The Mini-Pascal Compiler Project Syllabus for CSCI 3313

Created by Chen Shen, 10/10/2016

Updated by Joseph Crandall 2017-10-07

**Overview:**

* The purpose of this project is to write a mini-pascal compiler by using the tools Flex and Bison. The final deliverables for the project are the are the following four files:
  + Flex file - project.l
  + Bison file - project.y
  + C file - project.c
  + header file - project.h
* **project.y** uses Bison to specify the mini-pascal grammar for the compiler which is the given mini-pascal grammar. The **project.l** uses Flex to specify the lexical token for the mini-pascal language. The **project.h** has the data type and declaration of functions. The **project.c** has the definitions of functions that are called in the **project.y** file.

**TimeLine & Grading:**

* Over 8 weeks we will have 8 assignment. Each assignment will be worth 10 points. Each of the assignments will be posted on a Monday and will be due the following Monday at 11:59 PM. There is a 24 hour extension deadline with a 40% penalty. After a 24 hours extension, your assignment receives a zero.
  + Project Week 1 – Due 2017-10-16 11:59PM
  + Project Week 2 – Due 2017-10-23 11:59PM
  + Project Week 3 – Due 2017-10-30 11:59PM
  + Break 2017-11-06
  + Project Week 4 – Due 2017-11-13 11:59PM
  + Project Week 5 – Due 2017-11-20 11:59PM
  + Project Week 6 – Due 2017-11-27 11:59PM
  + Project Week 7 – Due 2017-12-4 11:59PM
  + Project Week 8 – Due 2017-12-11 11:59PM
  + Project Week 9 – Due 2017-12-18 11:59PM

**Template Code:**

* There arefour template files on Blackboard. Use them to start your own project.
  + project\_template.c
  + project\_template.h
  + project\_template.l
  + project\_template.y

**Book:**

* We will be using the Flex & Bison O’reilly book heavily for content and explanation. The PDF of the book is on Blackboard.

**Submission:**

* Place all of your files in a folder labeled lastname\_firstname\_weeknumber. Zip that folder and name the zip file lastname\_firstname\_weeknumber.zip. Submit the zip file on blackboard. For example for week one I would place all my files in a folder named crandall\_joseph\_1 (all lower case) and would then zip the folder. I would name the zip file crandall\_joseph\_1.zip . I would then submit that zip file on blackboard.

**Install Flex & Bison on Ubuntu 16.04**

* Flex and Bison should already be installed on the standard distribution of Ubuntu 16.04

**Example Compilation and Execution of Final Project**

josephcrandall@ubuntu:~/Desktop/Choi\_Lab/Lab/project\_week\_9$ flex project9.l

josephcrandall@ubuntu:~/Desktop/Choi\_Lab/Lab/project\_week\_9$ bison -d project9.y

josephcrandall@ubuntu:~/Desktop/Choi\_Lab/Lab/project\_week\_9$ gcc -o project9 project9.c project9.tab.c lex.yy.c

josephcrandall@ubuntu:~/Desktop/Choi\_Lab/Lab/project\_week\_9$ ./project9 sort.txt

arr = { 1, 3, 7, 5, 4}

arr = { 1, 3, 4, 5, 7}

parse done.

josephcrandall@ubuntu:~/Desktop/Choi\_Lab/Lab/project\_week\_9$

**Week 1:** Due: Oct 16th 2017, 11:59PM.

**Submit**: Flex file (project1.l), compiled Flex file (lex.yy.c).

**Concepts:**

* Scanning divides the input into meaningful chunks, called tokens, and parsing figures out how the tokens relate to each other.
  + Flex is responsible for scanning
  + Bison is responsible for parsing
* For example, consider this snippet of C code:

alpha = beta + gamma;

* + A scanner divides this into the tokens alpha , equal sign , beta , plus sign , gamma , and semicolon . Then the parser determines that beta + gamma is an expression, and that the expression is assigned to alpha .
* Each time the program needs a token, it calls yylex() , which reads a little input and returns the token. When it needs another token, it calls yylex() again. The scanner acts as a coroutine; that is, each time it returns, it remembers where it was, and on the next call it picks up where it left off.

**Your work:**

* Internalize the content presented in Chapter 1(p.1-19) and Chapter 2(p.20-46) and Chapter 3(p.47-80) of the Flex & Bison O’reilly book
  + This is a large amount of reading but if you put in the time it will greatly improve your understanding.
* Use the Advanced Calculator Lexer on page 68 of the flex & bison book as a guide to fill in the missing code indicated by the … for your Flex (project1.l) file. Do not implement the /\* built-in functions \*/ section of the Advanced Calculator Lexer, this is not needed for our compiler.

%option noyywrap nodefault yylineno

%{

#include "project1.h"

#include "project1.tab.h"

%}

/\* float exponent \*/

EXP ([Ex][-+]?[0-9]+)

%%

/\* single character ops \*/

".." { return DOTS; }

"+" |

…

…

…

…

…

"," |

";" |

":" |

"." |

"[" |

"]" |

"{" |

"}" |

"(" |

")" { return yytext[0]; }

/\* comparison ops, all are a CMP token \*/

">" { yylval.fn = 1; return CMP; }

…

…

…

…

"<=" { yylval.fn = 6; return CMP; }

"integer" { yylval.type\_c = 'a'; return STD\_TYPE; }

"real" { yylval.type\_c = 'b'; return STD\_TYPE; }

"program" { return PROGRAM; }

"var" { return VAR; }

"array" { return ARRAY; }

"of" { return OF; }

"begin" { return BGN; }

"end" { return END; }

/\* keywords \*/

"if" { return IF; }

…

…

…

"do" { return DO; }

"print" { return PRINT; }

/\* names \*/

[a-zA-Z][a-zA-Z0-9]\* { yylval.s = lookup(yytext); return ID; }

[0-9]+"."[0-9]+ |

[0-9]+ { yylval.d = atof(yytext); return NUMBER; }

"//".\*

[ \t\n]

. { yyerror("Mystery character.\n"); }

%%

* Compile your Flex file by running.

Username@ubuntu:~/PATH\_TO\_DIREECTORY/project\_week\_1$ flex project1.l

* If successful you will have created a lex.yy.c file.

**Week 2:** Due: Due: Oct 23rd 2017, 11:59PM

**Submit**: Flex file (project2.l), compiled Flex file (lex.yy.c), Bison file (project2.y), compiled Bison files (project2.tab.c & project2.tab.h), and Header file (project2.h).

**Concepts:**

**Your work:**

* Use the Flex file developed during week 1 and the Header file from the template files.
* The header file is given in the starting template. You have to change the header files name an and dependent functions names within the file
* Read Chapter 9. Advanced Flex and Bison from the Flex & Bison O'reilly book.
* Copy the Bison implemented grammar into your project2.h file. We will fill in the expressions as we work through the C code over the following weeks.
* Compile the bison code

%{

int yylex(void);

#include <stdio.h>

#include <stdlib.h>

#include "project2.h"

%}

%union {

struct ast \*a;

double d;

struct symbol \*s;

struct symlist \*sl;

struct numlist \*nl;

int fn;

char type\_c;

}

%token <d> NUMBER

%token <s> ID

%token PROGRAM VAR ARRAY OF INTEGER REAL BGN END IF THEN ELSE WHILE DO DOTS PRINT

%token <type\_c> STD\_TYPE

%nonassoc <fn> CMP

%right '='

%left '+' '-'

%left '\*' '/'

%nonassoc '|' UMINUS

%type <a> decl\_list decl stmt\_list stmt exp

%type <sl> id\_list

%type <nl> num\_list

%start program

%%

program: PROGRAM ID '(' id\_list ')' ';' decl\_list BGN stmt\_list END '.'

{ printf("new program.\n"); }

;

decl\_list: { /\*$$ = NULL;\*/ }

| decl ';' decl\_list { printf("new declaration.\n"); }

;

decl: VAR id\_list ':' STD\_TYPE { }

| VAR id\_list ':' ARRAY '[' NUMBER DOTS NUMBER ']' OF STD\_TYPE

{ }

;

stmt: IF exp THEN '{' stmt\_list '}' { }

| IF exp THEN '{' stmt\_list '}' ELSE '{' stmt\_list '}' { }

| WHILE exp DO '{' stmt\_list '}' { }

| exp

;

stmt\_list: stmt { printf("new statement.\n"); }

| stmt\_list ';' stmt { }

;

exp: exp CMP exp { }

| exp '+' exp { }

| exp '-' exp { }

| exp '\*' exp { }

| exp '/' exp { }

| '|' exp { }

| '(' exp ')' { }

| '-' exp %prec UMINUS { }

| NUMBER { }

| ID { }

| ID '[' exp ']' { }

| ID '[' exp ']' '=' exp { }

| ID '=' exp { }

| ID '=' '{' num\_list '}' { }

| PRINT '(' exp ')' { }

;

num\_list: NUMBER { }

| NUMBER ',' num\_list { }

;

id\_list: ID { }

| ID ',' id\_list { }

;

%%

* Compile your Bison file by running.

Username@ubuntu:~/PATH\_TO\_DIRECTORY/project\_week\_2$ bison -d project2.y

* If successful you will have created project2.tab.c and project2.tab.h files

**Week 3:** Due Oct 30rd 2017, 11:59PM

**Submit**: sort.txt

**Concepts:**

* Understand bubble sort

**Your work:**

* Fill in the missing components of bubble sort, indicated by … in sort.txt
* The sort.txt files is the file that flex and bison will be parsing and compiling.

program sort (arr, i, j, temp) ;

var i, j, temp: integer;

var arr: array [0..4] of integer;

begin

arr = { 1, 3, 7, 5, 4 };

print(arr);

while i < 5 do { j = i; while j > 0 do { if arr[j] < arr[j-1] then { temp = … ; arr[j] = … ; arr[j-1] = … }; j = … }; i = … };

print(arr)

end

.

**Break**: Nov 6th 2017

**Week 4:** Due Nov 13th 2017, 11:59PM

**Submit**: Flex file (project4.l), compiled Flex file (lex.yy.c), Bison file (project4.y), compiled Bison files (project4.tab.c & project4.tab.h), Header file (project4.h), C file (project4.c), and sort.txt.

**Concepts:**

* In the Bison file it is important to understand the possible data types for tokens or variables within the grammar. For example
  + data type: struct ast \* name: a
  + data type: double name: d

union {

struct ast \*a;

double d;

struct symbol \*s;

struct symlist \*sl;

struct numlist \*nl;

int fn;

char type\_c;

}

* Declare a token and use < > to assign the data type name to the token name

%token <d> NUMBER

%token <s> ID

%token PROGRAM VAR ARRAY OF INTEGER REAL BGN END IF THEN ELSE WHILE DO DOTS PRINT

%token <type\_c> STD\_TYPE

%nonassoc <fn> CMP

%right '='

%left '+' '-'

%left '\*' '/'

%nonassoc '|' UMINUS

%type <a> decl\_list decl stmt\_list stmt exp

%type <sl> id\_list

%type <nl> num\_list

%start program

* Everything that appears in the grammar rules of the bison file is either a token or a variable. The token is read from the source code file and the variable is designed in the grammar, usually for recursion, and substituted for a token or multiple tokens.
* Each token or variable could be assigned with a data type. For example:

%token <type\_c> STD\_TYPE

* + This statement assigns the token STD\_TYPE with the data type <type\_c> which is the ‘char’ in C.

%type <nl> num\_list

* + This statement assigns the variable num\_list with the data type <nl> which is ‘struct numlist \*’ defined in the header file.
* Make sure that your flex file has complete set of tokens. Remove the print actions. For each token to be read, return something. You could define your own words to return. For the token with some value associated with it, assigned the value before return. For comments, tab, newline and space, do nothing. For anything else, give an error warning.
* Optimize your grammar rules in the bison file. Less rules are usually better. For each rule, do nothing or you could just print something for debugging purposes. Make sure your rules will not cause a dead loop.
* /\* one line on the righ
* $$ left hand side, $ number is right hand side

**Your work:**

* Use the Flex file developed during week 1, the Header file from the template files, the Bison file developed during week 2, and the sort.txt developed during week 3.
* Design two functions in your C file
  + symhash converts an input string into an unsigned integer value.

static unsigned symhash(char \*sym)

* + lookup returns a symbol from the array of struct symbols which is declared in the header

struct symbol \* lookup(char \*sym)

* (Take things away) Finish the following methods and add them to you C file underneath the #include files

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include "project4.h"

int yyparse(void);

static unsigned symhash(char \*sym)

{

unsigned int hash = 0;

unsigned c;

while(c = \*sym++) hash = hash\*9 ^ c;

return hash;

}

struct symbol \* lookup(char \*sym)

{

struct symbol \*sp = &symtab[symhash(sym) % NHASH];

int scount = NHASH;

while(--scount >= 0) {

if(sp->name && !strcmp(sp->name, sym)) { return sp; }

if(!sp->name) {

sp->name = strdup(sym);

sp->value = 0;

return sp;

}

if(++sp >= symtab + NHASH) sp = symtab;

}

yyerror("symbol table overflow\n");

abort();

}

void yyerror(char \*s)

{

fprintf(stderr, "error: %s\n", s);

}

int main(int argc, char \*\*argv)

{

extern FILE \*yyin;

++argv; --argc;

yyin = fopen(argv[0], "r");

return yyparse();

}

* Compile your code using the following commands and run the executable project4 on for text file (sort.txt). If successful you should print a series of new declaration, new statement, and new program

Username@ubuntu:~/PATH\_TO\_DIRECTORY/project\_week\_4$ flex project4.l

Username@ubuntu:~/PATH\_TO\_DIRECTORY/project\_week\_4$ bison -d project4.y

Username@ubuntu:~/PATH\_TO\_DIRECTORY/project\_week\_4$ gcc -o project4 project4.c project4.tab.c lex.yy.c

project4.c: In function ‘main’:

project4.c:48:9: warning: implicit declaration of function ‘yyparse’ [-Wimplicit-function-declaration]

return yyparse();

^

project4.tab.c: In function ‘yyparse’:

project4.tab.c:1213:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-function-declaration]

yychar = yylex ();

^

Username@ubuntu:~/PATH\_TO\_DIRECTORY/project\_week\_4$ ./project4 sort.txt

new declaration.

new declaration.

new statement.

new statement.

new statement.

new statement.

new program.

**Week 5:** Due: Nov 13th 2017, 11:59PM

**Submit**: Flex file (project5.l), compiled Flex file (lex.yy.c), Bison file (project5.y), compiled Bison files (project5.tab.c & project5.tab.h), Header file (project5.h), C file (project5.c), sort.txt and executable file program5 .

**Concepts:**

* In the bison file, in each rule, the value on the left side of ‘:’ is denoted as $$ in the C code part, the values on the right side from left to right order, are denoted as $1, $2, $3 in the C code.
* For example:

id\_list: ID ',' id\_list { $$ = newsymlist($1, $3); }

* + the first id\_list on the left side of the : is $$, the value of ID is $1 , the value of ‘,’ is $2 (we do not use it) and the value of the second id\_list is $3.
* Understand the application of an abstract syntax tree in our project. The idea is that every single action in the program, is cast into one data type called ast \*. For example, these actions assignment, declaration and logic comparison at beginning are cast into a tree node type with three arguments: node type, left subtree and right subtree. After this the function double eval(struct ast \*) will resolve these actions by identifying them via their node type. For example, if the node type says it is an assignment action, the real assignment action will be done inside eval() function.
* Two advantages of this design are:
  + #1, you could call eval() recursively to resolve some very difficult logic in the program. The actual meaning of eval() is evaluating the whole tree. The tree has a recursive structure so the design of the eval() function has a recursive structure as well.
  + #2, each logic action in the program can become decoupled. You can easily remove old or add new actions in the compiler.
* Our work is implementing the following functions in C file and call them in Bison file.
* The complete definitions of these functions are also in the given header file.

**Your work:**

* Design the following nine functions in your C file (project5.c). The complete definitions of these functions are in the given header file (project5.h). Finally complete the corresponding calls made in the Bison file (project5.y).
  + 1. newast is used to grow the tree

struct ast \* newast(int nodetype, struct ast \*l, struct ast \*r)

* + 2 . newnum used to initialize a number value

struct ast \* newnum(double d)

* + 3. newprint used to print the tree value

struct ast \* newprint(struct ast \*l)

* + 4. newref used to initialize a symbol value

struct ast \* newref(struct symbol \*s)

* + 5. newdecl used to initialize the declaration with the type and initial value

struct ast \* newdecl(struct symlist \*sl, char type)

* + 6. newdeclarr used to initialize the array declaration

struct ast \* newdeclarr(struct symlist \*sl, int begin, int end, char type)

* + 7. treefree is used to free the memory associated with the tree

void treefree(struct ast \*a)

* + 8. newsymlist used to grow the symbol list

struct symlist \* newsymlist(struct symbol \*sym, struct symlist \*next)

* + 9. eval is used to evaluate the whole tree to the bottom

double eval(struct ast \*a)

* The C file (project5.c) is structured as follows, it is your job to fill in the missing code and then compile the file.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include "project5.h"

int yyparse(void);

static unsigned symhash(char \*sym)

{

/\* week 4 \*/

}

struct symbol \* lookup(char \*sym)

{

/\* week 4 \*/

}

struct ast \* newast(int nodetype, struct ast \*l, struct ast \*r)

{

struct ast \*a = malloc(sizeof(struct ast));

if(!a) {

yyerror("out of space");

exit(0);

}

a->nodetype = nodetype;

a->l = l;

a->r = r;

return a;

}

struct ast \* newnum(double d)

{

struct numval \* a = malloc(sizeof(struct numval));

if(!a) {

yyerror("out of space");

exit(0);

}

a->nodetype = 'K';

a->number = d;

return (struct ast \*)a;

}

struct ast \* newprint(struct ast \*l)

{

struct printcall \*a = malloc(sizeof(struct printcall));

if(!a) {

yyerror("out of space");

exit(0);

}

a->nodetype = 'P';

a->l = l;

return (struct ast \*)a;

}

struct ast \* newref(struct symbol \*s)

{

struct symref \*a = malloc(sizeof(struct symref));

if(!a) {

yyerror("out of space");

exit(0);

}

a->nodetype = 'N';

a->s = s;

return (struct ast \*)a;

}

struct ast \* newdecl(struct symlist \*sl, char type)

{

struct decl \*a = malloc(sizeof(struct decl));

if(!a) {

yyerror("out of space");

exit(0);

}

a->nodetype = 'X';

a->sl = sl;

a->type = type;

return (struct ast \*)a;

}

struct ast \* newdeclarr(struct symlist \*sl, int begin, int end, char type)

{

struct declarr \*a = malloc(sizeof(struct declarr));

if(!a) {

yyerror("out of space");

exit(0);

}

a->nodetype = 'Y';

a->sl = sl;

a->len = end - begin + 1;

if(a->len < 1) {

yyerror("too small size for array");

exit(0);

}

a->shift = begin;

a->type = type;

return (struct ast \*)a;

}

void treefree(struct ast \*a)

{

switch(a->nodetype) {

case '+':

case '-':

case '\*':

case '/':

case '1': case '2': case '3': case '4': case '5': case '6':

case 'L':

treefree(a->r);

case '|':

case 'M': case 'P':

treefree(a->l);

case 'K': case 'N': case 'U': case 'V': case 'T':

break;

case '=':

free(((struct symasgn \*)a)->v);

break;

case 'I': case 'W':

free(((struct flow \*)a)->cond);

if(((struct flow \*)a)->tl) treefree(((struct flow \*)a)->tl);

if(((struct flow \*)a)->el) treefree(((struct flow \*)a)->el);

break;

case 'X':

free(((struct decl \*)a)->sl);

break;

case 'Y':

free(((struct declarr \*)a)->sl);

break;

default: printf("internal error: free bad node %c\n", a->nodetype);

}

free(a);

}

struct symlist \* newsymlist(struct symbol \*sym, struct symlist \*next)

{

struct symlist \*sl = malloc(sizeof(struct symlist));

if(!sl) {

yyerror("out of space");

exit(0);

}

sl->sym = sym;

sl->next = next;

return sl;

}

double eval(struct ast \*a)

{

double v;

if(!a) {

yyerror("internal error, null eval");

return 0.0;

}

switch(a->nodetype) {

case 'K': v = ((struct numval \*)a)->number; break;

case 'N': if((((struct symref \*)a)->s)->type != 'a' && (((struct symref \*)a)->s)->type != 'b')

{ printf("using undeclared ID: %s\n", (((struct symref \*)a)->s)->name); }

v = (((struct symref \*)a)->s)->value; break;

case '=': if((((struct symasgn \*)a)->s)->type != 'a' && (((struct symasgn \*)a)->s)->type != 'b')

{ printf("using undeclared ID: %s\n", (((struct symasgn \*)a)->s)->name); }

v = ((struct symasgn \*)a)->s->value =

eval(((struct symasgn \*)a)->v); break;

case '+': v = eval(a->l) + eval(a->r); break;

case '-': v = eval(a->l) - eval(a->r); break;

case '\*': v = eval(a->l) \* eval(a->r); break;

case '/': v = eval(a->l) / eval(a->r); break;

case '|': v = fabs(eval(a->l)); break;

case 'M': v = -eval(a->l); break;

case '1': v = (eval(a->l) > eval(a->r))? 1 : 0; break;

case '2': v = (eval(a->l) < eval(a->r))? 1 : 0; break;

case '3': v = (eval(a->l) != eval(a->r))? 1 : 0; break;

case '4': v = (eval(a->l) == eval(a->r))? 1 : 0; break;

case '5': v = (eval(a->l) >= eval(a->r))? 1 : 0; break;

case '6': v = (eval(a->l) <= eval(a->r))? 1 : 0; break;

case 'I':

if(eval(((struct flow \*)a)->cond) != 0) {

if(((struct flow \*)a)->tl) {

v = eval(((struct flow \*)a)->tl);

} else

v = 0.0;

} else {

if(((struct flow \*)a)->el) {

v = eval(((struct flow \*)a)->el);

} else

v = 0.0;

}

break;

case 'W':

v = 0.0;

if(((struct flow \*)a)->tl) {

while(eval(((struct flow \*)a)->cond) != 0)

v = eval(((struct flow \*)a)->tl);

}

break;

case 'L': eval(a->l); v = eval(a->r); break;

default: printf("internal error: bad node %c\n", a->nodetype);

}

return v;

}

void yyerror(char \*s)

{

fprintf(stderr, "error: %s\n", s);

}

int main(int argc, char \*\*argv)

{

extern FILE \*yyin;

++argv; --argc;

yyin = fopen(argv[0], "r");

return yyparse();

}

* Implement the correct calls to the Bison file (project5.y) corresponding to the implemented C file (project5.c).

%{

int yylex(void);

#include <stdio.h>

#include <stdlib.h>

#include "project5.h"

%}

/\* week 2 \*/

%start program

%%

program: PROGRAM ID '(' id\_list ')' ';' decl\_list BGN stmt\_list END '.'

{ eval($7); eval($9); treefree($9); treefree($7); printf("parse done.\n"); }

;

decl\_list: { /\*$$ = NULL;\*/ }

| decl ';' decl\_list {if($3 == NULL) $$ = $1; else $$ = newast('L', $1, $3); }

;

decl: VAR id\_list ':' STD\_TYPE { $$ = newdecl($2, $4); }

| VAR id\_list ':' ARRAY '[' NUMBER DOTS NUMBER ']' OF STD\_TYPE { $$ = newdeclarr($2, $6, $8, $11); }

;

stmt: IF exp THEN '{' stmt\_list '}' { }

| IF exp THEN '{' stmt\_list '}' ELSE '{' stmt\_list '}' { }

| WHILE exp DO '{' stmt\_list '}' { }

| exp

;

stmt\_list: stmt { $$ = $1; }

| stmt\_list ';' stmt { $$ = newast('L', $1, $3); }

;

exp: exp CMP exp { }

| exp '+' exp { $$ = newast('+', $1, $3); }

| exp '-' exp { $$ = newast('-', $1, $3); }

| exp '\*' exp { $$ = newast('\*', $1, $3); }

| exp '/' exp { $$ = newast('/', $1, $3); }

| '|' exp { $$ = newast('|', $2, NULL); }

| '(' exp ')' { $$ = $2;}

| '-' exp %prec UMINUS { $$ = newast('M', $2, NULL); }

| NUMBER { $$ = newnum($1); }

| ID { $$ = newref($1); }

| ID '[' exp ']' { }

| ID '[' exp ']' '=' exp { }

| ID '=' exp { }

| ID '=' '{' num\_list '}' { }

| PRINT '(' exp ')' { $$ = newprint($3); }

;

num\_list: NUMBER { }

| NUMBER ',' num\_list { }

;

id\_list: ID { $$ = newsymlist($1, NULL); }

| ID ',' id\_list { $$ = newsymlist($1, $3); }

;

%%

* Compile your code using the following commands and run the executable project5 on for text file (sort.txt). If successful you should see a series of bad nodes resulting eventually in an Aborted (core dumped)

Username@ubuntu:~/PATH\_TO\_DIRECTORY/project\_week\_5$ flex project5.l

Username@ubuntu:~/PATH\_TO\_DIRECTORY/project\_week\_5$ bison -d project5.y

Username@ubuntu:~/PATH\_TO\_DIRECTORY/project\_week\_5$ gcc -o project5 project5.c project5.tab.c lex.yy.c

project5.c: In function ‘main’:

project5.c:48:9: warning: implicit declaration of function ‘yyparse’ [-Wimplicit-function-declaration]

return yyparse();

^

project5.tab.c: In function ‘yyparse’:

project5.tab.c:1213:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-function-declaration]

yychar = yylex ();

^

Username@ubuntu:~/PATH\_TO\_DIRECTORY/project\_week\_4$ ./project5 sort.txt

internal error: bad node p

internal error: bad node Y

internal error: bad node �

...

7f37f73dc000-7f37f73dd000 rw-p 00026000 08:01 924801 /lib/x86\_64-linux-gnu/ld-2.23.so

7f37f73dd000-7f37f73de000 rw-p 00000000 00:00 0

7ffc5f1b7000-7ffc5f1d9000 rw-p 00000000 00:00 0 [stack]

7ffc5f1fa000-7ffc5f1fc000 r--p 00000000 00:00 0 [vvar]

7ffc5f1fc000-7ffc5f1fe000 r-xp 00000000 00:00 0 [vdso]

ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]

Aborted (core dumped)

**Week 6:** Due: Due: Nov 13th 2017, 11:59PM

**Submit**: Flex file (project6.l), compiled Flex file (lex.yy.c), Bison file (project6.y), compiled Bison files (project6.tab.c & project6.tab.h), Header file (project6.h), C file (project6.c), sort.txt and executable file program6 .

**Concepts:**

* For each action, whether it is an assignment or an arithmetic operation, cast it into a tree node with an integer to indicate its type. So after reading the whole program, a tree has been built. Then the program goes through the tree and realizes the actions. The tree nodes with different type will be treated differently. This is the design.

**Your work:**

* Understand how the following Bison (from project5.y) rules are implemented in the C function double eval(struct ast \*);

exp: exp CMP exp { }

| exp '+' exp { $$ = newast('+', $1, $3); }

| exp '-' exp { $$ = newast('-', $1, $3); }

| exp '\*' exp { $$ = newast('\*', $1, $3); }

| exp '/' exp { $$ = newast('/', $1, $3); }

| '|' exp { $$ = newast('|', $2, NULL); }

| '(' exp ')' { $$ = $2;}

| '-' exp %prec UMINUS { $$ = newast('M', $2, NULL); }

| NUMBER { $$ = newnum($1); }

| ID { $$ = newref($1); }

| ID '[' exp ']' { }

| ID '[' exp ']' '=' exp { }

| ID '=' exp { }

| ID '=' '{' num\_list '}' { }

| PRINT '(' exp ')' { $$ = newprint($3); }

;

* Implement two functions in your C file (project6.c)
  + 1 . newcmp is used to grow a comparison action tree node

struct ast \* newcmp(int cmptype, struct ast \*l, struct ast \*r)

* + 2. newasgn used to assign a numerical value to symbol

struct ast \* newasgn(struct symbol \*s, struct ast \*v)

* The C file (project6.c) is structured as follows, it is your job to fill in the missing code and then compile the file.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include "project6.h"

int yyparse(void);

static unsigned symhash(char \*sym)

{

/\* week 4 \*/

}

struct symbol \* lookup(char \*sym)

{

/\* week 4 \*/

}

struct ast \* newast(int nodetype, struct ast \*l, struct ast \*r)

{

/\* week 5 \*/

}

struct ast \* newnum(double d)

{

/\* week 5 \*/

}

struct ast \* newcmp(int cmptype, struct ast \*l, struct ast \*r)

{

struct ast \*a = malloc(sizeof(struct ast));

if(!a) {

yyerror("out of space");

exit(0);

}

a->nodetype = '0' + cmptype;

a->l = l;

a->r = r;

return a;

}

struct ast \* newprint(struct ast \*l)

{

/\* week 5 \*/

}

struct ast \* newref(struct symbol \*s)

{

/\* week 5 \*/

}

struct ast \* newasgn(struct symbol \*s, struct ast \*v)

{

struct symasgn \*a = malloc(sizeof(struct symasgn));

if(!a) {

yyerror("out of space");

exit(0);

}

a->nodetype = '=';

a->s = s;

a->v = v;

return (struct ast \*)a;

}

struct ast \* newdecl(struct symlist \*sl, char type)

{

/\* week 5 \*/

}

struct ast \* newdeclarr(struct symlist \*sl, int begin, int end, char type)

{

/\* week 5 \*/

}

void treefree(struct ast \*a)

{

/\* week 5 \*/

}

struct symlist \* newsymlist(struct symbol \*sym, struct symlist \*next)

{

/\* week 5 \*/

}

static double callprint(struct printcall \*);

static double calldecl(struct decl \*);

static double calldeclarr(struct declarr \*);

static double callrefarr(struct symrefarr \*);

static double callasgnarr(struct symasgnarr \*);

static double callinitarr(struct syminitarr \*);

double eval(struct ast \*a)

{

/\* week 5 \*/

}

void yyerror(char \*s)

{

fprintf(stderr, "error: %s\n", s);

}

int main(int argc, char \*\*argv)

{

extern FILE \*yyin;

++argv; --argc;

yyin = fopen(argv[0], "r");

return yyparse();

}

* Implement the correct calls to the Bison file (project6.y) corresponding to the implemented C file (project6.c).

%{

int yylex(void);

#include <stdio.h>

#include <stdlib.h>

#include "project6.h"

%}

/\* week 5 \*/

%start program

%%

program: PROGRAM ID '(' id\_list ')' ';' decl\_list BGN stmt\_list END '.'

{ eval($7); eval($9); treefree($9); treefree($7); printf("parse done.\n"); }

;

decl\_list: { /\*$$ = NULL;\*/ }

| decl ';' decl\_list {if($3 == NULL) $$ = $1; else $$ = newast('L', $1, $3); }

;

decl: VAR id\_list ':' STD\_TYPE { $$ = newdecl($2, $4); }

| VAR id\_list ':' ARRAY '[' NUMBER DOTS NUMBER ']' OF STD\_TYPE { $$ = newdeclarr($2, $6, $8, $11); }

;

stmt: IF exp THEN '{' stmt\_list '}' { }

| IF exp THEN '{' stmt\_list '}' ELSE '{' stmt\_list '}' { }

| WHILE exp DO '{' stmt\_list '}' { }

| exp

;

stmt\_list: stmt { $$ = $1; }

| stmt\_list ';' stmt { $$ = newast('L', $1, $3); }

;

exp: exp CMP exp { $$ = newcmp($2, $1, $3); }

| exp '+' exp { $$ = newast('+', $1, $3); }

| exp '-' exp { $$ = newast('-', $1, $3); }

| exp '\*' exp { $$ = newast('\*', $1, $3); }

| exp '/' exp { $$ = newast('/', $1, $3); }

| '|' exp { $$ = newast('|', $2, NULL); }

| '(' exp ')' { $$ = $2;}

| '-' exp %prec UMINUS { $$ = newast('M', $2, NULL); }

| NUMBER { $$ = newnum($1); }

| ID { $$ = newref($1); }

| ID '[' exp ']' { }

| ID '[' exp ']' '=' exp { }

| ID '=' exp { $$ = newasgn($1, $3); }

| ID '=' '{' num\_list '}' { }

| PRINT '(' exp ')' { $$ = newprint($3); }

;

num\_list: NUMBER { }

| NUMBER ',' num\_list { }

;

id\_list: ID { $$ = newsymlist($1, NULL); }

| ID ',' id\_list { $$ = newsymlist($1, $3); }

;

%%

* Compile your code using the following commands and run the executable project6 on for text file (sort.txt). If successful you should see a series of bad nodes resulting eventually in an Aborted (core dumped)

Username@ubuntu:~/PATH\_TO\_DIRECTORY/project\_week\_6$ flex project6.l

Username@ubuntu:~/PATH\_TO\_DIRECTORY/project\_week\_6$ bison -d project6.y

Username@ubuntu:~/PATH\_TO\_DIRECTORY/project\_week\_6$ gcc -o project6 project6.c project6.tab.c lex.yy.c

project6.c: In function ‘main’:

project6.c:48:9: warning: implicit declaration of function ‘yyparse’ [-Wimplicit-function-declaration]

return yyparse();

^

project6.tab.c: In function ‘yyparse’:

project6.tab.c:1213:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-function-declaration]

yychar = yylex ();

^

Username@ubuntu:~/PATH\_TO\_DIRECTORY/project\_week\_4$ ./project5 sort.txt

internal error: bad node p

internal error: bad node Y

...

7ffc5f1fa000-7ffc5f1fc000 r--p 00000000 00:00 0 [vvar]

7ffc5f1fc000-7ffc5f1fe000 r-xp 00000000 00:00 0 [vdso]

ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]

Aborted (core dumped)

**Week 7:** Due: Due: Nov 13th 2017, 11:59PM

**Submit**: Flex file (project7.l), compiled Flex file (lex.yy.c), Bison file (project7.y), compiled Bison files (project7.tab.c & project7.tab.h), Header file (project7.h), C file (project7.c), sort.txt and executable file program7 .

**Concepts:**

* ...

**Your work:**

* Implement the following 6 functions
  + 1. newinitarr is used to combine the symbol (name of the array) and a list of numbers into a tree node

struct ast \* newinitarr(struct symbol \*s, struct numlist \*nl)

* + 2. newnumlist is generated in the whole list of numbers

struct numlist \* newnumlist(double num, struct numlist \*next)

* + 3. calldecl is used ...

static double calldecl(struct decl \*d)

* + 4. calldeclarr is used …

static double calldeclarr(struct declarr \* da)

* + 5. callprint is used …

static double callprint(struct printcall \*f)

* + 6 . callinitarr is a function called in eval() which does the real actions of array initialization

static double callinitarr(struct syminitarr \*ia)

* Also you need to add an additional 3 cases to eval()
* The C file (project7.c) is structured as follows, it is your job to fill in the missing code and then compile the file.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include "project7.h"

int yyparse(void);

static unsigned symhash(char \*sym)

{

/\* week 4 \*/

}

struct symbol \* lookup(char \*sym)

{

/\* week 4 \*/

}

struct ast \* newast(int nodetype, struct ast \*l, struct ast \*r)

{

/\* week 5 \*/

}

struct ast \* newnum(double d)

{

/\* week 5 \*/

}

struct ast \* newcmp(int cmptype, struct ast \*l, struct ast \*r)

{

/\* week 6 \*/

}

struct ast \* newprint(struct ast \*l)

{

/\* week 5 \*/

}

struct ast \* newref(struct symbol \*s)

{

/\* week 5 \*/

}

struct ast \* newasgn(struct symbol \*s, struct ast \*v)

{

/\* week 6 \*/

}

struct ast \* newinitarr(struct symbol \*s, struct numlist \*nl)

{

struct syminitarr \*a = malloc(sizeof(struct syminitarr));

if(!a) {

yyerror("out of space");

exit(0);

}

a->nodetype = 'T';

a->s = s;

a->nl = nl;

return (struct ast \*)a;

}

struct ast \* newdecl(struct symlist \*sl, char type)

{

/\* week 5 \*/

}

struct ast \* newdeclarr(struct symlist \*sl, int begin, int end, char type)

{

/\* week 5 \*/

}

void treefree(struct ast \*a)

{

/\* week 5 \*/

}

struct symlist \* newsymlist(struct symbol \*sym, struct symlist \*next)

{

/\* week 5 \*/

}

struct numlist \* newnumlist(double num, struct numlist \*next)

{

struct numlist \*nl = malloc(sizeof(struct numlist));

if(!nl) {

yyerror("out of space");

exit(0);

}

nl->n = num;

nl->next = next;

return nl;

}

static double callprint(struct printcall \*);

static double calldecl(struct decl \*);

static double calldeclarr(struct declarr \*);

static double callrefarr(struct symrefarr \*);

static double callasgnarr(struct symasgnarr \*);

static double callinitarr(struct syminitarr \*);

double eval(struct ast \*a)

{

/\* week 6 \*/

case 'P': v = callprint((struct printcall \*)a); break;

case 'X': v = calldecl((struct decl \*)a); break;

case 'Y': v = calldeclarr((struct declarr \*)a); break;

}

static double calldecl(struct decl \*d)

{

if(d->type == 'a') {

while(d->sl) {

((d->sl)->sym)->value = 0;

((d->sl)->sym)->type = 'a';

((d->sl)->sym)->arr\_len = 0;

((d->sl)->sym)->ini\_index = 0;

((d->sl)->sym)->arr\_head = NULL;

//printf("symbol: %s, type: %c\n", ((d->sl)->sym)->name, ((d->sl)->sym)->type);

d->sl = (d->sl)->next;

}

} else if(d->type == 'b') {

while(d->sl) {

((d->sl)->sym)->value = 0.0;

((d->sl)->sym)->type = 'b';

((d->sl)->sym)->arr\_len = 0;

((d->sl)->sym)->ini\_index = 0;

((d->sl)->sym)->arr\_head = NULL;

//printf("symbol: %s, type: %c\n", ((d->sl)->sym)->name, ((d->sl)->sym)->type);

d->sl = (d->sl)->next;

}

} else {

printf("bad declaration type: %c\n", d->type);

return 1;

}

return 0;

}

static double calldeclarr(struct declarr \*da)

{

if(da->type == 'a') {

while(da->sl) {

((da->sl)->sym)->value = -1;

((da->sl)->sym)->type = 'a';

((da->sl)->sym)->arr\_len = da->len;

((da->sl)->sym)->ini\_index = da->shift;

((da->sl)->sym)->arr\_head = (double \*)malloc(sizeof(double) \* da->len);

if(!(((da->sl)->sym)->arr\_head)) {

yyerror("out of space for array");

exit(0);

}

//printf("symbol: %s, type: %c\n", ((da->sl)->sym)->name, ((da->sl)->sym)->type);

da->sl = (da->sl)->next;

}

} else if(da->type == 'b') {

while(da->sl) {

((da->sl)->sym)->value = -1.0;

((da->sl)->sym)->type = 'b';

((da->sl)->sym)->arr\_len = da->len;

((da->sl)->sym)->ini\_index = da->shift;

((da->sl)->sym)->arr\_head = (double \*)malloc(sizeof(double) \* da->len);

if(!(((da->sl)->sym)->arr\_head)) {

yyerror("out of space for array");

exit(0);

}

//printf("symbol: %s, type: %c\n", ((da->sl)->sym)->name, ((da->sl)->sym)->type);

da->sl = (da->sl)->next;

}

} else {

printf("bad declaration type %c\n", da->type);

return 1;

}

return 0;

}

static double callprint(struct printcall \*f)

{

int i;

if((f->l)->nodetype == 'N') {

printf("%s", (((struct symref \*)(f->l))->s)->name);

}

if((f->l)->nodetype == 'N' && (((struct symref \*)(f->Ul))->s)->arr\_len > 0) {

printf(" = {");

for(i = 0; i < (((struct symref \*)(f->l))->s)->arr\_len - 1; i++) {

printf("%4.4g, ", \*((((struct symref \*)(f->l))->s)->arr\_head + i));

}

printf("%4.4g", \*((((struct symref \*)(f->l))->s)->arr\_head + (((struct symref \*)(f->l))->s)->arr\_len - 1));

printf("}\n");

return 0;

}

double v = eval(f->l);

printf(" = %4.4g\n", v);

return v;

}

static double callinitarr(struct syminitarr \*ia)

{

int i;

for(i = 0; i < ((ia->s)->arr\_len) && (ia->nl) != NULL; i++)

{

\*((ia->s)->arr\_head + i) = (ia->nl)->n;

ia->nl = (ia->nl)->next;

}

return 0;

}

void yyerror(char \*s)

{

fprintf(stderr, "error: %s\n", s);

}

int main(int argc, char \*\*argv)

{

extern FILE \*yyin;

++argv; --argc;

yyin = fopen(argv[0], "r");

return yyparse();

}

* Implement the correct calls to the Bison file (project7.y) corresponding to the implemented C file (project7.c).

%{

int yylex(void);

#include <stdio.h>

#include <stdlib.h>

#include "project7.h"

%}

/\* week 6 \*/

%start program

%%

program: PROGRAM ID '(' id\_list ')' ';' decl\_list BGN stmt\_list END '.' { eval($7); eval($9); treefree($9); treefree($7); printf("parse done.\n"); }

;

decl\_list: { /\*$$ = NULL;\*/ }

| decl ';' decl\_list {if($3 == NULL) $$ = $1; else $$ = newast('L', $1, $3); }

;

decl: VAR id\_list ':' STD\_TYPE { $$ = newdecl($2, $4); }

| VAR id\_list ':' ARRAY '[' NUMBER DOTS NUMBER ']' OF STD\_TYPE { $$ = newdeclarr($2, $6, $8, $11); }

;

stmt: IF exp THEN '{' stmt\_list '}' { }

| IF exp THEN '{' stmt\_list '}' ELSE '{' stmt\_list '}' { }

| WHILE exp DO '{' stmt\_list '}' { }

| exp

;

stmt\_list: stmt { $$ = $1; }

| stmt\_list ';' stmt { $$ = newast('L', $1, $3); }

;

exp: exp CMP exp { $$ = newcmp($2, $1, $3); }

| exp '+' exp { $$ = newast('+', $1, $3); }

| exp '-' exp { $$ = newast('-', $1, $3); }

| exp '\*' exp { $$ = newast('\*', $1, $3); }

| exp '/' exp { $$ = newast('/', $1, $3); }

| '|' exp { $$ = newast('|', $2, NULL); }

| '(' exp ')' { $$ = $2;}

| '-' exp %prec UMINUS { $$ = newast('M', $2, NULL); }

| NUMBER { $$ = newnum($1); }

| ID { $$ = newref($1); }

| ID '[' exp ']' { }

| ID '[' exp ']' '=' exp { }

| ID '=' exp { $$ = newasgn($1, $3); }

| ID '=' '{' num\_list '}' { newinitarr($1, $4); }

| PRINT '(' exp ')' { $$ = newprint($3); }

;

num\_list: NUMBER { $$ = newnumlist($1, NULL); }

| NUMBER ',' num\_list { $$ = newnumlist($1, $3); }

;

id\_list: ID { $$ = newsymlist($1, NULL); }

| ID ',' id\_list { $$ = newsymlist($1, $3); }

;

%%

* Compile your code using the following commands and run the executable project6 on for text file (sort.txt). If successful you should see a series of bad nodes resulting eventually in an Aborted (core dumped)

**Week 8:** Due: Due: Nov 13th 2017, 11:59PM

**Submit**: Flex file (project8.l), compiled Flex file (lex.yy.c), Bison file (project8.y), compiled Bison files (project8.tab.c & project8.tab.h), Header file (project8.h), C file (project8.c), sort.txt and executable file program8 .

**Concepts:**

* ...

**Your work:**

* **F**inish four functions related to array reference. Assign the values of the tree node a to the array member indexed by index. Get the value of an array with index. To test your code, you could insert print(arr[2]) in your text file.
  + 1. newrefarr the first function passes the values and cast into a tree node type

struct ast \* newrefarr(struct symbol \*s, struct ast \*index)

* + 2. newasgnarr passes the values and cast into a tree node type

struct ast \* newasgnarr(struct symbol \*s, struct ast \*index, struct ast \*v)

* + 3. callrefarr gets the value from array with index … 457

static double callrefarr(struct symrefarr \*sa)

* + 4. callasgnarr does the assignment work … 467

static double callasgnarr(struct symasgnarr \*aa)

* You must also add another additional three cases to eval()
* The C file (project8.c) is structured as follows, it is your job to fill in the missing code and then compile the file.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include "project8.h"

int yyparse(void);

static unsigned symhash(char \*sym)

{

/\* week 4 \*/

}

struct symbol \* lookup(char \*sym)

{

/\* week 4 \*/

}

struct ast \* newast(int nodetype, struct ast \*l, struct ast \*r)

{

/\* week 5 \*/

}

struct ast \* newnum(double d)

{

/\* week 5 \*/

}

struct ast \* newcmp(int cmptype, struct ast \*l, struct ast \*r)

{

/\* week 6 \*/

}

struct ast \* newprint(struct ast \*l)

{

/\* week 5 \*/

}

struct ast \* newref(struct symbol \*s)

{

/\* week 5 \*/

}

struct ast \* newrefarr(struct symbol \*s, struct ast \*index)

{

struct symrefarr \*a = malloc(sizeof(struct symrefarr));

if(!a) {

yyerror("out of space");

exit(0);

}

a->nodetype = 'U';

a->s = s;

a->index = index;

return (struct ast \*)a;

}

struct ast \* newasgnarr(struct symbol \*s, struct ast \*index, struct ast \*v)

{

struct symasgnarr \*a = malloc(sizeof(struct symasgnarr));

if(!a) {

yyerror("out of space");

exit(0);

}

a->nodetype = 'V';

a->s = s;

a->index = index;

a->v = v;

return (struct ast \*)a;

}

struct ast \* newasgn(struct symbol \*s, struct ast \*v)

{

/\* week 6 \*/

}

struct ast \* newinitarr(struct symbol \*s, struct numlist \*nl)

{

/\* week 7 \*/

}

struct ast \* newdecl(struct symlist \*sl, char type)

{

/\* week 5 \*/

}

struct ast \* newdeclarr(struct symlist \*sl, int begin, int end, char type)

{

/\* week 5 \*/

}

void treefree(struct ast \*a)

{

/\* week 5 \*/

}

struct symlist \* newsymlist(struct symbol \*sym, struct symlist \*next)

{

/\* week 5 \*/

}

struct numlist \* newnumlist(double num, struct numlist \*next)

{

/\* week 7 \*/

}

static double callprint(struct printcall \*);

static double calldecl(struct decl \*);

static double calldeclarr(struct declarr \*);

static double callrefarr(struct symrefarr \*);

static double callasgnarr(struct symasgnarr \*);

static double callinitarr(struct syminitarr \*);

double eval(struct ast \*a)

{

/\* week 7 \*/

case 'U': if((((struct symrefarr \*)a)->s)->type != 'a' && (((struct symrefarr \*)a)->s)->type != 'b')

{ printf("using undeclared ID: %s\n", (((struct symrefarr \*)a)->s)->name); }

v = callrefarr((struct symrefarr \*)a); break;

case 'V': if((((struct symasgnarr \*)a)->s)->type != 'a' && (((struct symasgnarr \*)a)->s)->type != 'b')

{ printf("using undeclared ID: %s\n", (((struct symasgnarr \*)a)->s)->name); }

v = callasgnarr((struct symasgnarr \*)a); break;

case 'T': if((((struct syminitarr \*)a)->s)->type != 'a' && (((struct syminitarr \*)a)->s)->type != 'b')

{ printf("using undeclared ID: %s\n", (((struct syminitarr \*)a)->s)->name); }

v = callinitarr((struct syminitarr \*)a); break;

}

static double calldecl(struct decl \*d)

{

/\* week 7 \*/

}

static double calldeclarr(struct declarr \*da)

{

/\* week 7 \*/

}

static double callprint(struct printcall \*f)

{

/\* week 7 \*/

}

static double callrefarr(struct symrefarr \*sa)

{

int arr\_index = (int)eval(sa->index);

if((sa->s)->arr\_len == 0) {

yyerror("wrong reference");

exit(0);

}

return \*(((sa->s)->arr\_head) + arr\_index - ((sa->s)->ini\_index));

}

static double callasgnarr(struct symasgnarr \*aa)

{

int arr\_index = (int)eval(aa->index);

double d = eval(aa->v);

\*(((aa->s)->arr\_head) + arr\_index - ((aa->s)->ini\_index)) = d;

return d;

}

static double callinitarr(struct syminitarr \*ia)

{

/\* week 7 \*/

}

void yyerror(char \*s)

{

fprintf(stderr, "error: %s\n", s);

}

int main(int argc, char \*\*argv)

{

extern FILE \*yyin;

++argv; --argc;

yyin = fopen(argv[0], "r");

return yyparse();

}

* Implement the correct calls to the Bison file (project8.y) corresponding to the implemented C file (project8.c).

%{

int yylex(void);

#include <stdio.h>

#include <stdlib.h>

#include "project8.h"

%}

/\* week 7 \*/

%start program

%%

program: PROGRAM ID '(' id\_list ')' ';' decl\_list BGN stmt\_list END '.' { eval($7); eval($9); treefree($9); treefree($7); printf("parse done.\n"); }

;

decl\_list: { /\*$$ = NULL;\*/ }

| decl ';' decl\_list {if($3 == NULL) $$ = $1; else $$ = newast('L', $1, $3); }

;

decl: VAR id\_list ':' STD\_TYPE { $$ = newdecl($2, $4); }

| VAR id\_list ':' ARRAY '[' NUMBER DOTS NUMBER ']' OF STD\_TYPE { $$ = newdeclarr($2, $6, $8, $11); }

;

stmt: IF exp THEN '{' stmt\_list '}' { }

| IF exp THEN '{' stmt\_list '}' ELSE '{' stmt\_list '}' { }

| WHILE exp DO '{' stmt\_list '}' { }

| exp

;

stmt\_list: stmt { $$ = $1; }

| stmt\_list ';' stmt { $$ = newast('L', $1, $3); }

;

exp: exp CMP exp { $$ = newcmp($2, $1, $3); }

| exp '+' exp { $$ = newast('+', $1, $3); }

| exp '-' exp { $$ = newast('-', $1, $3); }

| exp '\*' exp { $$ = newast('\*', $1, $3); }

| exp '/' exp { $$ = newast('/', $1, $3); }

| '|' exp { $$ = newast('|', $2, NULL); }

| '(' exp ')' { $$ = $2;}

| '-' exp %prec UMINUS { $$ = newast('M', $2, NULL); }

| NUMBER { $$ = newnum($1); }

| ID { $$ = newref($1); }

| ID '[' exp ']' { $$ = newrefarr($1, $3); }

| ID '[' exp ']' '=' exp { $$ = newasgnarr($1, $3, $6); }

| ID '=' exp { $$ = newasgn($1, $3); }

| ID '=' '{' num\_list '}' { $$ = newinitarr($1, $4); }

| PRINT '(' exp ')' { $$ = newprint($3); }

;

num\_list: NUMBER { $$ = newnumlist($1, NULL); }

| NUMBER ',' num\_list { $$ = newnumlist($1, $3); }

;

id\_list: ID { $$ = newsymlist($1, NULL); }

| ID ',' id\_list { $$ = newsymlist($1, $3); }

;

%%

* Compile your code using the following commands and run the executable project8 on for text file (sort.txt). If successful you should see a series of bad nodes resulting eventually in an Aborted (core dumped)

**Week 9:** Due: Due: Nov 13th 2017, 11:59PM

**Submit**: Flex file (project9.l), compiled Flex file (lex.yy.c), Bison file (project9.y), compiled Bison files (project9.tab.c & project9.tab.h), Header file (project9.h), C file (project9.c), sort.txt and executable file program9 .

**Concepts:**

* Finish the last function related to the while loop and if-then-else. This function passes the values and casts into a tree node type. The actual logic is given in the eval() function.

**Your work:**

* Implement the final function
  + 1. newflow passes the values and casts them into a tree node type

struct ast \* newflow(int nodetype, struct ast \*cond, struct ast \*tl, struct ast \*el)

* The C file (project9.c) is structured as follows, it is your job to fill in the missing code and then compile the file. Run the compiled executable on the text file (sort.txt).

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include "project9.h"

int yyparse(void);

static unsigned symhash(char \*sym)

{

/\* week 4 \*/

}

struct symbol \* lookup(char \*sym)

{

/\* week 4 \*/

}

struct ast \* newast(int nodetype, struct ast \*l, struct ast \*r)

{

/\* week 5 \*/

}

struct ast \* newnum(double d)

{

/\* week 5 \*/

}

struct ast \* newcmp(int cmptype, struct ast \*l, struct ast \*r)

{

/\* week 6 \*/

}

struct ast \* newprint(struct ast \*l)

{

/\* week 5 \*/

}

struct ast \* newref(struct symbol \*s)

{

/\* week 5 \*/

}

struct ast \* newrefarr(struct symbol \*s, struct ast \*index)

{

/\* week 8 \*/

}

struct ast \* newasgnarr(struct symbol \*s, struct ast \*index, struct ast \*v)

{

/\* week 8 \*/

}

struct ast \* newasgn(struct symbol \*s, struct ast \*v)

{

/\* week 6 \*/

}

struct ast \* newinitarr(struct symbol \*s, struct numlist \*nl)

{

/\* week 7 \*/

}

struct ast \* newdecl(struct symlist \*sl, char type)

{

/\* week 5 \*/

}

struct ast \* newdeclarr(struct symlist \*sl, int begin, int end, char type)

{

/\* week 5 \*/

}

struct ast \* newflow(int nodetype, struct ast \*cond, struct ast \*tl, struct ast \*el)

{

struct flow \*a = malloc(sizeof(struct flow));

if(!a) {

yyerror("out of space");

exit(0);

}

a->nodetype = nodetype;

a->cond = cond;

a->tl = tl;

a->el = el;

return (struct ast \*)a;

}

void treefree(struct ast \*a)

{

/\* week 5 \*/

}

struct symlist \* newsymlist(struct symbol \*sym, struct symlist \*next)

{

/\* week 5 \*/

}

struct numlist \* newnumlist(double num, struct numlist \*next)

{

/\* week 7 \*/

}

static double callprint(struct printcall \*);

static double calldecl(struct decl \*);

static double calldeclarr(struct declarr \*);

static double callrefarr(struct symrefarr \*);

static double callasgnarr(struct symasgnarr \*);

static double callinitarr(struct syminitarr \*);

double eval(struct ast \*a)

{

/\* week 8 \*/

}

static double calldecl(struct decl \*d)

{

/\* week 7 \*/

}

static double calldeclarr(struct declarr \*da)

{

/\* week 7 \*/

}

static double callprint(struct printcall \*f)

{

/\* week 7 \*/

}

static double callrefarr(struct symrefarr \*sa)

{

/\* week 8 \*/

}

static double callasgnarr(struct symasgnarr \*aa)

{

/\* week 8 \*/

}

static double callinitarr(struct syminitarr \*ia)

{

/\* week 7 \*/

}

void yyerror(char \*s)

{

fprintf(stderr, "error: %s\n", s);

}

int main(int argc, char \*\*argv)

{

extern FILE \*yyin;

++argv; --argc;

yyin = fopen(argv[0], "r");

return yyparse();

}

* Implement the correct calls to the Bison file (project9.y) corresponding to the implemented C file (project9.c).

}

%{

int yylex(void);

#include <stdio.h>

#include <stdlib.h>

#include "project9.h"

%}

/\* week 8 \*/

%start program

%%

program: PROGRAM ID '(' id\_list ')' ';' decl\_list BGN stmt\_list END '.' { eval($7); eval($9); treefree($9); treefree($7); printf("parse done.\n"); }

;

decl\_list: { $$ = NULL; }

| decl ';' decl\_list { if($3 == NULL) $$ = $1; else $$ = newast('L', $1, $3); }

;

decl: VAR id\_list ':' STD\_TYPE { $$ = newdecl($2, $4); }

| VAR id\_list ':' ARRAY '[' NUMBER DOTS NUMBER ']' OF STD\_TYPE { $$ = newdeclarr($2, $6, $8, $11); }

;

stmt: IF exp THEN '{' stmt\_list '}' { $$ = newflow('I', $2, $5, NULL); }

| IF exp THEN '{' stmt\_list '}' ELSE '{' stmt\_list '}' { $$ = newflow('I', $2, $5, $9); }

| WHILE exp DO '{' stmt\_list '}' { $$ = newflow('W', $2, $5, NULL); }

| exp

;

stmt\_list: stmt { $$ = $1; }

| stmt\_list ';' stmt { $$ = newast('L', $1, $3); }

;

exp: exp CMP exp { $$ = newcmp($2, $1, $3); }

| exp '+' exp { $$ = newast('+', $1, $3); }

| exp '-' exp { $$ = newast('-', $1, $3); }

| exp '\*' exp { $$ = newast('\*', $1, $3); }

| exp '/' exp { $$ = newast('/', $1, $3); }

| '|' exp { $$ = newast('|', $2, NULL); }

| '(' exp ')' { $$ = $2; }

| '-' exp %prec UMINUS { $$ = newast('M', $2, NULL); }

| NUMBER { $$ = newnum($1); }

| ID { $$ = newref($1); }

| ID '[' exp ']' { $$ = newrefarr($1, $3); }

| ID '[' exp ']' '=' exp { $$ = newasgnarr($1, $3, $6); }

| ID '=' exp { $$ = newasgn($1, $3); }

| ID '=' '{' num\_list '}' { $$ = newinitarr($1, $4); }

| PRINT '(' exp ')' { $$ = newprint($3); }

;

num\_list: NUMBER { $$ = newnumlist($1, NULL); }

| NUMBER ',' num\_list { $$ = newnumlist($1, $3); }

;

id\_list: ID { $$ = newsymlist($1, NULL); }

| ID ',' id\_list { $$ = newsymlist($1, $3); }

;

%%

* Compile your code using the following commands and run the executable project8 on for text file (sort.txt). If successful you should see the following

Username@ubuntu:~/PATH\_TO\_FILE/project\_week\_9$ flex project9.l

Username@ubuntu@ubuntu:~/PATH\_TO\_FILE/project\_week\_9$ bison -d project9.y

Username@ubuntu@ubuntu:~/PATH\_TO\_FILE/project\_week\_9$ gcc -o project9 project9.c project9.tab.c lex.yy.c

project9.c: In function ‘main’:

project9.c:496:9: warning: implicit declaration of function ‘yyparse’ [-Wimplicit-function-declaration]

return yyparse();

^

project9.tab.c: In function ‘yyparse’:

project9.tab.c:1213:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-function-declaration]

yychar = yylex ();

^

Username@ubuntu@ubuntu:~/PATH\_TO\_FILE/project\_week\_9$ ./project9 sort.txt

arr = { 1, 3, 7, 5, 4}

arr = { 1, 3, 4, 5, 7}

parse done.

Username@ubuntu@ubuntu:~/PATH\_TO\_FILE/project\_week\_9$