Project 1

Lexical analysis

Virtual Machine & Optimization Laboratory
Dept. of Electrical and Computer Engineering
Seoul National University

Project plan

1. Lexical analysis
2. Yacc programming
3. Semantic analysis
4. Code generation

Specification

목표

- lex utility를 이용하여 C+-(새롭게 정의한 C spec; subc라고 부름)에 대한 lexical analyzer를 만든다
- lexical analyzer는 token을 모두 찾아내서 결과를 화면에 출력한다

입력

- subc 프로그램
- nested comments와 **..**연산자를 지원
  - `/* comment_body1 /* comment_body2 */ */`
  - `1..5`

출력

- subc 프로그램의 각 token에 대해서 token의 **종류**(e.g. OP, KEY, INT), **값**(e.g. +, struct, 1), 프로그램에서 나타난 **횟수**
- 횟수는 identifier나 keyword인 경우에만 출력

Expected result

Input

```
/*************************
   /* nested comments*/

*************************/
struct _point {
    float x, y, z;
    int color;
} point[20];

struct _line {
    struct _point *p[2];
    int color;
} line[20];
```

Output

```
KEY    struct     1
ID     _point     1
OP     {
KEY    float      1
ID     x          1
OP     ,
ID     y          1
OP     ,
ID     z          1
OP     ;
…
```

subc.l

subc.l 작성

- Lexical analyzer의 spec을 정의
- flex로 compile하여 lexical analyzer에 해당하는 C 파일을 생성
  - 컴파일: flex subc.l
  - 결과 파일: lex.yy.c
- %% 연산자를 기준으로 세 영역으로 구분

```
declarations
%%
translation rules
%%
auxiliary procedures
```

- %{ … %} 안의 코드는 lex에 의해 생성되는 lex.yy.c 파일에 그대로 dump된다

subc.l

Declarations

- 형태: `name  definition`
- 사용: `digit [0-9]`
- definition: name에 해당하는 regular expression
- start condition을 정의
  - starting condition: 특정 rule을 활성화 하는 condition

```
%{
#include "subc.h"
/**************************************************************
{Other useful code segments can be here.}
**************************************************************/
int commentdepth=0;
%}

letter      [A-Za-z_]
%start normal comment
%%
```

subc.l

Translation rules

- 형태: `<cond>pattern  action`
- 사용: `<AA>{digit}  {printf("DIGIT\t%s",yytext);}`
- pattern: declaration에서 선언한 name과 lex의 특수기호들을 조합한 regular expression
- action: pattern을 찾았을 때 수행할 C코드
- 가장 핵심적인 부분, 나머지는 생략할 수 있지만 translation rules 파트는 반드시 작성해야 한다

Auxiliary procedures

- main함수 및 추가적으로 필요한 함수들을 정의

Global variables

- yytext: lex가 토큰을 인식하는 과정에서 임시로 저장하는 버퍼
- yyleng: yytext의 길이

subc.l

executable file 생성

- 생성된 lex.yy.c 파일을 다시 gcc로 컴파일
- 이 때 flex를 link 해주어야 한다 (-lfl)
- `gcc -o subc lex.yy.c -lfl`

생성된 executable의 실행

- `./subc`
- `./subc input.c`

Examples

- lex_example 폴더 예제 참조

Hash table

Hash table

- subc.h, hash.c에 인터페이스 함수를 구현
- `id* enter(int tokenType, char* name, int length);`

Hash table을 얼마나 잘 구현했는지는 프로그램의 오작동을 일으키지만 않으면 평가대상이 아님

subc.l의 맨 처음 %{ … %}부분에 subc.h를 include

Environment

Docker

- 프로젝트는 Docker container 로 진행될 예정
- Docker 는 다른 프로세스들로부터 독립된 실행 환경을 제공
- 자세한 세팅 방법은 프로젝트 문서 참조

Docker 를 사용 불가능한 경우

- 아래 패키지가 정상적으로 동작하는 어떠한 환경에서라도 진행 가능
  - Flex
  - GNU Bison
  - GCC
  - GNU Make
  - Git

Makefile

Makefile을 제공

- Source/header 를 추가함에 따라 수정해도 됨.
- 그러나, 반드시 `make all` 커맨드로 빌드 가능해야 함.

```
subc: lex.yy.o hash.o
    gcc -o subc lex.yy.o hash.o -lfl

lex.yy.o: lex.yy.c
    gcc -c -g lex.yy.c

lex.yy.c: subc.l
    flex subc.l

hash.o: hash.c subc.h
    gcc -c -g hash.c

clean:
    rm -f lex.yy.c
    rm -f *.o
    rm -f subc
```

Tips

Nested comments

- start condition을 활용
- regular expression으로 표현될 수 없으므로 lex의 start condition을 이용해서 normal mode와 comment mode 변경

.. 연산자

- lookahead operator를 활용

식별자(Identifier) 규칙

- 영문 대소문자, 숫자, 언더스코어(_)만 사용할 수 있다.
- 키워드를 식별자로 사용할 수 없다.
- 숫자로 시작될 수 없으며, 반드시 영문자나 언더스코어(_)로 시작 되어야 한다.

Tips

Translation rule

- pattern은 반드시 line의 맨 처음부터 시작해야 하며, 들여쓰기가 허용되지 않는다.
- action의 시작은 해당하는 pattern과 같은 줄에서 시작한다. action의 body를 여러 줄로 정의하는 것은 상관없다.

ex)

```
{integer} {
    printf(“int”);
}
```

Ambiguity

Integer

- 01과 같은 입력은 없다.
- + 또는 - 부호를 포함하지 않는다.

Float

- .123과 같은 입력은 없다.
- 무조건 소수점 .을 포함하고 소수점 이후에 숫자가 오지 않아도 된다.
- + 또는 - 부호를 포함하지 않는다.

C++ template

- 키워드 template은 없으므로 `foo<bar<int>>`와 같은 입력은 없다.

Submission

제출 기한

- Oct 9, 2024

제출 방법

- etl.snu.ac.kr을 통해서 제출

제출 파일

- Makefile을 포함한 subc.l, subc.h, hash.c 등 소스를 zip으로 압축
- 파일명: project1_학번.zip (학번 format은 20xx-xxxxx)

Notice

수업 게시판 확인

- 수정 또는 추가되는 사항은 항상 게시판을 통하여 공지

Output 포맷 지키기 (파일 이름, 출력)

소스코드에 주석달기

Cheating 금지 (F처리, 모든 코드 철저히 검사)

TA

- 이상현 (sanghyeon@snu.ac.kr)