FINAL PRESENTATION: INTRODUCTION TO SOFTWARE ENGINEERING

Control Flow Graph GENERATOR

Group 5 & 6

201011351 So Yeon LEE, 201011374 Seo Hui HA,
201160417 Bjarke LARSEN, 201160526 Jesse ONG PHO

Agenda

- The Review Process
  - Statement of purpose
  - DFD Modifications
    - Modified DFD Level 1
    - Modified DFD Level 2
    - Modified DFD Level 2 (Cont.)
    - Modified DFD Level 3
    - Original DFD Level 4
    - Modified DFD Level 4 (FSM)
    - Total DFD
    - Data Dictionary
- Explanation of source code
  - Process Specification
- Demonstration

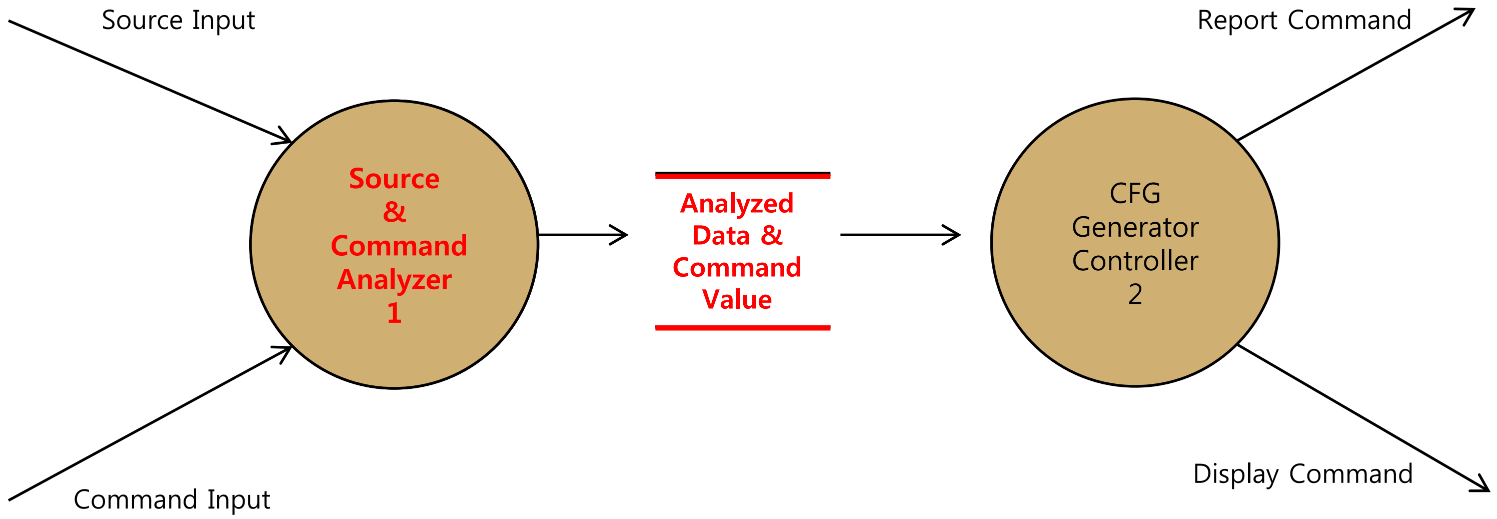
The Review Process

Statement of purpose

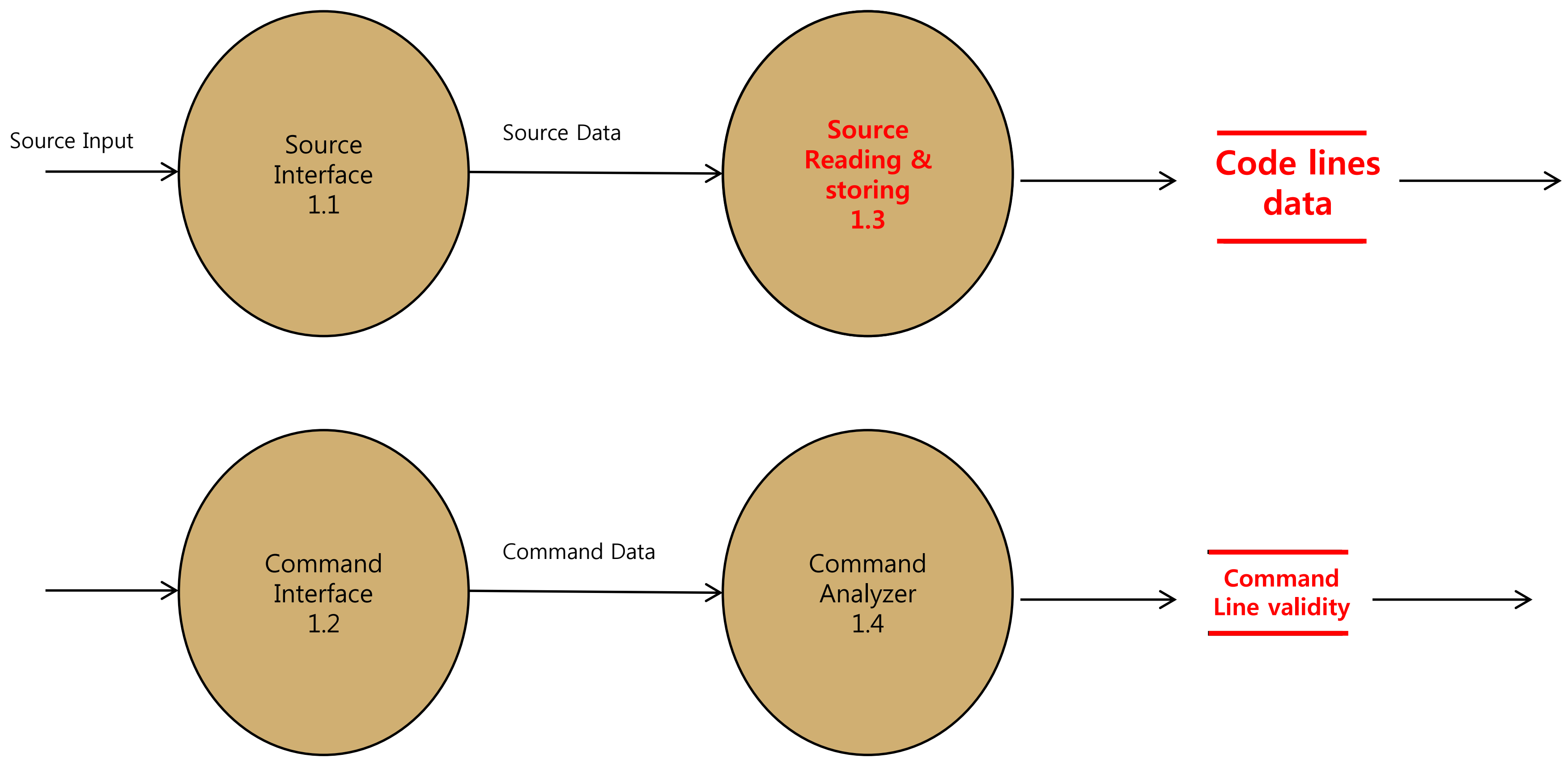
Draw a Control Flow Graph (CFG):

- 1. convert C source code to the CFG in text format to the console
- 2. Read in command line from user
- format: ./CG (example.c) (report.txt)
- 3.if user gave wrong format of command line,
- show help message and just end the program
- 4. show user the generating process of CFGs
- 5.notice user the start of converting
- 6. show whether the source reading was successful or not.
- if the codes were successfully read , show success message.
- if not, show error message and exit the program
- 7. Create a report file by listing all of the edges and blocks that were generated by this program according to the C source code
- 8. show report file name at last.
- 9.convert c source that contains a main function
- 10. Any user-libraries are not to be included.
- 11. Only applied to one source code file at a time that does not include any pointer data or so.

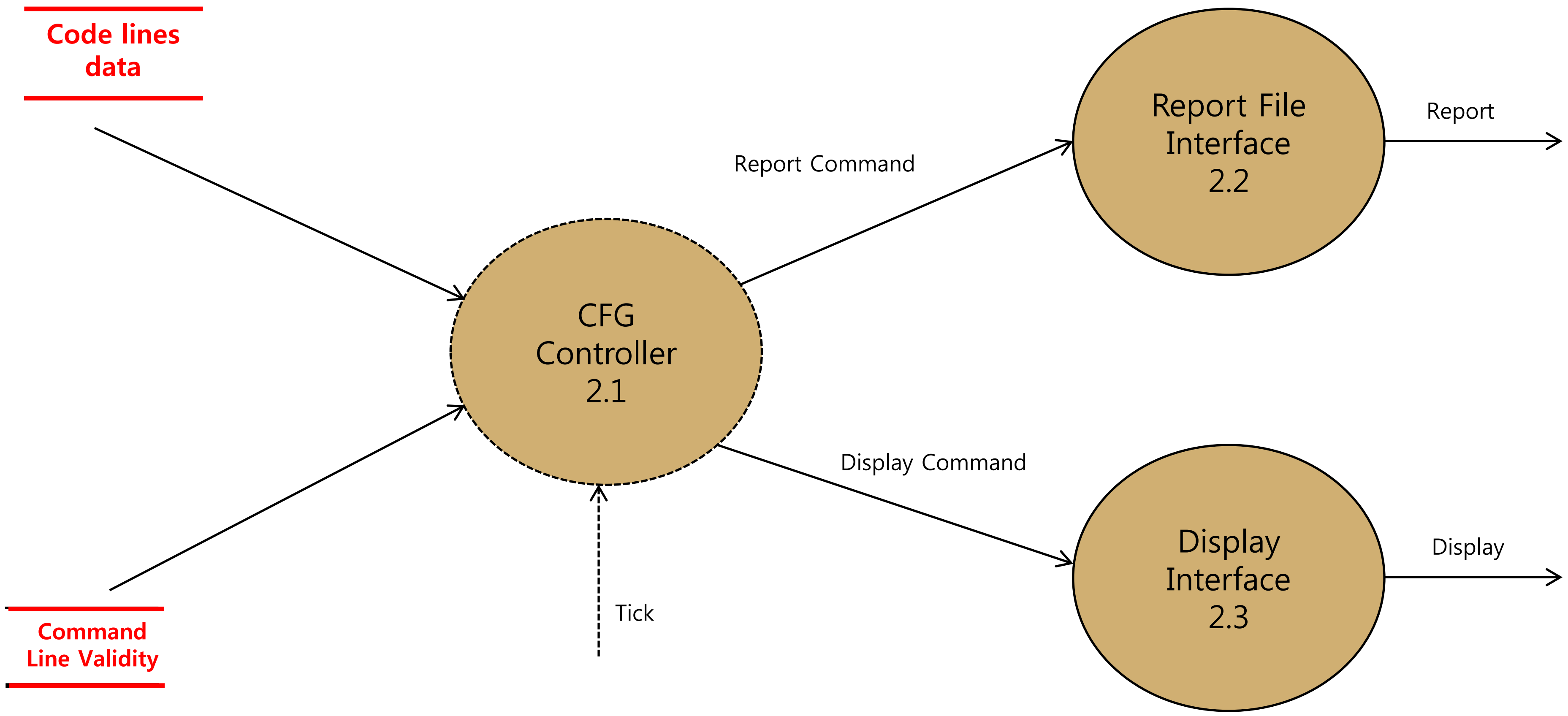
Modified DFD Level 1


Source Input
Command Input
Source & Command Analyzer 1
Analyzed Data & Command Value
CFG Generator Controller 2
Report Command
Display Command

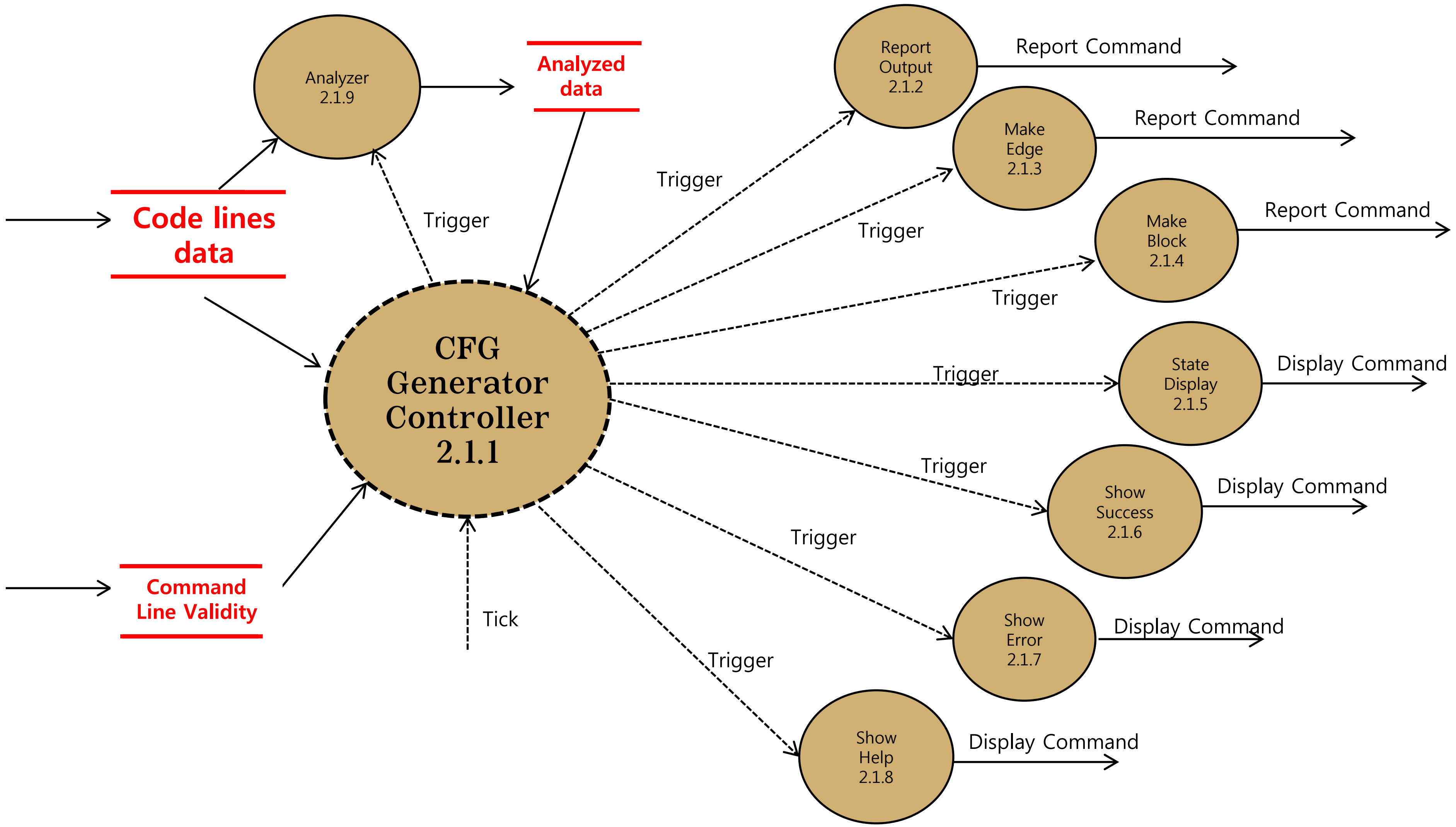
Modified DFD Level 2



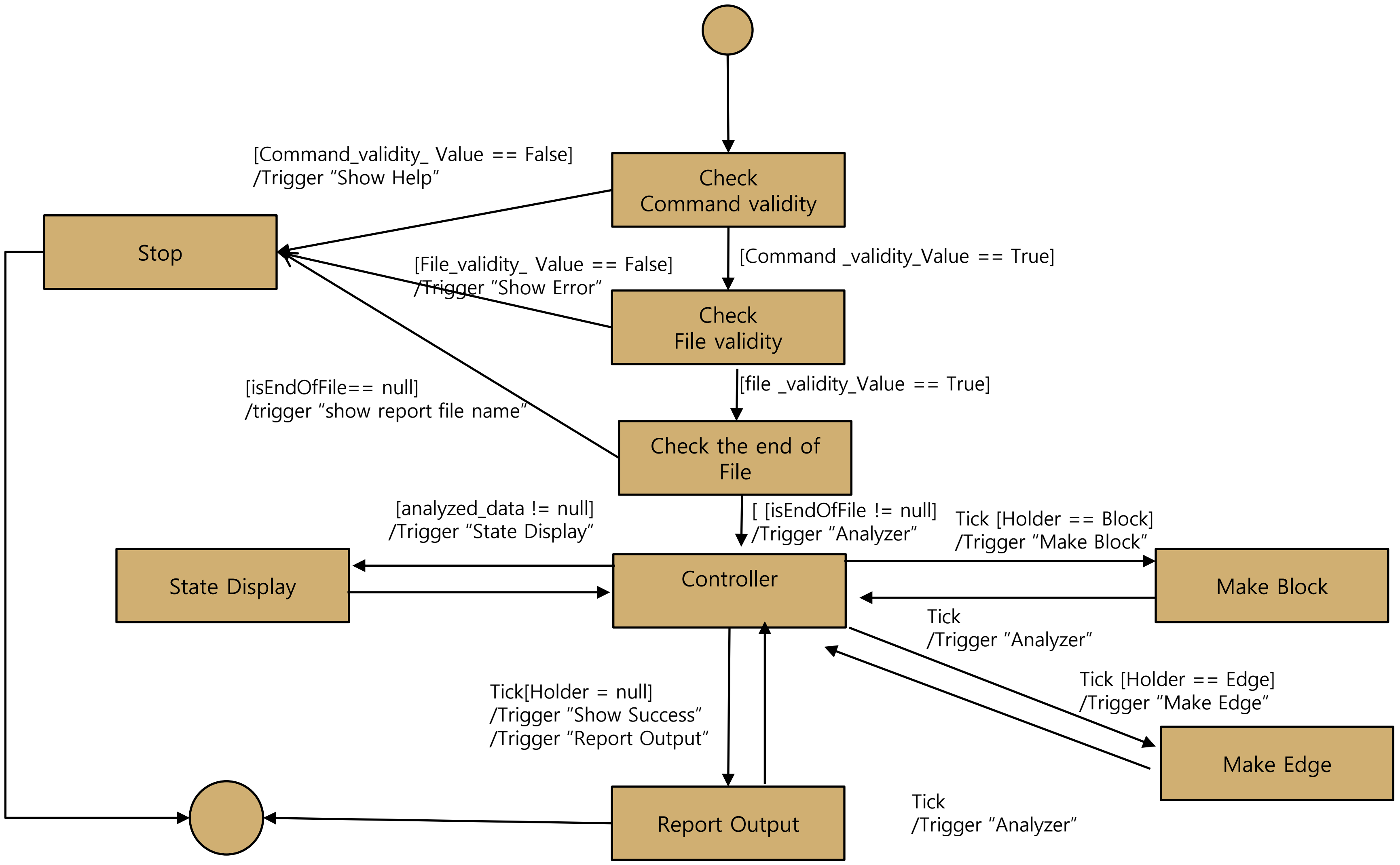
Modified DFD Level 2 (cont.)



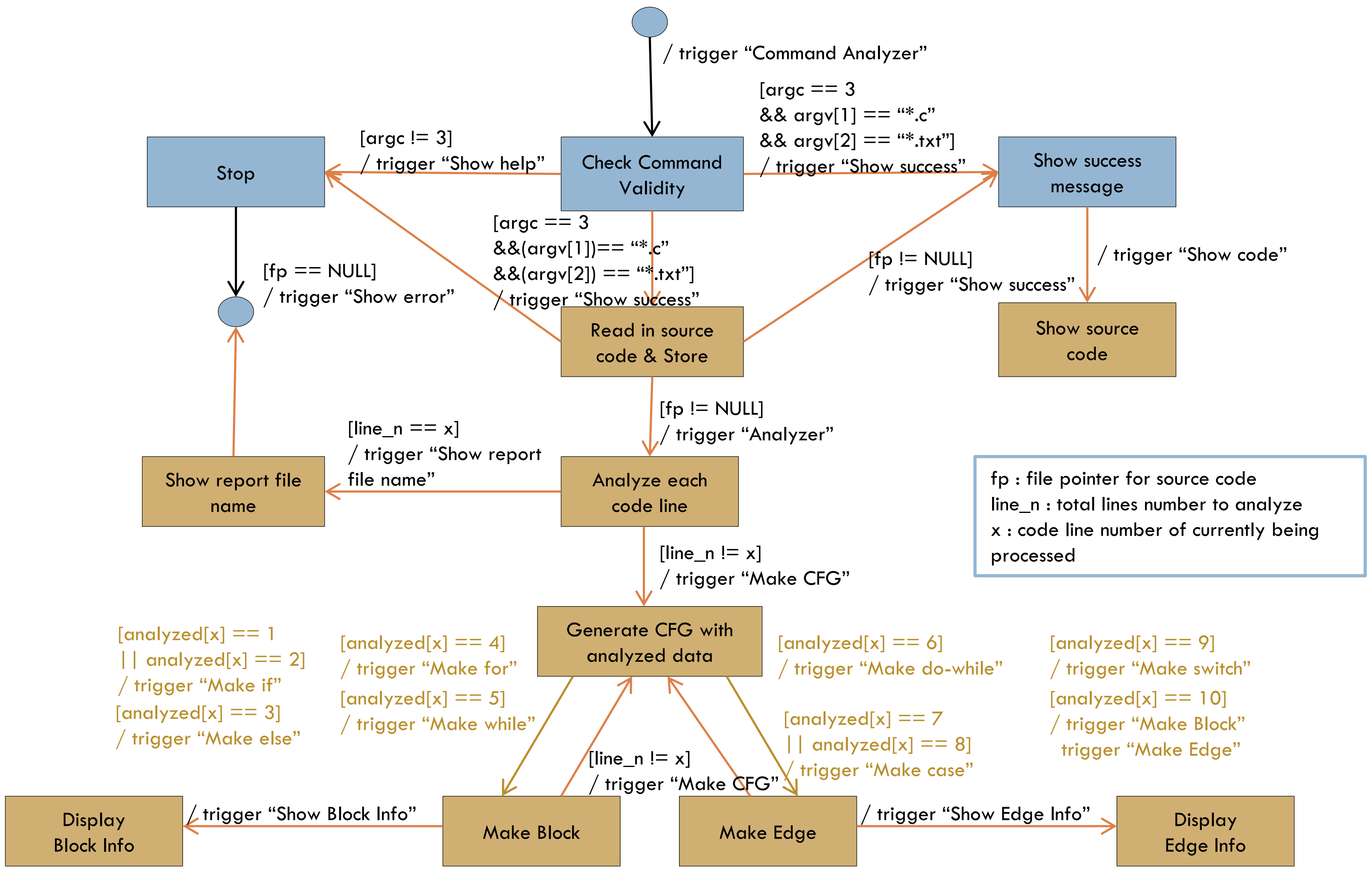
Modified DFD Level 3



Original DFD Level 4 (FSM)

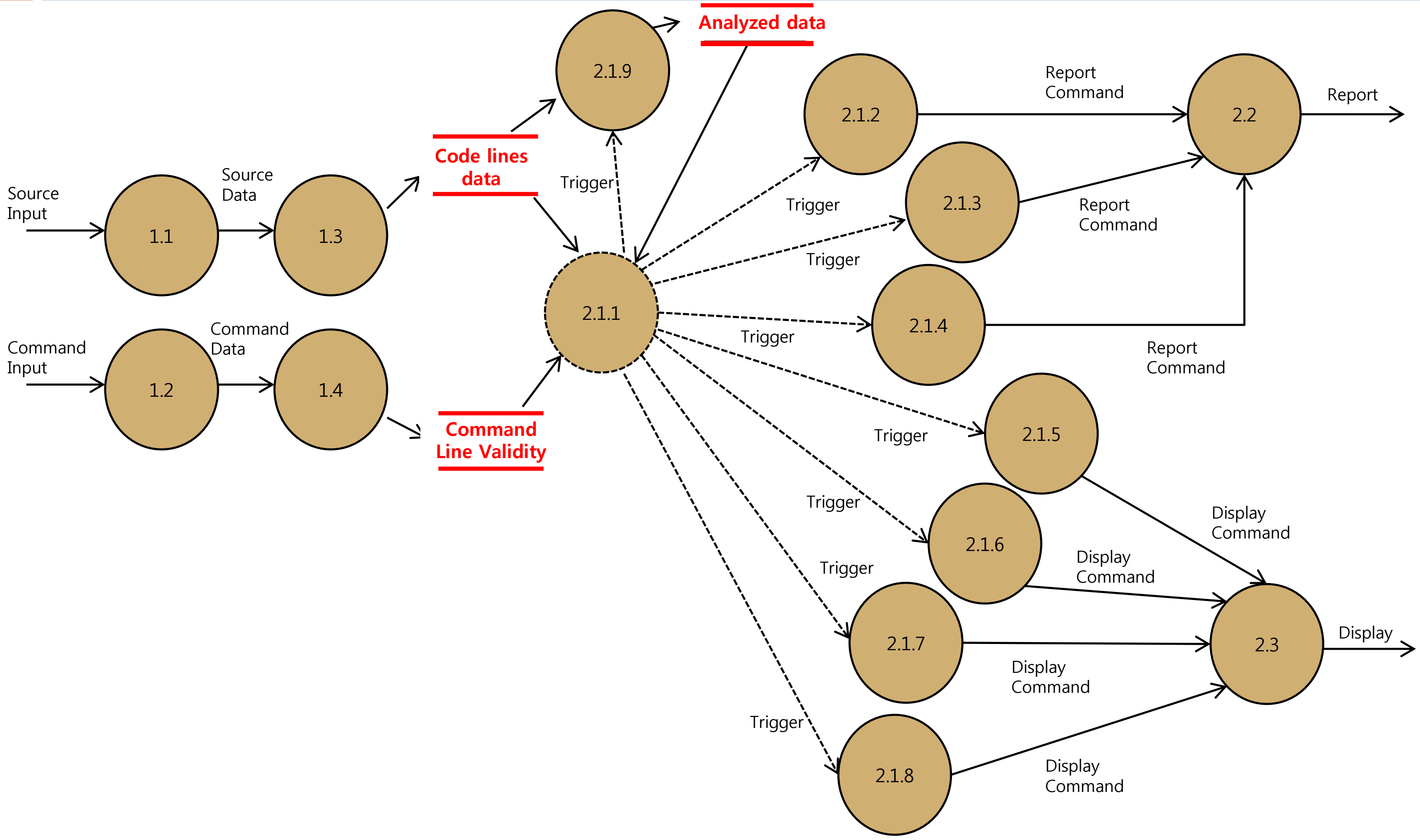


Modified DFD Level 4 (FSM)

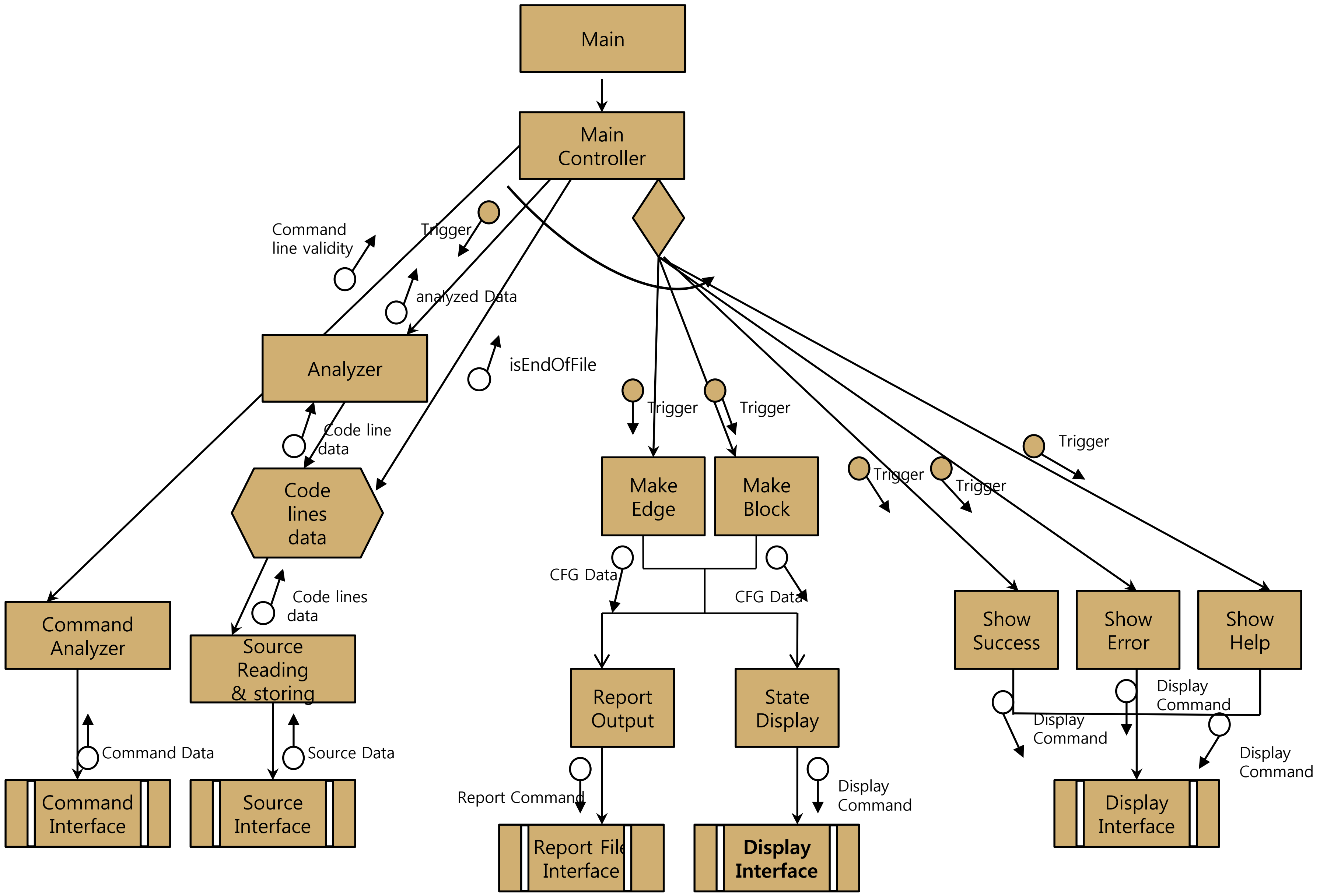

/ trigger "Command Analyzer"
Check Command Validity
[argc != 3] / trigger "Show help"
Stop
[argc == 3 && argv[1] == "*.c" && argv[2] == "*.txt"] / trigger "Show success"
Show success message
/ trigger "Show code"
Show source code
[argc == 3 &&(argv[1])== "*.c" &&(argv[2]) == "*.txt"] / trigger "Show success"
Read in source code & Store
[fp == NULL] / trigger "Show error"
[fp != NULL] / trigger "Show success"
[fp != NULL] / trigger "Analyzer"
Analyze each code line
[line_n == x] / trigger "Show report file name"
Show report file name
[line_n != x] / trigger "Make CFG"
Generate CFG with analyzed data
[analyzed[x] == 1 || analyzed[x] == 2] / trigger "Make if"
[analyzed[x] == 3] / trigger "Make else"
[analyzed[x] == 4] / trigger "Make for"
[analyzed[x] == 5] / trigger "Make while"
[analyzed[x] == 6] / trigger "Make do-while"
[analyzed[x] == 7 || analyzed[x] == 8] / trigger "Make case"
[analyzed[x] == 9] / trigger "Make switch"
[analyzed[x] == 10] / trigger "Make Block" trigger "Make Edge"
[line_n != x] / trigger "Make CFG"
Make Block
Make Edge
/ trigger "Show Block Info"
Display Block Info
/ trigger "Show Edge Info"
Display Edge Info

fp : file pointer for source code
line_n : total lines number to analyze
x : code line number of currently being processed

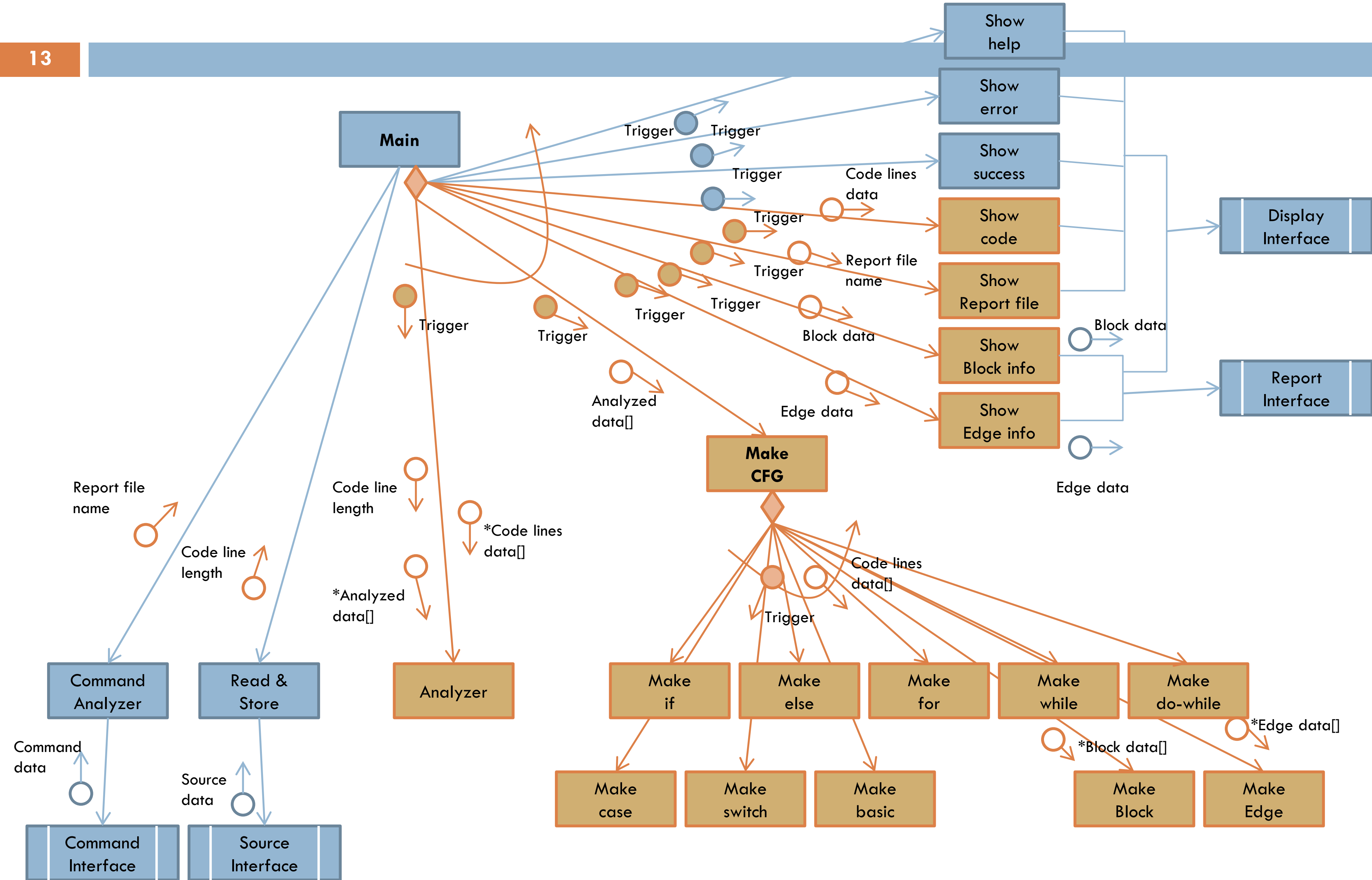
Total flow Diagram



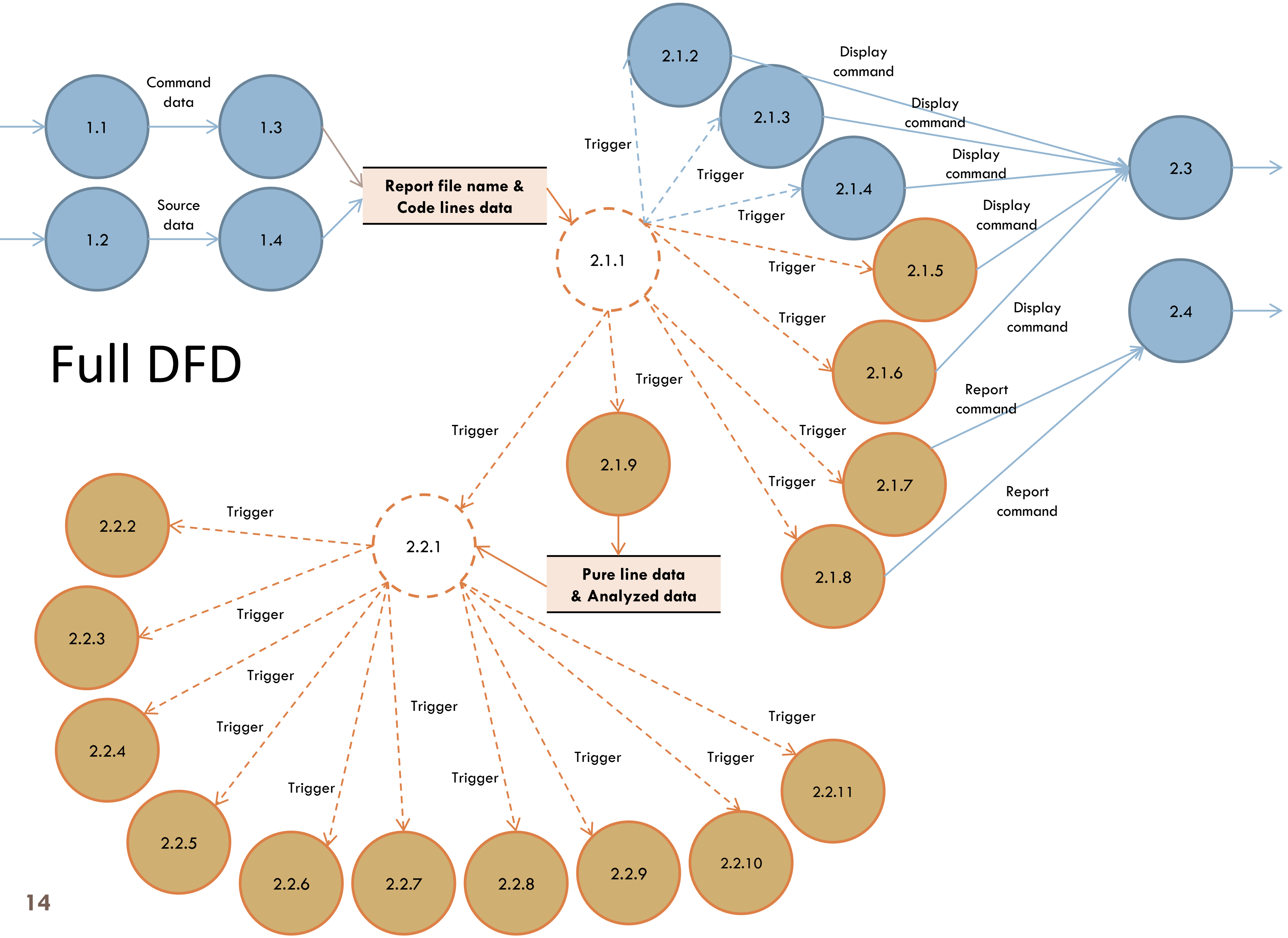
Original Structured Chart



Modified Structured Chart



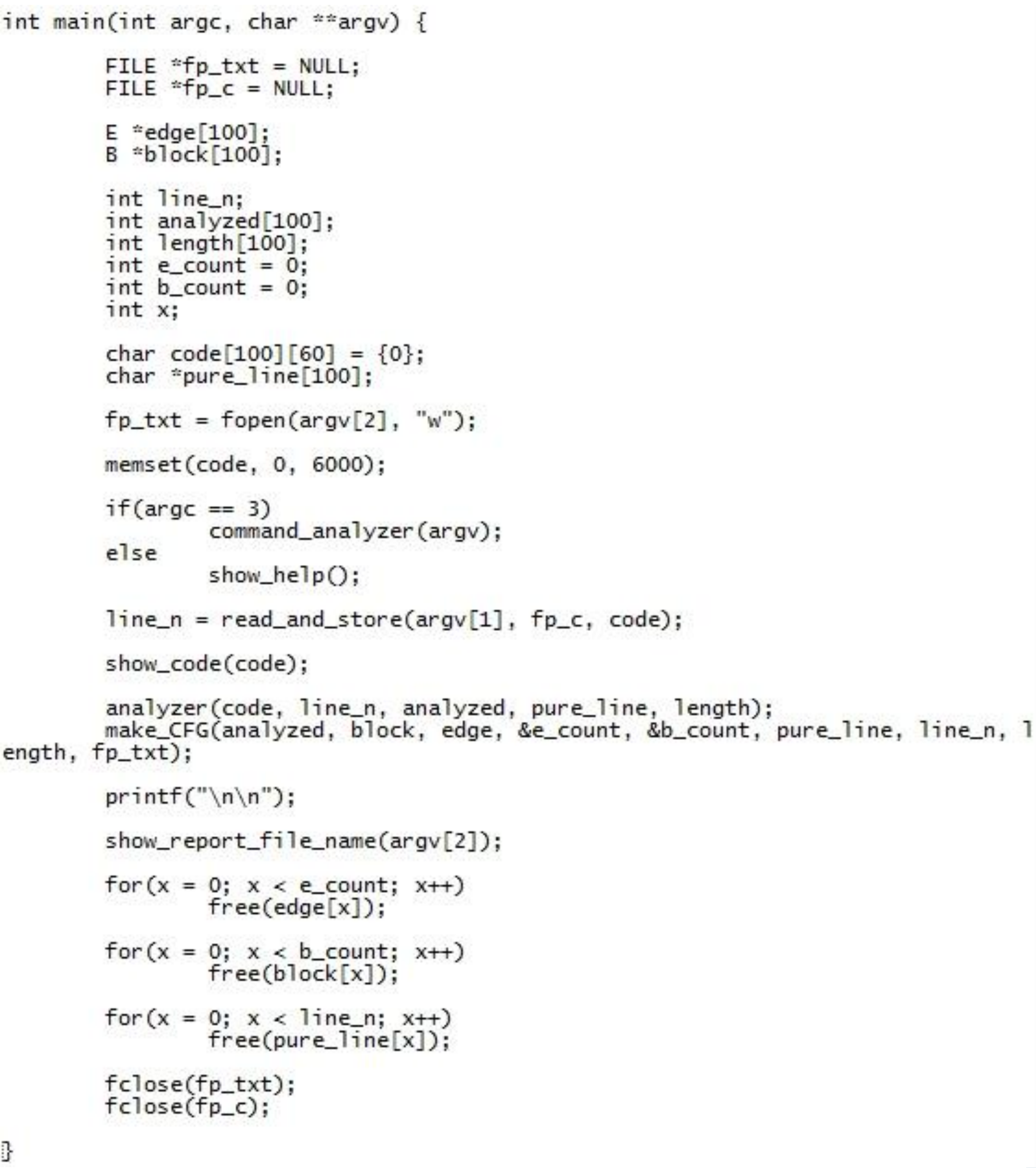
Full DFD


1.1
1.2
1.3
1.4
Command data
Source data
Report file name & Code lines data
2.1.1
2.1.2
2.1.3
2.1.4
2.1.5
2.1.6
2.1.7
2.1.8
2.1.9
Trigger
Display command
Report command
2.3
2.4
Pure line data & Analyzed data
2.2.1
2.2.2
2.2.3
2.2.4
2.2.5
2.2.6
2.2.7
2.2.8
2.2.9
2.2.10
2.2.11

Data Dictionary

Data name	Description	Type
Code[100][60]	code data lines that are read as it is from .c	char
Fp_c, Fp_txt(txt)	File pointer that indicates source.c and report.txt	FILE*
Line_n	Line number of all code lines in source.c	integer
Analyzed[ ]	Has number from 1~ to 10, each indicates 1: IF 2: Else if 3: Else 4.For 5:While 6: Do-while 7:Case 8:Default 9:Switch 10: Basic block and -1 for empty or new line	integer
Pure_line[ ] *(pure code line)*	Pointer arrays that would have dynamically allocated memories as much as each line`s length of all code lines without all spaces in front and the end of it	Char *
Length[ ]	The character number of each code lines	
e_count, b_count	e_count: counted number of all edges. Current e_count value is given as an edge number that is newly created B_count: counted number of all blocks. Given as an newly created block number	integer
Edge data	e_num: edge number to be newly created Start_b: the edge`s starting block number Dest_b: the edge`s destination block number	
Block data	b_num: block number to be newly created Cont: block`s contents that were read from pure line code	
edge[ ] block[ ]	Edge pointer array that would be dynamically allocated when new edges are created. Block pointer array that would be dynamically allocated when the new blocks need to created	E* B*

Explanation of source code



```
int main(int argc, char **argv) {

        FILE *fp_txt = NULL;
        FILE *fp_c = NULL;

        E *edge[100];
        B *block[100];

        int line_n;
        int analyzed[100];
        int length[100];
        int e_count = 0;
        int b_count = 0;
        int x;

        char code[100][60] = {0};
        char *pure_line[100];

        fp_txt = fopen(argv[2], "w");

        memset(code, 0, 6000);

        if(argc == 3)
                command_analyzer(argv);
        else
                show_help();

        line_n = read_and_store(argv[1], fp_c, code);

        show_code(code);

        analyzer(code, line_n, analyzed, pure_line, length);
        make_CFG(analyzed, block, edge, &e_count, &b_count, pure_line, line_n, l
ength, fp_txt);

        printf("\n\n");

        show_report_file_name(argv[2]);

        for(x = 0; x < e_count; x++)
                free(edge[x]);

        for(x = 0; x < b_count; x++)
                free(block[x]);

        for(x = 0; x < line_n; x++)
                free(pure_line[x]);

        fclose(fp_txt);
        fclose(fp_c);

}
```

Main function

Input	Argc, argv
Output	

Description

1.Reading ./CFG ex.c report.txt, it receives source code file name and report txt file name as input arguments from user
2.read in source code and store in code array
3. From code line, get pure code line excluding spaces in front and end
4. Start analyzing finding out whether each code line of statement is

If, else if, else, switch, case, default, while, for, do while, and just basic block or \n

5. According to above data, start generating CFG

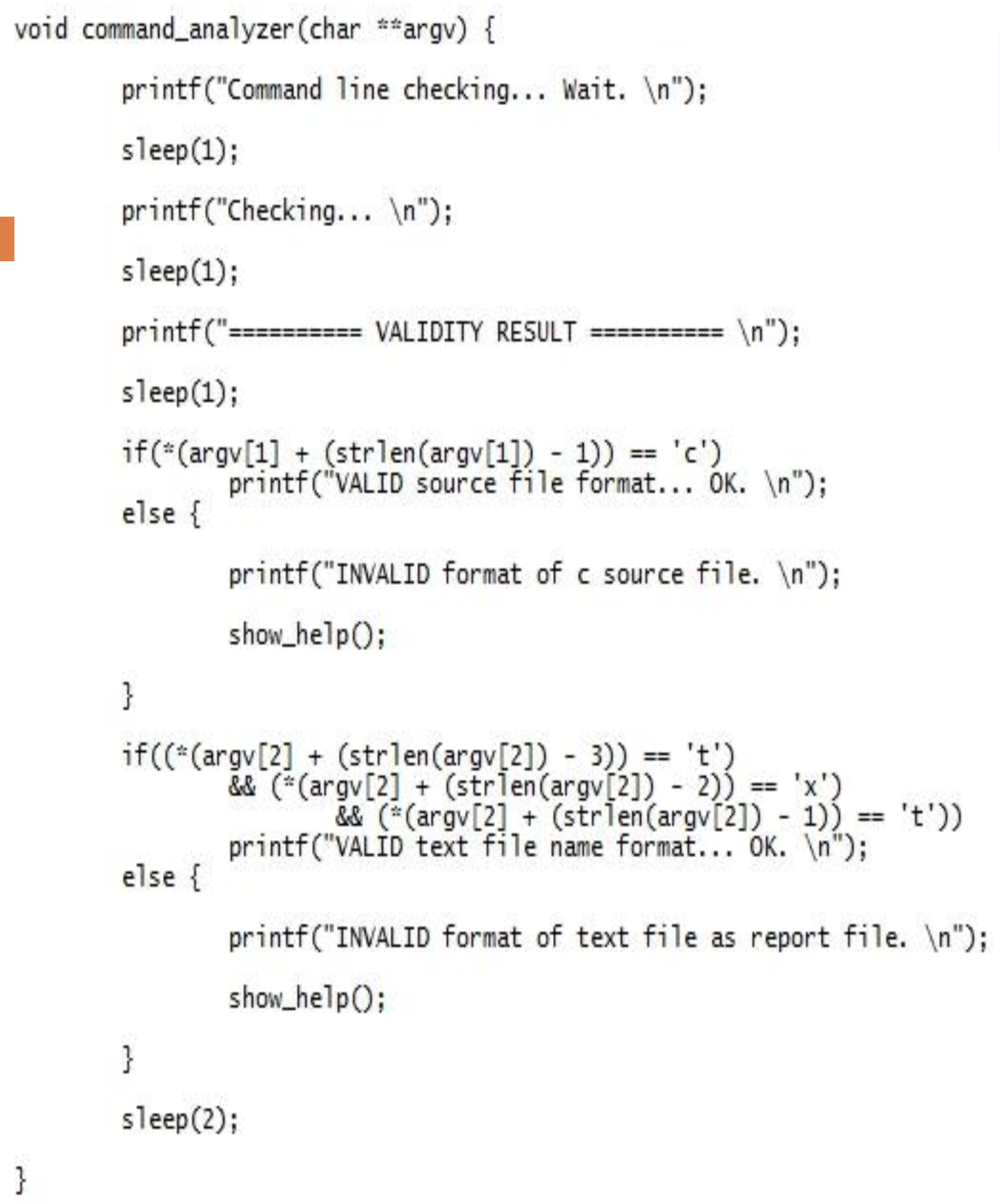
6. Show report file name created at last

7.Free all dynamically allocated block and edge`s space.

8. Close the file of source.c and report.txt

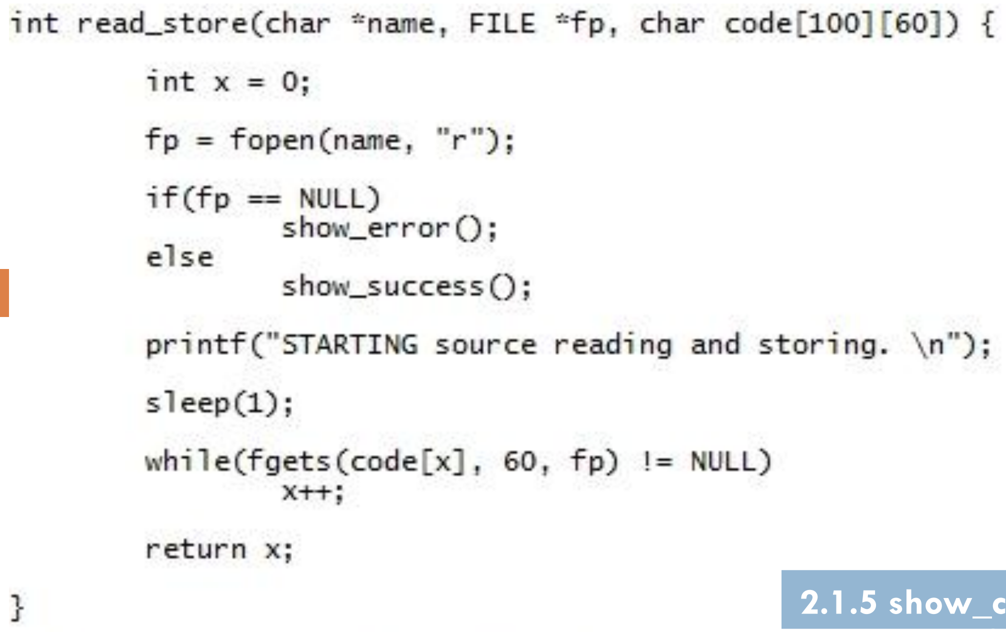
Data name	Description	Type
x	Used in make_if/ make_switch/make while, make_for and so on for all making functions Get x from make_CFG function as currently processed line number of code line(exactly, pure_line[ ]) when it generates blocks and edges	Int*
For_flag	Used in make_CFG() if the for_flag is 1, →change it to the back edge when it meets the end of while statement →revise block contents as increment formula (ex. i++) →turn off the flag when for statement ends. If for_flag is 0, just make basical downward edge	Integer 1 : on 0 : off
While_flag	Used in make_CFG() if the for_flag is 1, →change it to the back edge when it meets the end of while statement →turn off the flag when the while statement ends.	Integer 1 : on 0 : off
Do_while flag	Used in make_CFG() if the for_flag is 1, →change it to the back edge when it meets the end of do_while statement →turn off the flag when do_while statement ends.	Integer 1 : on 0 : off
End_of_while	line number of the end of while statement, returned from make_while()	Integer
End_of_do_while	line number of the end of do while statement, returned from make_do_while()	Integer
End_of_for	line number of the end of for statement, returned from make_for()	Integer

1.3 command_analyzer

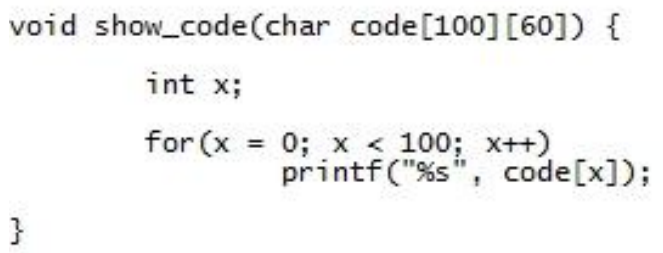


```c
void command_analyzer(char **argv) {

        printf("Command line checking... Wait. \n");

        sleep(1);

        printf("Checking... \n");

        sleep(1);

        printf("========== VALIDITY RESULT ========== \n");

        sleep(1);

        if(*(argv[1] + (strlen(argv[1]) - 1)) == 'c')
                printf("VALID source file format... OK. \n");
        else {

                printf("INVALID format of c source file. \n");

                show_help();

        }

        if((*(argv[2] + (strlen(argv[2]) - 3)) == 't')
                && (*(argv[2] + (strlen(argv[2]) - 2)) == 'x')
                        && (*(argv[2] + (strlen(argv[2]) - 1)) == 't'))
                printf("VALID text file name format... OK. \n");
        else {

                printf("INVALID format of text file as report file. \n");

                show_help();

        }

        sleep(2);

}
```

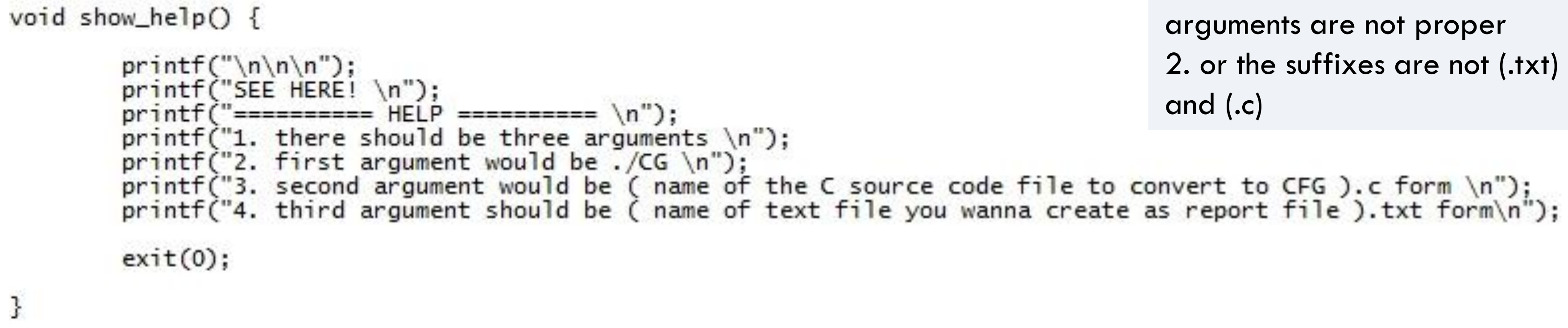
Input	Argv[1] (source.c argument) Argv[2] (report.txt argument)
Output	
Description	
Reading in the command line from user, it checks whether it finishes with .c for C source code and .txt for text file	



```
int read_store(char *name, FILE *fp, char code[100][60]) {

        int x = 0;

        fp = fopen(name, "r");

        if(fp == NULL)
                show_error();
        else
                show_success();

        printf("STARTING source reading and storing. \n");

        sleep(1);

        while(fgets(code[x], 60, fp) != NULL)
                x++;

        return x;

}
```



```
void show_code(char code[100][60]) {

        int x;

        for(x = 0; x < 100; x++)
                printf("%s", code[x]);

}
```



```
void show_help() {

        printf("\n\n\n");
        printf("SEE HERE! \n");
        printf("========== HELP ========== \n");
        printf("1. there should be three arguments \n");
        printf("2. first argument would be ./CG \n");
        printf("3. second argument would be ( name of the C source code file to convert to CFG ).c form \n");
        printf("4. third argument should be ( name of text file you wanna create as report file ).txt form\n");

        exit(0);

}
```

1.4 read & store

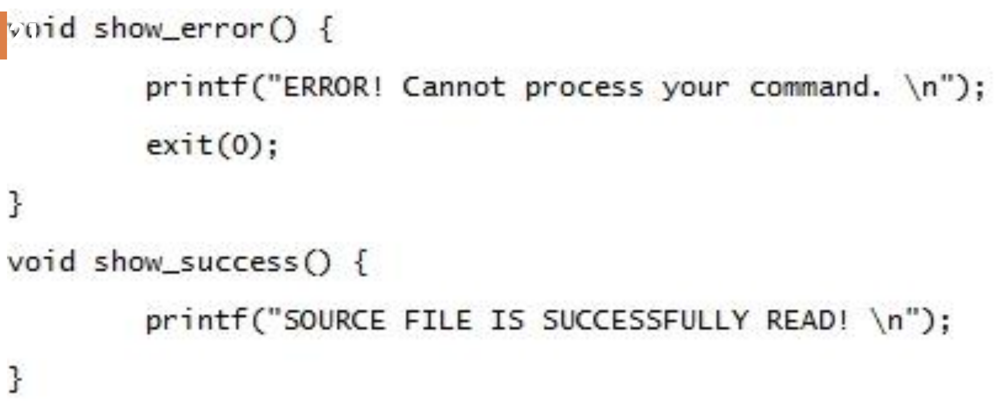
Input	Source.c , fp, code[][]
Output	*Code line number*
Description	
Reads source.c and store all lines in code array. If the file pointer fp is valid, show success message. If not, show error message.	

2.1.5 show_code

Input	Source data
Output	Display command
Description	
Print out all codes that were read	

2.1.2 show_help

Input	Trigger
Output	Display command
Description	
Show user help message 1. when the number of arguments are not proper 2. or the suffixes are not (.txt) and (.c)	



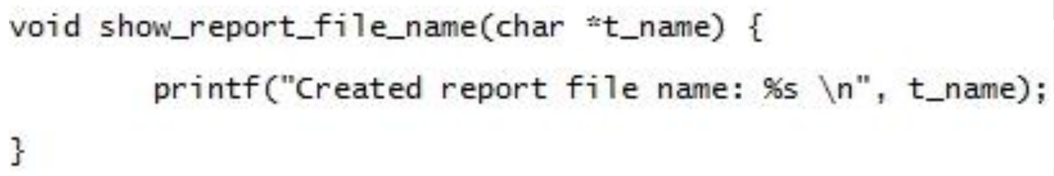
```
void show_error() {
        printf("ERROR! Cannot process your command. \n");
        exit(0);
}
void show_success() {
        printf("SOURCE FILE IS SUCCESSFULLY READ! \n");
}
```

2.1.3 show_error

Input	Trigger
Output	
Description	
Show error in case source reading was fail.	

2.1.4 show_success

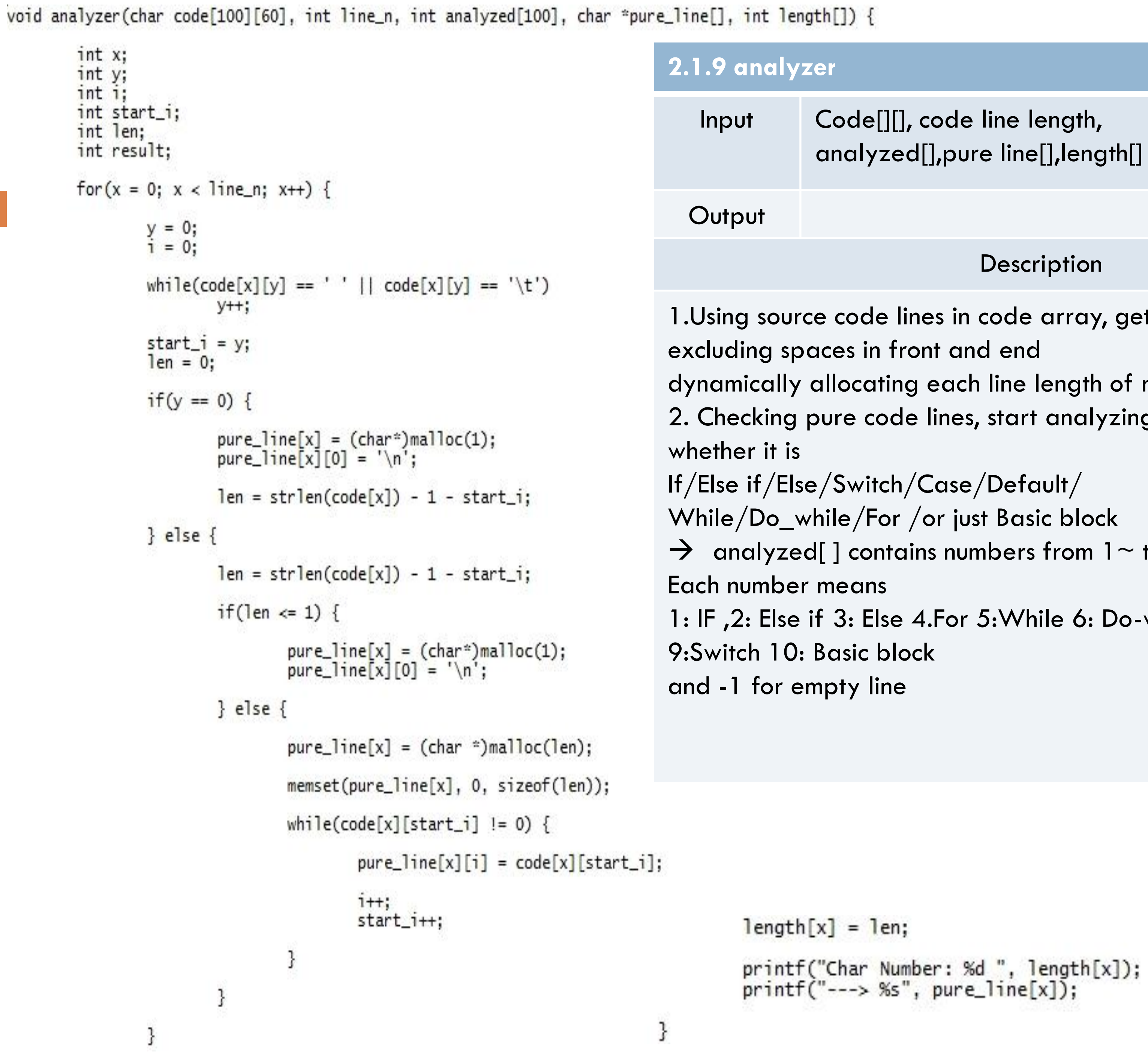
Input	Trigger
Output	
Description	
Show success if the source code and command line were read properly.	



```
void show_report_file_name(char *t_name) {
        printf("Created report file name: %s \n", t_name);
}
```

2.1.6 show_report_file_name

Input	File name
Output	Display command
Description	
Show report file name.	



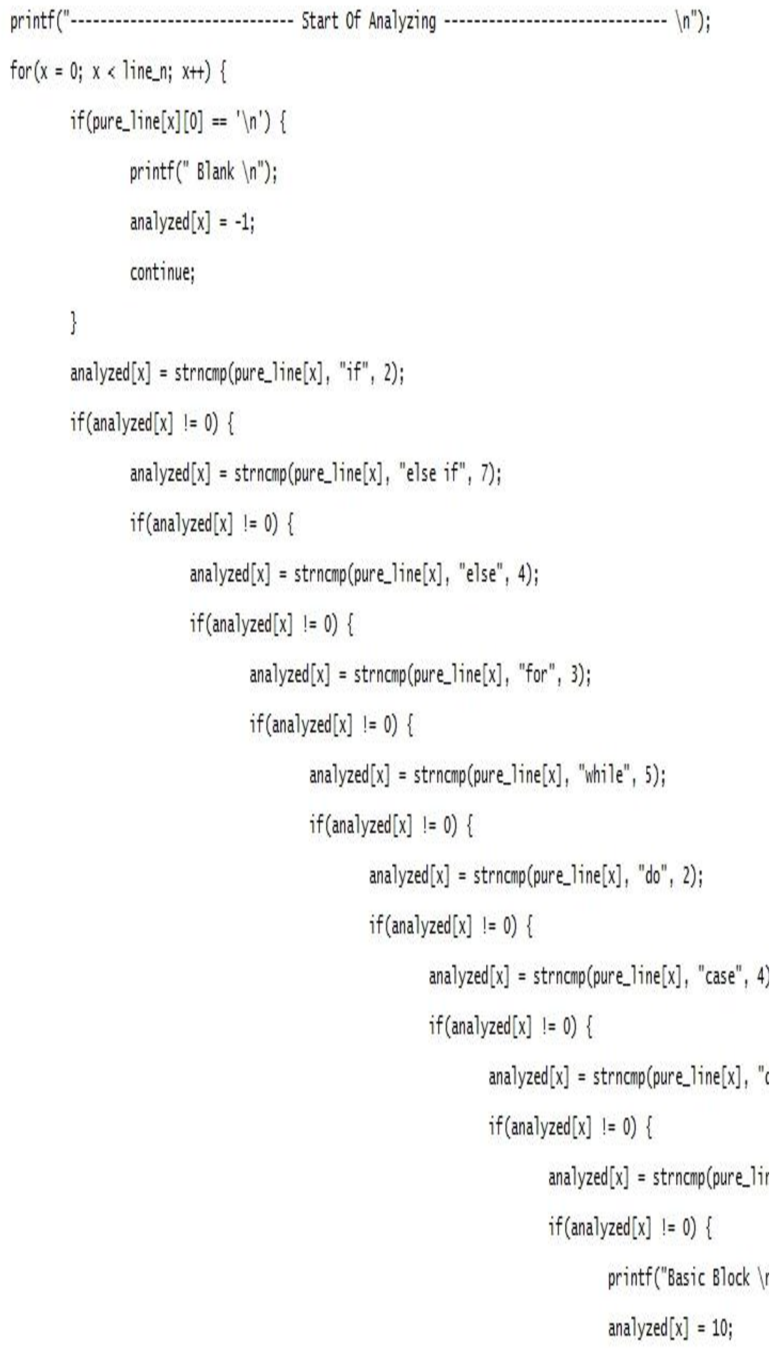
```c
void analyzer(char code[100][60], int line_n, int analyzed[100], char *pure_line[], int length[]) {

        int x;
        int y;
        int i;
        int start_i;
        int len;
        int result;

        for(x = 0; x < line_n; x++) {

                y = 0;
                i = 0;

                while(code[x][y] == ' ' || code[x][y] == '\t')
                        y++;

                start_i = y;
                len = 0;

                if(y == 0) {

                        pure_line[x] = (char*)malloc(1);
                        pure_line[x][0] = '\n';

                        len = strlen(code[x]) - 1 - start_i;

                } else {

                        len = strlen(code[x]) - 1 - start_i;

                        if(len <= 1) {

                                pure_line[x] = (char*)malloc(1);
                                pure_line[x][0] = '\n';

                        } else {

                                pure_line[x] = (char *)malloc(len);

                                memset(pure_line[x], 0, sizeof(len));

                                while(code[x][start_i] != 0) {

                                        pure_line[x][i] = code[x][start_i];

                                        i++;
                                        start_i++;

                                }

                        }

                }

                length[x] = len;

                printf("Char Number: %d ", length[x]);
                printf("---> %s", pure_line[x]);

        }
```

2.1.9 analyzer

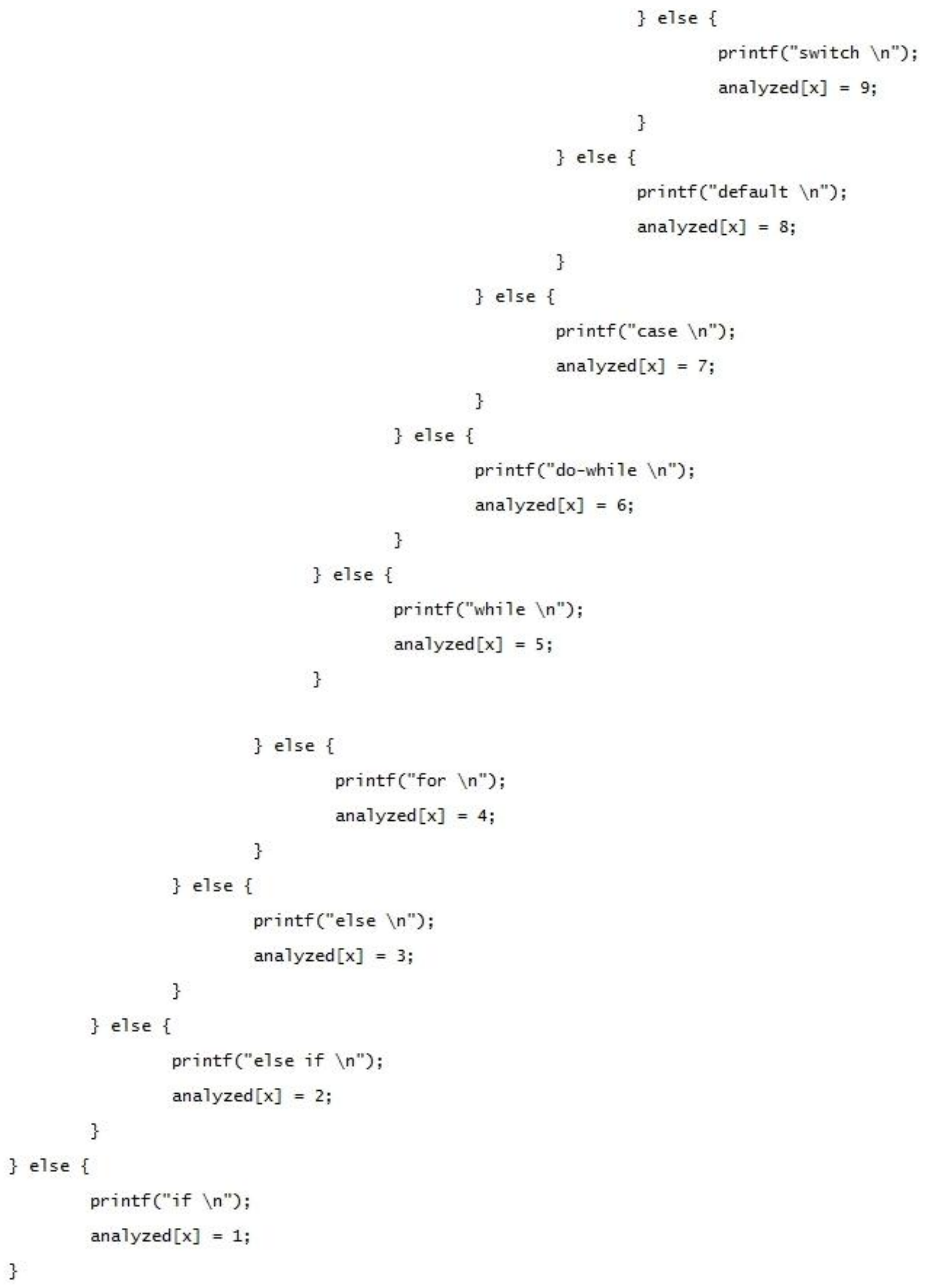
Input	Code[][], code line length, analyzed[],pure line[],length[]
Output	

Description

1.Using source code lines in code array, get pure code lines excluding spaces in front and end
dynamically allocating each line length of memories
2. Checking pure code lines, start analyzing whether it is
If/Else if/Else/Switch/Case/Default/
While/Do_while/For /or just Basic block
→ analyzed[ ] contains numbers from 1~ to 10
Each number means
1: IF ,2: Else if 3: Else 4.For 5:While 6: Do-while 7:Case 8:Default 9:Switch 10: Basic block
and -1 for empty line



```
printf("------------------------------ Start Of Analyzing ------------------------------ \n");
for(x = 0; x < line_n; x++) {
    if(pure_line[x][0] == '\n') {
        printf(" Blank \n");
        analyzed[x] = -1;
        continue;
    }
    analyzed[x] = strncmp(pure_line[x], "if", 2);
    if(analyzed[x] != 0) {
        analyzed[x] = strncmp(pure_line[x], "else if", 7);
        if(analyzed[x] != 0) {
            analyzed[x] = strncmp(pure_line[x], "else", 4);
            if(analyzed[x] != 0) {
                analyzed[x] = strncmp(pure_line[x], "for", 3);
                if(analyzed[x] != 0) {
                    analyzed[x] = strncmp(pure_line[x], "while", 5);
                    if(analyzed[x] != 0) {
                        analyzed[x] = strncmp(pure_line[x], "do", 2);
                        if(analyzed[x] != 0) {
                            analyzed[x] = strncmp(pure_line[x], "case", 4)
                            if(analyzed[x] != 0) {
                                analyzed[x] = strncmp(pure_line[x], "
                                if(analyzed[x] != 0) {
                                    analyzed[x] = strncmp(pure_li
                                    if(analyzed[x] != 0) {
                                        printf("Basic Block \
                                        analyzed[x] = 10;
```



```
                                    } else {
                                        printf("switch \n");
                                        analyzed[x] = 9;
                                    }
                                } else {
                                    printf("default \n");
                                    analyzed[x] = 8;
                                }
                            } else {
                                printf("case \n");
                                analyzed[x] = 7;
                            }
                        } else {
                            printf("do-while \n");
                            analyzed[x] = 6;
                        }
                    } else {
                        printf("while \n");
                        analyzed[x] = 5;
                    }

                } else {
                    printf("for \n");
                    analyzed[x] = 4;
                }
            } else {
                printf("else \n");
                analyzed[x] = 3;
            }
        } else {
            printf("else if \n");
            analyzed[x] = 2;
        }
    } else {
        printf("if \n");
        analyzed[x] = 1;
    }
}
```

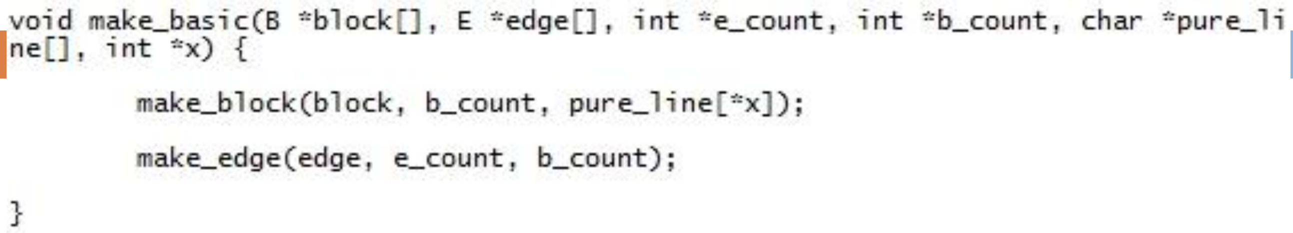
```c
void make_CFG(int analyzed[], B *block[], E *edge[], int *e_count, int *b_count, char *pure_line[], int line_n, int length[], FILE *txt) {

        int i;
        int j = line_n;
        int end;
        int for_flag;
        int start;
        int while_flag;
        int do_while_flag;
        int head_n;

        char incre[4];

        printf("\n------------------- START CREATING BLOCKS AND EDGES ------------------- \n");

        sleep(1);

        for(i = 0; i < j; i++) {

                switch(analyzed[i]) {

                        case 1:
                                make_if(block, edge, e_count, b_count, pure_line, &i, txt);

                                break;
                        case 2:
                                make_if(block, edge, e_count, b_count, pure_line, &i, txt);

                                break;
                        case 3:
                                make_else(block, edge, e_count, b_count, pure_line, &i, txt);

                                break;
                        case 4:
                                end = make_for(block, edge, e_count, b_count, pure_line, &i, length, incre, txt);

                                for_flag = 1;
                                start = *b_count - 1;

                                break;
                        case 5:
                                end = make_while(block, edge, e_count, b_count, pure_line, &i, length, txt);

                                while_flag = 1;
                                start = *b_count;

                                break;
                        case 6:
                                end = make_do_while(block, edge, e_count, b_count, pure_line, &i, length, txt);

                                do_while_flag = 1;
                                start = *b_count;

                                break;
```

```c
                        case 7:
                                make_case(block, edge, e_count, b_count, pure_line, &i, &head_n, txt);

                                break;
                        case 8:
                                make_default(block, edge, e_count, b_count, pure_line, &i, &head_n, txt);

                                break;
                        case 9:
                                make_switch(block, edge, e_count, b_count, pure_line, &i, &head_n, txt);

                                break;
                        case 10:
                                make_basic(block, edge, e_count, b_count, pure_line, &i);
                                show_block_info(block, b_count, txt);

                                if(i == end && while_flag == 1) {

                                        edge[*e_count - 1]->start_b = *b_count - 1;
                                        edge[*e_count - 1]->dest_b = start - 1;

                                        while_flag = 0;

                                }

                                if(i == end && do_while_flag == 1) {

                                        edge[*e_count - 1]->start_b = *b_count - 1;
                                        edge[*e_count - 1]->dest_b = start - 1;

                                        do_while_flag = 0;

                                }

                                if(i == end && for_flag == 1) {

                                        show_edge_info(edge, e_count, txt);

                                        make_block(block, b_count, pure_line[i]);
                                        strcpy(block[*b_count - 1]->cont, incre);
                                        show_block_info(block, b_count, txt);

                                        make_edge(edge, e_count, b_count);
                                        edge[*e_count - 1]->start_b = *b_count - 1;
                                        edge[*e_count - 1]->dest_b = start;

                                        for_flag = 0;

                                }

                                show_edge_info(edge, e_count, txt);

                                break;
```

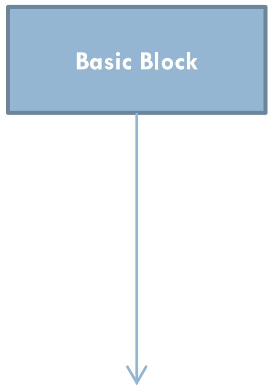
2.2.1 make_CFG

Input	Analyzed[ ], pure lines[], block[], b_count edge[], e_count, length[], txt
Output	

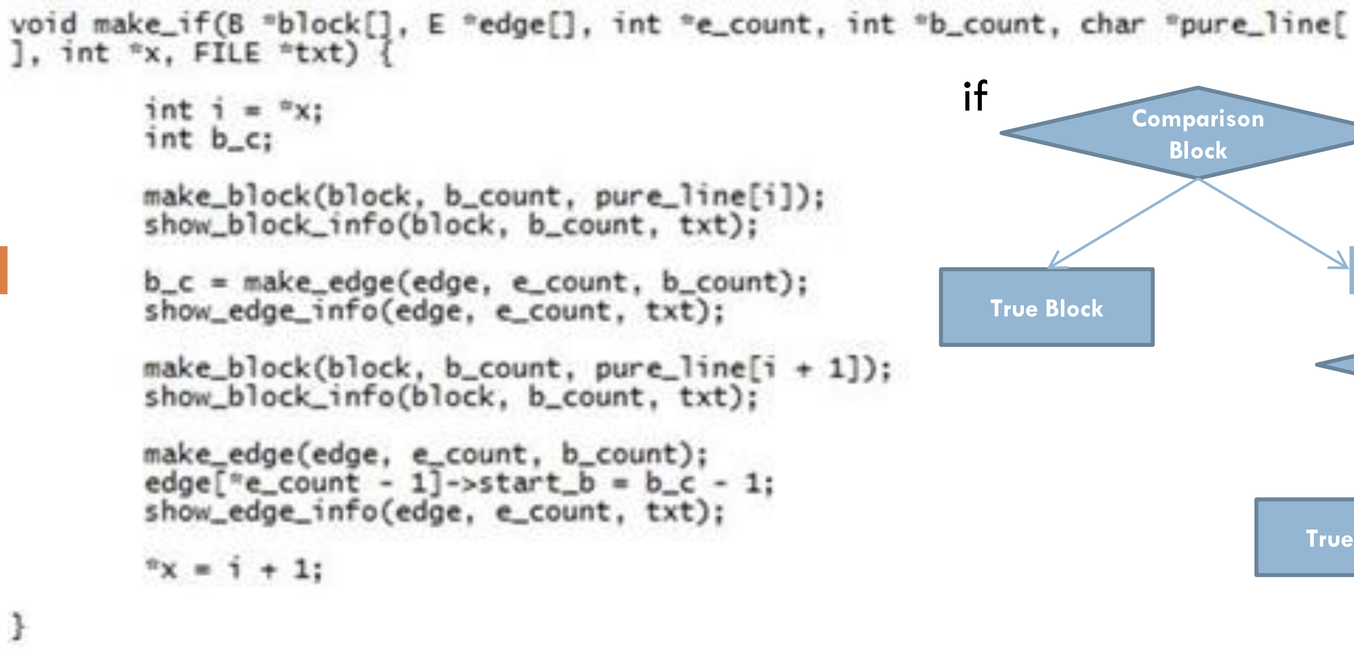
Description
1.make blocks and edges according to the analyzed data, (1~10 cases) 2.Show recently created edge`s and block`s info generating CFG. Handles if- else if cases samely and case-default too. Make CFG through all code lines in pure_lines[ ]. If length[ ]==1, consider that code line as “ } “, not especially distinguishing ‘ } ‘ In case they are recursive statements, use flags of while, do_while, for statements - When the flag is on and the end of recursive statement is met, revise edge`s destination block number to the starting block of it. (changing normal downward edge to the upward back edge) - After revising informations, show user recently created edge and block informations. - Turn off the flag.



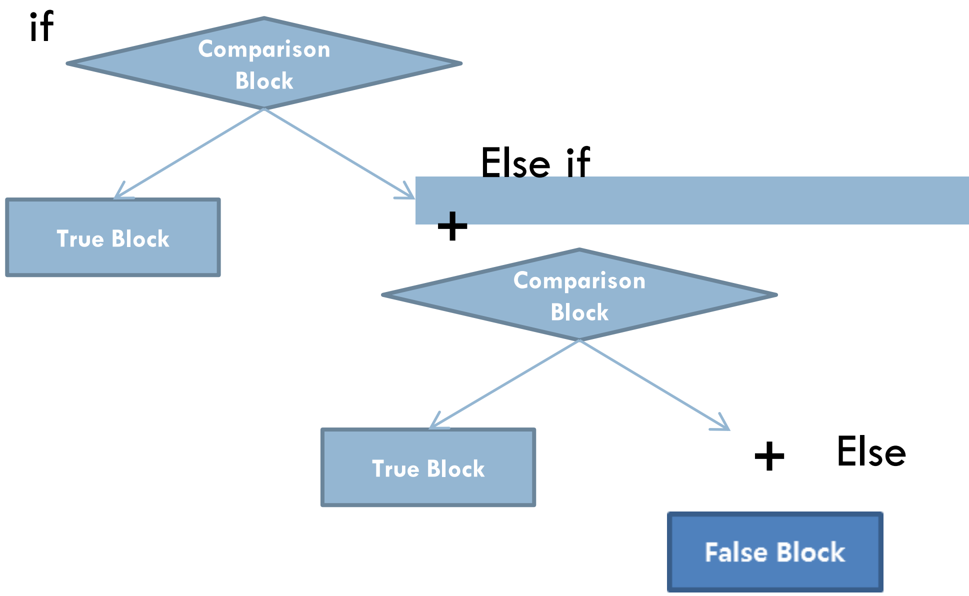
```
void make_basic(B *block[], E *edge[], int *e_count, int *b_count, char *pure_li
ne[], int *x) {

        make_block(block, b_count, pure_line[*x]);

        make_edge(edge, e_count, b_count);

}
```



2.2.8 make_basic	
Input	Trigger, block[ ],edge[ ], b_count, e_count, pure line[ ], x
Output	
Description	
Make one pair of block and edge for basical case. In case analyzed[x] value is 10	



```
void make_if(B *block[], E *edge[], int *e_count, int *b_count, char *pure_line[
], int *x, FILE *txt) {

        int i = *x;
        int b_c;

        make_block(block, b_count, pure_line[i]);
        show_block_info(block, b_count, txt);

        b_c = make_edge(edge, e_count, b_count);
        show_edge_info(edge, e_count, txt);

        make_block(block, b_count, pure_line[i + 1]);
        show_block_info(block, b_count, txt);

        make_edge(edge, e_count, b_count);
        edge[*e_count - 1]->start_b = b_c - 1;
        show_edge_info(edge, e_count, txt);

        *x = i + 1;

}
```

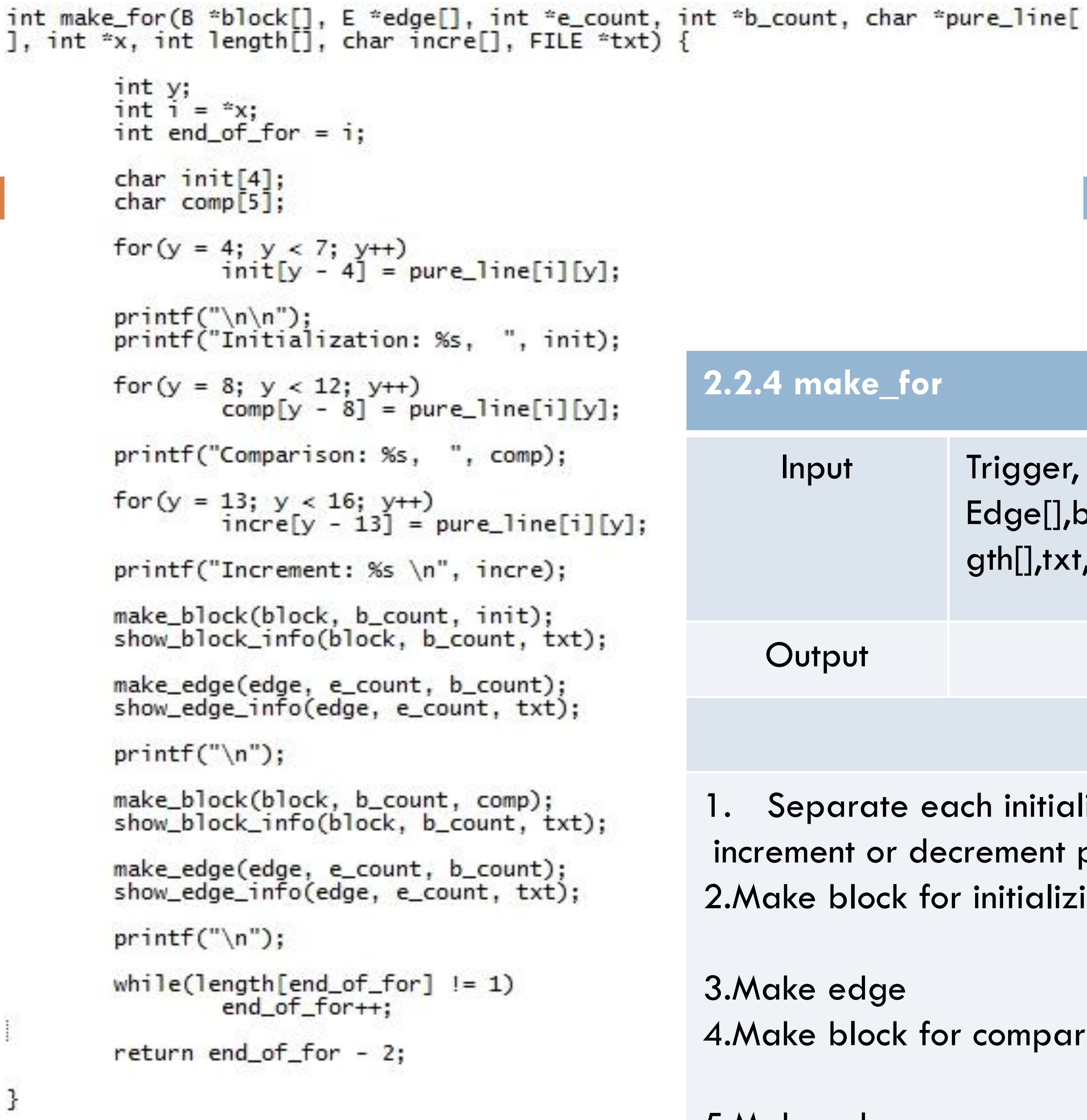


2.2.2 make_if (applied to else if case,too)

Input	Trigger, Edge[],block[],e_count,b_count,pure_line[],x,length[],txt
Output	
Description	
1.Make head block 2.Make left edge and left case block (true) 3.Make right edge (false edge)	

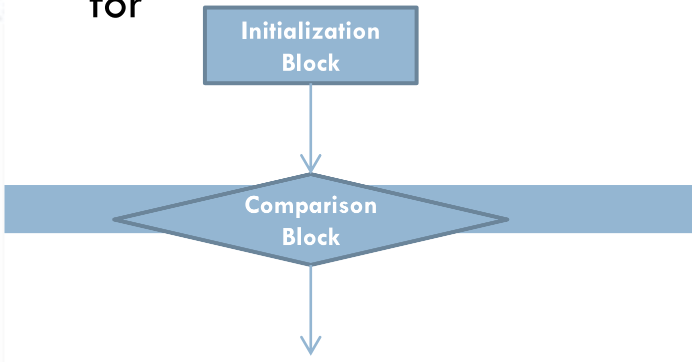
2.2.3 make_else

Input	Trigger, Edge[],block[],e_count,b_count,pure_line[],x, length[],txt
Output	
Description	
When only else comes out after if or else if, make right case block (false block)	



```
int make_for(B *block[], E *edge[], int *e_count, int *b_count, char *pure_line[
], int *x, int length[], char incre[], FILE *txt) {

        int y;
        int i = *x;
        int end_of_for = i;

        char init[4];
        char comp[5];

        for(y = 4; y < 7; y++)
                init[y - 4] = pure_line[i][y];

        printf("\n\n");
        printf("Initialization: %s,  ", init);

        for(y = 8; y < 12; y++)
                comp[y - 8] = pure_line[i][y];

        printf("Comparison: %s,  ", comp);

        for(y = 13; y < 16; y++)
                incre[y - 13] = pure_line[i][y];

        printf("Increment: %s \n", incre);

        make_block(block, b_count, init);
        show_block_info(block, b_count, txt);

        make_edge(edge, e_count, b_count);
        show_edge_info(edge, e_count, txt);

        printf("\n");

        make_block(block, b_count, comp);
        show_block_info(block, b_count, txt);

        make_edge(edge, e_count, b_count);
        show_edge_info(edge, e_count, txt);

        printf("\n");

        while(length[end_of_for] != 1)
                end_of_for++;

        return end_of_for - 2;

}
```

for

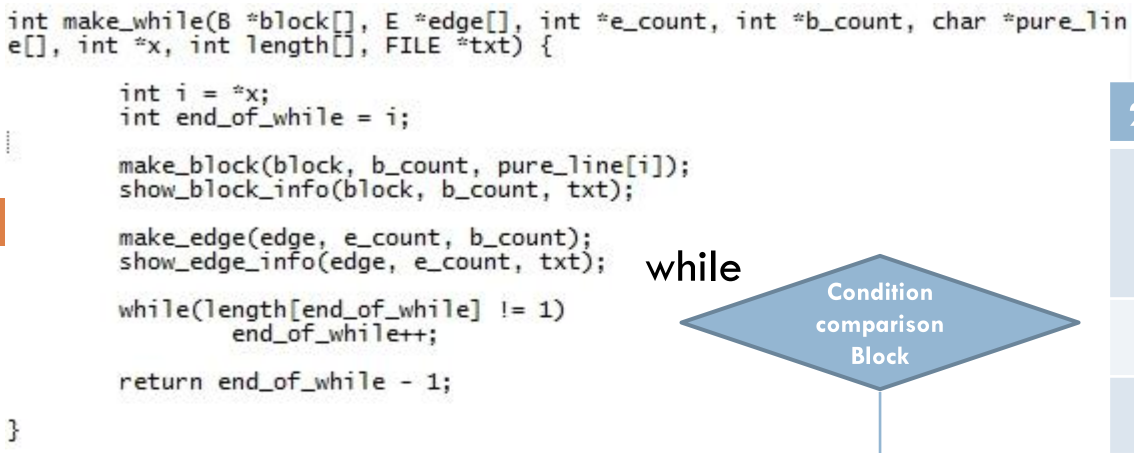


2.2.4 make_for

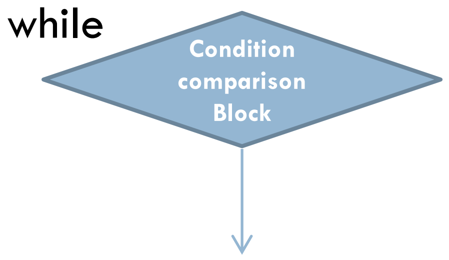
Input	Trigger, Edge[],block[],e_count,b_count,pure_line[],x,length[],txt,char incre[],txt
Output	

Description

1. Separate each initialization, comparison, increment or decrement part from for(~;~;~) code line
2.Make block for initializing part
Ex. [i=0]
3.Make edge
4.Make block for comparison part
Ex. [i<10]
5.Make edge



```
int make_while(B *block[], E *edge[], int *e_count, int *b_count, char *pure_lin
e[], int *x, int length[], FILE *txt) {

        int i = *x;
        int end_of_while = i;

        make_block(block, b_count, pure_line[i]);
        show_block_info(block, b_count, txt);

        make_edge(edge, e_count, b_count);
        show_edge_info(edge, e_count, txt);

        while(length[end_of_while] != 1)
                end_of_while++;

        return end_of_while - 1;

}
```

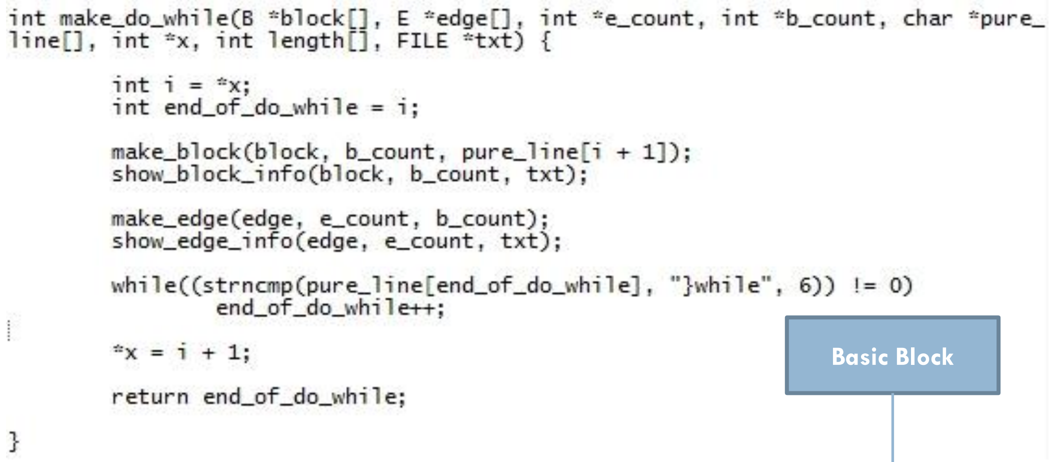


2.2.5 make_while

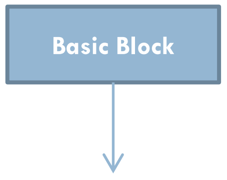
Input	Trigger, Edge[],block[],e_count,b_count,pure_line[],x,length[],txt
Output	End line number of while
Description	
1.Make block for while`s (condition) part 2.Make edge 3. Calculate end line number of while statement and return the number	

2.2.6 make_do_while

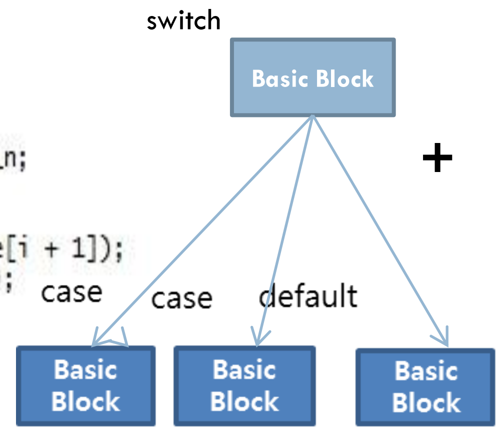
Input	Trigger, Edge[],block[],e_count,b_count,pure_line[],x,length[],txt
Output	End line number of do_while
Description	
1.Make a block with do`s next statement 2. Make edge 3. Calculate end line number of do_while statement and return the number	



```
int make_do_while(B *block[], E *edge[], int *e_count, int *b_count, char *pure_
line[], int *x, int length[], FILE *txt) {

        int i = *x;
        int end_of_do_while = i;

        make_block(block, b_count, pure_line[i + 1]);
        show_block_info(block, b_count, txt);

        make_edge(edge, e_count, b_count);
        show_edge_info(edge, e_count, txt);

        while((strncmp(pure_line[end_of_do_while], "}while", 6)) != 0)
                end_of_do_while++;

        *x = i + 1;

        return end_of_do_while;

}
```

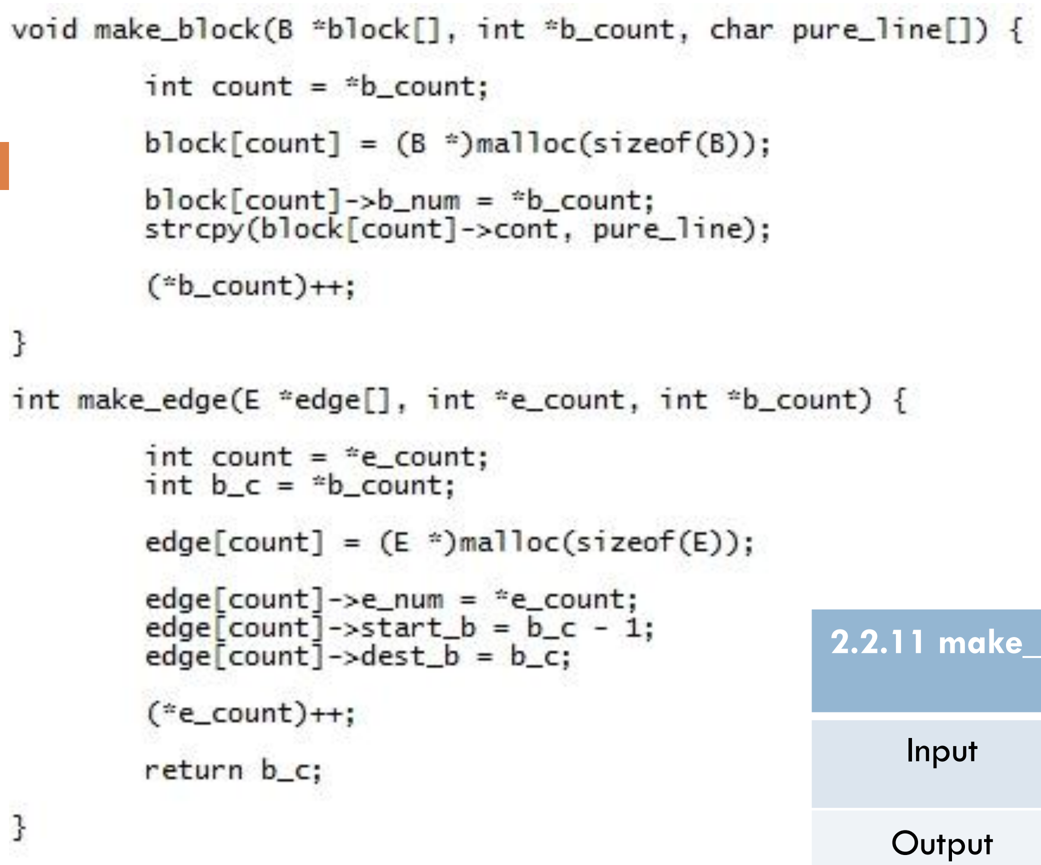


```
void make_switch(B *block[], E *edge[], int *e_count, int *b_count, char *pure_l
ine[], int *x, int *head_n, FILE *txt) {

        int i = *x;

        *head_n = *b_count;

        make_block(block, b_count, pure_line[i]);
        show_block_info(block, b_count, txt);

}

void make_case(B *block[], E *edge[], int *e_count, int *b_count, char *pure_lin
e[], int *x, int *head_n, FILE *txt) {

        int i = *x;

        make_edge(edge, e_count, b_count);
        edge[*e_count - 1]->start_b = *head_n;
        show_edge_info(edge, e_count, txt);

        make_block(block, b_count, pure_line[i + 1]);
        show_block_info(block, b_count, txt);

        *x = i + 1;

}

void make_default(B *block[], E *edge[], int *e_count, int *b_count, char *pure_
line[], int *x, int *head_n, FILE *txt) {

        int i = *x;

        make_edge(edge, e_count, b_count);
        edge[*e_count - 1]->start_b = *head_n;
        printf("\n");
        show_edge_info(edge, e_count, txt);

        make_block(block, b_count, pure_line[i + 1]);
        show_block_info(block, b_count, txt);

        *x = i + 1;

}
```



2.2.8 make_switch	
Input	Trigger, Edge[],block[],e_count,b_count,pure_line[],x,length[],txt
Output	
Description	
1.Make block for variable that is in the switch parenthesis	

2.2.7 make_case (applied to default case too)	
Input	Trigger, Edge[],block[],e_count,b_count,pure_line[],x,length[],txt
Output	
Description	
1. Make edge 2. Revise edge`s start block number to the stored head block number of switch 2. Make block	



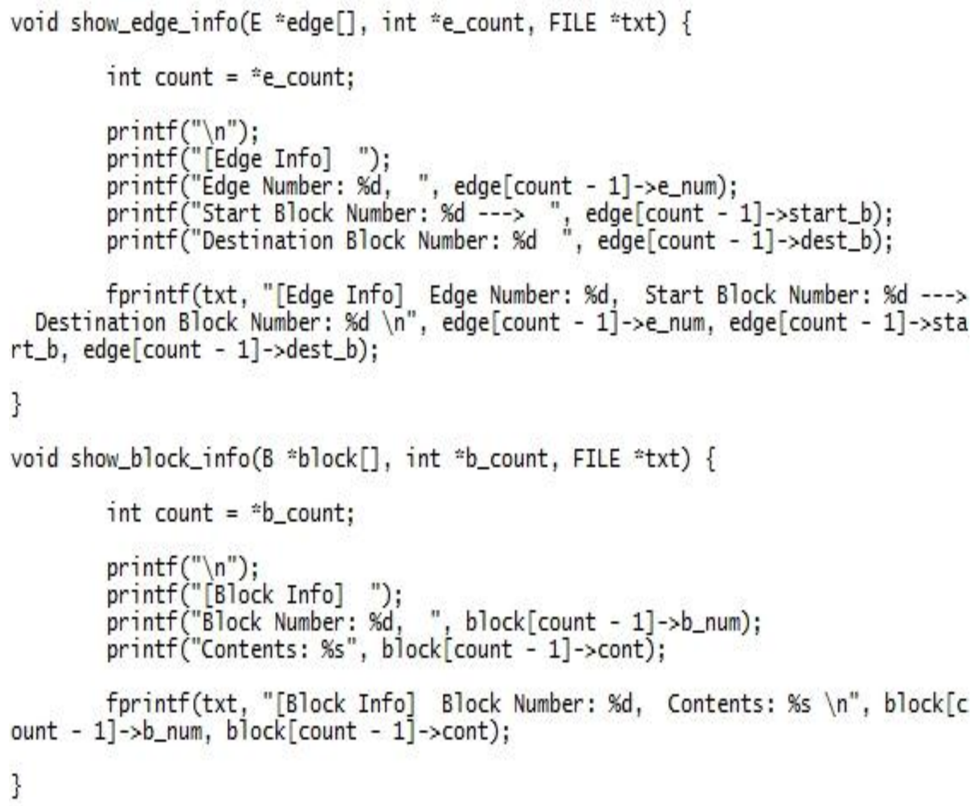
```
void make_block(B *block[], int *b_count, char pure_line[]) {

        int count = *b_count;

        block[count] = (B *)malloc(sizeof(B));

        block[count]->b_num = *b_count;
        strcpy(block[count]->cont, pure_line);

        (*b_count)++;

}

int make_edge(E *edge[], int *e_count, int *b_count) {

        int count = *e_count;
        int b_c = *b_count;

        edge[count] = (E *)malloc(sizeof(E));

        edge[count]->e_num = *e_count;
        edge[count]->start_b = b_c - 1;
        edge[count]->dest_b = b_c;

        (*e_count)++;

        return b_c;

}
```

2.2.10 make_block

Input	Trigger, block[], b_count, pure_line[]
Output	
Description	
1. Dynamically allocate new block space 2. give newly incremented block number 3. Give block contents from pure line array	

2.2.11 make_edge

Input	Trigger, edge[],e_count,b_count
Output	Block number
Description	
1. Dynamically allocate new edge space 2. Give newly created edge number 3. Give new edge starting block number 4. Give new edge destination block number	



```
void show_edge_info(E *edge[], int *e_count, FILE *txt) {

        int count = *e_count;

        printf("\n");
        printf("[Edge Info]  ");
        printf("Edge Number: %d,  ", edge[count - 1]->e_num);
        printf("Start Block Number: %d --->  ", edge[count - 1]->start_b);
        printf("Destination Block Number: %d  ", edge[count - 1]->dest_b);

        fprintf(txt, "[Edge Info]  Edge Number: %d,  Start Block Number: %d --->
  Destination Block Number: %d \n", edge[count - 1]->e_num, edge[count - 1]->sta
rt_b, edge[count - 1]->dest_b);

}

void show_block_info(B *block[], int *b_count, FILE *txt) {

        int count = *b_count;

        printf("\n");
        printf("[Block Info]  ");
        printf("Block Number: %d,  ", block[count - 1]->b_num);
        printf("Contents: %s", block[count - 1]->cont);

        fprintf(txt, "[Block Info]  Block Number: %d,  Contents: %s \n", block[c
ount - 1]->b_num, block[count - 1]->cont);

}
```

2.1.8 show_edge_info

Input	Trigger, Edge[], e_count, txt
Output	Display command, Report command
Description	
1.Show user edge`s all information 2.Write the edge information to text report file using file output	

2.1.7 show_block_info

Input	Trigger,Block[],b_count, txt
Output	Display command, Report command
Description	
1.Show user block`s all information 2.write block information to the text report file using file output	

Demonstration