Final Presentation

CFG Generator with SASD

Speaker : 200811425 Pyung Soek Kim
200811435 Sung Ho Shin
200811451 Hyung Yeol Lee
200811454 In Seo Jeon

Contents

Change in SASD

Code from the SASD

Demonstration

Changes in SASD

Statement of Purpose

Not Changed !!!

System Context Diagram

& Event List

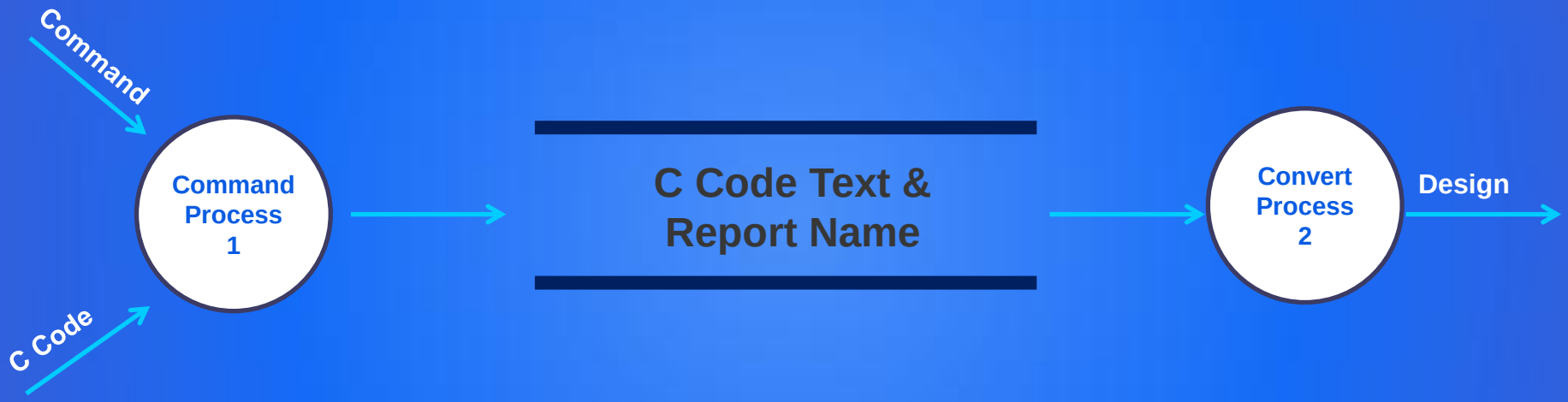
Input / Output Event	Description
Command	CUI Command 를 입력 받는다.
Design	Convert 된 Design 을 넘겨 준다.

Input/Output Event	Description	Format/Type
Command	String that includes Input File Name and Output File Name Ex) ./CG [input file name] [output file name]	char*
C Code	C code that will convert CFG C code must have *.c file extension C code Generated by C standard	*.c
Design	Data that saves information for drawing CFG and a information to create report file struct Design{ int nodesize; int edgesize; Node* node[DESIGN_SIZE]; Edge* edge[DESIGN_SIZE*2];}	struct

Data Flow Diagram

DFD Level 1

Changes in SASD



Char* Report Name - OK
[.txt] Text File ..???

Changes in SASD

Char* Report Name - OK
File* File Pointer !

DFD Level 2

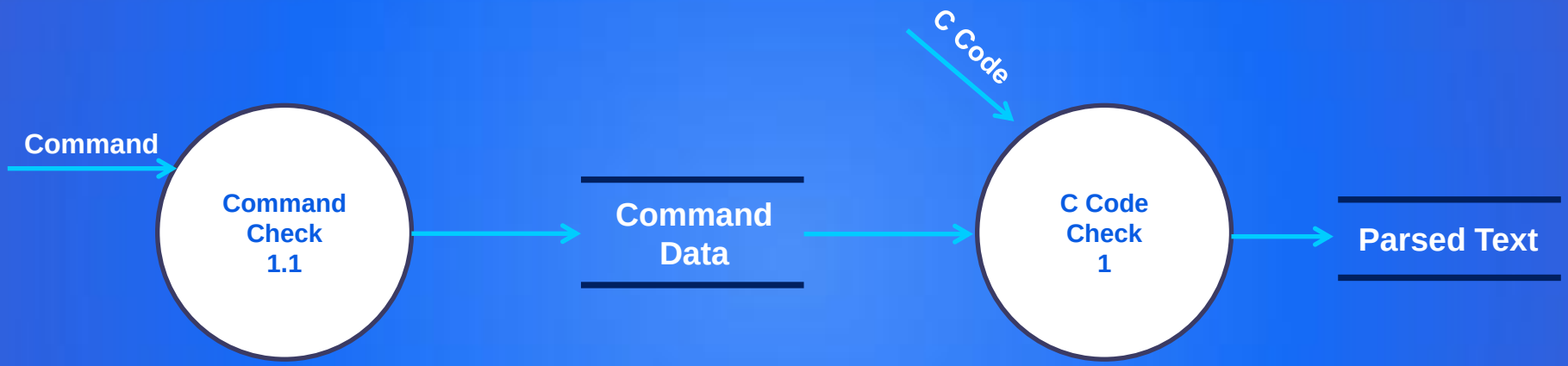
Changes in SASD

[Command Check] don't use [C Code]

One data flow with two data

Changes in SASD

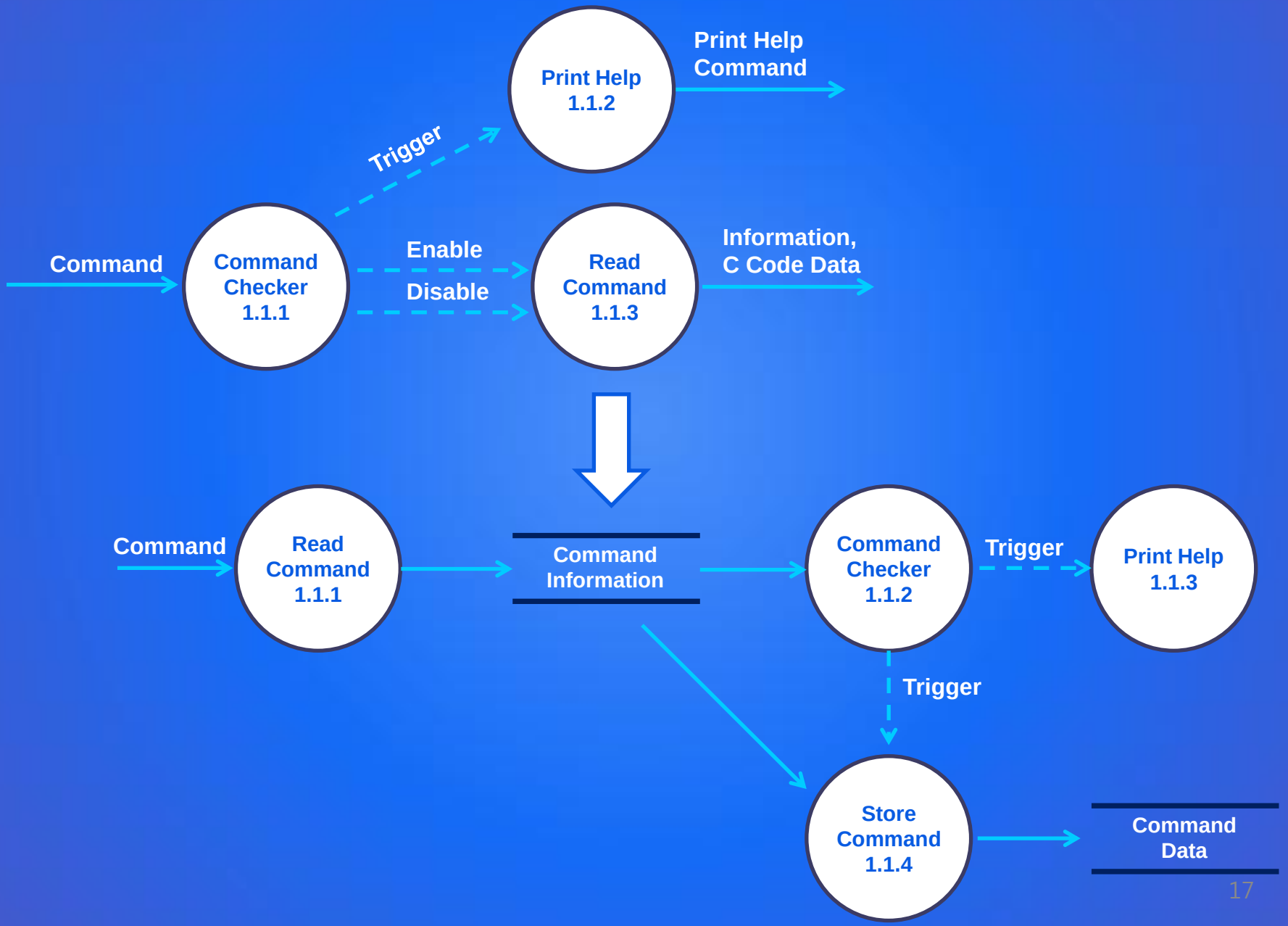
[C Code Check] use [C Code]



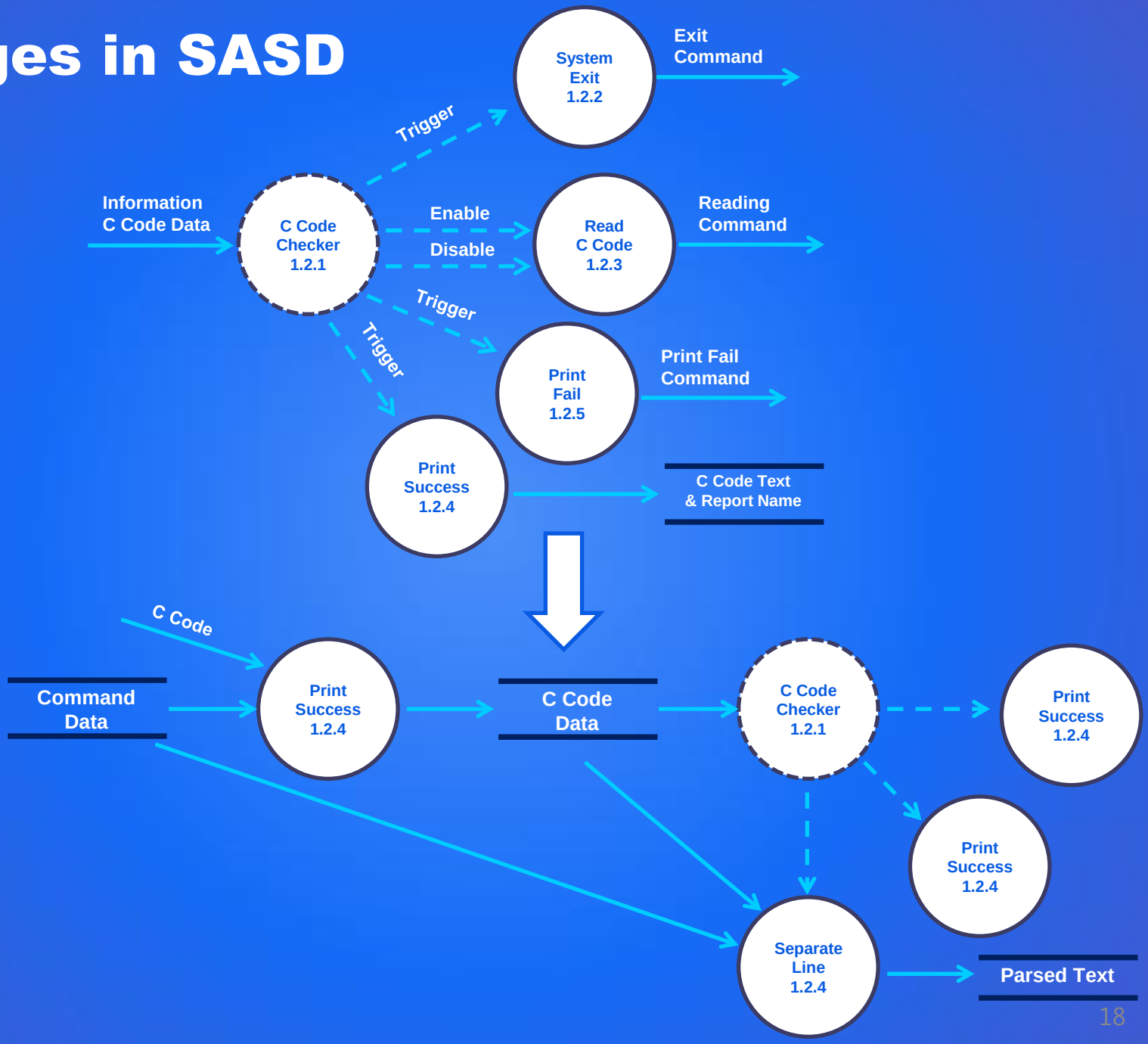
Make Data store

DFD Level 3

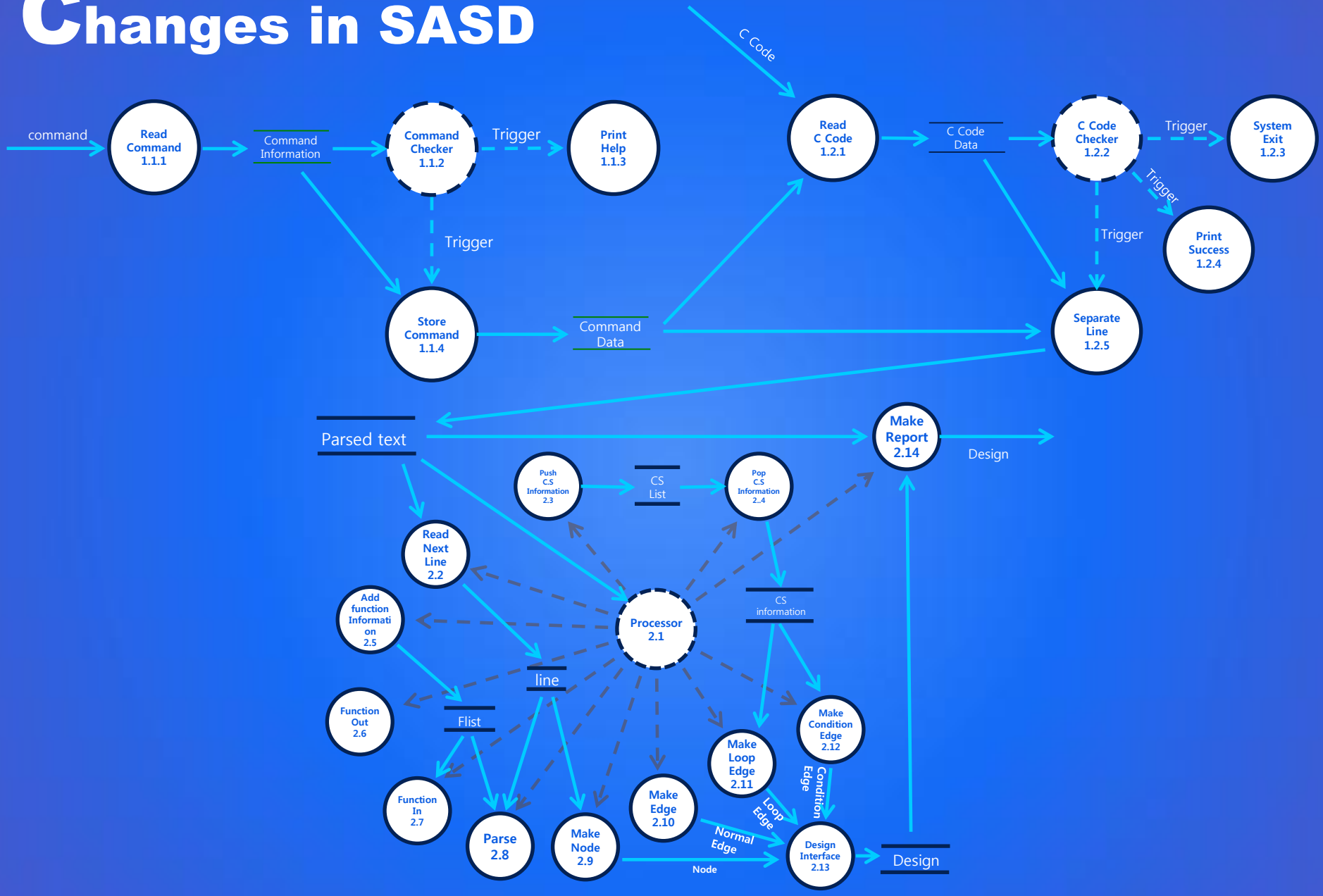
Changes in SASD



Changes in SASD

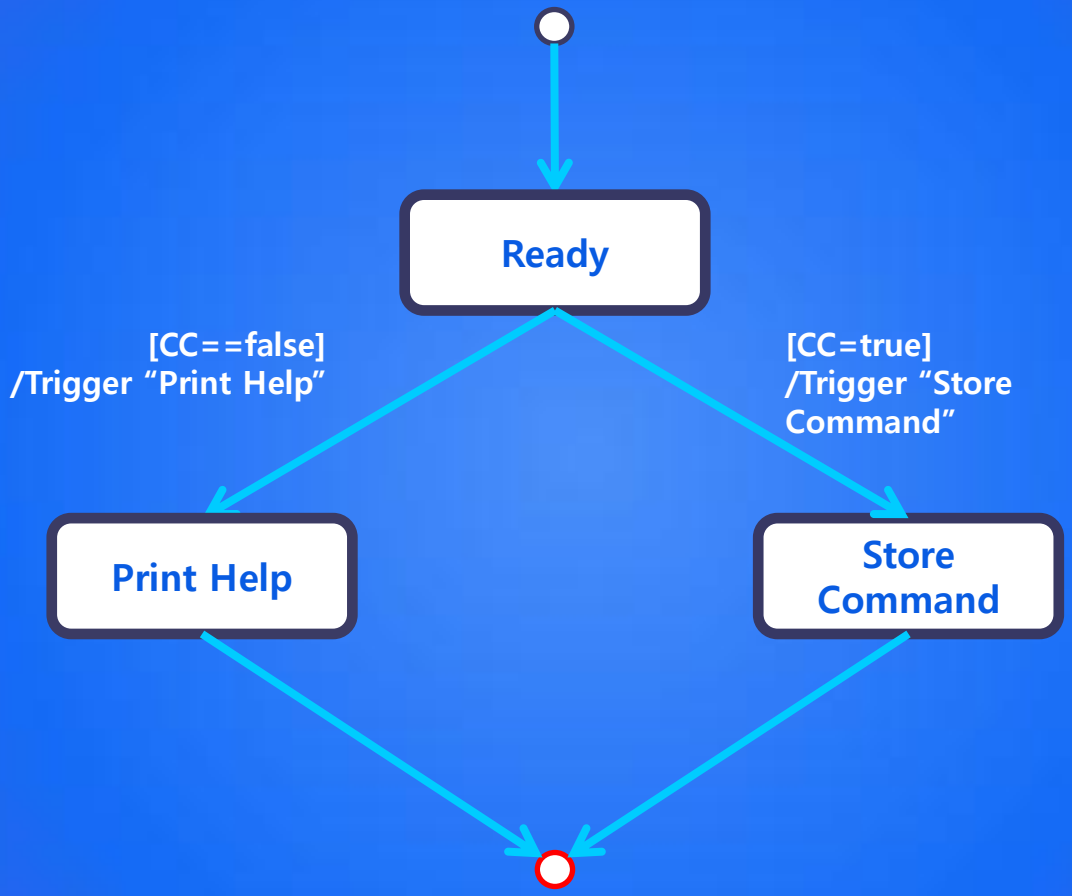


Changes in SASD

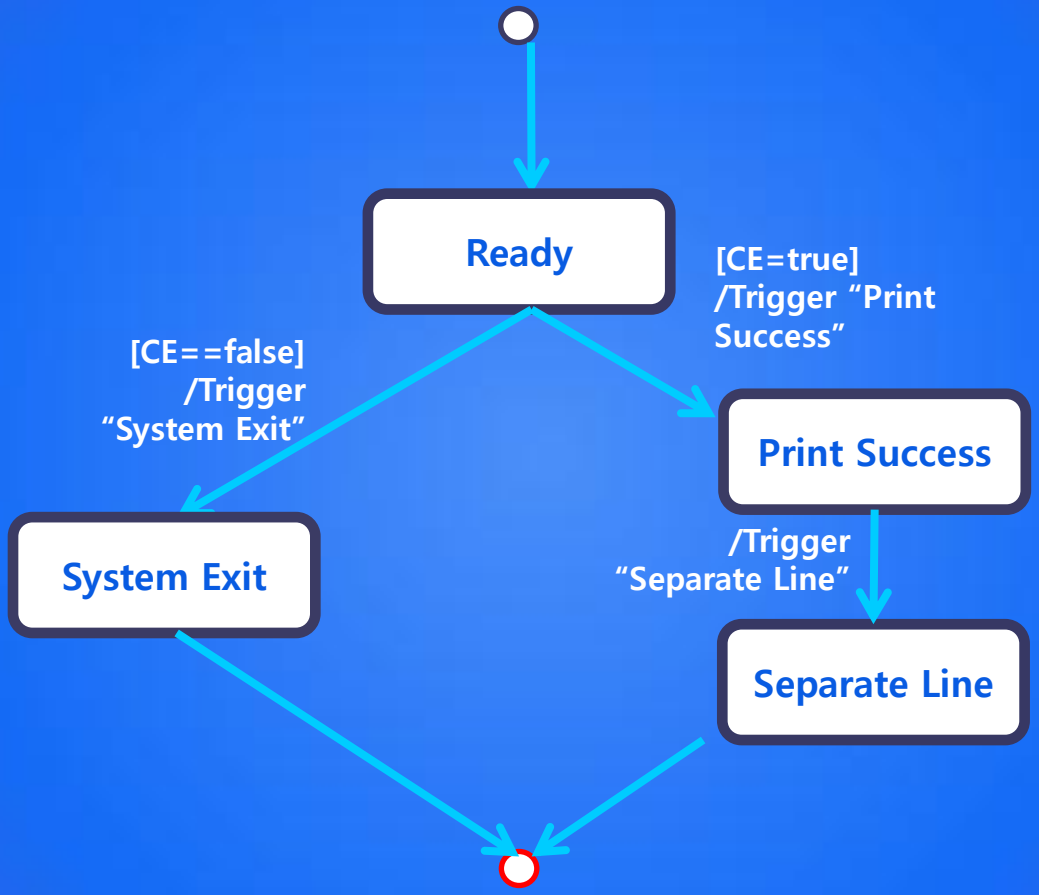


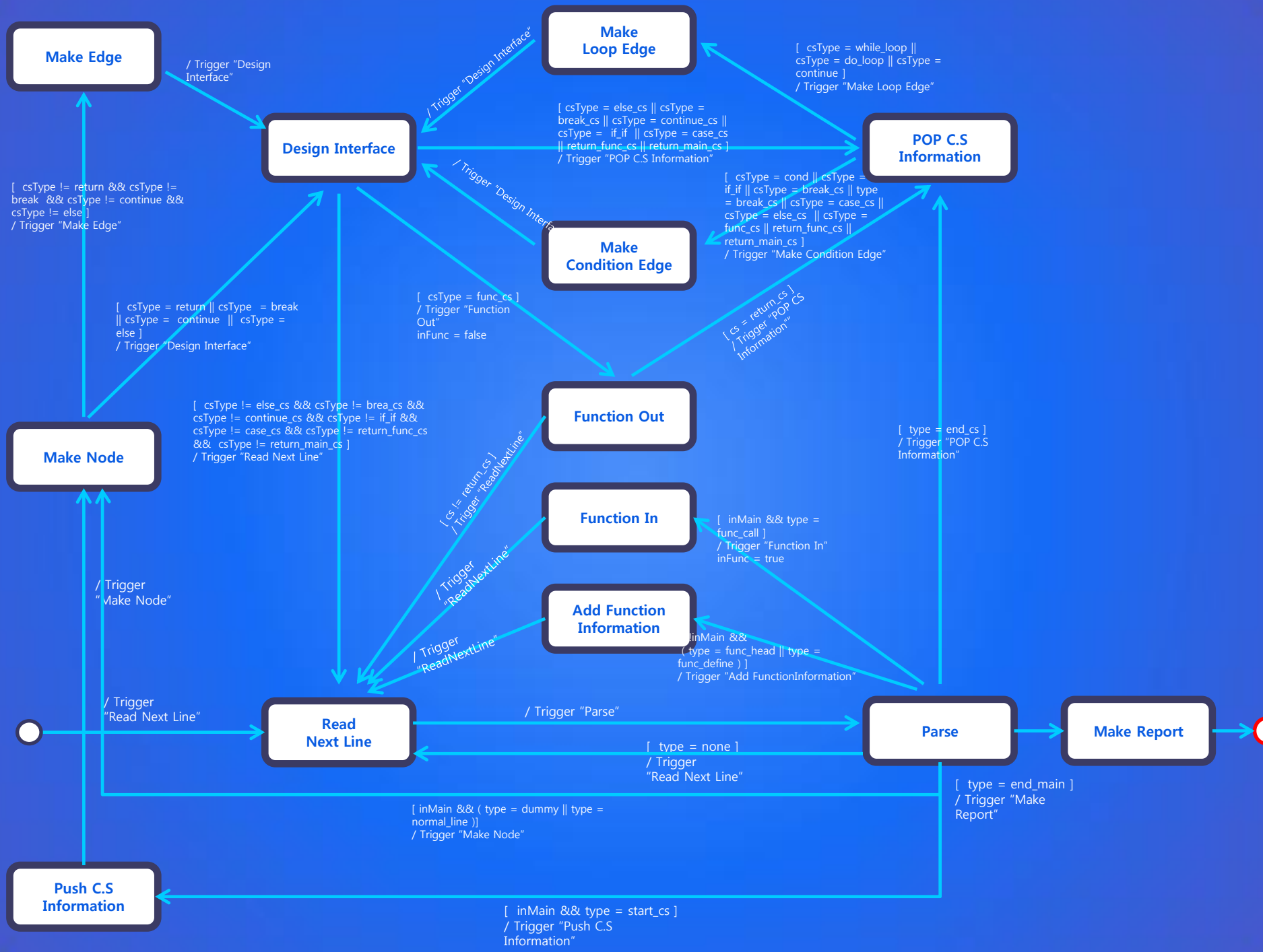
Finite State Machine

Changes in SASD



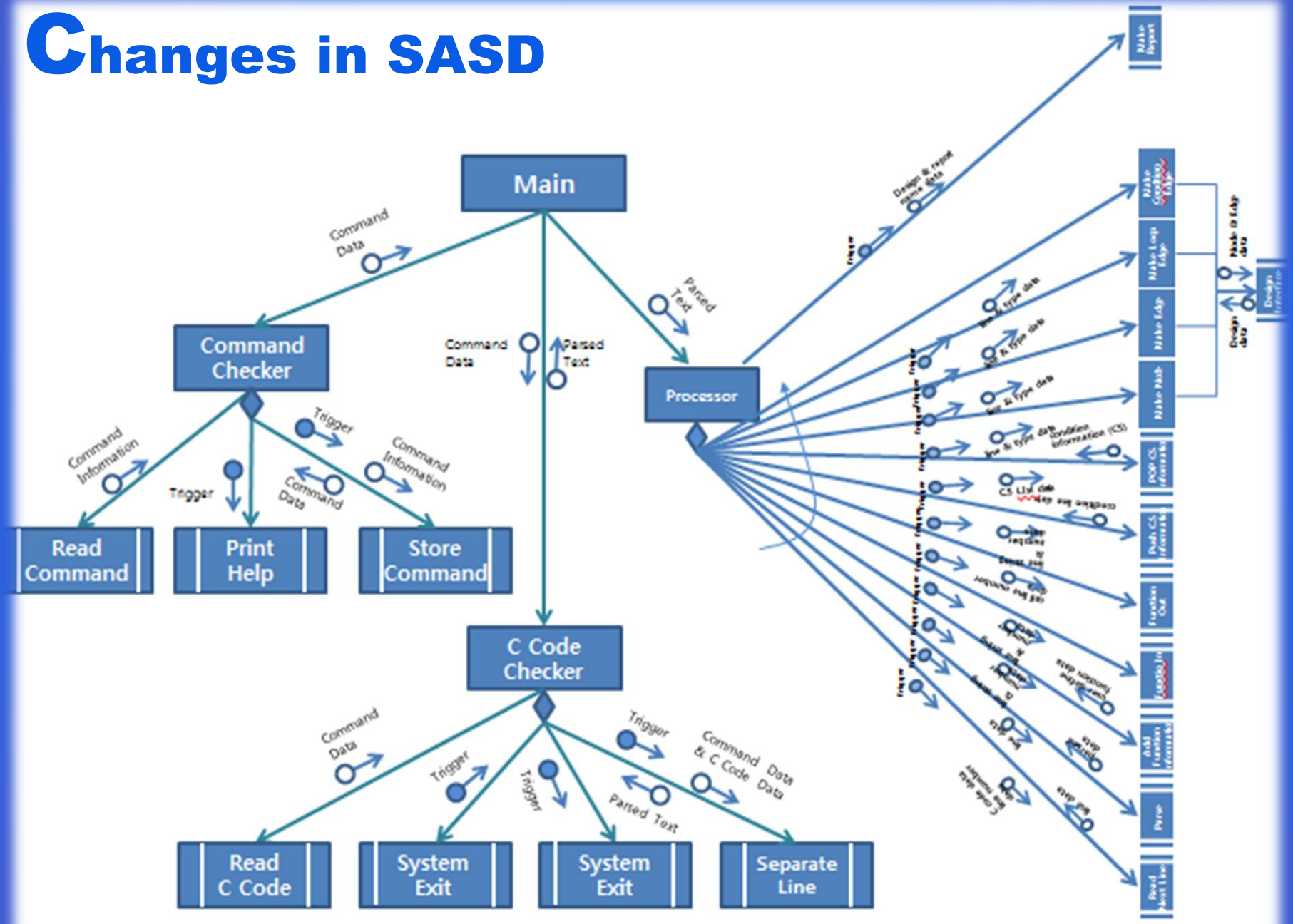
Changes in SASD




Make Edge
/ Trigger "Design Interface"
Make Loop Edge
/ Trigger "Design Interface"
[csType = while_loop || csType = do_loop || csType = continue]
/ Trigger "Make Loop Edge"
Design Interface
[csType = else_cs || csType = break_cs || csType = continue_cs || csType = if_if || csType = case_cs || return_func_cs || return_main_cs]
/ Trigger "POP C.S Information"
POP C.S Information
[csType != return && csType != break && csType != continue && csType != else]
/ Trigger "Make Edge"
/ Trigger "Design Interface"
Make Condition Edge
[csType = cond || csType = if_if || csType = break_cs || type = break_cs || csType = case_cs || csType = else_cs || csType = func_cs || return_func_cs || return_main_cs]
/ Trigger "Make Condition Edge"
[csType = return || csType = break || csType = continue || csType = else]
/ Trigger "Design Interface"
[csType = func_cs]
/ Trigger "Function Out"
inFunc = false
[cs = return_cs]
/ Trigger "POP CS Information""
Function Out
[csType != else_cs && csType != brea_cs && csType != continue_cs && csType != if_if && csType != case_cs && csType != return_func_cs && csType != return_main_cs]
/ Trigger "Read Next Line"
Make Node
[type = end_cs]
/ Trigger "POP C.S Information"
[cs != return_cs]
/ Trigger "ReadNextLine"
Function In
[inMain && type = func_call]
/ Trigger "Function In"
inFunc = true
/ Trigger "Make Node"
/ Trigger "ReadNextLine"
Add Function Information
/ Trigger "ReadNextLine"
[inMain && (type = func_head || type = func_define)]
/ Trigger "Add FunctionInformation"
/ Trigger "Read Next Line"
Read Next Line
/ Trigger "Parse"
Parse
Make Report
[type = none]
/ Trigger "Read Next Line"
[type = end_main]
/ Trigger "Make Report"
[inMain && (type = dummy || type = normal_line)]
/ Trigger "Make Node"
Push C.S Information
[inMain && type = start_cs]
/ Trigger "Push C.S Information"

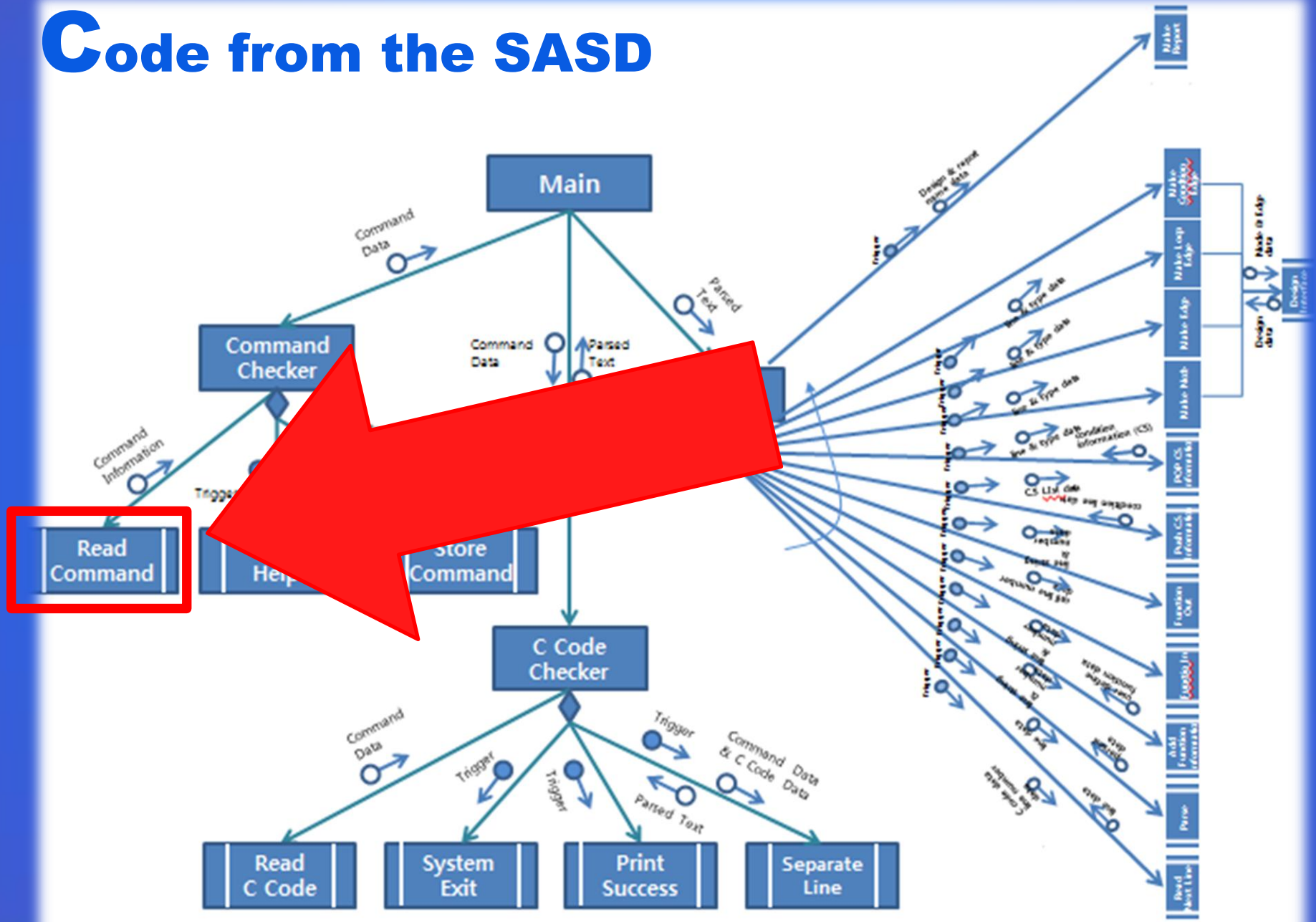
Structured Chart

Changes in SASD



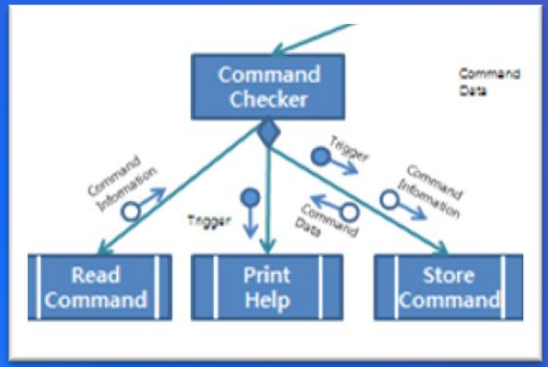
Code form the SASD

Code from the SASD



Code from the SASD

Read Command

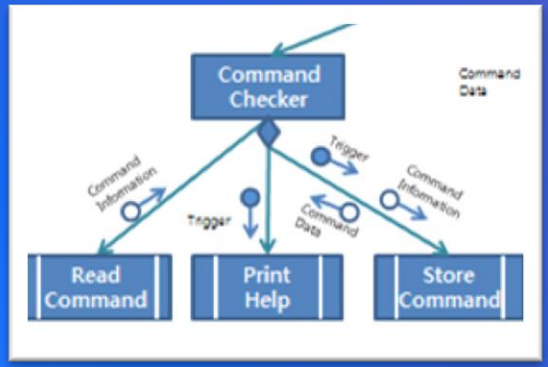



```
int main(int argc, char **argv)
{
    CommandInfo* cmdInfo;
    CommandData* cmdData;

    CCodeData* ccData;
    ParsedText* parsedData;


    cmdInfo = ReadCommand(argc,(char*)argv[1],(char*)argv[2]);

    cmdData = CommandChecker(cmdInfo);

    ccData = ReadCCode(cmdData);


    parsedData = CCodeChecker(cmdData,ccData);

    return 0;
}
```

Input Data is entering ReadCommand function in main

Code from the SASD

Read Command




```
#ifdef _DEBUG
    puts("#ReadCommand called");
#endif
```

If argument count is 3

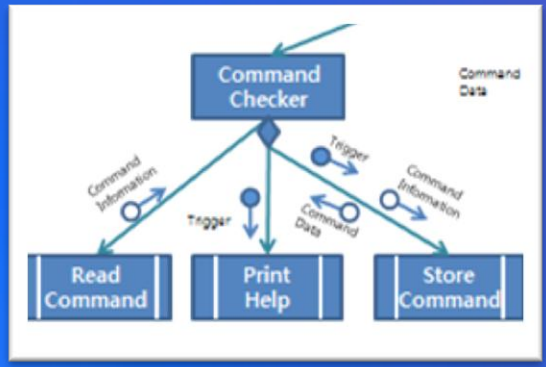
```
    if ( argc == 3){
        cmdInfo->hasError = TRUE;
        cmdInfo->inputFileName = argv1;
        cmdInfo->reportFileName = argv2;
```

If argument count is 2

```
    }else if(argc == 2)
    {
        char* buf = (char*)malloc(sizeof(char)*FILENAME_BUF);
        int index,it;

        index = 0;   it = 0;
        cmdInfo->hasError = TRUE;
        cmdInfo->inputFileName = argv1;

        strcpy(buf,argv1);
        strcat(buf,".report.txt");
        cmdInfo->reportFileName = buf;

    }else
```

Code from the SASD

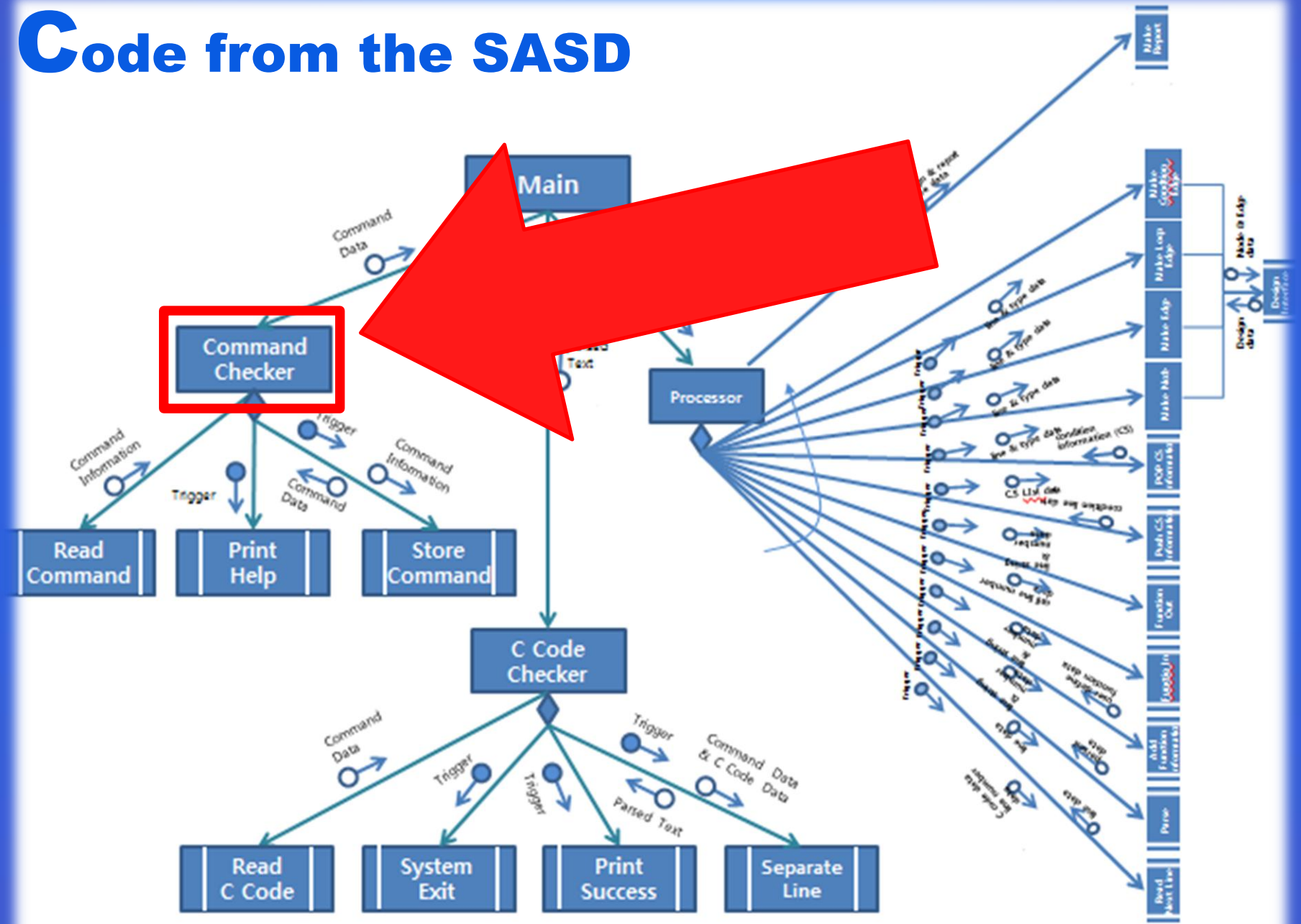
Read Command




```
        cmdInfo->hasError = TRUE;
        cmdInfo->inputFileName = argv1;

        strcpy(buf,argv1);
        strcat(buf,".report.txt");
        cmdInfo->reportFileName = buf;

    }else
    {
        return cmdInfo;
    }

#ifdef _DEBUG
        printf(" >argc:%d¥n >argv1:%s¥n >argv2:%s¥n", argc,argv1,argv2);
#endif

    return cmdInfo;
}
```

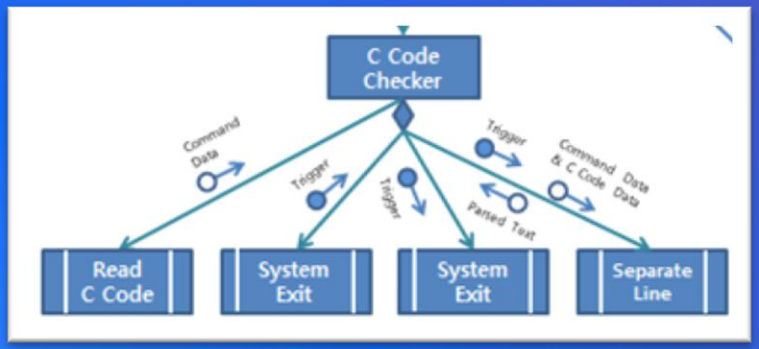
In the other case

Code from the SASD



Code from the SASD

Read C Code

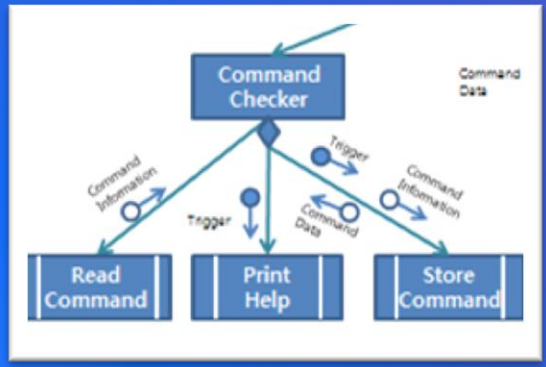



```
int main(int argc, char **argv)
{
    CommandInfo* cmdInfo;
    CommandData* cmdData;

    CCodeData* ccData;
    ParsedText* parsedData;



    cmdInfo = ReadCommand(argc,(char*)argv[1],(char*)argv[2]);

    cmdData = CommandChecker(cmdInfo);

    ccData = ReadCCode(cmdData);


    parsedData = CCodeChecker(cmdData,ccData);
```

Command Checker take command information from main function

Code from the SASD

Command Checker

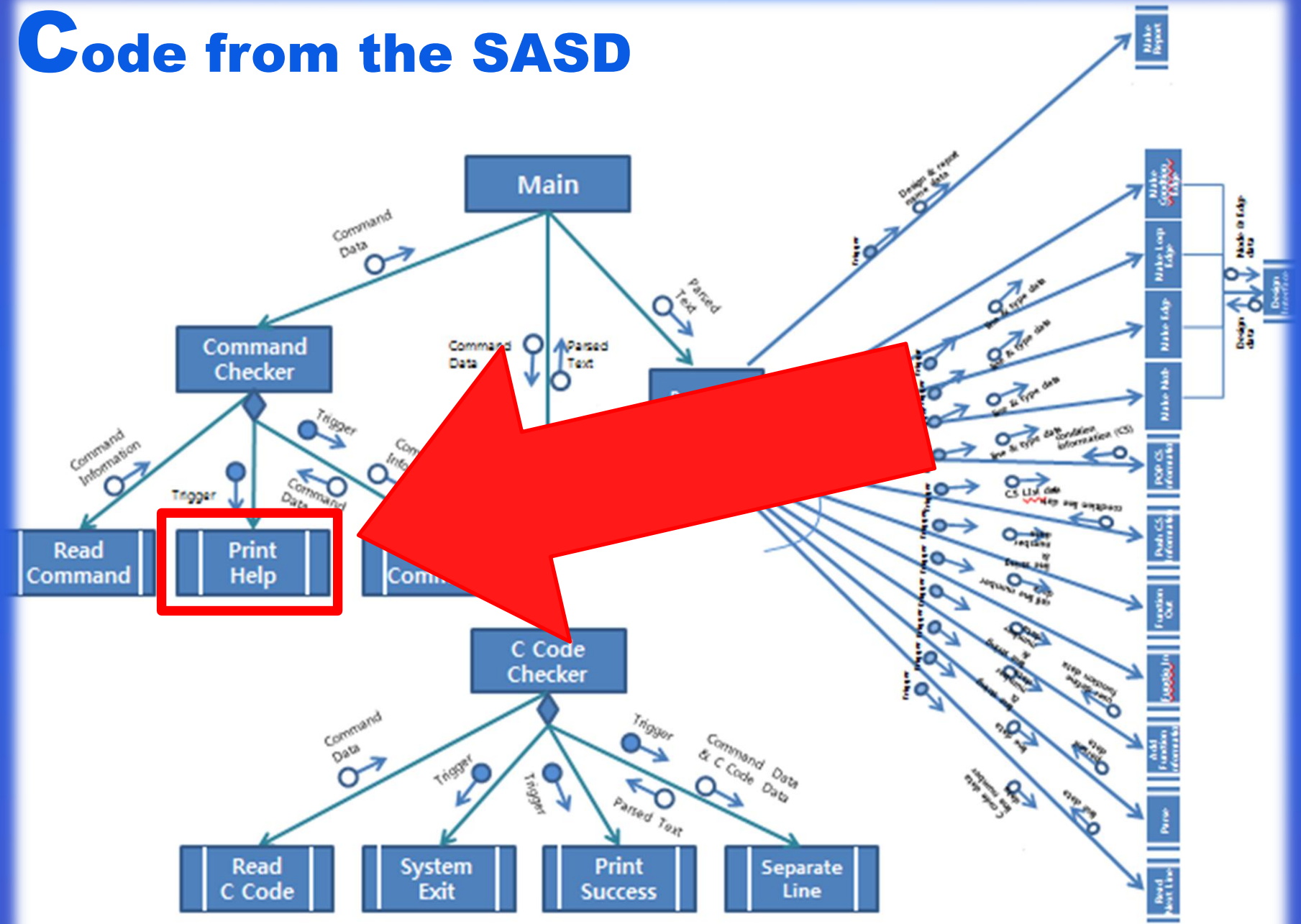



```
CommandData* CommandChecker(CommandInfo* cmdInfo)
{

    CommandData* tmp = StoreCommand(cmdInfo);
#ifdef _DEBUG
    puts("#CommandChecker called");
    printf(" >CommandChecker!₩n >inputFileName:%s₩n >outputFileName:%s₩n",cmdInfo->
inputFileName,cmdInfo->reportFileName);
#endif

    switch(cmdInfo->hasError)
    {
        case TRUE:
            return StoreCommand(cmdInfo);

        case FALSE:
            PrintHelp();
    }
        return NULL;
}
```

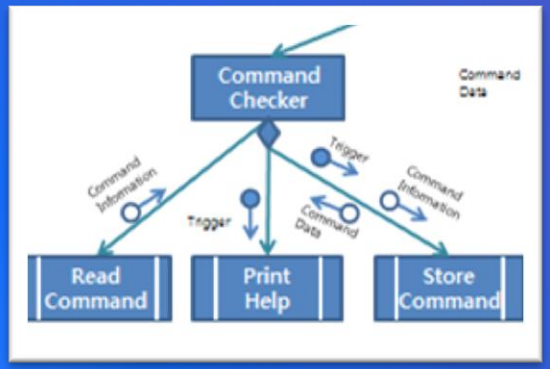
Command Checker detect command error

When command error occurred

Code from the SASD



Code from the SASD



Print Help

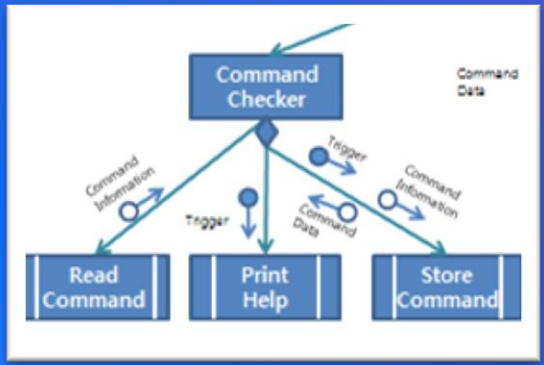
```
void PrintHelp()
{

#ifdef _DEBUG
    puts("#Print Help called");
#endif
    printf(HELP_MSG);
    exit(1);
}
```

help message in defined header

```
#define HELP_MSG ₩
    "₩n__ CFG Generator_Manual __₩n₩n₩
#Name₩n₩
 CG - CFG Generator ₩n₩n₩
#Usage₩n₩
 ./CG [ inputFileName ] [ reportFileName ] ₩n₩n₩
#Description ₩n₩n₩
this program make CFG ( Control Flwo Graph with c code ₩n₩n₩
 [ inputFileName ] ₩n₩
  > Space of inserting C Code File Name ₩n₩
  > It should be written clearly. ₩n₩n₩
 [ reportFileName ] ₩n₩
  > Space of inserting Report File Name ₩n₩
  > If it was empty, program make report File Name Like this₩n₩
    ₩' input File Name + .report.txt ₩' ₩n₩
```

Code from the SASD

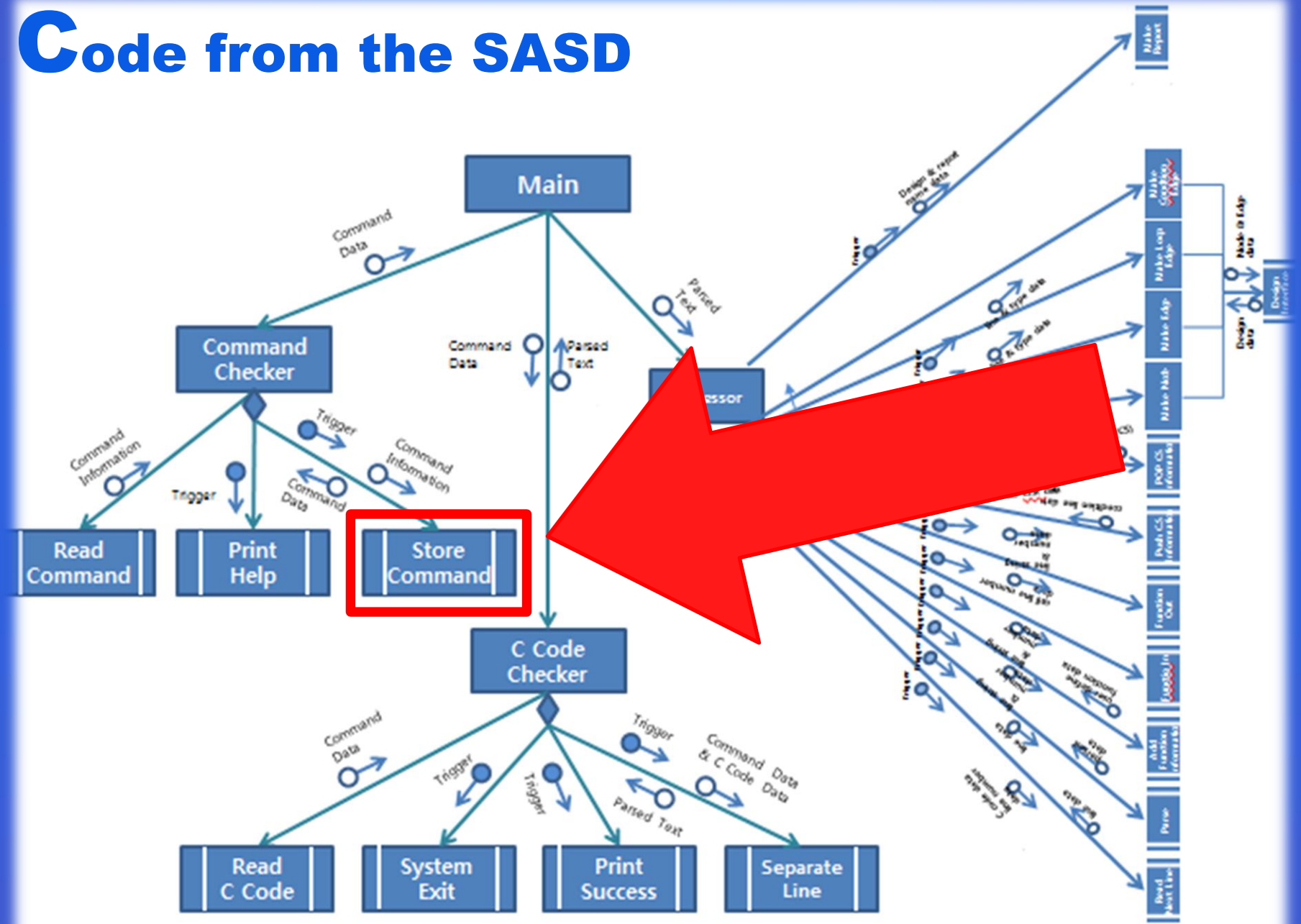
Command Checker




```
CommandData* CommandChecker(CommandInfo* cmdInfo)
{

    CommandData* tmp = StoreCommand(cmdInfo);
#ifdef _DEBUG
    puts("#CommandChecker called");
    printf(" >CommandChecker!₩n >inputFileName:%s₩n >outputFileName:%s₩n",cmdInfo->
inputFileName,cmdInfo->reportFileName);
#endif

    switch(cmdInfo->hasError)
    {
        case TRUE:
            return StoreCommand(cmdInfo);

        case FALSE:
            PrintHelp();
    }
        return NULL;
}
```

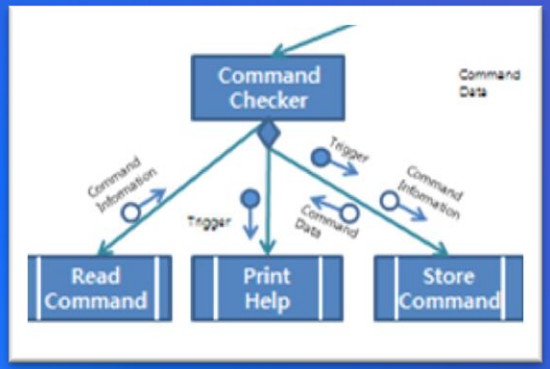
When error was not occurred

Code from the SASD



Code from the SASD

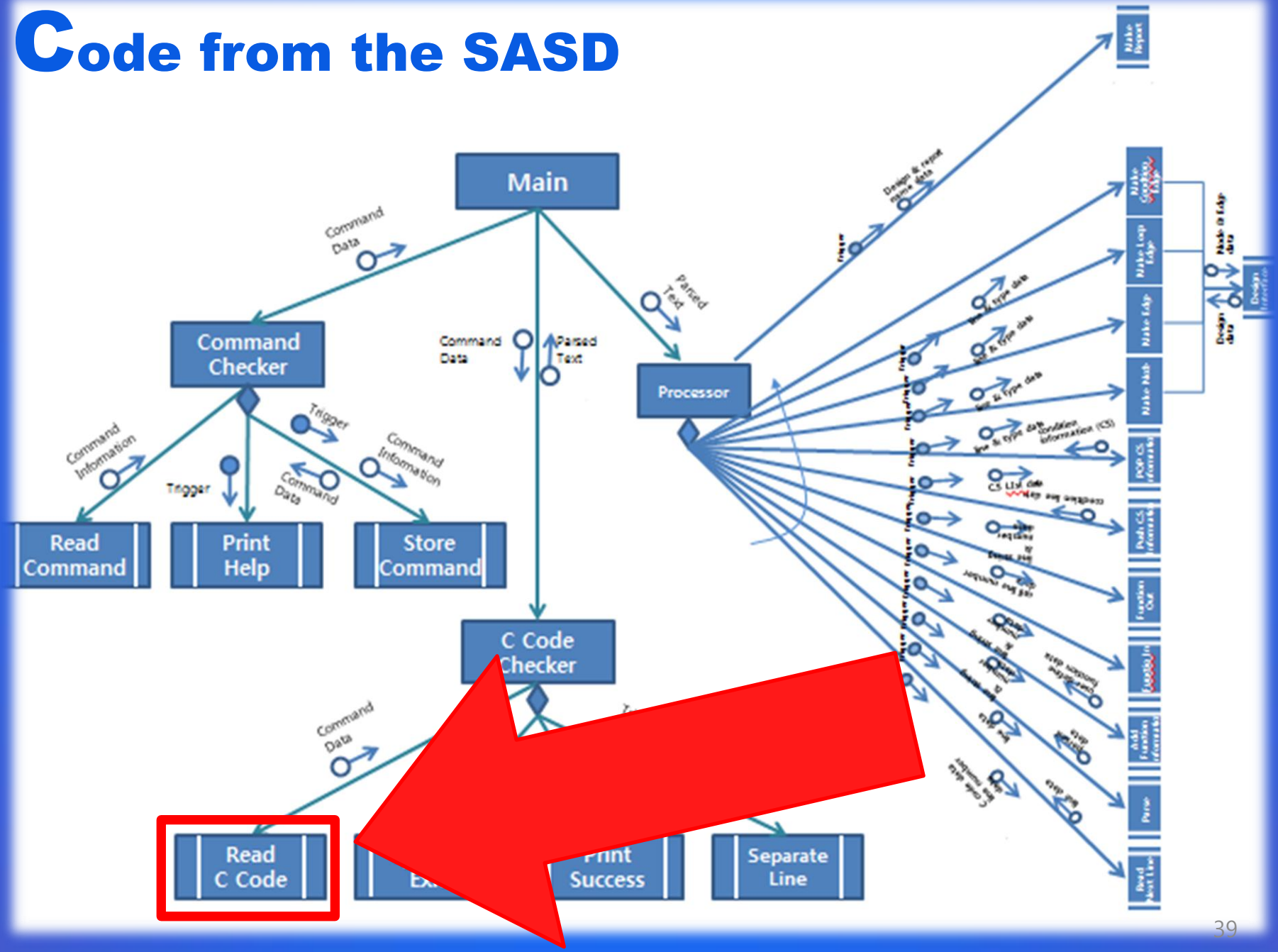
Store Command




```
CommandData* StoreCommand(CommandInfo* cmdInfo)
{

    CommandData* cmdData = (CommandData*)malloc(sizeof(CommandData));
    cmdData->inputFileName = cmdInfo->inputFileName;
    cmdData->reportFileName = cmdInfo->reportFileName;

#ifdef _DEBUG
    puts("#StoreCommand called");
    printf(" >inputFileName:%s₩n >outputFileName:%s₩n",
        cmdInfo->inputFileName, cmdInfo->reportFileName);
#endif
    return cmdData;
}
```

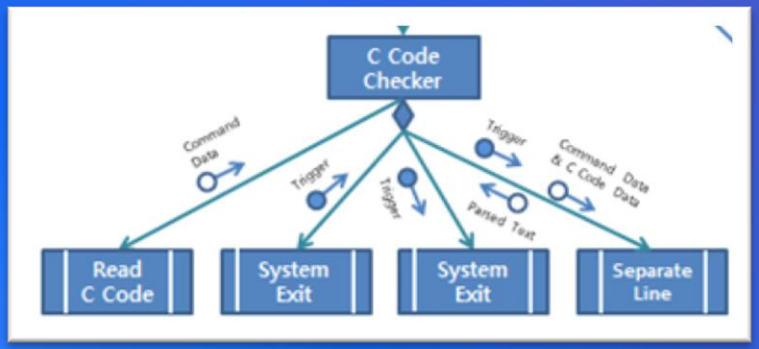
Command information convert command data

Code from the SASD



Code from the SASD

Read C Code

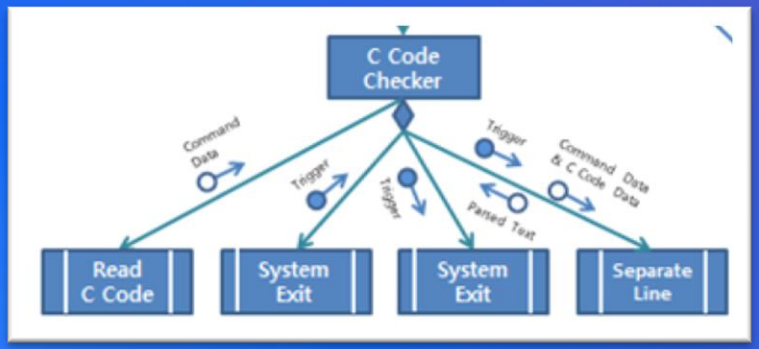



```
int main(int argc, char **argv)
{
    CommandInfo* cmdInfo;
    CommandData* cmdData;

    CCodeData* ccData;
    ParsedText* parsedData;


    cmdInfo = ReadCommand(argc,(char*)argv[1],(char*)argv[2]);

    cmdData = CommandChecker(cmdInfo);

    ccData = ReadCCode(cmdData);


    parsedData = CCodeChecker(cmdData,ccData);
```

Main function give command data to Read C Code function

Code from the SASD

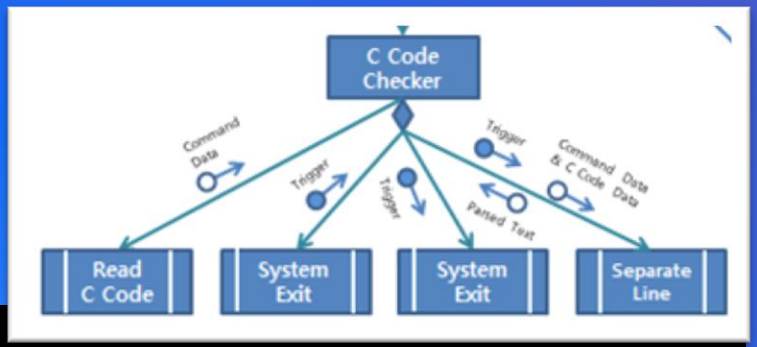
Read C Code




```
CCodeData* ReadCCode(CommandData* cmdData)
{
    int index=0;

    char* buf =(char*)malloc(sizeof(char)*BUFSIZE);
    CCodeData* ccData=(CCodeData*)malloc(sizeof(CCodeData));
    FILE* file = fopen(cmdData->inputFileName,"rt");
#ifdef _DEBUG
    puts("#ReadCCode called");
    printf(" >CommandData->inputFilename:%s\n >CommandData->reportData:%s\n",cmdDat
a->inputFileName,cmdData->reportFileName);
#endif
```

Code from the SASD

Read C Code




```
if(file==NULL)
{
    ccData->isOpen = FALSE;
    ccData->code = NULL;
```

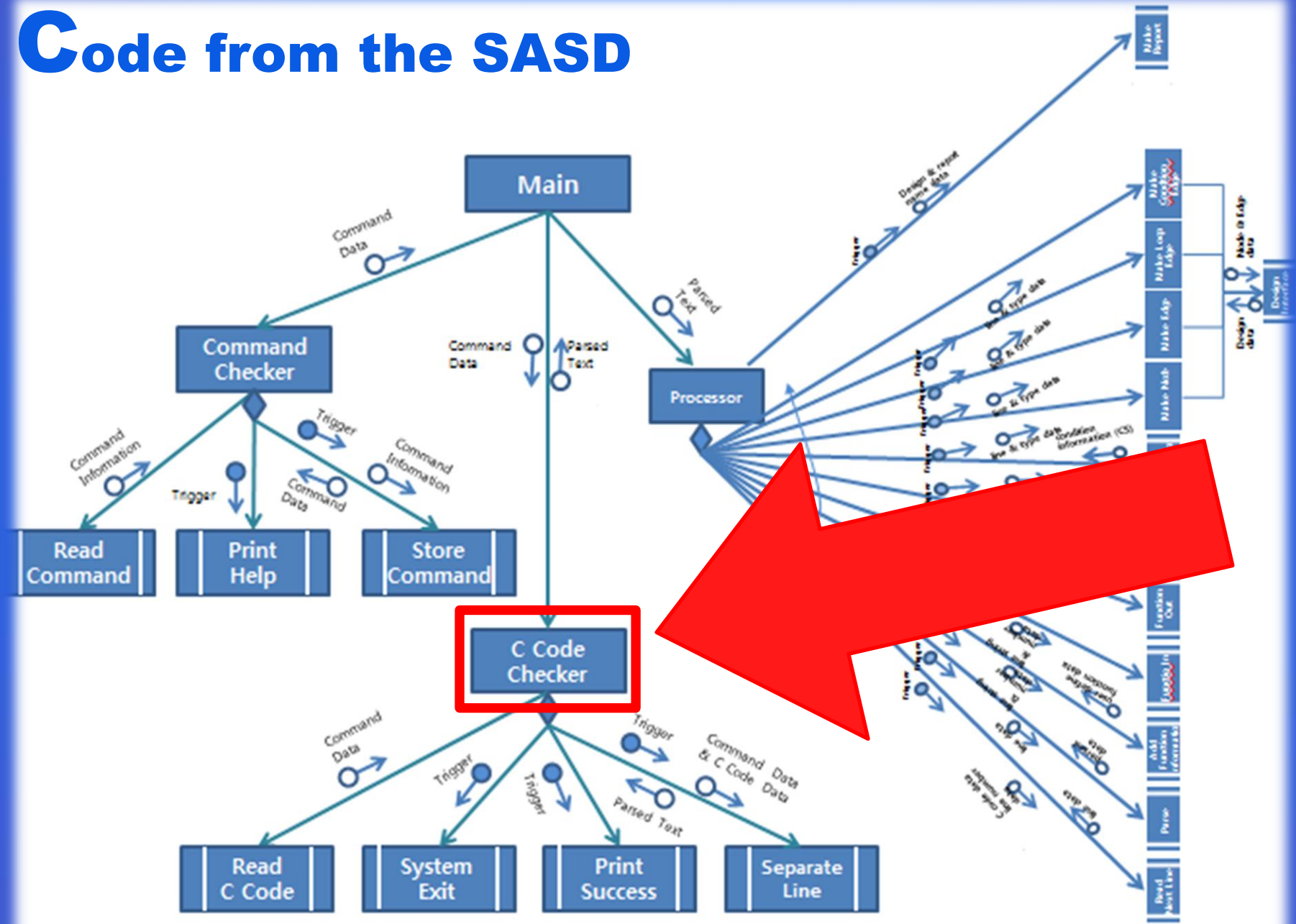
If file did not open

```
}else
{
    while(1)
    {
        buf[index++]=fgetc(file);
        if(feof(file)!=0)
            break;
    }

    buf[index]='¥0';
    ccData->code=buf;
#if
        puts("_____Read C Code");
        puts(ccData->code);
#endif
    }
#itdef _DEBUG
    printf(" >CCodeData->isOpen:%d¥n >CCodeData->code:%s¥n",ccData->isOpen,ccData->
code);
#endif
    return ccData;
}
```

If file is opened

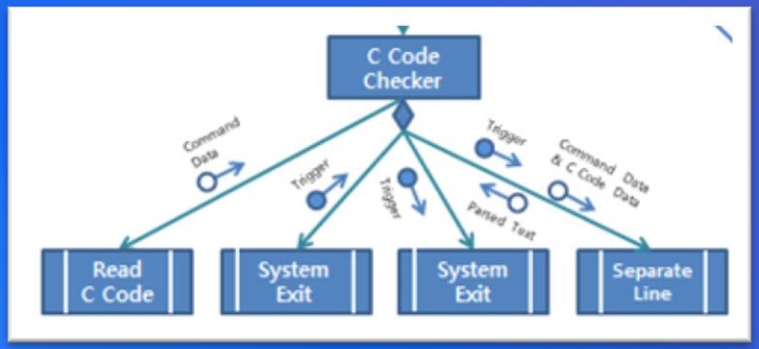
C Code Data is return

Code from the SASD



Code from the SASD

C Code Checker

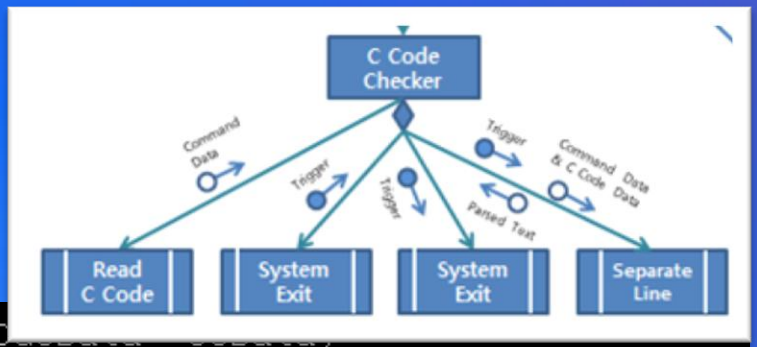



```
int main(int argc, char **argv)
{
    CommandInfo* cmdInfo;
    CommandData* cmdData;

    CCodeData* ccData;
    ParsedText* parsedData;


    cmdInfo = ReadCommand(argc,(char*)argv[1],(char*)argv[2]);

    cmdData = CommandChecker(cmdInfo);

    ccData = ReadCCode(cmdData);

    parsedData = CCodeChecker(cmdData,ccData);
```

C Code Checker take cmdData and ccData For checking file error

Code from the SASD

C Code Checker

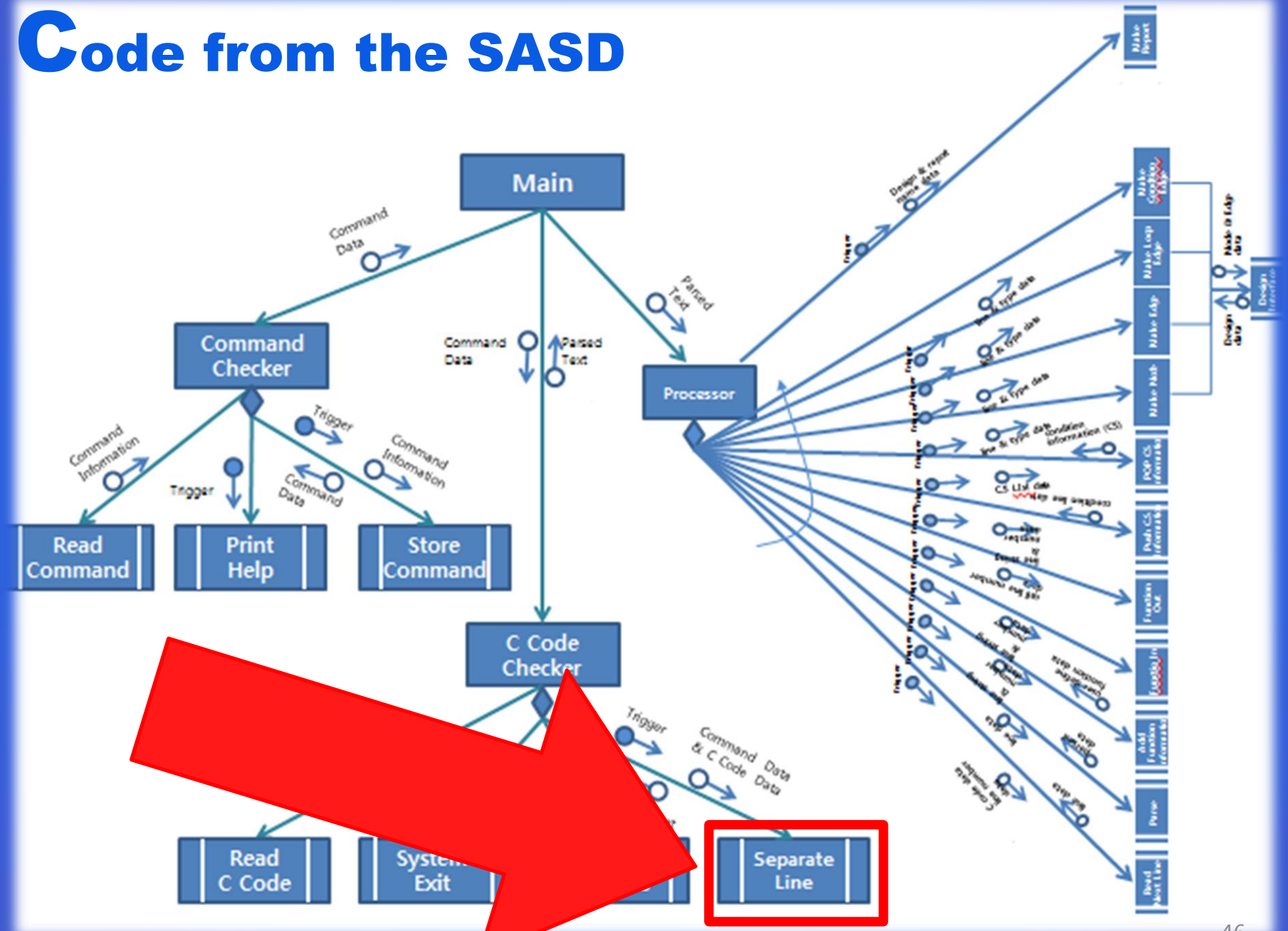



```
ParsedText* CCodeChecker(CommandData* cmdData,CC
{
#ifdef _DEBUG
    puts("#CCodeChecker called");
    printf(" >CCodeData->isOpen:%d₩n >CCodeData->code:%s₩n",ccData->isOpen,ccData->
code);
#endif
    switch(ccData->isOpen)
    {
    case TRUE:
        PrintSuccess();
        return SeperateLine(cmdData,ccData);

    case FALSE:
        SystemExit();
    }
#ifdef _DEBUG
    printf(" >CCodeChekcer : ccData -> isOpen has error element");
#endif
    SystemExit();
    return NULL;
}
```

The case no File error

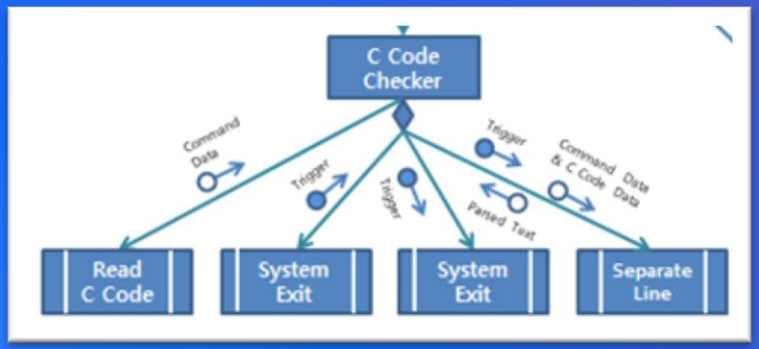
The case had File error

Code from the SASD



Code from the SASD

Separate Line



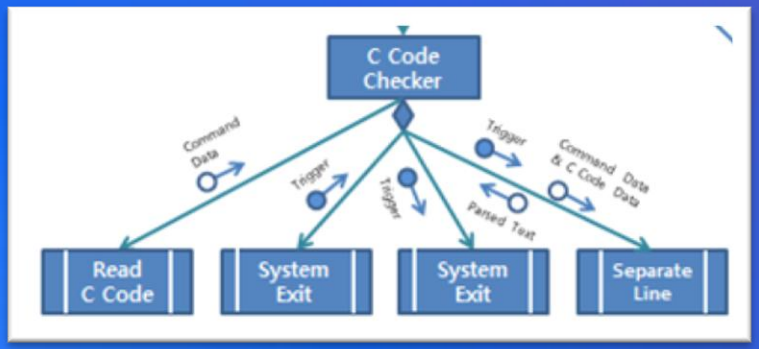
Remove White space

```
#include <stdio.h>

int main(

                )
{
        int i,j;
        j=10;

        for(i=0; i<10; i++)
        {
        printf("%d\n",i);
        }

        return 0;
}
```

```
#include <stdio.h>
int main( )
{
int i,j;
j=10;
for(i=0; i<10; i++)
{
printf("%d\n",i);
}
return 0;
}
```

Code from the SASD

Separate Line



Remove White space

```
//test2
#include <stdio.h>

int main(

                )
{
        int i,j;
        j=10;

        for(i=0;

                i<10; i++)
        {
        printf("%d\n",i);
        }

        return 0;
}
```

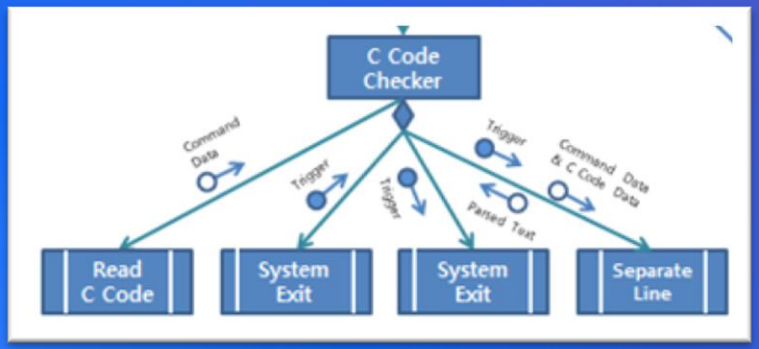
```
//test3

#include <stdio.h>

int main(                                       )
{
        int i,
              j;
        j=10;

        for(i=0;                i<10; i++)
        {
        printf("%d\n",i);
        }

        return 0;
}
```

They are SAME CODE !

```
#include <stdio.h>
int main( )
{
int i,j;
j=10;
for(i=0; i<10; i++)
{
printf("%d\n",i);
}
return 0;
}
```

Code from the SASD

Separate Line

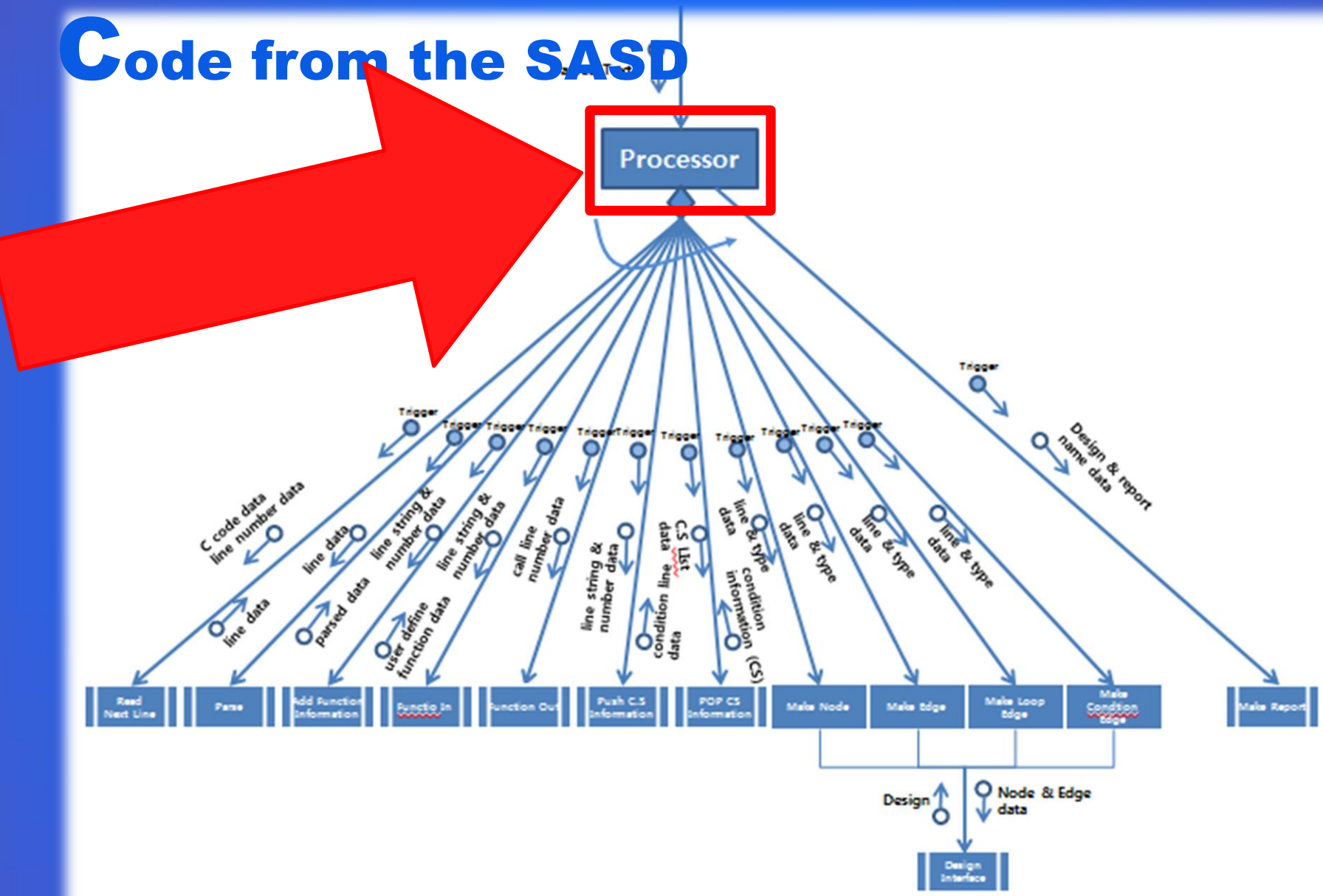


Remove Comment Lines

```
//파싱되면서 주석이 사라지는 것을 보여주기 위한 test
#include <stdio.h>

typedef unsigned char *byte_pointer; //주소값을 1byte 단위로

void show_bytes(byte_pointer start, int len)

{

    int i;

    for (i = 0; i < len; i++)

        printf(" %.2x", start[i]); //%.2x는 16진수로 2자리(1by

    printf("\n");

}


void show_float(float x) //floatx의 주소에 저장 되 있는 값을
tle endian 이다



{

    show_bytes((byte_pointer) &x, sizeof(float));

}




int main()
{
    float f=45000.67;
    show_float(f);
    printf("%f",f);
}
```

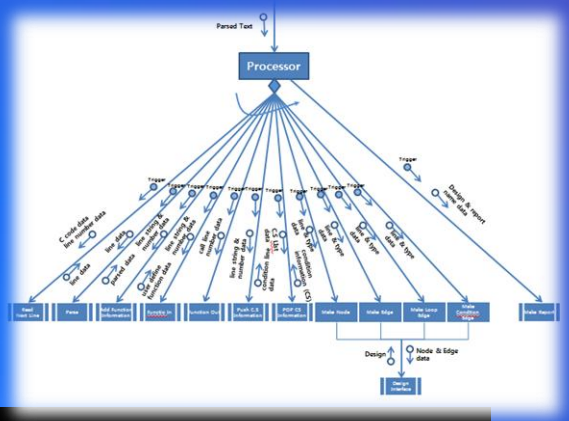
```
#include <stdio.h>
typedef unsigned char *byte_pointer;
void show_bytes(byte_pointer start, int len)
{
int i;
for (i = 0; i < len; i++)
printf(" %.2x", start[i]);

printf("\n");
}
void show_float(float x)
{
show_bytes((byte_pointer) &x, sizeof(float));
}
int main()
{
float f=45000.67;
show_float(f);
printf("%f",f);
}
```

Code from the SASD



Code from the SASD

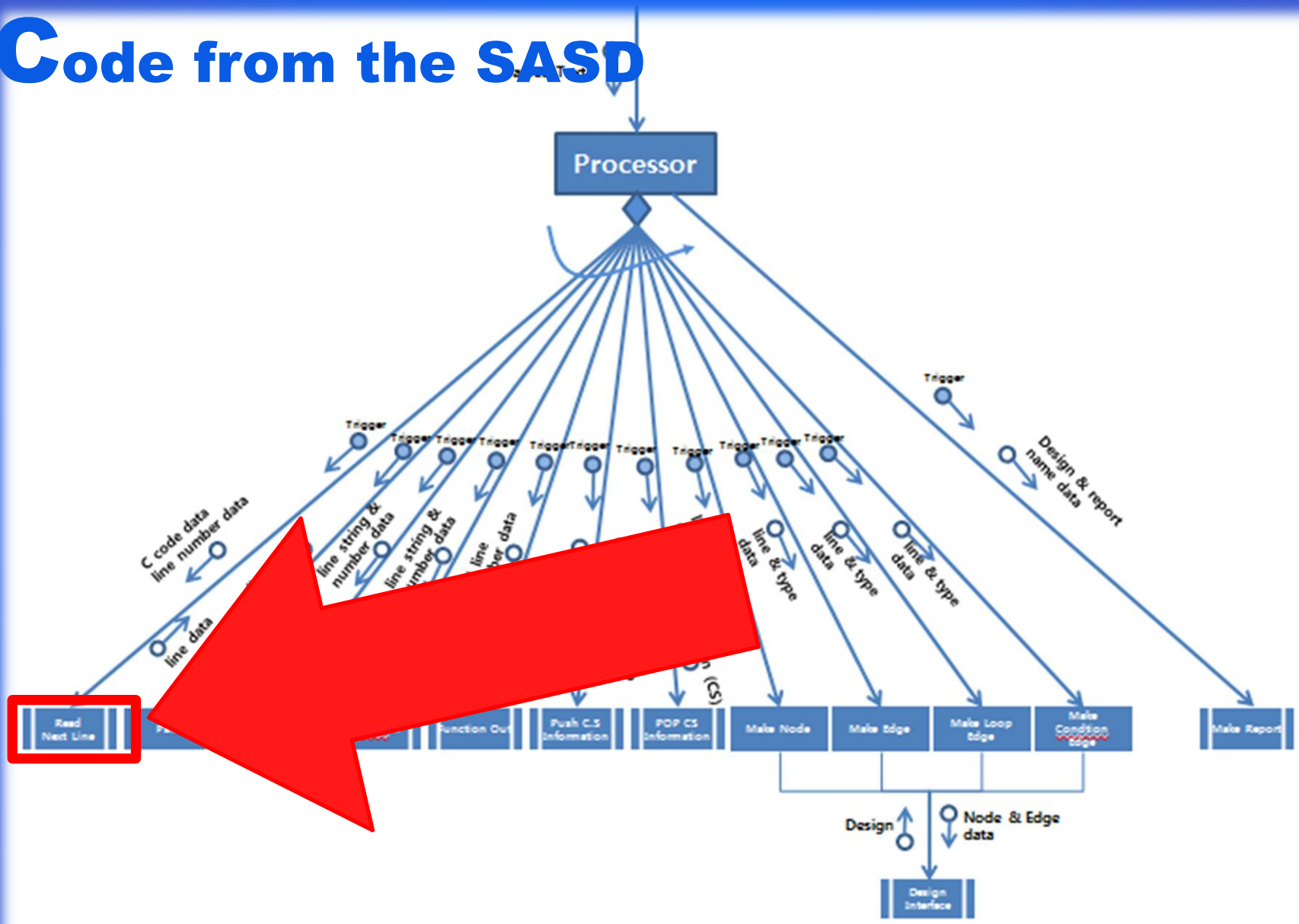
Processor




```
int main(int argc, char **argv)
{
    CommandInfo* cmdInfo;
    CommandData* cmdData;

    CCodeData* ccData;
    ParsedText* parsedData;



    cmdInfo = ReadCommand(argc,(char*)argv[1],(char*)argv[2]);

    cmdData = CommandChecker(cmdInfo);

    ccData = ReadCCode(cmdData);


    parsedData = CCodeChecker(cmdData,ccData);

    processor( parsedData );

    return 0;
```

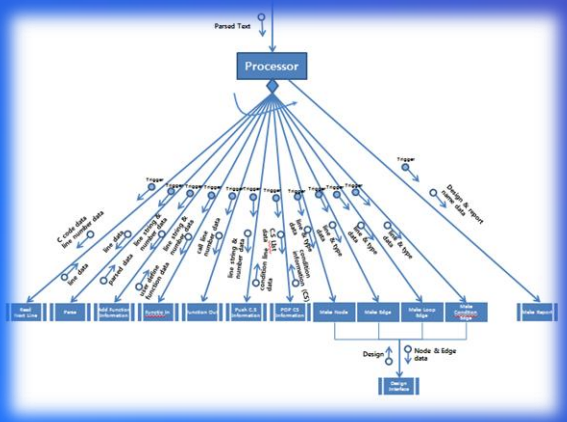
This process take parsed text from C Code Checker in main

Code from the SASD



Code from the SASD

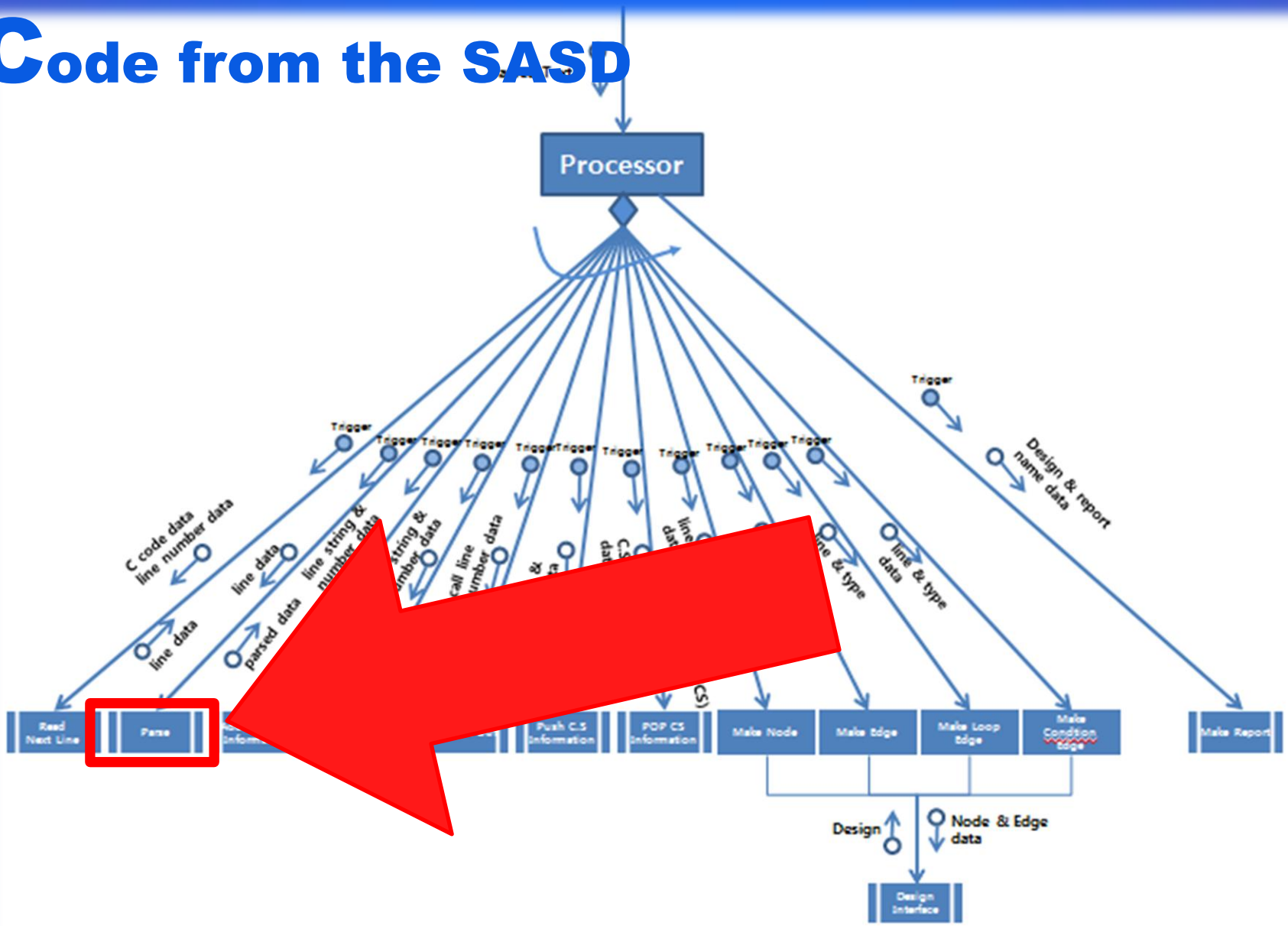
Processor




```
    DesignInterface( &design, makeNode( "start", start_node ), makeEdge( normal_edge ) );

    do{
        readNextLine( fp, &num, &line );
        type = parse( line, braket, &Flist );


        switch( type ){
            case open_braket:
                braket++;
                inBraket = TRUE;
                break;
            case close_braket:
                if( inMain == TRUE ){
                    if( braket == 1 ){
                        type = end_main;
                        inMain = FALSE;
                    }else if( braket > 1 ){
                        type = end_cs;
                    }
                }

                braket--;
                break;
            case start_main:
                DesignInterface( &design, makeNode( line[cur], normal_node ), makeEdge( normal_e
dge ) );
```

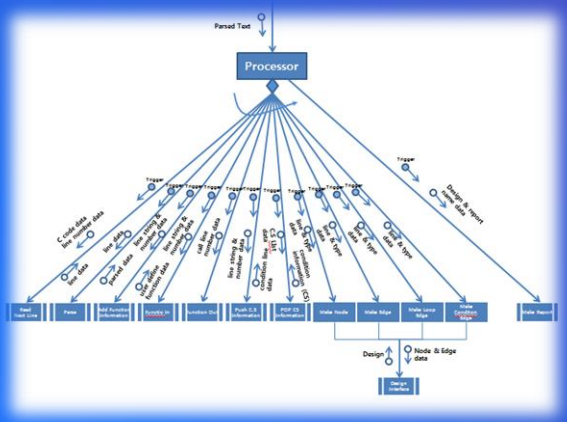
It read each Line from the text file

Code from the SASD



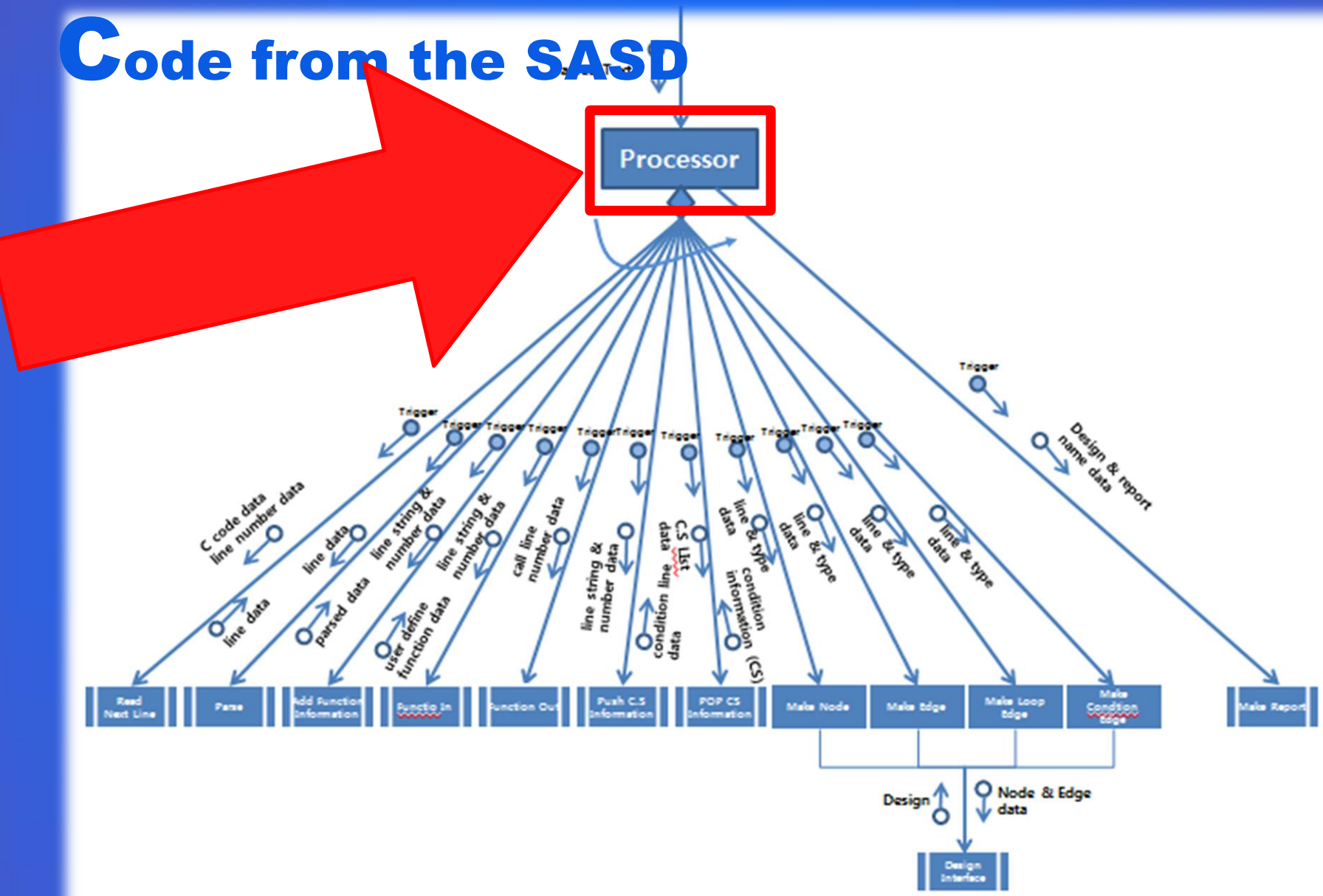
Code from the SASD

Processor



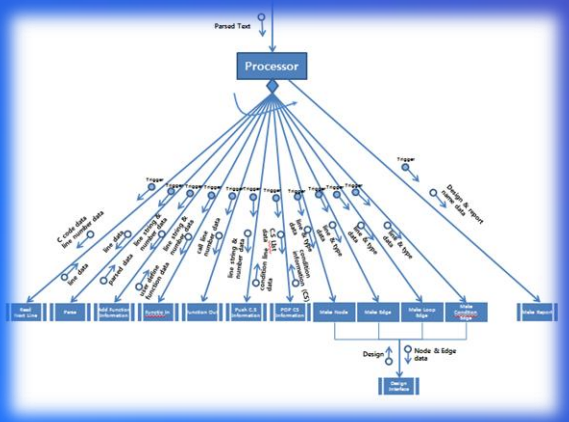

```
    DesignInterface( &design, makeNode( "start", start_node ), makeEdge( normal_edge ) );

    do{
        readNextLine( fp, &num, &line );
        type = parse( line, braket, &Flist );

        switch( type ){
            case open_braket:
                braket++;
                inBraket = TRUE;
                break;
            case close_braket:
                if( inMain == TRUE ){
                    if( braket == 1 ){
                        type = end_main;
                        inMain = FALSE;
                    }else if( braket > 1 ){
                        type = end_cs;
                    }
                }

                braket--;
                break;
            case start_main:
                DesignInterface( &design, makeNode( line[cur], normal_node ), makeEdge( normal_e
dge ) );
```

Code from the SASD



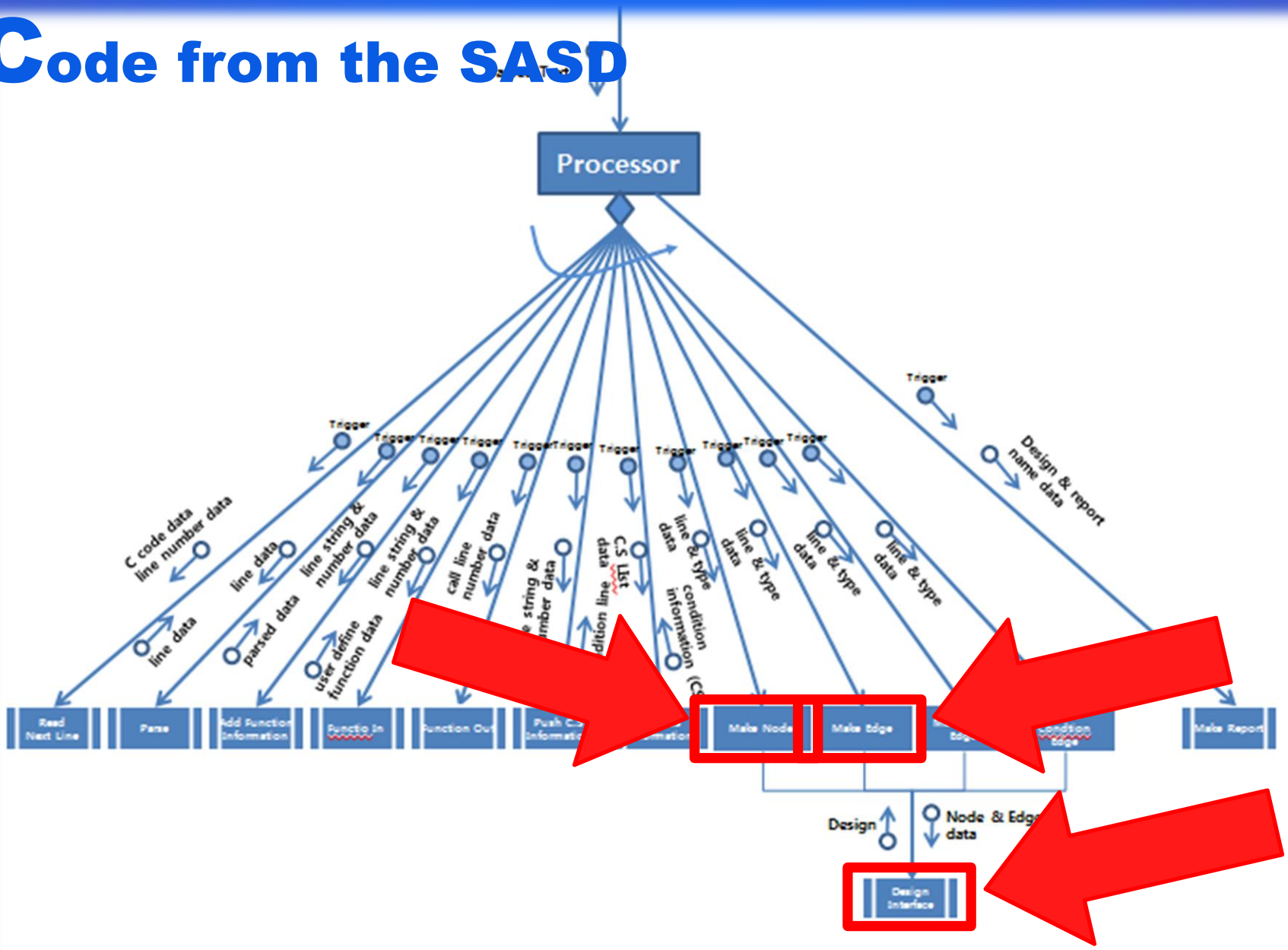
Code from the SASD

Processor



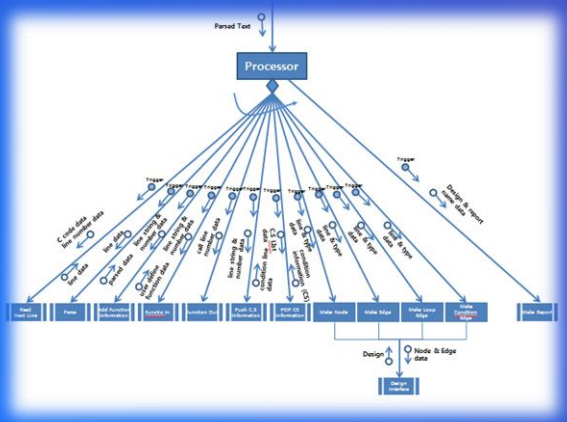

```
            case start_cs:
                if( inMain == TRUE ){
                    if( ( strchr( line[1], ';' ) == line[1] + strlen( line[1] ) - 1 ) ){
                        type = normal_line;
                    }
                    if( strstr( line[1], "while" ) == line[1] || strstr( line[1], "for" ) == lin
e[1] ){

                        csType = while_loop;
                    }else if( strstr( line[1], "do" ) == line[1] ){
                        csType = do_loop;
                    }else if( strstr( line[1], "if" ) == line[1] ){
                        if( strstr( line[2], "if" ) == line[2] ){
                            csType = if_if;
                        }else{
                            csType = cond;
                        }
                    }else if( strstr( line[1], "switch" ) == line[1] ){
                        csType = switch_cs;
                    }else if( strstr( line[1], "case" ) == line[1] ){
                        csType = case_cs;
                    }else if( strstr( line[1], "default" ) == line[1] ){
                        csType = default_cs;
                    }else if( strstr( line[1], "break" ) == line[1] ){

                            csType = break_cs;

                        type = start_cs;
                    }else if( strstr( line[1], "continue;" ) == line[1] ){
                        csType = continue_cs;
```

Code from the SASD



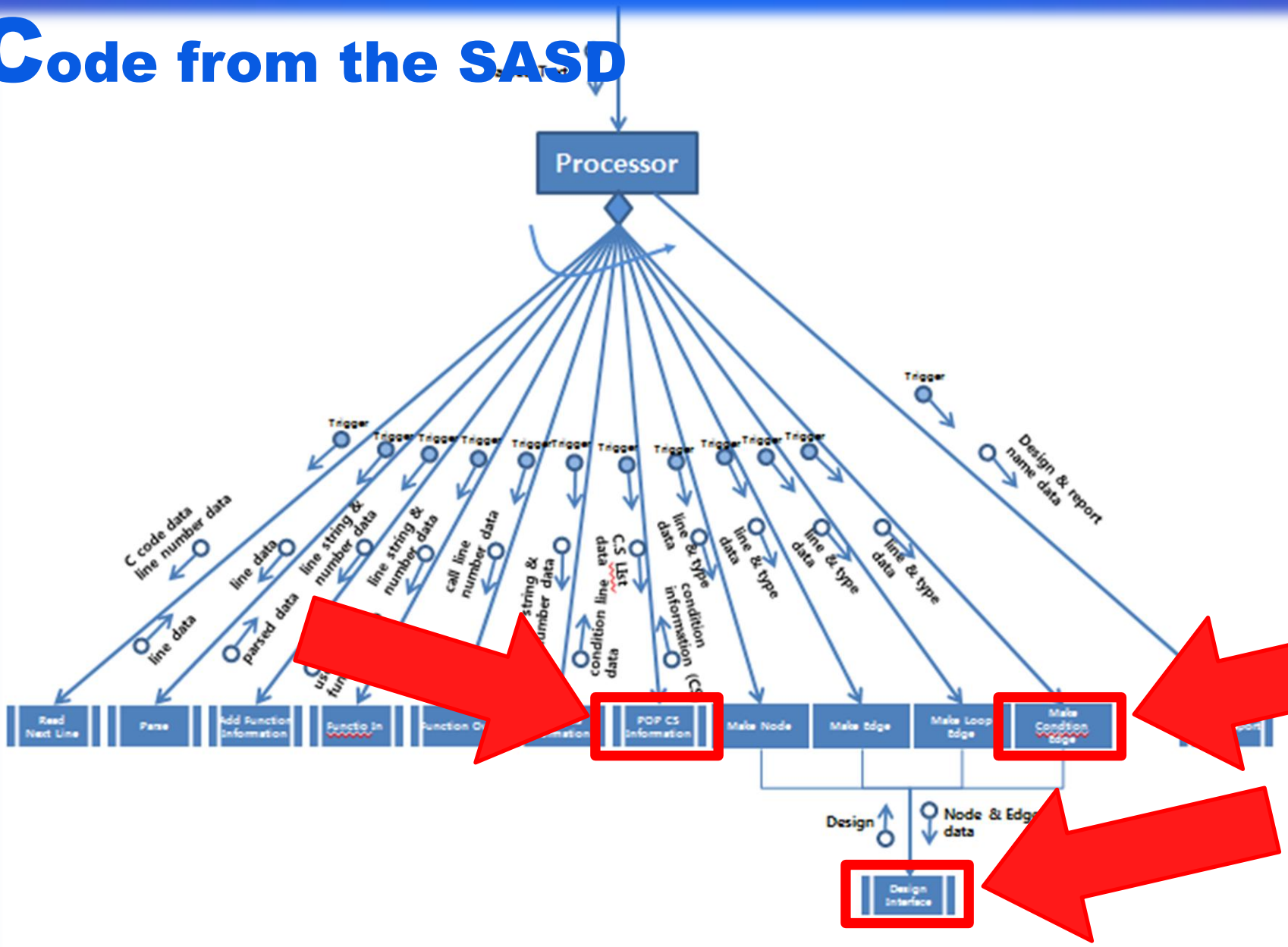
Code from the SASD

Processor



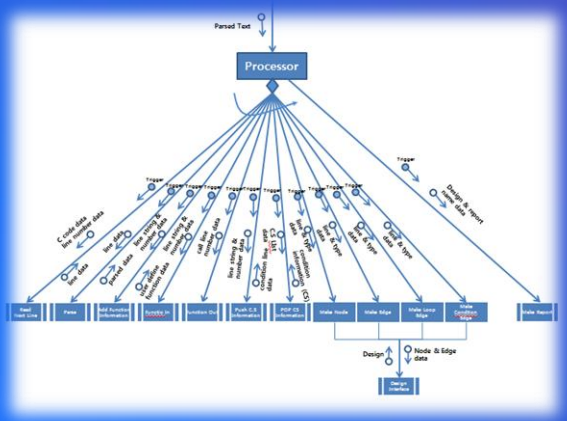

```
        if( inMain == TRUE ){
            if( braket >= 1 ){
                if( type == normal_line ){
                    DesignInterface( &design, makeNode( line[cur], normal_node ), makeEdge( norm
al_edge ) );
                }else if( type == dummy ){
                    DesignInterface( &design, makeNode( "dummy", dummy_node ), makeEdge( normal_
edge ) );
                }else if( type == start_cs ){
                    if( csType != break_cs && csType != continue_cs
                        && csType != return_func_cs && csType != return_main_cs ){
                        inBraket = FALSE;
                    }

                    switch( csType ){
                        case cond:

                            pushCSinfor( &csList, design.nodesize, cond );
                            DesignInterface( &design, makeNode( line[cur], if_node ), makeEdge(
normal_edge ) );
                            break;
                        case while_loop:

                            pushCSinfor( &csList, design.nodesize, while_loop );
                            DesignInterface( &design, makeNode( line[cur], while_node ), makeEdg
e( normal_edge ) );
                            break;
```

Code from the SASD



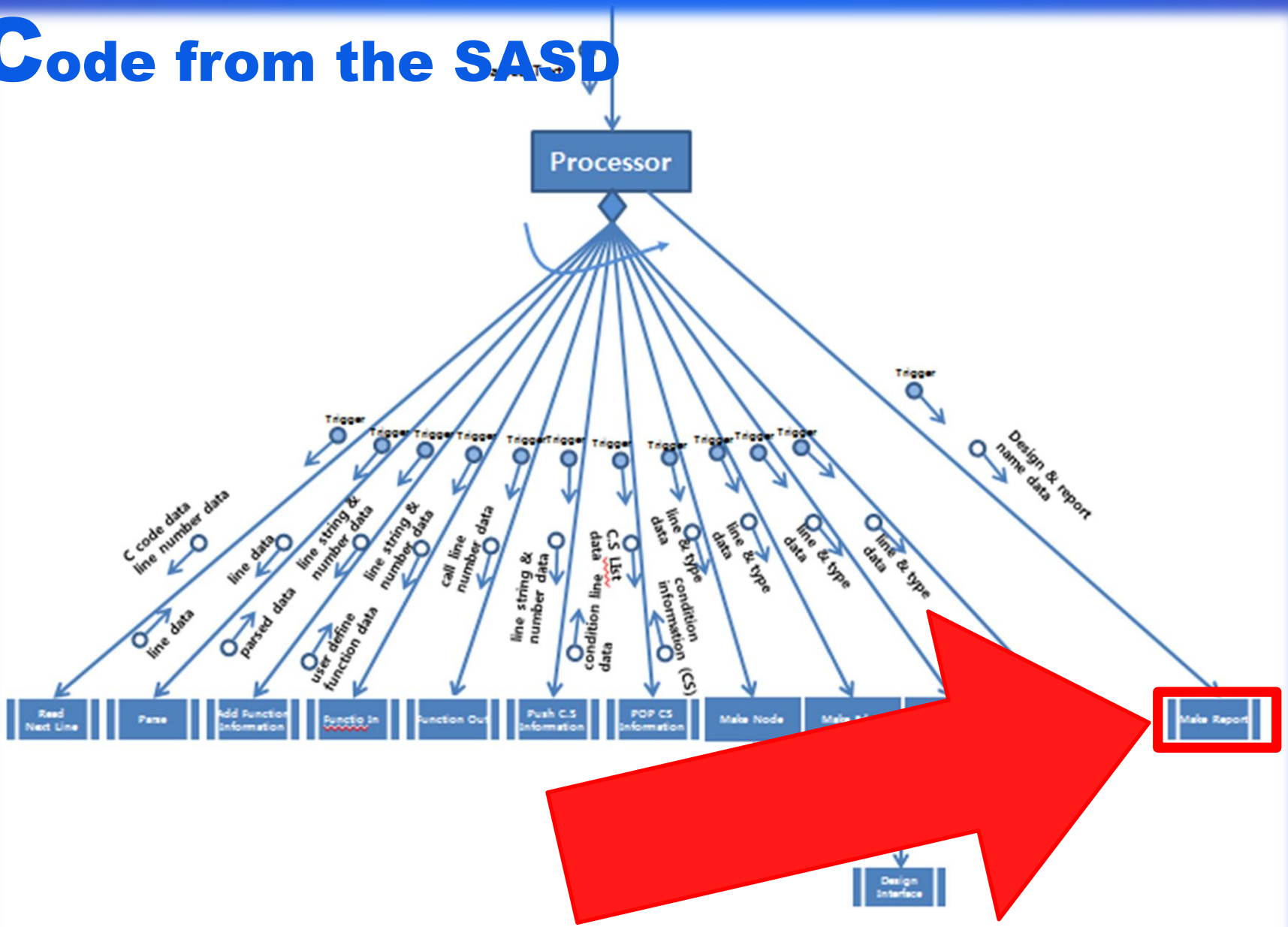
Code from the SASD

Processor



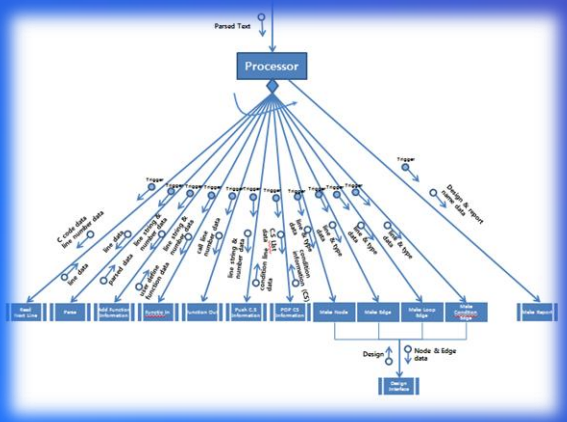

```
                    break;
                case return_func_cs:
                    pushCSinfor( &csList, design.nodesize, return_func_cs );
                    DesignInterface( &design, makeNode( line[cur], break_node ), NULL );
                    if( inBraket == FALSE ){
                        popCSinfor( &csList, &cs );
                        DesignInterface( &design, NULL, makeJumpEdge( cond_edge, cs->nod
eNum, design.nodesize ) );
                        inBraket = TRUE;
                    }
                    break;
            }else if( type == end_cs ){
                do{
                    popCSinfor( &csList, &cs );
                if(  == NULL ){
                    break;
                }
                csType = cs->csType;

                if( csType == switch_cs ){
                }else if( csType == cond || csType == else_cs || csType == case_cs ){
                    DesignInterface( &design, NULL, MakeConditionEdge( cs, design.nodesi
ze ) );
                }else if( csType == if_if ){
```

Code from the SASD



Code from the SASD

Processor




```
            }
        }
        DesignInterface( &design, makeNode( "FINISH :D", finish_node ), NULL );
    }

    if( feof( fp ) ){
        DesignInterface( &design, makeNode( "EOF @_@", eof_node ), NULL );
        break;
    }
    printf( "l:%s type:%d ct:%d ns:%d es:%dWn", line[ cur ], type, csType, design.nodesize,
design.edgesize );
    MakeReport( &design, NULL );
}while( type != end_main && line[cur] != NULL );

fclose( fp );
MakeReport( &design, parsedData->reportName );
}
```