



**Tecnológico de Monterrey**

## **Implementación de métodos computacionales**

- Programando un DFA



**Equipo:**

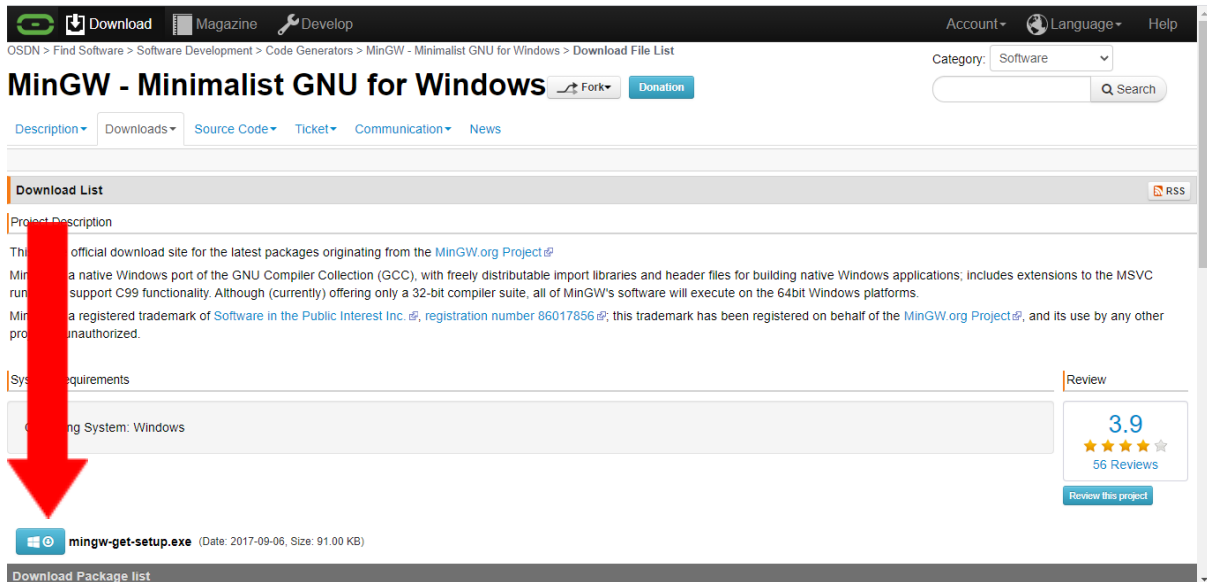
Carlos Estrada Ceballos - A01638214

Abigail Velasco García - A01638095

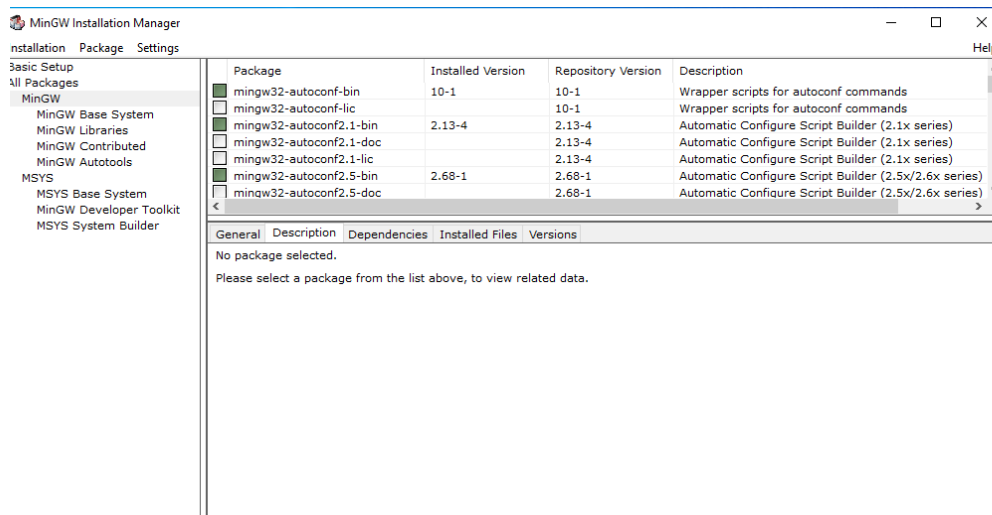
Natalia Velasco García - A01638047

## Manual de usuario: (Instrucciones para un usuario en Windows)

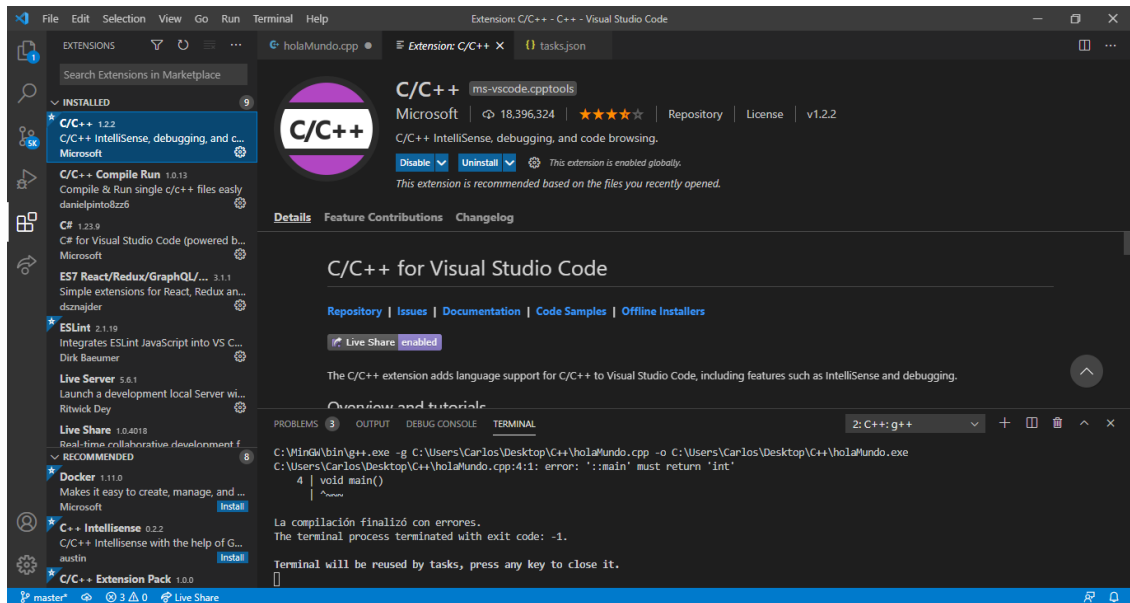
Para poder ejecutar nuestro programa se necesita de un compilador C++, recomendamos instalar mingw, existen diversos sitios desde donde descargarlo, [este](#) es uno de ellos, y descargamos el siguiente archivo:



Posteriormente ejecutamos el programa e instalamos las siguientes librerías necesarias para poder ejecutar nuestro programa.



En Visual Studio Code instalamos la siguiente extensión



Por último, checamos que nuestro documento task sea el correcto para poder ejecutar la compilación.

```

tasks.json
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "cppbuild",
      "label": "C/C++: cl.exe compilar archivo activo",
      "command": "cl.exe",
      "args": [
        "/Zi",
        "/EHsc",
        "/nologo",
        "/Fe:",
        "${fileDirname}\\${fileBasenameNoExtension}.exe",
        "${file}"
      ],
      "options": {
        "cwd": "${workspaceFolder}"
      },
      "problemMatcher": [
        "$msCompile"
      ],
      "group": "build",
      "detail": "compilador: cl.exe"
    }
  ]
}

```

```

tasks.json
{
  "type": "cppbuild",
  "label": "C/C++: cpp.exe compilar archivo activo",
  "command": "C:\\MinGW\\bin\\cpp.exe",
  "args": [
    "-g",
    "${file}",
    "-o",
    "${fileDirname}\\${fileBasenameNoExtension}.exe"
  ],
  "options": {
    "cwd": "${workspaceFolder}"
  },
  "problemMatcher": [
    "$gcc"
  ],
  "group": "build",
  "detail": "compilador: C:\\MinGW\\bin\\cpp.exe"
},
{
  "type": "cppbuild",
  "label": "C/C++: g++.exe compilar archivo activo",
  "command": "C:\\MinGW\\bin\\g++.exe",

```

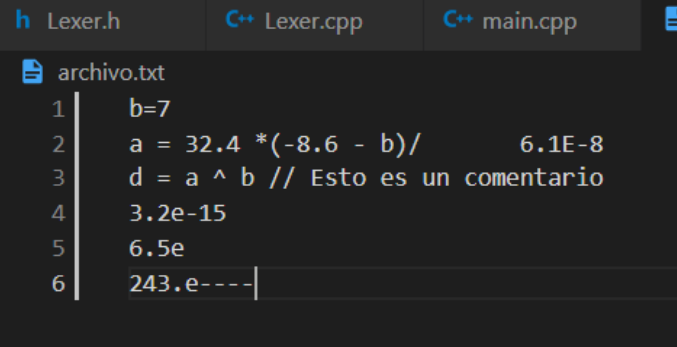
Para poder ejecutar un programa en C++, primero nos ubicamos en el archivo que contenga la función main, despues lo compilamos con lo que instalamos previamente, para acceder a ello presionamos Ctrl + Shift + B, nos desplegará un menú con las opciones y seleccionamos el compilador g++, esto nos creará un archivo ejecutable que podemos ejecutar desde nuestra terminal.

## Ejecutando el trabajo entregado

Nuestro trabajo está desarrollado en el lenguaje C++, consta de tres archivos, un archivo con extensión 'h' y dos archivos con extensión 'cpp', el primero cuenta con la definición de nuestra clase, mientras que uno de los archivos 'cpp' es la definición de dicha clase, el tercer archivo contiene la función main el cual es el encargado de arrancar el programa, los archivos tienen el nombre de Lexer.h, Lexer.cpp y main.cpp, además de estos tres archivos tenemos un cuarto el cual es nuestro archivo que proporciona la entrada a nuestro programa llamado archivo.txt. Para probarlo simplemente compilamos desde el archivo main, el cual cuenta con nuestra función main encargada de iniciar el programa, después ejecutamos el archivo .exe desde una terminal.

La función del programa es separar la entrada en tokens y categorizarlos según su tipo, es un analizador léxico. La entrada del programa es un archivo de texto que contiene las expresiones que se deseen analizar.

Este es un ejemplo de una entrada:



```
h Lexer.h  C++ Lexer.cpp  C++ main.cpp
archivo.txt
1   b=7
2   a = 32.4 *(-8.6 - b)/      6.1E-8
3   d = a ^ b // Esto es un comentario
4   3.2e-15
5   6.5e
6   243.e----|
```

Una vez que se tiene el archivo txt de entrada en el archivo main en la línea 18, se encuentra la función a la que se pasa la entrada, entre los paréntesis se debe escribir el nombre del archivo de entrada.



```
13
14 int main () {
15     Lexer Lexer1;
16     vector<Token> tokens;
17
18     ifstream ip("archivo.txt");
19     string mes;
20
21     cout << endl << "--TOKEN--      --TIPO--\n" << endl;
22
23     while(!ip.fail()) {
24         getline(ip, mes, '\n');
25         if(ip.fail()){
26             break;
27         }
28         tokens = Lexer1.LexerAritmetico(mes);
29         for(Token currToken : tokens) {
30             currToken.lPrint();
31         }
32     }
33
34     return 0;

```

La salida del programa es una tabla de dos columnas, en la primera están todos los tokens identificados y en la segunda el tipo de token al que corresponden, o error en caso de que un token esté mal escrito.

Ejemplo de salida:

```
--TOKEN--      --TIPO--
b              Variable
=              Asignacion
7              Entero
a              Variable
=              Asignacion
32.4           Flotante
*              Multiplicacion
(              Parentesis_que_abre
-              Resta
8.6           Flotante
-              Resta
b              Variable
)              Parentesis_que_cierra
/              Division
6.1E-8         Flotante_exponencial_negativo
d              Variable
=              Asignacion
a              Variable
^              Potencia
b              Variable
// Esto es un comentario      Comentario
3.2e-15        Flotante_exponencial_negativo
6.5e           Error_token
243.e-         Error_token
-              Resta
-              Resta
-              Resta
```

