**컴파일러 Project 보고서**

**20171614 김세연**

**#13-1**

C언어에서 식별자는 숫자와 문자(알파벳, \_)로 구성되고, 문자가 맨 처음에 온다.

예제에서 렉스의 입력으로 식별자를 정의하고, 식별자가 나타나면 식별자에 대한 토큰 값을 리턴하는 입력을 작성해야 하므로 정의 부분과 변환규칙 부분을 작성하면 된다.

정의 부분은 이름(식별자)와 표현식(정규 표현)으로 구성되며, 변환 규칙 부분은 표현식(정규 표현)과 표현식과 매칭되었을 때 수행할 수행 코드(C언어)로 구성된다. 각 부분은 %%로 구분된다.

따라서 규칙에 맞게 작성한 렉스 입력 파일은 다음과 같다.

Letter [a-zA-Z\_]

Digit [0-9]

%%

{Letter}({Letter} | {Digit})\* return tident

**#13-2**

변환 규칙 부분은 표현식(정규 표현)과 표현식과 매칭되었을 때 수행할 수행 코드(C언어)로 구성된다. 숫자를 만나면 “found integer number”를 출력해야 하므로, 이를 정규 표현과 C언어로 나타내면 다음과 같다. (기호 + : 한번 이상의 반복을 나타냄)

[0-9]+ printf(“found integer number\n”)

**#13-3**

야크의 변환규칙 부분에서는 문법의 규칙을 정의한다. 변환 규칙 부분에 대한 표현은 일련의 문법 규칙으로 구성되고, 각 규칙은 왼쪽 부분과 몸체의 형태를 취한다. 왼쪽 부분은 논터미널의 이름이고, 몸체는 이름이나 상수 문자가 될 수 있다. BNF로 나타낸 문법에서 <expr>, <term>은 논터미널 기호이고, +는 터미널 기호이므로 야크로 변환하면 다음과 같다.

expr : expr ’+’ term

**#13-4**

야크에서는 모호한 문법에서 충돌을 해결하기 위해 우선순위를 제공한다. 야크에선 터미널 중 가장 낮은 우선순위부터 정의하며, 왼쪽 결합법칙인 경우에는 LEFT로 오른쪽 결합법칙인 경우에는 RIGHT로 표시한다.

예제에서 주어진 입력이 ‘=’, 그 다음 ‘+’과 ‘-‘, 마지막으로 ‘\*’과 ‘/’ 이므로 우선순위는 ‘=’< ‘+’, ‘-‘< ‘\*’, ‘/’ 이다. 또한 =은 오른쪽 결합법칙을 취하고, 나머지 터미널은 왼쪽 결합법칙을 취하므로 a=c\*d-e+f/g의 연산 순서를 괄호로 표기하면 다음과 같다.

(a=(((c\*d)-e)+(f/g)))

**#13-5**

플렉스 파일은 렉스의 입력 형식과 같이 정의 부분, 변환 규칙 부분, 사용자 부프로그램 부분을 작성한다. 이때 정의 부분은 이름(식별자)과 일련의 표현식(정규 표현)으로 구성되고, 변환 규칙 부분은 표현식과 일련의 수행 코드(C언어)로 나타난다. 사용자 부프로그램 부분에서는 변환 구칙 부분에서 사용되는 부프로그램이 사용자에 의해 작성된다(C언어).

변환 규칙 부분에서 토큰을 표현할 때 정규표현으로 패턴을 정의하기 위해 메타 기호를 사용한다

테이블이(가) 표시된 사진

자동 생성된 설명

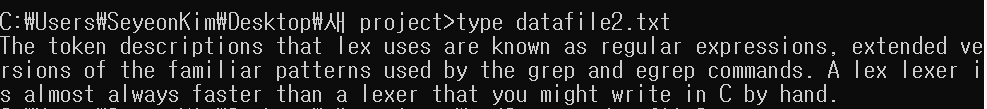
또, 렉스에서는 정보 제공과 복잡한 기능을 수행하기 위해 yytext(정규 표현에 의해 실제로 매칭된 문자열)과 같은 전역 변수나 함수(ex.yywrap())를 제공한다.

1. 위에 작성한 렉스 입력 형식에 맞춰 플렉스 입력 파일을 작성한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 데이터 파일을 작성한다



1. 실행

어휘분석기를 작성한 플렉스 파일로 생성한다.

텍스트이(가) 표시된 사진

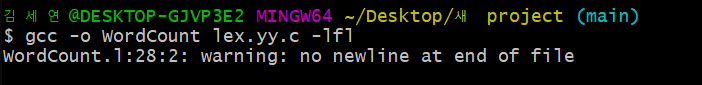
자동 생성된 설명

생성된 어휘분석기 lex.yy.c의 내용

텍스트이(가) 표시된 사진

자동 생성된 설명

어휘 분석기 실행



실행 파일이 생성된 것을 확인 할 수 있다.

1. 실행 결과

실행결과는 다음과 같다. 예제 풀이보다 글자수가 하나 적은데, 예제 풀이의 데이터파일은 첫 번째와 두번째 문장 사이에 공백이 2칸이라 차이가 생겼다. 단어 수, 라인 수는 예제의 결과와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**#13-6**

(설명은 13-5와 같다)

1. 렉스 파일 작성(test.l)

|  |
| --- |
| %{  #include <stdio.h>  #include <stdlib.h>  enum tnumber {TEOF,TIDEN, TNUM, TASSIGN, TADD, TSUBT, TMUL, TDIV, TSEMI, TBEGIN, TEND, TERROR};  %}  /\* 이름 n 치환식 \*/  letter [a-zA-Z]  digit [0-9]  %% /\*정의 부분\*/  "(" return(TBEGIN);  ")" return(TEND);  {letter}({letter}|{digit})\* return(TIDEN);  "=" return(TASSIGN);  "+" return(TADD);  "\*" return(TMUL);  "/" return(TDIV);  "-" return(TSUBT);  {digit}+ return(TNUM);  ";" return(TSEMI);  [ \t\n] ;  . return(TERROR);  %%/\*사용자 부프로그램 부분\*/  main(){  enum tnumber tn;  printf("Start of FLEX \n");    while((tn=yylex( )) != TEOF) switch (tn) {  case TBEGIN : printf("Left parenthesis \n"); break;  case TEND : printf("Right parenthesis \n"); break;  case TIDEN : printf("Identifier : %s\n", yytext); break;  case TASSIGN : printf("Assign\_op \n"); break;  case TADD : printf("Add\_op \n"); break;  case TMUL : printf("Mul\_op\n"); break;  case TDIV : printf("Div\_op\n"); break;  case TSUBT : printf("Subt\_op\n"); break;  case TNUM : printf("Number : %s\n",yytext); break;  case TSEMI : printf("Semicolon \n"); break;  case TERROR : printf("Error \n"); break;  }  yywrap( ); {printf("End of FLEX\n"); return 1;}  } |

1. 어휘 분석기 생성 및 생성한 어휘 분석기로 실행 파일 생성

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 실행 결과

텍스트이(가) 표시된 사진

자동 생성된 설명

(datafile.txt.의 내용 : ni = ba \* po – 60 + ni / (abc + 50); )

1. **손으로 어휘 분석한 결과 및 실제 실행한 결과 정리된 표**

|  |  |  |
| --- | --- | --- |
| 손으로 어휘 분석 | | 실제 실행 |
| 토큰 | 렉심 |
|  |  | Start of FLEX |
| identifier | ni | Identifier : ni |
| assign\_op | = | Assign\_op |
| identifier | ba | Identifier : ba |
| mul\_op | \* | Mul\_op |
| identifier | po | Identifier : po |
| subt | - | Subt\_op |
| number | 60 | Number : 60 |
| add\_op | + | Add\_op |
| identifier | ni | Identifier : ni |
| div\_op | / | Div\_op |
| left parenthesis | { | Left parenthesis |
| identifier | abc | Identifier : abc |
| add\_op | + | Add\_op |
| number | 50 | Number : 50 |
| right parenthesis | ) | Right parenthesis |
| semicolon | ; | Semicolon |
|  |  | End of FLEX |

손으로 분석한 결과와 플렉스로 만든 어휘분석기 결과가 같음을 알 수 있다.

**#13-7**

렉스 파일 작성법은 13-5에서 설명한 것과 같다.

야크 입력 파일은 선언 부분, 변환규칙 부분, 사용자 프로그램 부분으로 구성된다. 선언부분에선 %{와 %}사이에 시작기호를 정의하며, 사용할 헤더도 정의한다. 또 %token으로 어휘 분석기에서 반환되는 모든 토큰에 대해 정의한다. 변환 규칙 부분에서는 변환 규칙 부분에 대한 표현은 일련의 문법 규칙으로 구성되고, 각 규칙은 왼쪽 부분과 몸체의 형태를 취한다. 왼쪽 부분은 논터미널의 이름이고, 몸체는 이름이나 상수 문자가 될 수 있다. 마지막으로 사용자 프로그램 부분은 야크를 보조하는 에러 처리 루틴(C언어) 등을 작성한다.

1. 플렉스(렉스), 바이슨(야크) 파일을 작성한다. 이때 플렉스 파일 작성시 #include "y.tab.h“를 추가해 생성될 실행 파일이 #include "y.tab.h“ 코드를 포함하도록 한다. 또 바이슨 파일 작성시 바이슨에서 만들어진 파서 함수는 yyparse()로 고정되어 있고, 파싱에 성공하면 0을 리턴한다.

TT.L 파일 (플렉스)

|  |
| --- |
| %{  #include "y.tab.h“  %}  letter [A-Za-z]  digit [0-9]  id {letter}({letter}|{digit})\*  %% /\*토큰 정의 규칙\*/  "=" return(yytext[0]);  "\*" return(yytext[0]);  {id} return(ID);  [ \n\t\b]; |

TT.Y (바이슨)

|  |
| --- |
| %{  %}  %token ID  %%  S:L '=' R  ｜R;  L:'\*'R  ｜ID;  R:L;  %%  main()  {  if(yyparse()==0)  {  printf("The Parsing Complete \n");  }  else  {  printf("syntax error \n");  }  } |

1. 먼저 플렉스를 실행해 어휘 분석기를 생성한다(lex.xx.y) 그다음 바이슨을 실행해 파싱표 (y.output)과 구문분석기(y.tab.c) 가 생성되도록 하고 -d옵션으로 y.tab.h가 생성되도록 한다. 마지막으로 플렉스 라이브러리와 바이슨 라이브러리를 링크해 컴파일해 실행파일을 생성한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

테스트를 위한 데이터 파일 내용:

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 파싱 결과 : 예제와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

4. 파싱표 결과

y.output (바이슨이 생성한 파싱표)

|  |
| --- |
| Grammar  0 $accept: S $end  1 S: L '=' R  2 | R  3 L: '\*' R  4 | ID  5 R: L  Terminals, with rules where they appear  $end (0) 0  '\*' (42) 3  '=' (61) 1  error (256)  ID (258) 4  Nonterminals, with rules where they appear  $accept (6)  on left: 0  S (7)  on left: 1 2, on right: 0  L (8)  on left: 3 4, on right: 1 5  R (9)  on left: 5, on right: 1 2 3  state 0  0 $accept: . S $end  ID shift, and go to state 1  '\*' shift, and go to state 2  S go to state 3  L go to state 4  R go to state 5  state 1  4 L: ID .  $default reduce using rule 4 (L)  state 2  3 L: '\*' . R  ID shift, and go to state 1  '\*' shift, and go to state 2  L go to state 6  R go to state 7  state 3  0 $accept: S . $end  $end shift, and go to state 8  state 4  1 S: L . '=' R  5 R: L .  '=' shift, and go to state 9  $default reduce using rule 5 (R)  state 5  2 S: R .  $default reduce using rule 2 (S)  state 6  5 R: L .  $default reduce using rule 5 (R)  state 7  3 L: '\*' R .  $default reduce using rule 3 (L)  state 8  0 $accept: S $end .  $default accept  state 9  1 S: L '=' . R  ID shift, and go to state 1  '\*' shift, and go to state 2  L go to state 6  R go to state 10  state 10  1 S: L '=' R .  $default reduce using rule 1 (S) |

y.output을 표로 정리한 파싱표는 다음과 같다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 상태 | 구문 분석기의 행동 | | | | GOTO 함수 | | |
| = | \* | id | $ | S | R | L |
| 0 |  | s2 | s1 |  | 3 | 5 | 4 |
| 1 | r4 |  |  | r4 |  |  |  |
| 2 |  | s2 | s1 |  |  | 7 | 6 |
| 3 |  |  |  | s8 |  |  |  |
| 4 | s9,r5 |  |  | r5 |  |  |  |
| 5 |  |  |  | r2 |  |  |  |
| 7 | r5 |  |  | r5 |  |  |  |
| 8 | r3 |  |  | r3 |  |  |  |
| 9 |  |  |  | acc |  | 10 | 6 |
| 10 | r1 |  |  | r1 |  |  |  |

**#13-8**

설명은 13-7과 같다.

1. 플렉스(렉스), 바이슨(야크) 파일을 작성한다. 이때 플렉스 파일 작성시 #include "y.tab.h“를 추가해 생성될 실행 파일이 #include "y.tab.h“ 코드를 포함하도록 한다. 또 바이슨 파일 작성시 바이슨에서 만들어진 파서 함수는 yyparse()로 고정되어 있고, 파싱에 성공하면 0을 리턴한다.

**TT.L**

|  |
| --- |
| %{  #include "y.tab.h"  %}  letter [A-Za-z]  digit [0-9]  id {letter}({letter}|{digit})\*  %% /\*토큰 정의 규칙\*/  "(" return(yytext[0]);  "\*" return(yytext[0]);  ")" return(yytext[0]);  "+" return(yytext[0]);  {id} return(ID);  [ \n\t\b] ; |

**TT.Y**

|  |
| --- |
| %{  %}  %token ID  %%  E:E'+'T  |T;  T:T'\*'F  |F;  F:'('E')'  |ID;  %%  main()  {  if(yyparse()==0)  {  printf("The Parsing Complete \n");  }  else  {  printf("syntax error \n");  }  } |

1. 실행. 먼저 플렉스를 실행해 어휘 분석기를 생성한다(lex.xx.y) 그다음 바이슨을 실행해 파싱표 (y.output)과 구문분석기(y.tab.c) 가 생성되도록 하고 -d옵션으로 y.tab.h가 생성되도록 한다. 마지막으로 플렉스 라이브러리와 바이슨 라이브러리를 링크해 컴파일해 실행파일을 생성한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 테스트를 위해 작성한 데이터 파일과 실행결과 : 예제풀이와 결과가 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 파싱 표

y.output

|  |
| --- |
| Grammar  0 $accept: E $end  1 E: E '+' T  2 | T  3 T: T '\*' F  4 | F  5 F: '(' E ')'  6 | ID  Terminals, with rules where they appear  $end (0) 0  '(' (40) 5  ')' (41) 5  '\*' (42) 3  '+' (43) 1  error (256)  ID (258) 6  Nonterminals, with rules where they appear  $accept (8)  on left: 0  E (9)  on left: 1 2, on right: 0 1 5  T (10)  on left: 3 4, on right: 1 2 3  F (11)  on left: 5 6, on right: 3 4  state 0  0 $accept: . E $end  ID shift, and go to state 1  '(' shift, and go to state 2  E go to state 3  T go to state 4  F go to state 5  state 1  6 F: ID .  $default reduce using rule 6 (F)  state 2  5 F: '(' . E ')'  ID shift, and go to state 1  '(' shift, and go to state 2  E go to state 6  T go to state 4  F go to state 5  state 3  0 $accept: E . $end  1 E: E . '+' T  $end shift, and go to state 7  '+' shift, and go to state 8  state 4  2 E: T .  3 T: T . '\*' F  '\*' shift, and go to state 9  $default reduce using rule 2 (E)  state 5  4 T: F .  $default reduce using rule 4 (T)  state 6  1 E: E . '+' T  5 F: '(' E . ')'  '+' shift, and go to state 8  ')' shift, and go to state 10  state 7  0 $accept: E $end .  $default accept  state 8  1 E: E '+' . T  ID shift, and go to state 1  '(' shift, and go to state 2  T go to state 11  F go to state 5  state 9  3 T: T '\*' . F  ID shift, and go to state 1  '(' shift, and go to state 2  F go to state 12  state 10  5 F: '(' E ')' .  $default reduce using rule 5 (F)  state 11  1 E: E '+' T .  3 T: T . '\*' F  '\*' shift, and go to state 9  $default reduce using rule 1 (E)  state 12  3 T: T '\*' F .  $default reduce using rule 3 (T) |

y.output을 정리한 파싱표

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 상태 | 구문 분석기의 행동 | | | | | | GOTO 함수 | | |
| id | + | \* | ( | ) | $ | E | T | F |
| 0 | s1 |  |  | s2 |  |  | 3 | 4 | 5 |
| 1 |  | r6 | r6 |  | r6 | r6 |  |  |  |
| 2 | s1 |  |  | s2 |  |  | 6 | 4 | 5 |
| 3 |  | s8 |  |  |  | s7 |  |  |  |
| 4 |  | r2 | s9 |  | r2 | r2 |  |  |  |
| 5 |  | r4 | r4 |  | r4 | r4 |  |  |  |
| 6 |  | s8 |  |  | s10 |  |  |  |  |
| 7 |  |  |  |  |  | acc |  |  |  |
| 8 | s1 |  |  | s2 |  |  |  | 11 | 5 |
| 9 | s1 |  |  | s2 |  |  |  |  | 12 |
| 10 |  | r5 | r5 |  | r5 | r5 |  |  |  |
| 11 |  | r1 | s9 |  | r1 | r1 |  |  |  |
| 12 |  | r3 | r3 |  | r3 | r3 |  |  |  |

**#13-9**

설명은 13-7과 같다.

1. 플렉스(렉스), 바이슨(야크) 파일을 작성한다. 바이슨 파일 작성시 바이슨에서 만들어진 파서 함수는 yyparse()로 고정되어 있고, 파싱에 성공하면 0을 리턴한다. 야크 파일에서 사용자 프로그램 부분에 에러 처리 루틴인 yyerror함수를 추가했다.

**TT.L**

|  |
| --- |
| %{  %}  %% /\*토큰 정의 규칙\*/  "c" return(yytext[0]);  "d" return(yytext[0]);  [ \n\t\b] ; |

**TT.Y**

|  |
| --- |
| %{  %}  %%  S:C C;  C :'c'C  |'d';  %%  main()  {  yyparse();  printf("The Parsing Complete \n");  }  yyerror()  {  printf("syntax error \n");  exit(0);  } |

1. **플렉스와 바이슨으로 어휘분석기, 구분 분석기, 파싱표, 헤더 생성 후 실행파일 컴파일**

텍스트, 모니터, 화면, 닫기이(가) 표시된 사진

자동 생성된 설명

1. **데이터파일 내용과 실행 결과: 예제 풀이와 같은 결과가 나왔다.**

텍스트이(가) 표시된 사진

자동 생성된 설명

**4. 파싱표**

**y.output**

|  |
| --- |
| Grammar  0 $accept: S $end  1 S: C C  2 C: 'c' C  3 | 'd'  Terminals, with rules where they appear  $end (0) 0  'c' (99) 2  'd' (100) 3  error (256)  Nonterminals, with rules where they appear  $accept (5)  on left: 0  S (6)  on left: 1, on right: 0  C (7)  on left: 2 3, on right: 1 2  state 0  0 $accept: . S $end  'c' shift, and go to state 1  'd' shift, and go to state 2  S go to state 3  C go to state 4  state 1  2 C: 'c' . C  'c' shift, and go to state 1  'd' shift, and go to state 2  C go to state 5  state 2  3 C: 'd' .  $default reduce using rule 3 (C)  state 3  0 $accept: S . $end  $end shift, and go to state 6  state 4  1 S: C . C  'c' shift, and go to state 1  'd' shift, and go to state 2  C go to state 7  state 5  2 C: 'c' C .  $default reduce using rule 2 (C)  state 6  0 $accept: S $end .  $default accept  state 7  1 S: C C .  $default reduce using rule 1 (S) |

**파싱표**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| state | action | | | GOTO | |
| c | d | $ | S | C |
| 0 | s1 | s2 |  | 3 | 4 |
| 1 | s1 | s2 |  |  | 5 |
| 2 | r3 | r3 | r3 |  |  |
| 3 |  |  | s6 |  |  |
| 4 | s1 | s2 |  |  | 7 |
| 5 | r2 | r2 | r2 |  |  |
| 6 |  |  | acc |  |  |
| 7 | r1 | r1 | r1 |  |  |

**#13-10**

설명은 13-7과 같다.

1. **렉스, 야크 파일 작성**

**in2po.l**

|  |
| --- |
| %{  #include "in2po.tab.h"  %}  %%  [ \t]+ ;  [0-9] { yylval = yytext[0] - '0'; return DIGIT; }  [+\-\n] return yytext[0]; |

**in2po.y**

|  |
| --- |
| %{  #include <stdio.h>  #include <ctype.h>  %}  %token DIGIT  %%  line : expr '\n' { putchar('\n'); }  ;  expr : expr '+' term { putchar('+'); }  | expr '-' term { putchar('-'); }  | term  ;  term : DIGIT { printf("%d", yylval); }  ;  %%  int main()  {  if (yyparse() == 0) printf("파싱 성공! \n\n");  else printf("파싱 실패 \n\n");  } |

1. **플렉스와 바이슨으로 어휘분석기, 구분 분석기, 헤더 생성뒤 컴파일 해 실행파일 생성** (바이슨으로 구문분석기 생성시 -y 옵션을 제외해 파싱표가 생성되지 않았다)

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **실행 결과 : 중위표현식이 후위 표현식으로 잘 변환 되고, 성공 메시지가 리턴하는 것을 확인**

텍스트이(가) 표시된 사진

자동 생성된 설명