LAB REPORT

COMPILER DESIGN LAB EXAM REPORT

Amrith M
Roll No. 10
S7 CSE

Contents

1 Problem 1

1.1 Problem

Design a string replacing system that takes as input a text file F and two patterns P and R and replaces every occurrences of P with R.

1.2 Approach

We do this program using lex. Identify each token and check if the token is the given input P. If the token value matches with P, output R, otherwise output the token value directly. We write a custom function check() in the lex file to do this.

1.3 Solution Program

```
/**

Author   : Amrith M
Code     : String Replace
INPUT    : input.txt
           File contains P and R followed by file contents in
 ↪  the new line

Execution : lex prog.l && g++ lex.yy.c && ./a.out < input.txt

*/

%{
    #include<stdio.h>
    #include<string.h>
    #include<stdlib.h>
    #include<iostream>
    #include<fstream>
    #include<string>
    char p[100], r[100];
    void check(char *name);
    using namespace std;
%}

letter [a-zA-Z]
digit [0-9]
```

```
%%
[ \t]                                   {   ;   }
[\n]                                    {   yylineno++;
 ↪  printf("\n");    }
({letter}|{digit})+                     {   check(yytext);   }
.                                       {   printf("%s", yytext) ;
 ↪  }
%%

void check(char *name)
{
    if(strcmp(name, p) == 0)
    {
        printf("%s ",r);
    }
    else
    {
        printf("%s ",name);
    }
}

int yywrap()
{
    return 1;
}

int main()
{
    scanf("%s %s",p,r);
    cout << "\nP : " << p << " R: " << r << endl;
    yylex();
    cout << "\n";
    return 0;
}
```

1.4 Input & Output

```
pc114@pc114:~/Lab-Exam/amrithm_TVE15CS010_Q1$ cat input.txt
brown black
quick brown fox runs over the lazy dog. brown is a good color. coffee is brown.
pc114@pc114:~/Lab-Exam/amrithm_TVE15CS010_Q1$ ./a.out < input.txt

P : brown R: black

quick black fox runs over the lazy dog .black is a good color .coffee is black
.
```

2 Problem 2

2.1 Problem

Design and implement a recursive descent parser for the following language $L = \{\ w \in \{a, b\}^*$ $|\ w$ is of the form $a^n b^n c^m d^m$, $n, m >= 1\ \}$

2.2 Approach

We write the Context Free Grammar for this language first. The CFG is

$$S \rightarrow AB$$
$$A \rightarrow aAb$$
$$A \rightarrow ab$$
$$B \rightarrow cBd$$
$$B \rightarrow cd$$

We implement a function in the C++ program for each non terminals which are A , B and S.

2.3 Solution Program

```
/**

Author  : Amrith M
Code    : RDP
INPUT   : input.txt

Grammar : Productions.txt
Logic   : Custom Function for Each Non Terminal is made
```

```
Execution : g++ rdp.cpp && ./a.out < input.txt

*/


#include <iostream>
#include <fstream>
#include <string>
#include <map>
using namespace std;

//string s is the input
string s;
//ptr is the position of the next symbol
int ptr = 0;
//error flag is set to 1 if there is error

int error = 0;

void A()
{
    if(s[ptr] == 'a')
    {
        ptr++;
        A();
        if(s[ptr] == 'b')
        {
            ptr++;
            return;
        }
        else
            error = 1;
    }
}

void B()
{
    if(s[ptr] == 'c')
    {
        ptr++;
        B();
        if(s[ptr] == 'd')
```

```
        {
            ptr++;
            return;
        }
        else
            error = 1;
    }
    return;
}

void S()
{
    if(s.size() < 4)
    {
        error = 1;
        return;
    }
    A();
    if(error == 0)
        B();
    if(ptr != s.size())
        error = 1;
}

int main()
{
    ifstream infile("input.txt");

    string temp;

    //getline reads each line from input.txt , resets all
     ↪ parameters and parses
    while(getline(infile,temp))
    {
        error = 0;
        ptr = 0;

        cout << "\n" << temp;

        s = temp;

        S();
```

```
        if(error == 0)
            cout << "\tSuccessfully Parsed";
        else
            cout << "\tError in Parsing";
    }


    return 0;
}
```

2.4 Input & Output

```
pc114@pc114:~/Lab-Exam/amrithm_TVE15CS010_Q2$ cat input.txt
aabbccdd
abcd
aabbcd
aaabbbcccddd
aabbccd
ab
cd
abb
pc114@pc114:~/Lab-Exam/amrithm_TVE15CS010_Q2$ ./a.out

aabbccdd        Successfully Parsed
abcd    Successfully Parsed
aabbcd  Successfully Parsed
aaabbbcccddd    Successfully Parsed
aabbccd Error in Parsing
ab      Error in Parsing
cd      Error in Parsing
abb     Error in Parsingpc114@pc114:~/Lab-Exam/amrithm_TVE15CS010_Q2$
```

3 Problem 3

3.1 Problem

Design and implement a syntax directed translation scheme for converting an infix arithmetic expression formed of variables over English alphabet and arithmetic operators *, +, -, / where + has the highest precedence followed by - and then by * and / to its equivalent postfix expression.

3.2 Approach

We write the corresponding Syntax Directed Translation in the YACC program to generate the correct postfix expression.

3.3 Solution Program (LEX)

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include "y.tab.h"
%}

%option noyywrap

%%
[0-9]+                  {   yylval.val = atoi(yytext); return
↪  NUM; }
[a-zA-Z_][a-zA-Z0-9]*   {   yylval.name = strdup(yytext);
↪  return ID; }
[-+*/]                  {   return yytext[0]; }
[ \t\n]                 {   ;   }
%%
```

3.4 Solution Program (YACC)

```
/**

Author  : Amrith M
Code    : INFIX TO POSTFIX CONVERSION
INPUT   : input.txt

Grammar : Given Below

Execution : sh exec.sh && ./a.out < input.txt

*/

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

    void yyerror(char *err);
```

```
%}

//Union stores either name or val of an item at a time

%union
{
    char *name;
    int val;
};

//Precedence Given based on input

%left '*' '/'
%left '+' '-'

%token ID NUM
%type<name> ID
%type<val> NUM
%type<val> expression

%start prog

%%
prog : expression;
expression : expression '+' expression  { $$ = $1 + $3;
 ↪  printf("+");}
            | expression '*' expression { $$ = $1 * $3;
            ↪  printf("*");}
            | expression '-' expression { $$ = $1 - $3;
            ↪  printf("-");}
            | expression '/' expression { $$ = $1 / $3;
            ↪  printf("/");}
            | ID    {   printf("%s",$1); }
            | NUM   {   printf("%d",$1); }
            ;
%%

void yyerror(char *name)
{
    printf("\nError: %s", name);
}

int yywrap()
```

```
{
    return 1;
}

int main()
{
    //yyparse askes yylex for tokens and parses it

    yyparse();

    printf("\n");

    return 0;
}
```

3.5 Script To Execute

```
#!/bin/bash

rm lex.yy.c
rm y.tab.c
rm y.tab.h
yacc -d conv.y
lex conv.l
gcc lex.yy.c y.tab.c
./a.out < input.txt
```

3.6 Input & Output

```
pc114@pc114:~/Lab-Exam/amrithm_TVE15CS010_Q3$ cat input.txt
2-3*4+a-b+c*d/e
pc114@pc114:~/Lab-Exam/amrithm_TVE15CS010_Q3$ ./a.out < input.txt
23-4a+b-c+*d*e/
```