# CFG Structured analysis & Structured Design

Team4
200811441 윤동민
200811452 이희봉
Presenter>>200811458 조원진
200811416 김영훈

# Contents

- Structured Analysis
  - Statement of Purpose
  - System Context Diagram
  - Event List
  - Data Flow Diagram
  - Data Dictionary
- Structured Design
  - Structured Chart
- Code Implementation & Explanation
  - Code Analysis & Process Specification

# Structured Analysis

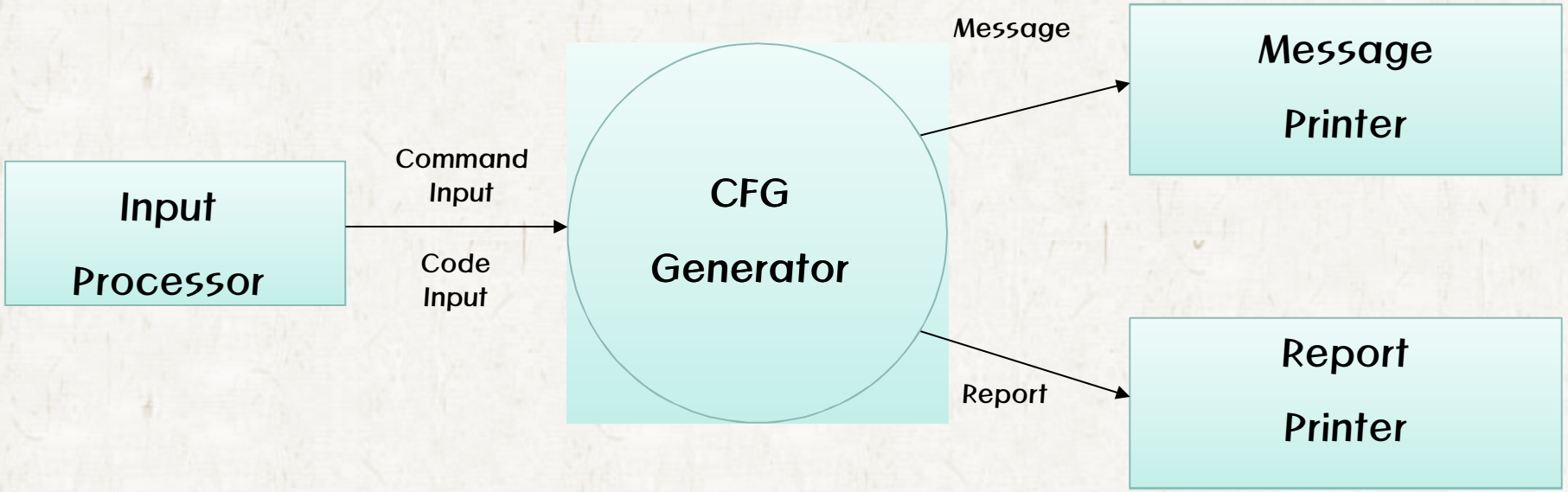
# Statement of Purpose(Modi.)

- It is the program that accept a C code and print CFG
- If user input wrong form of command, program print help message
- The program can optimize code through CFG

# Statement of Purpose(Cont.)

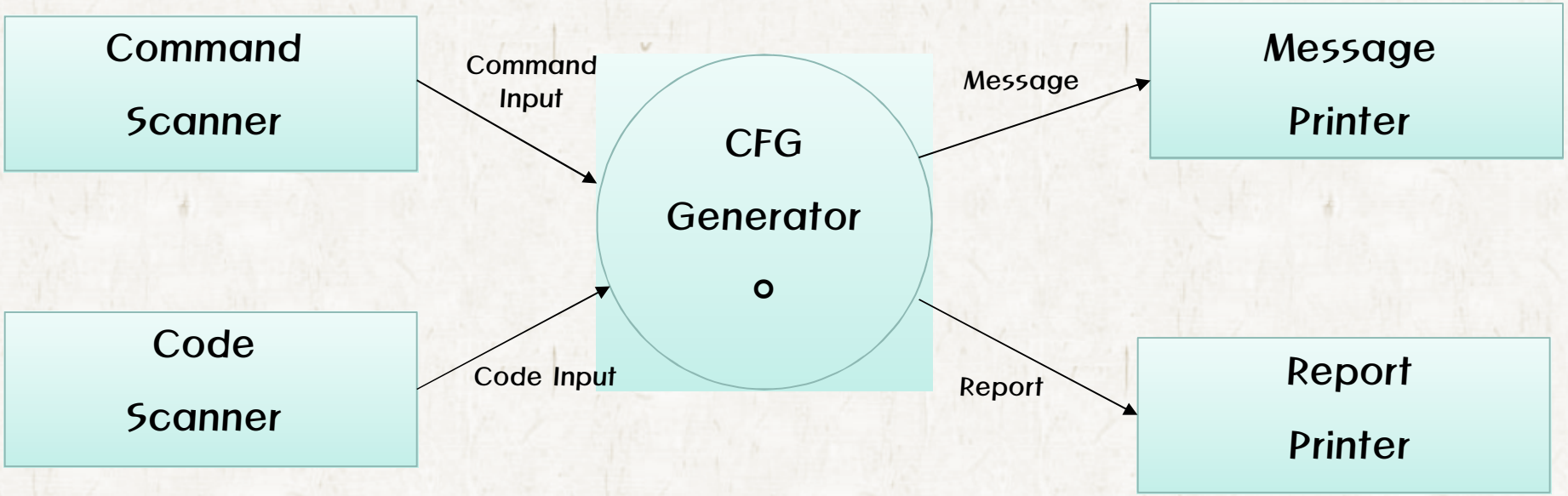
- Constraints
  - Size of C code that was inputed should be less than two hundred line.
  - The code should involve Main Function.
  - C code should be read by single file
  - The program shouldn't operate about header File that user defined
  - The program should operate code that not use pointer
  - All branches start with a '{'
    and ends with '}'
  - A brace should appear on the right side of 'Main Function'
  - A function is declared under Main function.
  - Remove the space between lines of code

# System Context Diagram




| Event List | |
|---|---|
| Input/Output Event | Description |
| Command Input | Receive a command from Input Processor |
| Code Input | Receive a C code from Input Processor |
| Message | Prints system messages to Message Printer |
| Report | Print CFG and .txt file generate |

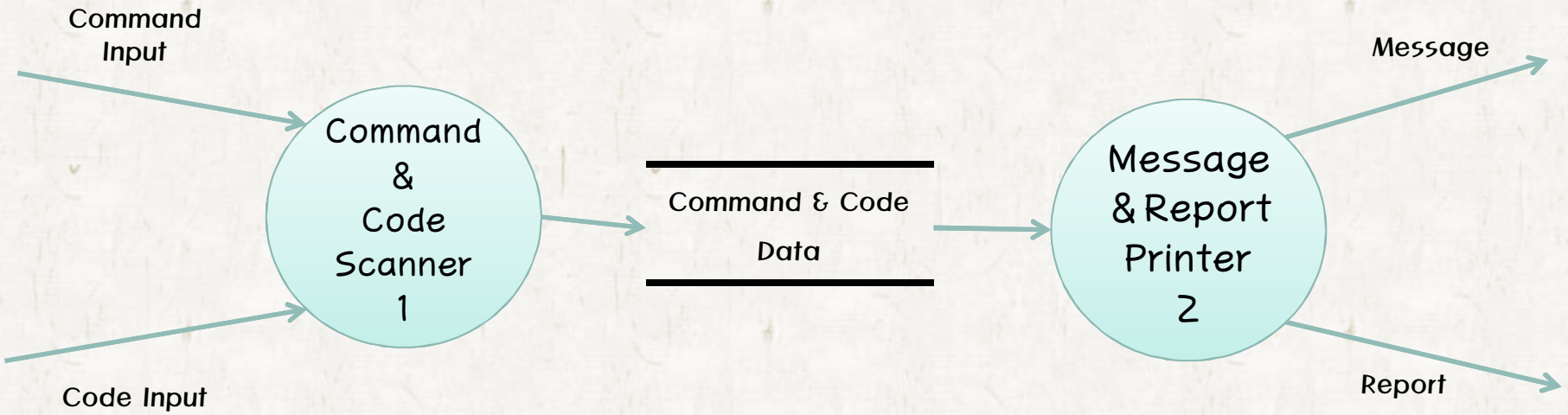
# DFD Level 0


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

# DFD Level 0 – Data Dictionary

| Input/Output | Description | Type |
|---|---|---|
| Command Input | Receive the Command | Char* |
| Code Input | Receive the C code to be converted | .c file |
| Message | Prints system messages to Message Printer | Char* |
| Report | Print CFG and .txt file generate | Char*<br>.txt file |

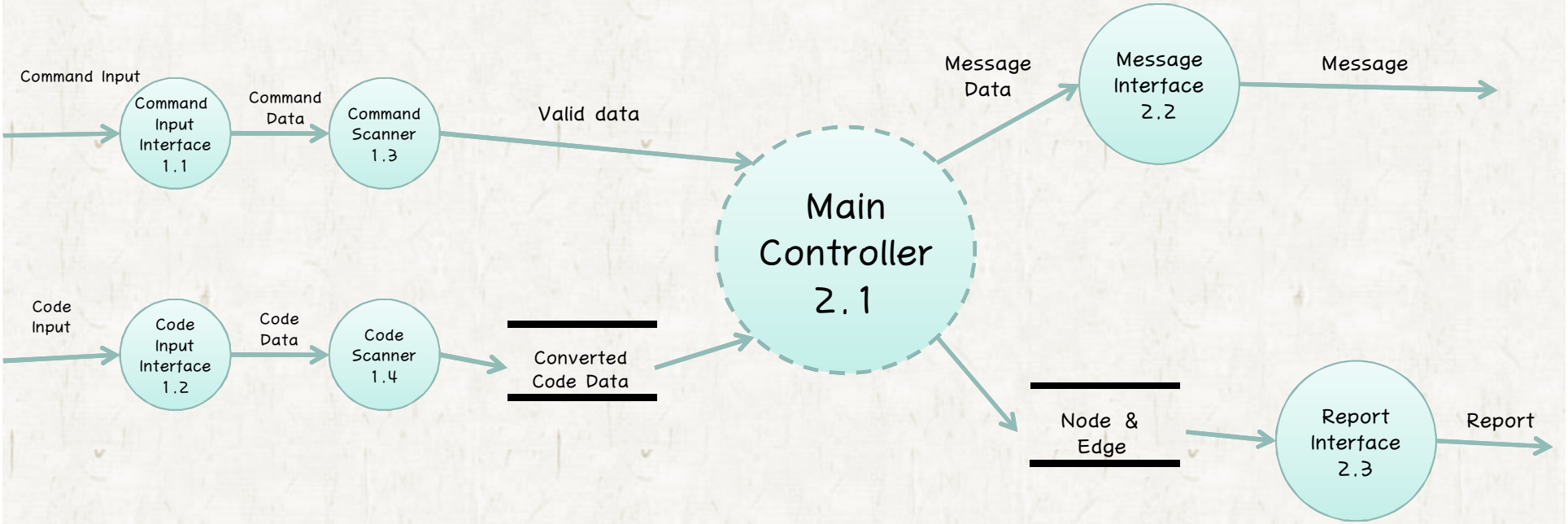
# DFD Level 1



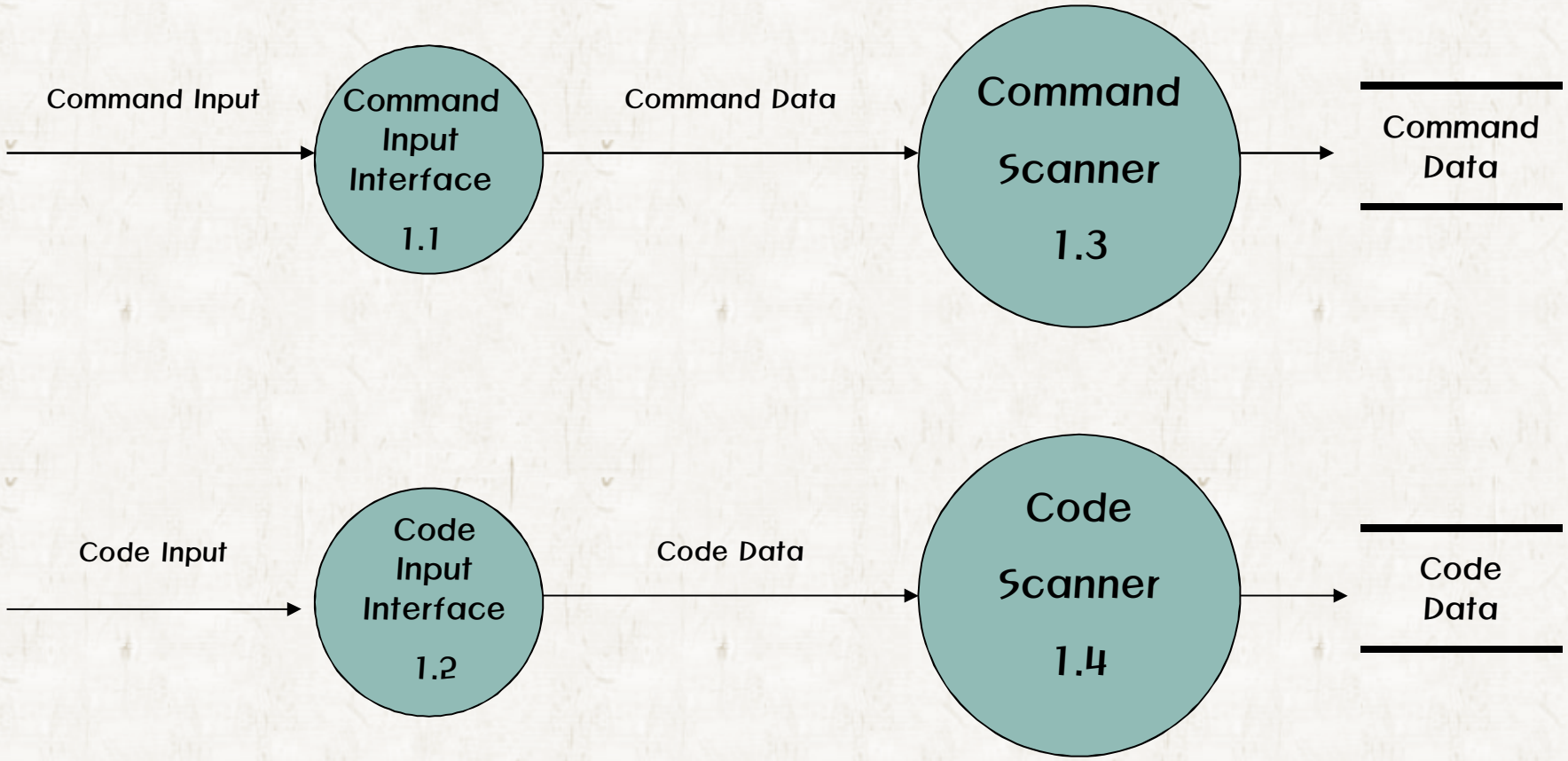

## DFD Level 1 Data Dictionary

| Input/Output Event | Description | Type |
|---|---|---|
| Command & Code Data | Pass the stored information | integer<br>Struct |

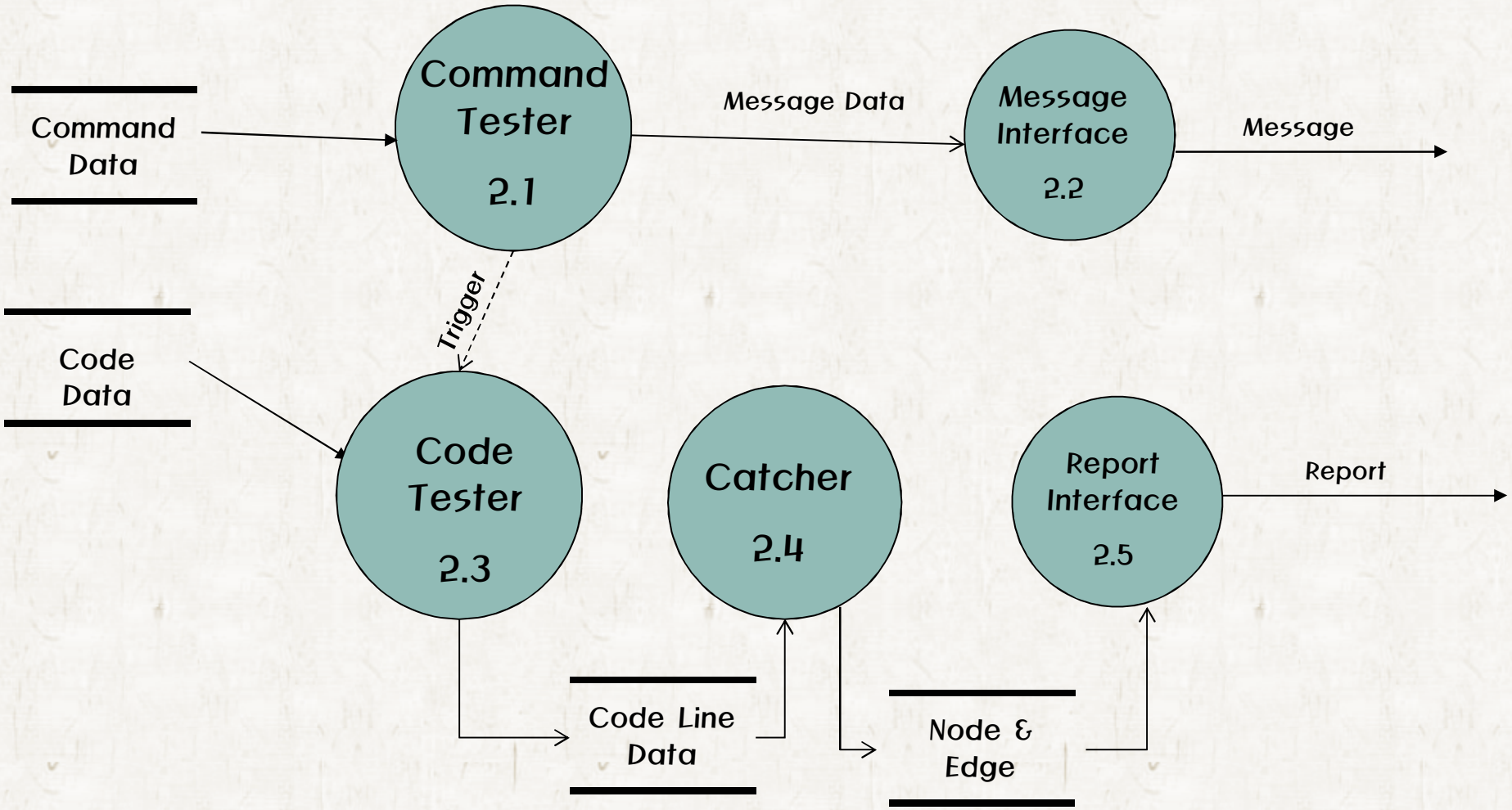
# DFD Level 2


Command Input
Command Input Interface 1.1
Command Data
Command Scanner 1.3
Valid data
Message Data
Message Interface 2.2
Message
Main Controller 2.1
Code Input
Code Input Interface 1.2
Code Data
Code Scanner 1.4
Converted Code Data
Node & Edge
Report Interface 2.3
Report

# DFD Level 2 (1/2)



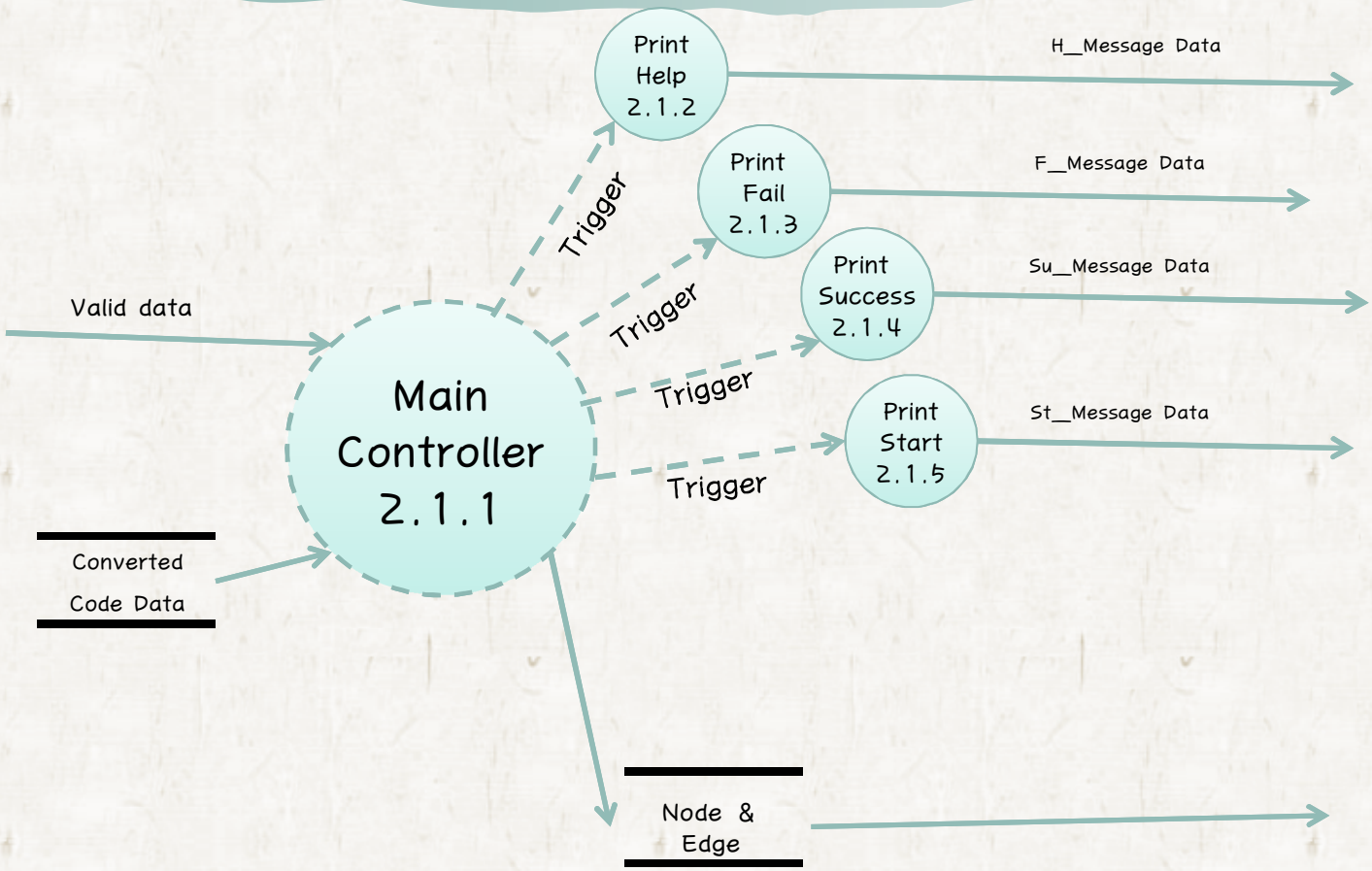
# DFD Level 2 (2/2)



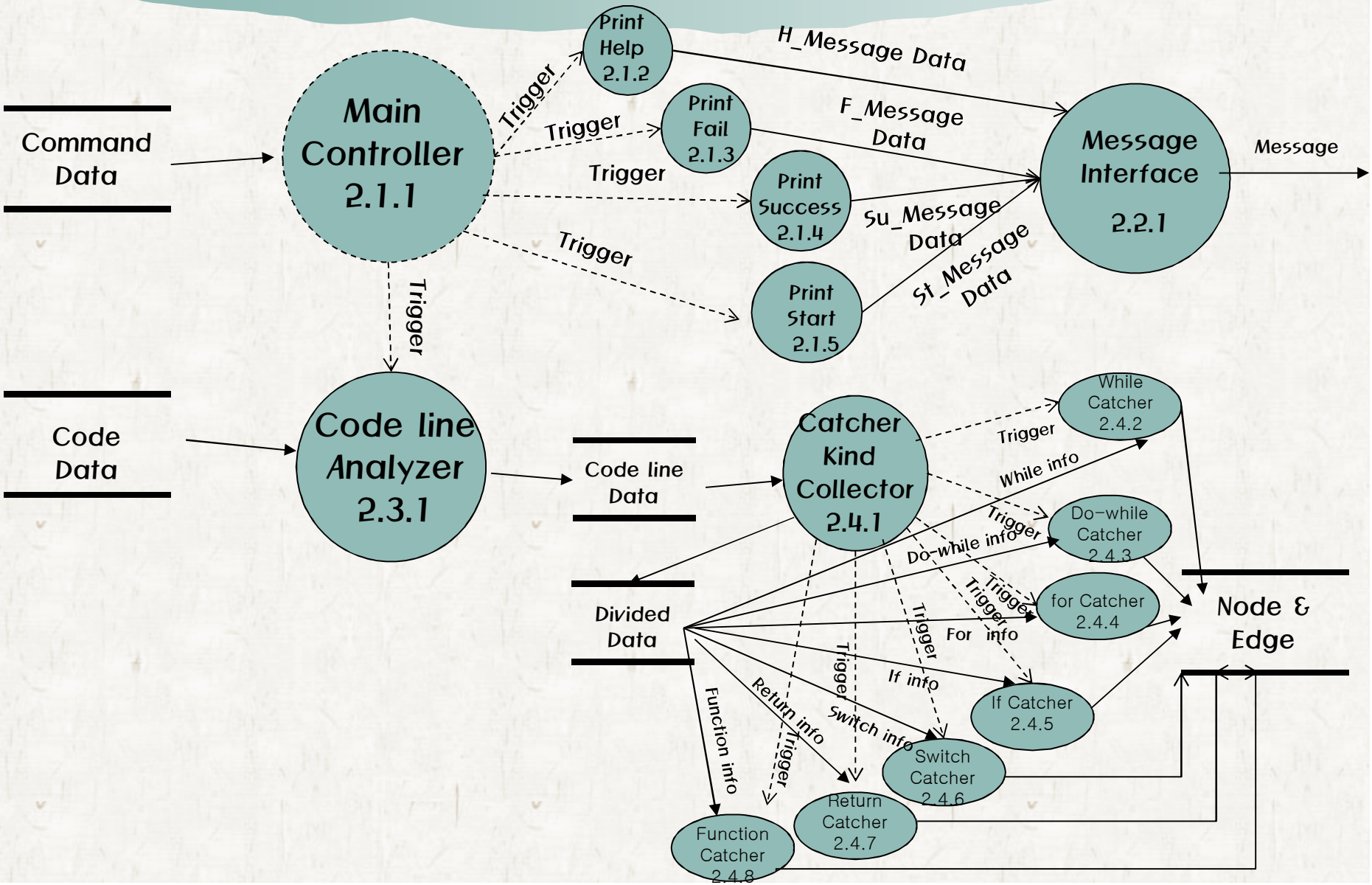
# DFD Level 2 – Data Dictionary

| Input/Output | Description | Type |
|---|---|---|
| Command Data | Pass the stored command data | Char* |
| Valid Data | Pass the integer data to system which can judge it | integer |
| Code Data | Pass the FILE pointer | FILE* |
| Converted Code Data | Pass the Converted Code Data | struct |
| Message Data | Pass the Message Data | Char* |
| Node & Edge | Pass the Stored Node & Edge Data | Char*[] |

# DFD Level 3


Print
Help
2.1.2
H__Message Data
Print
Fail
2.1.3
F__Message Data
Print
Success
2.1.4
Su__Message Data
Print
Start
2.1.5
St__Message Data
Trigger
Trigger
Trigger
Trigger
Valid data
Main
Controller
2.1.1
Converted
Code Data
Node &
Edge

# DFD Level 3

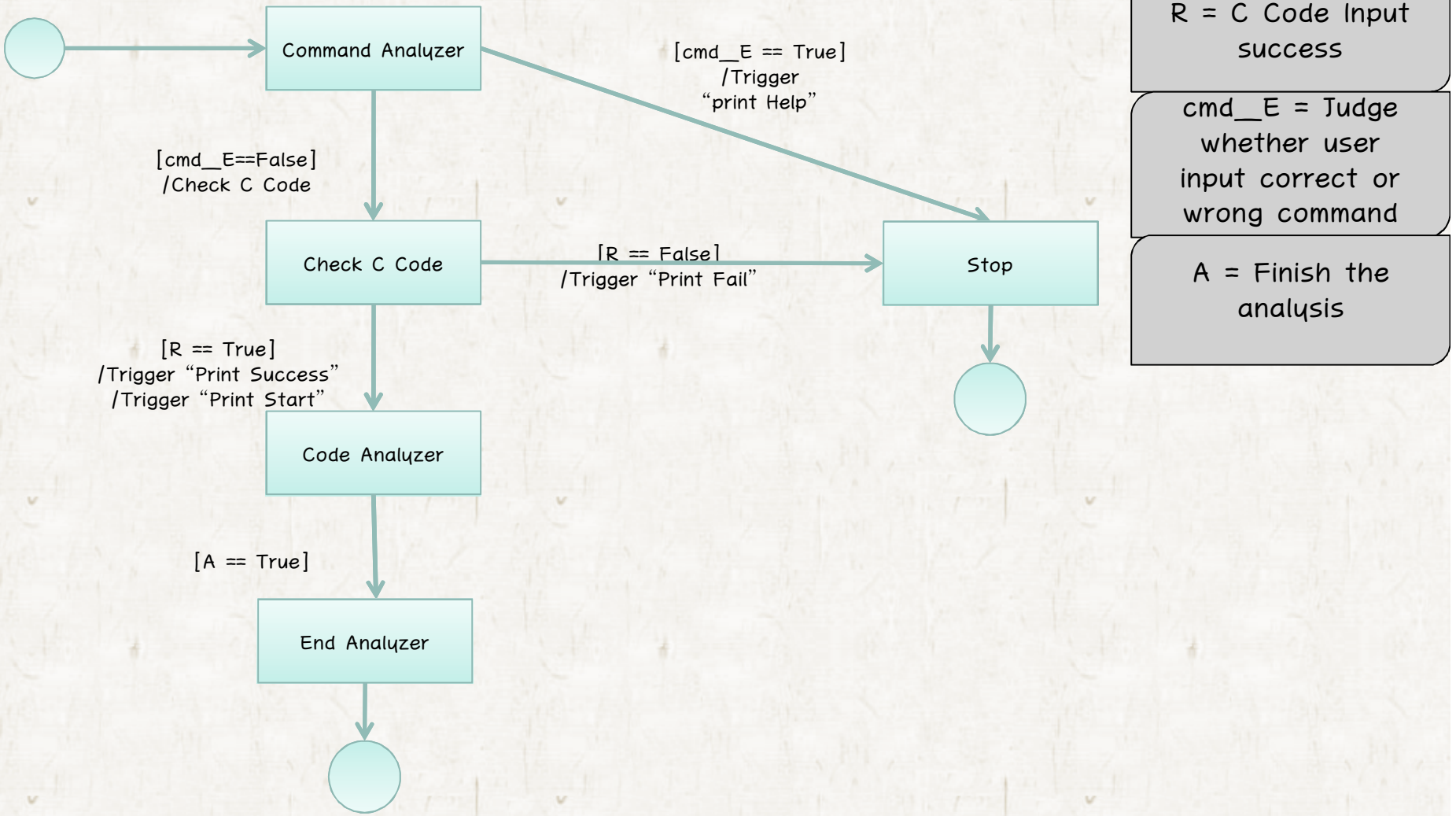

Command Data
Main Controller 2.1.1
Print Help 2.1.2
Print Fail 2.1.3
Print Success 2.1.4
Print Start 2.1.5
Trigger
H_Message Data
F_Message Data
Su_Message Data
St_Message Data
Message Interface 2.2.1
Message
Code Data
Code line Analyzer 2.3.1
Code line Data
Catcher Kind Collector 2.4.1
Divided Data
While Catcher 2.4.2
Do-while Catcher 2.4.3
for Catcher 2.4.4
If Catcher 2.4.5
Switch Catcher 2.4.6
Return Catcher 2.4.7
Function Catcher 2.4.8
While info
Do-while info
For info
If info
Switch info
Return info
Function info
Node & Edge

# DFD Level 3 – Data Dictionary

| Input/Output | Description | Type |
|---|---|---|
| H_Message Data | Pass the Help Message data | Char* |
| F_Message Data | Pass the Fail Message data | Char* |
| Su_Message Data | Pass the Success Message Data | Char* |
| St_Message Data | Pass the Start Message Data | Char* |
| Node & Edge | Stored the Node & Edge | Char* |

# DFD Level 4

## State Transition Diagram for Controller 2.1.1

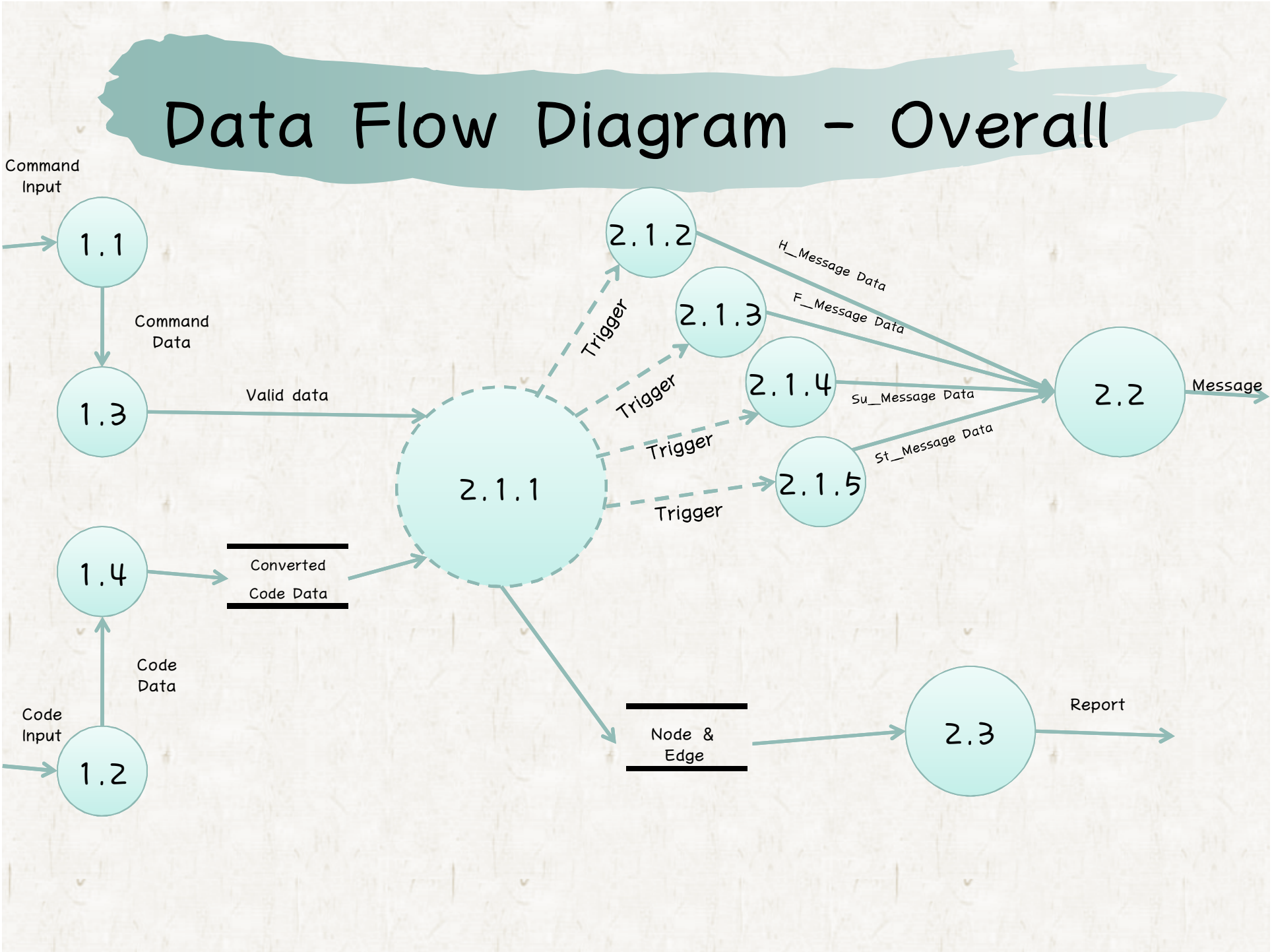



R = C Code Input success

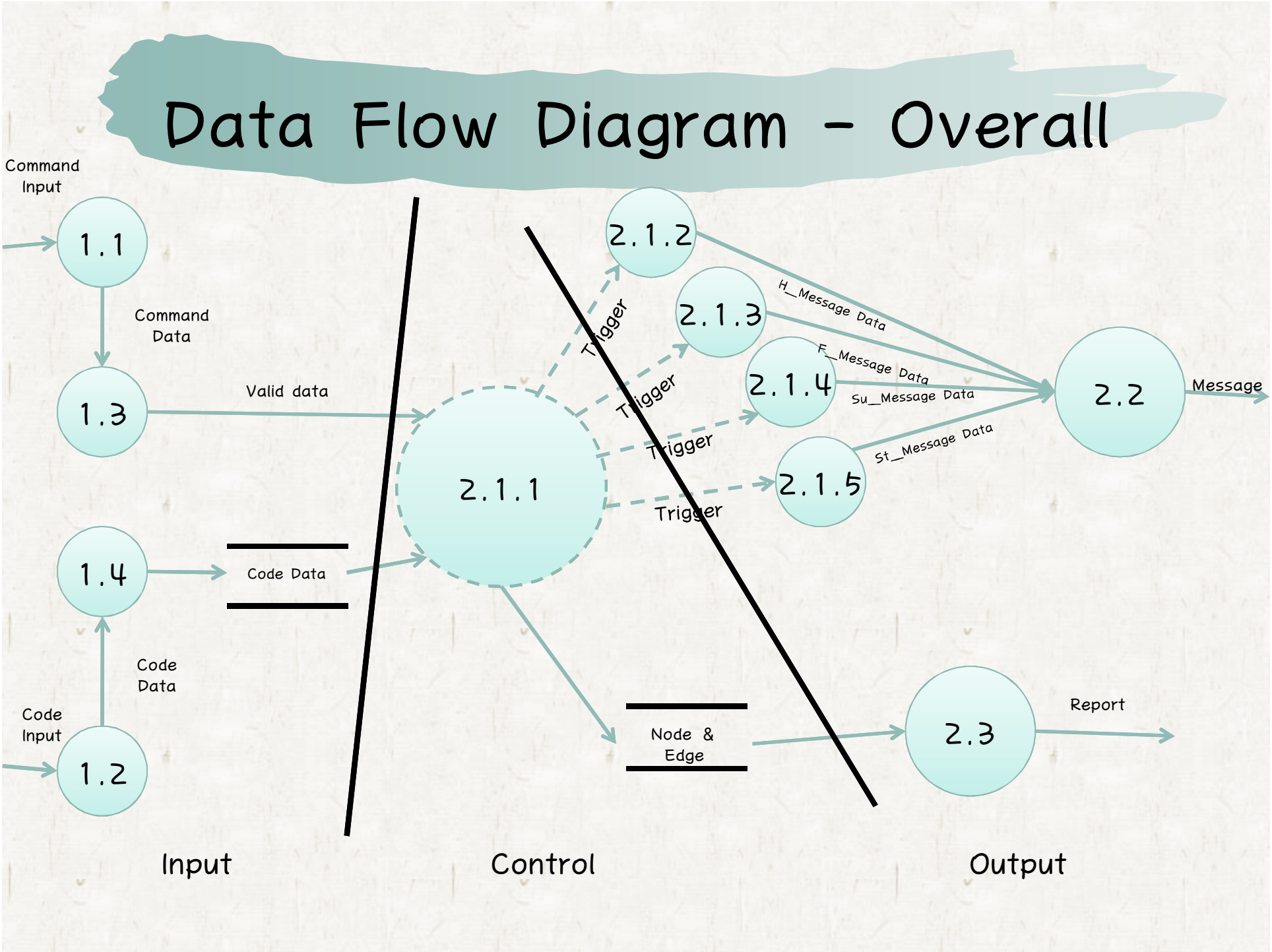
cmd_E = Judge whether user input correct or wrong command

A = Finish the analysis

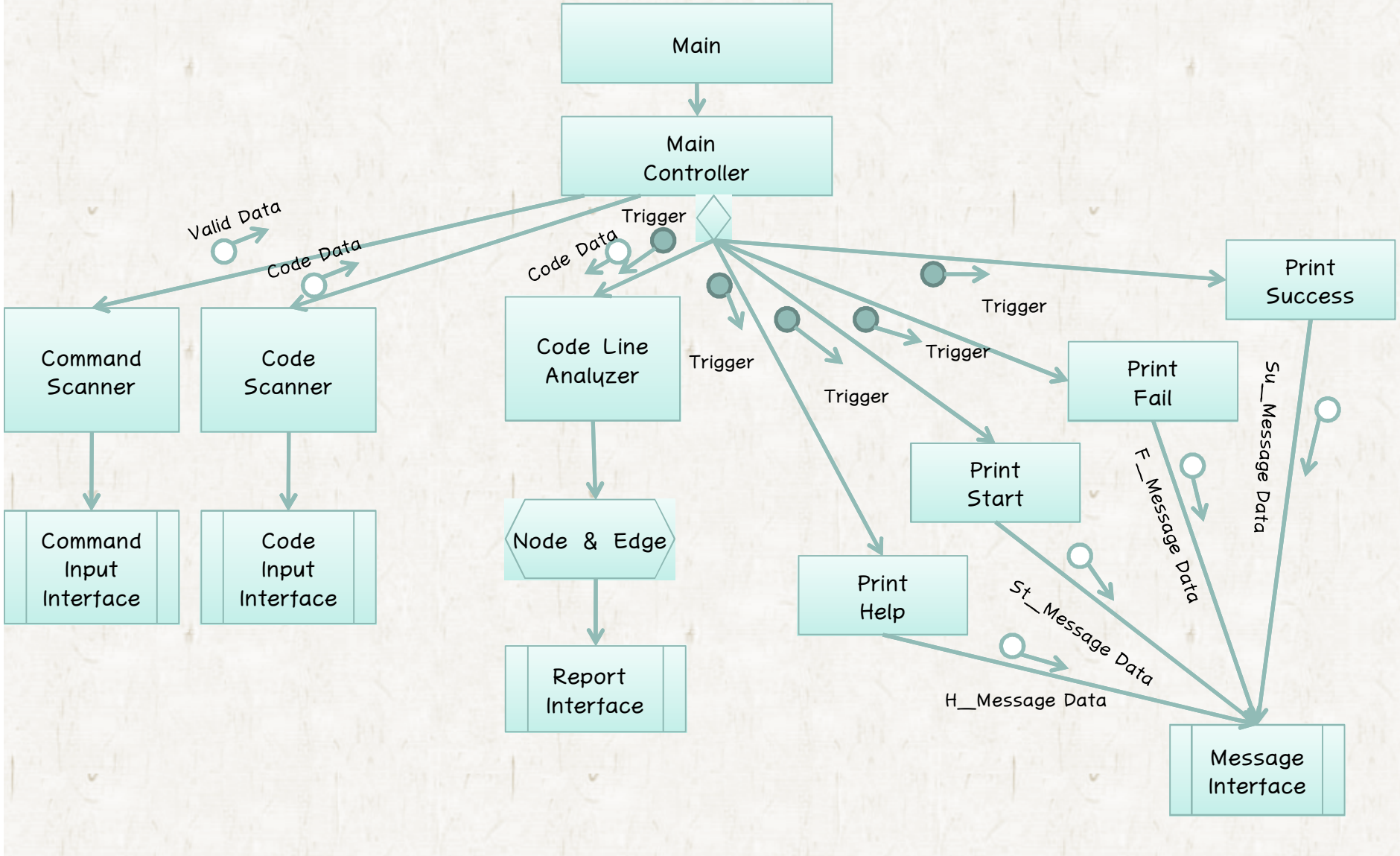
# Data Flow Diagram - Overall



# Structured Design


Data Flow Diagram - Overall
Command Input
1.1
Command Data
1.3
Valid data
1.4
Code Data
Code Data
Code Input
1.2
2.1.1
2.1.2
2.1.3
2.1.4
2.1.5
Trigger
Trigger
Trigger
Trigger
H_Message Data
F_Message Data
Su_Message Data
St_Message Data
2.2
Message
Node & Edge
2.3
Report
Input
Control
Output

# Structured Charts



# Code Implementation & Explanation

# Code Implementation & Explanation

```
void CommandInterface(char *s[],char *buf)
{
        if(s[2]==NULL)
                strcpy(buf,"ERROR");
        else
                strcpy(buf,s[2]);
}
```

| Reference No. | 1.1 |
|---|---|
| Name | Command Input Interface |
| Input | Command Input |
| Output | Command Data |
| Process Description | Receive a Command Input of the Command Scanner, and converts it to Command Data |

```
FILE* CodeInterface(FILE *p,char *arg)
{
        if(strstr(arg,".c"))
        {
                p=fopen(arg,"r");
                return p;
        }
        else
        {
                return p;
        }
}
```

| Reference No. | 1.2 |
|---|---|
| Name | Code Input Interface |
| Input | Code Input |
| Output | Code Data |
| Process Description | Receive a .c File and open the File and pass the FILE pointer |

# Code Implementation & Explanation

```
int CommandScanner(char *buf,int fCheck)
{
        int i=0,j;
        int valid;
        char *ch;
        ch=(char *)malloc(sizeof(char)*4);
        while(buf[i]!='.'){
                i++;
        }
        ch[0]=buf[i+1];
        ch[1]=buf[i+2];
        ch[2]=buf[i+3];
        ch[3]='\0';
        for(j = 0; j < 4; j++)
                ch[j] = toupper(ch[j]);
        valid=buf[i+4];
        if(!strcmp(ch, "TXT") && fCheck==0 && valid==0)
                return 1;
        else if(fCheck==1)
        {
                printf("2\n");
                return 2;
        }
        else
                return 3;
}
```

| Reference No. | 1.3 |
|---|---|
| Name | Command Scanner |
| Input | Command Data |
| Output | Valid Data |
| Process Description | Receive the Command Data and converts it to Integer data that Main Controller distinguishs it (1=Correct Command , 2=To fail FILE Input , 3= Send the Help message) |

# Code Implementation & Explanation

```
int CodeScanner(FILE *fp,Node *n)
{
        int k,j;
        int number;
        char *str="main(";
        if(fp==NULL)
        {
                return 1;
        }
        while(!feof(fp))
        {
                fgets(n[idx].s, sizeof(n[idx].s), fp);
                n[idx].LineNum=idx+1;
                idx++;
        }
        do{
                for(j=0;j<idx-1;j++)
                        n[j]=n[j+1];
                idx--;
        }while(!strstr(n[0].s,str));
        for(j=0;j<idx-1;j++)
                n[j]=n[j+1];
        idx--;
        return 0;
}
```

| Reference No. | 1.4 |
|---|---|
| Name | Code Scanner |
| Input | Code Data |
| Output | Converted Code Data |
| Process Description | Reads a Code data line by line, and converts it to new Code Data |

# Code Implementation & Explanation

```
void controller(char **argv)
{
        FILE *fp=NULL;
        FILE *ofp=NULL;
        Node n[200];
        Edge e[200];
        int checker=0;
        int validData,i;
        char buf[100]={0,};


        fp=CodeInterface(fp,argv[1]);
        checker=CodeScanner(fp,n);
        CommandInterface(argv,buf);
        validData=CommandScanner(buf,checker);


        if(validData==1)
        {
                printSuccess();
                printStart();

                CodeLineAnalyzer(n,e,0);
        }
        else if(validData==2)
        {
                printFail();

                exit(0);
        }
        else
        {
                printHelp();
                exit(0);
        }
        for(i=0; i<idx; i++)
        {
                strtok(n[i].s,"\n");
        }
        Report(n,e,ofp,argv[2]);


        fclose(fp);
}
```

| Reference No. | 2.1.1 |
|---|---|
| Name | Main Controller |
| Input | Valid Data , Converted code Data |
| Output | Node & Edge, trigger |
| Process Description | It is a main controller that determines CFG Generator After input Valid Data and Converted Code Data, delivery Node & Edge and trigger |

# Code Implementation & Explanation

```
void CodeLineAnalyzer(Node *n,Edge *e,int start)
{
        char temp1[100] = {0,};
        char temp2[100] = {0,};
        char temp3[100] = {0,};
        char ifnode[100][100];
        int i=0,l=0,z;
        int number,returnLine;
        printf("\n");
        for(i=start;i<idx;i++)
        {
                if(strstr(n[i].s,"if("))
                {
                        z=0;
                        strcpy(e[edgeidx].s,"[##if]");
                        sprintf(temp1,"%d",n[i].LineNum);
                        i++;
                        edgeidx++;
                        while(!(strstr(n[i].s,"else") && !strstr(n[i].s,"else if"))){
                                if(strstr(n[i].s,"else if(")){
                                        sprintf(temp2,"%d",n[i].LineNum);
                                        strcat(temp1,"->");
                                        strcat(temp1,temp2);
                                        strcpy(e[edgeidx].s,temp1);
                                        sprintf(temp1,"%d",n[i].LineNum);
                                        edgeidx++;
                                }
                                else{
                                        if(strstr(n[i].s,"}")){
                                                sprintf(ifnode[z],"%d",n[i-1].LineNum);
                                                z++;
                                        }
                                        else{
                                                strcpy(temp2,temp1);
                                                sprintf(temp3,"%d",n[i].LineNum);
                                                strcat(temp2,"->");
                                                strcat(temp2,temp3);
                                                strcpy(e[edgeidx].s,temp2);
                                                edgeidx++;
                                        }
                                }
                                i++;
                        }
                        while(!strstr(n[i].s,"}")){
                                if(strstr(n[i].s,"else")){
                                        strcpy(temp2,temp1);
                                        sprintf(temp3,"%d",n[i].LineNum);
                                        strcat(temp2,"->");
                                        strcat(temp2,temp3);
                                        strcpy(e[edgeidx].s,temp2);
                                        edgeidx++;
                                        i++;
                                }
                                else{
                                        sprintf(temp1,"%d",n[i-1].LineNum);
                                        sprintf(temp2,"%d",n[i].LineNum);
                                        strcat(temp1,"->");
                                        strcat(temp1,temp2);
                                        strcpy(e[edgeidx].s,temp1);
                                        edgeidx++;
                                        i++;
                                }
                        }
                        sprintf(temp1,"%d",n[i-1].LineNum);
                        sprintf(temp2,"%d",n[i+1].LineNum);
                        strcat(temp1,"->");
                        strcat(temp1,temp2);
                        strcpy(e[edgeidx].s,temp1);
                        edgeidx++;
                        for(l=0;l<z;l++)
                        {
                                strcpy(temp1,ifnode[l]);
                                strcat(temp1,"->");
                                strcat(temp1,temp2);
                                strcpy(e[edgeidx].s,temp1);
                                edgeidx++;
                        }
                        strcpy(e[edgeidx].s,"[##Endif]");
                        edgeidx++;
                }
```

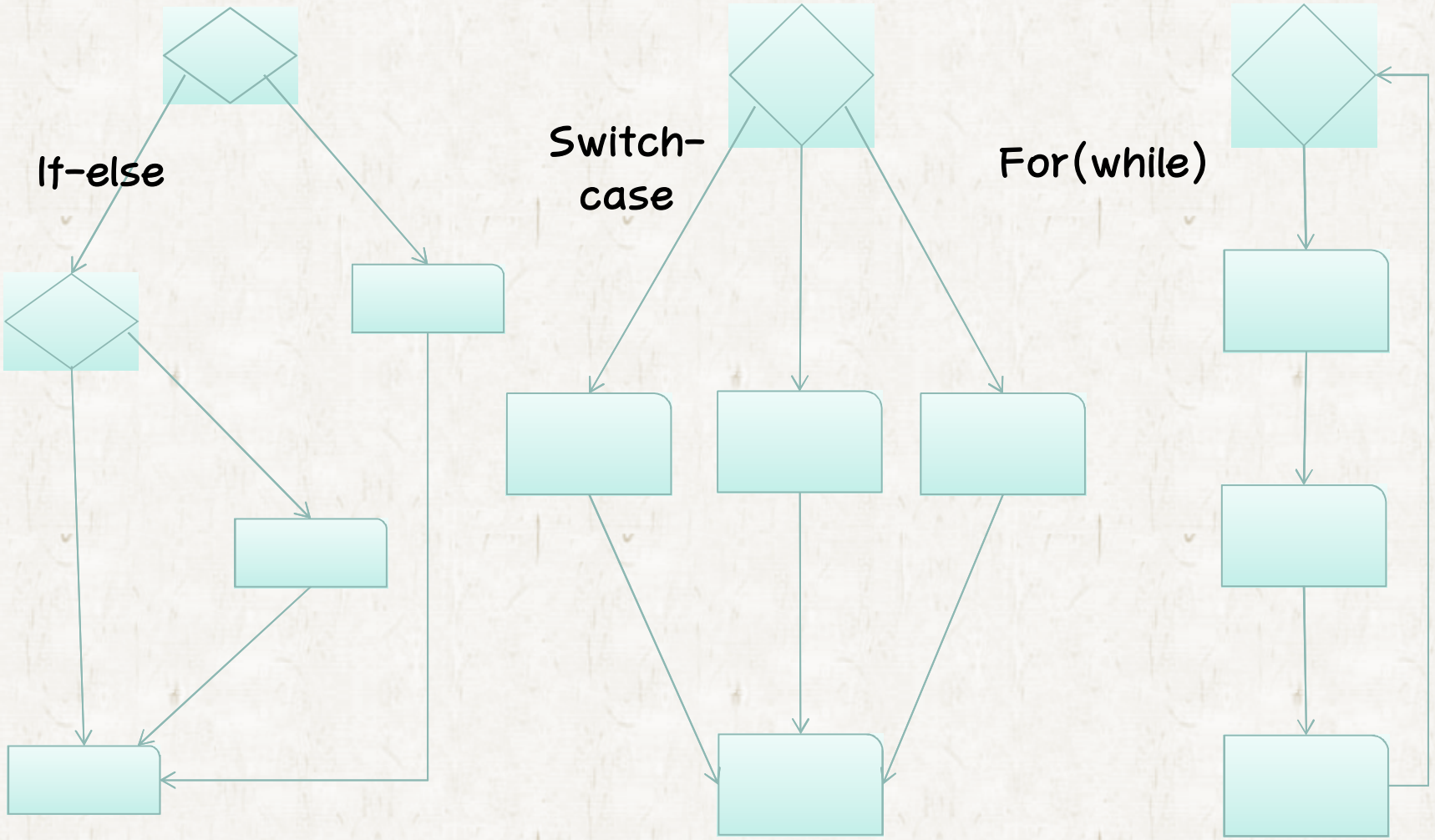
```
                else if(strstr(n[i].s,"for("))
                {
                        strcpy(e[edgeidx].s,"[##for]");
                        sprintf(temp1,"%d",n[i].LineNum);
                        i++;
                        edgeidx++;
                        while(!strstr(n[i].s,"}")){
                                sprintf(temp2,"%d",n[i-1].LineNum);
                                sprintf(temp3,"%d",n[i].LineNum);
                                strcat(temp2,"->");
                                strcat(temp2,temp3);
                                strcpy(e[edgeidx].s,temp2);
                                edgeidx++;
                                i++;
                        }
                        strcpy(temp2,temp1);
                        sprintf(temp3,"%d",n[i-1].LineNum);
                        strcat(temp3,"->back->");
                        strcat(temp3,temp2);
                        strcpy(e[edgeidx].s,temp3);
                        edgeidx++;
                        sprintf(temp1,"%d",n[i-1].LineNum);
                        sprintf(temp2,"%d",n[i+1].LineNum);
                        strcat(temp1,"->");
                        strcat(temp1,temp2);
                        strcpy(e[edgeidx].s,temp1);
                        edgeidx++;
                        strcpy(e[edgeidx].s,"[##Endfor]");
                        edgeidx++;
                }
                else if(strstr(n[i].s,"while(") && !strstr(n[i].s,"}while"))
                {
                        strcpy(e[edgeidx].s,"[##while]");
                        sprintf(temp1,"%d",n[i].LineNum);
                        i++;
                        edgeidx++;
                        while(!strstr(n[i].s,"}")){
                                sprintf(temp2,"%d",n[i-1].LineNum);
                                sprintf(temp3,"%d",n[i].LineNum);
                                strcat(temp2,"->");
                                strcat(temp2,temp3);
                                strcpy(e[edgeidx].s,temp2);
                                edgeidx++;
                                i++;
                        }
                        sprintf(temp3,"%d",n[i-1].LineNum);
                        strcat(temp3,"->back->");
                        strcat(temp3,temp2);
                        strcpy(e[edgeidx].s,temp3);
                        edgeidx++;
                        sprintf(temp1,"%d",n[i-1].LineNum);
                        sprintf(temp2,"%d",n[i+1].LineNum);
                        strcat(temp1,"->");
                        strcat(temp1,temp2);
                        strcpy(e[edgeidx].s,temp1);
                        edgeidx++;
                        strcpy(e[edgeidx].s,"[##Endwhile]");
                        edgeidx++;
        }
        else if(strstr(n[i].s,"switch("))
        {
                strcpy(e[edgeidx].s,"[##switch]");
                z=1;
                sprintf(temp1,"%d",n[i].LineNum);
                edgeidx++;
                i++;
                while(!strstr(n[i].s,"}")){
                        if(strstr(n[i].s,"case") || strstr(n[i].s,"default")){
                                strcpy(temp2,temp1);
                                sprintf(temp3,"%d",n[i+1].LineNum);
                                strcat(temp2,"->");
                                strcat(temp2,temp3);
                                strcpy(e[edgeidx].s,temp2);
                                edgeidx++;
                        }
                        else
                        {
                                if(strstr(n[i].s,"break;")){
                                        sprintf(ifnode[z],"%d",n[i-1].LineNum);
                                        z++;
                                }
                        }
                        i++;
                }
                for(l=0; l<z; l++)
                {
                        strcpy(temp2,ifnode[l]);
                        strcat(temp2,"->");
                        sprintf(temp3,"%d",n[i+1].LineNum);
                        strcat(temp2,temp3);
                        strcpy(e[edgeidx].s,temp2);
```

# Code Implementation & Explanation

```
                edgeidx++;
        }
        strcpy(e[edgeidx].s,"[##Endswitch]");
        edgeidx++;
}
else if(strstr(n[i].s,"do{"))
{
        strcpy(e[edgeidx].s,"[##do-while]");
        edgeidx++;
        sprintf(temp3,"%d",n[i].LineNum);
        strcpy(temp1,temp3);
        sprintf(temp2,"%d",n[i+1].LineNum);
        strcat(temp1,"=>");
        strcat(temp1,temp2);
        strcpy(e[edgeidx].s,temp1);
        i++;
        edgeidx++;
        while(!strstr(n[i].s,"}while"))
        {
                sprintf(temp1,"%d",n[i].LineNum);
                sprintf(temp2,"%d",n[i+1].LineNum);
                strcat(temp1,"=>");
                strcat(temp1,temp2);
                strcpy(e[edgeidx].s,temp1);
                edgeidx++;
                i++;
        }
                sprintf(temp1,"%d",n[i].LineNum);
                strcpy(temp2,temp1);
                strcat(temp2,"=back=>");
                strcat(temp2,temp3);
                strcpy(e[edgeidx].s,temp2);
                edgeidx++;
                sprintf(temp3,"%d",n[i+1].LineNum);
                strcpy(temp2,temp1);
                strcat(temp2,"=>");
                strcat(temp2,temp3);
                strcpy(e[edgeidx].s,temp2);
                edgeidx++;
                strcpy(e[edgeidx].s,"[##Enddo-while]");
                edgeidx++;
}
```

```
        else
        {
                if(strstr(n[i].s,"return"))
                {
                        sprintf(temp1,"%d",n[i].LineNum);
                        strcat(temp1,"=>");
                        strcat(temp1,"[End]");
                        strcpy(e[edgeidx].s,temp1);
                        edgeidx++;
                        returnLine=i;
                        i=100;
                }
                else
                {
                        sprintf(temp1,"%d",n[i].LineNum);
                        sprintf(temp2,"%d",n[i+1].LineNum);
                        strcat(temp1,"=>");
                        strcat(temp1,temp2);
                        strcpy(e[edgeidx].s,temp1);
                        edgeidx++;
                }
        }
}
if(func==1)
{
        strcpy(e[edgeidx].s,"[##Endfunction]");
        edgeidx++;
}
for(returnLine;returnLine<idx;returnLine++)
{
        if(strstr(n[returnLine].s,"{"))
        {
                func=1;
                strcpy(e[edgeidx].s,"[##function]");
                edgeidx++;
                CodeLineAnalyzer(n,e,returnLine);
        }
}
for(i=0;i<idx;i++){
        if((strstr(n[i].s,"}") && !strstr(n[i].s,"}while")) || strstr(n[i].s,"break;") || strstr(n[i].s,"case") || strstr(n[i].s,"default"))
        {
                for(l=i;l<(idx-1);l++)
                        n[l]=n[l+1];
                idx--;
                i--;
        }
```

# If else, switch case, for(while)문



# Code Implementation & Explanation

```
void printHelp(void)
{
        char *str="Wrong Command!!!Please Input Command Again!!";
        MessageInterface(str);
}
```

| Reference No. | 2.1.2 |
|---|---|
| Name | Print Help |
| Input | Trigger |
| Output | H_Message Data |
| Process Description | Accept generated Trigger from Main Controller and send H_Message Data. |

```
void printFail(void)
{
        char *str="Fail the Input C code!!";
        MessageInterface(str);
}
```

| Reference No. | 2.1.3 |
|---|---|
| Name | Print Fail |
| Input | Trigger |
| Output | F_Message Data |
| Process Description | Accept generated Trigger from Main Controller and send F_Message Data. |

# Code Implementation & Explanation

```
void printSuccess(void)
{
        char *str="Success the Input C code!!";
        MessageInterface(str);
}
```

| Reference No. | 2.1.4 |
|---|---|
| Name | Print Success |
| Input | Trigger |
| Output | Su_Message Data |
| Process Description | Accept generated Trigger from Main Controller and send Su_Message Data. |

```
void printStart(void)
{
        char *str="Start CFG generator!!";
        MessageInterface(str);
}
```

| Reference No. | 2.1.5 |
|---|---|
| Name | Print Start |
| Input | Trigger |
| Output | St_Message Data |
| Process Description | Accept generated Trigger from Main Controller and send St_Message Data. |

# Code Implementation & Explanation

```
void MessageInterface(char *str)
{
        printf("%s\n",str);
}
```

| Reference No. | 2.2 |
|---|---|
| Name | Message Interface |
| Input | Message Data |
| Output | Message |
| Process Description | Convert input Message Data to Message that will be printed. |

# Code Implementation & Explanation

```
void Report(Node *n,Edge *e,FILE *ofp,char *arg)
{
        int i;
        ofp=fopen(arg,"w");
        for(i=0;i<idx;i++){
                printf("=============================================\n");
                printf("node[%d]  = %s",i,n[i].s);
                printf("node[%d]-Line = %d\n",i,n[i].LineNum);
                printf("=============================================\n");
        }
        for(i=0;  i<edgeidx;  i++){
                printf("edge[%d]  = %s\n",i,e[i].s);
        }
        for(i=0;i<idx;i++)
        {
                fputs("node",ofp);
                fprintf(ofp,"%d      ",(i+1));
                fputs("LineNum",ofp);
                fprintf(ofp,"%d      ",n[i].LineNum);
                fputs(n[i].s,ofp);
                fputs("\r\n",ofp);
                fputs("=====================================\r\n",ofp);
        }
        for(i=0;i<edgeidx;i++)
        {
                fputs("edge",ofp);
                fprintf(ofp,"%d      ",(i+1));
                fputs(e[i].s,ofp);
                fputs("\r\n",ofp);
        }

}
```

| Reference No. | 2.3 |
|---|---|
| Name | Report interface |
| Input | Node & edge data |
| Output | Report |
| Process Description | Take node & edge data and converts it to string type and saves the string into a text file and then prints it to the screen |

Thank
you..