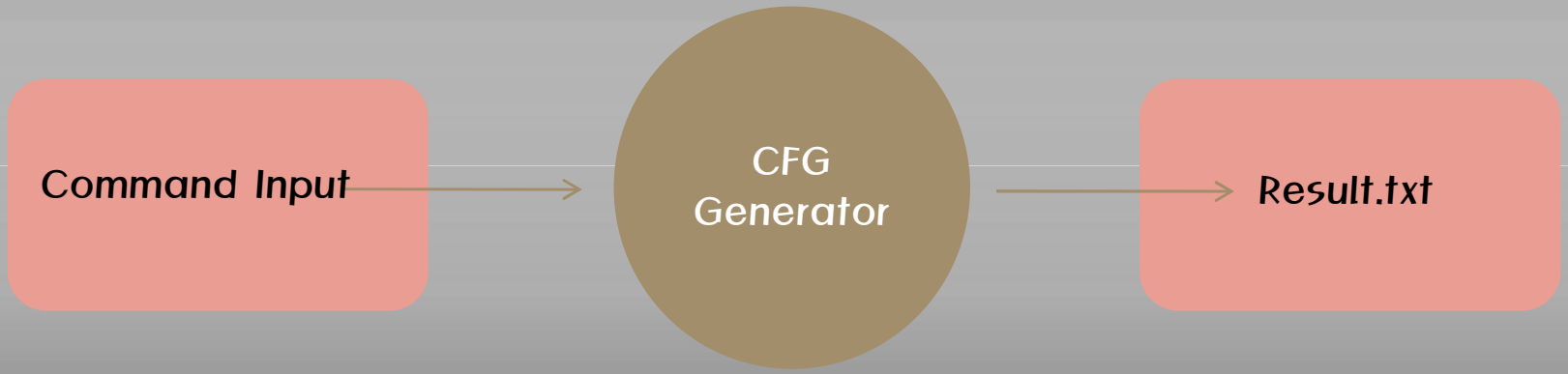
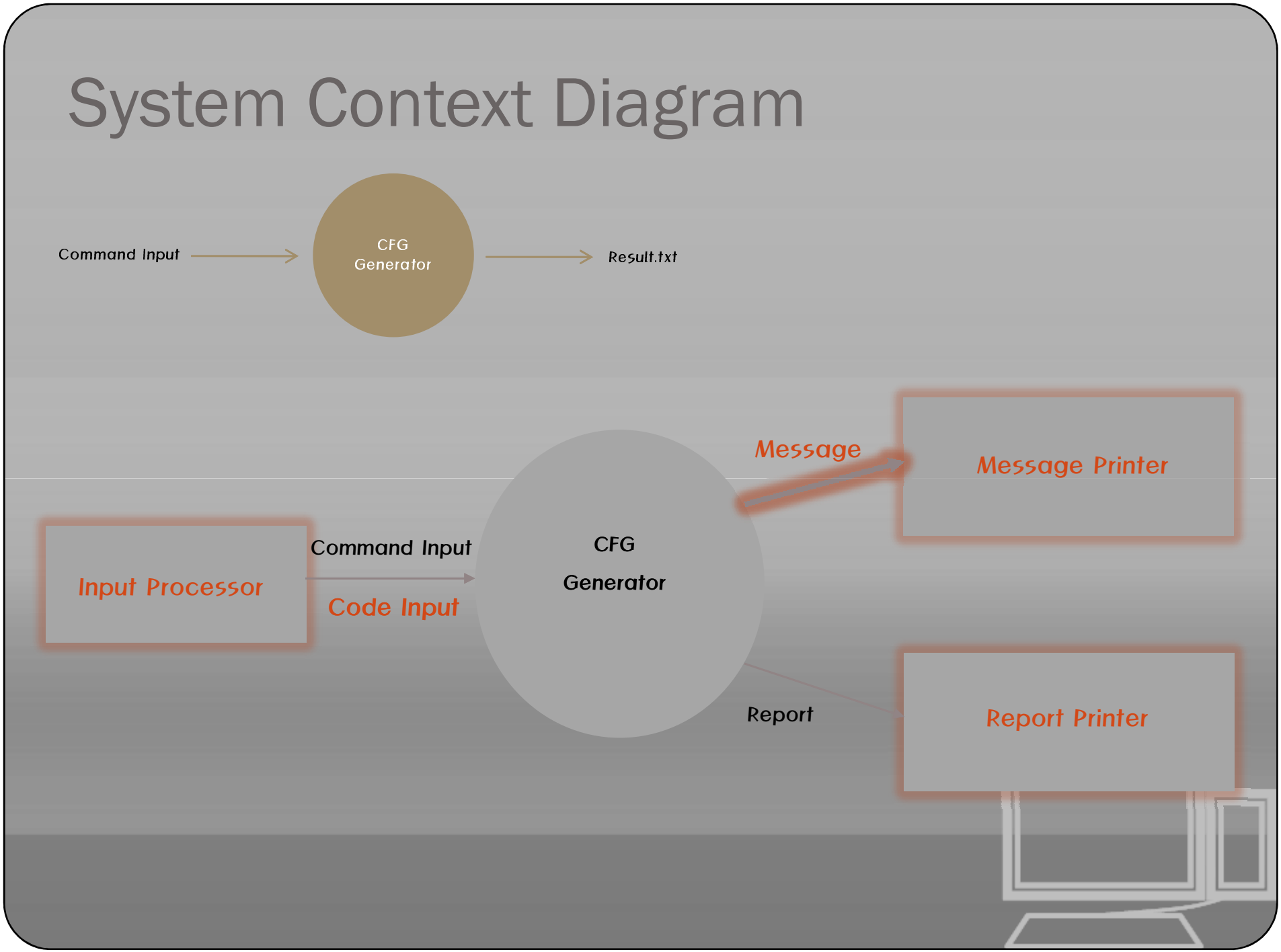
CFG Final Presentation

Team.9
201011308 김필제
201011325 고명준
200911408 이대희

System Context Diagram



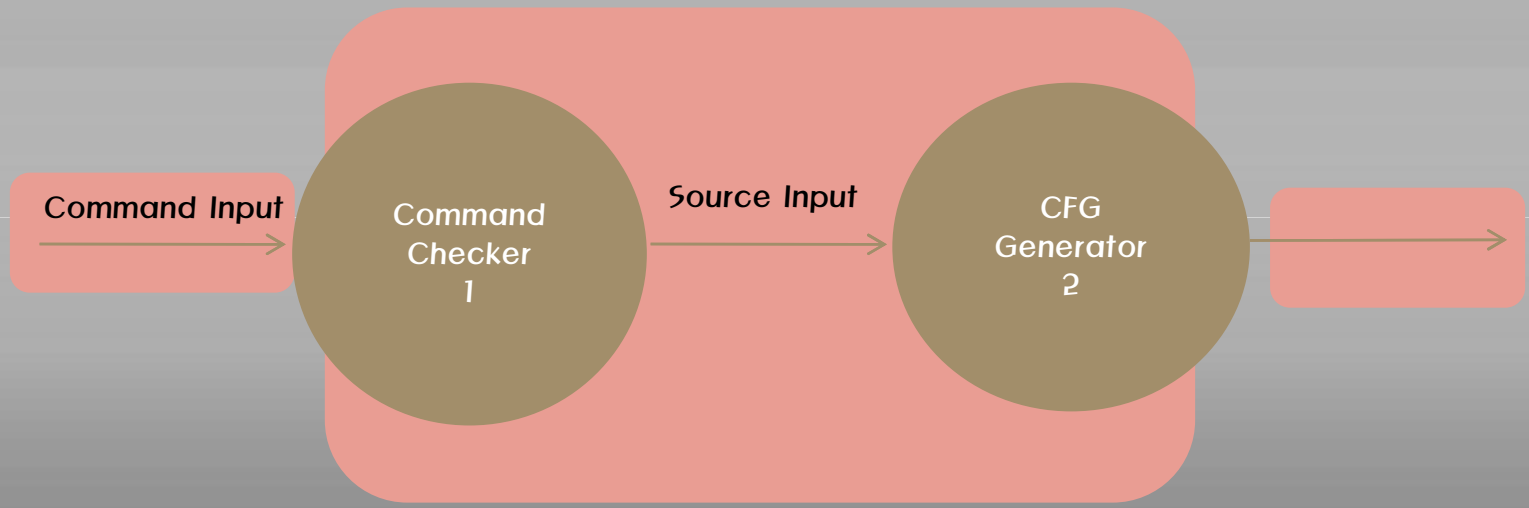
System Context Diagram



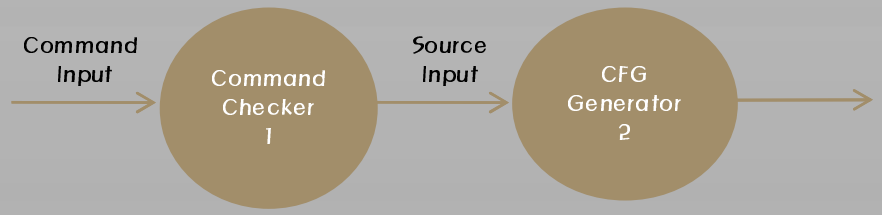
System Context Diagram_Data Dictionary

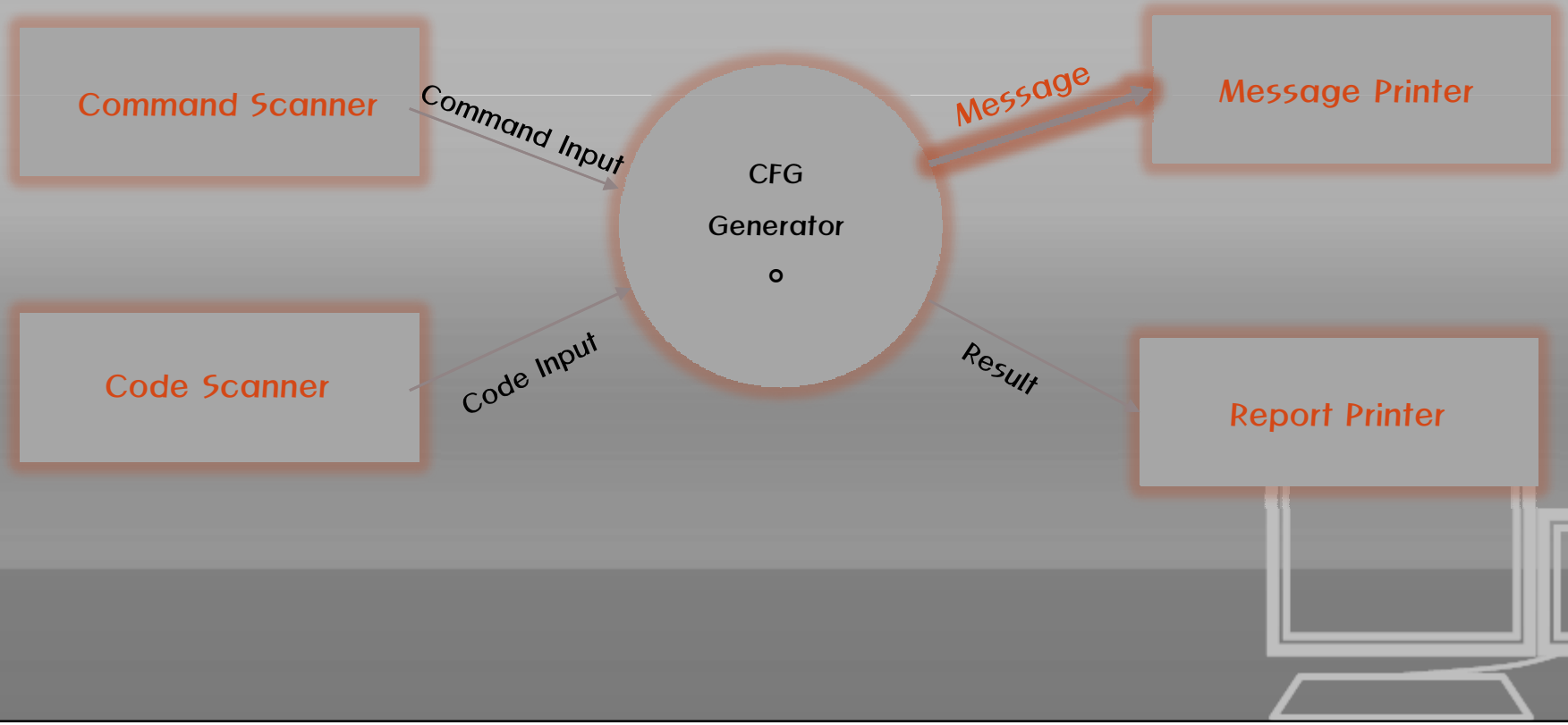
Input / Output Event	Description
Command Input	Receive the Command input from Input Processor.
Code Input	Receive the Code input from Input Processor.
Message	System message that should display on the Monitor.
Report	Informations that are converted into CFG.

DFD Level0


Command Input
Command
Checker
1
Source Input
CFG
Generator
2

DFD Level0


Command Input
Command Checker 1
Source Input
CFG Generator 2


Command Scanner
Command Input
CFG Generator o
Message
Message Printer
Code Scanner
Code Input
Result
Report Printer

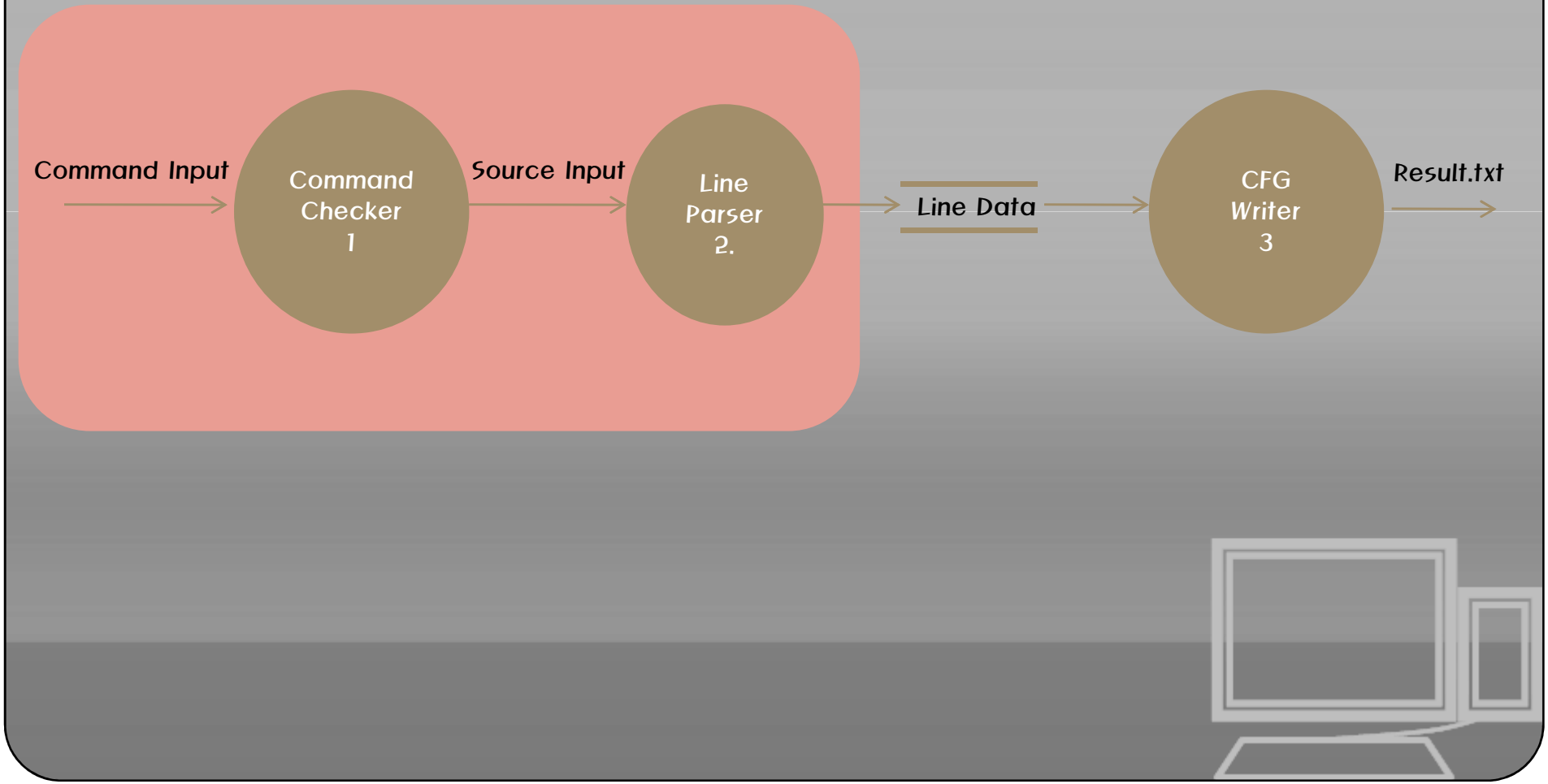
DFD Level0 Data Dictionary

Input / Output Event	Description	format
Command Input	Receive the Command input from Command Scanner.	String
Code Input	Receive the Code input from Code Scanner.	C file
Message	System message that should display on the Monitor.	String
Result	Informations that are converted into CFG.	txt.file

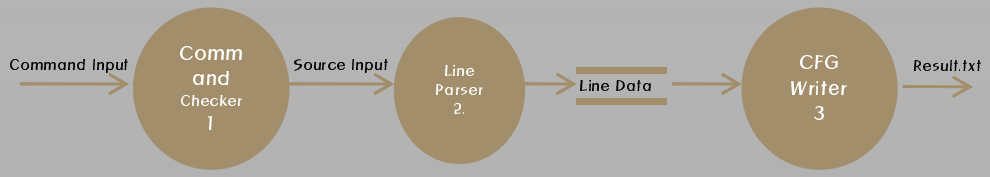
DFD Level0 Process specification

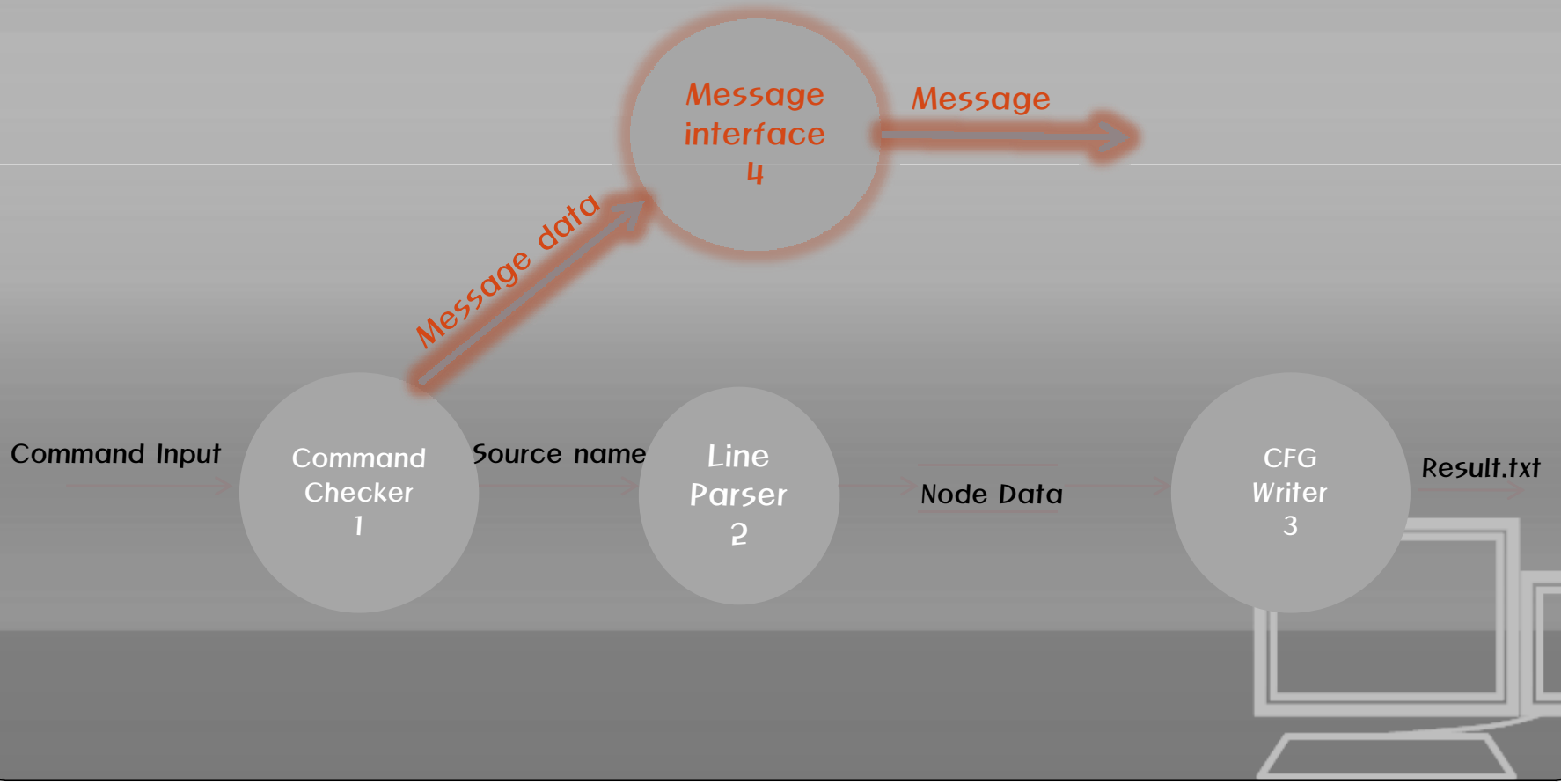
Reference NO.	0
Name	CFG Generator
Input	Command Input, Code Input
Output	Message, Report
Process Description	Receive the Command Input and Code Input and then, check or convert them into message and report output.

DFD Level1


Command Input
Command Checker 1
Source Input
Line Parser 2.
Line Data
CFG Writer 3
Result.txt

DFD Level1


Command Input
Comm and Checker 1
Source Input
Line Parser 2.
Line Data
CFG Writer 3
Result.txt


Message interface 4
Message
Message data
Command Input
Command Checker 1
Source name
Line Parser 2
Node Data
CFG Writer 3
Result.txt

DFD Level1 Data Dictionary

Input / Output Event	Description	Format
Line Data	String Data parsed from C code line in Source file	String

DFD Level1 Process Specification

Reference NO.	1
Name	Command Checker
Input	Command Input
Output	Message Data, Source name
Process Description	Receive Command Input. And analys the vaild of command input. After that, send a Message data to message interface and source name to line parser.

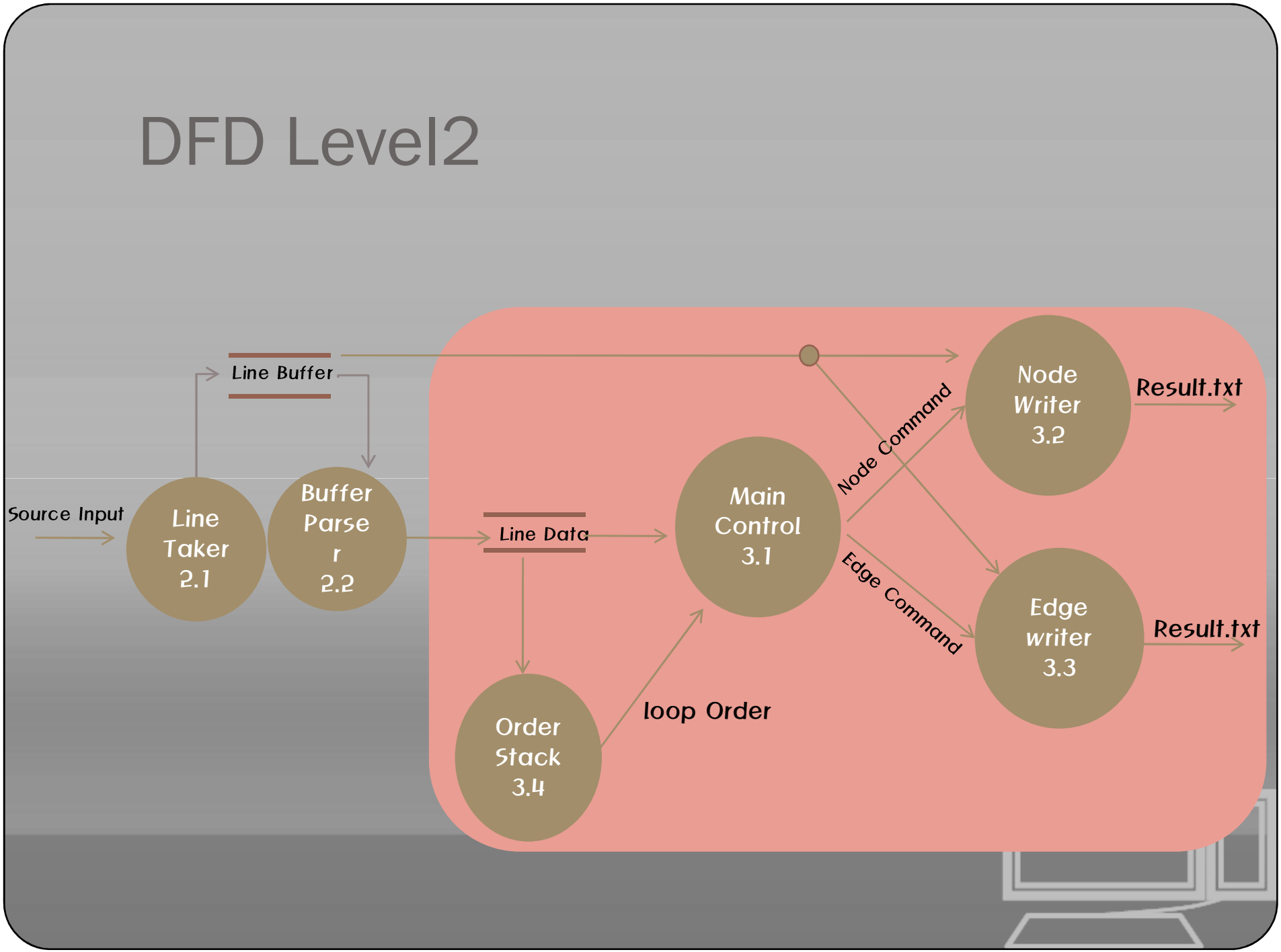
Reference NO.	2
Name	Line parser
Input	Source name
Output	Node data
Process Description	Get line by line in C source, parse data need to make CFG

DFD Level1 Process Specification

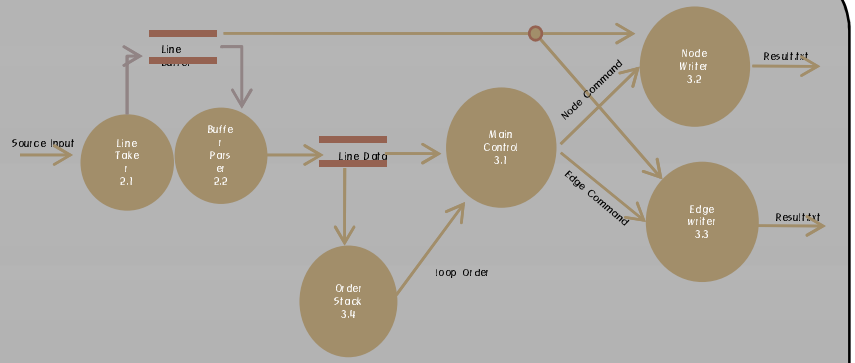
Reference NO.	3
Name	CFG writer
Input	Node data
Output	Result.txt
Process Description	Receive Node Data, draw CFG corresponding each case

Reference NO.	4
Name	Message interface
Input	Message Data
Output	Message
Process Description	Convert Message data into Message that can be send to Message printer.

DFD Level2


Line Buffer
Source Input
Line Taker 2.1
Buffer Parse r 2.2
Line Data
Main Control 3.1
Node Command
Edge Command
Node Writer 3.2
Result.txt
Edge writer 3.3
Result.txt
Order Stack 3.4
loop Order

DFD Level2



Command Input

Command Checker 1.1

Message data

Message interface 4.1

Message

Source name

Line Taker 2.1

Line Buffer

Buffer Parser 2.2

Node Data

Node Writer 3.2

Node_info

Edge Process 3.1

Edge_info

Result interface 3.3

Result.txt

DFD Level2 Data Dictionary

Input / Output Event	Description	format
Command Input	String that inputed by Commander.	String
Source Name	Name that should be converted into CFG.	String
Message Data/Message	System message that should display on the Monitor.	String
Report	Informations that are converted into CFG.	Txt.file
Node Info	Include the node info for CFG	Struct Array
Edge Info	Include the node info for CFG	String Array
Result.txt	Informations that are converted into CFG.	Txt file

DFD Level2 Process Specification

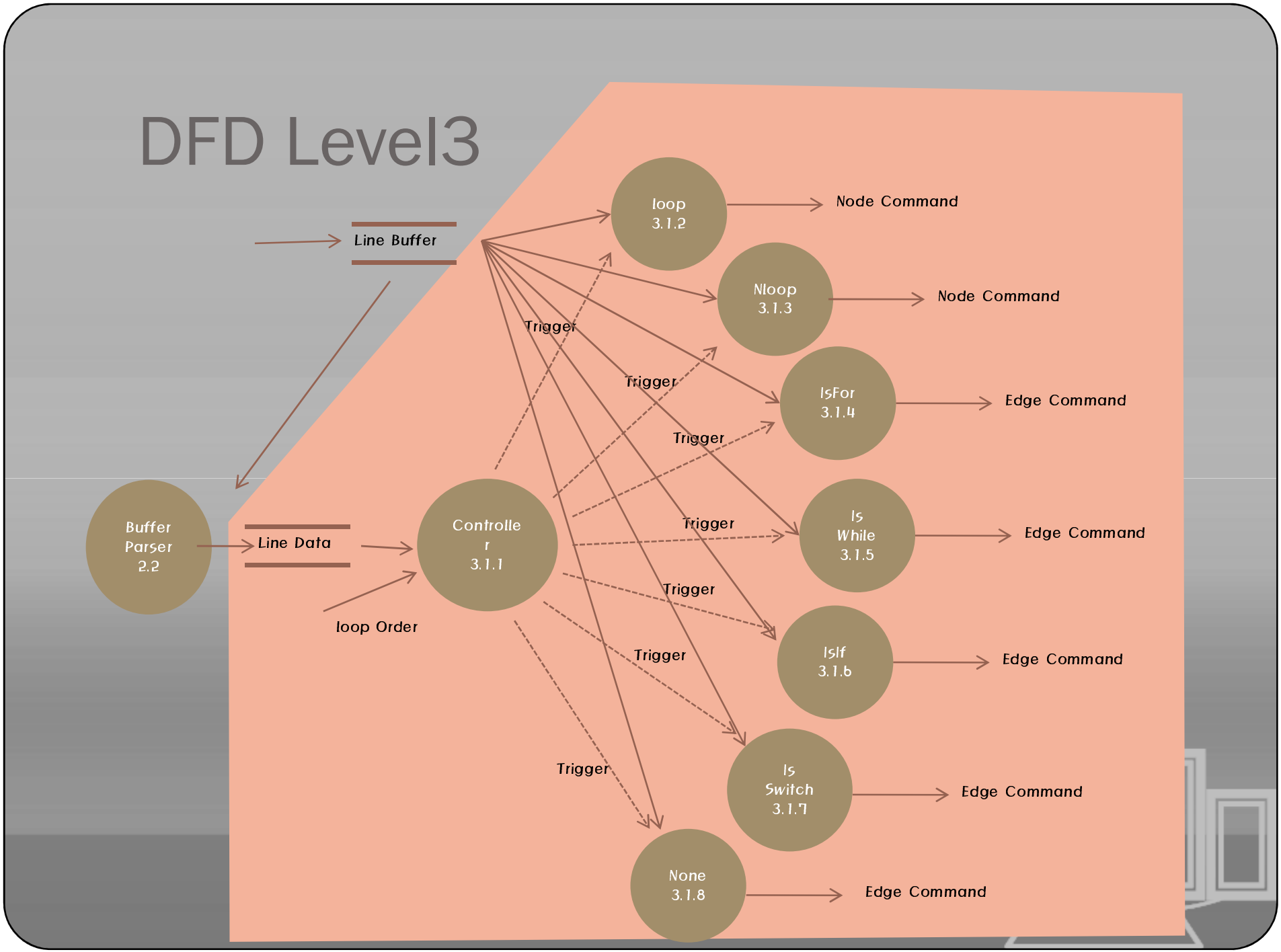
Reference NO.	2.1
Name	Line Taker
Input	Source Name
Output	Line Data
Process Description	Get line by line in C source

Reference NO.	2.2
Name	Buffer Parser
Input	Line Buffer
Output	Line Data
Process Description	Get buffer data and parse data need to make CFG

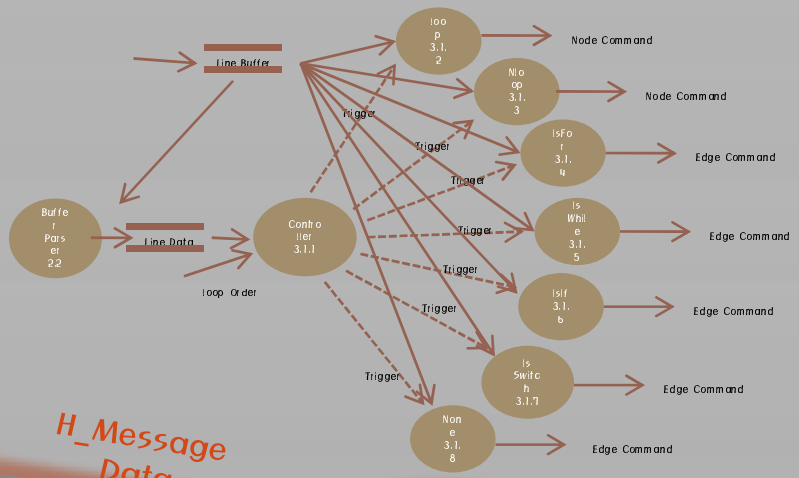
DFD Level2 Process Specification

Reference NO.	3.1
Name	Edge Process
Input	Line Data
Output	Edge info
Process Description	Receive the line Data and covert it into Edge info. And then send the Edge info to result interface.

Reference NO.	3.2
Name	Node Writer
Input	line data
Output	Node info
Process Description	Draw node line to result file corresponding each case


DFD Level3
Line Buffer
Buffer Parser 2.2
Line Data
loop Order
Controller 3.1.1
loop 3.1.2
Nloop 3.1.3
IsFor 3.1.4
Is While 3.1.5
IsIf 3.1.6
Is Switch 3.1.7
None 3.1.8
Trigger
Trigger
Trigger
Trigger
Trigger
Trigger
Trigger
Node Command
Node Command
Edge Command
Edge Command
Edge Command
Edge Command
Edge Command

DFD Level3



Command Input

Main Controller 1.1.1

Trigger

Print Help 1.1.2

Trigger

Print Fail 1.1.3

Trigger

Print Success 1.1.4

Trigger

Print Start 1.1.5

H_Message Data

F_Message Data

Su_Message Data

St_Message Data

Message Interface 4.1.1

Message

Source name

Line Taker 2.1.1

Line Buffer

Buffer Parser 2.2.1

Node Data

Edge Maker 3.1.1

Node Data2

Node Writer 3.2.1

Edge writer 3.1.2

Node_info

Edge_info

Result interfac e 3.3.1

Result.txt

DFD Level3 Process specification

Reference NO.	1.1.1
Name	Main Controller
Input	Command Input
Output	Trigger
Process Description	Trigger proper Printer.

Reference NO.	1.1.2
Name	Print Help
Input	Trigger
Output	H_Message data
Process Description	Send H_Message data to Message Interface.

DFD Level3 Process specification

Reference NO.	1.1.3
Name	Print fail
Input	Trigger
Output	F_Message data
Process Description	Send F_Message data to Message Interface.

Reference NO.	1.1.4
Name	Print Success
Input	Trigger
Output	Su_Message data
Process Description	Send Su_Message data to Message Interface.

DFD Level3 Process specification

Reference NO.	1.1.5
Name	Print Start
Input	Trigger
Output	St_Message data
Process Description	Send St_Message data to Message Interface.

Reference NO.	4.1.1
Name	Message interface
Input	Message Data
Output	Message
Process Description	Convert Message data into Message that can be send to Message printer.

DFD Level3 Process specification

Reference NO.	2.1.1
Name	Line Taker
Input	Source Name
Output	Node Data
Process Description	Get line by line in C source

Reference NO.	2.2.1
Name	Buffer Parser
Input	Line Buffer
Output	Node Data
Process Description	Get buffer data and parse data need to make CFG

level3 Process specification

Reference NO.	3.1.1
Name	Edge Maker
Input	Node Data
Output	Node Data2
Process Description	Convert the Inputed line data into output Node data2 that is about Edge.

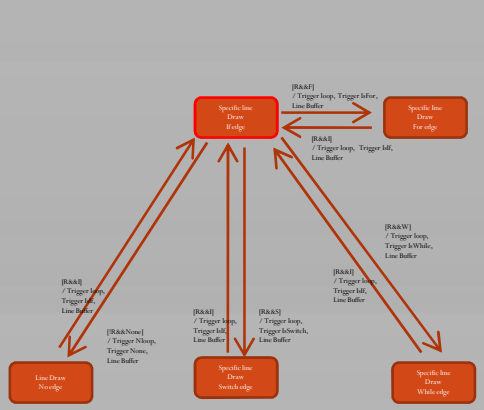
Reference NO.	3.1.2
Name	Edge writer
Input	Node Data2
Output	Edge_Info
Process Description	Receive the Node data2 and convert it into Edge_info.

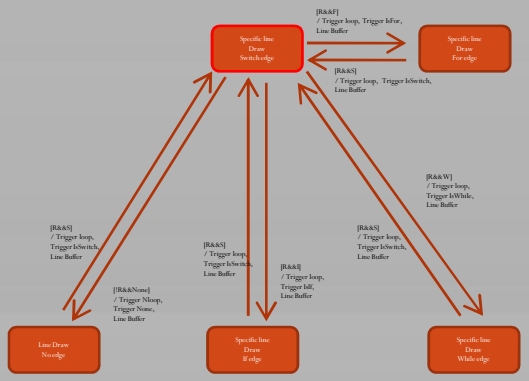
DFD Level3 Process specification

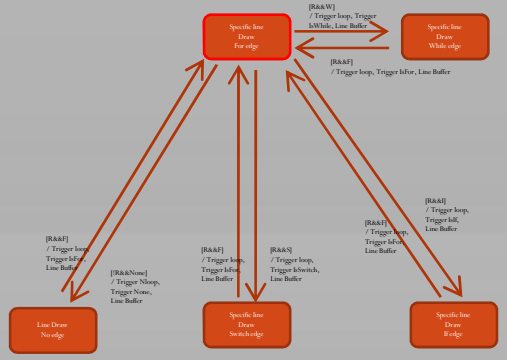
Reference NO.	3.2.1
Name	Node Writer
Input	Node data
Output	Node_info
Process Description	Draw node line to result file corresponding each case

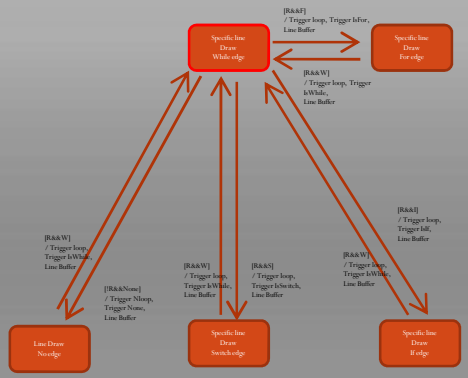
Reference NO.	3.3.1
Name	Result interface
Input	Node_info, Edge_info
Output	Result.txt
Process Description	Send the proper information that will be converted into CFG Report.txt

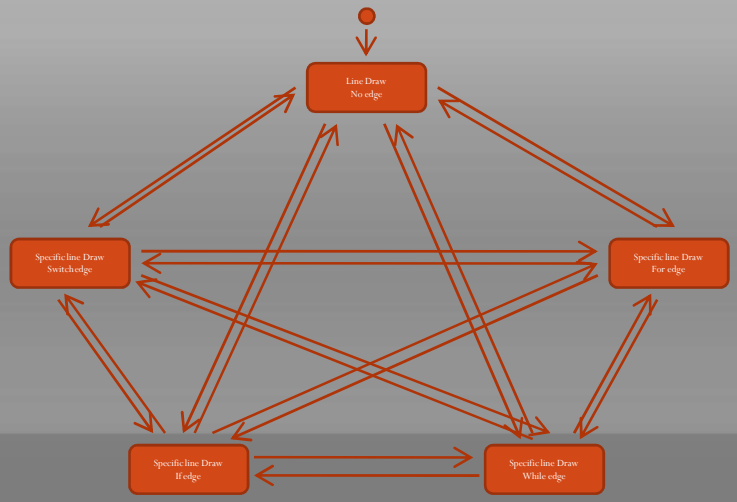
DFD Level 4

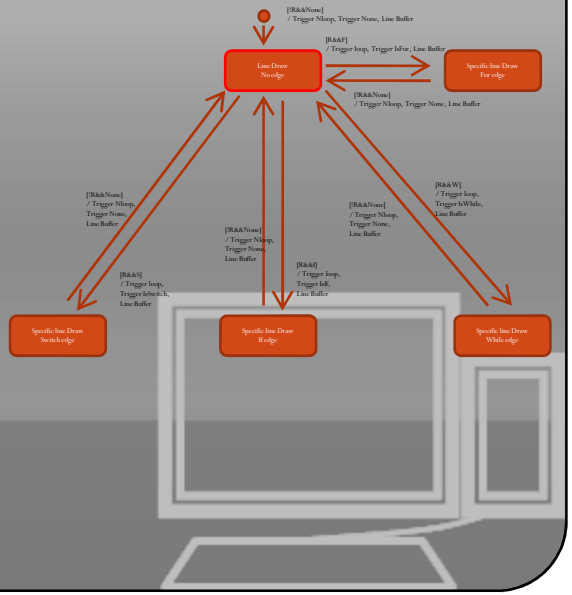

Specific line Draw If edge
[R&&F] / Trigger loop, Trigger IsFor, Line Buffer
Specific line Draw For edge
[R&&I] / Trigger loop, Trigger IsIf, Line Buffer
[R&&I] / Trigger loop, Trigger IsIf, Line Buffer
[R&&W] / Trigger loop, Trigger IsWhile, Line Buffer
[R&&I] / Trigger loop, Trigger IsIf, Line Buffer
[R&&I] / Trigger loop, Trigger IsIf, Line Buffer
[R&&S] / Trigger loop, Trigger IsSwitch, Line Buffer
[!R&&None] / Trigger Nloop, Trigger None, Line Buffer
Line Draw No edge
Specific line Draw Switch edge
Specific line Draw While edge


Specific line Draw Switch edge
[R&&F] / Trigger loop, Trigger IsFor, Line Buffer
Specific line Draw For edge
[R&&S] / Trigger loop, Trigger IsSwitch, Line Buffer
[R&&W] / Trigger loop, Trigger IsWhile, Line Buffer
[R&&S] / Trigger loop, Trigger IsSwitch, Line Buffer
[R&&S] / Trigger loop, Trigger IsSwitch, Line Buffer
[R&&I] / Trigger loop, Trigger IsIf, Line Buffer
[!R&&None] / Trigger Nloop, Trigger None, Line Buffer
Line Draw No edge
Specific line Draw If edge
Specific line Draw While edge

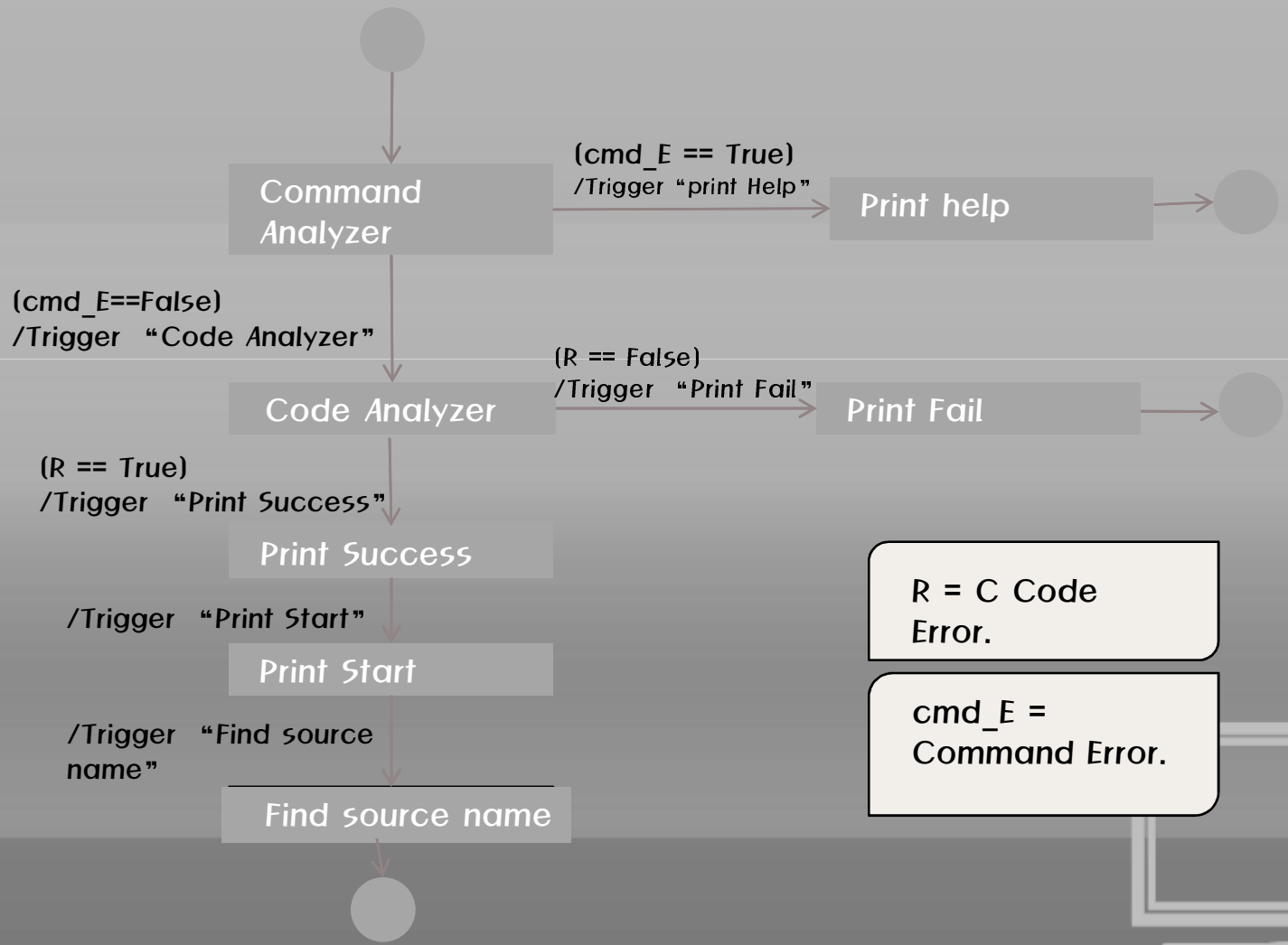

Specific line Draw For edge
[R&&W] / Trigger loop, Trigger IsWhile, Line Buffer
Specific line Draw While edge
[R&&F] / Trigger loop, Trigger IsFor, Line Buffer
[R&&I] / Trigger loop, Trigger IsIf, Line Buffer
[R&&F] / Trigger loop, Trigger IsFor, Line Buffer
[R&&F] / Trigger loop, Trigger IsFor, Line Buffer
[R&&S] / Trigger loop, Trigger IsSwitch, Line Buffer
[!R&&None] / Trigger Nloop, Trigger None, Line Buffer
Line Draw No edge
Specific line Draw Switch edge
Specific line Draw If edge


Specific line Draw While edge
[R&&F] / Trigger loop, Trigger IsFor, Line Buffer
Specific line Draw For edge
[R&&W] / Trigger loop, Trigger IsWhile, Line Buffer
[R&&I] / Trigger loop, Trigger IsIf, Line Buffer
[R&&W] / Trigger loop, Trigger IsWhile, Line Buffer
[R&&W] / Trigger loop, Trigger IsWhile, Line Buffer
[R&&S] / Trigger loop, Trigger IsSwitch, Line Buffer
[!R&&None] / Trigger Nloop, Trigger None, Line Buffer
Line Draw No edge
Specific line Draw Switch edge
Specific line Draw If edge

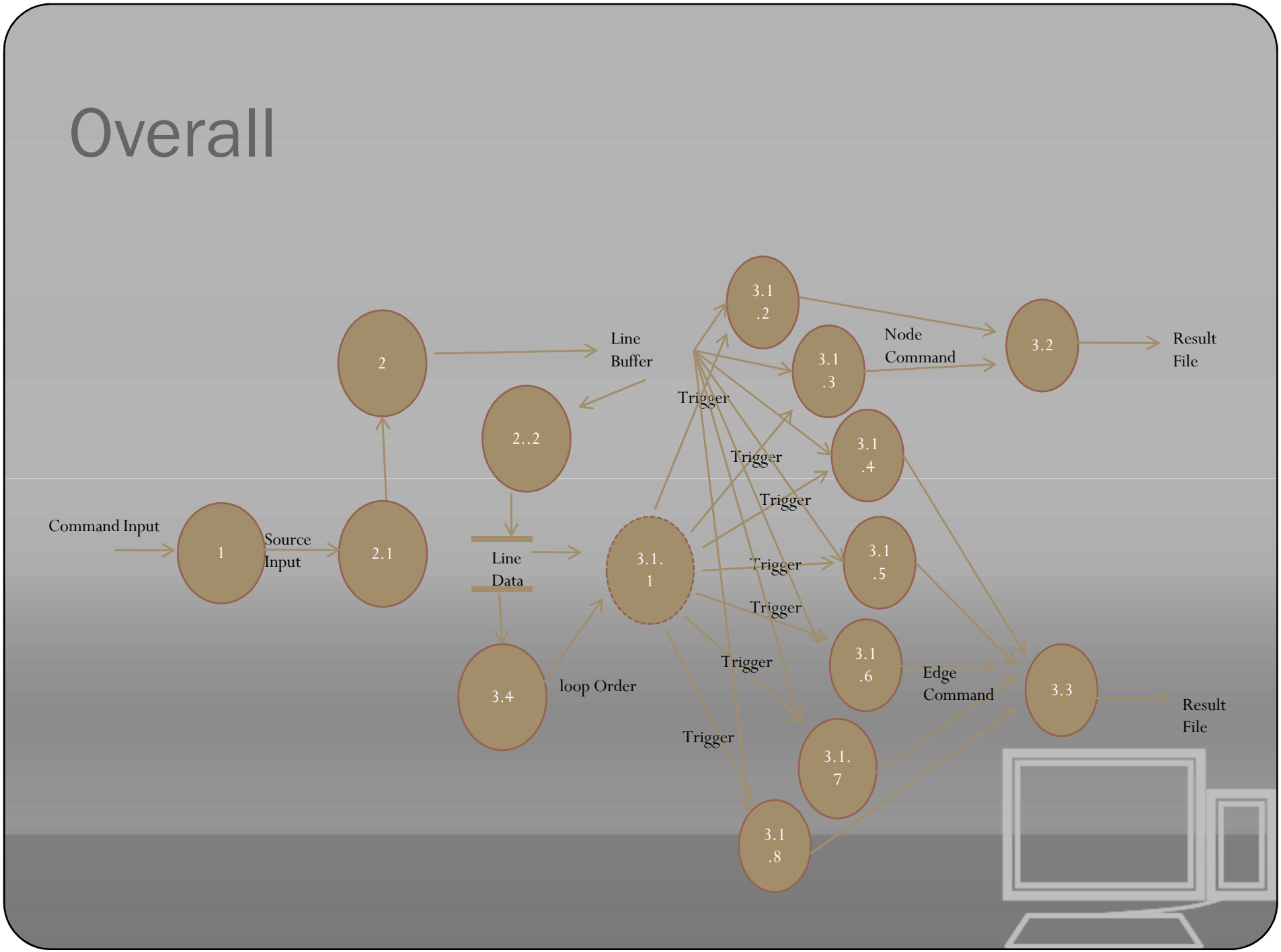

Line Draw No edge
Specific line Draw Switch edge
Specific line Draw For edge
Specific line Draw If edge
Specific line Draw While edge


[!R&&None] / Trigger Nloop, Trigger None, Line Buffer
Line Draw No edge
[R&&F] / Trigger loop, Trigger IsFor, Line Buffer
Specific line Draw For edge
[!R&&None] / Trigger Nloop, Trigger None, Line Buffer
[R&&W] / Trigger loop, Trigger IsWhile, Line Buffer
[!R&&None] / Trigger Nloop, Trigger None, Line Buffer
[!R&&None] / Trigger Nloop, Trigger None, Line Buffer
[R&&S] / Trigger loop, Trigger IsSwitch, Line Buffer
[R&&I] / Trigger loop, Trigger IsIf, Line Buffer
Specific line Draw Switch edge
Specific line Draw If edge
Specific line Draw While edge

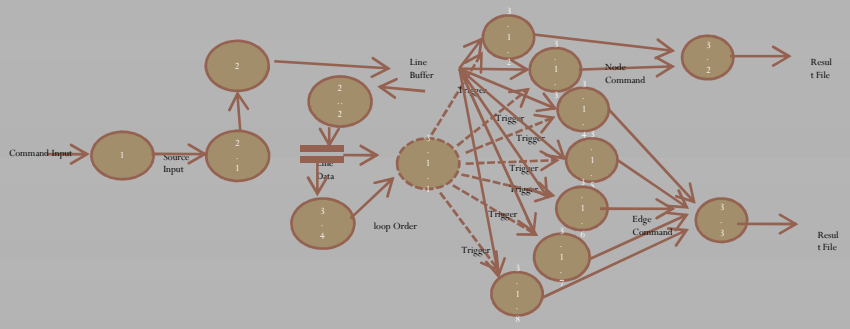
DFD Level4

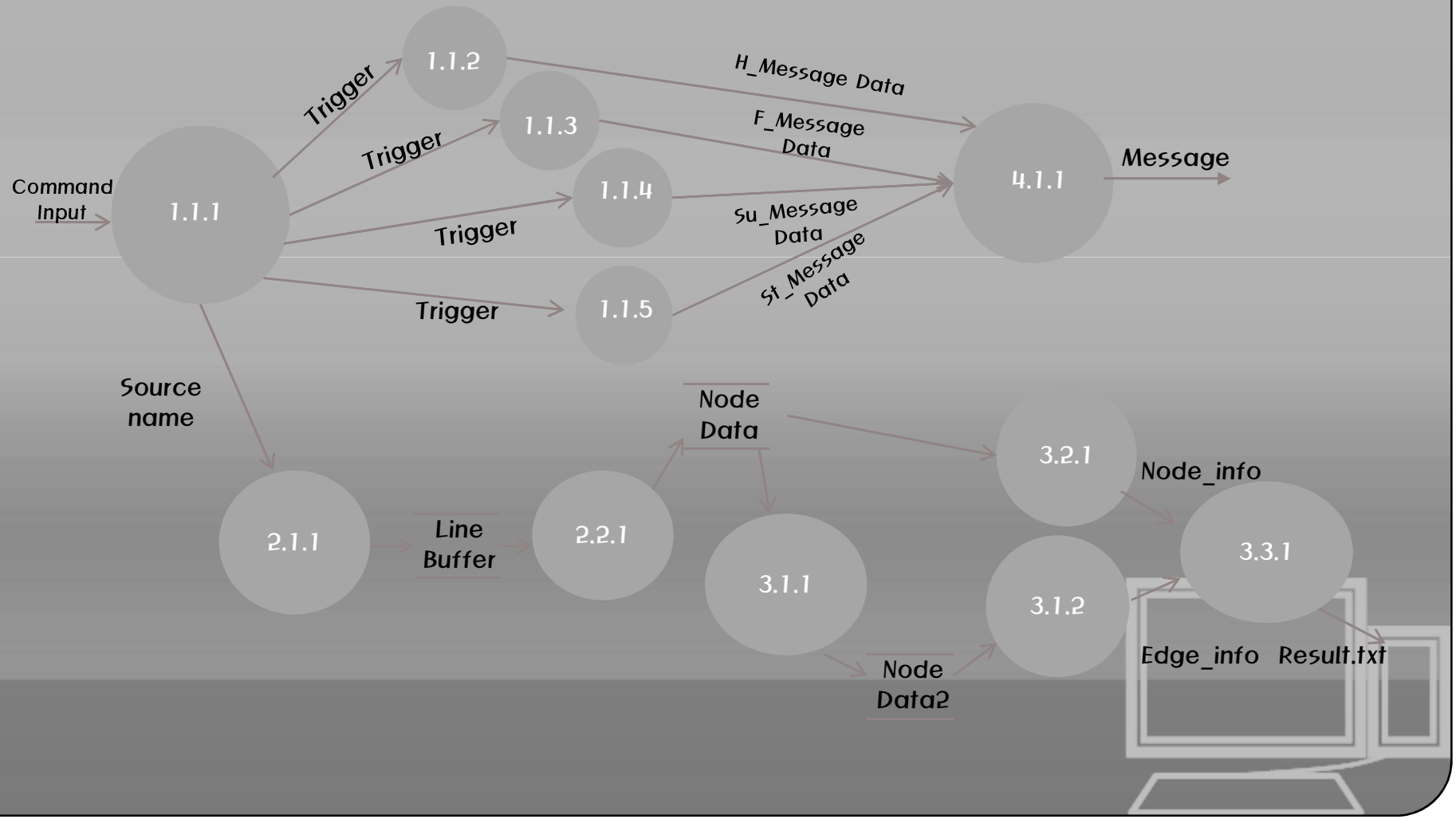

Command Analyzer
(cmd_E == True)
/Trigger "print Help"
Print help
(cmd_E==False)
/Trigger "Code Analyzer"
Code Analyzer
(R == False)
/Trigger "Print Fail"
Print Fail
(R == True)
/Trigger "Print Success"
Print Success
/Trigger "Print Start"
Print Start
/Trigger "Find source name"
Find source name
R = C Code Error.
cmd_E = Command Error.

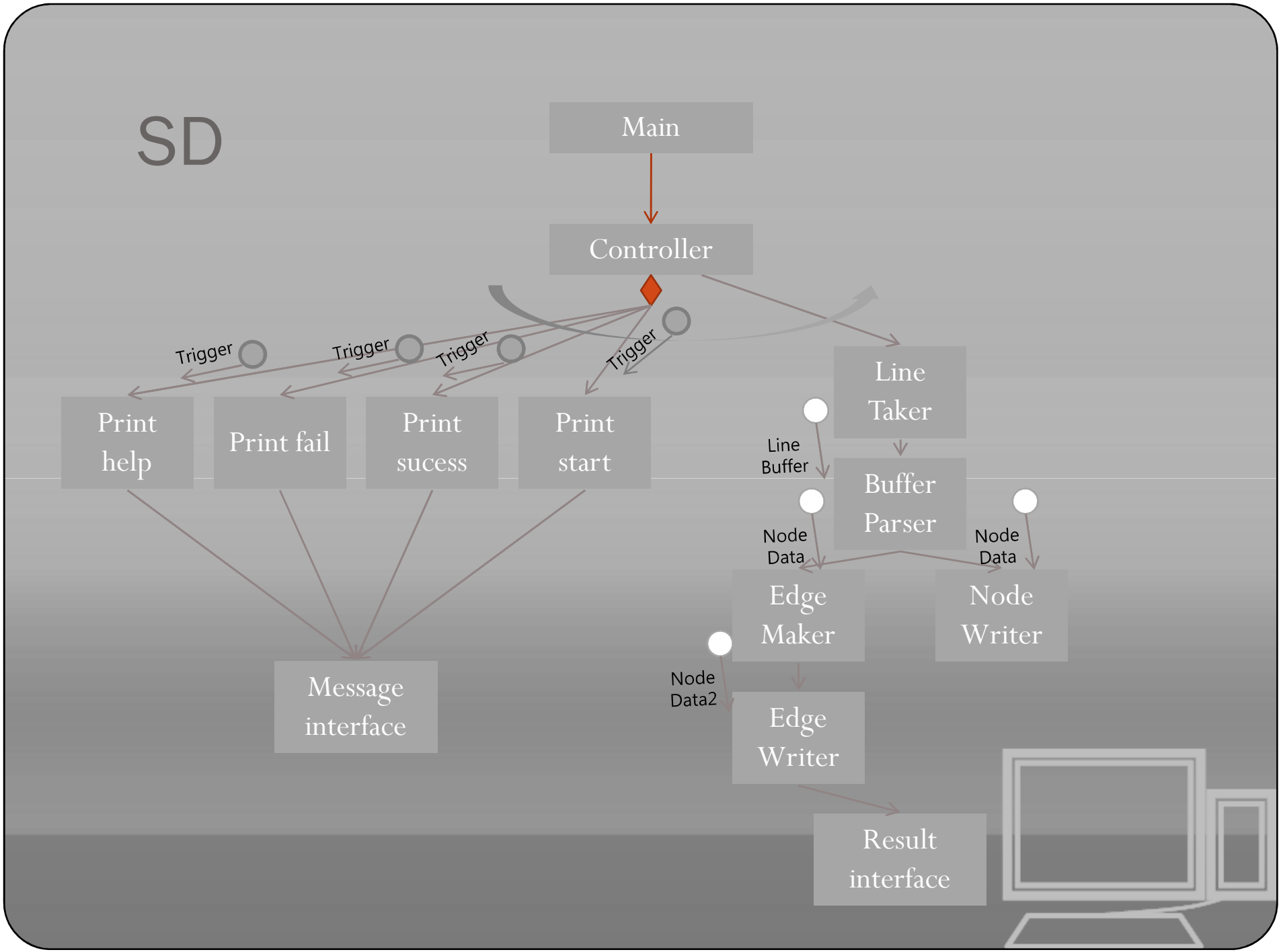
Overall


Command Input
1
Source Input
2.1
2
Line Buffer
2..2
Line Data
3.4
loop Order
3.1. 1
Trigger
3.1 .2
3.1 .3
3.1 .4
3.1 .5
3.1 .6
3.1. 7
3.1 .8
Node Command
3.2
Result File
Edge Command
3.3
Result File

Overall




Command Input
1.1.1
Trigger
1.1.2
Trigger
1.1.3
Trigger
1.1.4
Trigger
1.1.5
H_Message Data
F_Message Data
Su_Message Data
St_Message Data
4.1.1
Message
Source name
2.1.1
Line Buffer
2.2.1
Node Data
3.1.1
Node Data2
3.2.1
3.1.2
Node_info
3.3.1
Edge_info
Result.txt


SD
Main
Controller
Trigger
Trigger
Trigger
Trigger
Print help
Print fail
Print sucess
Print start
Line Taker
Line Buffer
Buffer Parser
Node Data
Node Data
Edge Maker
Node Writer
Node Data2
Edge Writer
Message interface
Result interface

About Code

Code Explain

```
struct data{
        char data(50);
        int start;
        int end;
};
```

Save Node and Edge data

```
char *command=malloc(100);
```

Save a command input.

```
char *source_name;
```

Save source name that will be converted into CFG.

```
char *output_name=malloc(30);
```

Save output name

```
char *buffer(250)={NULL};
```

Save each code's line.

```
Int line_n, Node_n, Edge_n;
```

The number of code's line, node and edge

```
struct data Node_data(250) = {NULL};
```

Save some information of Node.

```
struct data Edge_data(250) = {NULL};
```

Save some information of Edge.

Method *command_checker()

```
char *command_checker(char * command){

    //command의 문자열 길이를 알아낸후 띄어쓰기를 이용하여 #./CG Inputcode.c result.txt 를 각각 나눔
    size = strlen(command);
    for( i = 0; i < size; i++){
        if(command[i] == ' '){
            j++;
            n = 0;
        }
        else if(j == 0 )    check1[n++] = command[i];
        else if(j == 1)     check2[n++] = command[i];
        else if(j == 2)     check3[n++] = command[i];
    }

    //제대로 입력 되어있는지 확인중
    if(strcmp(check1, "#./CG") != 0){
        check_n = 1;
    }
    size = strlen(check2);
    if(check2[size-2] != '.' || check2[size-1] != 'c'){
        size = strlen(check3);
        check_n = 1;
    }
    size = strlen(check3);
    if(check3[size-4] != '.' || check3[size-3] != 't' || check3[size-2] != 'x' || check3[size-1] != 't'){
        check_n = 1;
    }
    // 잘못 입력 받았으면 헬프를 띄운후 종료
    if(check_n == 1){
        print_help();
        exit(0);
    }
    //화일이 있는지 확인
    fp=fopen(check2,"r");
    if(fp==NULL){
        print_fail();
        exit(0);
    }

    print_success();
    print_start();

    return check2;
}
```

Method Name	*command_checker
Input	char * command(command string)
Output	Check2(source_name that will be converted into CFG.)
Function	Get a command string. And check the command's validation. If command have some problem, call proper method(print_help or print_fail). If There aren't any problem, call proper method(print_success or printf_start).

Method linetaker()

```
int linetaker(char *source_name, char *buffer[]){

    int i = 0;
    FILE *fp;
    fp = fopen(source_name, "r");
    if(fp==NULL)
    {
        printf("연결안됨");
    }

    //한줄씩 입력받으면서 buffer에 한줄씩 저장함
    while(1){
        char *a=malloc(50);
        memset(a,0,50);
        if(!fgets(a,50,fp))
            break;
        buffer[i++] = a;
    }

    return (i-1);
}
```

Method Name	linetaker
Input	char *source_name, char *buffer()
Output	i-1
Function	Get a source_name, and open file that have same name with source_name. If there isn't that file, print "Not connected" . If that file exists, save each line in buffer array.

Method bufferparser()

```
int bufferparser(char *buffer[], struct data Node_data[], int line_n){

    int i, j, k, h, len, n_Node = 0, N = 0;
    //이부분에서 { } 이걸로 노드를 하나하나씩 Node_data에 저장함 return 값은 Node_data의 총 저장되는 노드의 개수
    for(i = 0; i <= line_n; i++){
        len = strlen(buffer[i]);
        for(j = 0; j < len; j++){
            if(buffer[i][j] == '{'){
                n_Node++;
                h = 0;
                for(k = 0; k < len-2; k++){
                    if(buffer[i][k] != '₩t')
                        Node_data[N].data[h++] = buffer[i][k];
                }
                Node_data[N].start = i;
                for(k = i+1; k <= line_n; k++){
                    len = strlen(buffer[k]);
                    for(h = 0; h < len; h++){
                        if(buffer[k][h] == '{'){
                            n_Node++;
                        }
                        else if(buffer[k][h] == '}'){
                            n_Node--;
                        }
                    }
                    if (n_Node == 0)
                        break;
                }
                Node_data[N].end = k;
                N++;
            }
        }
    }

    return (N-1);
}
```

Method Name	bufferparser
Input	char *buffer(), struct data Node_data(), int line_n
Output	N-1 (
Function	Get a code' s each line(buffer()), buffer' s number of line(line_n) and a struct variable(struct data Note_data()). Informations of nodes will save in this struct variable(Struct data Node_data()). At last return the number of nodes(N-1).

Method edgemaker()

```
int edgemaker(struct data Node_data[], struct data Edge_data[], int Node_n){

    int i, j, k, end = 0, N = 0;

    // make edge using if and else   , return:number of Edge_data
    for(i = 0; i <= Node_n; i++){
        if(Node_data[i].data[0] == 'i' && Node_data[i].data[1] == 'f'){
            end = 0;
            k = i;
            for(j = i+1; j<= Node_n; j++){
                if(Node_data[j].start > Node_data[k].end){
                    if(Node_data[j].data[0] == 'e' && Node_data[j].data[1] == 'l' && Node_data[j].data[2] == 's' && Node_data[j].data[3] == 'e'){
                        k = j;
                    }
                    else{
                        end = j;
                        break;
                    }
                }
            }
            k = i;
            for(j = i+1; j <= end; j++){
                if(Node_data[j].start > Node_data[k].end){
                    Edge_data[N].start = k;
                    Edge_data[N].end = j;
                    N++;
                    if(k != (end-1)){
                        Edge_data[N].start = k;
                        Edge_data[N].end = end;
                        N++;
                    }
                    k = j;
                }
                if(k != j){
                    Edge_data[N].start = j-1;
                    Edge_data[N].end = j;
                    N++;
                }
                else if(Node_data[j-1].data[0] != 'i' && Node_data[j-1].data[0] != 'e'){
                    Edge_data[N].start = j-1;
                    Edge_data[N].end = j;
                    N++;
                }
            }
                i = end;
        }

        // except if or else, link nodes
        Edge_data[N].start = i;
        Edge_data[N].end = i+1;
        N++;
    }
    return (N-1);
}
```

Method Name	edgemaker
Input	Struct data Node_data(), struct data Edge_data(), int Node_n
Output	N-1
Function	Get a information of nodes (Node_data()) , the number of nodes (Node_n) and a struct valiable named Edge_data(). This Edge_data variable will have some information of edges. At last, Return the number of edges (N-1).

Method resultinterface()

```
void resultinterface(struct data Node_data[], struct data Edge_data[], int Node_n, int Edge_n, char* output_name){

    int i;
    FILE *fp;
    fp = fopen(output_name, "w");
    if(fp==NULL)
        printf("fail");

    fprintf(fp, "-----------Node---------------₩r₩n");

    for(i = 0; i <= Node_n; i++){
        fprintf(fp, "%2d. (%2d, %2d) %s₩r₩n", i, Node_data[i].start, Node_data[i].end, Node_data[i].data);
    }
    fprintf(fp, "-----------Edge---------------₩r₩n");
    for(i = 0; i <= Edge_n; i++){
        if( i == 0)
            fprintf(fp, "%2d. (start -> %2d)   ", i, Edge_data[i].end);
        else if ( i == Edge_n)
            fprintf(fp, "%2d. (%2d -> end)   ₩r₩n", i, Edge_data[i].start);
        else
            fprintf(fp, "%2d. (%2d -> %2d)   ", i,  Edge_data[i].start, Edge_data[i].end);
        if( (i+1) % 5 == 0)
            fprintf(fp, "₩r₩n");
    }

    printf("report file name : %s₩n", output_name);
}
```

Method Name	resultinterface
Input	Struct data Node_data(), struct data Edge_data(), int Node_n, int Edge_n, char* output_name()
Output	void
Function	Make a result.txt that has a Control Flow Graph.

Execution