22BCE3939
Karan Sehgal
Compiler Design Lab
Digital Assignment 5

Q1:

Program to implement simple calculator using Lex and YACC in LLVM.

Code:

Calc.l (lex file)

```
%{
#include "calc.h"
#include "calc.tab.h"
%}

%%
[0-9]+      { yylval.num = atoi(yytext); return NUMBER; }
"+"         { return PLUS; }
"-"         { return MINUS; }
"*"         { return MULT; }
"/"         { return DIV; }
"("         { return LPAREN; }
")"         { return RPAREN; }
[\n]        { return EOL; }  /* Return EOL for newlines */
[ \t]       ; /* Skip other whitespace */
.           { printf("Unknown character %s\n", yytext); }
%%

int yywrap(void) {
    return 1;
}
~
```

```
%{
#include "calc.h"
#include "calc.tab.h"
%}

%%
[0-9]+      { yylval.num = atoi(yytext); return NUMBER; }
"+"         { return PLUS; }
"-"         { return MINUS; }
"*"         { return MULT; }
"/"         { return DIV; }
"("         { return LPAREN; }
")"         { return RPAREN; }
[\n]        { return EOL; }  /* Return EOL for newlines */
[ \t]       ; /* Skip other whitespace */
.           { printf("Unknown character %s\n", yytext); }
%%

int yywrap(void) {
   return 1;
}
```

calc.y (YACC)

```
/* calc.y */
%{
#include "calc.h"
%}

%union {
    int num;
    LLVMValueRef val;
}

%token <num> NUMBER
%token PLUS MINUS MULT DIV
%token LPAREN RPAREN EOL
%type <val> expr
%left PLUS MINUS
%left MULT DIV

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

line:     EOL
        | expr EOL    {
            printf("Result: %lld\n", LLVMConstIntGetSExtValue($1));
            LLVMDisposeMessage(LLVMPrintValueToString($1));
        }
        | error EOL  { yyerrok; }
        ;

expr:     expr PLUS expr { $$ = create_expr('+', $1, $3); }
        | expr MINUS expr { $$ = create_expr('-', $1, $3); }
        | expr MULT expr { $$ = create_expr('*', $1, $3); }
        | expr DIV expr { $$ = create_expr('/', $1, $3); }
        | LPAREN expr RPAREN { $$ = $2; }
        | NUMBER { $$ = LLVMConstInt(LLVMInt32Type(), $1, 0); }
        ;
%%
```

```
/* calc.y */
%{
#include "calc.h"
%}

%union {
   int num;
   LLVMValueRef val;
}

%token <num> NUMBER
%token PLUS MINUS MULT DIV
%token LPAREN RPAREN EOL
%type <val> expr
%left PLUS MINUS
%left MULT DIV

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

line:     EOL
       | expr EOL    {
          printf("Result: %lld\n", LLVMConstIntGetSExtValue($1));
          LLVMDisposeMessage(LLVMPrintValueToString($1));
       }
       | error EOL  { yyerrok; }
       ;

expr:     expr PLUS expr { $$ = create_expr('+', $1, $3); }
       | expr MINUS expr { $$ = create_expr('-', $1, $3); }
       | expr MULT expr { $$ = create_expr('*', $1, $3); }
       | expr DIV expr { $$ = create_expr('/', $1, $3); }
       | LPAREN expr RPAREN { $$ = $2; }
       | NUMBER { $$ = LLVMConstInt(LLVMInt32Type(), $1, 0); }
       ;
%%
```

calc.h (Header file)

```c
/* calc.h */
#ifndef CALC_H
#define CALC_H

#include <stdio.h>
#include <stdlib.h>
#include <llvm-c/Core.h>
#include <llvm-c/ExecutionEngine.h>
#include <llvm-c/Target.h>

extern LLVMModuleRef module;
extern LLVMBuilderRef builder;
extern LLVMExecutionEngineRef engine;
extern LLVMPassManagerRef pass_manager;

LLVMValueRef create_expr(int op, LLVMValueRef lhs, LLVMValueRef rhs);
int yylex(void);
int yyparse(void);
void yyerror(const char *s);
void yyrestart(FILE *input_file);  // Add this declaration

#endif
```

```
/* calc.h */
#ifndef CALC_H
#define CALC_H

#include <stdio.h>
#include <stdlib.h>
#include <llvm-c/Core.h>
#include <llvm-c/ExecutionEngine.h>
#include <llvm-c/Target.h>

extern LLVMModuleRef module;
extern LLVMBuilderRef builder;
extern LLVMExecutionEngineRef engine;
extern LLVMPassManagerRef pass_manager;

LLVMValueRef create_expr(int op, LLVMValueRef lhs, LLVMValueRef rhs);
int yylex(void);
int yyparse(void);
void yyerror(const char *s);
void yyrestart(FILE *input_file);  // Add this declaration

#endif
```

calc.c (c file)

```c
/* calc.c */
#include "calc.h"
#include <string.h>

LLVMModuleRef module;
LLVMBuilderRef builder;
LLVMExecutionEngineRef engine;
LLVMPassManagerRef pass_manager;

LLVMValueRef create_expr(int op, LLVMValueRef lhs, LLVMValueRef rhs) {
    switch (op) {
        case '+': return LLVMBuildAdd(builder, lhs, rhs, "addtmp");
        case '-': return LLVMBuildSub(builder, lhs, rhs, "subtmp");
        case '*': return LLVMBuildMul(builder, lhs, rhs, "multmp");
        case '/': return LLVMBuildSDiv(builder, lhs, rhs, "divtmp");
        default: return NULL;
    }
}

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

int main() {
    LLVMInitializeNativeTarget();
    LLVMInitializeNativeAsmPrinter();
    LLVMInitializeNativeAsmParser();

    module = LLVMModuleCreateWithName("calc_module");
    builder = LLVMCreateBuilder();

    LLVMLinkInMCJIT();
    LLVMCreateExecutionEngineForModule(&engine, module, NULL);

    printf("Calculator started. Enter expressions (Ctrl+D to exit):\n");
    yyparse();

    LLVMDisposeBuilder(builder);
    LLVMDisposeModule(module);
    return 0;
}
```

```
/* calc.c */
#include "calc.h"
#include <string.h>

LLVMModuleRef module;
LLVMBuilderRef builder;
LLVMExecutionEngineRef engine;
LLVMPassManagerRef pass_manager;

LLVMValueRef create_expr(int op, LLVMValueRef lhs, LLVMValueRef rhs) {
    switch (op) {
        case '+': return LLVMBuildAdd(builder, lhs, rhs, "addtmp");
        case '-': return LLVMBuildSub(builder, lhs, rhs, "subtmp");
        case '*': return LLVMBuildMul(builder, lhs, rhs, "multmp");
        case '/': return LLVMBuildSDiv(builder, lhs, rhs, "divtmp");
        default: return NULL;
    }
}

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

int main() {
    LLVMInitializeNativeTarget();
    LLVMInitializeNativeAsmPrinter();
    LLVMInitializeNativeAsmParser();

    module = LLVMModuleCreateWithName("calc_module");
    builder = LLVMCreateBuilder();

    LLVMLinkInMCJIT();
    LLVMCreateExecutionEngineForModule(&engine, module, NULL);

    printf("Calculator started. Enter expressions (Ctrl+D to exit):\n");
    yyparse();

    LLVMDisposeBuilder(builder);
    LLVMDisposeModule(module);
    return 0;
}
```

#Install bison
sudo apt-get install flex bison llvm-dev

Generate parser files
bison -d calc.y

Generate lexer files
flex calc.l

Compile everything together
gcc calc.c lex.yy.c calc.tab.c -o calculator $(llvm-config --cflags --ldflags --libs core executionengine mcjit native) -lfl

Output: