   Ejemplo: para un archivo de código fuente llamado programa.cxx, la siguiente sería la línea que permitiría su compilación:

   ```
   $g++ -std=c++11 -c programa.cxx
   ```

2. **Encadenamiento:** de todo el código de bajo nivel en el archivo ejecutable

   ```
   $g++ -std=c++11 -o nombre_de_mi_programa *.o
   ```

   nombre_de_mi_programa es cualquier nombre que el usuario quiera ponerle al archivo ejecutable del programa. Los archivos .o son los que se generan con el comando anterior de compilación.

   Ejemplo: luego de compilar el archivo de código fuente programa.cxx, se genera el archivo programa.o. La siguiente sería la línea que permite su encadenamiento en un ejecutable llamado mi_programa:

   ```
   $g++ -std=c++11 -o mi_programa programa.o
   ```

   Nota: Estos dos pasos (compilación y encadenamiento) pueden abreviarse en un sólo comando:

   ```
   $g++ -std=c++11 -o nombre_de_mi_programa *.c *.cxx *.cpp
   ```

   Ejemplo: la siguiente sería la línea que compila el archivo de código fuente programa.cxx y de una vez lo encadena en el archivo ejecutable mi_programa:

   ```
   $g++ -std=c++11 -o mi_programa programa.cxx
   ```

3. **Ejecución:** del programa ejecutable anteriormente generado

   ```
   $./nombre_de_mi_programa
   ```

   Ejemplo: la siguiente sería la línea que permite ejecutar el programa mi_programa :

   ```
   $./mi_programa
   ```

ATENCIÓN: Los archivos de encabezados (*.h, *.hpp, *.hxx) NO SE COMPILAN, se incluyen en otros archivos (encabezados o código fuente). Así mismo, los archivos de código fuente (*.c, *.cpp, *.cxx) NO SE INCLUYEN, se compilan.

3. Plan de pruebas

Al desarrollar un programa, resulta necesario realizar pruebas al código para garantizar su correcto funcionamiento bajo diferentes circunstancias (diferentes valores de entrada). En proyectos de desarrollo de gran envergadura, lo anterior se conoce como el plan de pruebas, y puede llegar a ser tan complejo que muchas veces se realiza de forma automática. En nuestro caso, definiremos un plan de pruebas como el conjunto de verificaciones realizadas a las funciones u operaciones principales del programa que desarrollemos. Ese conjunto de verificaciones incluirá la comparación de resultados esperados contra resultados obtenidos en (al menos) 3 casos diferentes para cada una de las funciones u operaciones. Como ejemplo, considere la siguiente función simple:

```
unsigned int suma ( int a, int b ) {
      return a + b;
}
```

El siguiente sería un posible plan de pruebas para verificar los resultados de la función. Se describe cada uno de los casos a probar, se indican los valores de entrada, se calcula manualmente (sin usar el código fuente) el resultado esperado y se transcribe el resultado obtenido luego de ejecutar la función en un programa:

Plan de pruebas: función suma			
Descripción de caso	Valores de entrada	Resultado esperado	Resultado obtenido
1: números positivos	a = 15, b = 32	47	47
2: un número 0	a = 0, b = 32	32	32
3: números negativos	a = 15, b = -32	-17	4294967279

Se puede evidenciar que las dos primeras pruebas son exitosas, pero la tercera tiene una discrepancia en el resultado. En este caso, se debe a que el tipo de dato que retorna la función es un entero sin signo, es decir, no puede ser un número negativo. Para garantizar que la función retorna el resultado esperado, sólo hace falta cambiar el tipo de dato del retorno.

4. Depuración con gdb

En el desarrollo de programas pueden llegar a cometerse diferentes errores en el código fuente. La mayoría de los problemas son detectados por el compilador, quien informa errores de sintaxis, escritura y uso del lenguaje de programación. Sin embargo, existe otro tipo de errores que ocurren en tiempo de ejecución, que suelen tener que ver con el manejo de memoria y elementos dinámicos y que llevan a la terminación anticipada del programa. Detectar y corregir estos errores implica realizar un proceso conocido como depuración del código. Hay muchas formas de realizar esta depuración, como imprimir mensajes bandera en puntos específicos del código, hacer una prueba de escritorio muy juiciosa o, la más eficaz, utilizar un programa depurador.

En este caso, utilizaremos gdb, depurador en línea de comandos para C++. Utilizando el programa ya compilado, éste se ejecuta dentro de **gdb** para detectar la línea de código donde ocurre la falla, en qué función o método, y quién ha hecho el llamado a esa función o método. Para realizar el proceso de depuración, se siguen los siguientes pasos:

1. **Compilación y encadenamiento:** de todo el código fuente compilable, indicando que el ejecutable se utilizará dentro del depurador

   ```
   $g++ -std=c++11 -g -o nombre_de_mi_programa *.c *.cxx *.cpp
   ```

   Se agrega la opción -g para indicar que el ejecutable se depurará con **gdb**.

2. **Ejecución del depurador:** sobre el archivo ejecutable del programa **gdb** nombre_de_mi_programa. Con esto se inicia el programa depurador, y queda listo para trabajar sobre el ejecutable indicado.

3. **Ejecución del programa:** dentro del entorno del depurador **run**
   El programa se ejecuta de la misma forma que en la línea de comandos, y si existe algún error de ejecución, el depurador indicará información relevante al respecto.

4. **Revisión:** de la secuencia de llamados a funciones o métodos dentro del programa que llevaron al error **backtrace**
   El depurador indica información acerca de los llamados previos al error dentro del programa.

5. **Salida:** del depurador, una vez se ha encontrado la línea de código causante del error
   Quit

5. ACTIVIDADES

1. Compile y ejecute directamente el programa "*excercise1.cpp*" desde la línea de comando siguiendo las instrucciones del numeral 2 de este manual. Tome una captura de pantalla a la terminal con el proceso realizado y la respuesta del programa.

2. Compile y ejecute por medio del depurador el programa "*excercise2.cxx*" desde la línea de comando siguiendo las instrucciones del numeral 2 y 4 de este manual. Tome una captura de pantalla a la terminal con el proceso realizado y la respuesta del programa para una combinación de entradas cualquiera.

3. Para el programa "*excercise2.cxx*" desarrolle el siguiente plan de pruebas para cada función y:

- complete las tablas de pruebas
- responda: ¿Cuáles funciones presentan errores en sus resultados?

Plan de pruebas: función Perímetro del rectángulo			
Descripción de caso	**Valores de entrada**	**Resultado esperado**	**Resultado obtenido**
1: Alto como el doble de Ancho	Ancho = 2, Alto = 4		
2: Alto igual a Ancho	Ancho = 3, Alto = 3		
3: un numero en cero	Ancho = 5, Alto = 0		

Plan de pruebas: función Área del rectángulo			
Descripción de caso	**Valores de entrada**	**Resultado esperado**	**Resultado obtenido**
1: Alto como el doble de Ancho	Ancho = 2, Alto = 4		
2: Alto igual a Ancho	Ancho = 3, Alto = 3		
3: un numero en cero	Ancho = 5, Alto = 0		

Plan de pruebas: función Distancia del rectángulo al origen			
Descripción de caso	**Valores de entrada**	**Resultado esperado**	**Resultado obtenido**
1: números positivos	x = 15, y = 32	35.34	
2: un número 0	x = 0, y = 32	32	
3: números iguales	x = 15, x = 15	21.21	

6. Entrega:

- En un documento agregue todas las capturas de pantalla de las actividades 1 y 2, junto con las tablas y respuestas de la actividad 3
- Adicione una captura de pantalla del directorio donde almacenó los códigos fuente del taller, de tal manera que se puedan evidenciar todos los archivos resultantes del proceso de compilación. El archivo de entrega debe ser cargado en PDF en el repositorio, en una carpeta debidamente identificada (Taller01).
- La evaluación del taller tendrá la siguiente escala para cada uno de los puntos:
  - **Excelente (5.0/5.0):** El estudiante presenta análisis y datos organizados (información) que le permiten llegar a una conclusión coherente sobre lo requerido. Además, el estudiante es capaz de explicar la coherencia de los resultados.
  - **Bueno (4.0/5.0):** El estudiante tiene una idea de cómo comparar los e interpretar los resultados, pero falla en el formalismo. Además, no presenta datos organizados o no hizo un análisis de la coherencia de los resultados.
  - **Regular (3.0/5.0):** El estudiante logra esbozar una idea de comparación e interpretación de los resultados, pero falla en presentarla de manera clara y contundente. No presenta datos organizados ni su correspondiente análisis.
  - **Malo (1.5/5.0):** El estudiante no logra entender las instrucciones presentadas, y su análisis es vago o ambiguo.
  - **No entregó (0.0/5.0):** No entregó el archivo del informe, o el archivo entregado no está en formato PDF