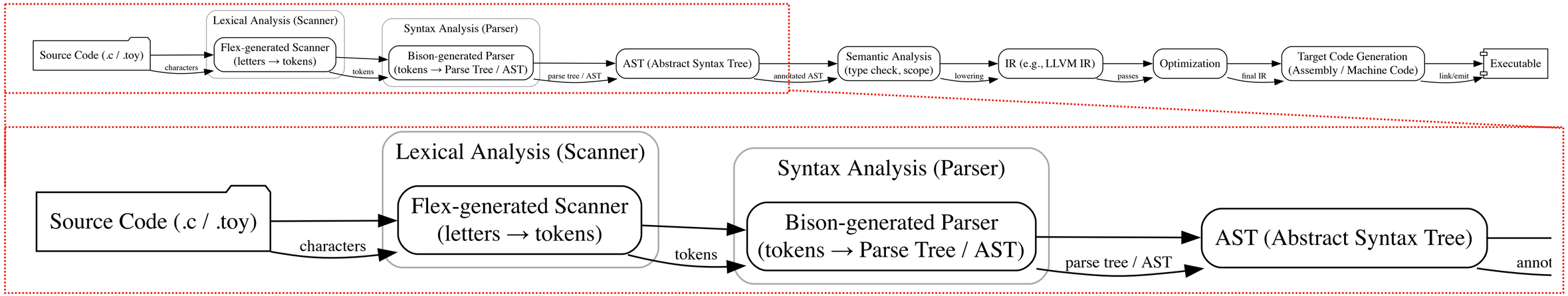
FLEX & BISON

본격적인 프로그래밍 언어 개발을 위한 도구 설명

경민기, 2025-09-23

프로그래밍 언어 개발



- 프로그래밍 언어는 크게 4단계로 구성됨
- lex → parse → AST → 코드생성
  - Lex: 소스코드를 문자 단위로 읽어 **토큰(token)** 으로 쪼갬
  - Parse: 토큰들을 문법 규칙에 따라 조합 → **구문 트리(Parse Tree) / 추상 구문 트리(AST)** 생성
  - AST
  - 코드 생성

FLEX
(fast lexical analyzer generator)

flex (fast lexical analyzer generator)

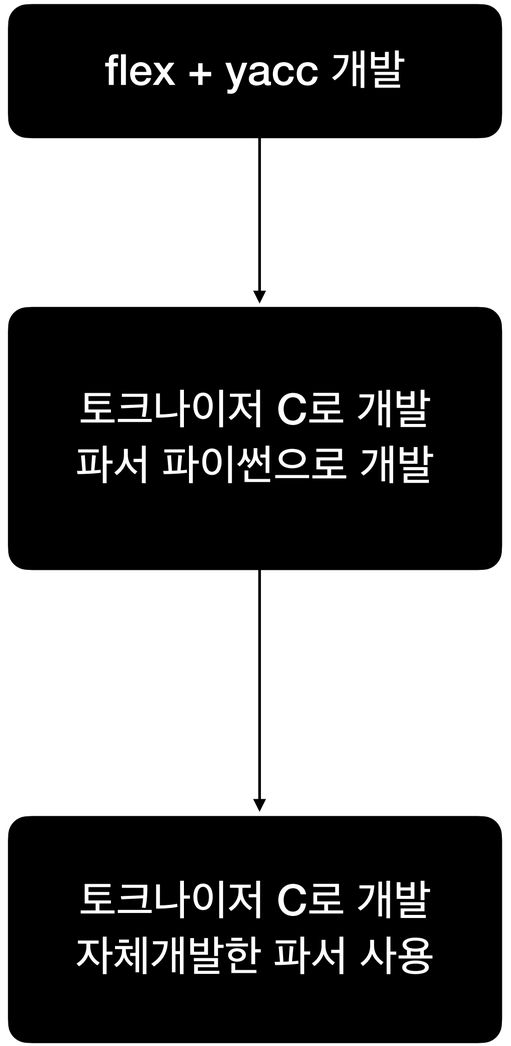
scanner generator for lexing in C and C++

- 사람이 일일이 getc() 돌려가며 토큰 파싱 코드를 짜는 대신, 패턴만 적으면 자동으로 코드 생성을 해 주는 도구
  - 1987년에 제작되고, 최신 ‘안정(stable)’ 릴리즈는 2017년
  - https://github.com/westes/flex
- 사용이유
  - 컴파일러/인터프리터 만들기
  - DSL(Domain-Specific Language) 구현
  - 텍스트 분석/필터링

프로그램 언어를 개발하는 방법

파이썬의 경우

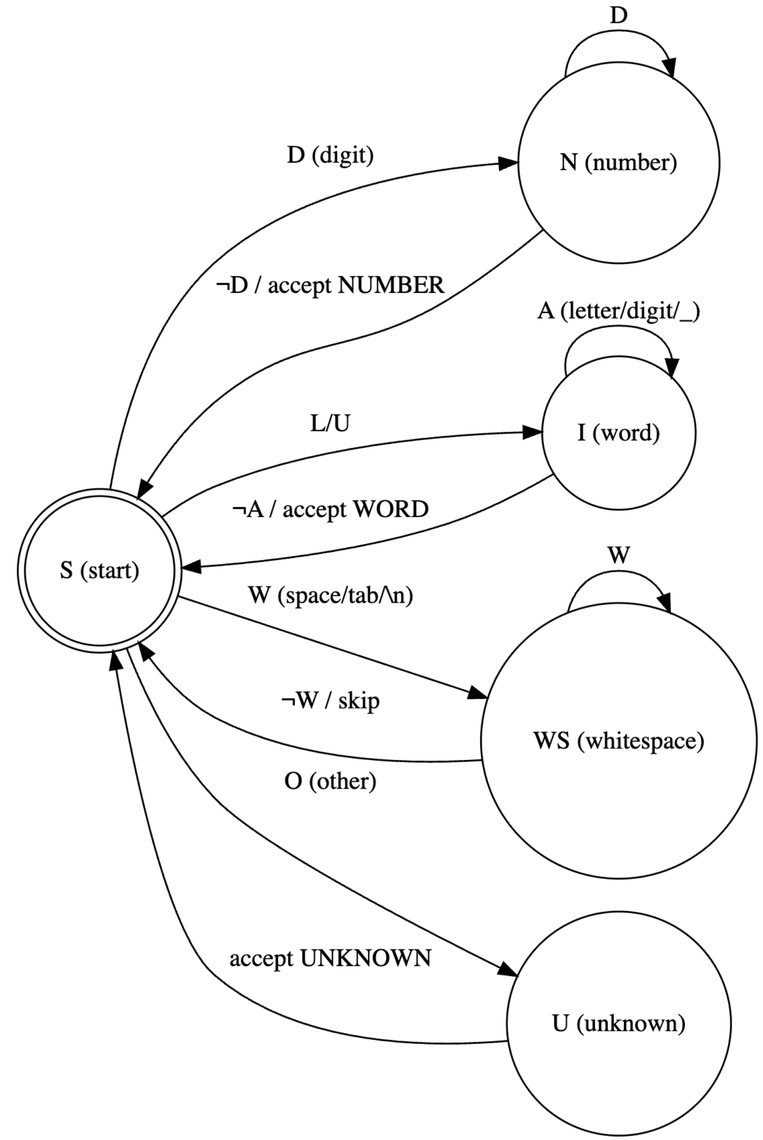
- 초기: Flex + Yacc 스타일 기반
- Python 1.x 이후:
  - 토크나이저는 직접 작성된 C 코드 (tokenizer.c)
  - 파서는 자체 개발한 pgen (Python Grammar Generator)
- Python 3.9 (2020)부터:
  - pgen을 버리고 PEG 기반 파서(pegen)로 교체
  - 이유: PEG(Parsing Expression Grammar)는 Python 같은 복잡한 문법(특히 f-string, 복잡한 comprehension 구문 등)을 표현하기 쉽기 때문



흥미로운 예외: Pypy
Python으로 구현된 Python 언어 구현체

- 먼저, RPython이라는, 기존 Python 문법을 엄격하게 만들어 컴파일이 가능토록한 언어를 만든다.
- 그리고, 이 RPython을 해석할수 있는 해석기(translate.py)를 일반적인 Python 코드로 작성한다.
- RPython의 효과적인 컴파일을 위해 다른 언어로 툴체인을 만든다.
- 전체 Python 구현(런타임)을 RPython 문법으로 작성한다.
- RPython으로 만든 Python 구현체를 앞서 순수 Python으로 만든 해석기로 컴파일한다.
- 이렇게 만들어진 구현체 후보를 이전 또는 다른 구현과 비교(성능 측정)해보고, 만족스럽지 않으면 수정한다.(nightly builds)
- 이렇게 하여 만족스러운 결과를 냈다면 출시하고, 출시한 구현체를 활용해 이 과정을 처음부터 다시 시작한다.(release)

예제 만들기


D
N (number)
D (digit)
¬D / accept NUMBER
A (letter/digit/_)
I (word)
L/U
¬A / accept WORD
S (start)
W
W (space/tab/\n)
WS (whitespace)
¬W / skip
O (other)
accept UNKNOWN
U (unknown)

FLEX 예제

기본적인 예제를 입력하고 실행하기

```
%option noyywrap

%{
#include <stdio.h>
%}

%%

[0-9]+              { printf("NUMBER: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]*   { printf("WORD: %s\n", yytext); }
[ \t\n]+            ;   /* 공백/개행 무시 */
.                   { printf("UNKNOWN: %s\n", yytext); }

%%

int main(void) {
    yylex();
    return 0;
}
```

코드 설명 (1) - 정의부

번역 안 하는 부분

```
%option noyywrap

%{
#include <stdio.h>
%}
```

- %option noyywrap
  - Flex가 만들어주는 기본 스캐너 함수는 yylex()인데, 입력이 끝(EOF)에 도달하면 자동으로 yywrap() 이라는 함수를 호출
  - Flex 스캐너에서 yywrap() 함수를 호출하지 않도록 하는 옵션
  - BISON 과 연동 안하기 때문에 쓰는 코드

- 정의부: %{ 부터 %}까지의 내용
  - 바로 C

코드 설명 (2) - 규칙부, 숫자

```
%%

[0-9]+              { printf("NUMBER: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]*   { printf("WORD: %s\n", yytext); }
[ \t\n]+            ;   /* 공백/개행 무시 */
.                   { printf("UNKNOWN: %s\n", yytext); }

%%
```

- [0-9]+ { printf("NUMBER: %s\n", yytext); }
  - [0-9]+ → 정규표현식, “숫자 하나 이상”
  - yytext → Flex가 자동으로 제공하는 현재 매칭된 문자열
  - 액션: 해당 문자열을 NUMBER: ... 형식으로 출력 flex 코드 복사

코드 설명 (3) - 규칙부, 문자

```
%%

[0-9]+              { printf("NUMBER: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]*   { printf("WORD: %s\n", yytext); }
[ \t\n]+            ;   /* 공백/개행 무시 */
.                   { printf("UNKNOWN: %s\n", yytext); }

%%
```

- [a-zA-Z_][a-zA-Z0-9_]* { printf("WORD: %s\n", yytext); }
  - 첫 글자: 알파벳이나 _
  - 나머지 글자: 알파벳/숫자/_ 가능
  - 액션: WORD: ... 출력

코드 설명 (4) - 규칙부, 공백문자

```
%%

[0-9]+              { printf("NUMBER: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]*   { printf("WORD: %s\n", yytext); }
[ \t\n]+            ;   /* 공백/개행 무시 */
.                   { printf("UNKNOWN: %s\n", yytext); }

%%
```

- [\t\n]+ ; /* 공백/개행 무시 */
  - 공백( ), 탭(\t), 개행(\n)이 하나 이상 나올 때
  - 액션 { ; } → 아무 것도 안 함 (즉 무시)
  - 코멘트(/* ... */)는 Flex 문법이 아니라 C 스타일 코멘트라서 그냥 주석임

코드 설명 (5) - 규칙부, 기타문자

```
%%

[0-9]+              { printf("NUMBER: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]*   { printf("WORD: %s\n", yytext); }
[ \t\n]+            ;   /* 공백/개행 무시 */
.                   { printf("UNKNOWN: %s\n", yytext); }

%%
```

- . { printf("UNKNOWN: %s\n", yytext); }
  - . → 임의의 문자 하나
  - 액션 { ; } → 아무 것도 안 함 (즉 무시)
  - 위에서 걸리지 않은 문자들을 전부 UNKNOWN으로 출력

코드 설명 (6) - 사용자코드

```
int main(void) {
    yylex();
    return 0;
}
```

- yylex()
  - yylex() → Flex가 생성하는 스캐너 함수
  - 입력을 한 글자씩 읽어오고 규칙(정규식)과 매칭되면 대응하는 액션을 실행
  - EOF(파일 끝)까지 반복

예제 실행

```
(base) mingi_kyung@MacBook-Pro simple_flex % ls
scanner.l
(base) mingi_kyung@MacBook-Pro simple_flex % flex scanner.l
(base) mingi_kyung@MacBook-Pro simple_flex % ls
lex.yy.c        scanner.l
(base) mingi_kyung@MacBook-Pro simple_flex % gcc -o scanner lex.yy.c
(base) mingi_kyung@MacBook-Pro simple_flex % ls
lex.yy.c        scanner         scanner.l
(base) mingi_kyung@MacBook-Pro simple_flex % echo "hello 123 world42" | ./scanner
WORD: hello
NUMBER: 123
WORD: world42
(base) mingi_kyung@MacBook-Pro simple_flex %
```

추가해볼 내용

- 주석을 추가해봅시다

BISON

Bison

- 1980년대 초, 리처드 스톨만이 GNU 툴체인을 만들기 시작했을 때, 기존의 Yacc는 유닉스에 묶여 있어서 자유롭게 배포할 수 없었음
- 그래서 GNU 프로젝트에서 만든 Yacc(Yet Another Compiler Compiler) 의 자유 소프트웨어 대체품인 bison을 만듬
- https://www.gnu.org/software/bison/
- Yacc(야크) → Bison(들소)

예제

- Flex(scanner.l):
  - 입력 "x = 1+2" → 토큰 IDENT(x) '=' NUMBER(1) '+' NUMBER(2)
    Bison(parser.y): 규칙 IDENT '=' expr와 매칭 액션: setvar("x", 3.0) 실행 출력: x = 3 이후 x + 4 입력 시 getvar("x") 호출 → 결과 7

scanner.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "parser.tab.h"
%}
%option noyywrap

%%

[0-9]+(\.[0-9]+)?      { yylval.num = strtod(yytext, NULL); return NUMBER; }
[a-zA-Z_][a-zA-Z0-9_]* { yylval.id = strdup(yytext); return IDENT; }

[ \t]+                 ;     /* 공백 무시 */
\n                     { return '\n'; }

"=="                   { return '='; }  /* 방어: 실수로 == 써도 =로 처리 */
"="                    { return '='; }
"("                    { return '('; }
")"                    { return ')'; }
"+"|"-"|"*"|"/"        { return yytext[0]; }

.                      { /* 알 수 없는 문자 */
                         fprintf(stderr, "unknown char: '%s'\n", yytext);
                       }

%%
```

scanner.l 정의부

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "parser.tab.h"
%}
%option noyywrap
```

- #include "parser.tab.h"
  - Bison이 생성한 토큰 심볼(예: NUMBER, IDENT)과 %union 타입(여기서는 num, id)을 가져옴
  - 반드시 Bison으로 헤더를 먼저 생성(bison -d parser.y)한 뒤 Flex를 돌려야 함
- %option noyywrap
  - EOF에서 yywrap()을 호출하지 않게 하여 -lfl 링크 없이 깔끔히 끝냄

scanner.l 규칙부 (1)

```
%%

[0-9]+(\.[0-9]+)?      { yylval.num = strtod(yytext, NULL); return NUMBER; }
[a-zA-Z_][a-zA-Z0-9_]* { yylval.id = strdup(yytext); return IDENT; }

[ \t]+                 ;     /* 공백 무시 */
\n                     { return '\n'; }

"=="                   { return '='; }  /* 방어: 실수로 == 써도 =로 처리 */
"="                    { return '='; }
"("                    { return '('; }
")"                    { return ')'; }
"+"|"-"|"*"|"/"        { return yytext[0]; }

.                      { /* 알 수 없는 문자 */
                         fprintf(stderr, "unknown char: '%s'\n", yytext);
                       }

%%
```

scanner.l 규칙부(2)

```
[0-9]+(\.[0-9]+)?      { yylval.num = strtod(yytext, NULL); return NUMBER; }
```

- 정수 또는 소수(12, 3.14) 인식
- yytext를 double로 변환해 yylval.num에 저장 → Bison에서 NUMBER가 <num> 타입으로 전달

scanner.l 규칙부 (3)

```
[a-zA-Z_][a-zA-Z0-9_]* { yylval.id = strdup(yytext); return IDENT; }
```

- C 스타일 식별자: 첫 글자 영문/밑줄, 이후 영문/숫자/밑줄
- 문자열을 strdup으로 동적 복사 → 메모리 해제는 파서 쪽에서 처리(예제에서는 IDENT 소비 시 free($1);).

scanner.l 규칙부 (6)

```
[ \t]+                  ;     /* 공백 무시 */
\n                      { return '\n'; }
```

- 스페이스/탭은 무시(토큰 생성 안 함)
- 개행을 토큰으로 보냄(문장/라인 단위 문법을 위해 Bison이 개행을 토큰으로 사용)

scanner.l 규칙부 (7)

```
"=="                { return '='; }  /* 방어: 실수로 == 써도 =로 처리 */
"="                 { return '='; }
"("                 { return '('; }
")"                 { return ')'; }
"+"|"-"|"*"|"/"     { return yytext[0]; }
```

- 연산자/구분자를 문자 자체로 반환
- Bison 문법에서 | '(' expr ')' 같은 식으로 사용

parser.y 선언부 (1)

parser.tab.c 파일에 수정없이 복사됨

- %{ %} 안의 코드는 그대로 C 파일로 복사
- 변수 저장용 심볼 테이블과 getvar, setvar 함수가 정의됨
- yylex()는 FLEX가 만들어 주는 스캐너 함수
- yyerror()는 문법 오류 발생 시 호출

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* --- 아주 단순한 심볼 테이블 --- */
typedef struct { char *name; double value; } binding_t;
static binding_t table[256];
static int tsize = 0;

static double getvar(const char *name) {
    for (int i = 0; i < tsize; ++i)
        if (strcmp(table[i].name, name) == 0) return table[i].value;
    return 0.0; /* 선언 없는 변수는 0으로 */
}

static void setvar(const char *name, double v) {
    for (int i = 0; i < tsize; ++i)
        if (strcmp(table[i].name, name) == 0) { table[i].value = v; return; }
    if (tsize < (int)(sizeof(table)/sizeof(table[0]))) {
        table[tsize].name = strdup(name);
        table[tsize].value = v;
        ++tsize;
    }
}

int yylex(void);
void yyerror(const char *s) { fprintf(stderr, "syntax error: %s\n", s); }
%}
```

parser.y 선언부 (2), 심볼테이블

변수 저장할 때 사용. getvar(), setvar()으로 접근함

```
typedef struct {
    char *name;    // 변수 이름 (동적 문자열 복사)
    double value; // 변수 값 (실수)
} binding_t;
static binding_t table[256];  // 최대 256개 변수 저장
static int tsize = 0;          // 현재까지 저장된 변수 개수
```

parser.y 선언부 (3)

Bison이 파서를 생성할 때 참고해서 내부 코드와 자료구조를 만드는 선언부 지시문

```
/* --- yylval 타입 --- */
%union {
    double num;   /* NUMBER의 값 */
    char  *id;    /* IDENT의 문자열 */
}

/* --- 토큰(스캐너가 반환) --- */
%token <num> NUMBER
%token <id>  IDENT

/* --- 연산자 우선순위/결합성 --- */
%left '+' '-'
%left '*' '/'

/* --- 비단말 타입 --- */
%type <num> expr term factor
```

parser.y 규칙부 (1)

```
%%

input
    : /* empty */
    | input line
    ;

line
    : '\n'
    | expr '\n'          { printf("= %g\n", $1); }
    | IDENT '=' expr '\n'{
          setvar($1, $3);
          printf("%s = %g\n", $1, $3);
          free($1);
      }
    ;

expr
    : expr '+' term      { $$ = $1 + $3; }
    | expr '-' term      { $$ = $1 - $3; }
    | term
    ;

term
    : term '*' factor    { $$ = $1 * $3; }
    | term '/' factor    {
          if ($3 == 0) { fprintf(stderr, "division by zero\n"); $$ = 0.0; }
          else $$ = $1 / $3;
      }
    | factor
    ;

factor
    : NUMBER             { $$ = $1; }
    | IDENT              { $$ = getvar($1); free($1); }
    | '(' expr ')'       { $$ = $2; }
    ;

%%
```

parser.y 규칙부 (2)

```
%%

input
    : /* empty */
    | input line
    ;
```

- 프로그램 전체는 여러 line으로 구성됨

parser.y 규칙부 (3)

```
line
    : '\n'
    | expr '\n'          { printf("= %g\n", $1); }
    | IDENT '=' expr '\n'{
          setvar($1, $3);
          printf("%s = %g\n", $1, $3);
          free($1);
      }
    ;
```

- 한 줄이 비어 있으면 무시
- 식과 개행이면 결과 출력
- 변수 대입이면 심볼 테이블에 저장

parser.y 규칙부 (4)

```
expr
    : expr '+' term      { $$ = $1 + $3; }
    | expr '-' term      { $$ = $1 - $3; }
    | term
    ;

term
    : term '*' factor    { $$ = $1 * $3; }
    | term '/' factor    {
          if ($3 == 0) { fprintf(stderr, "division by zero\n"); $$ = 0.0; }
          else $$ = $1 / $3;
      }
    | factor
    ;
```

parser.y 사용자정의부

```
int main(void) {
    printf("Mini calc: + - * /, variables (e.g., x=1+2).  Ctrl+D to quit.\n");
    return yyparse();
}
```

예제 실행

```
(base) mingi_kyung@MacBook-Pro simple_bison % vi scanner.l
(base) mingi_kyung@MacBook-Pro simple_bison % vi parser.y
(base) mingi_kyung@MacBook-Pro simple_bison % bison -d parser.y
(base) mingi_kyung@MacBook-Pro simple_bison % flex scanner.l
(base) mingi_kyung@MacBook-Pro simple_bison % cc parser.tab.c lex.yy.c -o calc
(base) mingi_kyung@MacBook-Pro simple_bison % ./calc
Mini calc: + - * /, variables (e.g., x=1+2).  Ctrl+D to quit.
1+2
= 3
x=3
x = 3
x+4
= 7
(2*3)*4
= 24
y=x*2
y = 6
y/0
division by zero
= 0
(base) mingi_kyung@MacBook-Pro simple_bison %
```