Purpose Of The Program

Given a control flow graph, create the Dominator Tree.
Use the dominator information to detect Back Edges and the corresponding Loops.

Directory Structure

1. **source** : Contain the source code files.
    a. Main.c
        i. The main program which invoke various utility and routine functions.
    b. Routines.c
        i. Function definition targetted to the given problem.
    c. Utils.c
        i. Basic utility functions for set (Union , Intersection), stack operations, graph operations(Finding successor, predecessor), Memory related utilities(CleanUp routines, Allocation routines).
    d. Header.h
        i. Contain the functional prototypes, global variable declarations.
2. **Test** : Contain 10 sample input files for different control flow graphs.
3. **Gold** : Contain sample outputs for the above 10 input files.

How To Compile

```
gcc source/*.c
```

How To Execute

```
./a.out  <input_file_name>
```

Assumptions

If 'N' is the number of vertices of the graph, then the vertices MUST BE LABELLED FROM 1 to N, but in any order.

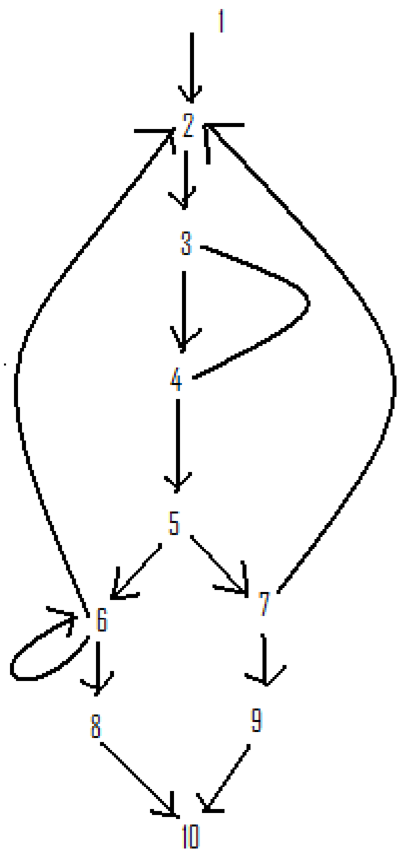
Input Format

The program takes input from a file, whose format is as follows:

```
<N>                  /*The Number of Vertices Of Control Flow Graph*/
<Entry_Point>        /*An integer From 1 to N*/
<I>   <J>            /*I and J are are two integers(from 1 - N) representing the edge I-J*/
<J>   <K>            /*J and K are are two integers(from 1 - N) representing the edge J-K*/
```

Sample Input

Consider the following graph,



The corresponding input file:

```
10
1
1 2
2 3
3 4
4 5
5 6
5 7
6 8
7 9
8 10
9 10
6 6
6 2
7 2
4 3
```

Output

The Output is dumped in a file "Output.Info".

Sample Output

For the above sample input the generated output will be as follows:

```
'D(a) = Set Of Nodes Which Dominates a'
------------------------
 Dominator Information
------------------------

 D(1) = [ 1 ]
 D(2) = [ 1 2 ]
 D(3) = [ 1 2 3 ]
 D(4) = [ 1 2 3 4 ]
 D(5) = [ 1 2 3 4 5 ]
 D(6) = [ 1 2 3 4 5 6 ]
 D(7) = [ 1 2 3 4 5 7 ]
 D(8) = [ 1 2 3 4 5 6 8 ]
 D(9) = [ 1 2 3 4 5 7 9 ]
 D(10) = [ 1 2 3 4 5 10 ]
```

```
                    --------------------
                     The Dominator Tree
                    --------------------

   1
      2
         3
            4
               5
                  6
                     8
                  7
                     9
                  10

              -------------------------------------
               Back Edge        Natural Loop
              -------------------------------------

               7 --> 2            [ 2 7 5 4 3 ]
               6 --> 6            [ 6 ]
               6 --> 2            [ 2 6 5 4 3 ]
               4 --> 3            [ 3 4 ]

'If Any Two BackEdges Have The Same 'Head', We Combine The Coresponding
Loops'
              ------------------------
               Combined Natural Loops
              ------------------------

              The Following BackEdges can be combined :

               1 ) {7 --> 2} {6 --> 2}
               The Corresponding Merged Natural Loop : [2 3 4 5 6 7 ]
```

Output Interpretation

1. For each node ‘a’ of the control flow graph, the set of dominators D(a) is displayed.
2. The above dominator tree could be read as follows:

2	*: child of 1*
3	*: child of 2*
4	*: child 0f 3*
5	*: child of 4*
6, 7, 10	*: children of 5*
8	*: child of 6*

9 : child of 7

3. In the control flow graph, using the dominator information the backedges are determined and for each of them the corresponding Natural Loop is displayed.
4. *If* two back edges have the same ***head*** then their corresponding natural loops could be merged.
   The output depicts which backedges could be merged and the resultant merged natural loop.

Design Documentation

1. The input control flow graph is read from the input file and an adjescency matrix **ADJ_MATRIX** of size N*N is populated.
   a. Relevent functions: **void InitializationRoutine(FILE*);**
2. Using the above information the dominator of each node is found using the following algorithm:

```
for each n∈N do Domin(n) = N;
  Domin(entry) = {entry};
  repeat
       for each n∈N-{entry} do
            T = N;
            for each p∈pred(n) do
                 T = T ∩Domin(p);
            Domin(n) = {n} ∪T;
  until Domin is unchanged
```

   The above dominator info is stored in a N*N matrix **DOMINATOR_INFO.**
   a. The relevent functions: **void createDominatorInfo(void);**
3. Using the above information the Dominator Tree is created.
   a. The relevent functions: **void createDominatorTree(void);**
4. Using the **DOMINATOR_INFO,** all the back edges in the control flow graph is determined, from the following definition of backedge
   back-edge a -> b such that b dom a
   a. The relevent functions:
   void findBackEdgesAndNaturalLoops(void);
5. For each back edge, its natural loop is determined from the following definition
   Given back edge a -> b natural loop is a sub graph which contains a,

b and all the nodes which can reach a without passing through b

For that the following algorithm is used :

```
stack = empty;
loop = {b};
insert(a);
while stack is not empty do {
    pop m of the stack;
    for each predecessor p of m do insert(p);
}

procedure insert(m);
    if m is not in loop then {
        loop = loop U {m};
        push m onto stack;
    }
```

a . The revevent functions: **void findNaturalLoop(int , int);**

6. The backedges with the same head can be combined and this is implemented in function: **void mergeLoops(void);**