'11. 12. 8, Presentation #5

Control Flow Graph Generator

T11

200412351 정백균

200811468 강민석

200913987 이승효

201011340 신승우

Contents.

- Modification of SASD
- Source Code with SASD
- Generating Example & DEMO

Modification of SASD

Statement of Purpose

◆ *Software Requirement Specification(SRS)*

① **Input** : the file written by C-language and commands are entered by user.

② **Output** : CUI and report File that holds the information of the CFG

- The report show all states and edges of CFG.

◆ *Environment*

③ **It operated in Cygwin, execution of the software is in the form Command Line Commands.**

Ex) #gcc ./CG Inputcode.c result.txt gcc –o CG main.c

- When a user inputted unpermitted command, the program shows 'help'

◆ *Source Code*

④ **C source code has 100~200 lines. It includes main function.**

⑤ **The code operates for a single file and The C source code doesn't have user defined header files.**

⑥ **The C source code doesn't include pointers.**

⑦ **Only normally compiled files will be operated.**

Statement of Purpose

- ***Our CFG Program***

⑧ **Every statement is specified using a number, order of the files that received input are same as Numerical order.**

⑨ **Between Block and Block is connected to the appropriate edge and generate CFG by Block.**

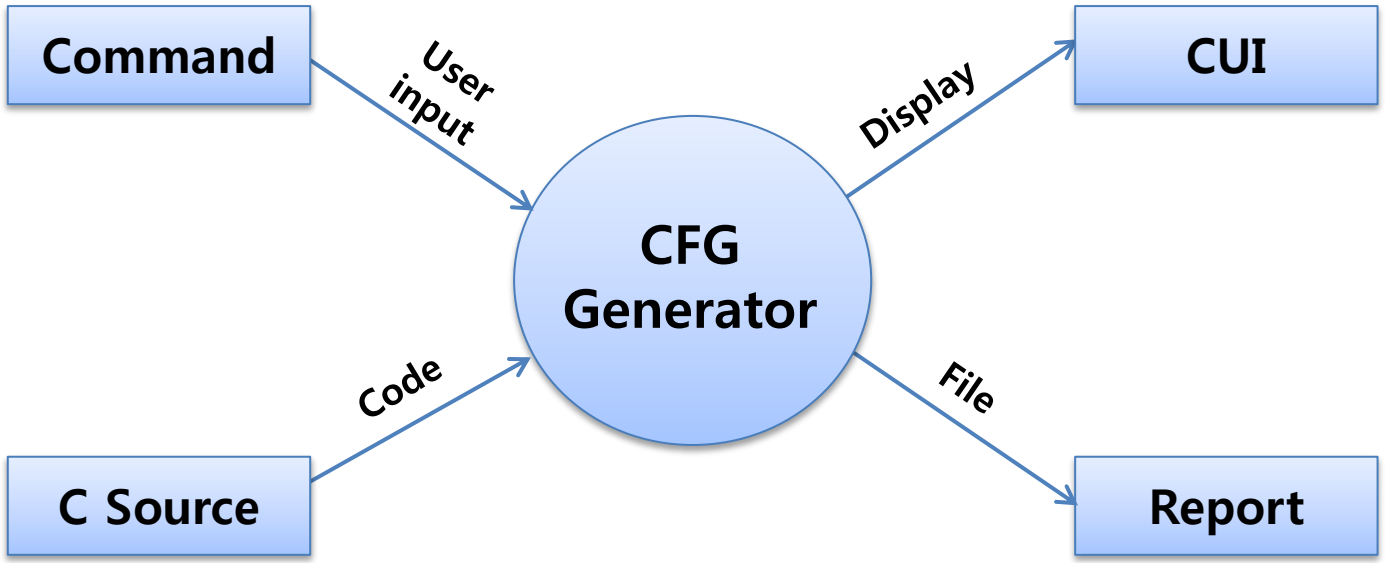
⑩ **Need to process about function calls, ‘for’, ‘while’ and ‘switch’ statements.**

⑪ **Between Block and Block is connected to the appropriate edge**

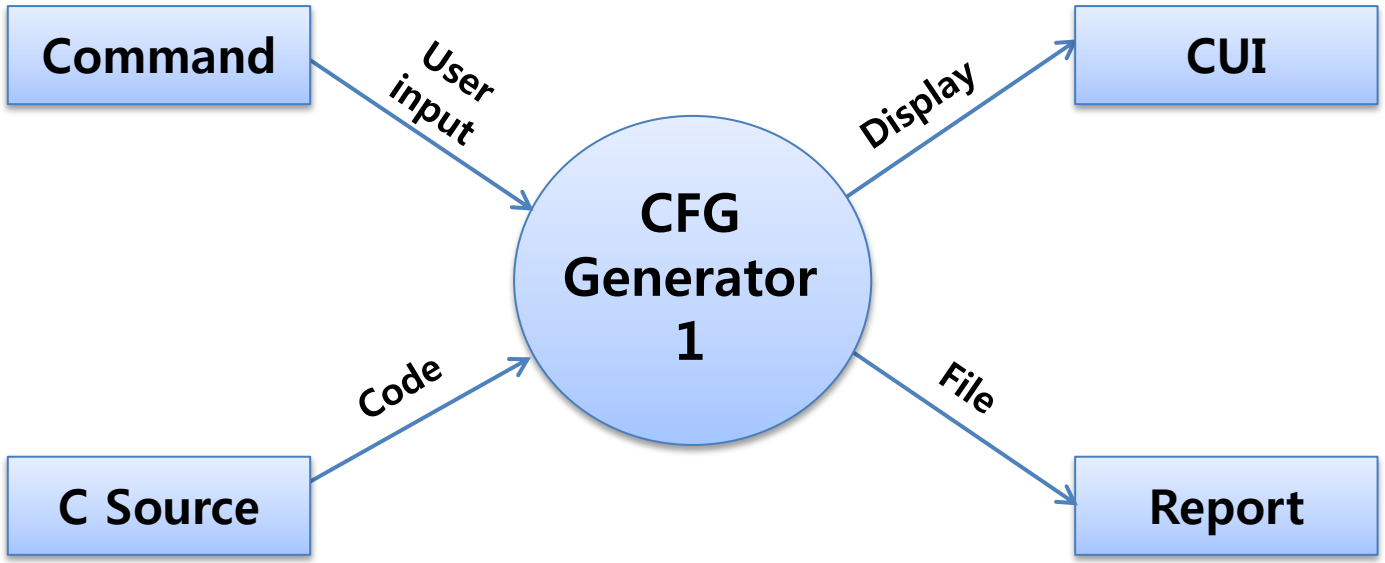
⑫ **CFG is composed of the result of the conversion Report, Block, Edge and Error Message.**

⑬ **Input C Source file is one instruction in a line**

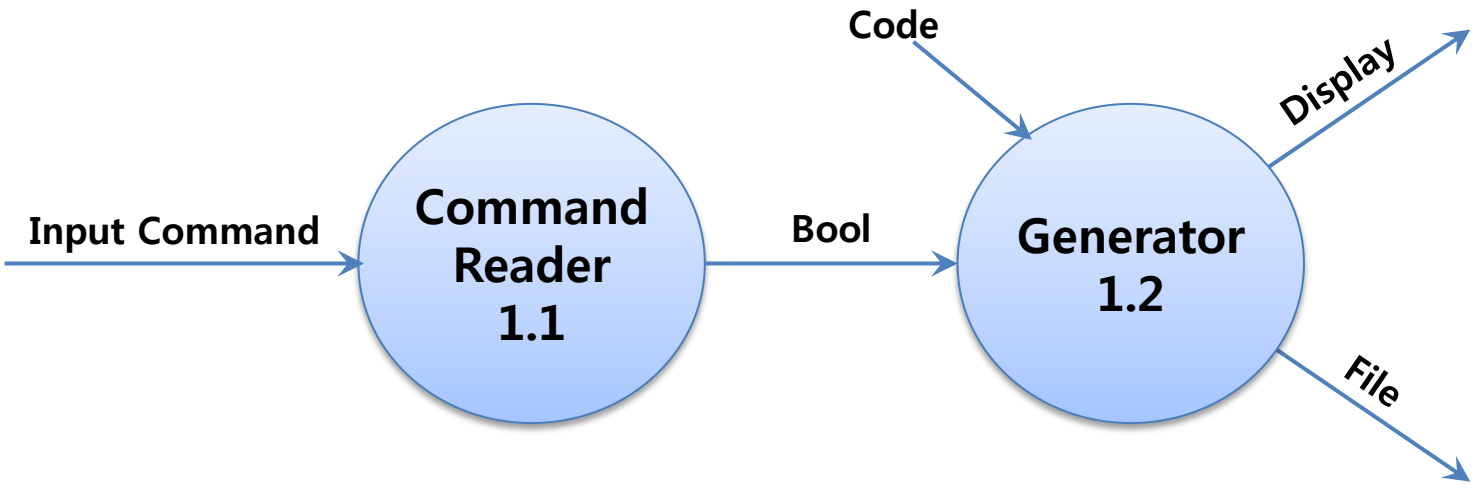
System Context Diagram



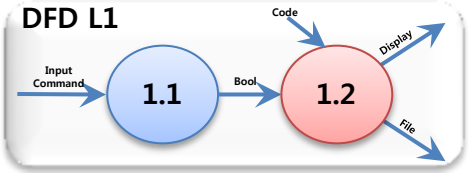
DFD Level 0


Command
User input
CUI
Display
CFG Generator 1
Code
File
C Source
Report

DFD Level 1


Code
Display
Input Command
Command Reader 1.1
Bool
Generator 1.2
File

DFD Level 2



Code

Convert
1.2.2

Trigger

Bool

Controller
1.2.1

Trigger

Show
Error
Message
1.2.3

Error
Message

CFG

Create
File
1.2.4

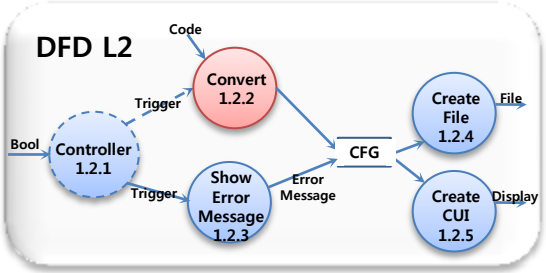
File

Create
CUI
1.2.5

Display

DFD Level 3

- Convert



Code

Trigger

Determine Leader 1.2.2.1

List of Leader

Create of Basic Block 1.2.2.2

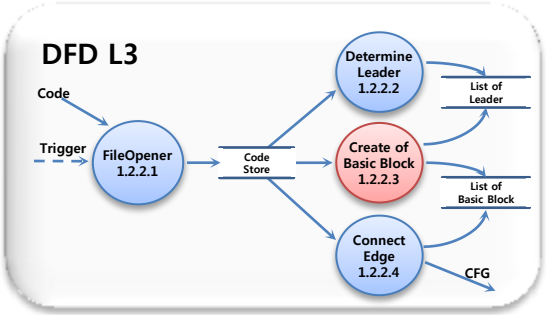
List of Basic Block

Connect Edge 1.2.2.3

CFG

DFD Level 4.1

- Create of Basic Block



Code Store

List of Leader

Leader Reader 1.2.2.3.1

Trigger

List of Leader

Trigger

Trigger

List of Leader

Trigger

Normal Block 1.2.2.3.2

Branch Block 1.2.2.3.3

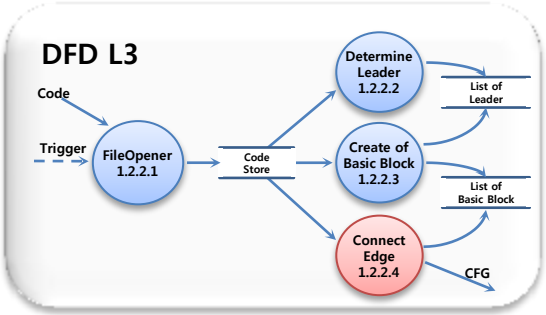
Repeat Block 1.2.2.3.4

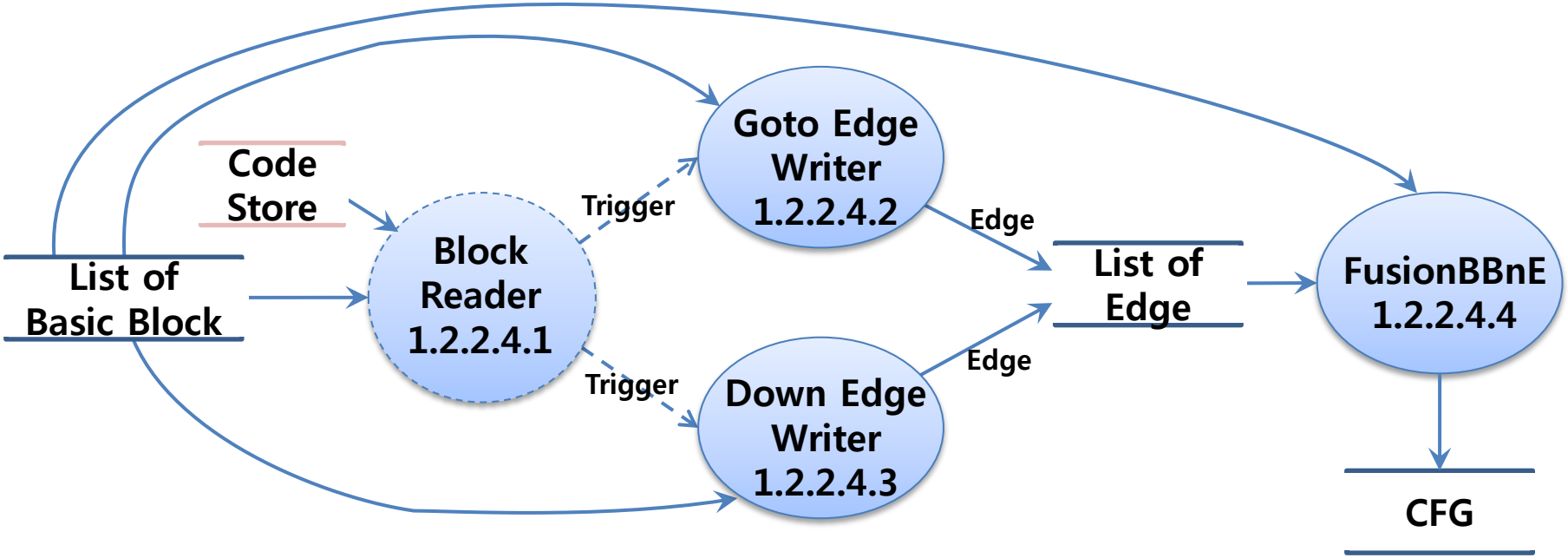
Function Block 1.2.2.3.5

List of Basic Block

DFD Level 4.2

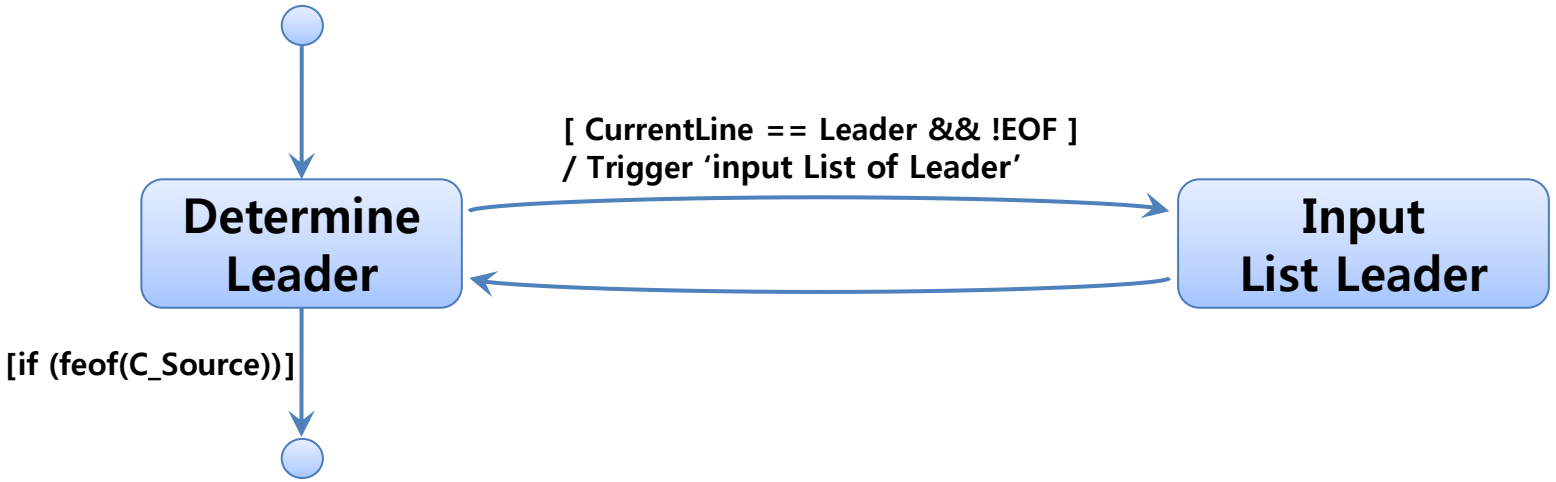
- Connect Edge





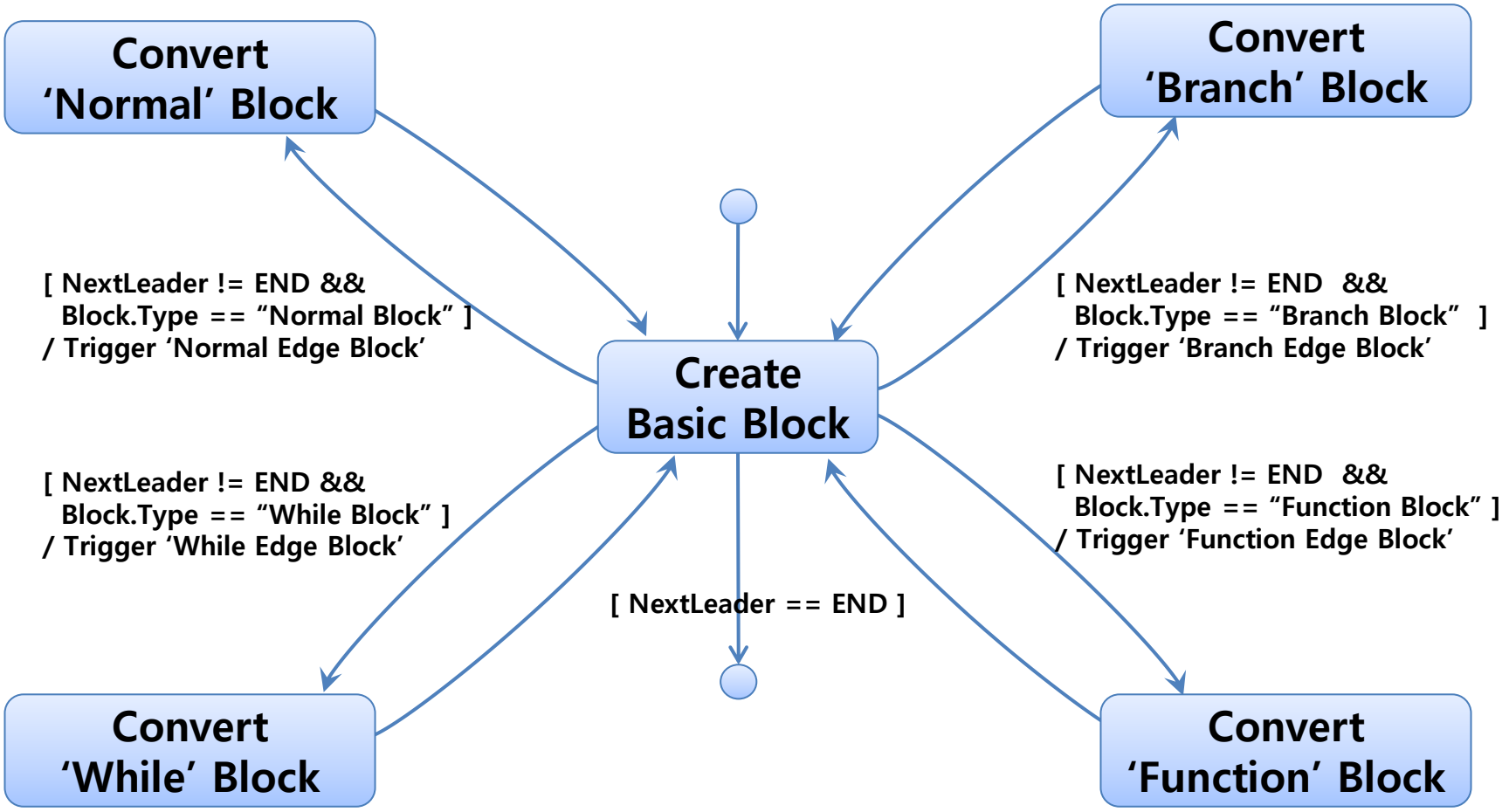
DFD Level 5

- Finite State Machine 1



DFD Level 5

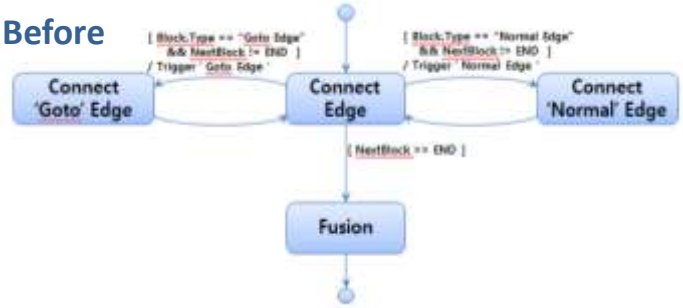
- Finite State Machine 2

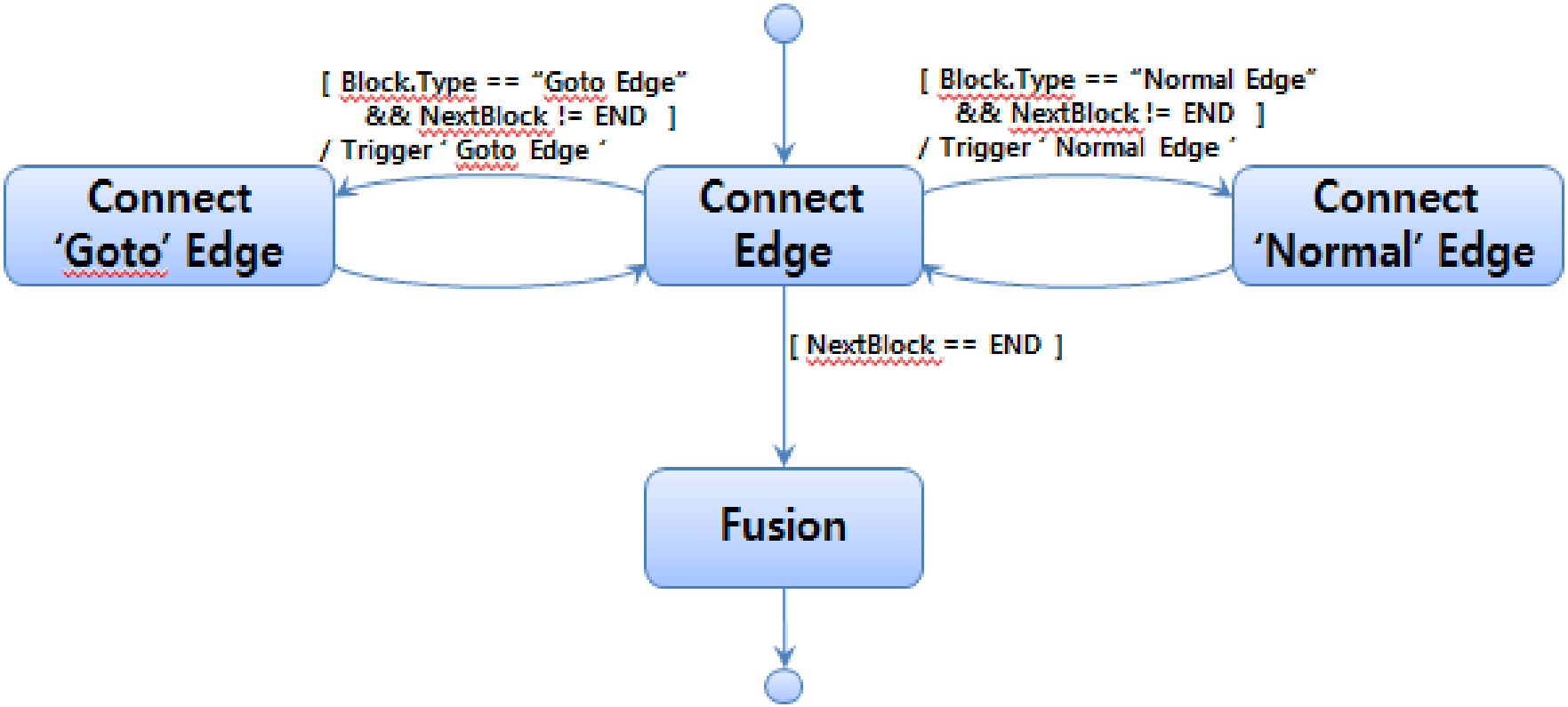


DFD Level 5

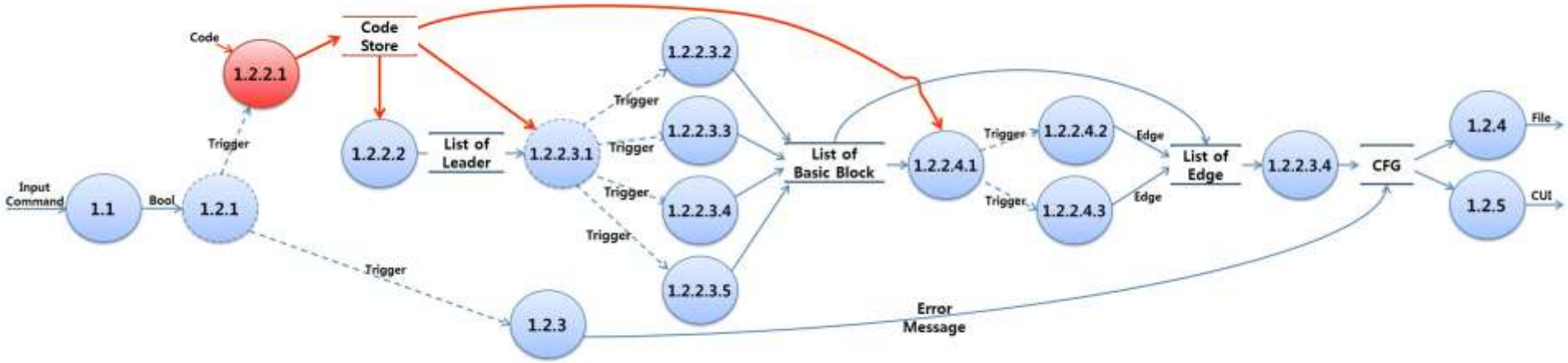
- Finite State Machine 3

Before

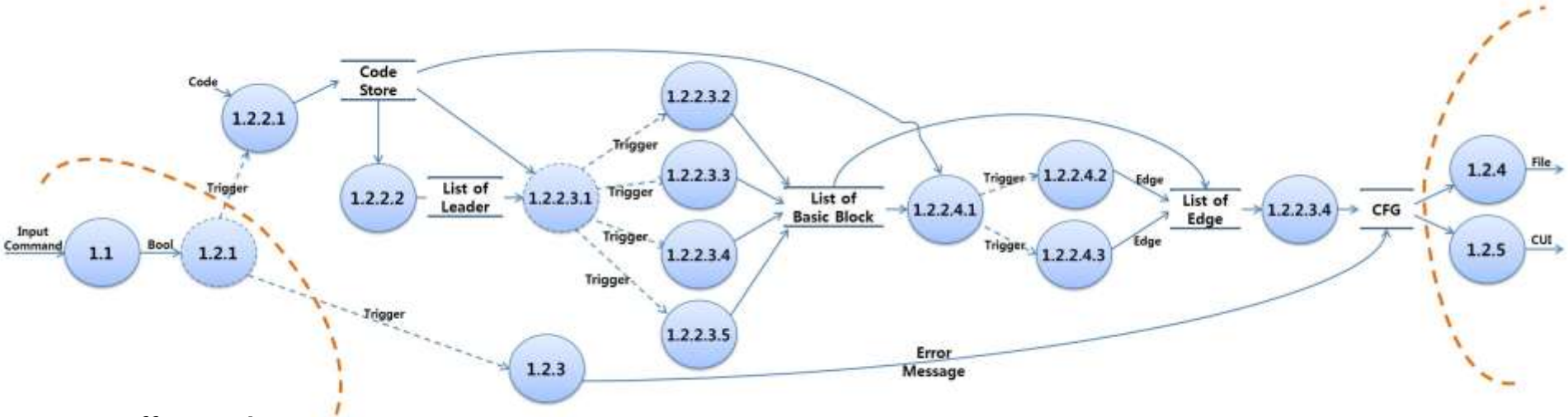




DFD - Overall


Input Command
1.1
Bool
1.2.1
Trigger
Code
1.2.2.1
Code Store
1.2.2.2
List of Leader
1.2.2.3.1
Trigger
Trigger
Trigger
Trigger
1.2.2.3.2
1.2.2.3.3
1.2.2.3.4
1.2.2.3.5
List of Basic Block
1.2.2.4.1
Trigger
Trigger
1.2.2.4.2
1.2.2.4.3
Edge
Edge
List of Edge
1.2.2.3.4
CFG
1.2.4
File
1.2.5
CUI
Trigger
1.2.3
Error Message

In DFD Overall

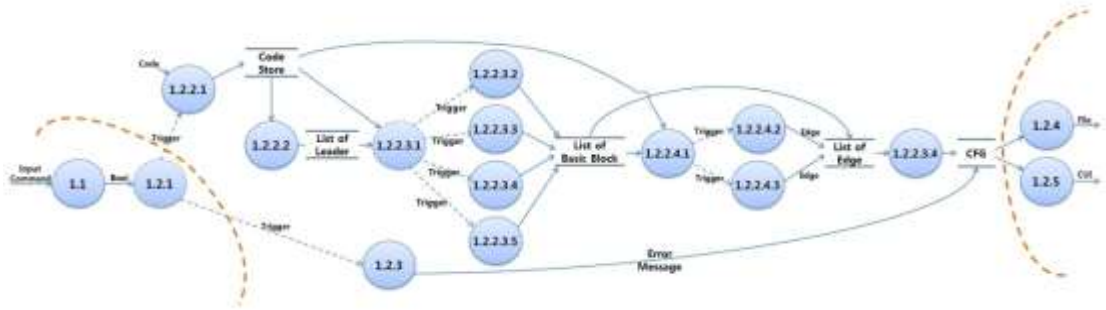


Afferent Flow (Input)

Central Transformation (Control)

Efferent Flow (Output)

Structured Chart(Basic)



main

Input command

True / False

True / False

Command Reader

CFG Generator

True

False

Controller

Error Message

Source

Source

List of Leader

List of Block

Source

List of Leader

List of Edge

List of Block

List of Edge

List of Block

CFG

CFG

File open

Determine Leader

Create of Basic Block

Connect Edge

Fusion BB n E

Trigger

Leader

Block

Node

Trigger

Leader

Block

Node

Trigger

Leader

Block

Node

Trigger

Leader

Block

Node

Trigger

Block

Edge

Trigger

Block

Edge

CFG

CFG

Normal Block

Branch Block

Repeat Block

Function Block

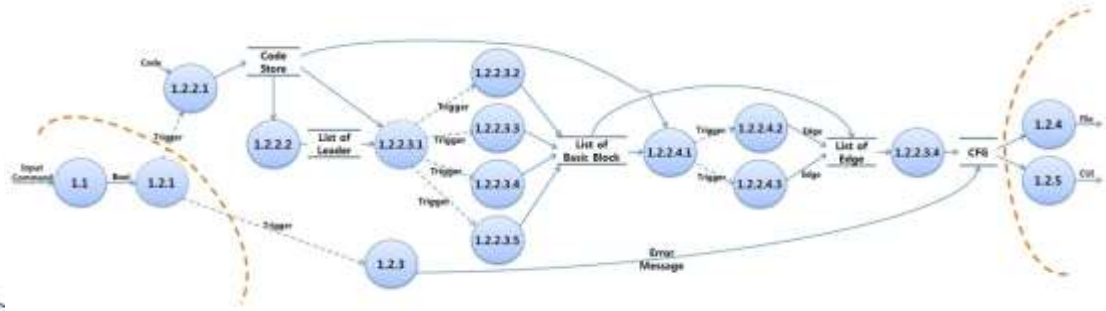
Goto Edge Writer

Down Edge Writer

Create File

Create CFG

Structured Chart(Advanced)



main
Input command
True / False
Command Reader
CFG Generator
True
False
Controller
Error Message
Source
Source
List of Leader
List of Block
Source
List of Leader
List of Edge
List of Block
List of Block
List of Edge
CFG
CFG
File open
Determine Leader
Create of Basic Block
Connect Edge
Fusion BB n E
Trigger
Leader
Block Node
Trigger
Leader
Block Node
Trigger
Leader
Block Node
Trigger
Leader
Block Node
Trigger
Block
Edge
Trigger
Block
Edge
CFG
CFG
Normal Block
Branch Block
Repeat Block
Function Block
Goto Edge Writer
Down Edge Writer
Create File
Create CFG

Source Code with SASD

Implementation

- Define enum and struct

◆ Leader

```
struct tag_Node_of_Leader
{
    int Leader_Num;
}typedef NodeLeader;
```

◆ Basic Block

```
typedef int Line;
struct tag_Node_of_Basic_Block
{
    int     BlockNumber;
    B_TYPE  Block_Type;
    Line    firstLine;
    Line    endLine;
    E_TYPE* Edges;
    int     NumofEdge;
}typedef NodeBlock;
```

◆ CFG

```
struct tag_CFG
{
    Bool            bSuccess;
    List*           blockList;
    List*           edgeList;
    char*           strErr;
}typedef CFG;
```

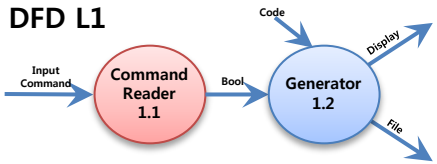
◆ Edge

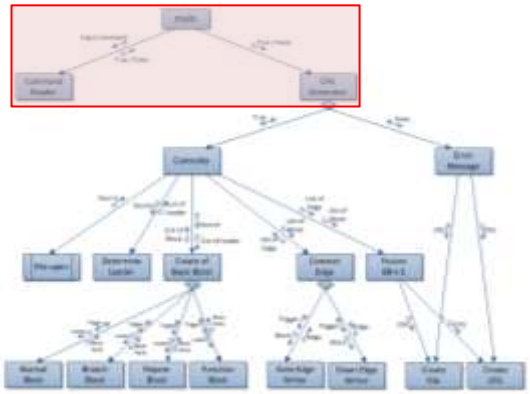
```
struct tag_Node_of_Edge
{
    int                 Edge_Number;
    Basic_Block*        Source;
    Basic_Block*        Destin;
}typedef NodeEdge;

enum tag_Edge_Type
{
    ET_NORMAL,
    ET_GOTO
}typedef E_TYPE;
```

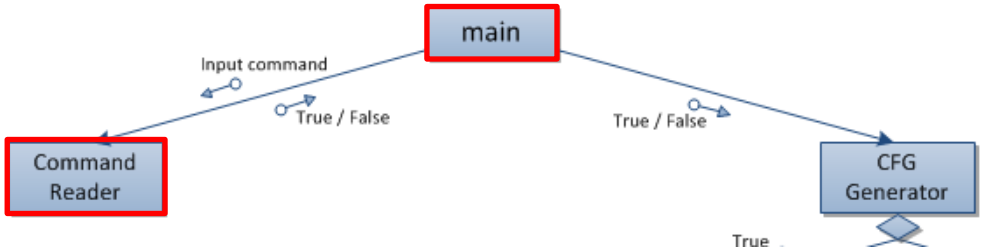
```
enum tag_Block_Type
{
    BT_NORMAL, // 0
    BT_BREAK, // 1

    BT_REPEAT_FOR, // 11
    BT_REPEAT_WHILE, // 12
    BT_FUNCTION // 13
}typedef B_TYPE;

enum tag_Repeat_Stack_Type
{
    RST_NOT_REPEAT,
    RST_FOR,
    RST_WHILE
}typedef RS_TYPE;
```

Implementation

- main , CommandReader





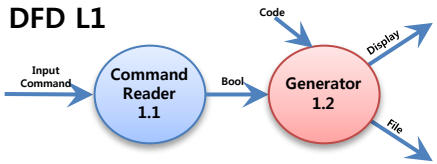


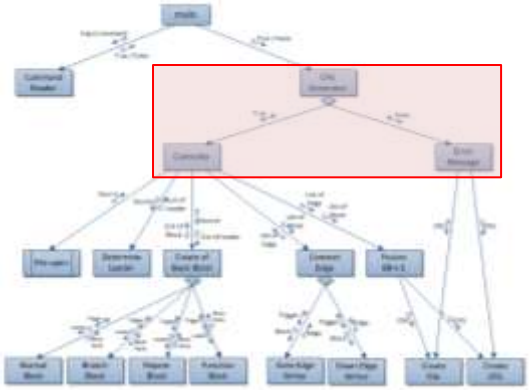
Reference No.	1.1
Name	Command Reader
Input	Command
Output	bool
Description	When executing the program after receiving the command, the **command checks the suitability and conformity** to the option value to output.

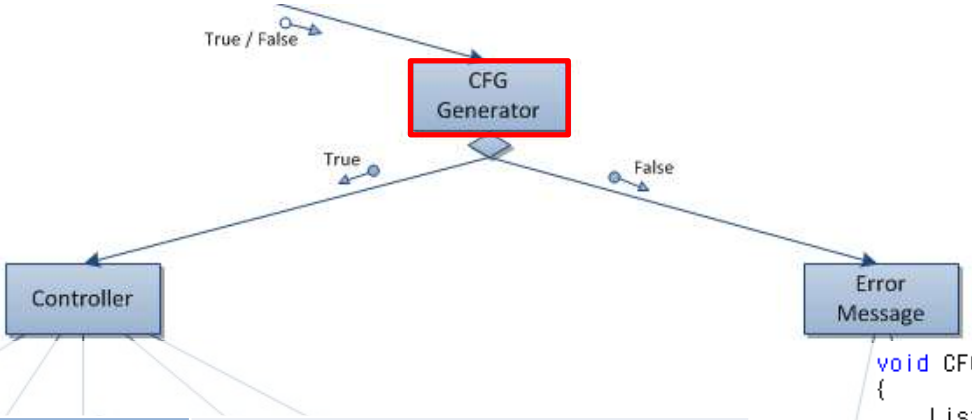
```
int main(int argc, char* argv[])
{
    Bool    bCommand;
    ERROR   error;

    bCommand = CommandReader(argc, argv, &error);
    CFG_Generator(bCommand);

    return 0;
}
Bool CommandReader(int argn, char* _inputCommand[], ERROR* error)
{
    long handle;
    FILE* fp;

    if (argn != 3)
    {
        *error = FILE_NOT_FOUND;
        return FALSE;
    }

    fp = fopen(_inputCommand[1], "r");

    if (fp == NULL)
    {
        *error = ARGUMENT_ERROR;
        return FALSE;
    }

    fclose(fp);
    return TRUE;
}
```

Implementation

- Generator





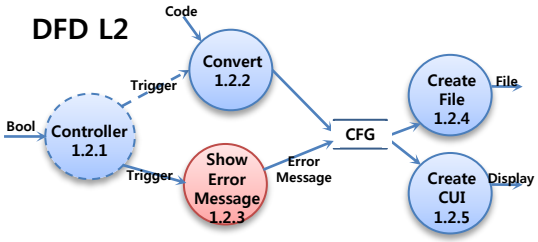


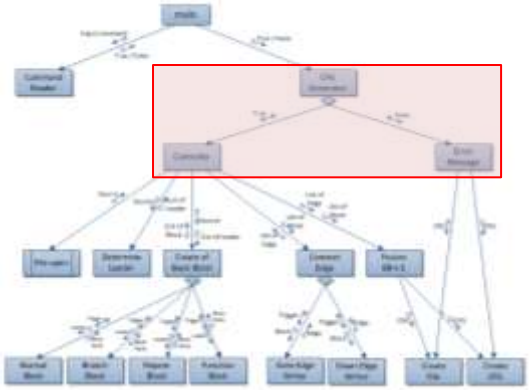
Reference No.	1.2
Name	Generator
Input	Bool
Output	CUI, File
Description	Depending on the **results of Command Check** reads **C-Source**, it will output a corresponding **message**.

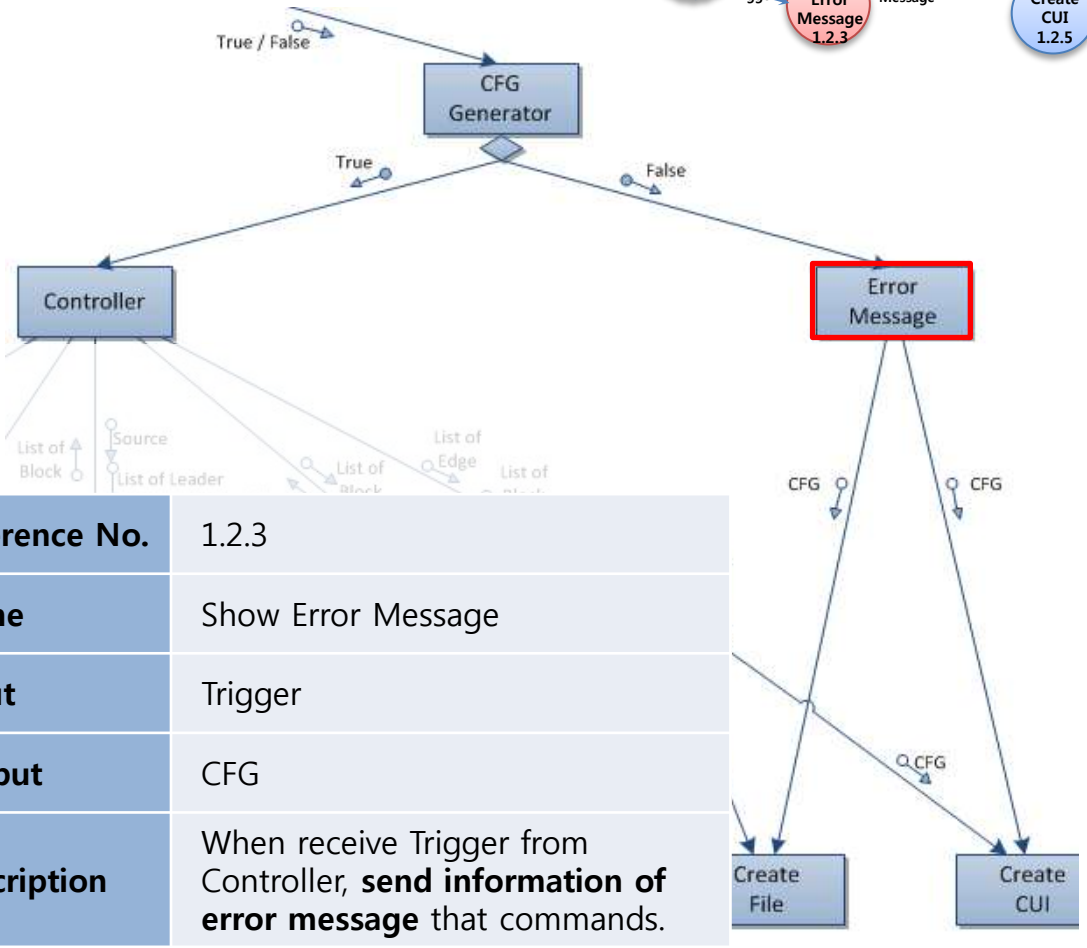
```
void CFG_Generater(char* filename, Bool _bInput)
{
    List* Leader_List = NULL;
    FILE* source;
    ERROR error;

    int i;
    int index = 0;

    if (_bInput == FALSE)
    {
        if (filename == NULL)   error = FILE_NOT_FOUND;
        else                    error = ARGUMENT_ERROR;

        ErrorMessage(_bInput, error);
    }
    else
    {
        source = fopen(filename, "r");
        Leader_List = Determine_Leader(source);
        Create_BasicBlock(source, Leader_List);
        Connect_Edge(source);
        g_CFG = *Fusion_BBnE(&g_ListofBlock, &g_ListofEdge);

    }

    Create_CUI(g_CFG);
    Create_File(g_CFG);
```

Implementation

- ErrorMessage





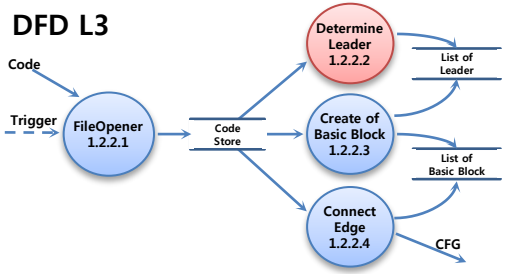


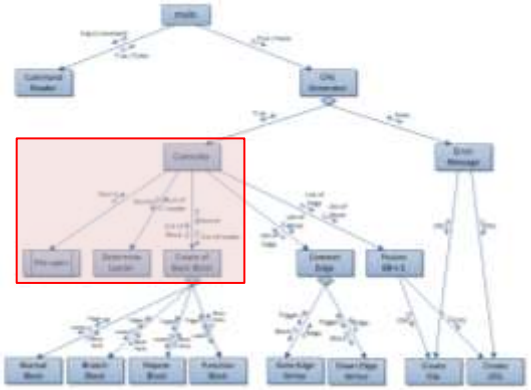
Reference No.	1.2.3
Name	Show Error Message
Input	Trigger
Output	CFG
Description	When receive Trigger from Controller, **send information of error message** that commands.

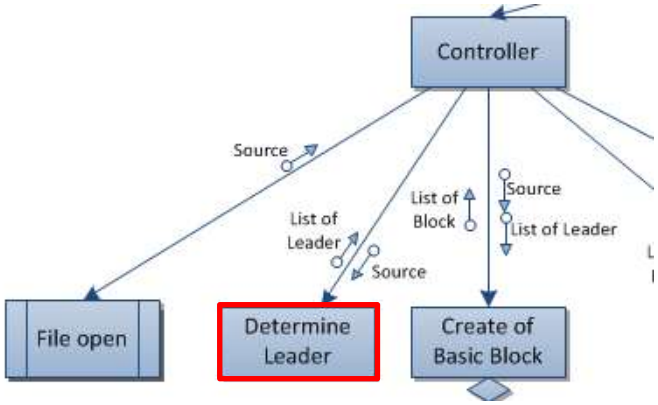
```
void ErrorMessage(Bool _false, ERROR error)
{
    char* messege;
    int len;
    if (error == FILE_NOT_FOUND)
    {
        len = strlen("file not found");
        messege = (char*)malloc(len + 1);
        strcpy(messege, "file not found");

    }
    else if (error == ARGUMENT_ERROR)
    {
        len = strlen("argument error");
        messege = (char*)malloc(len + 1);
        strcpy(messege, "argument error");
    }

    g_CFG.bError = _false;
    g_CFG.strErr = messege;

    printf("Command Error₩n");
    printf("Error Messege->cfg.txt₩n");
}
```

Implementation

- Determine Leader





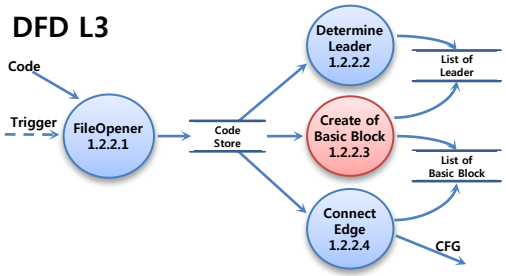


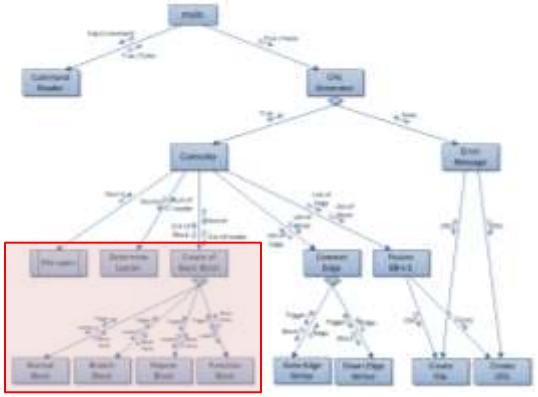
Reference No.	1.2.2.2
Name	Determine Leader
Input	Code Store(Source)
Output	List of Leader
Description	When receive a Code Store from File Opener, making **List of Leader** that finding each block's leader, **save a Data store and return**.

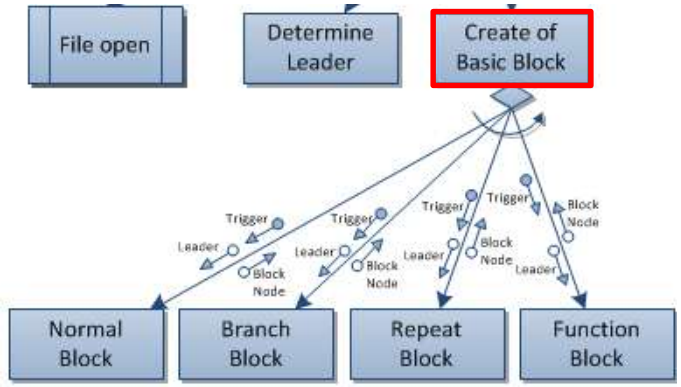
```
List* Determine_Leader(FILE* _source)
{
    //create list
    list = (List*)malloc(sizeof(List));
    memset(list, 0, sizeof(List));

    //main search
    while (fgets(szbuf, BUFFERSIZE, _source) != NULL)
    {
        if (strstr(szbuf, "main") != NULL)  break;

        //function search.....what the hell
        if (szbuf[0] != '#' && (strchr(szbuf, '(') ) && parenth == 0)
        {
            if (FUNCTION_NUM == function_count) {    }
            strtok(szbuf, " ");
            strcpy(function_Table[function_count], strtok(NULL, " ("));
            function_count++;
        }
        if (strchr(szbuf, '{')) parenth++;
        if (strchr(szbuf, '}')) parenth--;
        count++;
    }

    count = Search_Leader(list, _source, count, false);
    tempNode = list->Begin;
    return list;
}
```

Implementation

- Create Basic Block







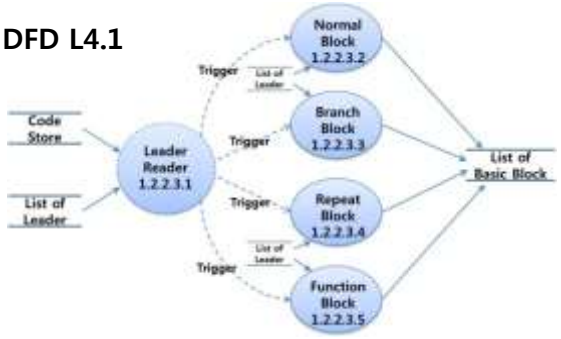
Reference No.	1.2.2.3
Name	Create Basic Block
Input	List of Leader, Code Store
Output	List of Basic Block
Description	**Receive leader's lists**, create basic block, **make List** and so on **save at CFG Data store**.

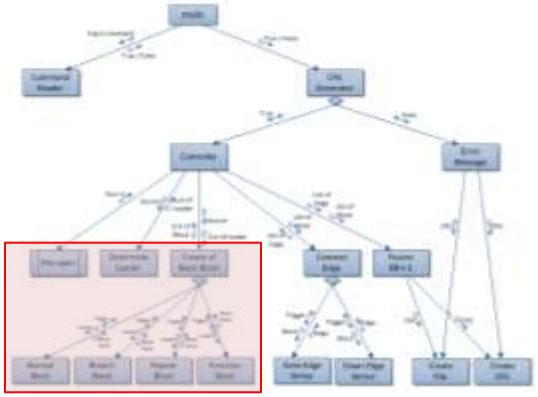
```
Basic_Block*    Create_BasicBlock(FILE* _source, List* _leader)
{
    Node*       TestNode;

    g_ListofBlock.Begin = NULL;  // g_ListofBlock 초기화.
    Stack_for_RST.Begin = NULL;

    while(1)
    {
        nFlag = 0;  // Flag 리셋.

        if(((NodeLeader*)List_Get(_leader,Number)) == NULL) break;
        memset(szbuf,0,BUFFERSIZE);                    // initialize buffer
        fseek(_source,0L,SEEK_SET);                    // initialize file location

        for(j = 0; j < ((NodeLeader*)List_Get(_leader,Number))->Leader_Num ; j++)
        {
            fgets(szbuf,BUFFERSIZE,_source);           // read a line
        }

        //branch..
        if((strstr(szbuf,"else ") != NULL || strstr(szbuf,"else{") != NULL
              || strstr(szbuf,"else₩n") != NULL) && nFlag == 0)     // if Leader is else
        {
            if(strstr(szbuf,"if") != NULL)  // check if statement
            {
                Branch_Block(BT_BRANCH_ELSE_IF,((Node*)List_Get_Index(_leader,Number)));
            }
            else
            {
                Branch_Block(BT_BRANCH_ELSE,((Node*)List_Get_Index(_leader,Number)));
            }
            nFlag = 1;
        }

        if((strstr(szbuf,"switch ") != NULL && nFlag == 0)
              || (strstr(szbuf,"switch(") != NULL && nFlag == 0))
        {
            ...
        }
    }
```

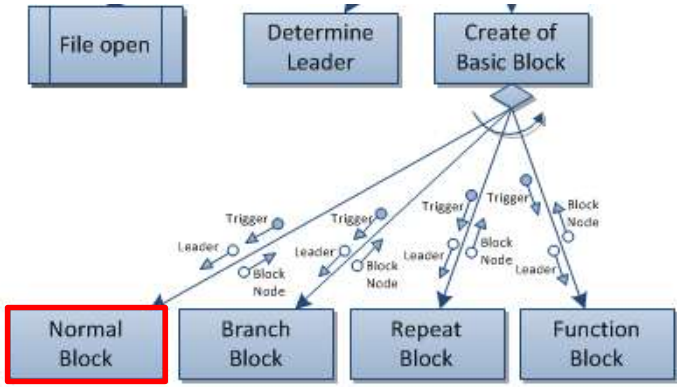
Implementation

- Normal_Block

DFD L4.1







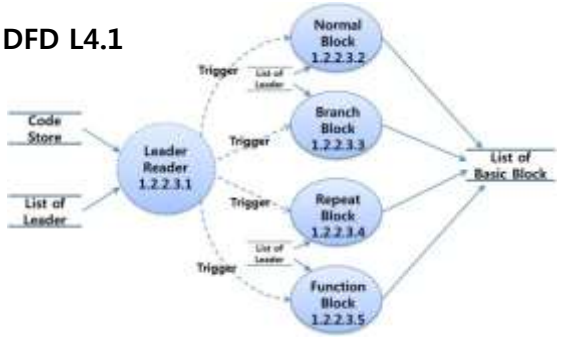
Reference No.	1.2.2.3.2
Name	Normal Block
Input	List of Leader, Trigger
Output	Basic Block Node
Description	When you receive a Node of Leader, make Normal Block and add list of Basic Block as a node.

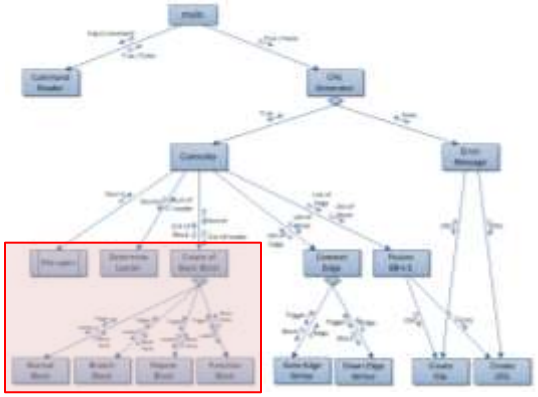
```
NodeBlock*      Normal_Block(Bool _trigger, Node* _Node)
{
    int i = 0;

    NodeBlock*  TempBlock;
    TempBlock = (NodeBlock*)malloc(sizeof(NodeBlock));
    TempBlock->Edges = NULL;
    TempBlock->firstLine = ((NodeLeader*)(_Node->pData))->Leader_Num;
    if(_Node->pNext == NULL) TempBlock->endLine = -1;
    else TempBlock->endLine = ((NodeLeader*)(_Node->pNext->pData))->Leader_Num -1;


    switch(_trigger)
    {
    case BT_NORMAL:
        TempBlock->NumofEdge = 1;
        TempBlock->Edges = (E_TYPE*)malloc(sizeof(E_TYPE)*TempBlock->NumofEdge);
        TempBlock->Edges[0] = ET_NORMAL;
        TempBlock->Block_Type = _trigger;
        break;

    default:
        TempBlock->NumofEdge = 0;
        printf("Trigger Err₩n");
        break;
    }

    List_Add_Last(&g_ListofBlock,TempBlock);
```

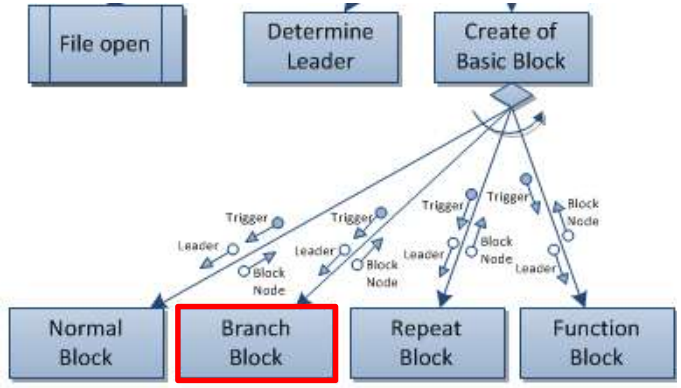
Implementation

- Branch_Block

DFD L4.1







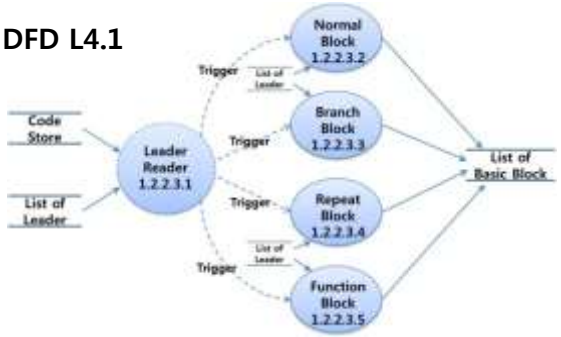
Reference No.	1.2.2.3.3
Name	Branch Block
Input	List of Leader, Trigger
Output	Basic Block Node
Description	When you receive a Node of Leader, make Branch Block and add list of Basic Block as a node.

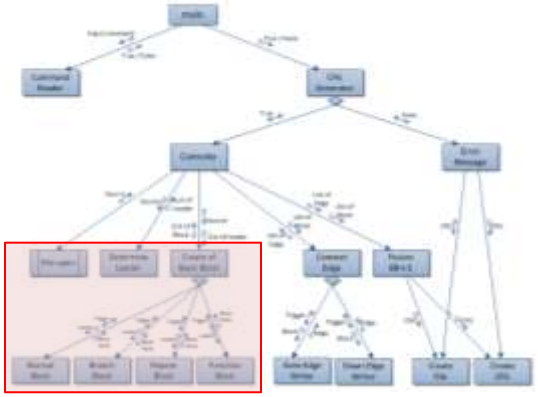
```
NodeBlock*      Branch_Block(Bool _trigger, Node* _Node)
{
    int i = 0;

    NodeBlock*  TempBlock;
    TempBlock = (NodeBlock*)malloc(sizeof(NodeBlock));
    TempBlock->Edges = NULL;
    TempBlock->firstLine = ((NodeLeader*)(_Node->pData))->Leader_Num;
    if(_Node->pNext == NULL) TempBlock->endLine = -1;
    else TempBlock->endLine = ((NodeLeader*)(_Node->pNext->pData))->Leader_Num -1;


    switch(_trigger)
    {
    case BT_BRANCH_IF:
        TempBlock->NumofEdge = 2;
        TempBlock->Edges = (E_TYPE*)malloc(sizeof(E_TYPE)*TempBlock->NumofEdge);
        TempBlock->Edges[0] = ET_GOTO;
        TempBlock->Edges[1] = ET_NORMAL;
        TempBlock->Block_Type = _trigger;
        break;
    case BT_BRANCH_ELSE:
        TempBlock->NumofEdge = 2;
        TempBlock->Edges = (E_TYPE*)malloc(sizeof(E_TYPE)*TempBlock->NumofEdge);
        TempBlock->Edges[0] = ET_NORMAL;
        TempBlock->Edges[1] = ET_GOTO;
        TempBlock->Block_Type = _trigger;
        break;
    case BT_BRANCH_ELSE_IF:

    case BT_BRANCH_SWITCH:

    case BT_BRANCH_CASE:

    case BT_BRANCH_DEFAULT:

    default:
        TempBlock->NumofEdge = 0;
        printf("Trigger Err₩n");
        break;
    }
    List_Add_Last(&g_ListofBlock,TempBlock);
}
```

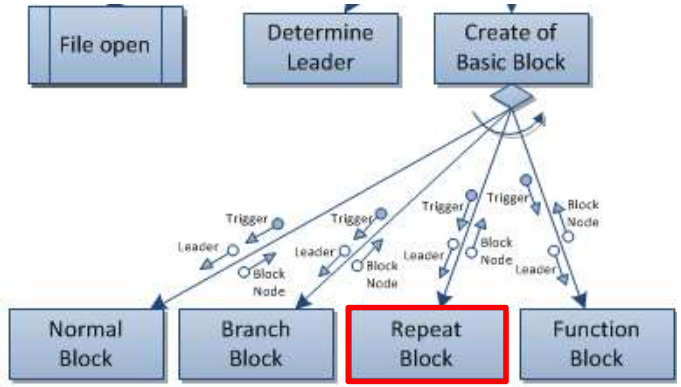
Implementation

- Repeat_Block

DFD L4.1







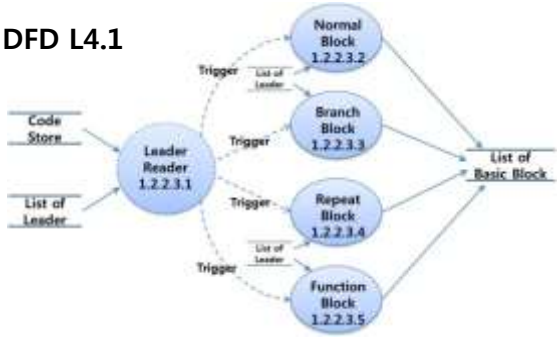
Reference No.	1.2.2.3.4
Name	Repeat Block
Input	List of Leader, Trigger
Output	Basic Block Node
Description	When you receive a Node of Leader, make RepeatBlock and add list of Basic Block as a node.

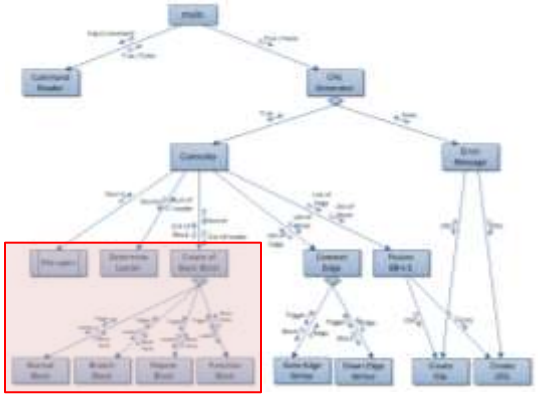
```
NodeBlock*      Repeat_Block(Bool _trigger, Node* _Node)
{
    int i = 0;

    NodeBlock*  TempBlock;
    TempBlock = (NodeBlock*)malloc(sizeof(NodeBlock));
    TempBlock->Edges = NULL;
    TempBlock->firstLine = ((NodeLeader*)(_Node->pData))->Leader_Num;
    if(_Node->pNext == NULL) TempBlock->endLine = -1;
    else TempBlock->endLine = ((NodeLeader*)(_Node->pNext->pData))->Leader_Num -1;


    switch(_trigger)
    {
    case BT_REPEAT_FOR:
        TempBlock->NumofEdge = 2;
        TempBlock->Edges = (E_TYPE*)malloc(sizeof(E_TYPE)*TempBlock->NumofEdge);
        TempBlock->Edges[0] = ET_GOTO;
        TempBlock->Edges[1] = ET_NORMAL;
        TempBlock->Block_Type = _trigger;
        break;
    case BT_REPEAT_WHILE:
        TempBlock->NumofEdge = 2;
        TempBlock->Edges = (E_TYPE*)malloc(sizeof(E_TYPE)*TempBlock->NumofEdge);
        TempBlock->Edges[0] = ET_NORMAL;
        TempBlock->Edges[1] = ET_GOTO;
        TempBlock->Block_Type = _trigger;
        break;
    default:
        TempBlock->NumofEdge = 0;
        printf("Trigger Err₩n");
        break;
    }
    List_Add_Last(&g_ListofBlock,TempBlock);
    printf("₩nRepeat₩n");
}
```

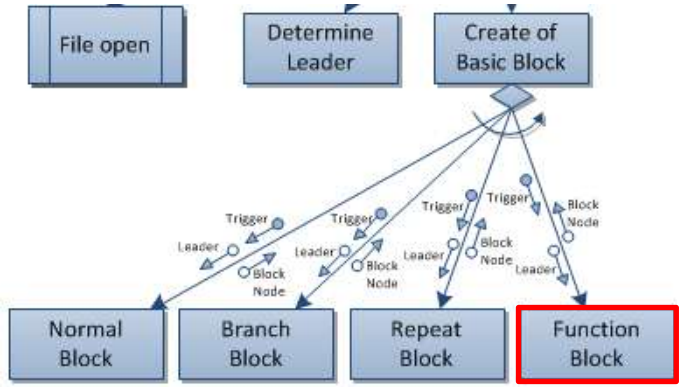
Implementation

- Function Block

DFD L4.1





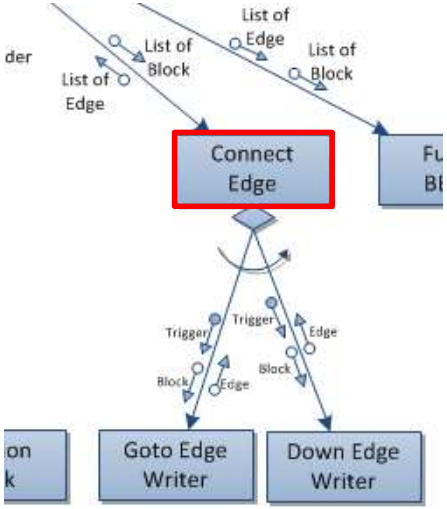


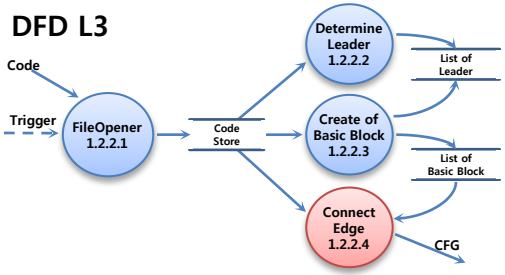
Reference No.	1.2.2.3.5
Name	Function Block
Input	List of Leader, Trigger
Output	Basic Block Node
Description	When you receive a trigger, processing of the function block and as the form of CFG Phase Data is exported.

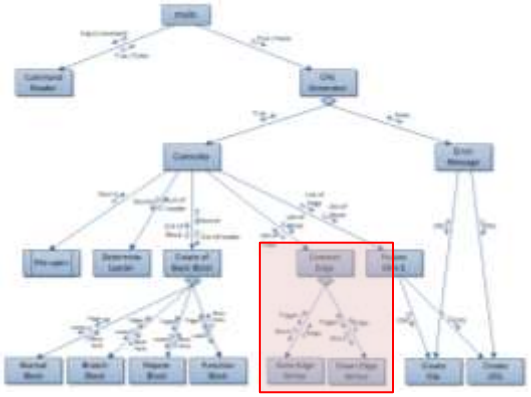
```
NodeBlock*      Function_Block(Bool _trigger, Node* _Node)
{
    int i = 0;

    NodeBlock*  TempBlock;
    TempBlock = (NodeBlock*)malloc(sizeof(NodeBlock));
    TempBlock->Edges = NULL;
    TempBlock->firstLine = ((NodeLeader*)(_Node->pData))->Leader_Num;
    if(_Node->pNext == NULL) TempBlock->endLine = -1;
    else TempBlock->endLine = ((NodeLeader*)(_Node->pNext->pData))->Leader_Num -1;

    switch(_trigger)
    {
    case BT_FUNCTION:
        TempBlock->NumofEdge = 1;
        TempBlock->Edges = (E_TYPE*)malloc(sizeof(E_TYPE)*TempBlock->NumofEdge);
        TempBlock->Edges[0] = ET_NORMAL;
        TempBlock->Block_Type = _trigger;
        break;
    default:
        TempBlock->NumofEdge = 0;
        printf("Trigger Err\n");
        break;
    }

    List_Add_Last(&g_ListofBlock,TempBlock);

    printf("\nFunction\n");
}
     return TRUE;
}
```

Implementation

- Connect Edge







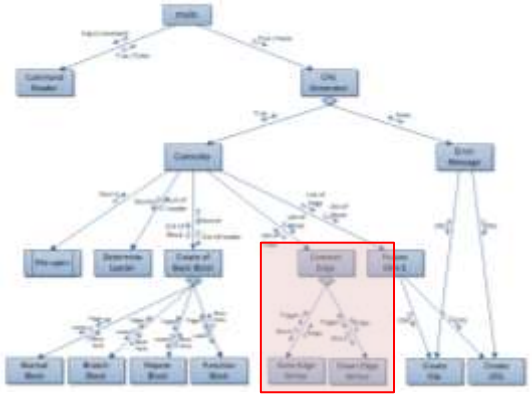
Reference No.	1.2.2.4
Name	Connect Edge
Input	List of Basic Block, Code Store
Output	CFG
Description	Receive list of Basic Block, make Edges, and save CFG to data store.

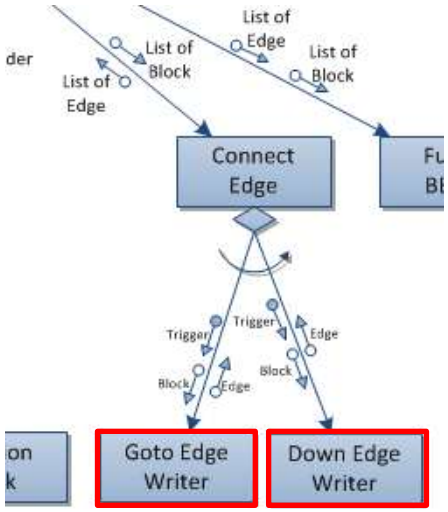
```
Edge*   Connect_Edge(FILE* source, List _block)
{
    List g_ListofEdge;

    int tmp_Block_Edge_Nums;// temp block.. number of edge its Block
    int Array_Index = 0;

    while(1) // about Block
    {
        // Number of edge its Block
        tmp_Block_Edge_Nums = ((NodeBlock*)List_Get(&g_ListofBlock, Block_Number))->NumofEdge;

        while(1) // about Edge
        {
            if ( ((NodeBlock*)List_Get(&g_ListofBlock, Block_Number))->Edges[Array_Index] == 0 )
                Down_Edge_Writer( );              // Down Edge라면
            else if (((NodeBlock*)List_Get(&g_ListofBlock, Block_Number))->Edges[Array_Index] == 1 )
                Goto_Edge_Writer(source);         // Goto Edge라면

            tmp_Block_Edge_Nums = tmp_Block_Edge_Nums - 1; // drawed edge.. so edge-1
            if ( tmp_Block_Edge_Nums == 0 ) // if next_edge is null
                break; // end!
            else
                Array_Index++; // to find next edge .. so edge+1
        }
        Array_Index = 0;
        Block_Number++;

        if ( ((NodeBlock*)List_Get(&g_ListofBlock, Block_Number)) == NULL ) // if next block is null
            break;
    }
    return g_ListofEdge; // Return Edge List
}
```

Implementation

- Goto Edge Writer
- Down Edge Writer

DFD L4.2





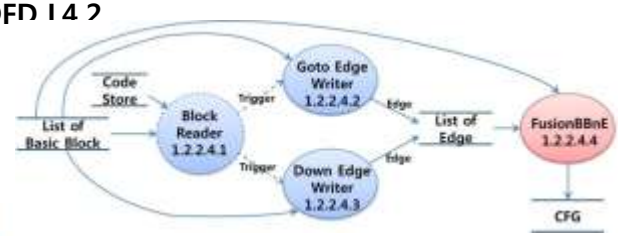

```
void Goto_Edge_Writer(FILE* source, List* g_ListofEdge, List _block)
{
    /*************** BRANCH BLOCK에 대한 처리 *****************/

    if ( ((NodeBlock*)List_Get(&g_ListofBlock, Block_Number))->Block_Type == BT_BRANCH_IF ) // if문의
        if_processor(source);

    else if ( ((NodeBlock*)List_Get(&g_ListofBlock, Block_Number))->Block_Type == BT_BRANCH_SWITCH )
        switch_processor(source);

    /*************** REPEAT BLOCK에 대한 처리 *****************/

    else if ( ((NodeBlock*)List_Get(&g_ListofBlock, Block_Number))->Block_Type == BT_REPEAT_FOR ) //
        for_processor(source);

    else if ( ((NodeBlock*)List_Get(&g_ListofBlock, Block_Number))->Block_Type == BT_REPEAT_WHILE )
        while_processor(source);
```

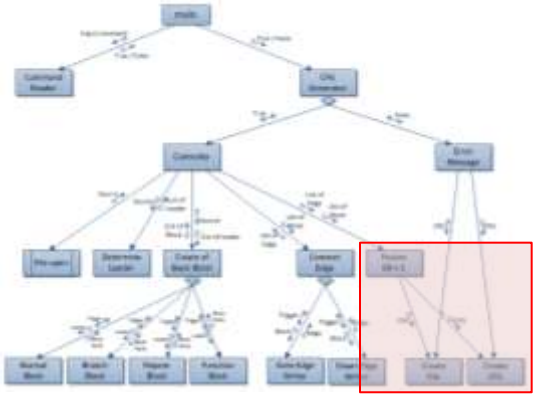
Reference No.	1.2.2.4.2	Reference No.	1.2.2.4.3
Name	Goto Edge Writer	Name	Down Edge Writer
Input	List of Basic Block, Trigger	Input	List of Basic Block, Trigger
Output	CFG	Output	CFG
Description	When you receive a trigger and List of Basic Block, this process makes **Goto Edge statements** each of Basic Block.	Description	When you receive a trigger and List of Basic Block, this process makes **Down Edge statements** each of Basic Block.

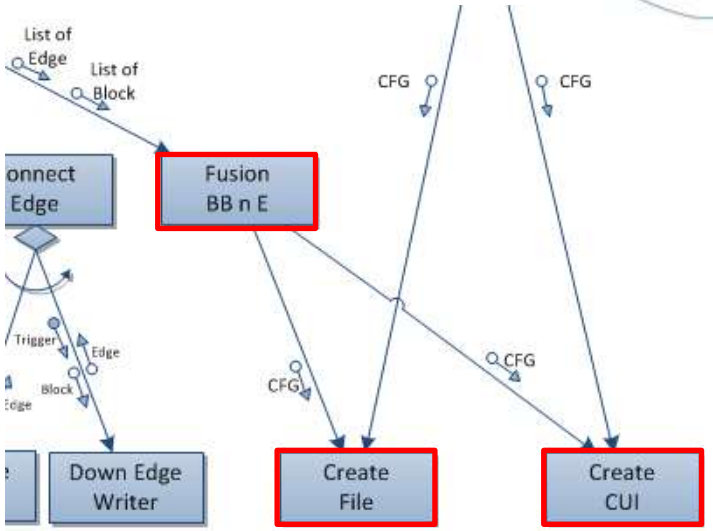
Implementation

- FusionBBnE

DFD 1.4.2







Reference No.	1.2.2.4.4
Name	FusionBBnE
Input	List of Edge, List of Basic Block
Output	CFG
Description	When you receive List of Block and Edge, this process save CFG by the order

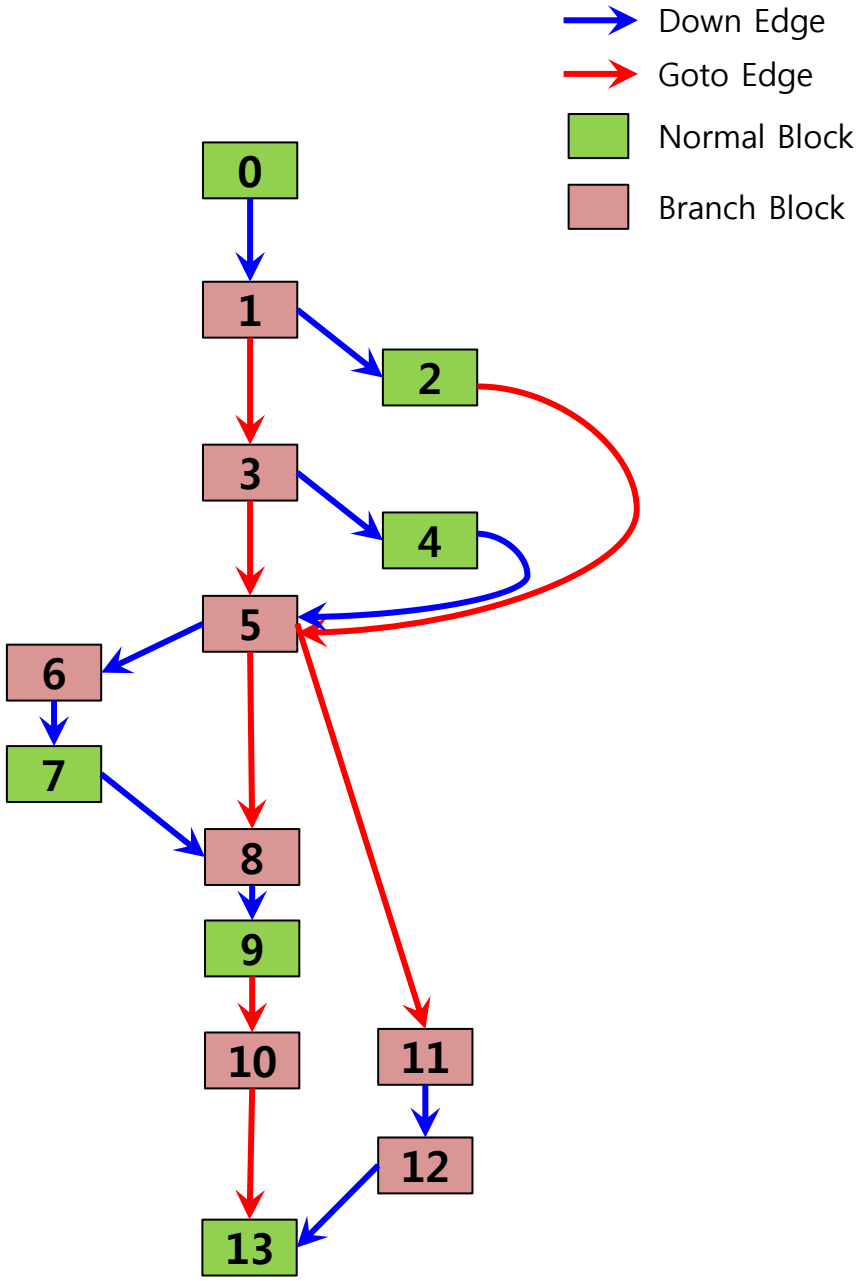
Reference No.	1.2.4
Name	Create File
Input	CFG

```
CFG Fusion_BBnE(List* BlockList, List* EdgeList)
{
    CFG cfg;
    cfg.blockList = BlockList;
    cfg.edgeList = EdgeList;
    cfg.bSuccess = TRUE;
    cfg.strErr = NULL;

    return cfg;
}
```

Name	Create CUI
Input	CFG
Output	Display
Description	CUI Data received a comprehensive data is output to the console screen.

Generating Example

```
#include <stdio.h>

int main(void)
{
    int direction = 1;                          0
    int x = 1, speed, n;
    int y = 1, length = 0;

    if (direction == 0)                         1
    {
        x++ ;                                   2
    }
    else if (direction == 1                     3
    {
        x-- ;                                   4
    }
    switch (length)                             5
    {
        case 1:                                 6
            speed = 100 ;
            n = 3 ;                             7

        case 2:                                 8
            gotoxy (MAX_X/2 , MAX_Y/2) ;
            printf("Game Clear ₩n") ;
            Sleep ( 300 ) ;                     9
            break ;                             10

        default:                                11
            break;                              12
    }
    return 0;                                   13
}
```



Demo !!!

Thank You!!