**Design and Development of Compiler**

**for C- Language**

Phase 4: Design and Implementation of Code Generator

과목명: [CSE4120] 기초 컴파일러 구성

담당교수: 서강대학교 컴퓨터공학과 정 성 원

개발자: 18조 윤제형 20151575

오지민 20151567

개발기간: 2019. 6. 24 - 2019. 6. 25

Project4 결과 보 고 서

프로젝트 제목: Design and Development of Compiler for C-Language:

Phase 4: A Code Generator

제출일: 2019. 6. 25.

개발자: 18조 오지민(20151567) 윤제형(20151575)

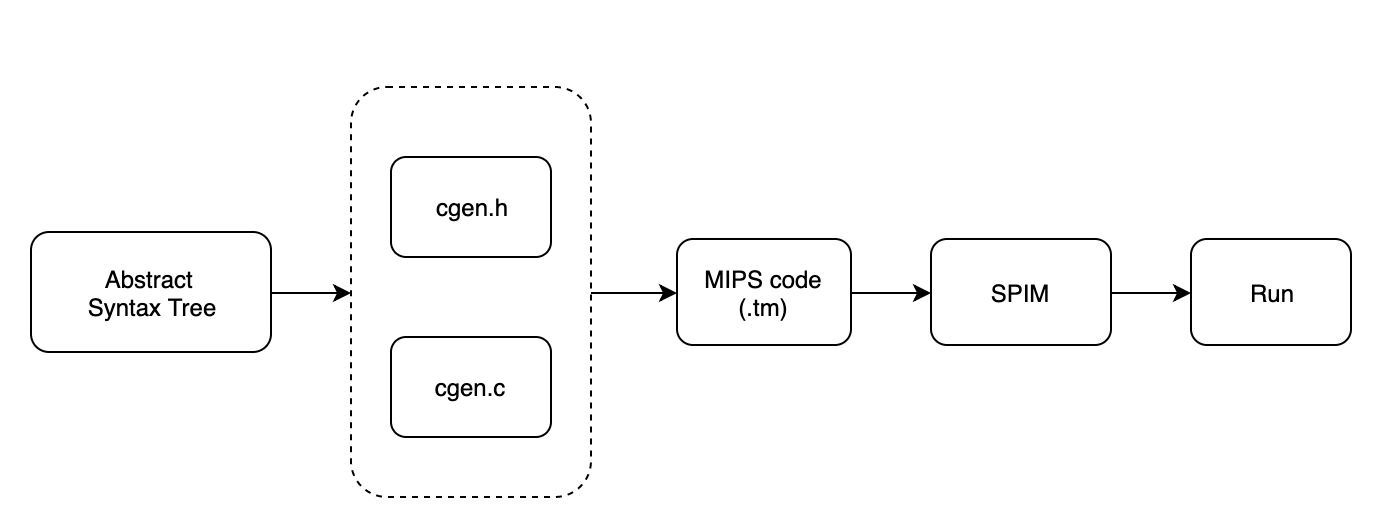
**1. 개발 목표**

이전에 개발한 Project3의 결과물을 토대로, cgen.h cgen.c를 추가하여 교재 491~492 페이지에서 정의하는 C- 문법에 맞는 컴파일러의 code generator를 작성한다. 이때 작성된 프로그램의 실행결과 나타나는 MIPS 파일은 제공되는 SPIM 상에서 정상적으로 작동해야 한다. 또한 input, ouput 함수를 추가해서 SPIM 상에서 입출력이 가능하게 한다.

**II. 개발 범위 및 내용**

**가. 개발 범위**

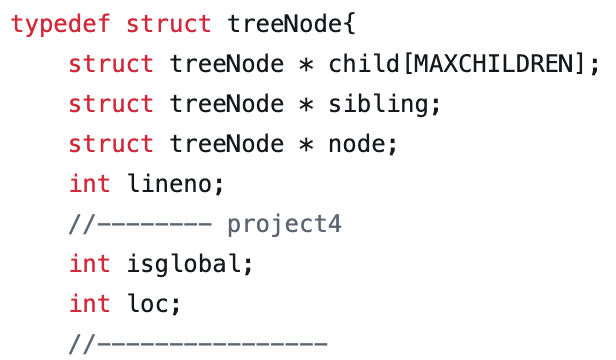
- 교재 491~492 페이지에서 정의하는 C- 문법에 맞는 컴파일러의 code generator를 작성한다. Project3을 통해서 생성된 AST가 가지고 있는 정보를 가지고, SPIM이 Assemble 할 수 있는 Assemble Code를 만들어 내서, 최종적으로 SPIM으로 C- 언어의 코드를 Generate 시킨다.



**나. 개발 내용**

**globals.h**

- analyze.c에서 AST에 새롭게 저장되어야 할 int형 변수인 isglobal(global변수인지 아닌지를 판단하기 위한 값, global variable이면 1, 아니면 0을 저장), frame내에 fp를 기준으로 상대적 위치를 저장하는 int형 변수인 loc를 treeNode Structure에 추가한다.



**analyze.c**

- symbol table에 node를 추가하기 전에 location과 global인지 여부를 treeNode Structure에 저장한다. symbol table에 저장할 때 계산하였던 frame 내의 위치를 저장해준다.

**symtab.c**

- st\_lookup\_isglobal()를 새로 만들어주었다. total\_sym, 즉 현재 symbol table의 head가 가리키고 있는 곳부터 해당 name을 가진 node를 찾는다. 물론 현재 scope에서 접근 가능한 scope까지 찾는다. 찾다가 global\_sym까지 오게 되면, global variable을 의미하고 1을 return하고 그렇지 않으면 0을 return한다.

**cgen.c**

- C- 언어에 맞춰서 MIPS 파일을 작성하기 위해 codeGen(), node type에 따라 호출되는 genDec(), genExp(), genStmt() 등을 새롭게 작성한다. input, output함수에 따른 action도 call node일 때에 따로 저장해주어서 입출력이 가능하게 바꿔주었다. 사용한 명령어들은 SPIM manual의 A.10을 참고하였다.

**III. 추진 일정 및 개발 방법**

**가. 추진 일정**

6/24 : tiny machine을 위한 tm 코드 작성 시도, SPIM으로 변경 및 analyze.c, globals.h 추가로 구현

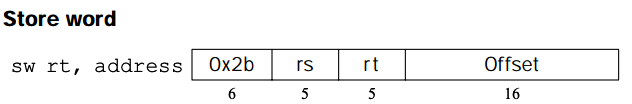
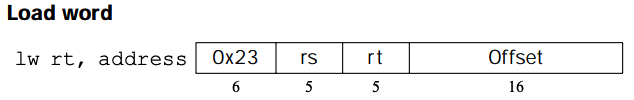
6/25 : cgen.c 작성 및 testcase를 통해 프로그램 수정 및 보완, 보고서 작성

**나. 개발 방법**

LINUX 개발 환경에서 이루어졌으며 C-문법과 MIPS 코드를 이해하고 MIPS 명령어들을 이해하고 숙지한다. 여러 testcase로 실험해보면서 개발을 진행하고 프로그램의 안정성과 신뢰성을 확보한다.

SPIM manual을 참고하여 MIPS 명령어를 이용한 코드를 작성한다. 주로 사용되는 명령어들은 다음과 같다.





**다. 연구원 역할 분담**

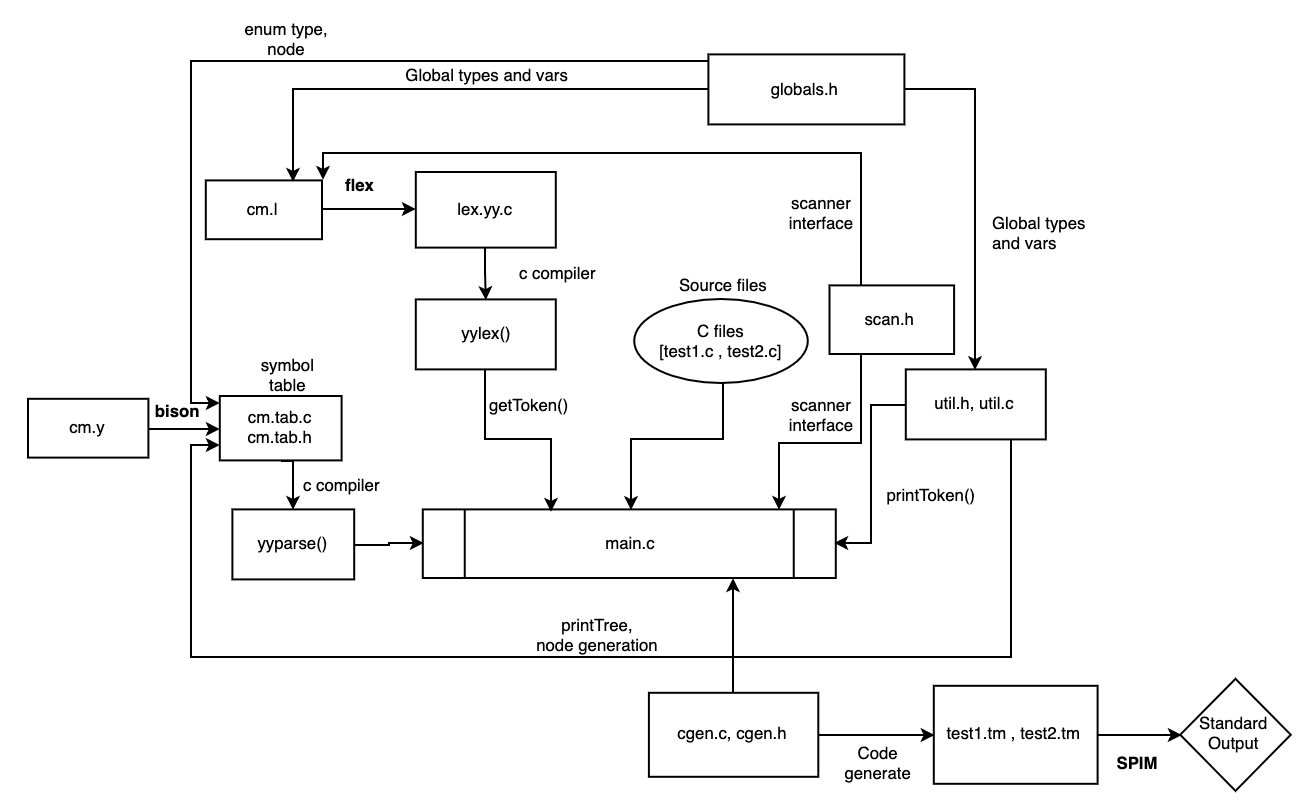
- 오지민 : cgen.c 구현 및 오류 체크 사항 확인

- 윤제형 : cgen.c 구현 및 오류 체크 사항 확인

**IV. 연구 결과**

**1. 합성 내용**

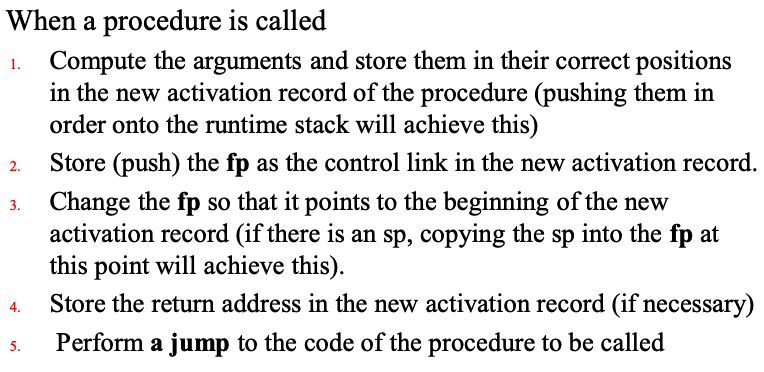
**1-1 구성도 :**

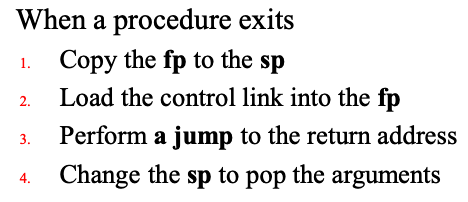


**1-2 project4 추가 부분 설명 :**

**- cgen.h, cgen.c**

main에서 cgen.c에 정의되어 있는 codeGen()을 호출한다. 입력되어 들어온 tree의 node type에 따라 해당하는 적절한 code를 generate한다. node type에 따라 다른 함수를 호출하고 recursive 호출을 적절히 활용하여 자식 노드의 연산이 먼저 필요한 경우도 처리하였다. register $fp, $sp, $gp 를 사용하여 값을 저장하고, 접근하였다. 저장하는 방식은 강의자료를 참고하였다.





**2. 분석 내용**

Project3에서 만든 AST를 가지고 그곳에 저장된 정보들을 이용해서 C-에 대한 Assemble을 수행 할 수 있는 Code를 만든다. Project3에서 만들어진 Tree에서 Location 정보, global variable인지에 대해 확인한 정보를 추가시킨 후 저장된 Node 정보를 이용해서 $sp(stack pointer),$fp(frame pointer),$gp(global pointer)와 같은 포인터 들을 사용해서 저장된 위치 정보를 이용해 code가 Assemble 해야 할 내용을 만든다.

기본적으로 구조는 책에 나와 있는 tiny언어에 대한 Cgen.c 코드를 기반으로 하되 C-에 맞게 확장 시키고, Input, output 함수를 설정해서 입출력이 가능 하게 만든다. (자세한 내용은 3.제작내용에 후술)

**3. 제작 내용**

**- void codeGen()**

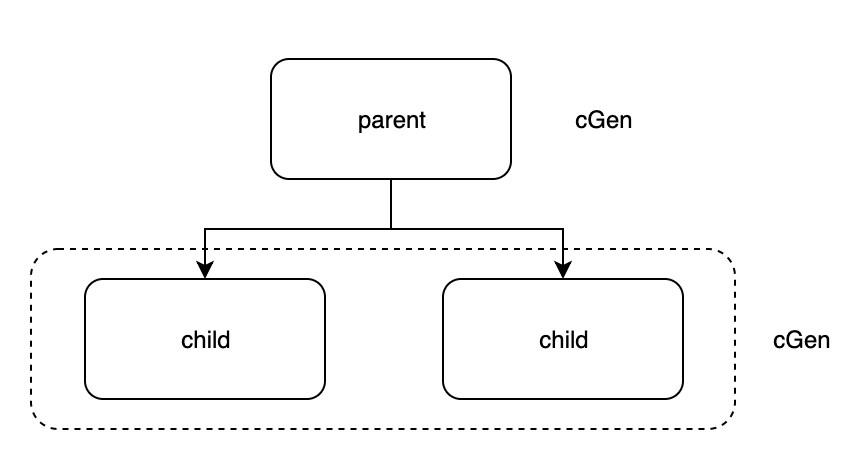
codeGen 함수에서는 .text, .globl 와 같은 assembler directives을 먼저 작성해주고, opcode reg1, reg2, reg3 와 같은 코드 generating 을 위해 cGen으로 ADT를 넘겨서 code generate을 시작한다.

**- static int allocate\_label()**

label이 필요한 경우 L(번호)의 형식을 위해 번호를 다루는 함수이다.

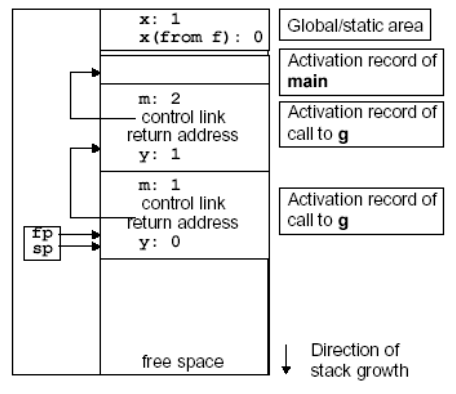
**- static void cGen()**

cGen에서는 넘어온 Tree를 각각의 Kind에 따라서 구분하여 그에 맞는 함수들에게 넘겨주는 역할을 한다. genDec, genStmt, genExp를 호출하여 자식을 cGen으로 넘기기 전에 적절한 코드를 생성하고 자식을 cGen으로 보낸 후의 코드도 적절히 생성한다.



**- static void genDec()**

함수 call시 선언되어있는 곳으로 가야하기 때문에 함수 선언시 control link와 return address, local variable들을 fp, sp 기준으로 저장한다. 저장 방식은 다음과 같다



**- static void genExp()**

Id이고 assign 인 경우에는 주소값을, 아닌 경우에는 그 variable이 가진 값을 $t0에 저장한다. operater가 있는 연산식의 경우 왼쪽 child와 오른쪽 child을 recursive하게 호출하여 최종 결과값은 $t0가 가지고 있도록 한다.

**- static void genStmt()**

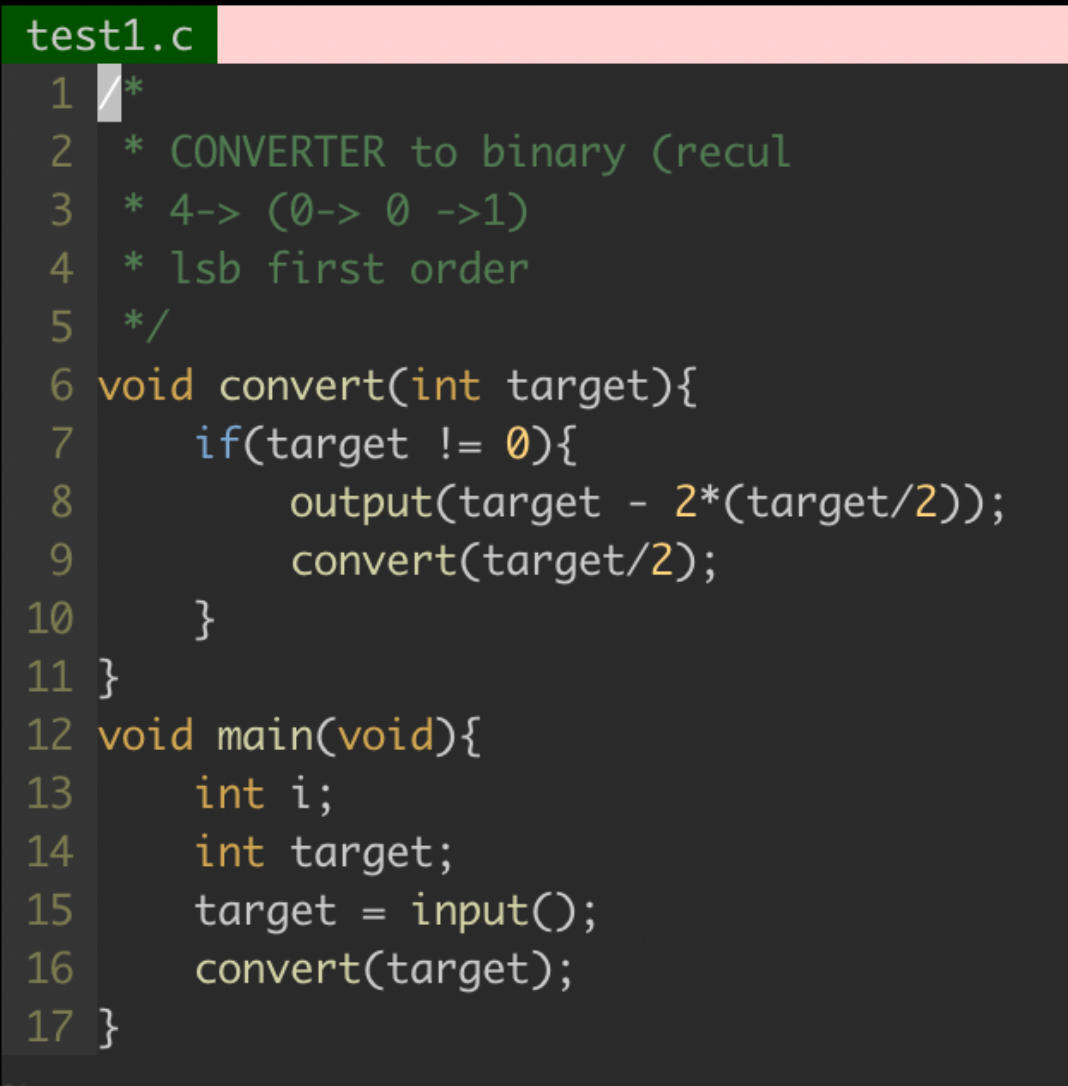
if/else문의 경우 조건에 성립하지 않을 경우 점프할 곳의 label이 필요하므로 allocate\_label을 통해 if/else문을 처리한다. 반복문의 경우에도 되돌아갈 곳의 label이 필요하므로 allocate\_label을 통해 처리한다. return의 경우 fp, sp를 set하기 위한 공통적인 코드를 따로 라벨링한 후 return 노드를 만났을 때 그곳으로 점프한다. call인 경우 input, output함수에 대한 처리를 해두고, user function호출의 경우 argument를 저장해준다.

**4. 시험**

두 가지 테스트케이스로 최종 완성한 프로그램을 test한다

**test1.c**

10진수를 입력받아 2진수로 변환하는 코드이다. LSB부터 출력하여 MSB까지 출력한다.



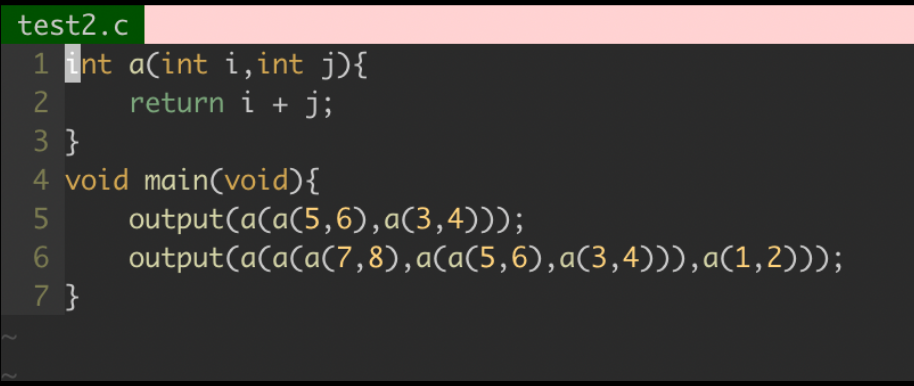
다음은 두 번 돌려 본 결과이다.

15 = 1111

16 = 1000

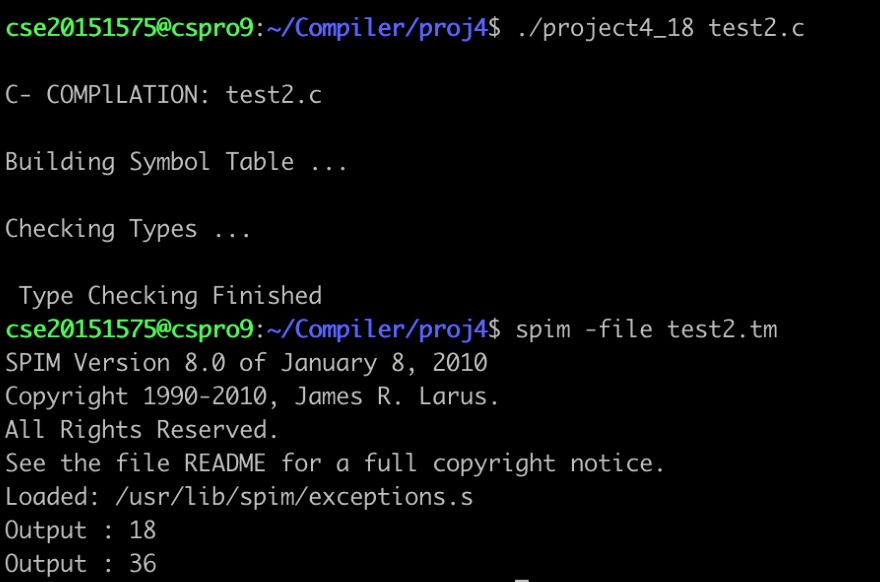


**test2.c**



a 함수를 복잡한 call 형식으로 출력해보았다.

결과는 다음과 같다.



**5. 평가**

C-에 대한 내용을 전체적으로 커버 할 수 있었다. 기본적인 Syntax 에러와 Semantic 에러를 기존의 프로젝트에서 잡아내어 error-free한 코드가 넘어온다고 가정할 수 있으며 넘어오는 Code를 가지고 Generate 시키기 때문에 에러 코드로 인한 점에서는 안정성을 어느 정도 확보 할 수 있다.

testcase를 5개 이상 돌려보며 에러를 해결하였으며 정상적으로 돌아가는 MIPS 코드를 만들었다고 할 수 있다.

**V. 기타**

**1. 연구조원 기여도**

오지민 : 50%

윤제형 : 50%

**2. 자체 평가**

팀원끼리 협동하여 각각 기존 코드 내용에서 추가할 부분을 찾아서 추가하고 확장되어야 할 부분에 대해서 분석하고 tm코드로 코드를 생성한 후 제대로 생성했는지 함께 확인하고 눈으로 디버깅해보고, output함수를 통해 출력해보며 확인하기도 하였다. 서로 협동하여 코드를 작성하고 git으로 코드를 합쳐가며 코드 관리를 하였다. 또한 프로젝트에 대한 testcase도 최대한 많은 케이스에 대해 만들어서 안정성과 신뢰도가 있다고 판단된다.