Taller - Compilación, pruebas y depuración

William Gómez Roa

wa.gomez@javeriana.edu.co
Bioingeniería y Ciencia de Datos

1 Ejercicio 1

1.1

Con el siguiente comando se generó la compilación de los archivos pedidos.

```
g++ -std=c++14 -o main14 exercise1.cxx rectangle.cxx
```

1.2

Luego, el programa se ejecutó correctamente en la terminal utilizando este comando.

```
./ main14
```

2 Ejercicio 2

2.1 Plan de pruebas

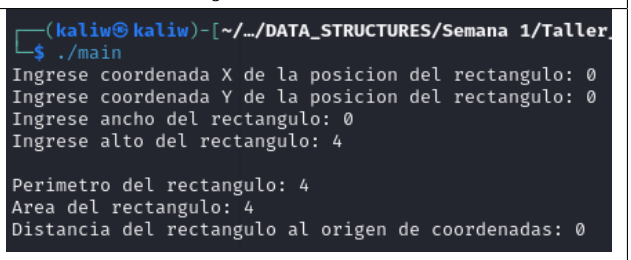
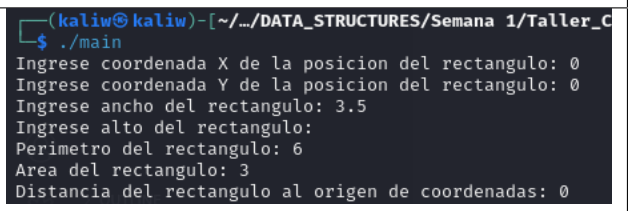
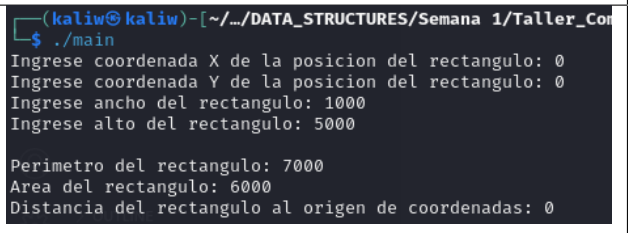
Para provar la efectividad de las funciones que calculan el área y el perímetro del rectangulo, se realizó un plan de pruebas con 3 casos diferentes. En las siguientes tablas se consignan los resultados esperados y obtenidos para estas 2 funciones. Se puede ver que los resultados obtenidos en su mayoría fallarón para ambas funciones (exeptuando el perimetro del rectangulo con ancho=0 y alto $= 0$).

Plan de pruebas: función perimeterRect			
Descripción del caso	**Valores de entrada**	**Resultado esperado**	**Resultado obtenido**
1: Rectángulo con valores nulos	Ancho = 0, Alto = 4	4	4
2: Rectángulo con valores decimales	Ancho = 3.5, Alto = 6.2	19.4	6
3: Rectángulo con valores grandes	Ancho = 1000, Alto = 500	3000	7000

Plan de pruebas: función areaRect			
Descripción del caso	**Valores de entrada**	**Resultado esperado**	**Resultado obtenido**
1: Rectángulo con valores nulos	Ancho = 0, Alto = 4	0	8
2: Rectángulo con valores decimales	Ancho = 3.5, Alto = 6.2	21.7	3
3: Rectángulo con valores grandes	Ancho = 1000.0, Alto = 500.0	500000	1500

2.2 Evidencias

A continuación se muestran evidencias del mal funcionamiento del codigo antes de realizar las correciones pertinentes.

Evidencia: función perimeterRect	
Descripción del caso	**Evidencias de ejecución**
Caso 1: Rectángulo con valores nulos	
Caso 2: Rectángulo con valores decimales	
Caso 3: Rectángulo con valores grandes	

2.3 Corrección del código

Para corregir el codigo y pasar todas las pruebas satisfactoriamente es necesario realizar unas correciones.

Rectangulos con valores nulos

La función para calcular el perimetro es incorrecta, la forma corregida es la siguiente:

```
perim = 2.0 * (rect.width + rect.height);
```

La función para calcular el area es incorrecta, la forma corregida es la siguiente:

```
area = rect.width * rect.height;
```

Rectangulos con valores decimales

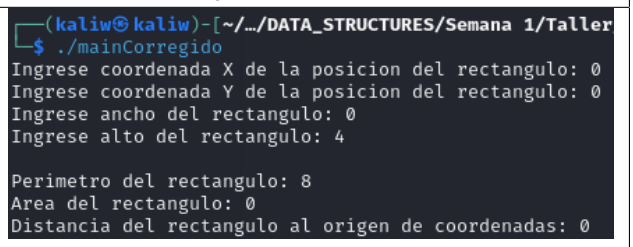
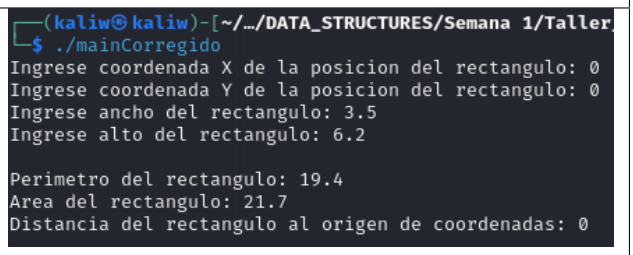
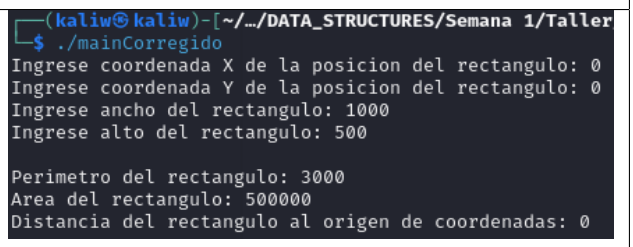
La declaración del alto y ancho del rectangulo son valores **int**, para que funcione con decimales, este tipo de dato se debe modificar en su declaración:

```
struct Rectangle {
  float posX;
  float posY;
  float width;
  float height;
};
```

Rectangulos con valores grandes

Con los cambios anteriores, esta prueba funciona correctamente.

A continuación se adjuntan evidencias del codigo corregido para cada plan de pruebas.

Evidencia: función perimeterRect	
Descripción del caso	**Evidencias de ejecución**
Caso 1: Rectángulo con valores nulos	
Caso 2: Rectángulo con valores decimales	
Caso 3: Rectángulo con valores grandes	

3 Ejercicio 3

3.1

El codigo pudo ser compilado sin errores con el siguiente comando:

```
g++ -std=c++14 -o  exercise2 exercise2.cpp
```

3.2

Así se ejecutó el programa:

```
./exercise2
```

Sin embargo, el codigo presenta errores en el tiempo de ejecución y termina antes de finalizar el programa con el siguiente error:

```
┌──(kaliw㉿kaliw)-[~/…/DATA_STRUCTURES/Semana 1
└─$ ./exercise2
Creating Node, 1 are in existence right now
Creating Node, 2 are in existence right now
Creating Node, 3 are in existence right now
Creating Node, 4 are in existence right now
The fully created list is:
4
3
2
1

Now removing elements:
Creating Node, 5 are in existence right now
Destroying Node, 4 are in existence right now
4
3
2
1

zsh: segmentation fault  ./exercise2
```

3.3

Para compilar y encadenar el archivo para su uso dentro del depurador se utilizó el siguiente comando:

```
g++ -std=c++14 -g -o  exercise2_gdb exercise2.cpp
```

3.4

El error que se genera, como se observa en la imagen siguiente, ocurrío en la linea 32 del archivo exercise2.cpp, e indica que el puntero "this" es nulo (0x0) por lo que retornando el valor *next_* de un puntero nulo y se genera el error de *Segmentation fault*.

Además utilizando *backtrace* podemos ver que el error ocurre el la función *main* exactamente en la linea 131 . Tambien nos muestra que esta fallando la función *LinkedList¡int¿::remove* en la línea 88.

En conclusión es un problema con el manejo de la memoria de los datos de la *pila*.

```
(gdb) run
Starting program: /home/kaliw/GITHUB/DATA_STRUCTURES/Semana 1/Taller_Compilation/try3/exercise2_gdb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Creating Node, 1 are in existence right now
Creating Node, 2 are in existence right now
Creating Node, 3 are in existence right now
Creating Node, 4 are in existence right now
The fully created list is:
4
3
2
1

Now removing elements:
Creating Node, 5 are in existence right now
Destroying Node, 4 are in existence right now
4
3
2
1


Program received signal SIGSEGV, Segmentation fault.
0×0000555555555726 in Node<int>::next (this=0×0) at exercise2.cpp:32
32          return next_;
(gdb)
(gdb) backtrace
#0  0×0000555555555726 in Node<int>::next (this=0×0) at exercise2.cpp:32
#1  0×000055555555561d in LinkedList<int>::remove (this=0×55555556b2b0, item_to_remove=@0×7fffffffdccc: 1) at exercise2.cpp:88
#2  0×00005555555552ea in main (argc=1, argv=0×7fffffffde08) at exercise2.cpp:131
(gdb)
```

3.5

Para corregir el codigo y hacer que este se ejecute sin errores es necesario modificar lo siguiente:

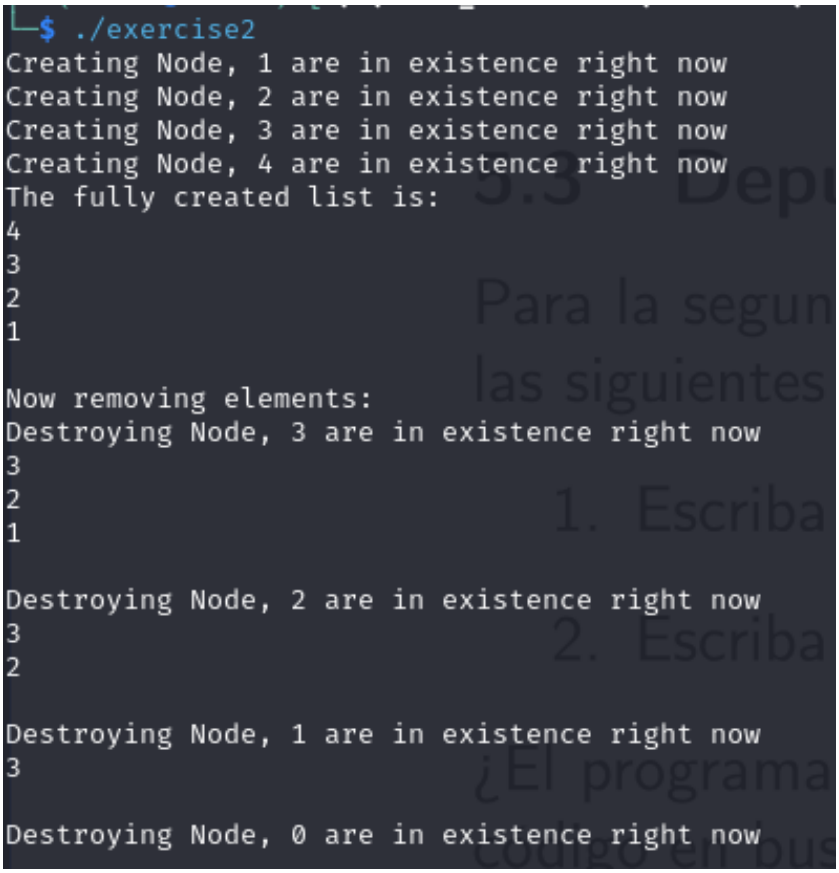
```
if (marker->value() == item_to_remove) {
        if (temp == 0) { // marker is the first element in the list
          if (marker->next() == 0) {
            head_ = 0;
            delete marker; // marker is the only element in the list
            marker = 0;
          } else {
            head_ = new Node<T>(marker->value(), marker->next());
            delete marker;
            marker = 0;
          }
          return 0;
        } else {
          temp->next (marker->next());
          delete temp;
          temp = 0;
          return 0;
        }
      }
```

Por:

```
 if (marker->value() == item_to_remove) {
    if (temp == 0) { // marker is the first element in the list
        if (marker->next() == 0) {
            head_ = 0;
        } else {
            head_ = marker->next();
        }
        delete marker;
        return 0;
    } else {
        temp->next(marker->next());
        delete marker;
        return 0;
    }
}
```

El problema es que estabamos creando un nuevo nodo con el mismo valor y el puntero *next* de la variable *marker* y posteriomente, en la siguiente linea de codigo, borramos *marker*, dejando así el nuevo nodo *head_* apuntando a un lugar erroneo.

Por lo tanto, para corregir el codigo se actualizó *head_* sin crear un nuevo nodo y el programa funcionó correctamente, como se puede ver a continuación:


└─$ ./exercise2
Creating Node, 1 are in existence right now
Creating Node, 2 are in existence right now
Creating Node, 3 are in existence right now
Creating Node, 4 are in existence right now
The fully created list is:
4
3
2
1
Now removing elements:
Destroying Node, 3 are in existence right now
3
2
1
Destroying Node, 2 are in existence right now
3
2
Destroying Node, 1 are in existence right now
3
Destroying Node, 0 are in existence right now