

# *U N I V E R S I D A D N A C I O N A L D E S A N A G U S T Í N*

*FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS*

*ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN*



## ***Ciencia de la Computación***

*Estudiante: Rommel Mario Ccorahua Lozano*

*Profesor: Álvaro Henry Mamani*

*Aliaga*

*Grupo: "A"*

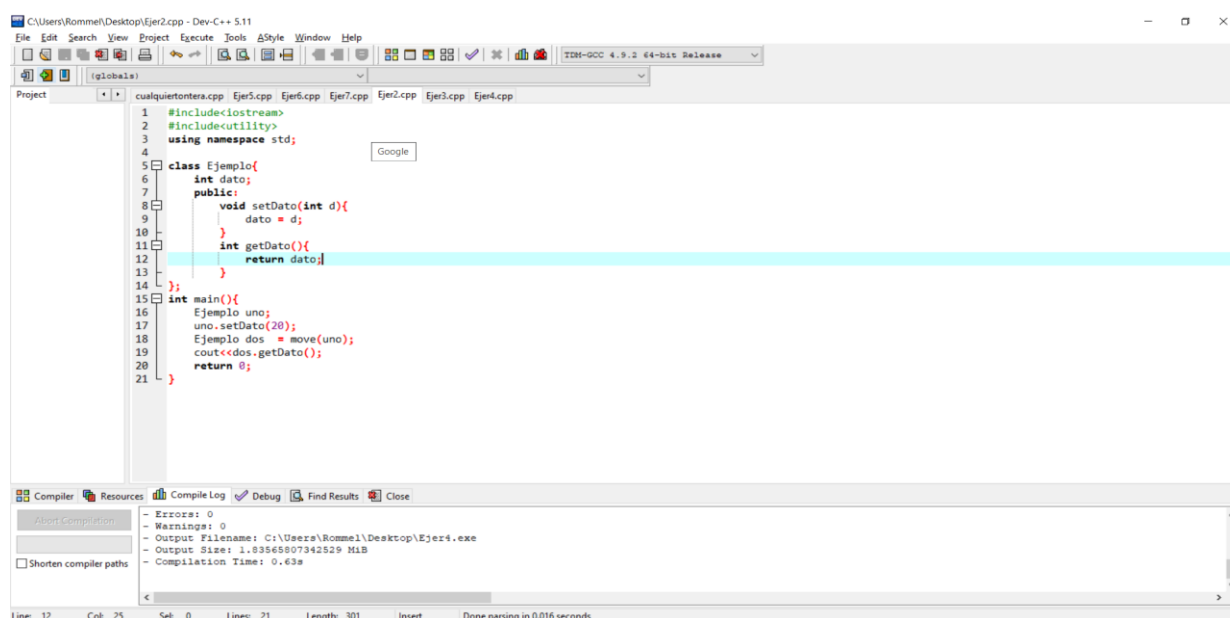
## 1. Investigar y describir sobre Move Semantics, lvalue y rvalue

**MOVE SEMANTICS:** Proporciona una forma de “mover” el contenido de los objetos entre los objetos mismos en vez de copiarlos. Permite que un objeto, bajo ciertas condiciones, tome posesión de los recursos externos de otro objeto.

**LVALUE:** Se llama así porque los valores pueden aparecer en el lado izquierdo de una expresión de asignación. Es una expresión cuya dirección de memoria puede ser manipulada. También cualquier expresión que retorne una referencia lvalue (como en el caso de operaciones de indexación como `lista[indice]` u operaciones de indirección como `*ptr`). Cualquier cosa a la que pueda hacer asignaciones es un **lvalue**.

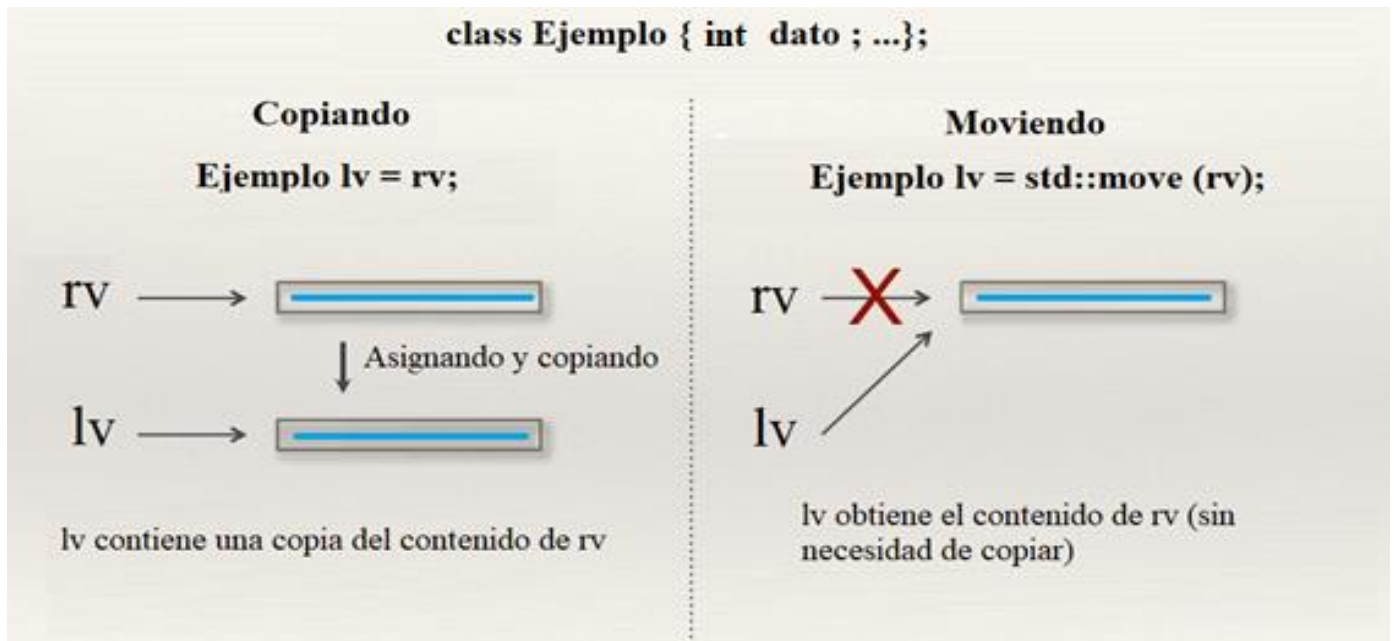
**RVALUE:** Se llama así porque los valores pueden aparecer en el lado derecho de una expresión de asignación. Es un valor que carece de nombre que no persistirá después del final de dicha expresión. Es un valor que no está asociado con ningún objeto. No es posible obtener la dirección en memoria de un rvalue.

## 2. Implementar y explicar un ejemplo explicando Move Semantics, lvalue y rvalue



En este ejemplo se declara una clase llamada *Ejemplo*, la cual tiene un único dato miembro llamado **dato** y sus respectivas funciones: setter y getter, luego se declaran dos objetos de tipo *Ejemplo*, al primero se le establece el valor de su **dato**, y luego se hace uso de `std::move()` para convertir el objeto **rv** a una referencia rvalue.

Lo que tomaría más tiempo y recursos asignando y copiando los datos, se puede ahorrar haciendo uso de Move Semantics, “**otorgando la propiedad de un objeto**” con `std::move()` sin necesidad de copiar algún dato. Más detalles son dados en la siguiente imagen.



### 3. Investigar y describir sobre referencia lvalue y referencia rvalue

#### Referencia a LVALUE:

La referencia a LVALUE es la que más común que ya se estudió con anterioridad. Consiste en declarar un **alias** o una **etiqueta** de un objeto existente.

Las referencias a un lvalue se deben inicializar al momento de declararse puesto que no son en sí una variable sino más bien un modo distinto de llamar a una variable y se declaran poniendo el operador & después del tipo de dato.

La declaración **t & n** declara una referencia a un lvalue con nombre “n” del tipo de dato “t”.

Como las referencias no son objetos, no existen matrices de referencias, ni existen punteros a las referencias y ni referencias a referencias.

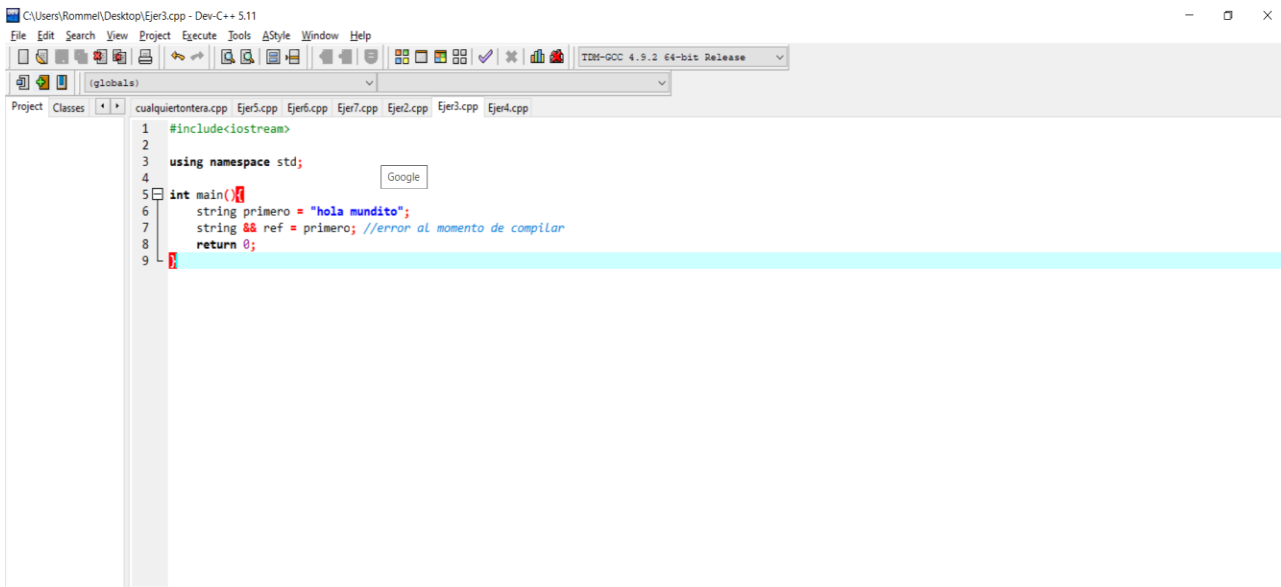
#### Referencia a RVALUE:

Las referencias a RVALUE se pueden usar para extender la vida útil de los objetos temporales. Para hacer uso de la referencia “rvalue” se debe colocar en la declaración dos ampersand “&&” después del tipo de dato seguido del nombre de la referencia.

La declaración **S&& D** declara a D como una referencia a un rvalue del tipo S

Su funcionalidad es referenciar objetos temporales, lo cual está intrínsecamente ligado a la semántica de movimiento.

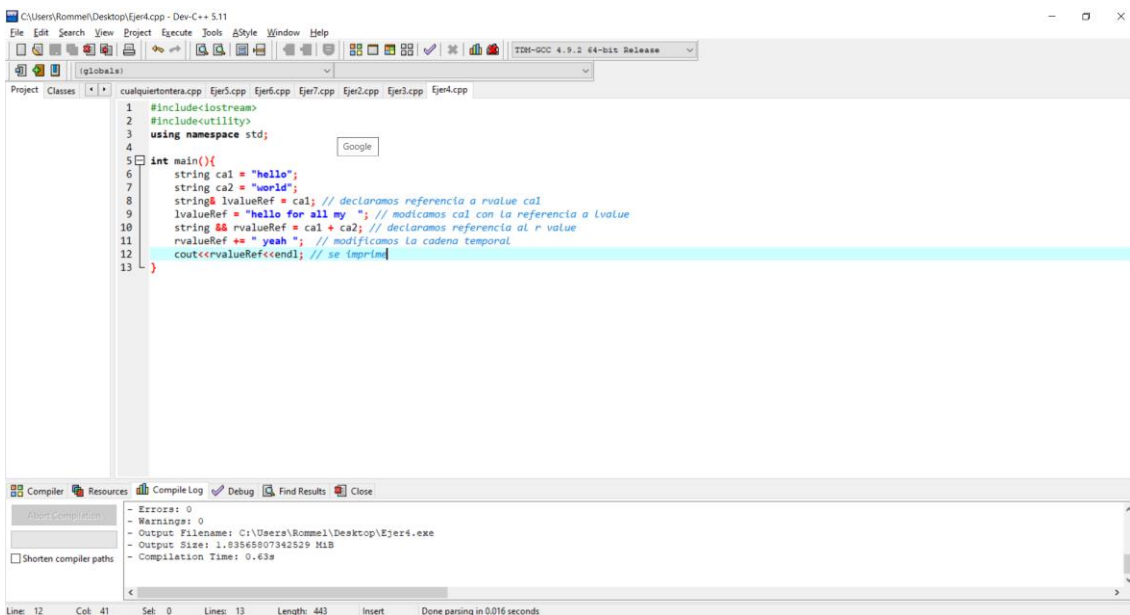
- Una referencia rvalue no puede enlazarse con un lvalue. De modo que, el código siguiente conduciría a un error de compilación:



#### 4. Implementar y explicar un ejemplo explicando referencia a lvalue y referencia a rvalue

En este ejemplo se hace uso de ambas referencias, primero se declara dos variables de tipo string: “Hola” y “mundo”. Luego se declara una referencia a lvalue llamada **lValueRef**, que referencia al objeto de tipo string **s1**, mediante esta referencia se modifica el valor del string **s1**.

Luego se declara una referencia a un rvalue llamada **rValueRef**, el cual referencia la siguiente expresión: **s1 + s2**, que representa la concatenación de dos cadenas el cual obviamente es un rvalue, luego mediante la referencia se puede modificar este rvalue. Gracias a esta referencia se puede realizar diversas acciones sobre este tipo de expresiones rvalue como modificarlas o imprimirlas.



#### 5. Investigar, describir e implementar un ejemplo usando std::move

En C ++ 11, **std :: move** es una función de la biblioteca estándar que tiene un único propósito: convertir su argumento en un valor rvalue. Podemos pasar un valor lvalue a **std :: move**, y devolverá una referencia de valor

*rvalue. std::move se define en el encabezado de la utilidad. Al usar **move** estamos robándole el valor de un lvalue despojándolo de este y dejándolo en un estado indefinido por lo que es una buena idea dejar siempre los objetos robados en un estado bien definido. Idealmente, este debería ser un "estado nulo", donde el objeto se restablece a su estado no iniciado o cero. Ahora se puede hablar sobre por qué: con std::move, el objeto que se está robando puede que no sea temporal después de todo. El usuario puede querer reutilizar este objeto (el cual ahora vacío) nuevamente, o probarlo de alguna manera, y puede planificar en consecuencia.*

```

1 #include<iostream>
2 #include<utility>
3
4 using namespace std;
5
6 void swap(int &a, int &b){ // uso de move para intercambio de valores en el metodo swap
7     int temporal = move(a); // ahora el valor de a va para temporal, a queda vacío
8     a = move(b); // a el cual esta vacío recibe el valor de b, por lo que se queda vacío
9     b = move(temporal); // b recibe el valor de temporal, el cual ahora se queda vacío y morira al terminar el metodo
10 }
11
12 int main(){
13     int a(10);
14     int b(20);
15     cout<<"Valor de a : "<<a<<endl;
16     cout<<"Valor de b : "<<b<<endl;
17     swap(a,b);
18     cout<<"\nValor de a despues de intercambiar : "<<a<<endl;
19     cout<<"Valor de b despues de intercambiar : "<<b<<endl;
20     return 0;
21 }

```

Compiler: 0 Errors, 0 Warnings. Output File: C:\Users\Rommel\Desktop\Ejer5.exe. Compilation Time: 0.86s.

## 6. Investigar, describir e implementar un ejemplo usando el constructor move (move constructor)

*Un constructor de movimientos se ejecuta solo cuando construyes un objeto. Un operador de asignación de movimiento se ejecuta en un objeto previamente construido. Es exactamente el mismo escenario que en el caso de copia. Si no se declara explícitamente, el compilador las generará. El move constructor se invoca cuando se da la inicialización y cuando se pasa como argumento de una función*

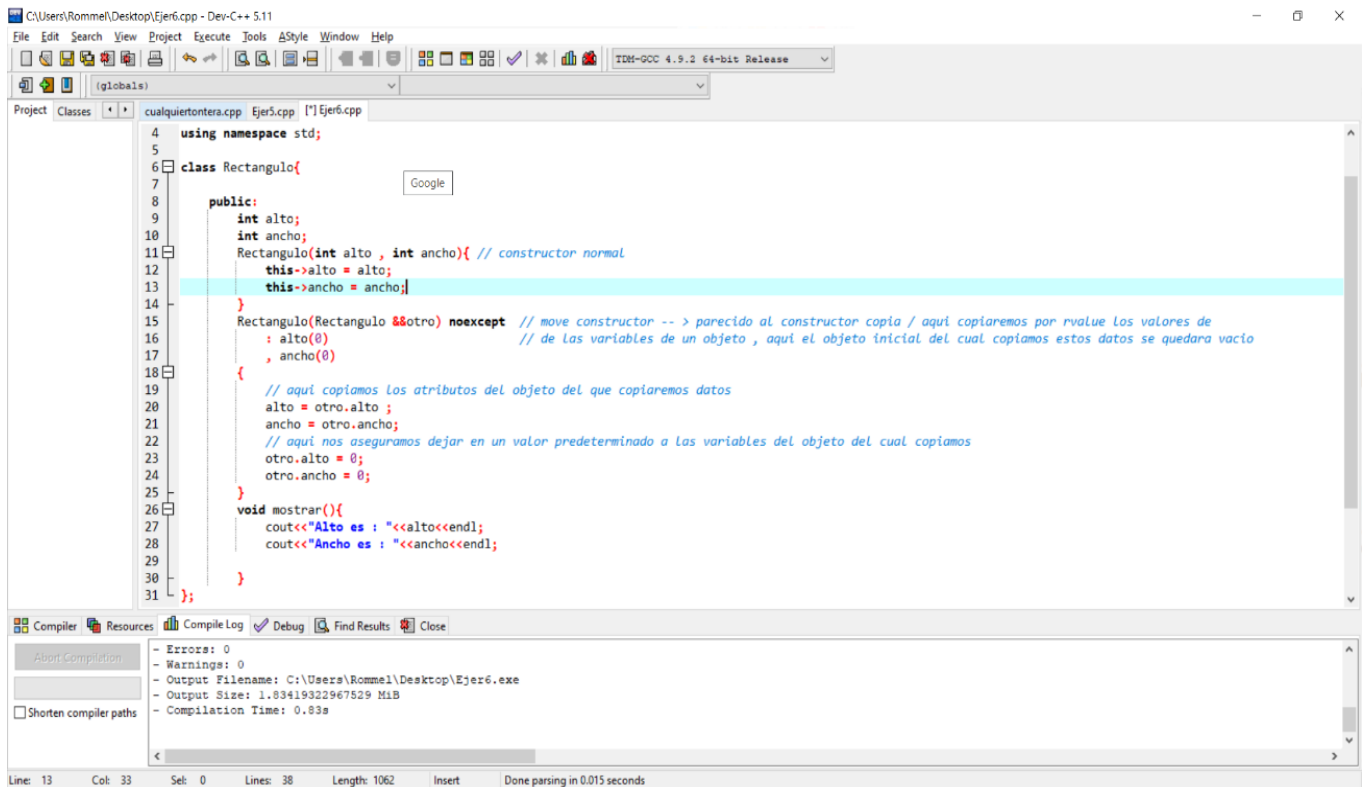
```

1 #include<iostream>
2 #include<utility>
3
4 using namespace std;
5
6 class Rectangulo{
7     int alto;
8     int ancho;
9     public:
10     Rectangulo(int alto, int ancho)
11     {
12         this->alto=alto;
13         this->ancho=ancho;
14     }
15     Rectangulo& operator=(Rectangulo &&otro) noexcept
16     {
17         if(this != &otro) // verifica que el objeto no se llame asimismo
18         {
19             this->ancho = 0; // desasigna Los valores
20             this->alto = 0;
21
22             alto = otro.alto; // mueve Los valores del objeto ingresado(a copiar)
23             ancho = otro.ancho;
24
25             otro.alto = 0; //deja al objeto del cual copiamos, en valores nulos o cero
26             otro.ancho = 0;
27         }
28         return *this; // retorna *this
29     }
30 };
31
32 int main(){
33     Rectangulo primero(10,20);
34     Rectangulo asignacion = move(primero);
35 }

```

Compiler (3): 0 Errors, 0 Warnings. Output File: C:\Users\Rommel\Desktop\Ejer7.exe. Compilation Time: 0.86s.

## 7. Investigar, describir e implementar un ejemplo usando el operador de asignación move (move assignment constructor)



```
4 using namespace std;
5
6 class Rectangulo{
7
8 public:
9     int alto;
10    int ancho;
11    Rectangulo(int alto , int ancho){ // constructor normal
12        this->alto = alto;
13        this->ancho = ancho;
14    }
15    Rectangulo(Rectangulo &&otro) noexcept // move constructor -- > parecido al constructor copia / aqui copiaremos por rvalue los valores de
16        : alto(0) // de las variables de un objeto , aqui el objeto inicial del cual copiamos estos datos se quedara vacio
17        , ancho(0)
18    {
19        // aqui copiamos los atributos del objeto del que copiaremos datos
20        alto = otro.alto ;
21        ancho = otro.ancho;
22        // aqui nos aseguramos dejar en un valor predeterminado a las variables del objeto del cual copiamos
23        otro.alto = 0;
24        otro.ancho = 0;
25    }
26    void mostrar(){
27        cout<<"Alto es : "<<alto<<endl;
28        cout<<"Ancho es : "<<ancho<<endl;
29    }
30 }
31 ;
```

Compiler Output:

```
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Rommel\Desktop\Ejer6.exe
- Output Size: 1.83419322967529 MiB
- Compilation Time: 0.83s
```

Se utiliza para transferir un objeto temporal a un objeto existente. El operador de asignación movimiento, como la mayoría de los operadores de C ++, puede ser sobrecargado. Al igual que el operador de asignación de copia es una función miembro especial.

Cabe resaltar que si no se define un operador de asignación de movimiento, el compilador nos generara uno por defecto.

El operador de asignación movimiento es diferente que un constructor movimiento debido a que un operador de asignación movimiento se llama en un objeto existente, mientras que un constructor movimiento se llama en un objeto creado por la operación.