SPL-1 Project Report, 2020

Control Flow Graph
from C source file

Course Code: SE305

Submitted by:
Ashraful Islam Shanto Sikder
BSSE Roll: 1124
Exam Roll: 1116
Session: 2018-2019

Supervised by:
Abdus Satter
Designation: Lecturer
Institute of Information Technology
University of Dhaka

Supervisor's Signature **.....**

Table of Contents

1. Introduction…….3

1.1 Background Study……3

1.2 Challenges……..……...5

2. Overview……....6

3. User Manual……...8

4. Conclusion……10

1. Introduction

Control Flow Graph (CFG) is a graph representation of a program, where statements are represented by nodes and statement-dependency or jumps are represented by directed edges. Control flow graph shows all the paths that can be traversed during a program execution..

This project is about generating a Control Flow Graph of a program. 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.

1.1. Background Study

I had to study the following topics to implement the algorithms needed for the project.

Types of Statements in C :

C programs are a collection of statements, statements are executable parts of the program that will do some action. In general all arithmetic actions and logical actions falls under Statements Categories. There are few Statement categories -

- **Expression Statements** : X = Y + 10 **;**

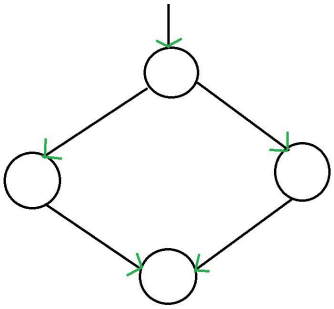
- **Selection Statements :**

  - if

  - else if

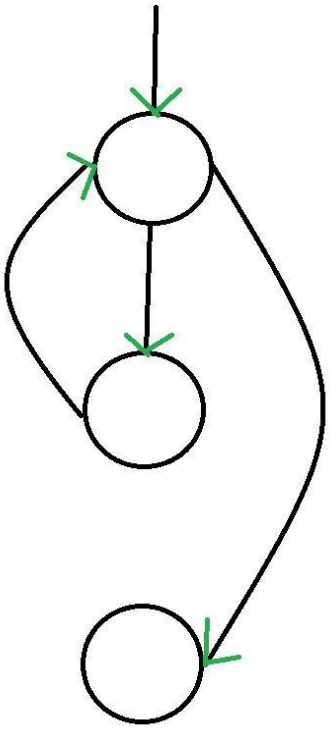
- else

- switch



If-then-else

- **Iterative Statements :**
  - Loops



Loops

- **Jump Statements :**
    - goto , continue, break;

*images were collected from
https://www.geeksforgeeks.org/software-engineering-control-flow-graph-cfg/

Parsing :

Parsing or syntactic analysis is an analysis where a string of commands , usually a program , is separated into more easily processed components, which are analyzed for correct syntax and then attached to tags that define each component.

1.2 Challenges

The challenges I had to face while working on this project -

● The source code file could have comments and unnecessary white spaces.

● Establishing relations inside and outside branchings was very tricky. This was the toughest part of the project.

● Debugging programs while observing multiple variables and their interactions with multiple methods.

4. Overview

The project has the following algorithm implementations-

Detecting Lines as Expression/Selection/Iteration statements:
A simple C program has mainly three types of statements - *Expression* , *Selection* **(**if-else) and *Iteration* **(**loops). Here are some examples how these were identified -

```
public boolean isStatement(String statement){
    statement = statement.replaceAll("\\s","");
    return (statement.charAt(0)!='/' && statement.charAt(1)!='/' && statement.charAt(statement.length()-1)==';');
}
public boolean isIf(String statement){
    statement = statement.replaceAll("\\s","");
    if(statement.length()>=3){
        if(statement.charAt(0)=='i' && statement.charAt(1)=='f' && statement.charAt(2)=='('){
            return true;
        }
    }
    return false;
}

public boolean isElseIf(String statement){
    statement = statement.replaceAll("\\s","");statement = statement.replaceAll("\\s","");
    if(statement.length()>=7){
        if(statement.charAt(0)=='e' && statement.charAt(1)=='l' && statement.charAt(2)=='s'
                                    && statement.charAt(3)=='e' &&  statement.charAt(4)=='i'
                                    && statement.charAt(5)=='f' &&  statement.charAt(6)=='('){
            return true;
        }
    }
    return false;
}
```

```
public boolean isFor(String statement){
    statement = statement.replaceAll("\\s","");
     if(statement.length()>=4){
        if(statement.charAt(0)=='f' && statement.charAt(1)=='o' && statement.charAt(2)=='r' &&  statement.charAt(3)=='('){
            return true;
        }
    }
    return false;
}
```

Establishing relations(edges) between nodes:

In this project, we considered the program only contains Expression, Selection and Iterative Statements.

Selection statements branch the current node into a minimum of two branches. All the branchings again land on a same particular node after their execution. The code below makes the edges between all the branchings and the landing node.

```
if(branchingsOfThisBranch.size()>0){
    for(int i=0; i<branchingsOfThisBranch.size(); i++){
        branchingsOfThisBranch.get(i).childs.add(curNode);
    }
    branchingsOfThisBranch.clear();
}
```

Iteration statements create a back edge from the end of the loop-block to the loop declaration statement. The code below creates the back edge for the loops.

```
if(inLoop==true) {
    curNode.childs.add(branchRoot);
}
```

Recursion was used for building relationships in every branching and between roots of the branchings and terminators.

Depth First Search Traversal : After establishing all the relationships(edges) between nodes, dfs was used to create the Adjacency List and Adjacency Matrix.

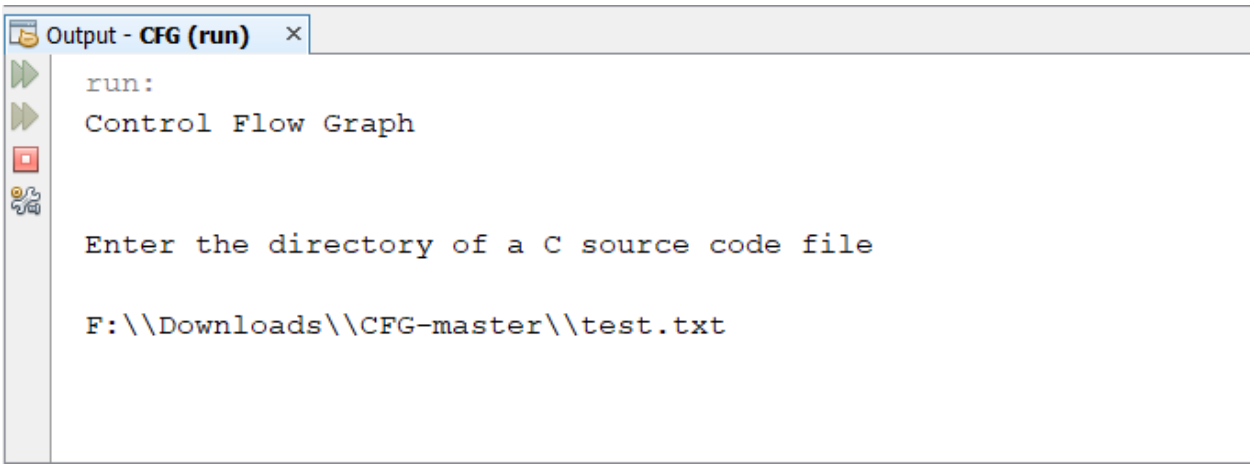
```
public void dfs(Node cur, int prev){
    //System.out.println(prev + " " + cur.nodeNumber+" "+cur.Statement);
    vis[cur.nodeNumber] = true;

    for(int i=0; i<cur.childs.size(); i++){
        int nodeNo = cur.childs.get(i).nodeNumber;
        if(vis[nodeNo]==false){
            dfs(cur.childs.get(i), cur.nodeNumber);
        }
    }

    for(int i=0; i<cur.childs.size(); i++){
        adj[cur.nodeNumber][cur.childs.get(i).nodeNumber] = 1;
    }
}
```

User Manual

The accurate directory of the C source code file should be given as input -



Then the project will run and output the CFG for the given C code in the following format -

```
root node no: 2
root node statement: int b = 20;

Adjacency List:
        0  ->
        1  ->
        2  ->   3 6 9
        3  ->   4
        4  ->   5
        5  ->   12
        6  ->   7
        7  ->   8
        8  ->   12
        9  ->   10
        10  ->   11
        11  ->   12
        12  ->   13
        13  ->   14 16
        14  ->   15
        15  ->   13
        16  ->   17 19
        17  ->   18
        18  ->   16
        19  ->   20
        20  ->

Adjacency Matrix:
        0       0 0 0 0 0 0 0 0 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 0 0 0 0 0 0 0 0
        2       0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
        3       0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
        4       0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
        5       0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
        6       0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
        7       0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
        8       0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
        9       0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
```

Conclusion :

The project, from the problem-solving point of view, was very interesting to me. It was focused on creating and implementing algorithms rather than using some library tools and frameworks. Although the project currently needs to use a simpler C source file, I hope to add more features to this to handle more complex scenarios. And I had a great learning experience doing the project, which, I hope will help me in the upcoming days.