SPL-1 Project Report,2023

Control Flow Graph

From C source file

SE 305 : Software Project Lab

Submitted by:

Mst. Fareya Azam

BSSE Roll: 1331

Session: 2020-2021

Supervised by:

Dr. Sumon Ahmed

Designation: Associate Professor

Institute of Information Technology

Supervisor's Signature…………………………………..

Institute of Information Technology

University of Dhaka

Date: 21-05-2023

Table of contents

1.Introduction:

Control flow graph is graphical representation of a program, where statements are represented by nodes and statement dependency or jumps are represented by direct edges. Control flow graph shows all the path that we can traverse during execution.

The project aims to develop a program that will take a source code file in C as input and generate its Control flow graph as output.

1.1Background study:

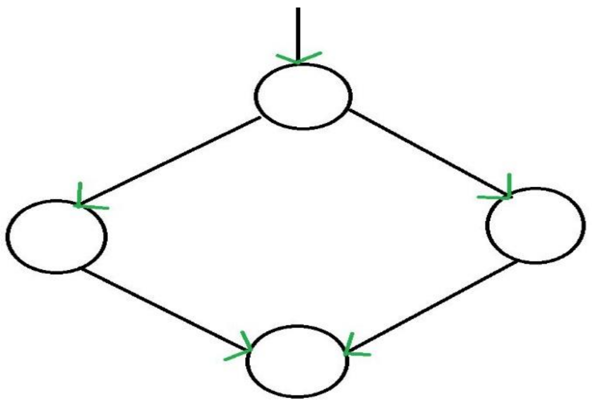
I had to study the following topics for the project:

Types of statement in C:

Statements are the executable parts of the program that will do some actions. Types of statements that we need are:

a. Selection statement:

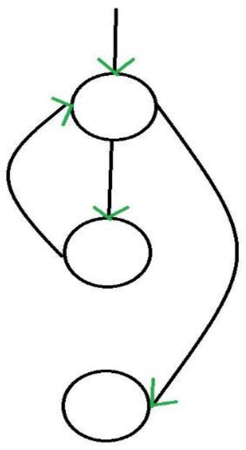
1. if
2. else if
3. else
4. Switch

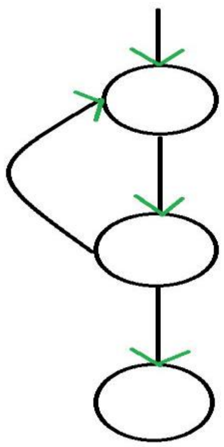

If-then-else

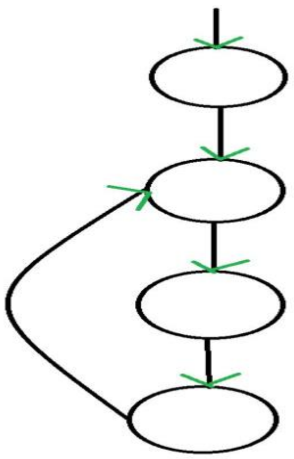
b. Iterative statement:

Loops:

- Do loop
- While loop
- For loop


while


do-while


for

C. Jump Statements:

break, goto, continue.

D. Expressive statement:

Y=X+10;

Lexical Analysis:

it helps to tokenize the source code.

DFS algorithm, BFS algorithm:

These algorithms are used to visualize the Control Flow of a graph.

1.2 Challenges

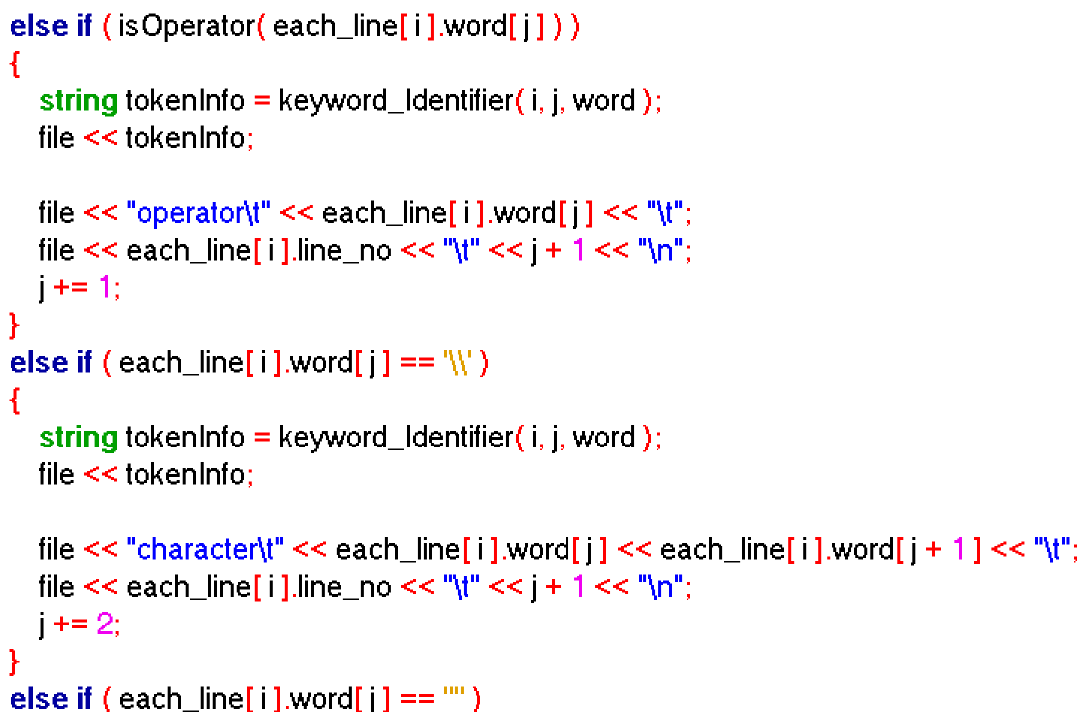
The challenges I had face while working on this project:

- The source code could have comments and unnecessary white spaces.
- Establishing relations between edges was very difficult.
- Debugging programs while observing multiple variables and their interactions with multiple methods.

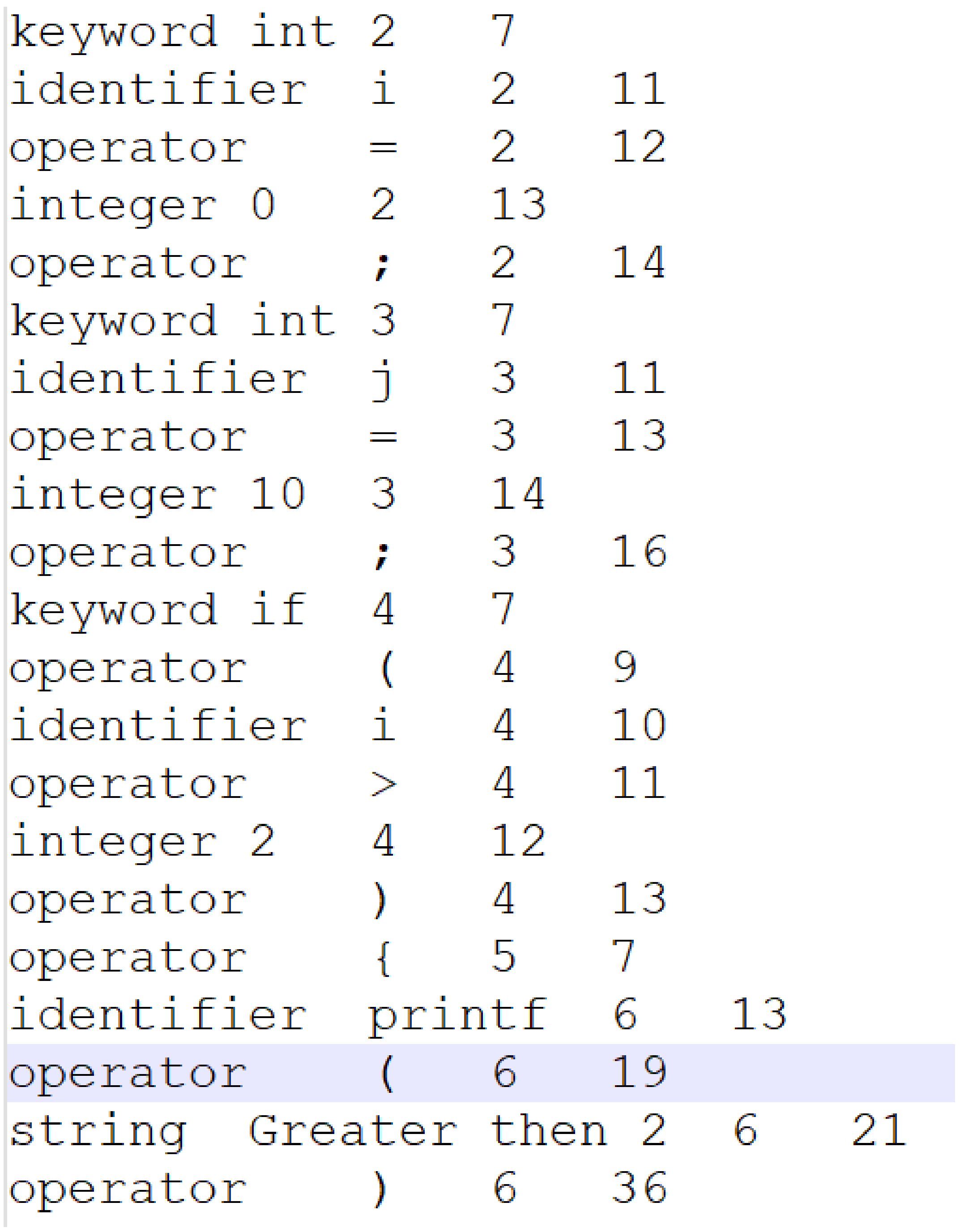
2.Overview

The project has the following algorithm implementations:

- Take a C source code as input and tokenize it and store the tokens in a .txt file with token Type, line number and column number.



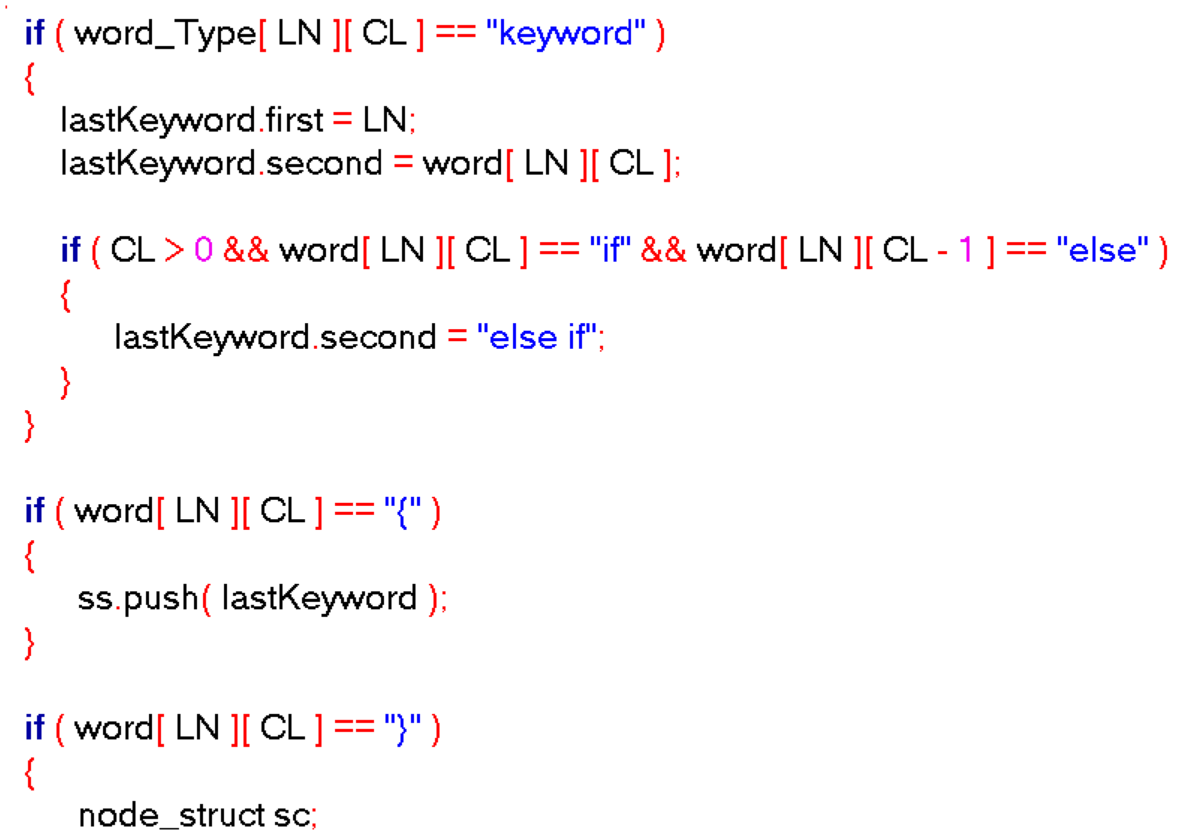
```
else if ( isOperator( each_line[ i ].word[ j ] ) )
{
   string tokenInfo = keyword_Identifier( i, j, word );
   file << tokenInfo;

   file << "operator\t" << each_line[ i ].word[ j ] << "\t";
   file << each_line[ i ].line_no << "\t" << j + 1 << "\n";
   j += 1;
}
else if ( each_line[ i ].word[ j ] == '\\' )
{
   string tokenInfo = keyword_Identifier( i, j, word );
   file << tokenInfo;

   file << "character\t" << each_line[ i ].word[ j ] << each_line[ i ].word[ j + 1 ] << "\t";
   file << each_line[ i ].line_no << "\t" << j + 1 << "\n";
   j += 2;
}
else if ( each_line[ i ].word[ j ] == '"' )
```



```
keyword int 2   7
identifier  i   2   11
operator    =   2   12
integer 0   2   13
operator    ;   2   14
keyword int 3   7
identifier  j   3   11
operator    =   3   13
integer 10  3   14
operator    ;   3   16
keyword if  4   7
operator    (   4   9
identifier  i   4   10
operator    >   4   11
integer 2   4   12
operator    )   4   13
operator    {   5   7
identifier  printf  6   13
operator    (   6   19
string  Greater then 2  6   21
operator    )   6   36
```

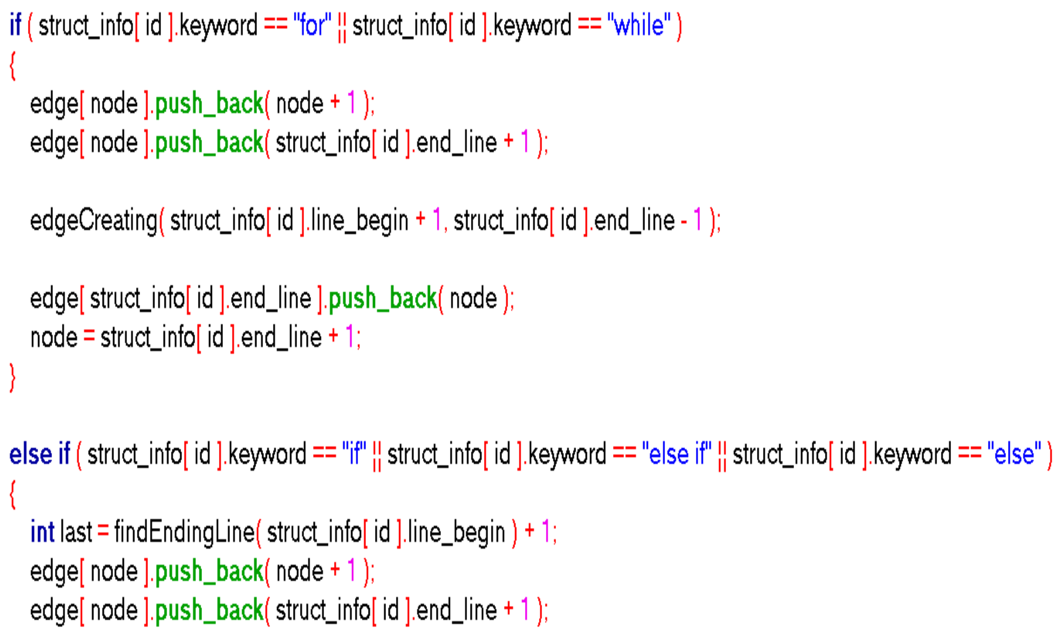
- Detecting expression/Selection/Iteration statements:

Here is the sample code on how this were done:



```
if ( word_Type[ LN ][ CL ] == "keyword" )
{
   lastKeyword.first = LN;
   lastKeyword.second = word[ LN ][ CL ];

   if ( CL > 0 && word[ LN ][ CL ] == "if" && word[ LN ][ CL - 1 ] == "else" )
   {
       lastKeyword.second = "else if";
   }
}

if ( word[ LN ][ CL ] == "{" )
{
    ss.push( lastKeyword );
}

if ( word[ LN ][ CL ] == "}" )
{
    node_struct sc;
```

- Establishing relations between nodes:
  In this project we consider the program only contains Expression, selection and iterative statements.

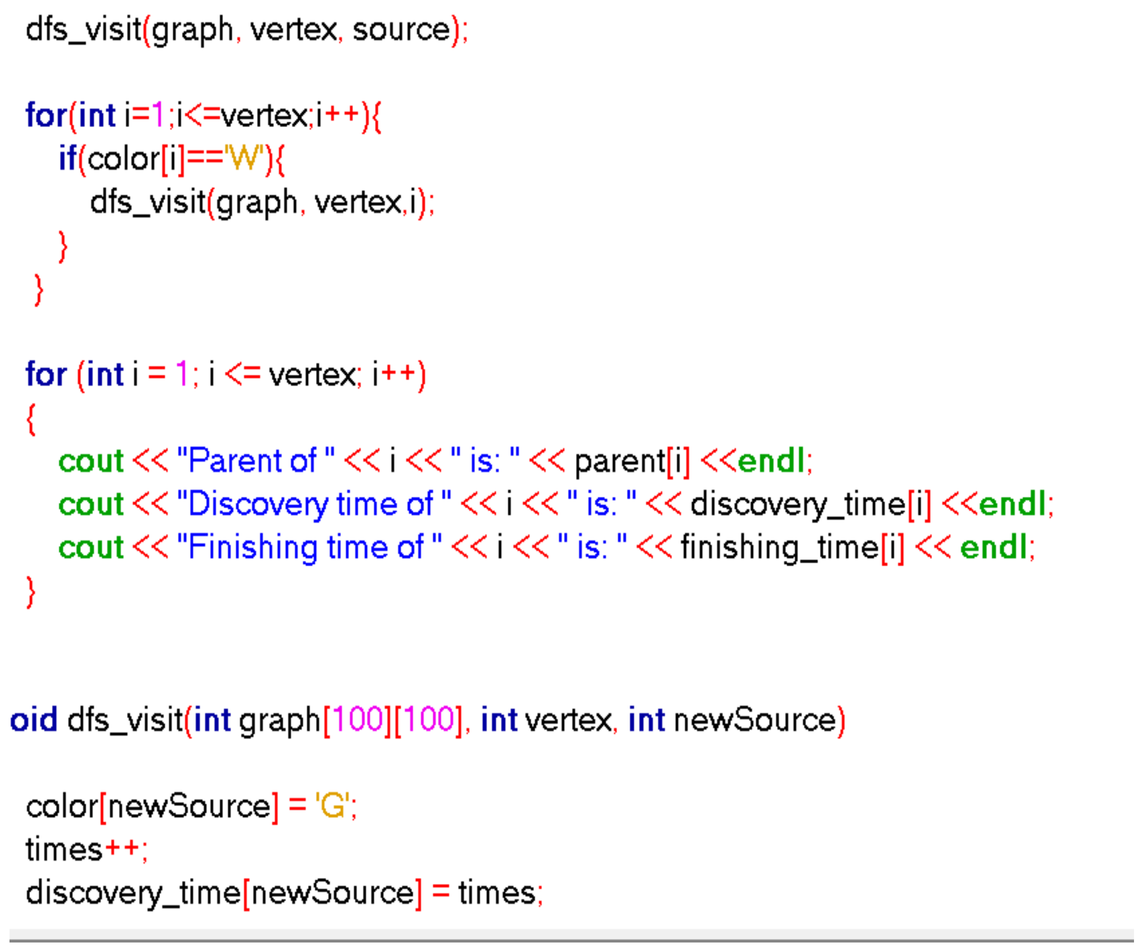


```
if ( struct_info[ id ].keyword == "for" || struct_info[ id ].keyword == "while" )
{
   edge[ node ].push_back( node + 1 );
   edge[ node ].push_back( struct_info[ id ].end_line + 1 );

   edgeCreating( struct_info[ id ].line_begin + 1, struct_info[ id ].end_line - 1 );

   edge[ struct_info[ id ].end_line ].push_back( node );
   node = struct_info[ id ].end_line + 1;
}

else if ( struct_info[ id ].keyword == "if" || struct_info[ id ].keyword == "else if" || struct_info[ id ].keyword == "else" )
{
   int last = findEndingLine( struct_info[ id ].line_begin ) + 1;
   edge[ node ].push_back( node + 1 );
   edge[ node ].push_back( struct_info[ id ].end_line + 1 );
```

- Creating   matrix:
  In this part I have created a matrix which will represent the graph.

- I have used DFS and BFS algorithm for visualizing the flow of control in the graph. Using coloring technic I detect the currently visited and unvisited nodes in the graph.

Here is a part of this code :

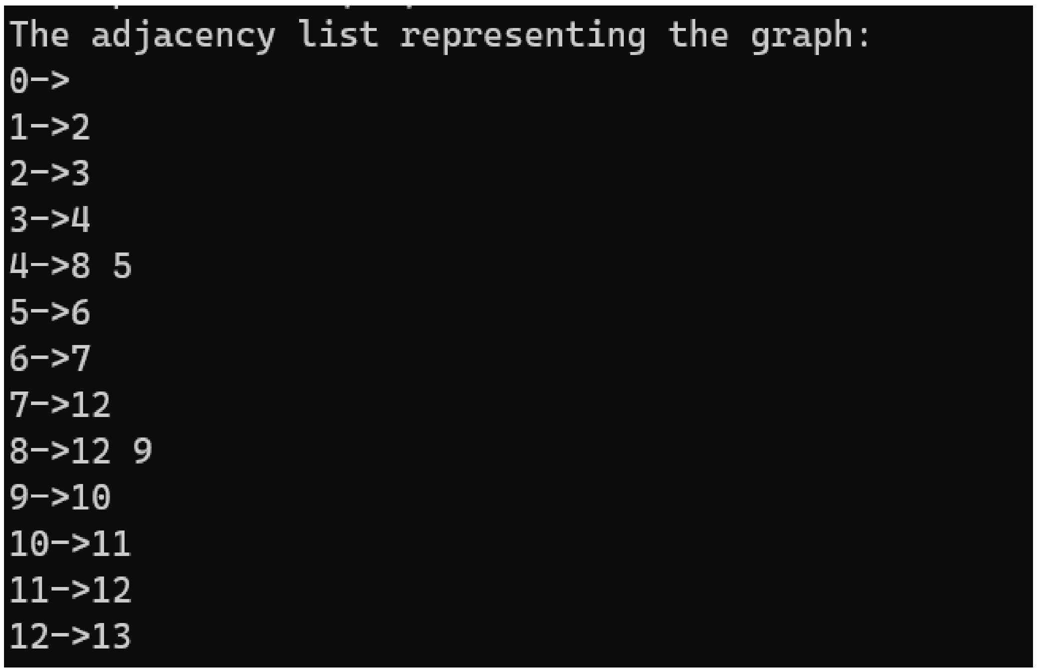


```
dfs_visit(graph, vertex, source);

for(int i=1;i<=vertex;i++){
   if(color[i]=='W'){
      dfs_visit(graph, vertex,i);
   }
 }

for (int i = 1; i <= vertex; i++)
{
   cout << "Parent of " << i << " is: " << parent[i] <<endl;
   cout << "Discovery time of " << i << " is: " << discovery_time[i] <<endl;
   cout << "Finishing time of " << i << " is: " << finishing_time[i] << endl;
}


oid dfs_visit(int graph[100][100], int vertex, int newSource)

 color[newSource] = 'G';
 times++;
 discovery_time[newSource] = times;
```

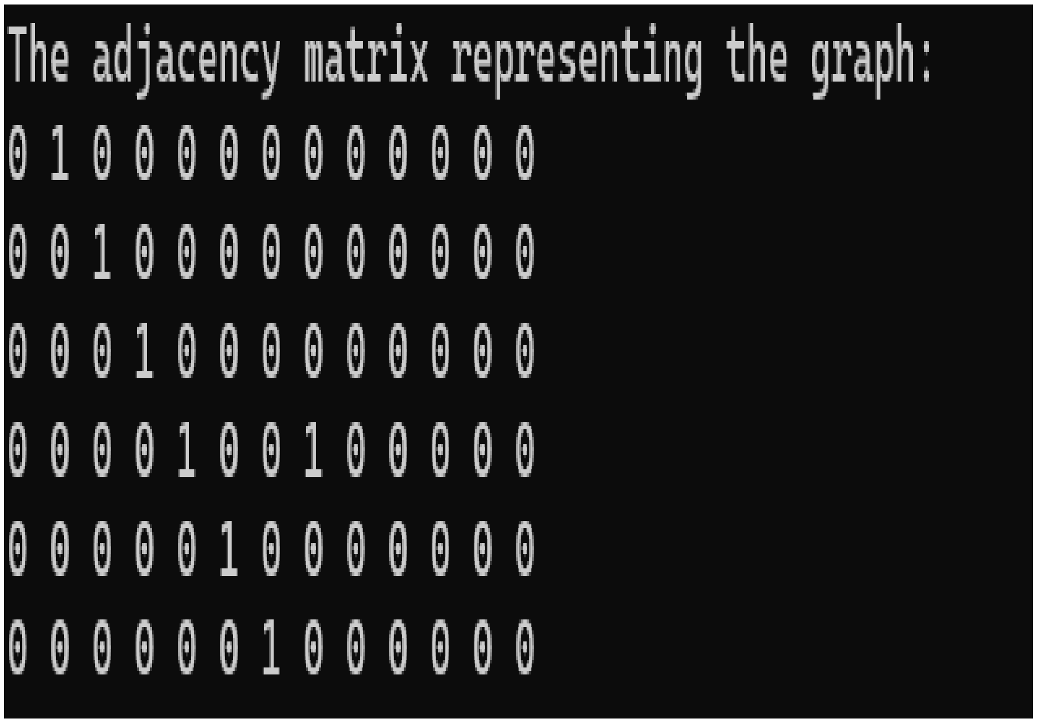
3.User manual

Before executing the code we need to create a C file that will contain the source code, for which it will create a CFG.

The project will run and show the output in the following format:



```
The adjacency list representing the graph:
0->
1->2
2->3
3->4
4->8 5
5->6
6->7
7->12
8->12 9
9->10
10->11
11->12
12->13
```



```
The adjacency matrix representing the graph:
0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0
```

4.Conclusion:

This project, from the problem solving point of view, was very interesting to me. It was focused on creating and implementing algorithms .I have gained deeper knowledge on how the compiler works and how the function calling happens .I hope in future I can add more features to this to handle complex scenarios.

References:

https://groups.seas.harvard.edu/courses/cs153/2018fa/lectures/Lec17-CFG-dataflow.pdf

https://www.geeksforgeeks.org/software-engineering-control-flow-graph-cfg/

https://en.wikipedia.org/wiki/Control-flow_graph