

Compiler Design Lab

WEEK 10 Part-1

1. Write a Lex and Yacc program to Validate and Evaluate the prefix and postfix expression.

CODE

```
%option noyywrap

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

int stack[100];
int top=-1;

void push(int val){ stack[++top]=val; }
int pop(){ return stack[top--]; }

int isOperator(char c){
    return (c=='+'||c=='-'||c=='*'||c=='/'||c=='%');
}

int evalPostfix(char exp[]){
    int i;
```

```

for(i=0;exp[i];i++){
    char c=exp[i];
    if(c>='0'&&c<='9') push(c-'0');
    else if(isOperator(c)){
        int b=pop();
        int a=pop();
        switch(c){
            case '+': push(a+b); break;
            case '-': push(a-b); break;
            case '*': push(a*b); break;
            case '/': push(a/b); break;
            case '%': push(a%b); break;
        }
    }
}
return pop();
}

```

```

int evalPrefix(char exp[]){
    int len=strlen(exp);
    for(int i=len-1;i>=0;i--){
        char c=exp[i];
        if(c>='0'&&c<='9') push(c-'0');
        else if(isOperator(c)){
            int a=pop();
            int b=pop();
            switch(c){
                case '+': push(a+b); break;
                case '-': push(a-b); break;
                case '*': push(a*b); break;
                case '/': push(a/b); break;
            }
        }
    }
}

```

```

        case '%': push(a%b); break;
    }
}
}
return pop();
}
}%

%%

[a-zA-Z0-9+*/% \n-]+ {
    char exp[100]; strcpy(exp, yytext);
    int op=0,num=0;
    for(int i=0;i<strlen(exp);i++){
        if(isOperator(exp[i])) op++;
        else if(exp[i]>='0'&&exp[i]<='9') num++;
    }
    if(op==num-1){
        printf("Valid Postfix Expression\n");
        printf("Result = %d\n", evalPostfix(exp));
    }
    else if(op==num+1){
        printf("Valid Prefix Expression\n");
        printf("Result = %d\n", evalPrefix(exp));
    }
    else printf("Invalid Expression\n");
}
.\n ;
%%

```

```

int main(){
    printf("Enter prefix or postfix expression:\n");

```

```
yylex();  
return 0;  
}
```

OUTPUT

```
Enter expression:  
45+6*  
^Z  
Valid postfix expression  
PS C:\Users\Arc\Desktop\lex> .\w101.exe  
Enter expression:  
45+6  
^Z  
Invalid expression  
PS C:\Users\Arc\Desktop\lex> .\w101.exe  
Enter expression:  
+4*56  
^Z  
Valid postfix expression
```

2. Write a Lex and Yacc program to Validate Boolean expression and flow of control statements.

CODE

```
%option noyywrap  
%{  
#include<stdio.h>  
%}  
  
%%  
if|else|then|while|for|do|switch|case|break|continue {printf("Keyword: %s\n",yytext);}  
"=="|"!="| "<="| ">="| "<"| ">" {printf("Relational Operator: %s\n",yytext);}
```

```

"&&"|"||"|"!" {printf("Logical Operator: %s\n",yytext);}
"=" {printf("Assignment Operator: %s\n",yytext);}
"(" {printf("Opening Parenthesis\n");}
")" {printf("Closing Parenthesis\n");}
[a-zA-Z_][a-zA-Z0-9_]* {printf("Identifier: %s\n",yytext);}
[0-9]+ {printf("Number: %s\n",yytext);}
";" {printf("Semicolon\n");}
[ \t\n] ;
. {printf("Invalid symbol: %s\n",yytext);}
%%

int main(){
printf("Enter Boolean expression or control statement:\n");
yylex();
return 0;
}

```

OUTPUT

```

Enter Boolean expression or control statement:
if (m == n || o == p && q == r) x == 1;
Keyword: if
Opening Parenthesis
Identifier: m
Relational Operator: ==
Identifier: n
Logical Operator: ||
Identifier: o
Relational Operator: ==
Identifier: p
Logical Operator: &&
Identifier: q
Relational Operator: ==
Identifier: r
Closing Parenthesis
Identifier: x
Relational Operator: ==
Number: 1
Semicolon

```

3. Implementation of symbol Table for c declarations using Lex and Yacc.

CODE

```
%option noyywrap
```

```
%{
```

```
#include<stdio.h>
```

```
#include<string.h>
```

```
struct symbol{
```

```
    char name[50];
```

```
    char type[20];
```

```
}table[100];
```

```
int count=0;
```

```
char currentType[20];
```

```
void insert(char *name,char *type){
```

```
    for(int i=0;i<count;i++)
```

```
        if(strcmp(table[i].name,name)==0) return;
```

```
    strcpy(table[count].name,name);
```

```
    strcpy(table[count].type,type);
```

```
    count++;
```

```
}
```

```
%}
```

```
%%
```

```
int|float|char|double {
```

```
    strcpy(currentType,yytext);
```

```
}
```

```
[a-zA-Z_][a-zA-Z0-9_]* {
```

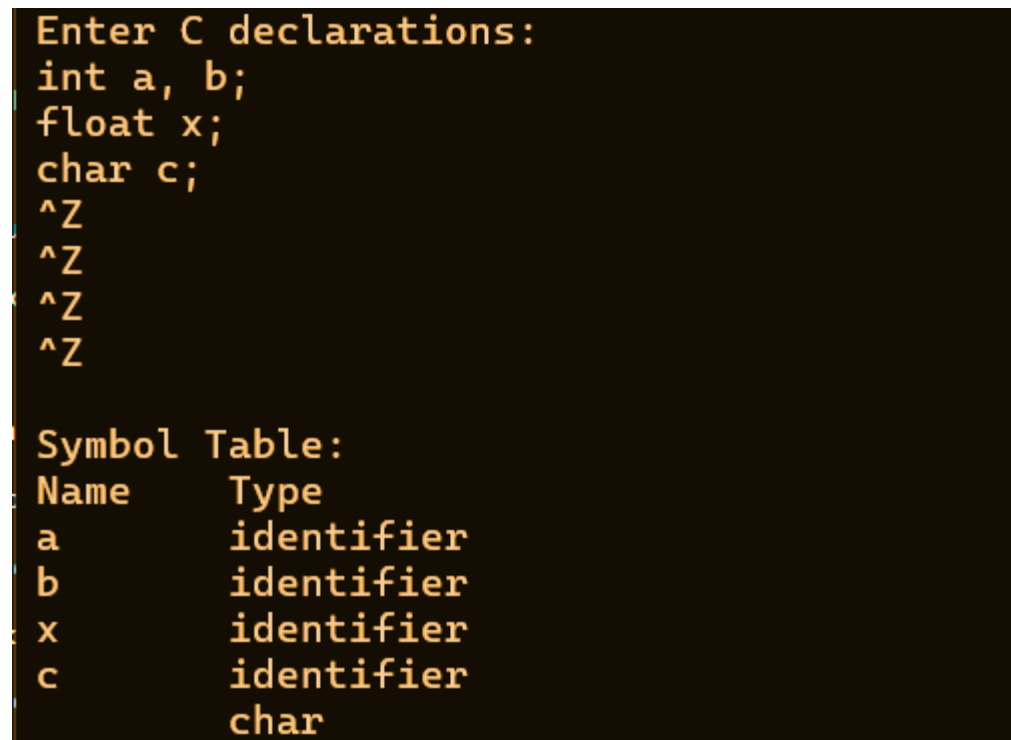
```

insert(yytext,currentType);
}
[ \t\n;,\[\]\{\}<>()=] ;
. ;
%%

int main(){
printf("Enter C declarations:\n");
yylex();
printf("\nSymbol Table:\n");
printf("Name\tType\n");
for(int i=0;i<count;i++)
printf("%s\t%s\n",table[i].name,table[i].type);
return 0;
}

```

OUTPUT



```

Enter C declarations:
int a, b;
float x;
char c;
^Z
^Z
^Z
^Z

Symbol Table:
Name      Type
a          identifier
b          identifier
x          identifier
c          identifier
char      char

```

Compiler Design Lab

WEEK 11

1. Write a Lex and Yacc program to implement the following.

a. Convert the given source code to TAC.

Input:

a = b + c * 20

Output1:

t1 = id3 * 20.0

id1 = id2 + t1

CODE

```
/* tac.l */
%{
#include "y.tab.h"
#include <string.h>
extern YYSTYPE yylval;
%}

%%

[a-zA-Z_][a-zA-Z0-9_]* { yylval.str = strdup(yytext); return ID; }
[0-9]+(\.[0-9]+)?      { yylval.str = strdup(yytext); return NUM; }
```



```

[-+*/=]      { return yytext[0]; }
[ \t]        ; /* Skip whitespace */
\n           { return 0; } /* End of input on newline */
.            { return yytext[0]; }
%%

```

```

int yywrap() {
    return 1;
}

```

YAAC

```

/* tac.y */
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

```

void yyerror(const char *s);
int yylex(void);
char* newtemp();

```

```

int temp_count = 1;
%}

```

```

/* Define the union for token types */
%union {
    char* str;
}

```

```

%token <str> ID NUM

```

%left '+' '-'

%left '*' '/'

%type <str> S E T F

%%

```
S : ID '=' E {
    printf("%s = %s\n", $1, $3);
    printf("\n");
    exit(0);
}
;
```

```
E : E '+' T {
    $$ = newtemp();
    printf("%s = %s + %s\n", $$, $1, $3);
}
| T    { $$ = $1; }
;
```

```
T : T '*' F {
    $$ = newtemp();
    printf("%s = %s * %s\n", $$, $1, $3);
}
| F    { $$ = $1; }
;
```

```
F : '(' E ')' { $$ = $2; }
| ID    { $$ = $1; }
| NUM   { $$ = $1; }
;
```

%%

```
char* newtemp() {  
    char* temp = (char*)malloc(sizeof(char)*4);  
    sprintf(temp, "t%d", temp_count++);  
    return temp;  
}
```

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

```
int main() {  
    printf("Enter expression (e.g., a=b+c*20): ");  
    yyparse();  
    return 0;  
}
```

OUTPUT

```
Enter expression :  
a = b + c * 20  
^Z  
t1 = c * 20.0  
a = b + t1
```

2. Validate the nested for-loop statement. Convert the nested for-loop statement to Single for-loop statement.

Input:

```
for (int i = 1; i < 10; i++) {  
  for (int j = 1; j < 10; j++) {  
    printf( "%d , %d", i , j);  
  }  
}
```

Output:

```
for( int i =1, j= 1; i <10 && j < 10 ; i++ , j++) {  
  printf( "%d , %d", i , j);  
}
```

CODE

```
%{  
#include "y.tab.h"  
%}  
  
%%  
"for"      { return FOR; }  
"int"      { return INT; }  
"printf"   { return PRINTF; }  
[a-zA-Z_][a-zA-Z0-9_]* { yylval.str = strdup(yytext); return ID; }  
[0-9]+     { yylval.str = strdup(yytext); return NUM; }  
"<"       { return LT; }  
"++"      { return INC; }  
";"       { return SEMI; }  
"("       { return LPAREN; }  
")"       { return RPAREN; }  
"{"       { return LBRACE; }  
"}"       { return RBRACE; }  
"="       { return ASSIGN; }  
"\\\"%d %d\\\" { yylval.str = strdup(yytext); return FORMAT_STR; }  
","       { return COMMA; }  
[ \\t\\n]+ ;  
.         { }  
%%  
  
int yywrap() { return 1; }
```

YAAC

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

```
void yyerror(const char* s);
int yylex();
%}
```

```
%union {
    char* str;
}
```

```
%token <str> ID NUM FORMAT_STR
%token FOR INT PRINTF LT INC SEMI LPAREN RPAREN LBRACE RBRACE ASSIGN COMMA
```

```
%type <str> var init limit body print_vars
```

```
%%
```

```
start: nested_loop { exit(0); };
```

```
nested_loop: FOR LPAREN INT var ASSIGN init SEMI var LT limit SEMI var INC RPAREN LBRACE
            FOR LPAREN INT var ASSIGN init SEMI var LT limit SEMI var INC RPAREN LBRACE
            body
            RBRACE
        RBRACE
    {
        printf("\n--- Converted Code ---\n");
        printf("for(int %s=%s, %s=%s; %s < %s && %s < %s; %s++, %s++) {\n",
            $4, $6, $16, $18, $4, $10, $16, $22, $4, $16);
        printf("    %s\n", $26);
        printf("}\n");
    }
    ;
```

```
var: ID { $$ = $1; };
init: NUM { $$ = $1; };
limit: NUM { $$ = $1; };
```

```
body: PRINTF LPAREN FORMAT_STR COMMA print_vars RPAREN SEMI {
    char* temp = (char*) malloc(100);
    sprintf(temp, "printf(\"%%d,%%d\", %s);", $5);
    $$ = temp;
}
;
```

```

print_vars: ID COMMA ID {
    char* temp = (char*) malloc(50);
    sprintf(temp, "%s, %s", $1, $3);
    $$ = temp;
};

%%

void yyerror(const char *s) {
    fprintf(stderr, "Parse error: %s. Make sure the input exactly matches the required
format.\n", s);
}

int main() {
    printf("Enter the nested for-loop code and press Ctrl+D when done:\n");
    yyparse();
    return 0;
}

```

OUTPUT

```

Enter nested for-loop statement:
for (int i = 1; i < 10; i++ ) { for (int j = 1; j < 10; j++) { printf("%d , %d", i , j); } }
Keyword: for
Open bracket
Keyword: int
Identifier: i
Number: 1
Identifier: i
Relational operator: <
Number: 10
Identifier: i
Increment/Decrement: ++
Close bracket
Open brace
Keyword: for
Open bracket
Keyword: int
Identifier: j
Number: 1
Identifier: j
Relational operator: <
Number: 10
Identifier: j
Increment/Decrement: ++
Close bracket
Open brace
Keyword: printf
Open bracket
Identifier: d
Identifier: d
Identifier: i
Identifier: j
Close bracket
Close brace
Close brace
^Z

Converted Single Loop:
for(int i=1,j=1; i<10 && j<10; i++,j++) {
    printf("%d , %d", i , j);
}

```

3. Validate the nested if-else statement. Convert the nested if-else statement to Single if else statement.

Input: `if(x != y) { If(x > y) { x = 0 } else { x = 1 } } else { x = 1 }`

Output : `if (x != y && x > y) x = 0 else x = 1`

CODE

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

"if"      { return IF; }
"else"    { return ELSE; }
[a-zA-Z_][a-zA-Z0-9_]* { yylval.str = strdup(yytext); return ID; }
[0-9]+    { yylval.str = strdup(yytext); return NUM; }
"!="     { return NE; }
">"      { return GT; }
"="      { return ASSIGN; }
"("      { return LPAREN; }
")"      { return RPAREN; }
"{"      { return LBRACE; }
"}"      { return RBRACE; }
[ \t\n;]+ ;
.        ;
%%

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

YAAC

%{

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#ifndef _GNU_SOURCE

#define _GNU_SOURCE

#endif

void yyerror(const char* s);

int yylex();

%}

%union {

char* str;

}

%token <str> ID NUM

%token IF ELSE NE GT ASSIGN LPAREN RPAREN LBRACE RBRACE

%type <str> cond asgn

%%

start: nested_if { exit(0); };

nested_if: IF LPAREN cond RPAREN LBRACE

IF LPAREN cond RPAREN LBRACE

asgn


```

RBRACE
ELSE LBRACE
    asgn
RBRACE
RBRACE
{
    printf("\n--- Converted Code ---\n");
    printf("if(%s && %s) {\n", $3, $8);
    printf("    %s;\n", $11);
    printf("} else {\n");
    printf("    %s;\n", $16);
    printf("}\n");
}
;

```

```

cond: ID NE ID { asprintf(&$$, "%s != %s", $1, $3); }
    | ID GT ID { asprintf(&$$, "%s > %s", $1, $3); }
;

```

```

asgn: ID ASSIGN NUM { asprintf(&$$, "%s = %s", $1, $3); }
;

```

```
%%
```

```

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

```

```

int main() {
    printf("Enter the nested if-else code and press Ctrl+D when done:\n");
    yyparse();
    return 0;
}

```

}

OUTPUT

```
Enter nested if-else statement:
if (x != y) { if (x > y) { x = 0; } else { x = 1; } } else { x = 1; }
Keyword: if
Open parenthesis
Identifier: x
Relational operator: !=
Identifier: y
Close parenthesis
Open brace
Keyword: if
Open parenthesis
Identifier: x
Relational operator: >
Identifier: y
Close parenthesis
Open brace
Identifier: x
Assignment operator
Number: 0
Close brace
Keyword: else
Open brace
Identifier: x
Assignment operator
Number: 1
Close brace
Close brace
Keyword: else
Open brace
Identifier: x
Assignment operator
Number: 1
Close brace
^Z

Converted Single if-else Statement:
if (x != y && x > y)
    x = 0;
else
    x = 1;
```