Introduction to Software Engineering

System Anaylsis & System Design

CFG Generator

TEAM 1

201011309 권선일, 201011336 백인선

201011357 이주희, 201011342 안혜수

Statement of Purpose

1. While the CFG is being created, show the start Message to the consol window.
2. If CFG has been made, only show the Success Message and the CFG will be saved at File.
3. If making CFG has been failed , show the Error Message and end the all the Process.
4. We only handle the C source codes.(*.c)
5. We make only the Main() Function of the source to CFG.
6. If there is no Block such as '{}', and also not the proper path of the File, we cannot make the CFG. So the CFG Generator will be conclude as an error and ends the program.

Statement of Purpose

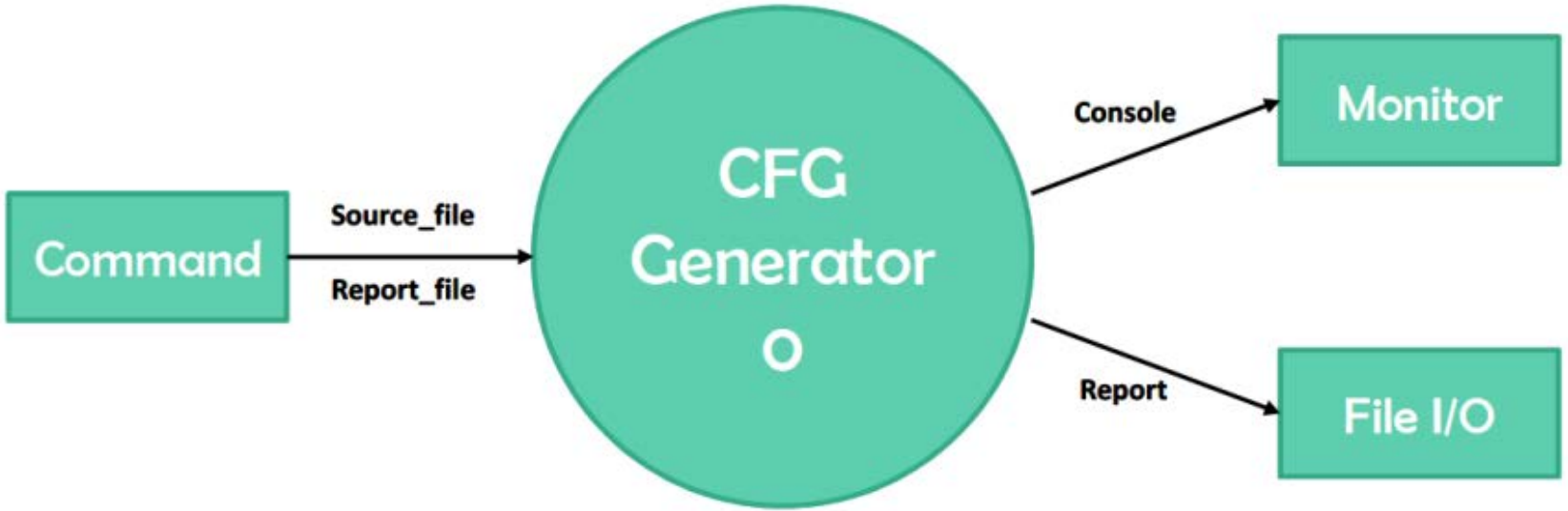
7. We would not consider all the Compile error, only will con sider the Error that effects making the CFG.(Such as missin g of the '{}' Block, or absence of the Main() Function)
8. If the CFG has been made, we will save the CFG's BB List and Edge List to each line to the File.
9. The form of saving CFG is, as BB List we will save '[{ (Source), [(In-Edge)], [(Out-Edge)] }, ..., {...}]]'. And as for Edge List, '[{ (Start BB), (End BB) }, ..., {...}]'.
10. Help Message will be printed when the path of the File is i ncorrect..
    - The form of the Help Message.

      $./cg <source code filepa(*.c)> <report filepath(*.txt)>

Statement of Purpose

- Statement indicates prior semicolon(;) to nex t semicolon(;).
- Basic Block is a basic unit compose CFG. The inside of the BB can be the CFG itself and ca n be s single Statement.
- CFG(Control Flow Graph) is a Graph that co nnects Basic Blocks by Edges. This CFG can have another CFG in the BB.

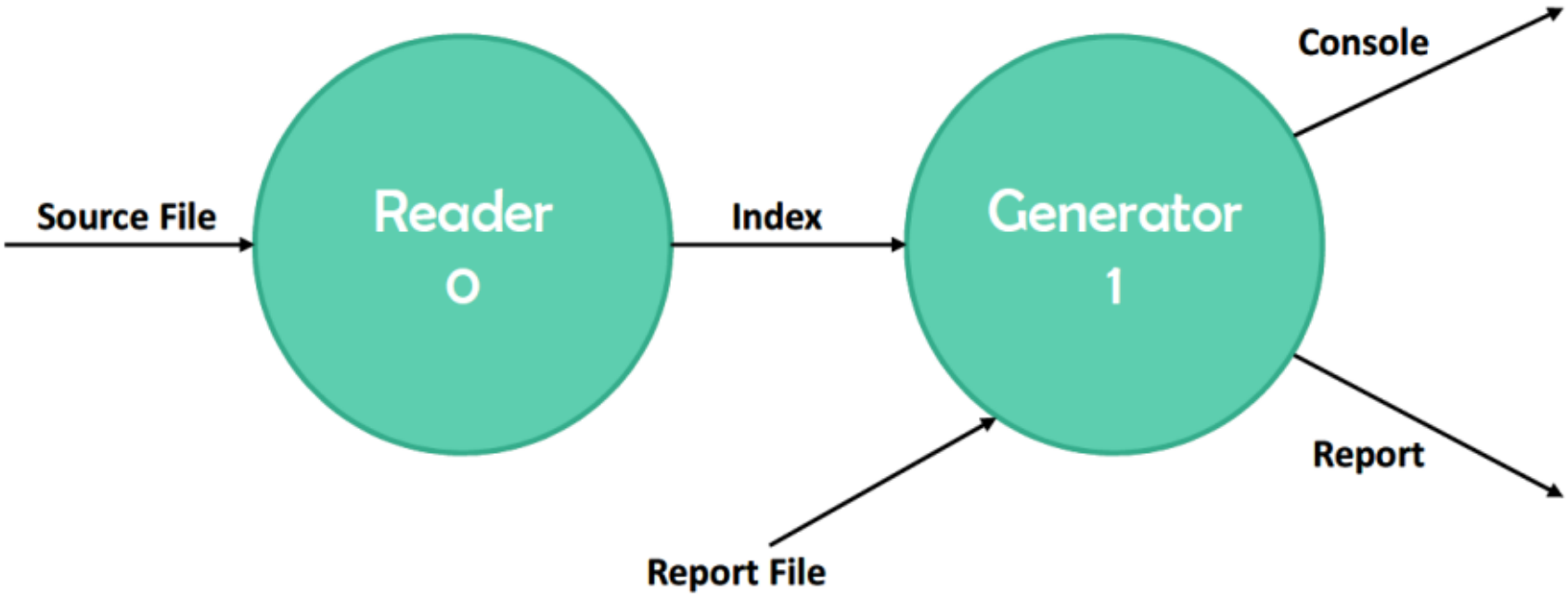
System Context Diagram



Event List

Input/Output Event	Description	Format/Type
Source_File	The path of the File of C source code	String(*.c)
Report_File	The path of the File which contains the completed CFG	String(*.txt)
Console	Success/Start/Error Message	The String that prints out the console
Report	The string that prints out the Basic Blocks and Edges	The String that prints into File

DFD – Level 0


Source File
Reader
0
Index
Generator
1
Console
Report
Report File

Level 0 - Data Dictionary

Input/Output Event	Description	Format/Type
Index	It makes easier to read the **C Source Code** from the **Generator**. **Index** has **typeNum**, **source**, **BlockNum**, **In/Out Edge Num**.	The Type and Source are a form of struct (1) TypeNum 1-1-1 if : type = 1.11 1-1-2 else if : type = 1.12 1-1-3 else : type = 1.13 1-2 switch : type = 1.2 1-3 for : type = 1.3 1-4 while : type = 1.4 1-5 일반statement = type = 1.0 (2) Source : the source of the index (3) BlockNum : the Block Number of the index (4) In/OutEdgeNum : the Block In/Out Edge Number of the index

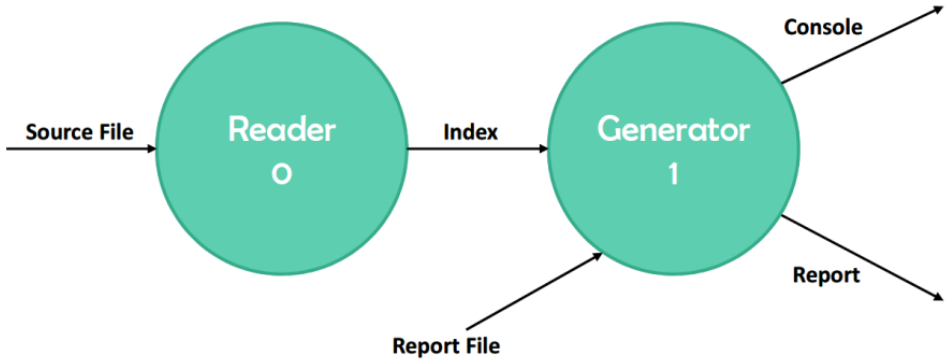
Level 0 - Specification

Name	Reader
Reference Number	0
Input	Source_File
Output	Index(+Data Structure)
Description	It receives the Source File and after the Parsing, returns to Index. If the path of the Source File is wrong(or does not exist or not the form of *.c), put NULL to Index.

Level 0 - Specification

Name	Generator
Reference Number	1
Input	Index(+Data Structure)
Output	Message, Report
Description	It receives the Index then makes the Basic Blocks. And it makes CFG then writes to the Report_File. Also it could prints out Console Message(Start, Success, Fail).

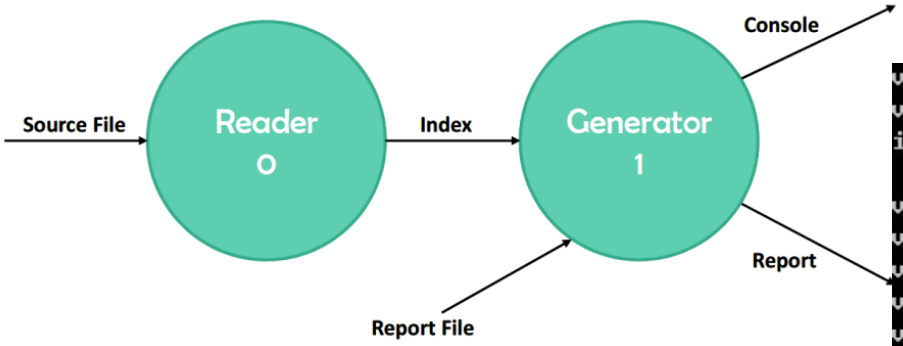
Reader Code



Name	Reader
Reference Number	0
Input	Source_File
Output	Index(+Data Structure)
Description	It receives the Source File and after the Parsing, returns to Index. If the path of the Source File is wrong(or does not exist or not the form of *.c), put NULL to Index.

```
char preProcessed[BUFSIZE]={0,};
static int blockNumCount=1;

Reader* create_Reader(char* i_sourceFile);
void destroy_Reader(Reader* i_Reader);
void clear_index(IndexList* i_list);
void destroy_indexList(IndexList* i_list);
Index* create_Index(char* i_code, int i_blockNum, Type i_typeNum);
void insert_Index(IndexList* i_list, Index* i_node);
IndexList* create_indexList();

IndexList* Reader_play(Reader* i_reader);
char* pre_Processor(char* i_file);
int _check_comment(char* i_line);
TokenList* tokenizer(char* i_preProcessed);

Type _type_checker(char* i_str);
int _check_declaration(char* i_line);

TokenList* create_TokenList();
void clear_token(TokenList* i_list);
void destroy_TokenList(TokenList* i_list);
void insert_token(TokenList* i_list, char* i_token);

char* tokenMessenger(TokenList* i_list, int back);
IndexList* indexer(TokenList* i_pTokenList);
IndexList* indexeController(TokenList* i_tokenList, IndexList* i_indexList);

char* _wrap_str(TokenList* i_tokenList, char* i_init);
char* _wrap_str2(IndexList* i_indexList, char* i_init);
char* _wrap_str3(TokenList* i_tokenList, char* i_init);
```

Generator Code




```
void generator_play(IndexList* i_indexList);
void mainGenerator(IndexList* i_iList, int i_valid);
int validator(IndexList* i_indexList);

void blockCreator(Index* i_index);
void _generalCreator(Index* i_in);
void _ifCreator(Index* i_in);
void _switchCreator(Index* i_in);
void _forCreator(Index* i_in);
void _whileCreator(Index* i_in);
Index* nextIndexNode(IndexList* i_iList, Index* startNode);
void _inedge_display(Index* i_index);
void _outedge_display(Index* i_index);
void _branch_display(Index* ofIn);
void print(char* i_message);
char* _print_start();
char* _print_fail();
char* _print_success();
```

Name	Generator
Reference Number	1
Input	Index(+Data Structure)
Output	Message, Report
Description	It receives the Index then makes the Basic Blocks. And it makes CFG then writes to the Report File. Also it could prints out Console Message(Start, Success, Fail).

Index Struct

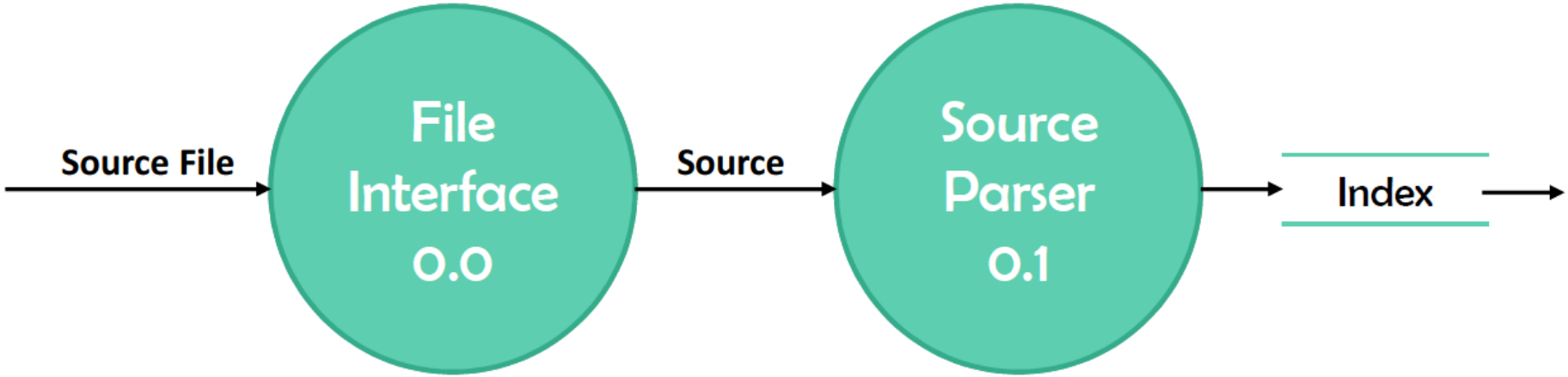
Input/Output Event	Description	Format/Type
Index	It makes easier to read the C Source Code from the Generator. Index has typeNum, source, BlockNum, In/Out Edge Num.	The Type and Source are a form of struct (1) TypeNum 1-1-1 if : type = 1.11 1-1-2 else if : type = 1.12 1-1-3 else : type = 1.13 1-2 switch : type = 1.2 1-3 for : type = 1.3 1-4 while : type = 1.4 1-5 일반statement = type = 1.0 (2) Source : the source of the index (3) BlockNum : the Block Number of the index (4) In/OutEdgeNum : the Block In/Out Number of the index

```
typedef enum STATE_Type {GENERAL, IF, ELSEIF, ELSE, SWITCH, FOR, WHILE}Type;
```

```
typedef struct _Index_type
{
        Type typeNum;
        int blockNum;
        char* source;
        struct _Index_type* outEdge;
        struct _Index_type* inEdge;
        struct _IndexList_type* branch_list;
}Index;
```

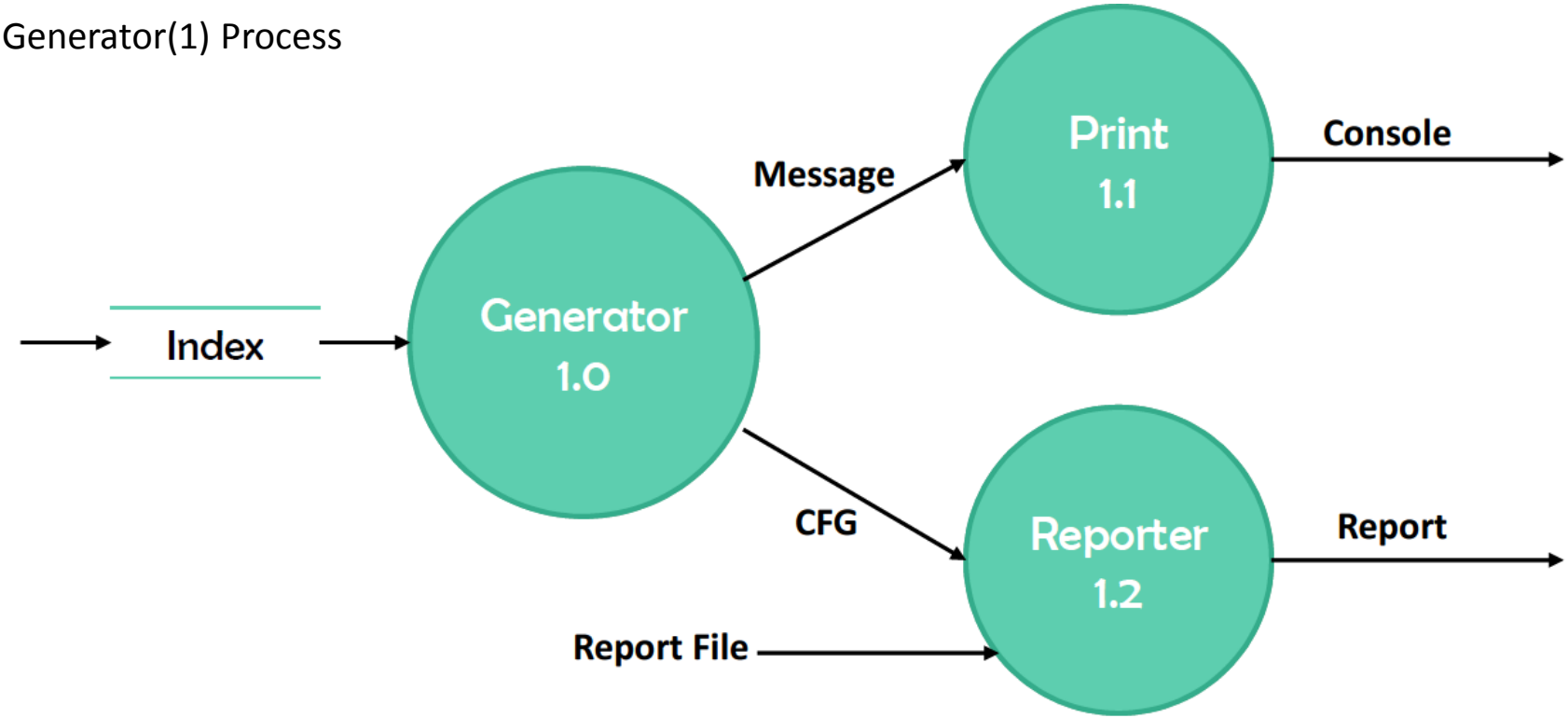
DFD – Level 1

- Reader Process



DFD – Level 1

- Generator(1) Process



Level 1 - Data Dictionary

Input/Output Event	Description	Format/Type
Source	Reads the Source_File and makes it as a string.	String
Message	The Message that prints out to Console.	String
CFG	The Graph that Generator made.	String(Graph)

Level 1 - Specification

Name	File Interface
Reference Number	0.0
Input	Source_File
Output	Source
Description	Reads all the Source_File and gives to Source, the string type. If the path is incorrect, puts NULL into Source.

Level 1 - Specification

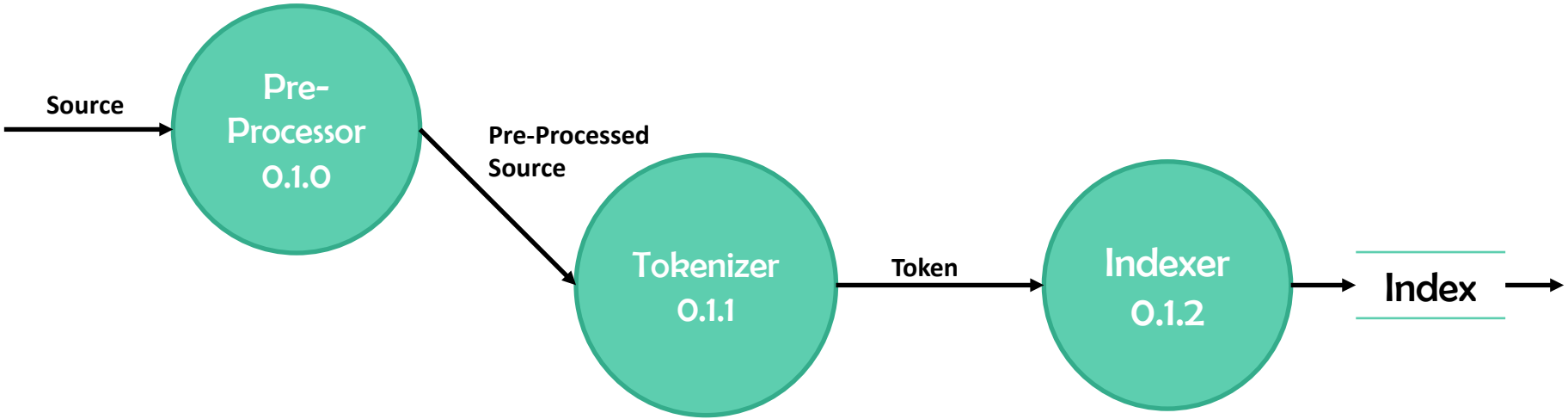
Name	Source Parser
Reference Number	0.1
Input	Source
Output	Index
Description	Reads the Source, and translates to Index. At here if the Source is NULL, the whole Structure Index will be NULL.

Level 1 - Specification

Name	Generator
Reference Number	1.0
Input	Index
Output	Message, CFG
Description	Reads the Index and translates to CFG. Message : "Start" – When the CFG has being made. "Success" – When the CFG has been made successfully. "Fail" – If the Source of the Index is incorrect or the Index is NULL, and if the Index is improper.

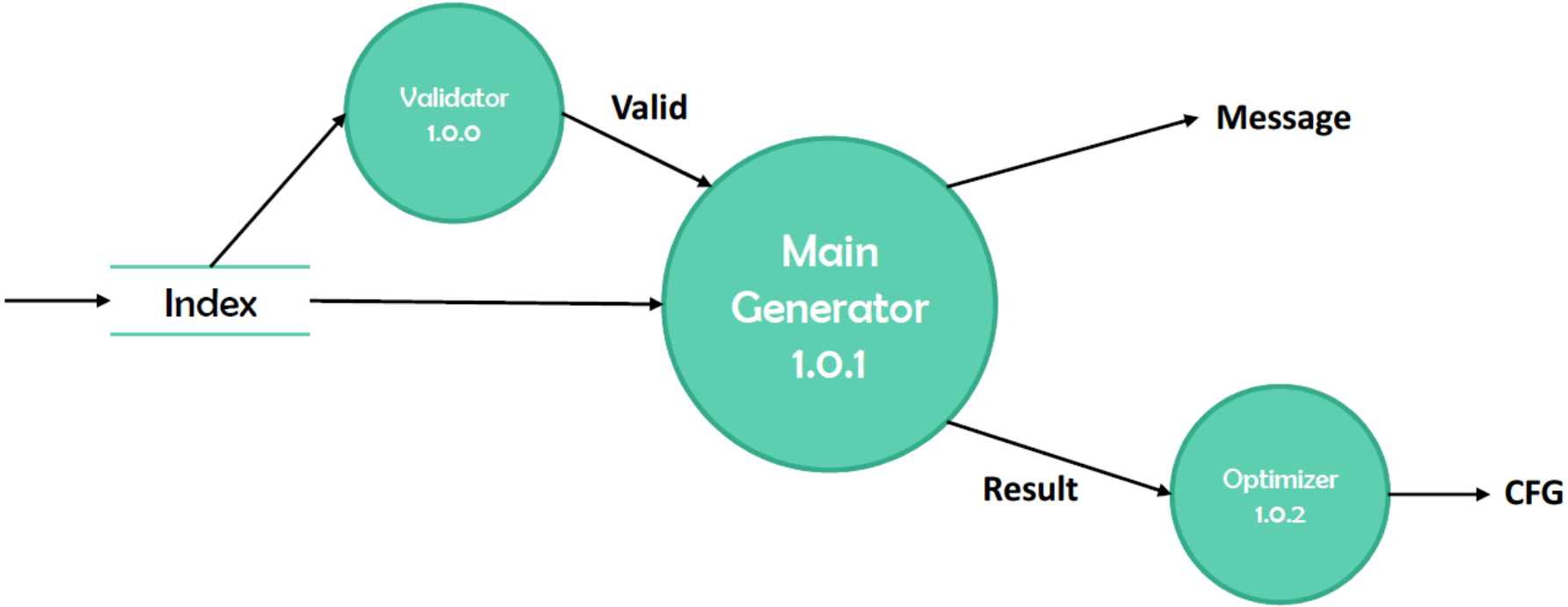
DFD – Level 2

- Source Parser Process



DFD – Level 2

- Generator(1.0) Process



Level 2 - Data Dictionary

Input/Output Event	Description	Format/Type
Pre-processed Source	The C Source Code that has no comments and spaces.	String
Token	The piece of the string list splited as '{', '}', ';'.	String(+Data Strcuture)
Valid	It checks if the Index is correct and saves.	True/False
Result	The piece of the CFG made from Generator. It is being made after gathered from Optimizer.	Piece of CFG

Level 2 - Specification

Name	Pre-Processor
Reference Number	0.1.0
Input	Source
Output	Pre-processed Source
Description	Reads the Source and after deletes comments, spaces, and enters, put it to the Pre-processed Source. If the Source is NULL, the output will be NULL too.

Level 2 - Specification

Name	Tokenizer
Reference Number	0.1.1
Input	Pre-processed Source
Output	Token(+Data Structure)
Description	It saves the result of the Pre-processed Source which has been divided by '{', '}', ';', to the Token. If Pre-processed is NULL, puts NULL to the Token and makes it an Error.

Level 2 - Specification

Name	Indexer
Reference Number	0.1.2
Input	Token(+Data Strcuture)
Output	Index
Description	It handles Token and translate to Index.

Level 2 - Specification

Name	Validator
Reference Number	1.0.0
Input	Index
Output	Valid
Description	Reads Index and check if the Index is correct. If it is correct return True, else return False -Case of False : TypeNum == NULL || Source == NULL || BlockNum == NULL || InEdgeNum == NULL || OutEdgeNum == NULL - Case of True : if not False

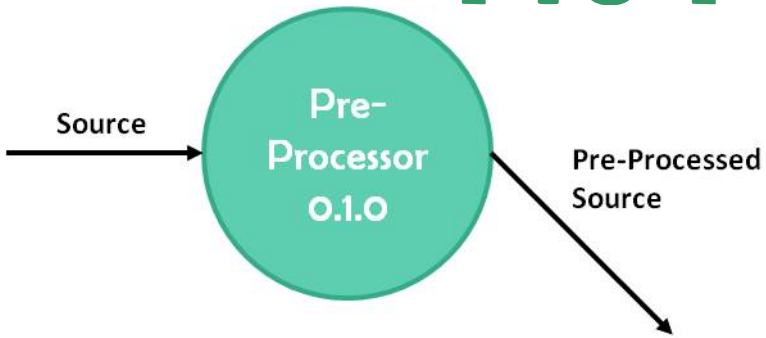
Level 2 - Specification

Name	Main Generator
Reference Number	1.0.1
Input	Valid, Index
Output	Message, Result
Description	If the Valid is True, print “Start” to Message and begins making the CFG. Else the Valid is False, print “Fail” to Message. And then the Result of the Parsing will be gathered in Optimizer and becomes CFG

Level 2 - Specification

Name	Optimizer
Reference Number	1.0.2
Input	Result
Output	CFG
Description	When the all Result has been moved, combines with the former CFG. When the CFG is completed makes it suitable and returns CFG.

Pre-Processor

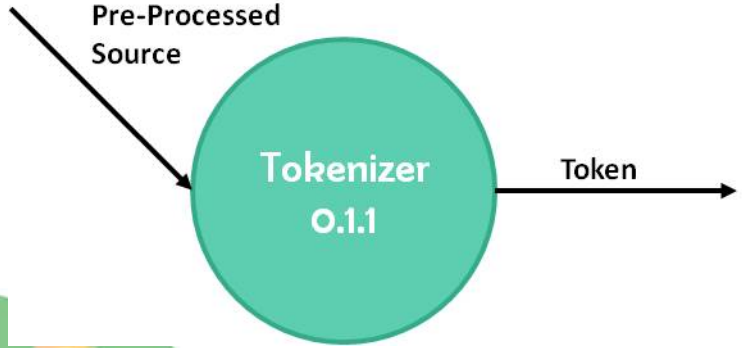


Name	Pre-Processor
Reference Number	0.1.0
Input	Source
Output	Pre-processed Source
Description	Reads the Source and after deletes commer spaces, and enters, put it to the Pre-processed Source. If the Source is NULL, the output will be NULL too.

```
char* pre_Processor(char* i_file)
{
	FILE* file = fopen(i_file, "rt");
	FILE* tmp = fopen("tmp.txt", "wt");
	FILE* tmp2;
	char rCode[BUFSIZE]={0,};
	char c=0;
	int i =0;
	char row_c=0;

	if(file)
	{
		printf("파일오픈성공!₩n");
		while((fgets(rCode,BUFSIZE,file))!=0)
		{
			if(rCode[0]=='#' || rCode[0]=='₩n')
				continue;
			if(_check_comment(rCode))
				continue;
			i=0;
			row_c=0;

			while(1)
			{
				if(rCode[i]==0)
					break;

				row_c=rCode[i];

				if(row_c=='₩t' || row_c=='₩n')
				{
					i++;
					continue;
				}
				fputc(row_c, tmp);
				i++;
			}
		}
	}
```

```
else
{
	return preProcessed;
}

rewind(tmp);
fclose(file);
fclose(tmp);

tmp2 = fopen("tmp.txt", "rt");
fgets(preProcessed, BUFSIZE, tmp2);
fclose(tmp2);
return preProcessed;
}
```

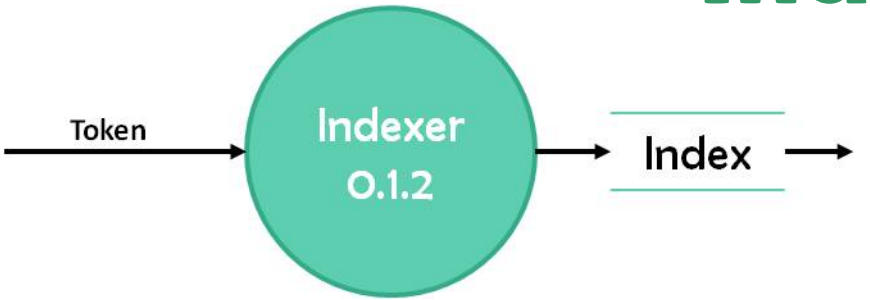
Tokenizer



Name	Tokenizer
Reference Number	0.1.1
Input	Pre-processed Source
Output	Token(+Data Structure)
Description	It saves the result of the Pre-processed Source which has been divided by '{', '}', ';', to the Token. If Pre-processed is NULL, puts NULL to the Token and makes it an Error.

```
TokenList* tokenizer(char* i_preProcessed)
{
        char* pToken = 0;
        char* pSeperator=";";
        char* token[100];
        int i=0;
        TokenList* rTokenlist= create_TokenList();

        if(i_preProcessed==0)
                return 0;

        pToken = strtok(i_preProcessed, pSeperator);
        pToken = strtok(NULL, pSeperator);

        while(pToken!=NULL)
        {
                printf(" %s \n", pToken);
                if(!_check_declaration(pToken))
                {
                        printf("########## %s\n", pToken);
                        insert_token(rTokenlist, pToken);
                }
                pToken=strtok(NULL, pSeperator);
        }
        return rTokenlist;
}
```

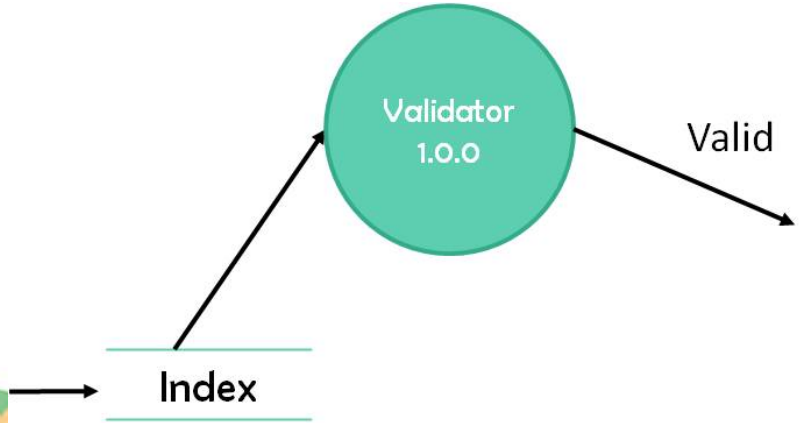
Indexer



Name	Indexer
Reference Number	0.1.2
Input	Token(+Data Strcuture)
Output	Index
Description	It handles Token and translate to Index.

```
IndexList* indexer(TokenList* i_pTokenList)
{
        char* tCursor=0;
        int wrapflag=0;
        IndexList* rIndexList=create_indexList();
        printf("Indexer실행₩n");

        indexeController(i_pTokenList, rIndexList);
        printf("리스트반환");
        return rIndexList;
}
```

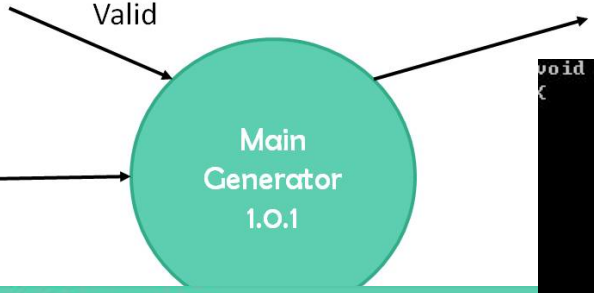
Validator




```
int validator(IndexList* i_indexList)
{
        Index* node=0;
        if(i_indexList==NULL)
                return 0;
        else
        {
                node = i_indexList->next;
                if(node==NULL)
                        return 0;
        }
        return 1;
}
```

Name	Validator
Reference Number	1.0.0
Input	Index
Output	Valid
Description	Reads Index and check if the Index is correct. If it is correct return True, else return False -Case of False : TypeNum == NULL || Source == NULL || BlockNum == NULL || InEdgeNum == NULL || OutEdgeNum == NULL - Case of True : if not False

Main Generator

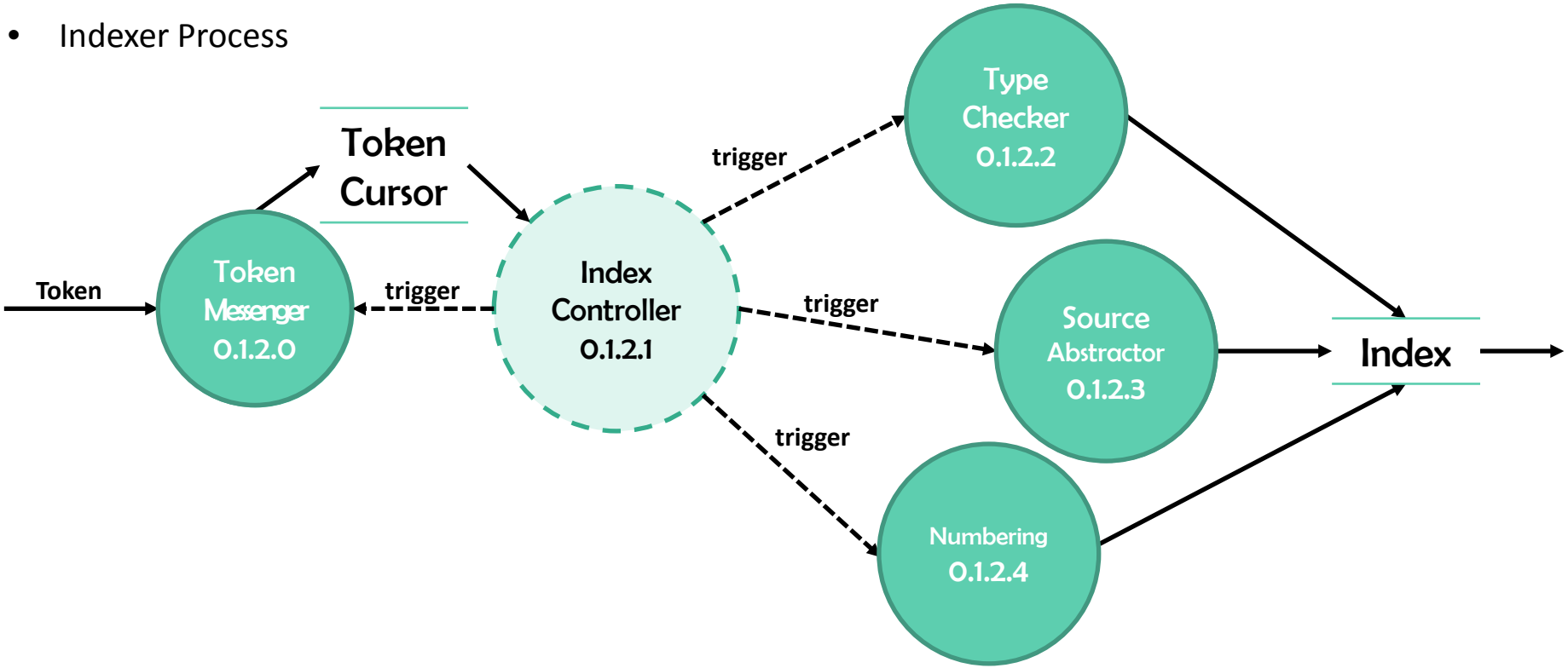



```
void mainGenerator(IndexList* i_iList, int i_valid)
{
        Index* node=i_iList->next;

        printf("MAINGENERATOR");
        printf("START의 outEdge 주소 = %d₩n",node->outEdge);
        if(i_valid==0)
        {
                print(_print_fail());
                return;
        }

        for(node = i_iList->next; node!=i_iList->tail; node=node->outEdge)
        {
                blockCreator(node);
                printf("creator 완료₩n");
        }

}
```

Name	Main Generator
Reference Number	1.0.1
Input	Valid, Index
Output	Message, Result
Description	If the Valid is True, print "Start" to Message and begins making the CFG. Else the Valid is False, print "Fail" to Message. And then the Result of the Parsing will be gathered in Optimizer and becomes CFG

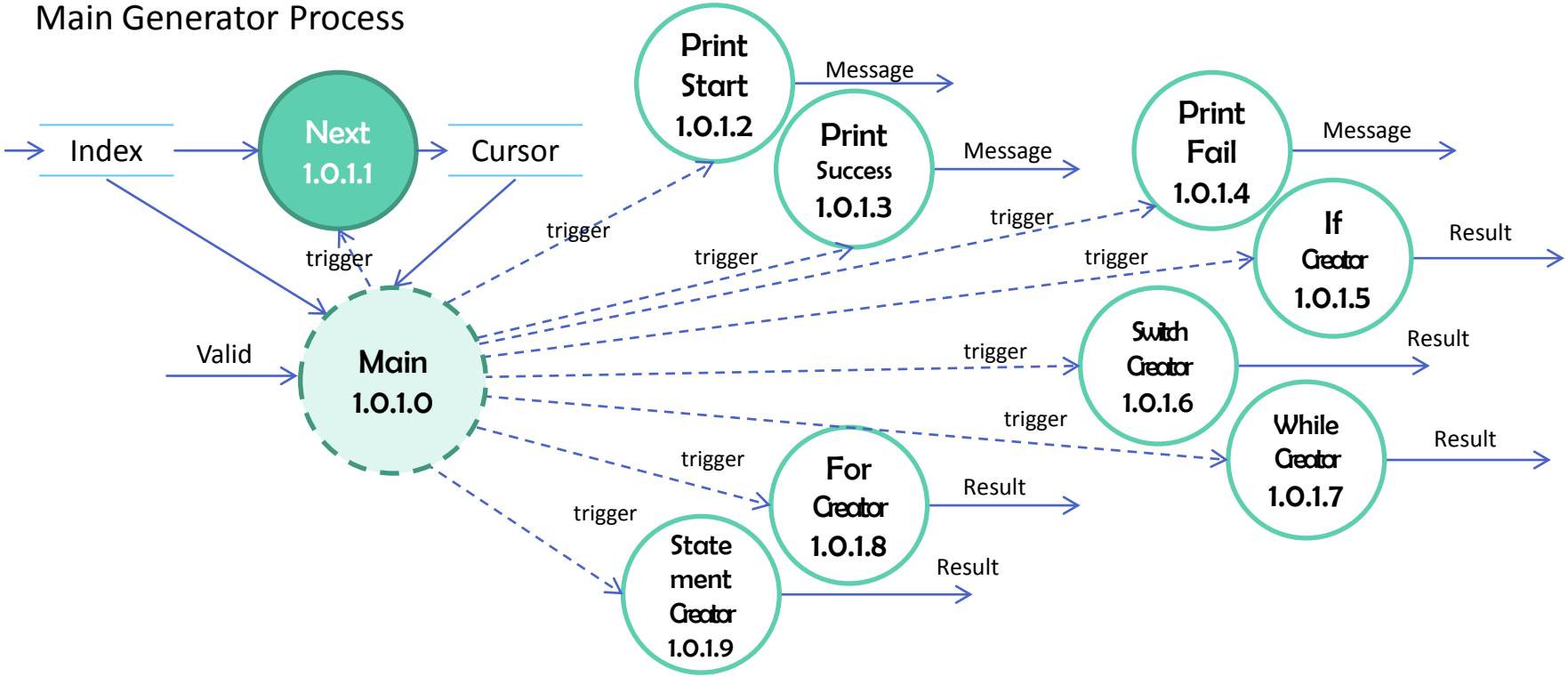
DFD – Level 3

- Indexer Process



DFD – Level 3

- Main Generator Process



Level 3 - Data Dictionary

Input/Output Event	Description	Format/Type
Token Cursor	The pointer directs one Token	'Err', 'Start', 'End' / Pointer OR Constant
Cursor	The pointer which returns the Index form of List by moving Index forward each.	Pointer

Level 3 - Specification

Name	Token Messenger
Reference Number	0.1.2.0
Input	Token(+Data Strcuture)
Output	Token Cursor
Description	Reads the Token, and each time the Controller Triggers, returns the proper Token Cursor. If Token is NULL, Token Cursor will get the constant 'Err'. The first and the last of the Token Cursor will get the constant 'Start' and 'End'.

Level 3 - Specification

Name	Index Controller
Reference Number	0.1.2.1
Input	Token Cursor
Output	trigger
Description	Reads the Token Cursor, and do the right work to each Token Cursor.

Level 3 - Specification

Name	TypeChecker
Reference Number	0.1.2.2
Input	Trigger
Output	Index(+Data Structure)
Description	Divides the General Statements and If,else if, else / switch / while / for then decides the type.

Level 3 - Specification

Name	Source Abstractor
Reference Number	0.1.2.3
Input	Trigger
Output	Index
Description	It abstracts the Source that is applicable with CFG, from the Tokened Statement. Then puts into the Source.

Level 3 - Specification

Name	Numbering
Reference Number	0.1.2.4
Input	Trigger
Output	Index(+Data Structure)
Description	It makes the Number of the CFG's Basic Blocks and Block's in-edge or out-edge properly. Then puts into Index.

Level 3 - Specification

Name	Main
Reference Number	1.0.1.0
Input	Index, Cursor, Valid
Output	Trigger
Description	It confirms the Cursor's TypeNum and Triggers the Creator that matches with the TyperNum. Also, if the Valid is false, Trigger the Print Fail. And the first Valid to be read is True, Trigger the Print Start and the last Valid to be read is True, Trigger Print Success.

Level 3 - Specification

Name	Next
Reference Number	1.0.1.1
Input	Index
Output	Cursor
Description	Reads the proper Index and each time it is called moves to the next and returns the proper Cursor.

Level 3 - Specification

Name	Print Success
Reference Number	1.0.1.2
Input	Trigger
Output	Message
Description	If Triggered put “Success” to Message

Name	Print Fail
Reference Number	1.0.1.3
Input	Trigger
Output	Message
Description	If Triggered put “Fail” to Message

Level 3 - Specification

Name	Print Start
Reference Number	1.0.1.4
Input	Trigger
Output	Message
Description	If Triggered put “Start” to Message

Level 3 - Specification

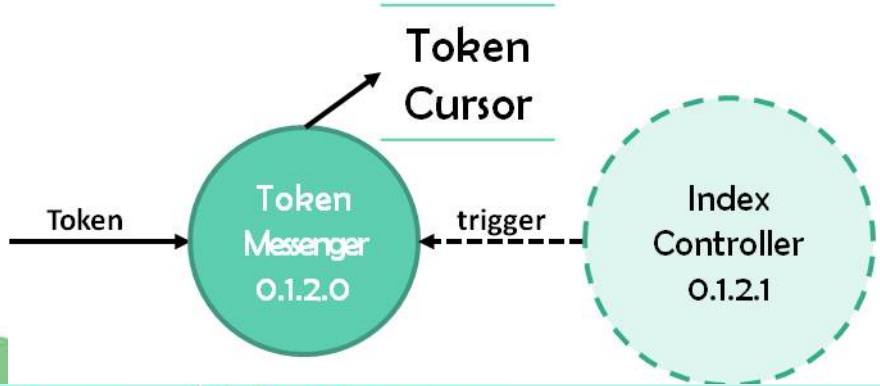
Name	If Creator
Reference Number	1.0.1.5
Input	Trigger
Output	Result
Description	If Triggered make CFG that accords with IF Statement.

Name	Switch Creator
Reference Number	1.0.1.6
Input	Trigger
Output	Result
Description	If Triggered make CFG that accords with SWITCH Statement.

Level 3 - Specification

Name	While Creator
Reference Number	1.0.1.7
Input	Trigger
Output	Result
Description	If Triggered make CFG that accords with WHILE Statement.
Name	**For Creator**
Reference Number	1.0.1.8
Input	Trigger
Output	Result
Description	If Triggered make CFG that accords with For Statement.

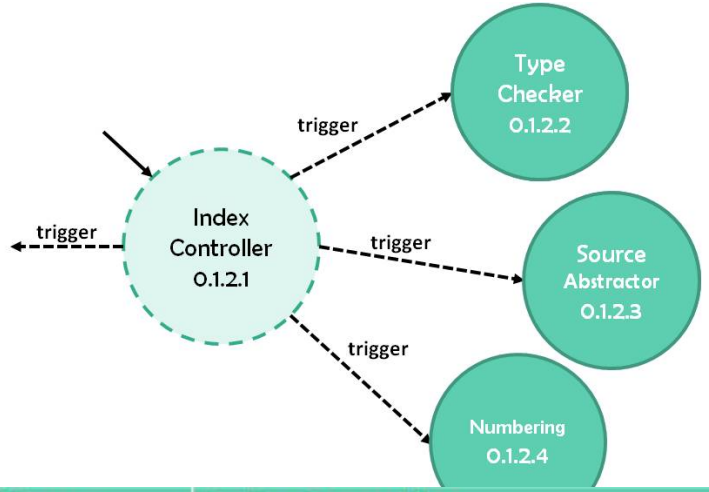
Token Messenger



Name	Token Messenger
Reference Number	0.1.2.0
Input	Token(+Data Strcuture)
Output	Token Cursor
Description	Reads the Token, and each time the Controller Triggers, returns the proper Token Cursor. If Token is NULL, Token Cursor will get the constant 'Err'. The first and the last of the Token Cursor will get the constant 'Start' and 'End'.

```
char* tokenMessenger(TokenList* i_list, int back)
{
	static int c=0;
	Token* node = i_list->next;
	int i=0;
	if(back==1)
	{
		c--;
		return 0;
	}

	else
	{
		if(i_list->size==c-1)
			return 0;

		c++;
		for(i=0;i<c;i++)
		{
			if(node->next!=i_list->tail && node->next!=i_list->next)
			{
				node = node->next;
			}
		}
		return node->str;

	}
}
```

Index Controller



Name	Index Controller
Reference Number	0.1.2.1
Input	Token Cursor
Output	trigger
Description	Reads the Token Cursor, and do the right work to each Token Cursor.

```
IndexList* indexeController(TokenList* i_tokenList, IndexList* i_indexList)
{
        IndexList* nbList=0;
        Index* nNode=0;
        Index* bNode=0;
        Index* tNode = 0;
        char* nowToken;
        char* init_wrap = 0;
        char* test =0;
        char* test2=0;
        char fortmp[100]={0,};

        printf("indexercontrol");
        while(1)
        {
                printf("₩n==================₩n");
                tNode = nNode;
                nowToken = tokenMessenger(i_tokenList, 0);
                printf(" - 메신저에서 넘어온 현재처리할 token = %s₩n", nowToken);
                if(nowToken==NULL)
                        return;
                switch(_type_checker(nowToken))
                {
                case ELSEIF:
                        printf("Elseif처리₩n");
                        test = (char *)malloc(sizeof(char)*50);
                        test2 = _wrap_str(i_tokenList, nowToken);
                        bNode = create_Index(test2, blockNumCount++, ELSEIF);
                        insert_Index(nNode->branch_list, bNode);
                        printf("- %s₩n", test2);
                        break;
```

Index Controller(cont.)

```
case IF:
        printf("if처리₩n");

        nNode = create_Index(strtok(nowToken,"{"), blockNumCount++, IF);
        insert_Index(i_indexList, nNode);
        nowToken = strtok(NULL, "{");
        test2 = _wrap_str(i_tokenList, nowToken);
        bNode = create_Index(test2, blockNumCount++, IF);
        nNode->branch_list=create_indexList();
        insert_Index(nNode->branch_list, bNode);
        printf("- %s₩n", nowToken);
        printf("- %s₩n", test2);
        break;

case ELSE:
        printf("else처리₩n");
        test2 = _wrap_str(i_tokenList, nowToken);
        bNode = create_Index(test2, blockNumCount++, ELSE);
        insert_Index(nNode->branch_list,bNode);
        printf("- %s₩n",test2);
        break;
case FOR:
        printf("for처리₩n");
        test2=_wrap_str3(i_tokenList, nowToken);
        printf("** %s₩n₩n", test2);
        nNode=create_Index(strtok(test2,"{"),blockNumCount++, FO
        insert_Index(i_indexList, nNode);
        bNode = create_Index(strtok(NULL,"{"), blockNumCount++,
        printf("+++%s₩n₩n", bNode->source);
        nNode->branch_list = create_indexList();

        insert_Index(nNode->branch_list, bNode);
        break;
case SWITCH:
        printf("switch처리₩n");
        nNode = create_Index(strtok(nowToken,"{"), blockNumCount
        insert_Index(i_indexList, nNode);
        test2 = _wrap_str(i_tokenList, strtok(NULL, "{"));
        printf("========%s₩n₩n₩n₩n₩n₩n", test2);

        _wrap_str2(i_indexList, test2);
        break;
```

```
case WHILE:
        printf("while처리₩n");
        test2 = _wrap_str(i_tokenList, nowToken);
        nNode = create_Index(strtok(test2, "{"), blockNumCount++, WHILE);
        printf("= %s₩n", nNode->source);
        bNode = create_Index(strtok(NULL, "{"), blockNumCount++, WHILE);
        printf("= %s₩n", bNode->source);
        nNode->branch_list = create_indexList();
        insert_Index(nNode->branch_list, bNode);
        insert_Index(i_indexList, nNode);
        break;
default:
        printf("일반문처리₩n");
        test = (char *)malloc(sizeof(char)*100);
        strcpy(test,nowToken);
        if(!(strlen(nowToken)==1 && nowToken[0]==')'))
        {
                nNode = create_Index(test, blockNumCount++, GENERAL);
                printf("==== %s₩n₩n", test);
                insert_Index(i_indexList, nNode);
        }
        break;
}
```

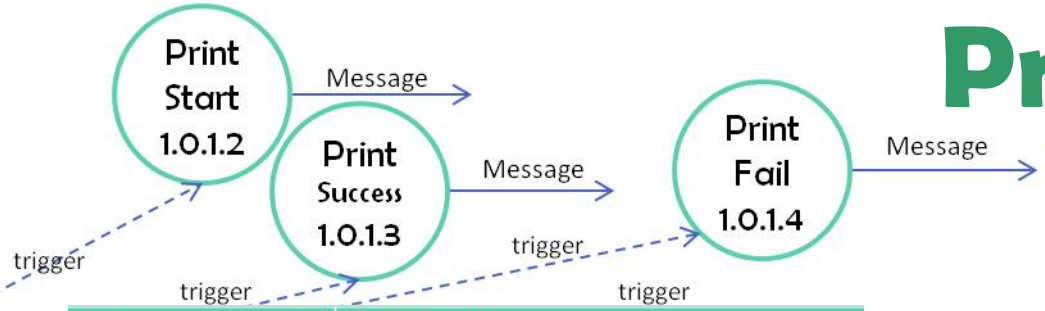
```
Type _type_checker(char* i_str)
{
        if(strstr(i_str, "else if"))
                return ELSEIF;

        if(strstr(i_str, "if"))
                return IF;

        if(strstr(i_str, "else"))
                return ELSE;

        if(strstr(i_str, "switch"))
                return SWITCH;

;tr(i_str, "for"))
                return FOR;

;tr(i_str, "while"))
                return WHILE;

GENERAL;
```

Next

Name	Next
Reference Number	1.0.1.1
Input	Index
Output	Cursor
Description	Reads the proper Index and each time it is called moves to the next and returns the proper Cursor.

```
Index* nextIndexNode(IndexList* i_iList, Index* i_startNode)
{
        Index* node=0;
        node = i_iList->next;

        for(node=node; node!=i_iList->tail; node=node->outEdge)
        {
                if(node == i_startNode)
                {
                //      node = node->outEdge;
                        return node;
                }
        }
}
```

Print Messages



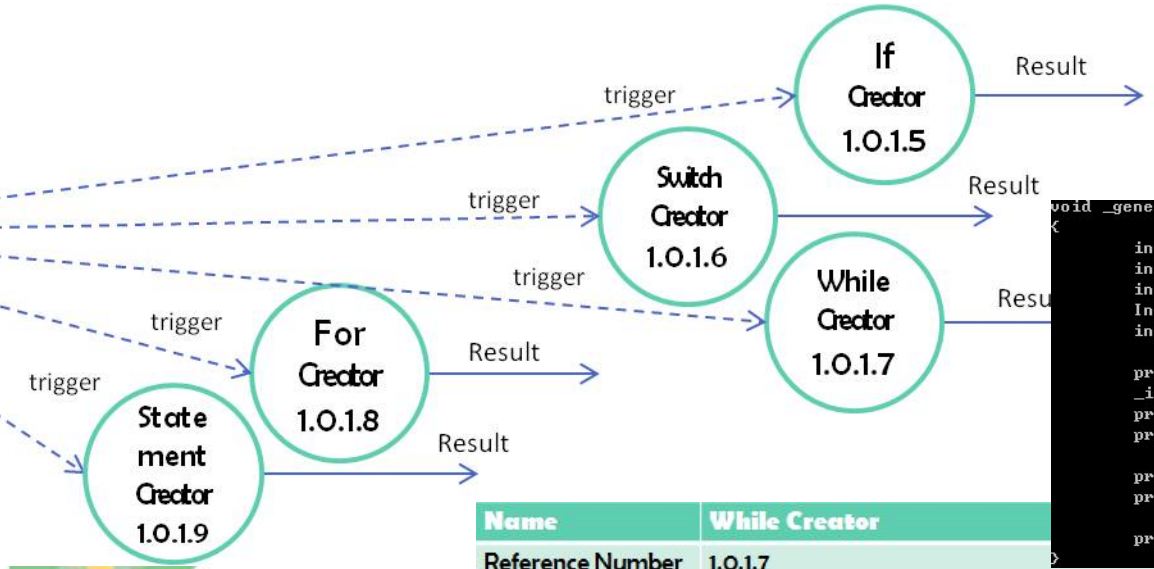
Name	Print Success
Reference Number	1.0.1.2
Input	Trigger
Output	Message
Description	If Triggered put "Success" to Message

Name	Print Fail
Reference Number	1.0.1.3
Input	Trigger
Output	Message
Description	If Triggered put "Fail" to Message

Name	Print Start
Reference Number	1.0.1.4
Input	Trigger
Output	Message
Description	If Triggered put "Start" to Message

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

char* _print_start()
{
        char* r = "START!!";
        return r;
}

char* _print_success()
{
        char* r= "SUCCESS!!";
        return r;
}

char* _print_fail()
{
        char* r= "FAIL";
        return r;
}
```

Creators




```
void _generalCreator(Index* i_in)
{
        int i=0;
        int tnum=0;
        int bNum=i_in->inEdge->branch_list->size;
        Index* tnode=0;
        int in=0, out=0;

        printf("        ## BLOCK[%2d]", i_in->blockNum);
        _inedge_display(i_in);
        printf("%2d)", i_in->outEdge->blockNum);
        printf(" ##\n");

        print
        print
        print
}
```

```
void _ifCreator(Index* i_in)
{
        int i=0;
        int tnum=0;
        int bNum=i_in->branch_list->size;
        Index* tnode=0;

        printf("        ## BLOCK[%2d] ", i_in->blockNum);
        _inedge_display(i_in);
        _outedge_display(i_in);
        printf(" %s\n", i_in->source);
        printf("_____________________________________________\n");
```

```
void _switchCreator(Index* i_in)
{
        int i=0;
        int tnum=0;
        int bNum = i_in->branch_list->size;
        Index* tnode=0;
        printf("        ## [%2d] ",i_in->blockNum);
        _inedge_display(i_in);
        _outedge_display(i_in);
        printf(" %s\n", i_in->source);
        printf("_____________________________________________\n");
        _branch_display(i_in);
        printf("\n\n");
```

Name	If Creator
Reference Number	1.0.1.5
Input	Trigger
Output	Result
Description	If Triggered make CFG th

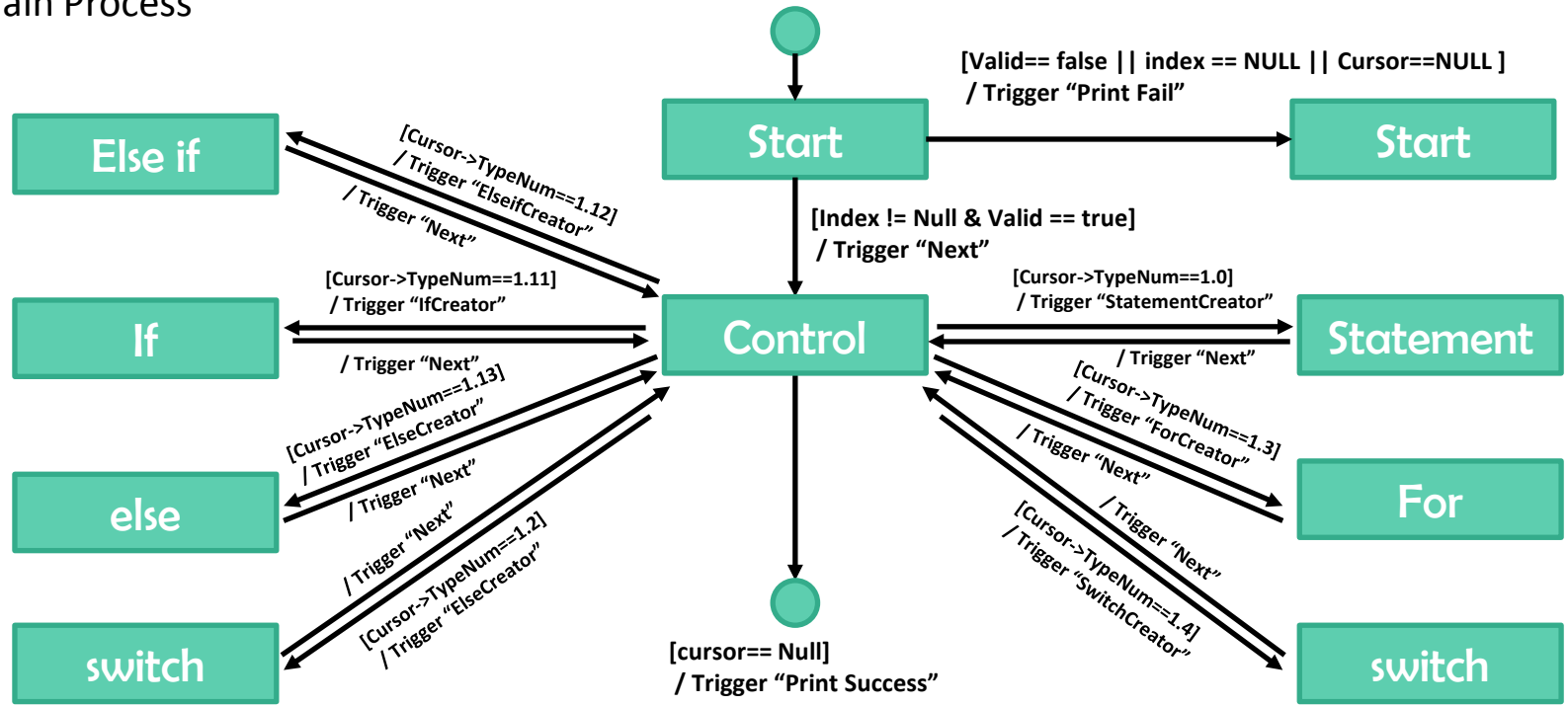
Name	Switch Creator
Reference Number	1.0.1.6
Input	Trigger
Output	Result
Description	If Triggered make CFG that accords with SWITCH Statement.

Name	While Creator
Reference Number	1.0.1.7
Input	Trigger
Output	Result
Description	If Triggered make CFG that accords with WHILE Statement.

Name	For Creator
Reference Number	1.0.1.8
Input	Trigger
Output	Result
Description	If Triggered make CFG that accords with For Statement.

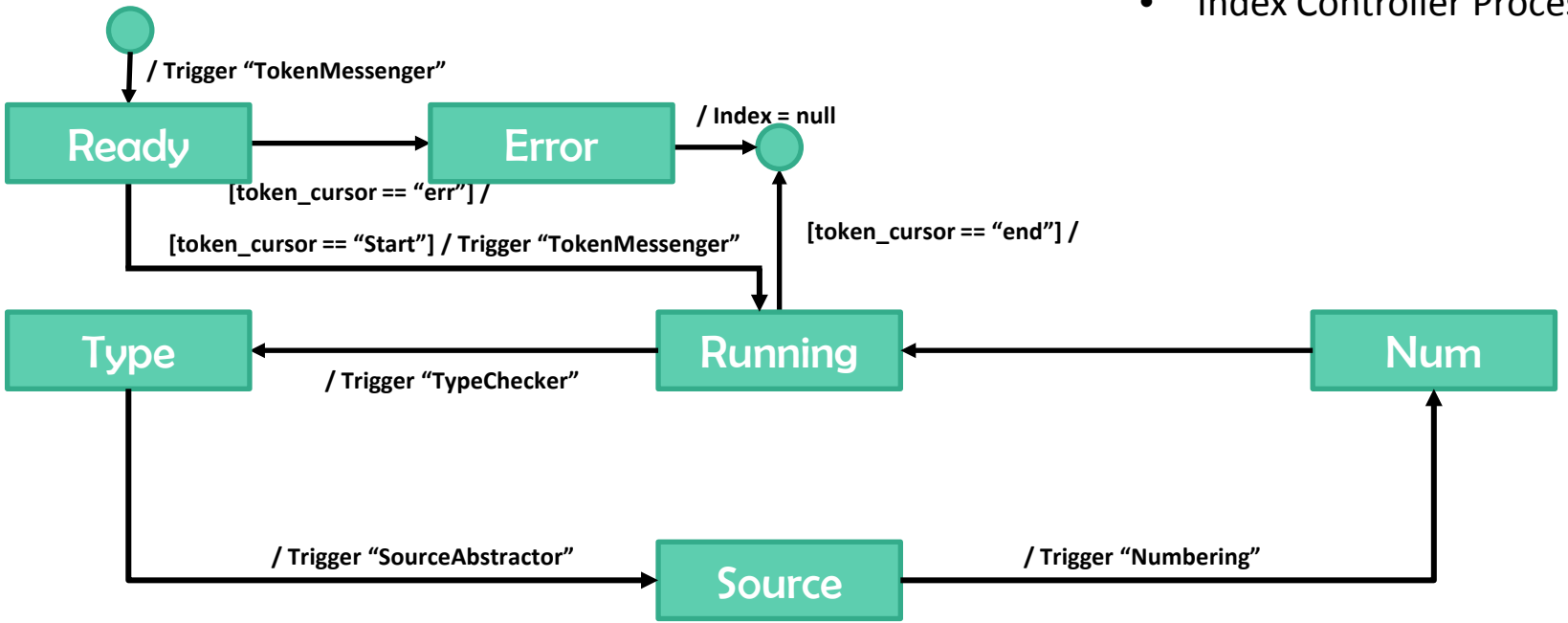
FSM – Level 4

- Main Process

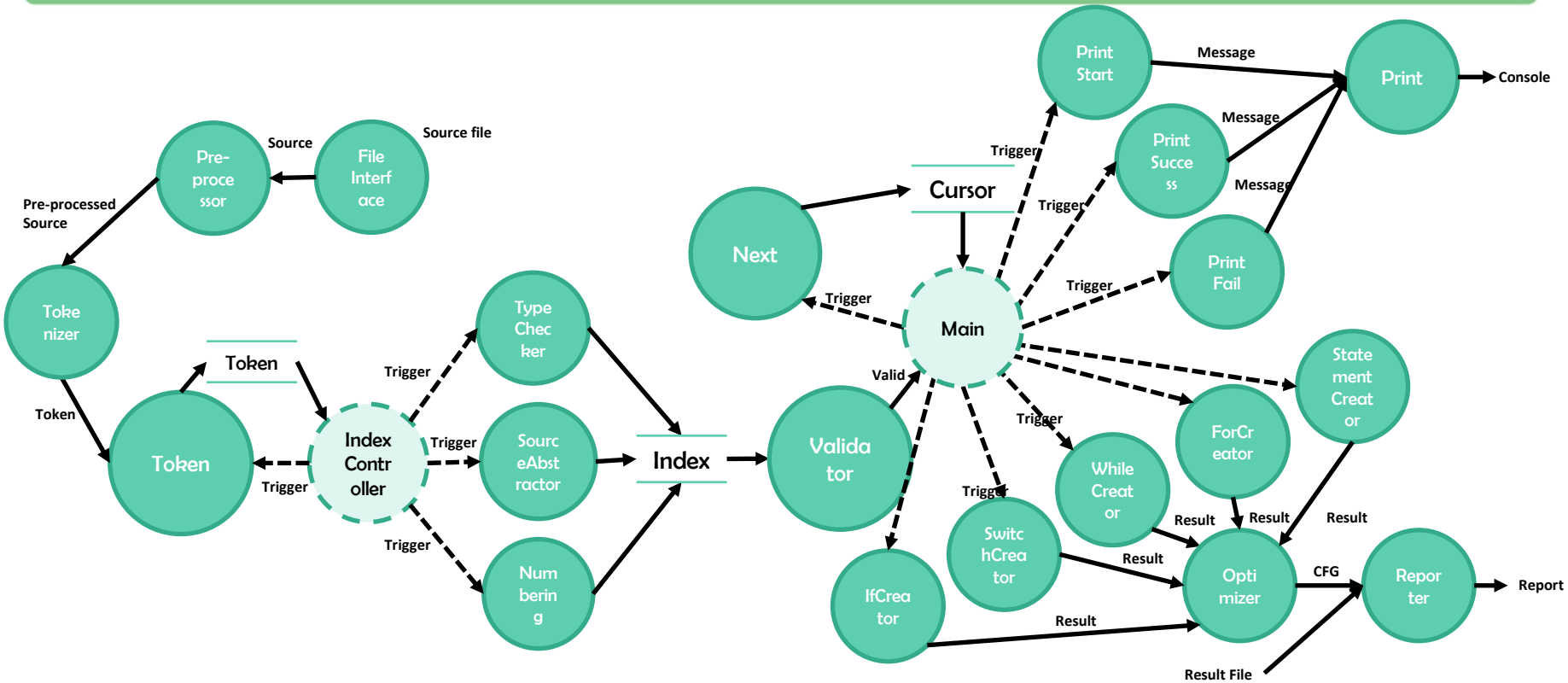


FSM – Level 4

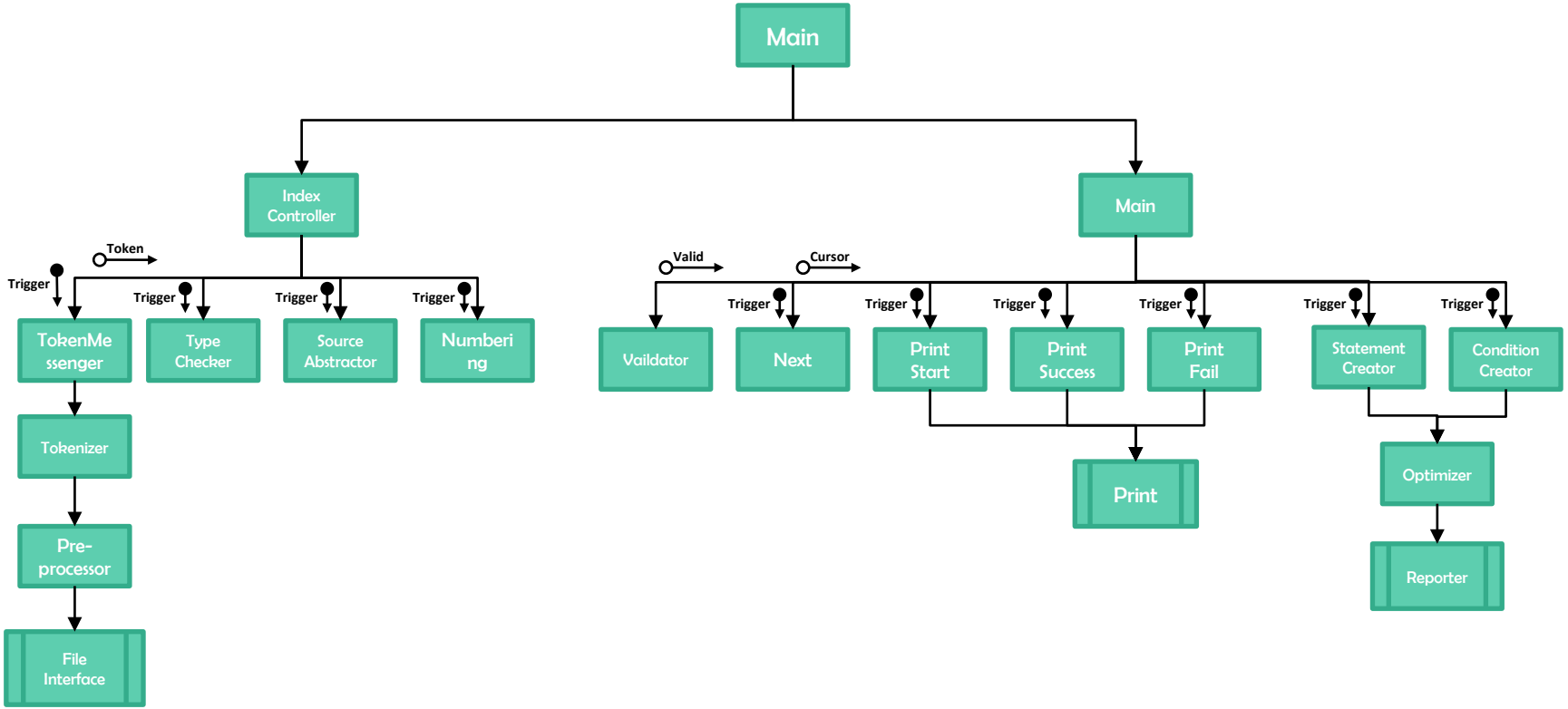
- Index Controller Process



DFD


Pre-
proce
ssor
Source
File
Interf
ace
Source file
Pre-processed
Source
Toke
nizer
Token
Token
Token
Index
Contr
oller
Trigger
Trigger
Trigger
Type
Chec
ker
Sourc
eAbst
ractor
Numb
erin
g
Index
Cursor
Next
Trigger
Main
Valid
Valida
tor
IfCrea
tor
Switc
hCrea
tor
While
Creat
or
ForCr
eator
State
ment
Creat
or
Print
Start
Print
Succe
ss
Print
Fail
Message
Print
Console
Result
Opti
mizer
CFG
Repor
ter
Report
Result File

Structured Charts



Thank You!!