



Pontificia Universidad Javeriana  
Departamento de Ingeniería de Sistemas  
Estructuras de Datos  
Ejercicio inicial: Compilación, pruebas y depuración,  
2024-10

## 1 Objetivos

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

## 2 Compilación con 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

esto 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.

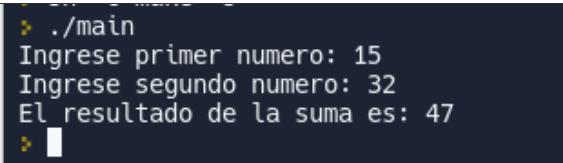
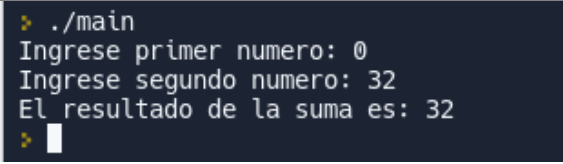
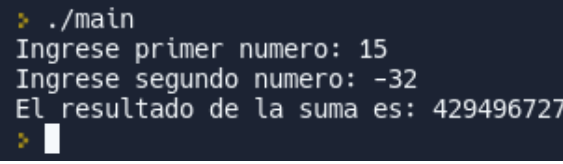
Como ejemplo, considere la siguiente función simple:

```
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

También se suele adjuntar evidencias de la ejecución de cada uno de los casos de prueba, es decir, pantallazos que muestran la ejecución del programa y el resultado obtenido con las entradas indicadas para cada caso:

Descripción de caso	Evidencia de ejecución
1: números positivos	
2: un número 0	
3: números negativos	

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 en el código y repetir el caso de prueba para verificar que ahora sí el resultado esperado coincide con el resultado obtenido.

## 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 Desarrollo del ejercicio

El desarrollo del ejercicio consistirá en generar un informe escrito (en formato PDF únicamente) con el resultado de la ejecución de las diferentes tareas aquí propuestas. Junto con este enunciado se entrega un archivo comprimido que contiene el conjunto de programas a usar. Descomprima el archivo en una ubicación adecuada (fácil de acceder por línea de comandos) y utilizando la consola o línea de comandos ubíquese en la carpeta que acaba de descomprimir. A continuación se describe qué se puede hacer con estos archivos.

### 5.1 Compilación

Para la primera compilación se utilizarán únicamente los archivos `exercise1.cxx`, `rectangle.h` y `rectangle.cxx`. Revise el código fuente en los archivos y luego realice las siguientes tareas:

1. Escriba y ejecute en la consola el comando completo que permite compilar y encadenar estos archivos sin errores.
2. Escriba y ejecute en la consola el comando que permite ejecutar el programa anteriormente compilado.

### 5.2 Plan de pruebas

Una vez compilado el programa, ejecútelo un par de veces para familiarizarse con el objetivo del programa, que también pudo haber deducido a partir de la lectura de su código fuente. El objetivo ahora será probar la funcionalidad ofrecida por el programa (sin hacer modificaciones al código entregado).

1. Defina un plan de pruebas (siguiendo estrictamente el formato ya explicado) para las operaciones “perímetro del rectángulo” y “área del rectángulo” en el programa (un plan de pruebas para cada operación). Defina mínimo 3 casos de prueba diferentes en cada plan, incluyendo valores nulos y/o negativos.
2. Aplique los planes de pruebas (cada uno con tres casos) anteriormente definidos sobre el programa compilado, y complete adecuadamente los resultados obtenidos en el formato.
3. ¿Las pruebas fueron exitosas? En caso afirmativo, reporte con evidencias (pantallazos) las ejecuciones exitosas de las pruebas. En caso negativo, identifique y reporte las modificaciones que requiere el código fuente para que los resultados sean aceptables. Verifique que con las modificaciones sugeridas las pruebas son finalmente exitosas, usando los mismos casos de prueba ya definidos en el formato y generando las evidencias adecuadas.

### 5.3 Depuración

Para la segunda compilación se utilizará únicamente el archivo `exercise2.cpp`. Revise el código fuente y realice las siguientes tareas:

1. Escriba y ejecute en la consola el comando completo que permite compilar y encadenar el archivo sin errores.
2. Escriba y ejecute en la consola el comando que permite ejecutar el programa anteriormente compilado.

¿El programa corre sin errores? Si es así, hasta aquí llega el ejercicio. En caso contrario, es necesario depurar el código en busca de la fuente del error. Para eso, siga los pasos indicados para la depuración con `gdb`, teniendo en cuenta:

3. ¿Cuál es el comando completo a escribir en la consola que permite compilar y encadenar el archivo para su uso dentro del depurador?
4. Al depurar el programa dentro de `gdb`, ¿cuál es el error que se genera y en qué parte del código?
5. ¿De qué forma puede corregirse el problema para que el programa se ejecute satisfactoriamente (sin errores)?

## 6 Evaluación

La entrega del informe en PDF se hará a través de la respectiva asignación de BrightSpace, antes del miércoles 31 de enero a las 9:00am. La evaluación del ejercicio tendrá la siguiente escala para cada uno de los puntos:

- **Excelente (5.0/5.0):** El estudiante presenta respuestas y evidencias de la ejecución completa de las tareas propuestas.
- **Bueno (4.0/5.0):** El estudiante presenta algunas respuestas y evidencias de la ejecución de las tareas propuestas, fallando sólo en pequeños detalles de precisión.
- **Regular (3.0/5.0):** El estudiante no logra presentar de manera clara y contundente las respuestas y evidencias de la ejecución completa de las tareas propuestas.
- **Malo (1.5/5.0):** El estudiante no logra entender las instrucciones presentadas, y sus respuestas y/o evidencias son insuficientes para demostrar la ejecución de las tareas propuestas.
- **No entregó (0.0/5.0):** No entregó el archivo del informe, o el archivo entregado no está en formato PDF.