SASD of CFG Generator

Team 6
Speaker : 200811425 김평석
200811435 신성호
200811451 이형열
200811454 전인서

Contents

- **Structured Analysis**
  - **Statement of Purpose**
  - **System Context Diagram**
  - **Event List**
  - **Data Flow Diagram**
  - **Final State Machine**
  - **Data Dictionary**
  - **Process Specification**
- **Structured Design**
  - **Structured Charts**

Structured Analysis

Statement of Purpose

Statement of Purpose

Statement of Purpose
입력은 C 코드 출력은 CFG를 그린 txt파일로 한다.
CUI 에서 Command 가 성공적으로 입력되지 않으면 Command에 대한 도움말을 출력해준다.
C 코드가 성공적으로 입력되면 입력성공이란 말과 함께 CFG로의 변환과정을 보여주며 변환한다. 만약 성공적으로 입력되지 않으면 입력실패란 말과 함께 프로그램을 종료한다.
입력된 C 코드는 text 파일로 변환해서 CFG로 디자인한다.
변환할 때에는 C 코드를 위에서부터 1 Line 씩 읽는다.

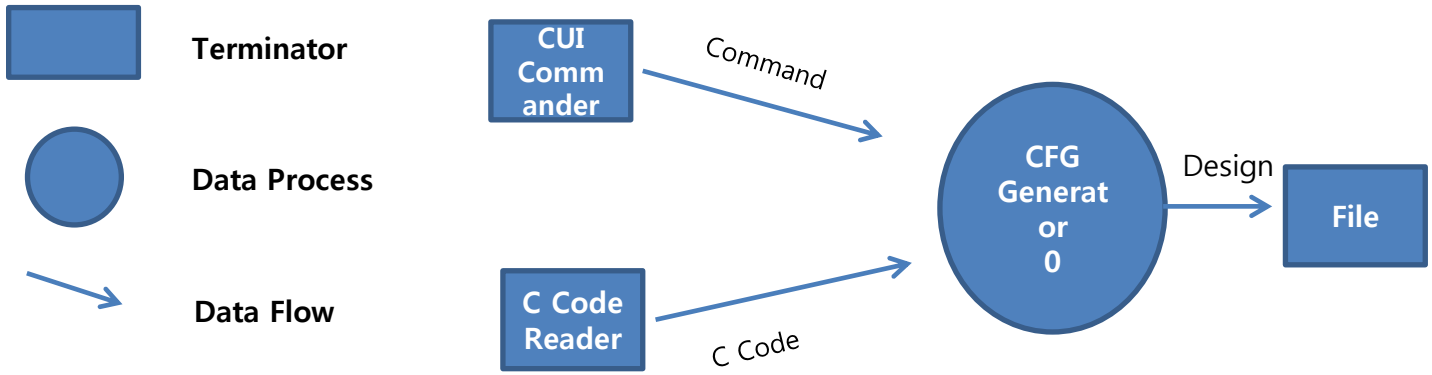
Statement of Purpose

Statement of Purpose
CFG는 Node 와 Edge 로 이루어져 있다.
하나의 Node 마다 1 Line 씩의 정보를 갖는다.
각각의 Node 는 C 코드의 위쪽 Line 부터 차례대로 번호를 갖는다.
제어문이 있을 경우, 각각의 제어문에 따른 Edge를 그리는 방법을 가진다.
함수호출이 있을 경우, Stack 을 사용하여 현재 어떤 함수 내에 있는지, 어떤 함수가 호출되어 있는지의 정보를 저장한다.
평범한 문장에서는 하나의 Node에서 하나의 Edge가 나가며, 특수한 문장(제어문)에서는 하나의 Node에서 둘의 Edge가 나간다.

Statement of Purpose

Statement of Purpose
포인터를 사용하지 않은 C 코드를 대상으로 한다.
단일 파일로 되어 있는 코드에 대하여 작동한다.
사용자가 정의한 헤더를 사용한 파일에 대해서는 작동하지 않는다.
코드의 길이는 100~200줄 내외의 프로그램을 대상으로 한다.
Main Function을 포함하는 코드여야 한다.
goto문을 사용하지 않은 C 코드를 대상으로 한다.
사용자 정의 함수의 선언과 정의가 main함수 이전에 되어있어야 한다.

System Context Diagram & Event List



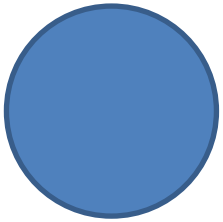
Input/Output Event	Description	Format/Type
Command	String that includes Input File Name and Output File Name Ex) ./CG [input file name] [output file name]	char*
C Code	C code that will convert CFG C code must have *.c file extension C code Generated by C standard	*.c
Design	Data that saves information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

Data Flow Diagram

Notation of Data Flow Diagram

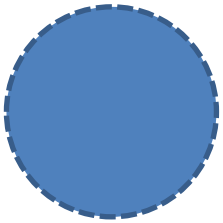
Terminator

Data Store



Data Process

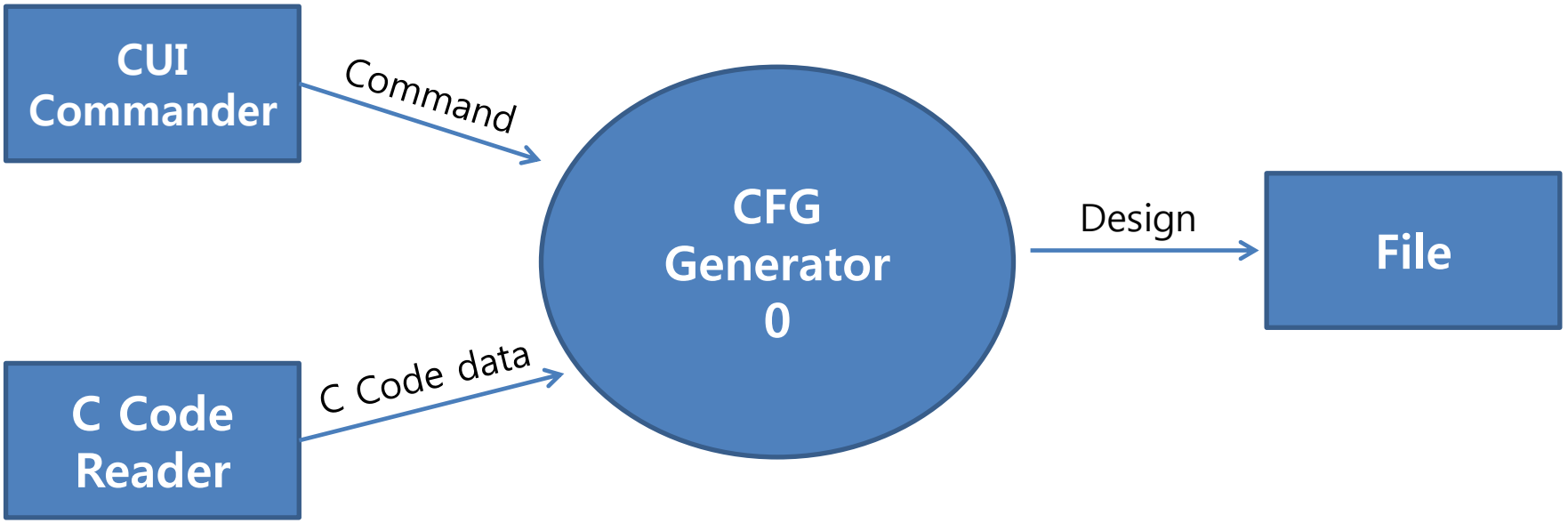
Data Flow



Control Process

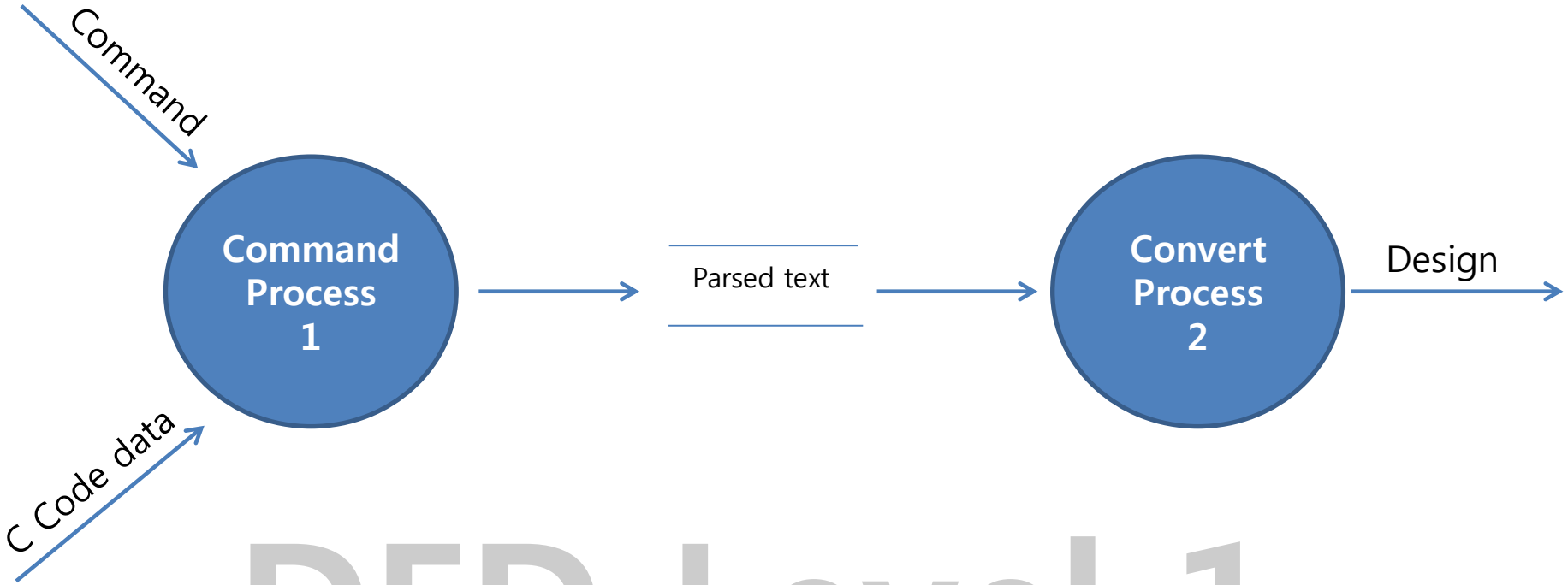
Control Flow

DFD Level 0



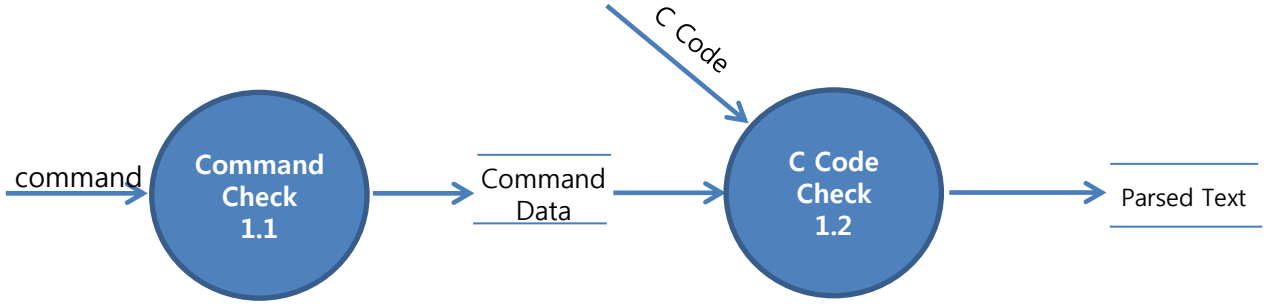
DFD Level 0

DFD Level 1

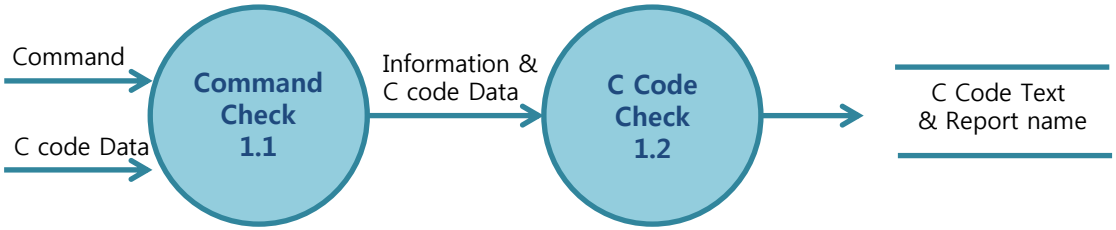


DFD Level 1

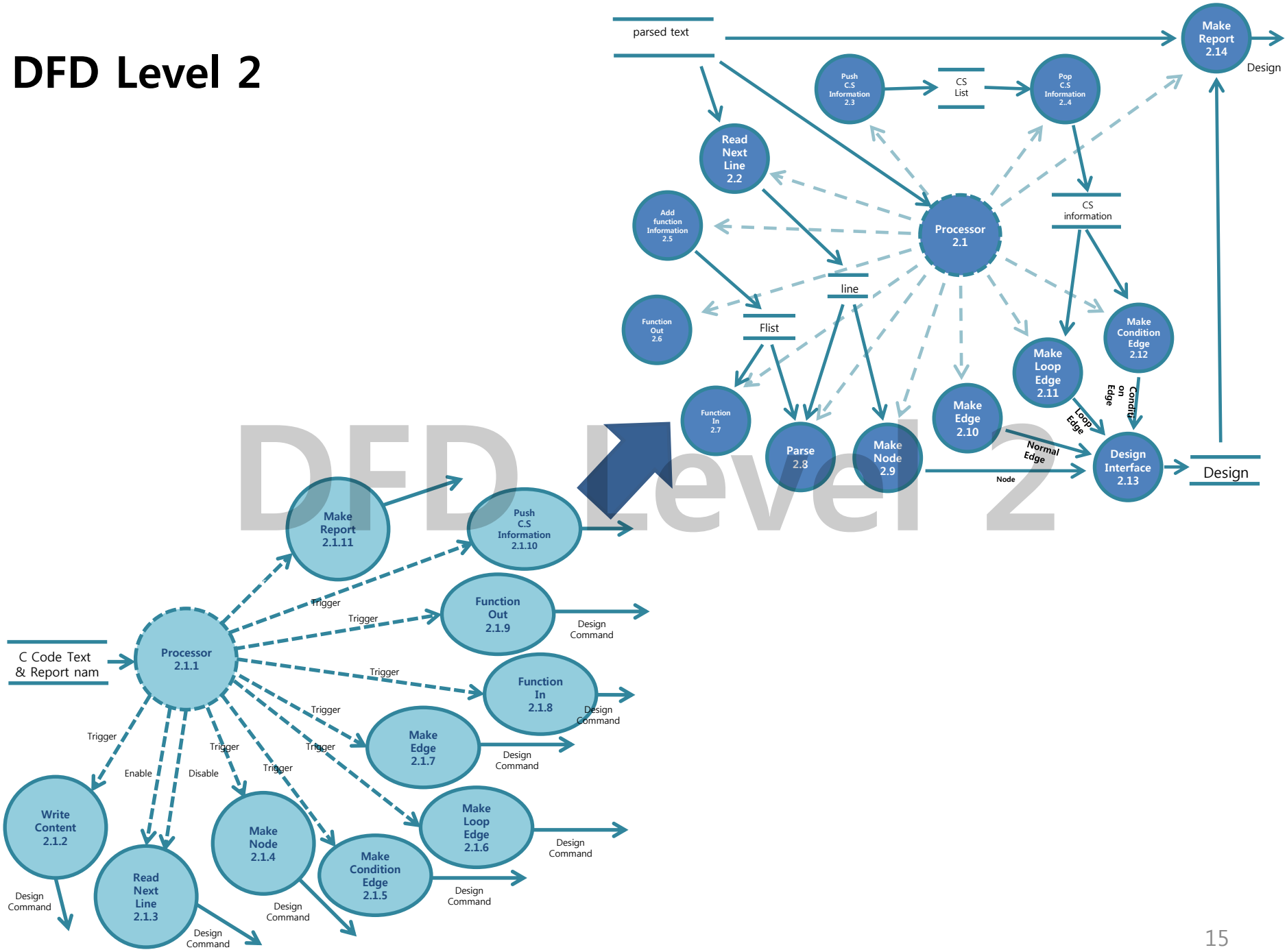
DFD Level 2



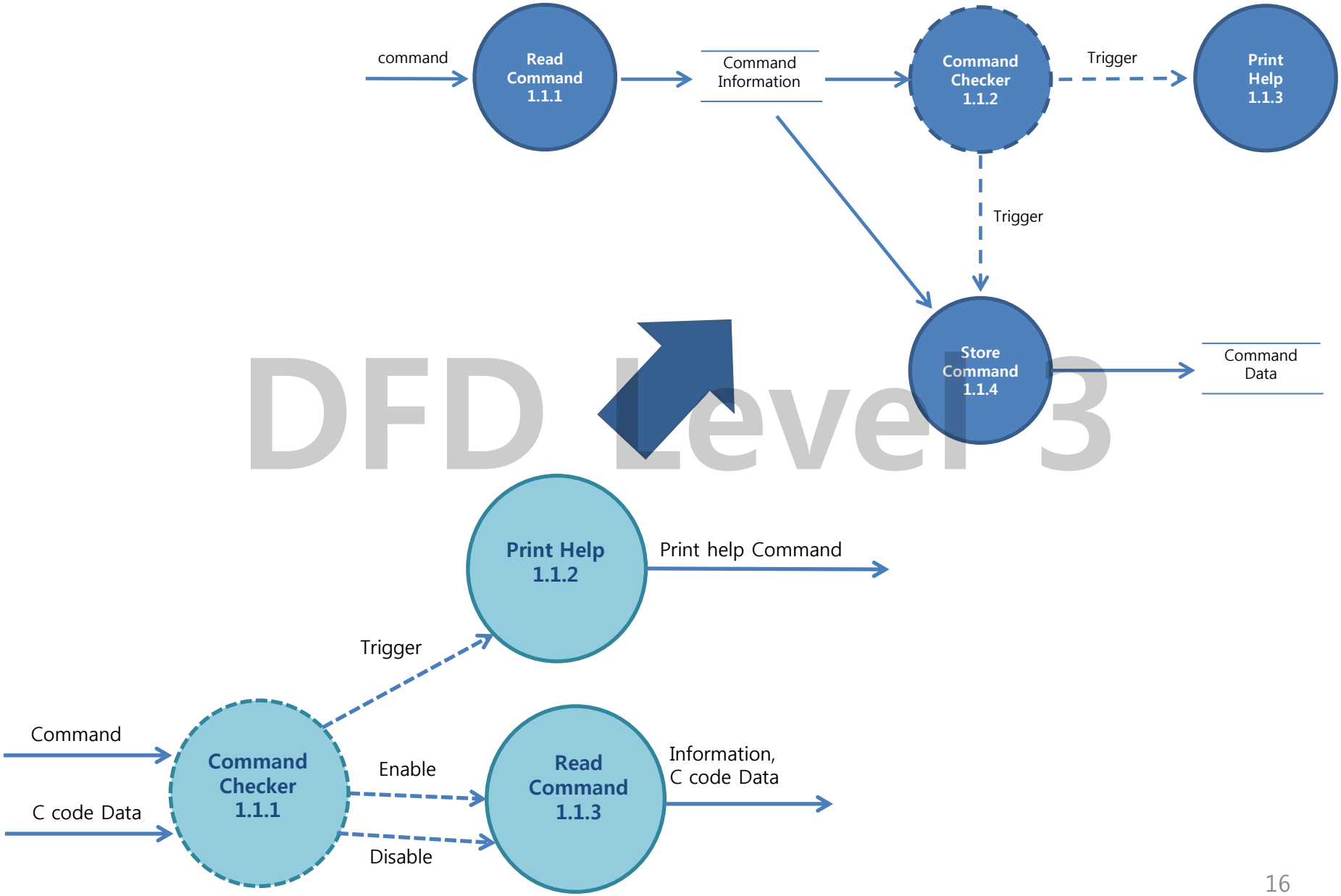
DFD Level 2



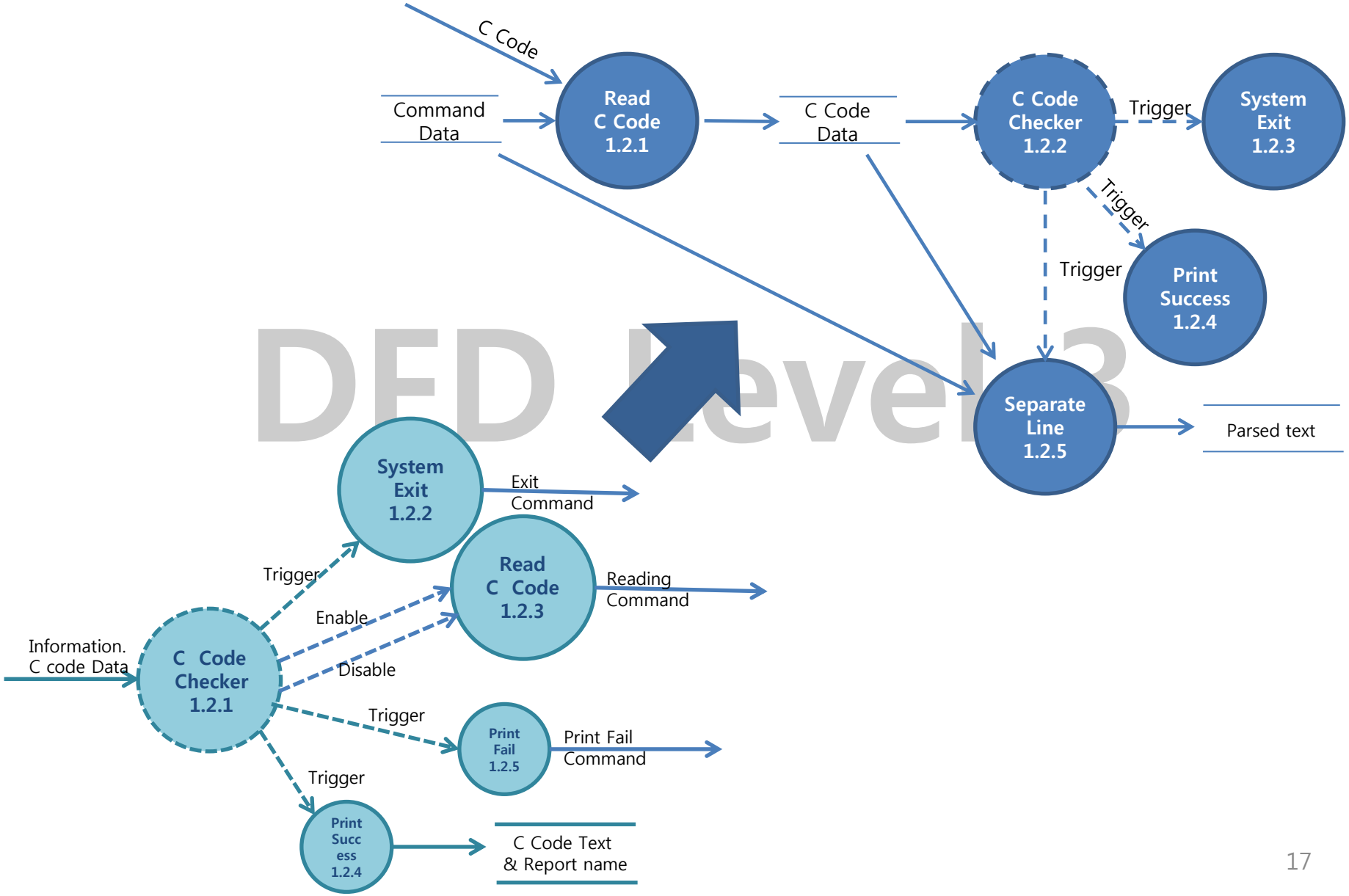
DFD Level 2


parsed text
Make Report 2.14
Design
Push C.S Information 2.3
CS List
Pop C.S Information 2..4
Read Next Line 2.2
Add function Information 2.5
Processor 2.1
CS information
line
Function Out 2.6
Flist
Make Condition Edge 2.12
Make Loop Edge 2.11
Function In 2.7
Parse 2.8
Make Node 2.9
Make Edge 2.10
Loop Edge
Normal Edge
Node
Design Interface 2.13
Design
DFD Level 2
Make Report 2.1.11
Push C.S Information 2.1.10
C Code Text & Report nam
Processor 2.1.1
Trigger
Function Out 2.1.9
Design Command
Function In 2.1.8
Make Edge 2.1.7
Make Loop Edge 2.1.6
Make Condition Edge 2.1.5
Make Node 2.1.4
Read Next Line 2.1.3
Write Content 2.1.2
Enable
Disable

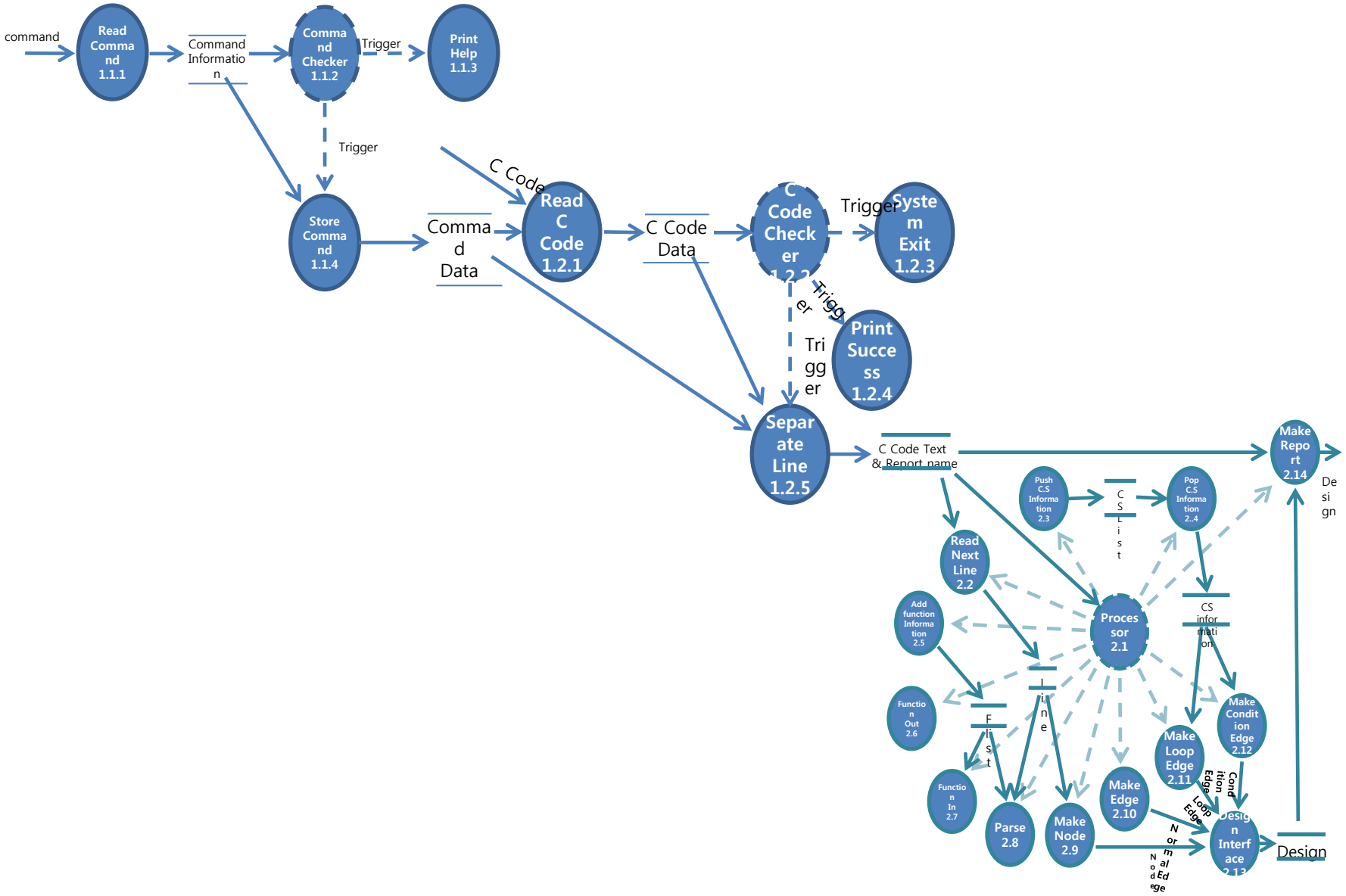
DFD Level 3


command
Read Command 1.1.1
Command Information
Command Checker 1.1.2
Trigger
Print Help 1.1.3
Trigger
Store Command 1.1.4
Command Data
DFD Level 3
Print Help 1.1.2
Print help Command
Trigger
Command
C code Data
Command Checker 1.1.1
Enable
Disable
Read Command 1.1.3
Information, C code Data

DFD Level 3

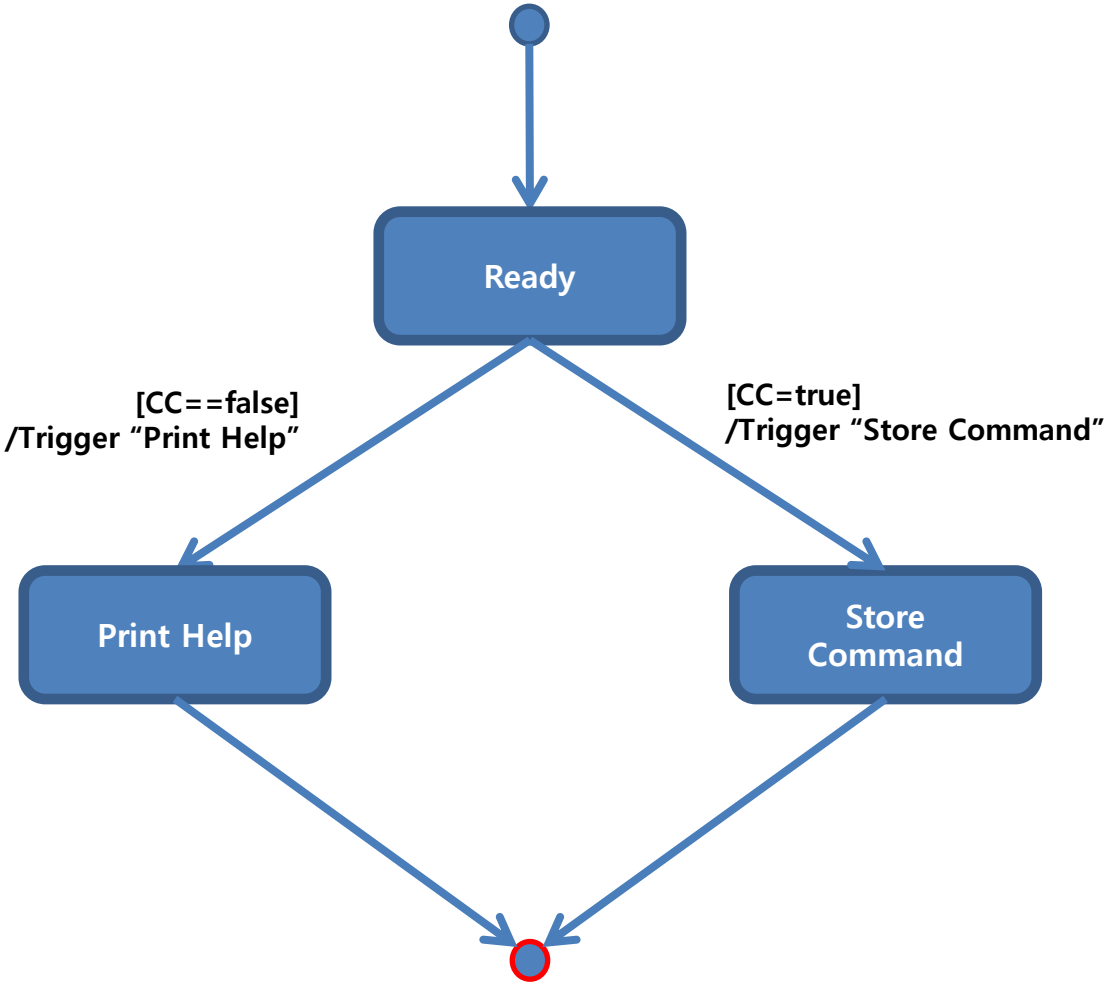

C Code
Command Data
Read C Code 1.2.1
C Code Data
C Code Checker 1.2.2
Trigger
System Exit 1.2.3
Trigger
Trigger
Print Success 1.2.4
Separate Line 1.2.5
Parsed text
DFD Level 3
System Exit 1.2.2
Exit Command
Trigger
Read C Code 1.2.3
Reading Command
Enable
Information. C code Data
C Code Checker 1.2.1
Disable
Trigger
Print Fail 1.2.5
Print Fail Command
Trigger
Print Succ ess 1.2.4
C Code Text & Report name

Data Flow Diagram



Final State Machine

Finite State Machine – 1.1.2 Command Checker



Notation Condition

State

Terminal State

Initial State

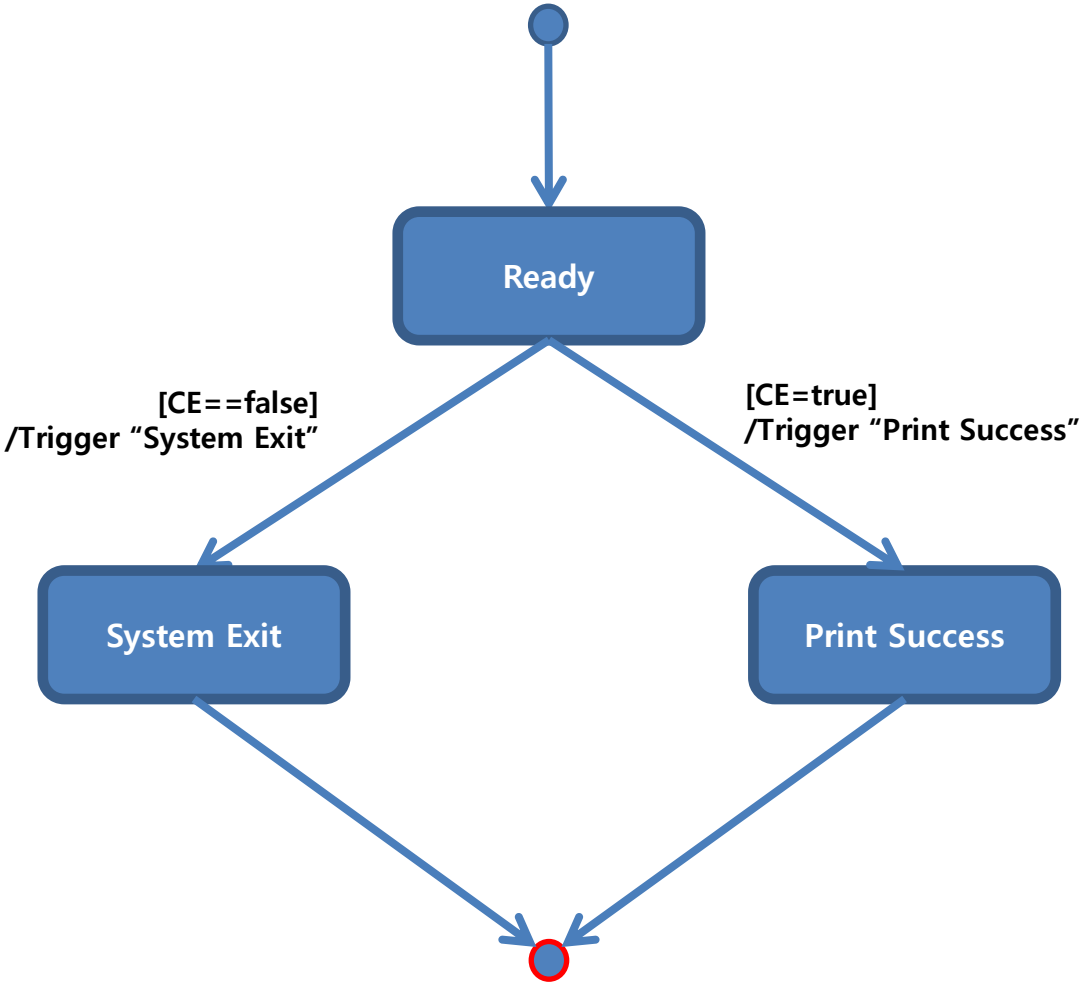
flow

CC : Command Correct

- Ture : valid command

- Flase : invalid command

Finite State Machine – 1.2.2 C Code Checker

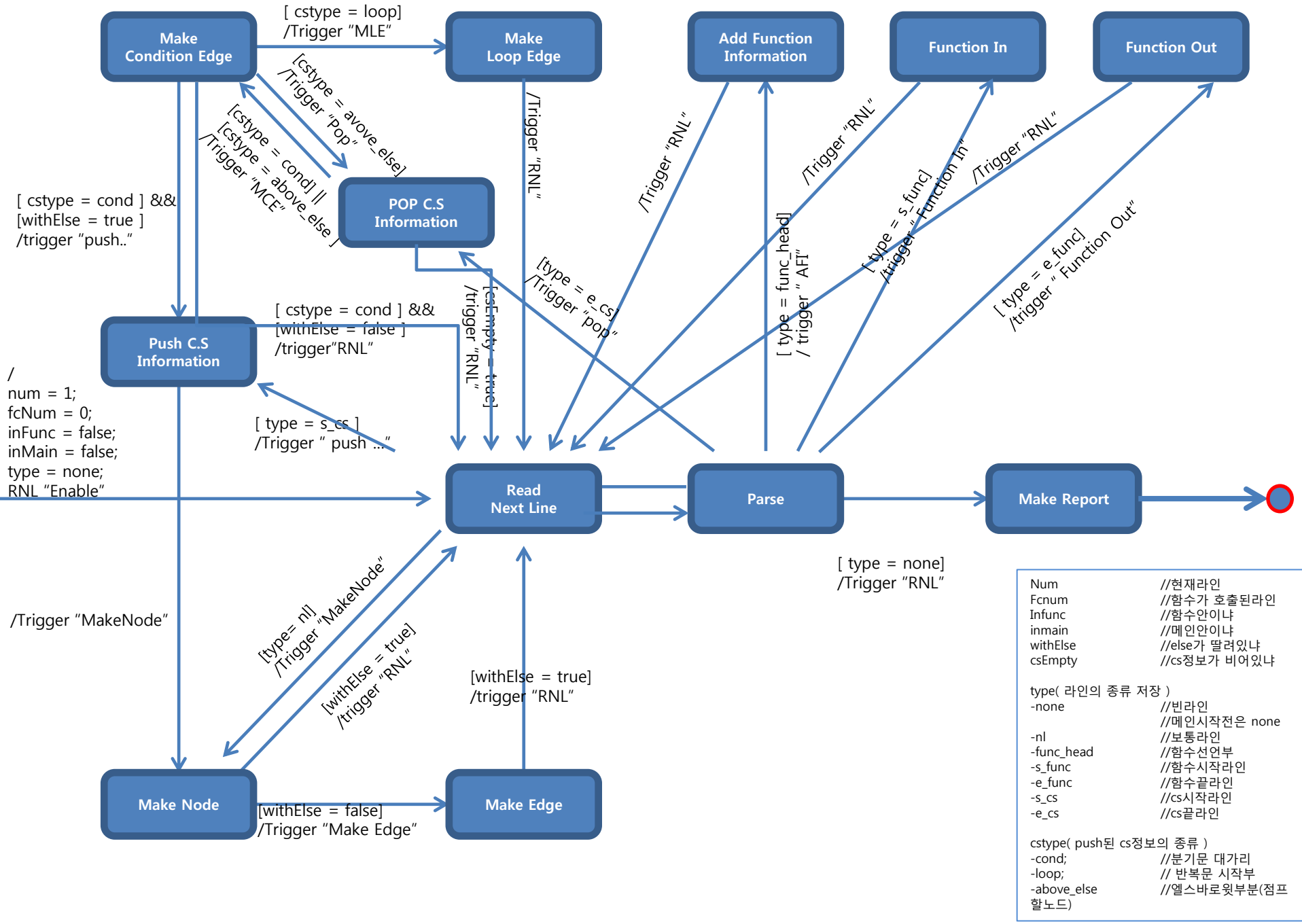


Condition

CE : C Code file Existence

- Ture : vailid C Code file
- Flase : invalid C Code file

Finite State Machine – 1.2.2 C Code Checker



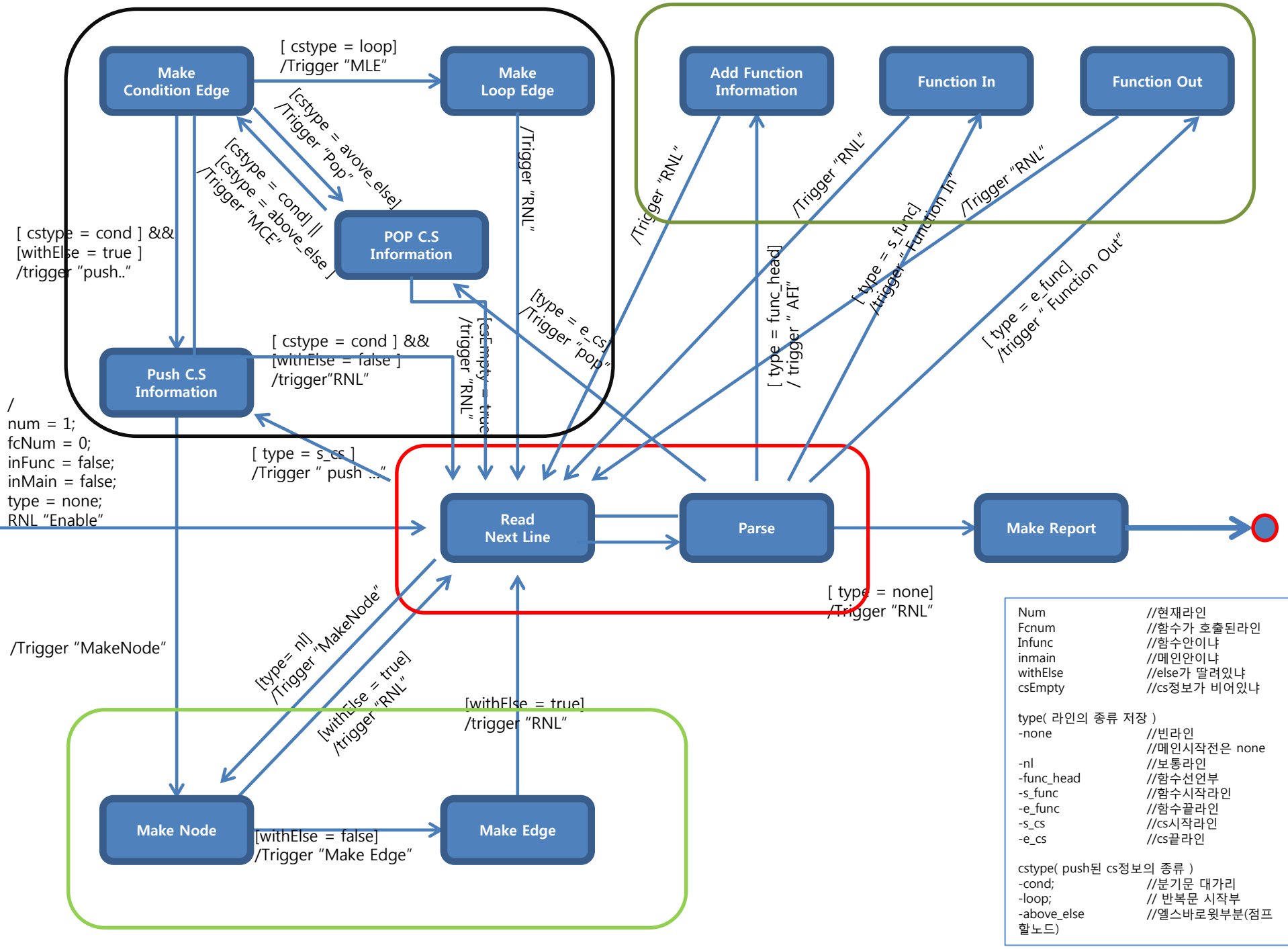
Num	//현재라인
Fcnum	//함수가 호출된라인
Infunc	//함수안이냐
inmain	//메인안이냐
withElse	//else가 딸려있냐
csEmpty	//cs정보가 비어있냐

type(라인의 종류 저장)

-none	//빈라인 //메인시작전은 none
-nl	//보통라인
-func_head	//함수선언부
-s_func	//함수시작라인
-e_func	//함수끝라인
-s_cs	//cs시작라인
-e_cs	//cs끝라인

cstype(push된 cs정보의 종류)

-cond;	//분기문 대가리
-loop;	// 반복문 시작부
-above_else	//엘스바로윗부분(점프할노드)



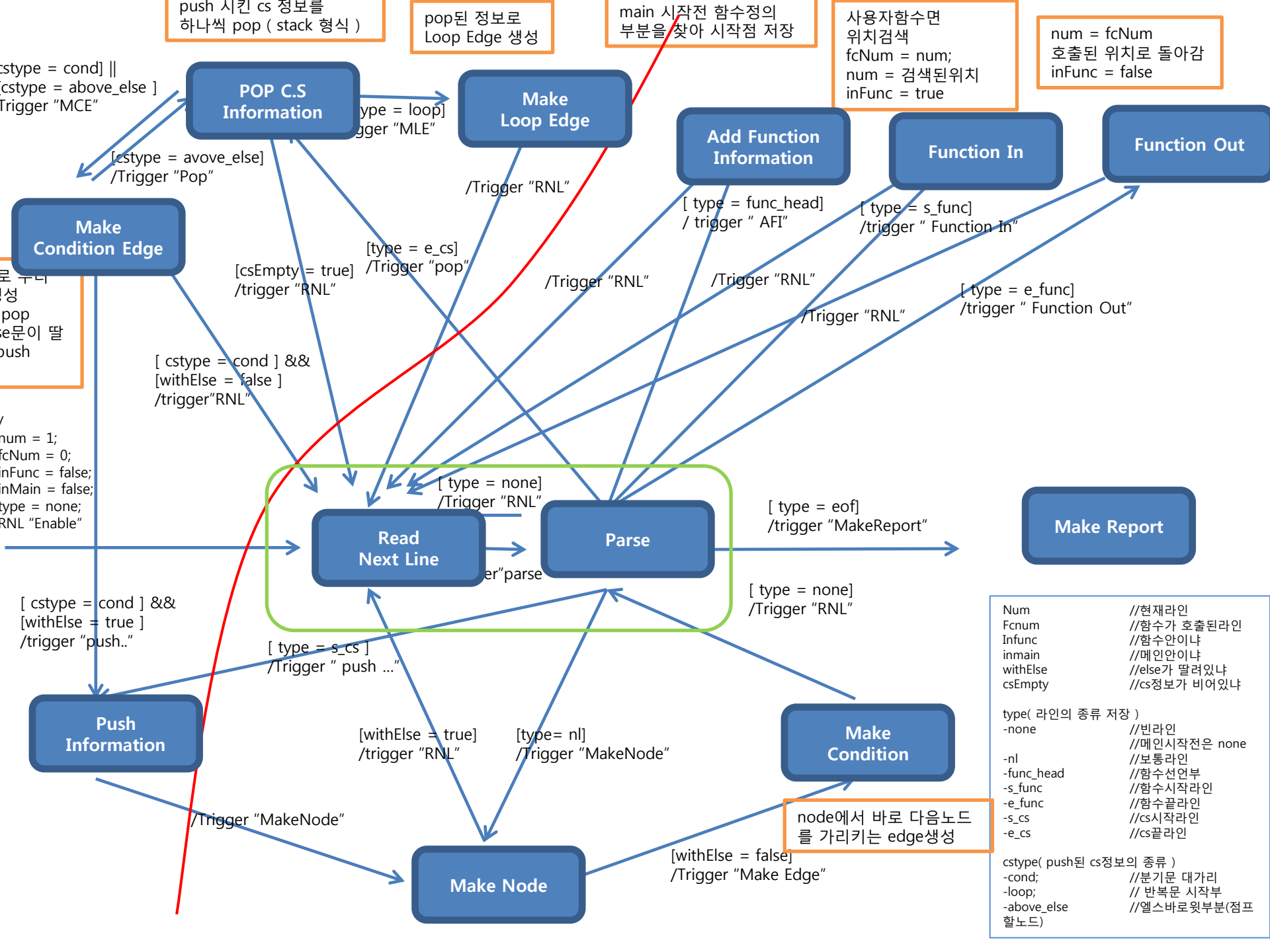
Num	//현재라인
Fcnum	//함수가 호출된라인
Infunc	//함수안이냐
inmain	//메인안이냐
withElse	//else가 딸려있냐
csEmpty	//cs정보가 비어있냐

type(라인의 종류 저장)

-none	//빈라인 //메인시작전은 none
-nl	//보통라인
-func_head	//함수선언부
-s_func	//함수시작라인
-e_func	//함수끝라인
-s_cs	//cs시작라인
-e_cs	//cs끝라인

cstype(push된 cs정보의 종류)

-cond;	//분기문 대가리
-loop;	// 반복문 시작부
-above_else	//엘스바로윗부분(점프할노드)


push 시킨 cs 정보를
하나씩 pop (stack 형식)
pop된 정보로
Loop Edge 생성
main 시작전 함수정의
부분을 찾아 시작점 저장
사용자함수면
위치검색
fcNum = num;
num = 검색된위치
inFunc = true
num = fcNum
호출된 위치로 돌아감
inFunc = false
cstype = cond] ||
[cstype = above_else]
Trigger "MCE"
POP C.S
Information
ype = loop]
gger "MLE"
Make
Loop Edge
Add Function
Information
Function In
Function Out
[cstype = avove_else]
/Trigger "Pop"
/Trigger "RNL"
[type = func_head]
/ trigger " AFI"
[type = s_func]
/trigger " Function In"
Make
Condition Edge
[csEmpty = true]
/trigger "RNL"
[type = e_cs]
/Trigger "pop"
/Trigger "RNL"
/Trigger "RNL"
[type = e_func]
/trigger " Function Out"
/Trigger "RNL"
[cstype = cond] &&
[withElse = false]
/trigger"RNL"
num = 1;
fcNum = 0;
inFunc = false;
inMain = false;
type = none;
RNL "Enable"
[type = none]
/Trigger "RNL"
Read
Next Line
Parse
[type = eof]
/trigger "MakeReport"
Make Report
er"parse
[type = none]
/Trigger "RNL"
[cstype = cond] &&
[withElse = true]
/trigger "push.."
[type = s_cs]
/Trigger " push ..."
Push
Information
[withElse = true]
/trigger "RNL"
[type= nl]
/Trigger "MakeNode"
Make
Condition
node에서 바로 다음노드
를 가리키는 edge생성
/Trigger "MakeNode"
Make Node
[withElse = false]
/Trigger "Make Edge"
Num //현재라인
Fcnum //함수가 호출된라인
Infunc //함수안이냐
inmain //메인안이냐
withElse //else가 딸려있냐
csEmpty //cs정보가 비어있냐
type(라인의 종류 저장)
-none //빈라인
//메인시작전은 none
-nl //보통라인
-func_head //함수선언부
-s_func //함수시작라인
-e_func //함수끝라인
-s_cs //cs시작라인
-e_cs //cs끝라인
cstype(push된 cs정보의 종류)
-cond; //분기문 대가리
-loop; // 반복문 시작부
-above_else //엘스바로윗부분(점프
할노드)

Data Dictionary

Data Dictionary Level 0

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 EX) struct Design{ int nodesize; int edgesize; Node* node[DESIGN_SIZE]; Edge* edge[DESIGN_SIZE*2];}	struct

Data Dictionary Level 1

Input/Output Event	Description	Format / Type

Parsed Text & Report Name	Parsed Text & Repot Name has it's two data exist in struct form
	1.Parsed Text is data that parsed by [Inspect process]. This data send [Convert process].
	2.Report Name is Report file's name that show list of 'states' and 'edges' of CFG
		struct{
		char*,
		file*
		}

Data Dictionary Level 2

Input/Output Event	Description	Format/Type
Command Data	Save command Data if the input command is right, [Command check] will analyze the command and separate input file name and output file name save it into struct form	Struct{ char*, char*}

Data Dictionary Level 3

Input/Output Event	Description	Format/Type

Command Information	Command Information has it's three data exist in struct form
	1. Data that read command and save it after checking if it's right
	If command is right than data has True, else data has False
	2. Data that save Input File Name and Output File Name received from command in char* form
		Struct{
		int,
		char *,
		char *
		}

Data Dictionary Level 3

Input/Output Event	Description	Format/Type

C code data	C code data has it's two data exist in struct form
	1.Data that read Input File Name and save it after checking if C code's exist
	If C code is exist than data has True, else data has False
	2.Data Save C code received from [Read C code] in char* form
		struct{
		int,
		char*
		}

Data Dictionary Level 3

Input/Output Event	Description	Format/Type
line	one line belong to source code The standard of one line is newline character	char*

Data Dictionary Level 3

Input/Output Event	Description	Format/Type

Node	Node has it's three data exist in struct form
	
	Data that saves Node information(id, content, type) sent by [Make Node]
	1. Id is Node's number.
	2. Content is Node's content.
	3. Type is Node's type.
		struct Node{
		int id = 0;
		char *content = "node"
		enum type = type;
		}

Data Dictionary Level 3

Input/Output Event	Description	Format/Type

Edge	Edge has it's four data exist in struct form
	Data that saves Edge information(type, id, pid, cid) sent by [Make Edge]
	
	1. Type is Edge's type.
	2. Id is Edge's number.
	3. Pid is parent Node's number.
	4. Cid is child Node's number.
		struct Edge{
		enum type;
		int id;
		int pid;
		int cid;
		}

Data Dictionary Level 3

Input/Output Event	Description	Format/Type

Loop Edge	Loop Edge has it's four data exist in struct form
	Data that saves Edge information(type, id, pid, cid) sent by [Make Loop Edge]
	
	1. Type is Edge's type.
	2. Id is Edge's number.
	3. Pid is parent Node's number.
	4. Cid is child Node's number.
		struct Edge{
		enum type;
		int id;
		int pid;
		int cid;
		}

Data Dictionary Level 3

Input/Output Event	Description	Format/Type

Condition Edge	Condition Edge has it's four data exist in struct form
	Data that saves Edge information(type, id, pid, cid) sent by [Make Condition Edge]
	
	1. Type is Edge's type.
	2. Id is Edge's number.
	3. Pid is parent Node's number.
	4. Cid is child Node's number.
		struct Edge{
		enum type;
		int id;
		int pid;
		int cid;
		}

Data Dictionary Level 3

Input/Output Event	Description	Format/Type

Flist	Flist has it's four data exist in struct form
	Data that saves user defined function information sent by [Add Function Information]
	
	1.function[ ] is data that save function defined part.
	2.line[ ] is data that save function defined part's line.
	3.size is data that save list's size
		struct funcLIst{
		char *function[ ];
		int line[ ];
		int size;
		}

Data Dictionary Level 3

Input/Output Event	Description	Format/Type

CS List	CS List has it's two data exist in struct form
	Data that saves condition sentence's information received from [push cs Information]
	
	1. Data that saves Pushed Control Sentence's list
	2.Data that saves the size of control sentence's list size
		struct cs{
		cs* list[ ];
		int size;
		}

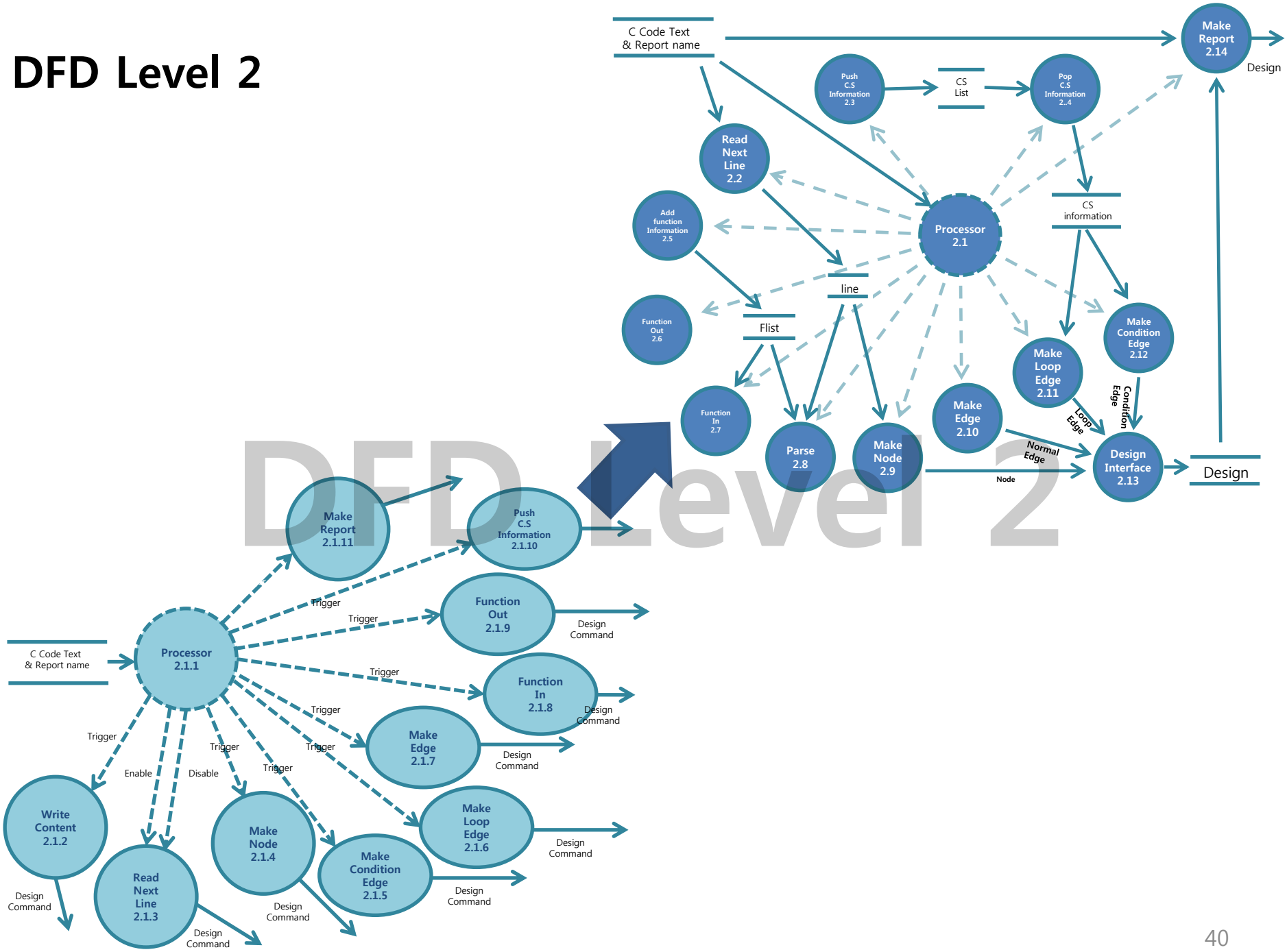
Data Dictionary Level 2

Input/Output Event	Description	Format/Type

CS	CS has it's two data exist in struct form
	Data that saves Control Sentence's Information received from [Pop CS Information]
	
	1.Data that saves pop Control Sentence Information
	2.Data that saves control sentence's line
		struct cs{
		enum cstype;
		int nodenum;
		}

Process Specification

DFD Level 2


C Code Text & Report name
Make Report 2.14
Design
Push C.S Information 2.3
CS List
Pop C.S Information 2..4
Read Next Line 2.2
Add function Information 2.5
Processor 2.1
CS information
line
Function Out 2.6
Flist
Make Condition Edge 2.12
Make Loop Edge 2.11
Function In 2.7
Make Edge 2.10
Condition Edge
Loop Edge
Normal Edge
Parse 2.8
Make Node 2.9
Node
Design Interface 2.13
Design
DFD Level 2
Make Report 2.1.11
Push C.S Information 2.1.10
Trigger
Function Out 2.1.9
Design Command
C Code Text & Report name
Processor 2.1.1
Function In 2.1.8
Make Edge 2.1.7
Enable
Disable
Write Content 2.1.2
Make Node 2.1.4
Make Loop Edge 2.1.6
Make Condition Edge 2.1.5
Read Next Line 2.1.3

Process Specification Level 2

Reference	2.2
Name	Read Next Line
Input	C Code text, num / trigger
Output	String
Process Description	this process return next line, after it read c source code and current line number. the returned string value is a line of c source code that is written until next line character like ' ₩n. ' . the string is returned with ' char* ' type. [#1 move value 'pos' , file pointer, to the line processed #2 f gets(char*) #3 return char*]

Process Specification Level 2

Reference	2.3
Name	push CS information
input	num(int), cstype(enum)/ trigger
output	push cs to cslist

Process Description	when the type is start cs, thie process is called. if parse result is cs(control sentence) , the type of ahead cs and line information is stored with struct. when cs end , which will be using to make condition edge and loop.
	
	cstype(pushed cs information)
	-cond; //head of branch
	-loop; // head of loop
	-above_else //above of else
	ex) test <<
	}else
	
	struct cs{
	enum cstype;
	int nodenum;
	}

Process Specification Level 2

Reference	2.4
Name	pop CS information
input	cslist(array of cs pointer) / trigger
output	CS
Process Description	Like Stack, Data structure, the most rent cs stored is pop. Base on cs information, 'loop edge' or 'condition edge' are generated

Process Specification Level 2

Reference	2.5
Name	Add Function Information
input	line(char*), num(int) / trigger
output	function head(char*)

Process Description	before entering the 'Main function' , this process store user defined function .
	
	ex)if a head of line have structure of function definition add this function information at list.
	
	struct funcLIst{
	char *function[]; // text of function defined
	int line[]; // defined function line
	int size; // list size
	}

Process Specification Level 2

Reference	2.6
Name	Function Out
input	num(int), fcnum(int)
output	-
Process Description	If there is at the end of function with fcnum(the function called), this process return the position of the function called . ex) num = fcnum

Process Specification Level 2

Reference	2.7
Name	Function In
input	num(int), fcnum(int)
output	-
Process Description	If a read line is User defined function , store line number and change the current line number with function defined number. In order to come back, store the line that the function called.

Process Specification Level 2

Reference	2.8
Name	Parse
input	line(char*), Flist(struct) / trigger
output	type(enum)
Process Description	Decide type of current line. That is operated by [ReadNextLine] with parsed current line Type (of line) -none // empty line . it is none that above of main -nl // normal code line -func_head //portion of call start. -s_func // function started line -e_func // function ended line -c_func // function called line -s_cs //cs start line -e_cs //cs end line

Process Specification Level 2

Reference	2.9
Name	Make Node
input	string line(char*), num(int), type(enum) / trigger
output	Node(struct *)

Process Description	Make Node with 'line', 'num' aand 'type'.
	'line' is stored in Node's content and num is assigned to size of N ode list .
	Because Node is made of each line, 'num' is set to id of Node.
	type of current line is assigned to type in Node.
	at the end of this process , Node Pointer is stored in
	
	ex)
	struct Node{
	int id = 0;
	char *content = “node”
	enum type = type;
	}

Process Specification Level 2

Reference	2.10
Name	Make Edge
input	parentID(int) / trigger
output	Edge*

Process Description	After parentID is inserted, next parentID will be set chlidID of it's child to using to make edge. because each line is a edge , next no de will be child of previous Node. the child ID is assigned with siz e of edge list.
	
	ex)
	struct Edge{
	enum type;
	int id;
	int pid;
	int cid;
	}

Process Specification Level 2

Reference	2.11
Name	Make Loop Edge
input	cs(struct), num(int)
output	Edge(struct Node*)

Process Description	Generate make edge with pop cs information . Parent is set Current node and child is set pop node. which are using to make loop edge (back edge)
	ex)
	struct Edge{
	enum type;
	int id;
	int pid;
	int cid;
	}

Process Specification Level 2

Reference	2.12
Name	Make Condition Edge
input	cs(struct), num(int)
output	Edge(struct Node*)

Process Description	The data made up pop cs information is used to make condition edge. Current Node is set to child. Also pop Node is set parent . And then this process generate condition edge, which made of parent and child node number.
	
	ex)
	struct Edge{
	enum type;
	int id;
	int pid;
	int cid;
	}
	*참고
	else문 이면 다시 pop
	condition인데 else문이 딸려있으면 else문 push

Process Specification Level 2

Reference	2.13
Name	Design Interface
input	Edge(struct Edge*), Node(struct Node*)
output	Design

Process Description	Add Edge and Node to designed which include each list . Also this process restrict number of node and edge . (node / 100) , (edge / 200)
	struct Design{
	int nodesize;
	int edgesize;
	Node* node[DESIGN_SIZE];
	Edge* edge[DESIGN_SIZE*2];
	}

Process Specification Level 2

Reference	2.14
Name	Make Report
input	Design, Reprot Name(char*) / trigger
output	CFG
Process Description	print cfg generated by this process to console and file name with ‘report name’ .with information of node and edge in ‘design’ .

Process Specification Level 3

Reference	1.1.1
Name	Read Command
Input	Command
Output	Command Information
Process Description	This process get a Command from CUI and store struct with command information.

Process Specification Level 3

Reference	1.1.2
Name	Command Checker
Input	Command Information
Output	Trigger
Process Description	This process check validity of command with command information. it is error that command don't follow fixed format. when command information have a error, this process call 'Print Help' process with trigger and command information. there is no error, this process call 'Store Command' with trigger and command information.

Process Specification Level 3

Reference	1.1.3
Name	Print Help
Input	Trigger
Output	-
Process Description	The case that command have error occurs this process that print ' help message ' to console with error point. after print ' help message ', this process occur absorbing this process.

Process Specification Level 3

Reference	1.1.4
Name	Store Command
Input	Command Information, Trigger
Output	Command Data
Process Description	this process convert 'Command Information' to 'Command Data'. boolean data on Command Data is dropped.

Process Specification Level 3

Reference	1.2.1
Name	Raed C Code
Input	C Code, Command Data
Output	C Code Data
Process Description	this process read c code from c code file. also this process take command data to open c code file. when the file is opened successfully, this process store static value to 1, meaning "true“ and store c code to the other value in struct. when this process can't open the c code file, it store the value to 0, meaning "false".

Process Specification Level 3

Reference	1.2.2
Name	C Code Checker
Input	C Code Data
Output	Trigger
Process Description	this process is given command data and c code data structed. when c code data have a error sign, this process call "system exit process". in the other case, the process call "print success".

Process Specification Level 3

Reference	1.2.3
Name	System Exit
Input	Trigger
Output	-
Process Description	when this process is called, the case would be ' c code file ' had not opened. after this process is called, like written name .

Process Specification Level 3

Reference	1.2.4
Name	Print Success
Input	Trigger
Output	-
Process Description	After command check phase and c code check phase are passed, [C Code Checker] call this process, [Print success]. Also this process integrate 'Command' and ' 'C Code Data '. the operation does not simply integrate the data, but it drop the ' c code file name ' that is not needed in [Convert Process]. the integrated data is stored in

Process Specification Level 3

Reference	1.2.5
Name	Separate Line
Input	Trigger, Command Data, C Code Data
Output	C Code text & Report Name
Process Description	separate the command data to command type and c code data name end generate the c code text from the exist file which name is that the separated form the user input command report name too. make a file to write parse text that made by this process this parsing means to make c code more useful

Structured Design
- Structured Charts

Notation

Modules

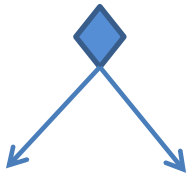
Module call

Library Modules

Data Flow

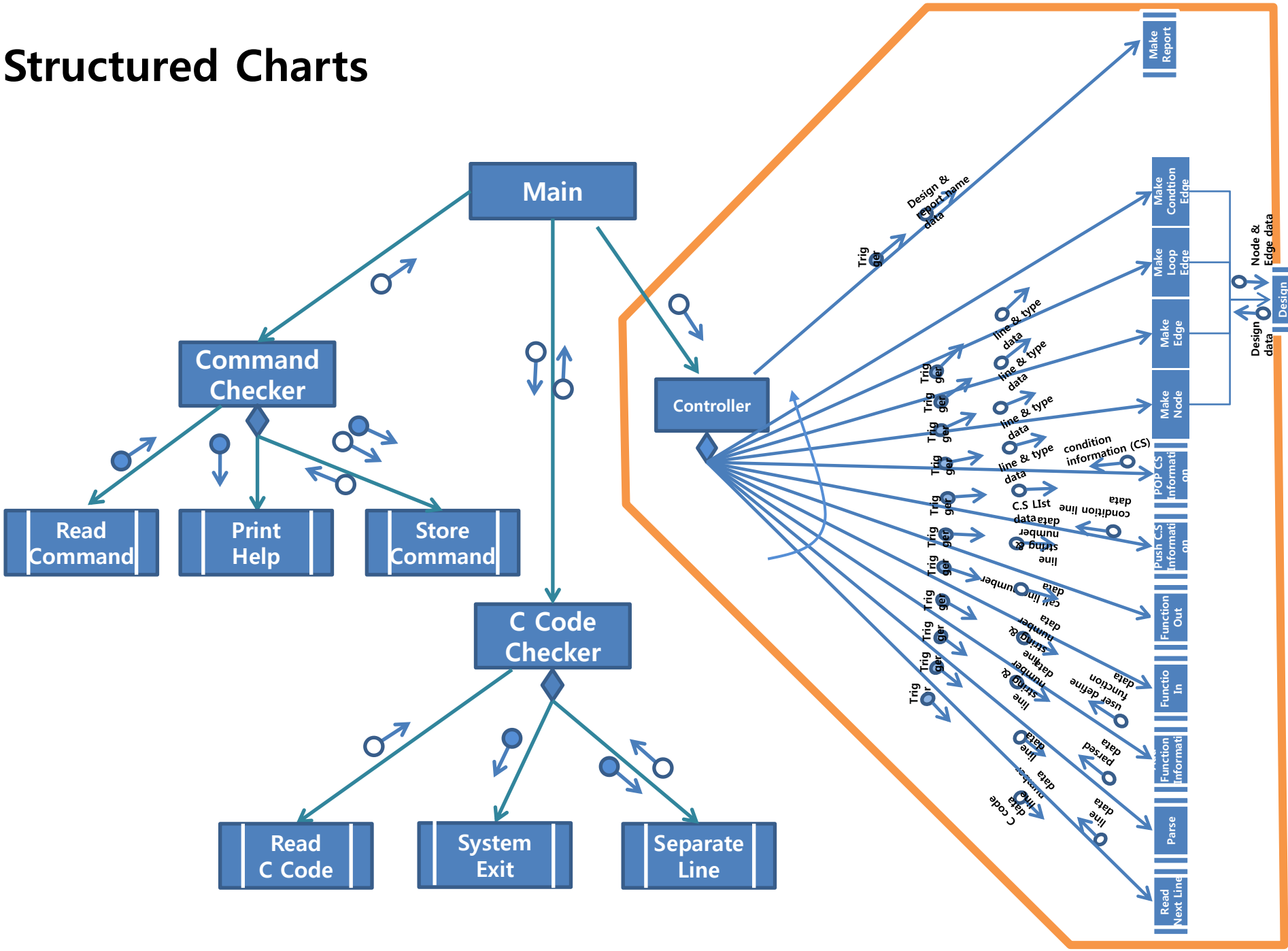
Control Flow

Data module

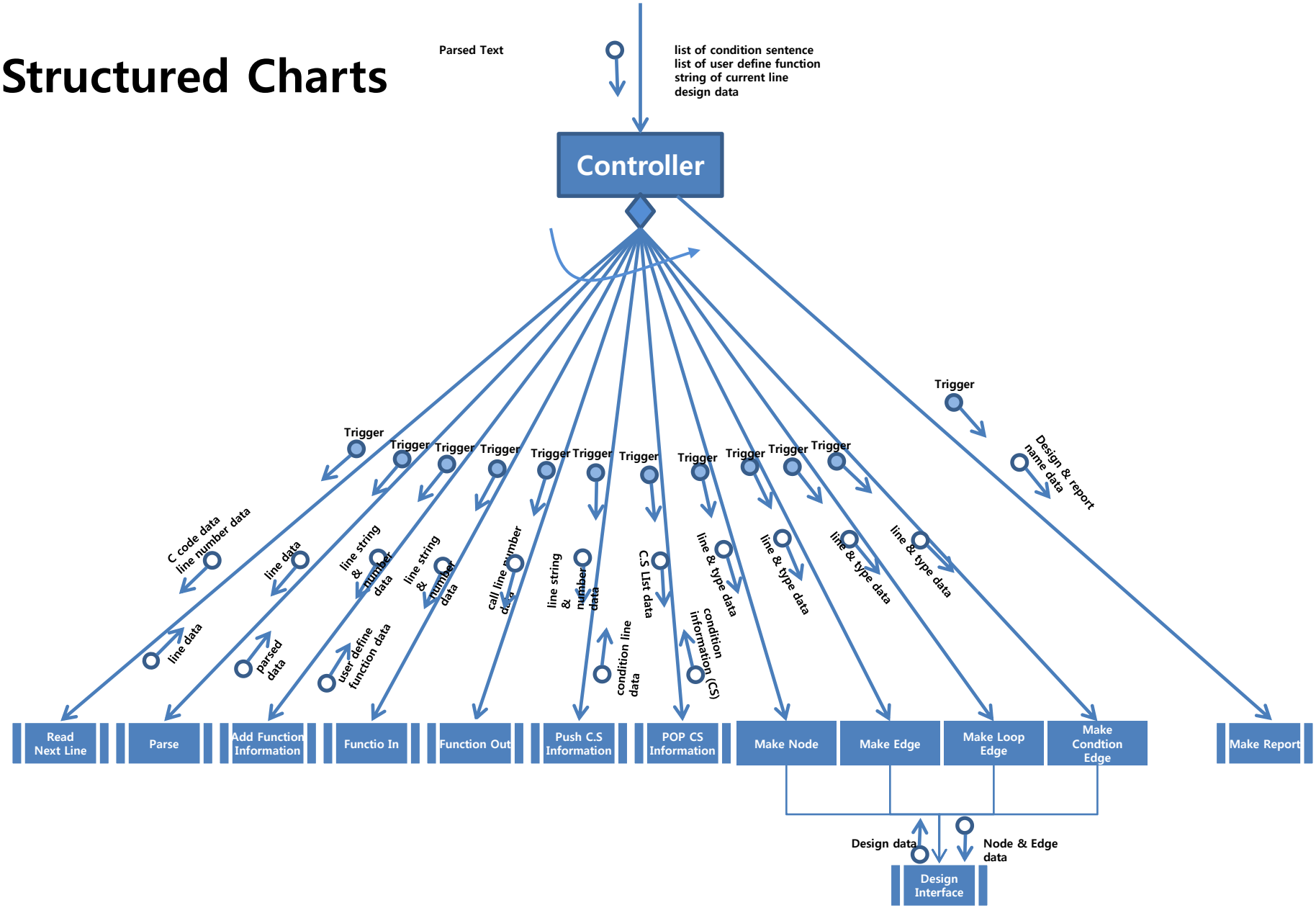


Decision

Structured Charts



Structured Charts



Thank Yoo and us