2020 COMPILER PROJECT-2

이 프로젝트의 목표는 Input Program을 컴파일하고 Target Code를 생성하는 Simple Compiler를 만드는 것입니다. 주어진 Grammar로부터 Scanner, Parser, Code Generator로 컴파일러를 구현해야 하며, 작성한 코드에는 Error handling Routines, Comments가 있어야 합니다. 본 프로젝트에서 보고자 하는 바는 프로젝트의 Design, Parsing table, Understanding Compiler이기 때문에 만약 완성하지 못하더라도 제출하면 부분점수 부여 할 예정입니다.

수업시간에 배웠던 것처럼, Scanner는 Input Program을 읽고 Token 으로 분리 하여 Parser로 전달합니다. 그런 다음 Parser는 Code Generator에서 처리된 Syntax Tree를 빌드 합니다. 컴파일러 모듈은 원하는 대로 설계할 수 있지만, 최소 Parsing Table는 만들어야합니다. Parser를 구현하기 위해 LL(1)이나 LR(1) parser를 빌드 할 수 있습니다.

구현 언어는 C, C++, C#, Python, Java 중 하나를 사용하시되, 이 이외에 다른 언어를 사용하시려면 사전에 TA에게 말해주세요. 여러분이 만든 Parser의 출력 내용을 Code Generator를 통해 Psuedo Code 로 작성하면 되는데, Psuedo Code 코드 생성을 위한 Instruction set 은 <표 2>에 주어져 있습니다.

▶ 입력

☆ 표 1에 주어진 문법에 맞게 작성된 Program File로 File 이름은 다음과 같이 한다. 즉 여러분이 만든 Compiler를 구동하기 위해서는

compiler2020 <input_file_name> (예를 들면, compiler2020 testfile)

를 입력하도록 한다.

▶ 출력

☆ 표 2에 주어진 Instruction Set을 이용한 Code File로, File의 이름에 Code 라는 Extension을 붙인다. 예를 들면, 상기한 입력의 File이 입력되면 그 출력 File 이름은 testfile.code가 된다. 또한 symbol table을 담은 testfile.symbol을 제출하기 바랍니다.

myProgram.code

▶ 처리 조건

☆ 프로그래밍 환경은 UNIX(Linus, AIX 등) 이며 구현 언어는 C, C++, Java, python 중 선택하기를 권장합니다. 만약, 다른 언어를 사용하기를 원하시면 사전에 TA에게 이야기해야 합니다.

☆ 목적 Code는 표 2 에 주어진 명령어(Instruction)들 만을 사용해야 합니다. 부득이한 경우 새로운 명령어를 사용할 때는 Document에 명확하게 명시해야 합니다.

☆ Function은 "BEGIN function_name"로 시작하고 "END function_name"로 종료합니다.

☆ Register는 충분히 많다고 가정하나 될 수 있으면 적은 숫자의 Register를 사용하도록 하고 사용된 Register의 개수를 Code의 마지막 줄에 출력하도록 합니다.

☆ 부득이한 사정으로 Grammar를 변경한 경우에는 Documentation 에 무엇을 어떻게 그리고 왜 변경했는지를 명시해야 합니다.

▶ 과제 제출물 < [CP]Project2_team#.zip (ex. [CP]Project2_team3.zip) >

1) Print-out of your Source Programs
2) Test input file and output file with a symbol table
3) Documentation(necessary and important)
5) Due Date : 2020. 12. 20. Sunday. 23.59. (이후 제출 시 하루에 20% 감점)

Table 1 < Grammar >

```
prog    ::=   word "("  ")" block ;
decls   ::=   decls decl
        | ;
decl    ::=   vtype word ";" ;
vtype   ::=   int | char
        | ;
block   ::=   "{" decls slist "}"
        | ;
slist   ::=   slist stat
        |     stat ;
stat    ::=   IF cond THEN block ELSE block
        |     WHILE cond block
        |     word "=" expr ";"
        |     EXIT expr ";"
        | ;
cond    ::=   expr ">" expr
        |     expr "==" expr ;
expr    ::=   term
        |     term "+" term ;
term    ::=   fact
        |     fact "*" fact ;
fact    ::=   num
        |     word ;
word    ::=   ([a-z] | [A-Z])* ;
num     ::=   [0-9]*
```

Table 2 < instruction set>

LD Reg#1, addr(or num)	Load var (or num) into the Reg#1
ST Reg#1, addr	Store value of Reg#1 into var
ADD Reg#1, Reg#2, Reg#3	Reg#1 = Reg#2 + Reg#3
MUL Reg#1, Reg#2, Reg#3	Reg#1 = Reg#2 x Reg#3
LT Reg#1, Reg#2, Reg#3	1 if (Reg#2 < Reg#3), 0 otherwise , store into Reg#1
JUMPF Reg#1 label	Jump to label if Reg#1 contains 0
JUMPT Reg#1 label	Jump to label if Reg#1 contains Non-0
JUMP label	Jump to label without condition