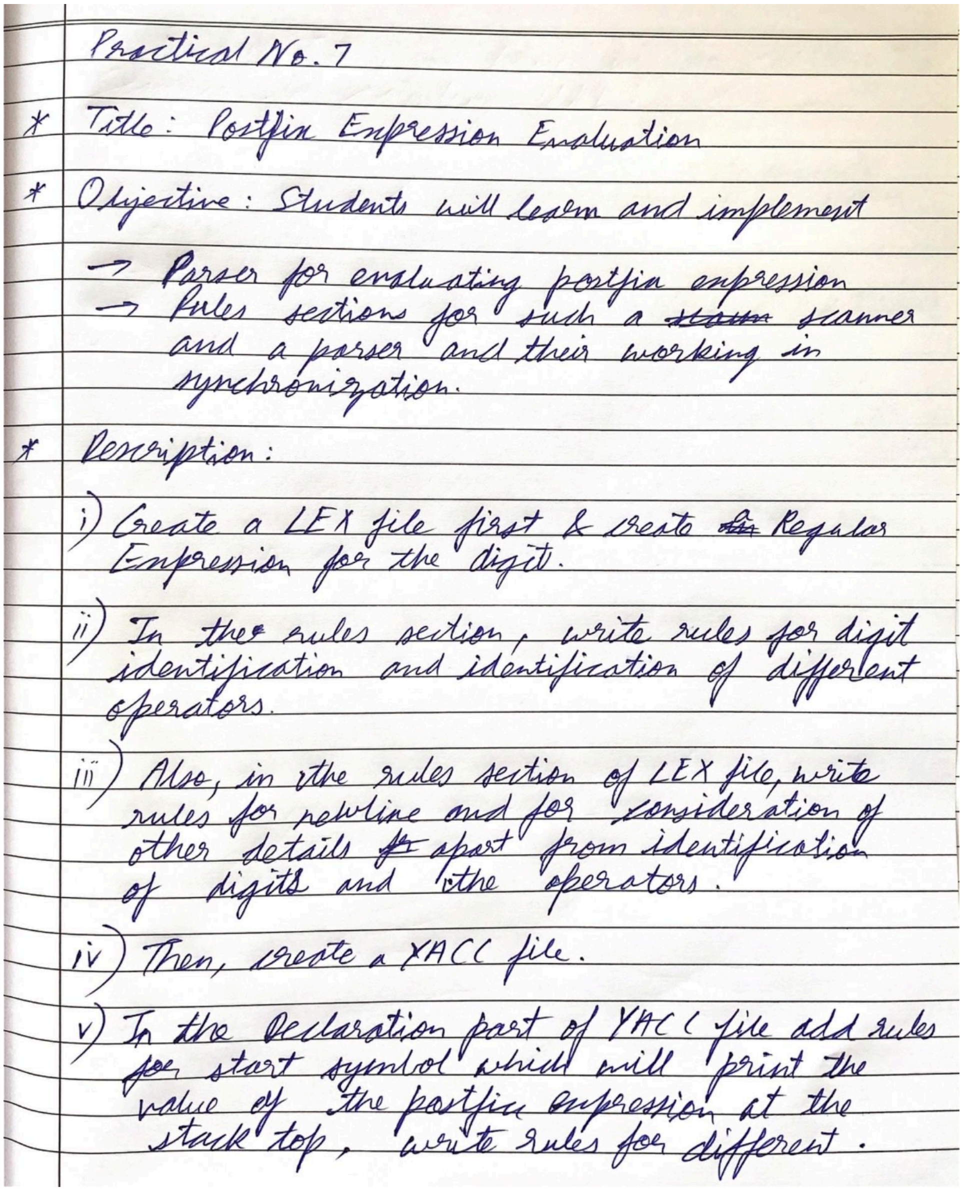


Practical No. 7

* Title: Postfix Expression Evaluation

* Objective: Students will learn and implement

→ Parser for evaluating postfix expression
→ Rules sections for such a ~~scann~~ scanner and a parser and their working in synchronization.

* Description:

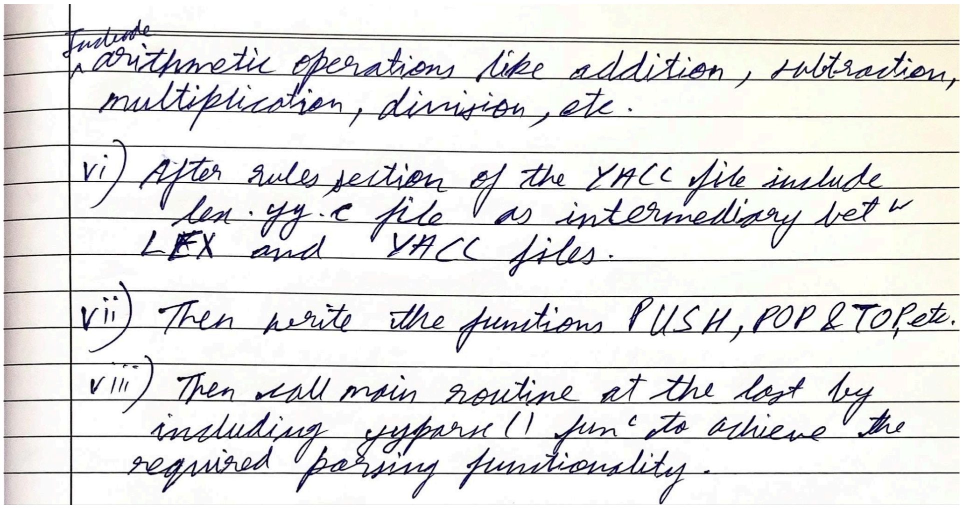
i) Create a LEX file first & create ~~an~~ Regular Expression for the digit.

ii) In the~~e~~ rules section, write rules for digit identification and identification of different operators.

iii) Also, in the rules section of LEX file, write rules for newline and for consideration of other details ~~ft~~ apart from identification of digits and the operators.

iv) Then, create a YACC file.

v) In the Declaration part of YACC file add rules for start symbol which will print the value of the postfix expression at the stack top, write rules for different.



Include arithmetic operations like addition, subtraction, multiplication, division, etc.

vi) After rules section of the YACC file include lex.yy.c file as intermediary betn LEX and YACC files.

vii) Then write the functions PUSH, POP & TOP, etc.

viii) Then call main routine at the last by including yyparse() func to achieve the required parsing functionality.

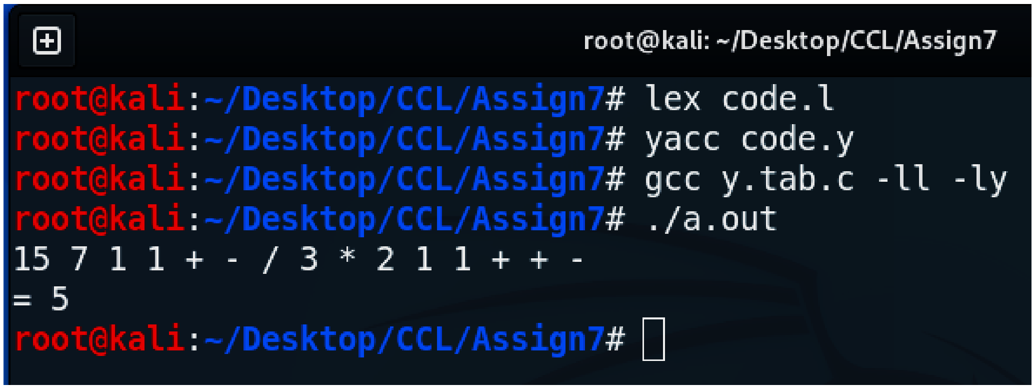
Code for LEX:

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

%}

%%

[0-9]+  { yylval = atoi(yytext); return ID; }
[\t\n ] { /* Ignore whitespace */ }
.       { return yytext[0]; }

%%

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

Code for YACC:

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

    int yylex(void);  // Declaration of yylex function
    void yyerror(const char *s);  // Declaration of yyerror function
    void push(int val);
    int pop();
    int top();
%}

%token ID

%%

S     : E  {printf("= %d\n",top());}
      ;
E     : E E '+' {push(pop() + pop());}
     | E E '-' {int temp = pop(); push(pop() - temp);}
     | E E '*' {push(pop() * pop());}
     | E E '/' {int temp = pop(); push(pop() / temp);}
     | ID      {push(yylval);}
     ;

%%

#include "lex.yy.c"

int st[100];
int i = 0;

void push(int val)
{
    assert(i < 100);
    st[i++] = val;
}

int pop()
{
    assert(i > 0);
    return st[--i];
}

int top()
{
    assert(i > 0);
    return st[i-1];
}

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

int main()
{
    yyparse();
    return 0;
}
```

Output:



```
root@kali:~/Desktop/CCL/Assign7# lex code.l
root@kali:~/Desktop/CCL/Assign7# yacc code.y
root@kali:~/Desktop/CCL/Assign7# gcc y.tab.c -ll -ly
root@kali:~/Desktop/CCL/Assign7# ./a.out
15 7 1 1 + - / 3 * 2 1 1 + + -
= 5
root@kali:~/Desktop/CCL/Assign7# 
```

* Conclusion: Thus, we have implemented parser for evaluating postfix expression.