Name : Dasari Jayanth

Roll No : 2019BCS-016

Course :System Software Lab

Course Code : BCCS 3106 - 2021

# **Assignment Number - 3a**

Date Of Submission: **23-09-2021**

**Aim:**

Write a program for lex/flex scanner generator to determine the character with the highest frequency in the given string.

**Procedure:**

First we initialize an array z of size 129 since there are 128 standard ascii characters and we are trying to find the character with highest frequency in the user string. In the next step using predefined terminology (regex expression form). We calculate and directly store the occurrences of each character till 128 ascii values if a character is not present we assign value ‘0’. Now, we initialize a variable char\_freq with the INT\_MIN value(max possible min value) and then we compare all the frequency values of all characters stored in array z using a for loop and if any character has higher occurrence than the char\_freq then we store that value in char\_freq and its index(character) in variable ‘character’. Finally, the max occurrence is stored in char\_freq and char ascii value or index in character variables respectively.

**Code:**

%{

#include <limits.h>

int z[129]={0};

%}

%%

[a-zA-Z]? {z[(int)yytext[0]]++; }

\n {return 0;}

%%

int yywrap() {}

int main(){

int i, char\_freq=INT\_MIN, character;

printf("Enter the string: \n");

yylex();

for(i = 0;i < 129;i++){

if(char\_freq < z[i]){

char\_freq = z[i];

character= i;

}

}

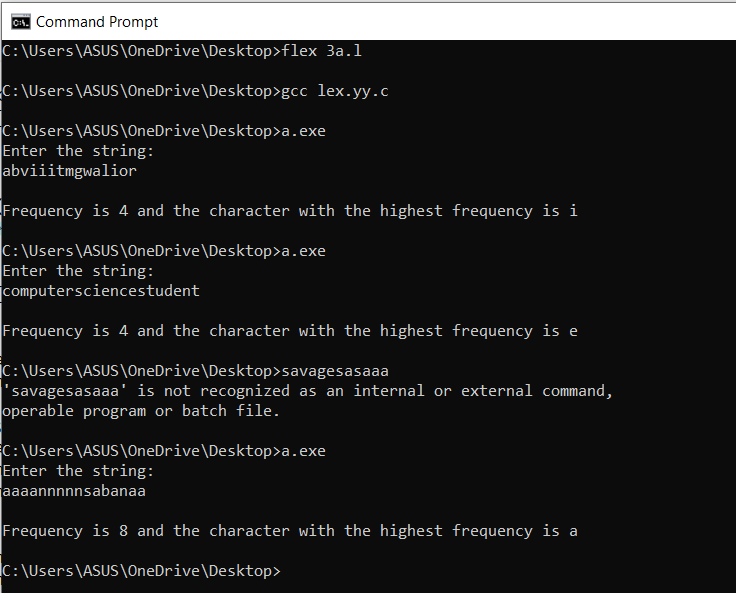
printf("\nFrequency is %d and the character with the highest frequency is %c",char\_freq, character);

printf("\n");

return 0;

}

**Input/Output -3a:**



**Inference:**

Input String: ‘abviiitmgwalior’.

Output: frequency:4 character with high frequency: ‘i’

Reasoning: ‘i’ is the character with highest frequency ‘4’ in the input string.

Input String: ‘aaaannnnnsabanaa’.

Output: frequency:8 character with high frequency: ‘a’

Reasoning: ‘a’ is the character with highest frequency ‘8’ in the input string.

**Hence**, we can conclude that the code written above returns the character with highest frequency and its frequency in the given string.(by user).

# 

# **Assignment Number - 3b**

Date Of Submission: **23-09-2021**

**Aim:**

Write a program for the lex/flex scanner generator to do Infix to Postfix Conversion.

**Procedure:**

**yytext** is the pointer that points at the string or char or word we entered.

First we initialize the stack and stack functions like push, pop, top, priority. Priority is used to find the precedence among operators[‘(‘, ’)’, ‘+’, ‘-’, ‘\*’, ‘/’]. Pop is to delete top ele from stack. Push is to enter (push) ele inside the stack.(it will be at the top). We scan the string from left to right.

If the input is operand then directly print it else if it is operator then

1. If the stack is empty, push it to the stack.
2. If its precedence value is greater than the precedence value of the character on the element on top of stack, push it to stack.
3. If its precedence value is lower or equal then pop the top ele from stack and push the current ele into stack which will be new top.

While considering the parenthesis if we encounter ‘(‘ then directly add it to the stack and the remaining process is normal. If we don't encounter the ’)’ then the expression is invalid. If we encounter ‘)’ then all the operators inside parentheses will be printed in reverse order.

Repeat the above steps until the total expression is read and traverse the stack at the end.

**Code:**

%{

char stack[100];

int t=0;

void push(char);

void pop();

char top();

int priority(char c);

%}

%%

[a-zA-Z]? {printf("%c",yytext[0]);}

[+\-\\*\/\(\)\^] { char op=yytext[0];

if(t==0||op=='(')

push(op);

else if(op==')'){

while(top()!='('){

printf("%c",top());

pop();

}

pop();

}

else if(priority(op)>=priority(top())){

push(op);

}

else if(priority(op)<=priority(top())){

while(priority(op)<=priority(top())){

printf("%c",top());

pop();

}

push(op);

}

}

[\n] {

int i;

for(i=t-1;i>=0;i--)

if(stack[i]!='('&&stack[i]!=')'){

printf("%c",stack[i]);

}

t=0;

}

%%

int yywrap(){}

int main(){

yylex();

}

void pop(){

t--;

}

char top(){

return stack[t-1];

}

void push(char c){

stack[t]=c;

t++;

}

int priority(char c){

switch(c){

case '(':return 0;

case ')':return 0;

case '+':return 1;

case '-':return 1;

case '\*':return 2;

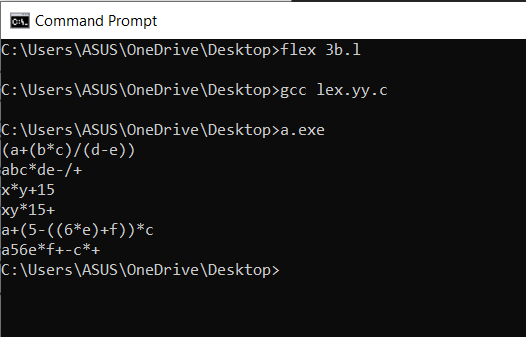
case '/':return 2;

case '^':return 3;

}

}

**Input/Output -3b:**

****

**Inference:**

Input : x\*y+15

Output : xy\*15+

Reasoning: x is directly printed. ‘\*‘ is pushed into the stack, ‘y’ is also directly printed. Precedence of ‘+’ is less than ‘\*’ so ‘\*’ is popped and printed and ‘+’ is pushed into the stack. ’1’, ’5’ are directly printed. Expression is completely read, now the stack is traversed if any elements are there.

**Hence**, we can conclude that the above written code gives the postfix expression of the given infix expression as the result.(by converting it using precedences)