Design and Development of Compiler  
for C- Language  
(설계 프로젝트 수행 결과)

1. 각 단계별 결과 보고서

과목명: [CSE4120] 기초 컴파일러 구성

담당교수: 서강대학교 컴퓨터공학과 정 성 원

개발자: 20131612 최대운

개발기간: 2019. 3. 19. ~ 2019. 3. 26.

각 단계별 결과 보 고 서

프로젝트 제목: Design and Development of Compiler for C-Language:  
 Phase 1: Design and Implementation of Lexical Analyzer

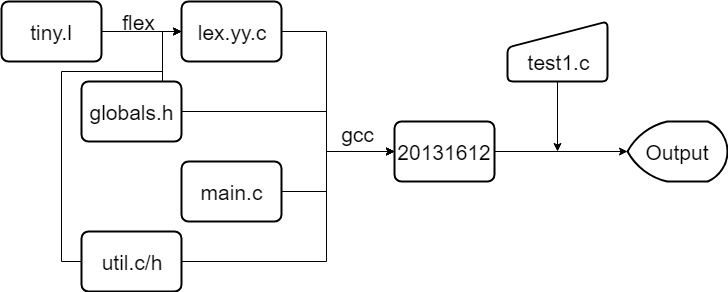
제출일: 2019. 3. 26.  
개발자: 20131612 최대운

1. 개발 목표  
   - Linux 환경에서 소스파일을 입력으로 넣으면 해당 소스파일에 있는 Token들을 분류하여 테이블 형태로 출력하는 프로그램을 제작한다.
2. 개발 범위 및 내용
   1. 개발 범위  
      - 교재에 제시된 스켈레톤 코드를 수정하여 Lexical Analyzer를 제작한다.
   2. 개발 내용  
      - Globals.h에 Enum값을 추가 및 수정하여 reserved word, ID, Value, Operator의 Token을 명시한다.  
      - tiny.l파일(lex파일)에 ID, Value가 될 수 있는 알파벳 및 숫자 집합을 Regular Expression으로 추가한다. Rule을 추가 및 수정하여 reserved word, ID, Value, Operator 등으로 구분 할 수 있게한다.  
      - Util.c의 printToken 함수를 각 Token에 맞게 출력할 수 있도록 수정한다.  
      - main 함수에서 입력으로 주어진 소스 파일의 끝까지 Parsing 하게 한다.
3. 추진 일정 및 개발 방법
   1. 추진 일정

|  |  |
| --- | --- |
| 일정 | TODO |
| 03. 20. | 프로젝트 명세서 파악 및 교재 정독 |
| 03. 22. | 개발 |
| 03. 24. | 개발 |
| 03. 25. | 수정 사항 최종 확인 |
| 03. 26. | 제출 |

* 1. 개발 방법  
     - 스켈레톤 코드는 책에 나와있는 코드를 사용한다.  
     (저자 홈페이지 참조, [**http://www.cs.sjsu.edu/~louden/cmptext**](http://www.cs.sjsu.edu/~louden/cmptext))  
     - 필요시 lex 문법을 매뉴얼에 찾아본다.  
     - Linux 환경은 cspro9를 이용해서 개발한다.

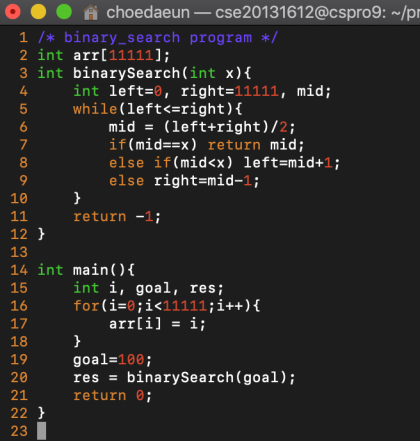
1. 연구 결과
   1. 기초 컴파일러 구성
      1. 합성 내용

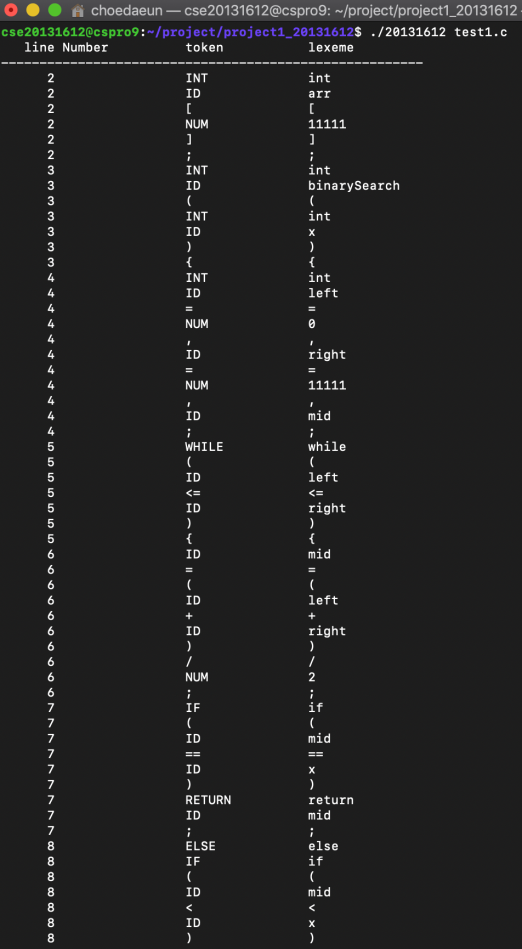
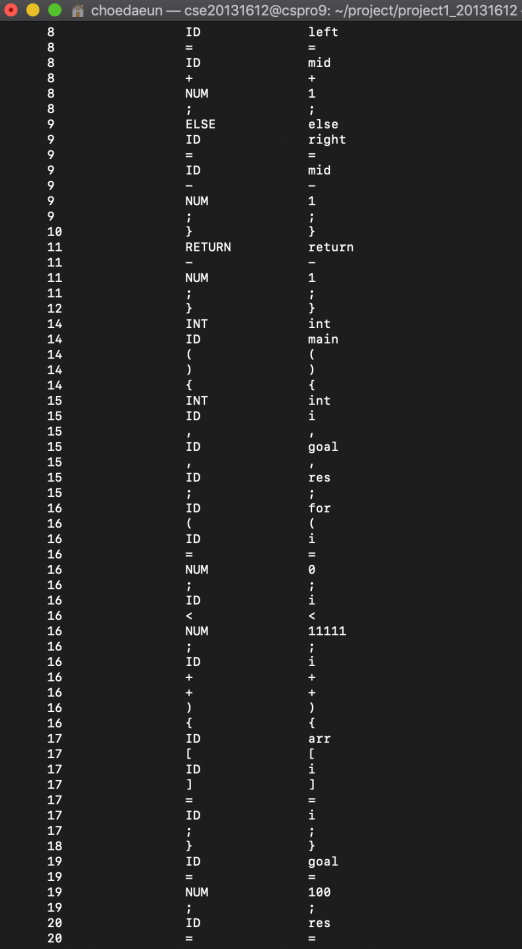
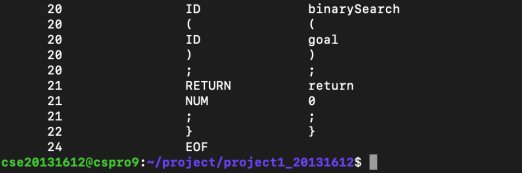


* + 1. 분석 내용  
       - Tokenize를 위한 Regular Expression은 다음과 같다.  
        digit [0-9]  
        number {digit}+  
        letter [a-zA-Z]  
        identifier {letter}+  
        newline \n  
        whitespace [ \t]+  
       - Lex파일에 추가한 Rule은 다음과 같다.

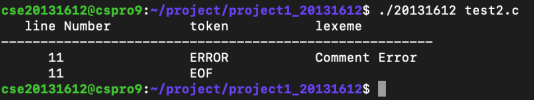
|  |  |
| --- | --- |
| Rule | Enum value |
| “if” | IF |
| “else” | ELSE |
| “int” | INT |
| “void” | VOID |
| “while” | WHILE |
| “return” | RETURN |
| “==” | EQ |
| “!=” | NOTEQ |
| “<=” | LTET |
| “>=” | GTET |
| “/\*” | 주석의 시작, “\*/”가 나올때까지 text consume. \*/가 나오지 않고 파일의 끝에 도달 하면 ERROR |
| “=” | ASSIGN |
| “<” | LT |
| “>” | GT |
| “+” | PLUS |
| “-“ | MINUS |
| “\*” | TIMES |
| “/” | OVER |
| “(“ | LPAREN |
| “)” | RPAREN |
| “{“ | LBRACE |
| “}” | RBRACE |
| “[“ | LSQBRACKET |
| “]” | RSQBRACKET |
| “;” | SEMI |
| “,” | COMMA |
| {number} | NUM |
| {identifier} | ID |
| {newline} | Increase line number |
| {whitespace} | Do nothing |

- /\* 토큰에 대한 처리는, /\* 토큰 이후에 “\*/”이 나올 때까지 중간의 모든 text를 consume한다. 만약 \*/가 나오지 않고 파일의 끝에 도달하게 되면 출력 테이블에 Comment error를 출력한다. 정상적으로 /\*가 처리된다면 출력 테이블에는 아무것도 출력하지 않고 line number만 증가하게된다.  
- Lex파일에 yywrap 함수를 추가한다. 특별한 기능은 없다.  
- globals.h에 위 Enum value를 추가한다.  
- util.c의 printToken 함수를 명세서에 제시된 형식에 맞춰 출력하도록 수정한다.  
- main함수에서 getToken 함수를 파일의 끝이 리턴 될 때까지 호출한다. getToken함수는 Lex 파일에 정의되어 있다.   
- getToken 함수 내부에서는 yylex 함수를 호출해 현재 어떤 token이 넘어 왔는지 알 수 있다. TraceScan 플래그는 활성화시켜 위에서 수정한 printToken함수를 통해 명세서에서 요구한 line number, token과 lexeme을 출력할 수 있다.

* + 1. 제작 내용  
       

위 예제(test1.c)에 대한 출력 결과는 아래와 같다.  
   




다음 예제(test2.c) 에 대한 출력 결과는 아래와 같다.  


* + 1. 시험 내용  
       제시된 입력 결과에 대해 정확한 출력 결과를 보인다. 다만, 이 프로젝트는 C- Language를 정의하고 진행한 프로젝트로 일반적으로 쓰이는 C언어에 비해 사용되는 예약어의 개수나 변수 명명 규칙이 다르기 때문에 예외상황에 취약하다.
    2. 평가 내용  
       본 프로젝트는 간단한 프로젝트로 매우 큰 용량의 파일이 아니라면 빠른 시간에 수행 가능하다.

1. 기타
   1. 자체 평가  
      교재와 프로젝트 명세서를 잘 활용해 수월하게 진행 할 수 있었다.
   2. 느낀점  
      기초적인 Lexical analyzer를 제작해보면서 컴파일러의 과정의 초반부를 이해할 수 있었다.