# **Compiler Construction**

# **Lab 4**

## Task 1: Implement Stack Using Array

#include <stdio.h>

int stack[100], choice, n, top, x, i;

void push(void);

void pop(void);

void display(void);

int main()

{

//clrscr();

top = -1;

printf("\n Enter the size of STACK[MAX=100]:");

scanf\_s("%d", &n);

printf("\n\t STACK OPERATIONS USING ARRAY");

printf("\n\t--------------------------------");

printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");

do

{

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

scanf\_s("%d", &choice);

switch (choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

} while (choice != 4);

return 0;

}

void push()

{

if (top >= n - 1)

{

printf("\n\tSTACK is over flow");

}

else

{

printf(" Enter a value to be pushed:");

scanf\_s("%d", &x);

top++;

stack[top] = x;

}

}

void pop()

{

if (top <= -1)

{

printf("\n\t Stack is under flow");

}

else

{

printf("\n\t The popped elements is %d", stack[top]);

top--;

}

}

void display()

{

if (top >= 0)

{

printf("\n The elements in STACK \n");

for (i = top; i >= 0; i--)

printf("\n%d", stack[i]);

printf("\n Press Next Choice");

}

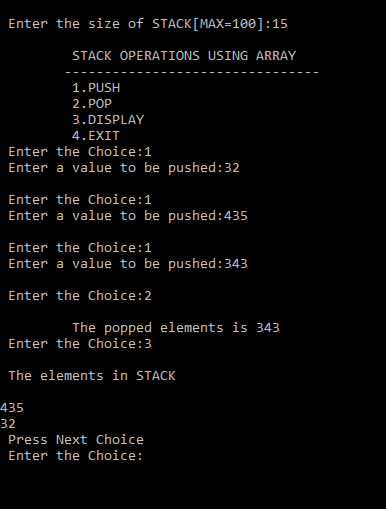
else

{

printf("\n The STACK is empty");

}

}



## Task 2: Implement Queue Using Array

#include<stdio.h>

#include<conio.h>

#define n 5

void main()

{

int queue[n], ch = 1, front = 0, rear = 0, i, j = 1, x = n;

//clrscr();

printf("Queue using Array");

printf("\n1.Insertion \n2.Deletion \n3.Display \n4.Exit");

while (ch)

{

printf("\nEnter the Choice:");

scanf\_s("%d", &ch);

switch (ch)

{

case 1:

if (rear == x)

printf("\n Queue is Full");

else

{

printf("\n Enter no %d:", j++);

scanf\_s("%d", &queue[rear++]);

}

break;

case 2:

if (front == rear)

{

printf("\n Queue is empty");

}

else

{

printf("\n Deleted Element is %d", queue[front++]);

x++;

}

break;

case 3:

printf("\n Queue Elements are:\n ");

if (front == rear)

printf("\n Queue is Empty");

else

{

for (i = front; i<rear; i++)

{

printf("%d", queue[i]);

printf("\n");

}

break;

case 4:

exit(0);

default:

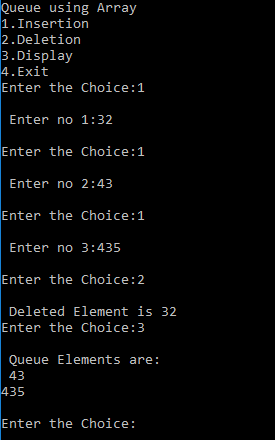
printf("Wrong Choice: please see the options");

}

}

}

}



## Task 3 and 4:

## Generate parser tree

## Find terminals and non-terminals of above tree

/\*

GRAMMAR

Exp -> Exp + Term

|Exp - Term

|Term

Term -> Term / Factor

|Term \* Factor

|Factor

Factor -> id |operator

id is any alphabet or number

\*/

#include <iostream>

#include <cctype>

using namespace std;

struct node {

char node\_type;

char data;

int no\_of\_child\_nodes = -1;

node \*next[10];

node\* prev;

};

int input\_type(char ch) {

//return 1 if ch is an identifier

//return 2 if ch is +,-

//return 3 if ch is /,\*

if (ch == '+' || ch == '-') {

return 1;

}

else if (ch == '/' || ch == '\*') {

return 2;

}

else{

return 3;

}

}

void main() {

int no\_of\_terminals = 0, no\_of\_non\_terminals = 0;

node\* n;

node\* h;

node\* t;

char in[10];

cout << "Enter ";

cin >> in;

n = new node;

n->node\_type = 'E';

t = n;

h = n;

no\_of\_non\_terminals++;

int j;

for (int i = 0; i < strlen(in); i++) {

//Always keeping track of where t is and moving accordingly

if (t->node\_type == 'E') {

switch (input\_type(in[i])) {

// Expression ---> operator (+,-)

case 1: // read +,-

cout << "E ---> ";

cout << in[i];

cout << "\n\n";

n = new node; //create new node

n->data = in[i]; // data = current input

n->next[0] = NULL; //leaf node (aka terminal)

n->prev = t; //new node points to root node

j = t->no\_of\_child\_nodes; // t is pointing to E (the root node)

j++; // the number of child nodes = j+1

t->next[j] = n; // root node now points to a new leaf node

t->no\_of\_child\_nodes = j; //increment no\_of\_child\_nodes of root node

no\_of\_terminals++;

break;

// Expression ---> Term ---> operator (/,\*)

case 2: // read \*,/

n = new node;

n->node\_type = 'T';

n->prev = t; // New T node points to root node E

j = t->no\_of\_child\_nodes;

j++;

t->next[j] = n; // E points to new node T

t->no\_of\_child\_nodes = j;// A new child node of E

t = n; // t points to new node

cout << "T ---> ";

cout << in[i];

cout << "\n\n";

n = new node;

n->data = in[i];

n->next[0] = NULL;

n->prev = t;

t->no\_of\_child\_nodes = j;// A new child node of T

j = t->no\_of\_child\_nodes;

j++;

t->next[j] = n; //T points to new node

no\_of\_non\_terminals++;

no\_of\_terminals++;

break;

// Expression ---> Term ---> Factor ---> identifier

case 3: //id, num, or op

n = new node;

n->node\_type = 'T';

n->prev = t; // New T node points to root node E

j = t->no\_of\_child\_nodes;

j++;

t->next[j] = n; // E points to new node T

t->no\_of\_child\_nodes = j;// A new child node of E

t = n; // t points to new node

n = new node; //creating new Factor node

n->node\_type = 'F';

n->prev = t;

j = t->no\_of\_child\_nodes;

j++;

t->next[j] = n; //T points to new node F

t->no\_of\_child\_nodes = j;

t = n;

cout << "E ---> T ---> F ---> ";

cout << in[i];

cout << "\n\n";

n = new node;

n->data = in[i];

n->next[0] = NULL;

n->prev = t;

j = t->no\_of\_child\_nodes;

j++;

t->next[j] = n; //F points to new node

t->no\_of\_child\_nodes = j;

t = n;

no\_of\_non\_terminals+=2;

no\_of\_terminals++;

break;

}

}

else if (t->node\_type == 'T') {

switch (input\_type(in[i])) {

case 1:

t = t->prev; //t now points to E. t is at root node

cout << "E ---> ";

cout << in[i];

cout << "\n\n";

n = new node; //create new node

n->data = in[i]; // data = current input

n->next[0] = NULL; //leaf node (aka terminal)

j = t->no\_of\_child\_nodes; // t is pointing to E (the root node)

j++; // the number of child nodes = j+1

t->next[j] = n; // root node now points to a new leaf node

t->no\_of\_child\_nodes = j; //increment no\_of\_child\_nodes of root node

n->prev = t; //new node points to root node

no\_of\_terminals++;

break;

case 2:

cout << "T ---> ";

cout << in[i];

cout << "\n\n";

n = new node; //create new node

n->data = in[i]; // data = current input

n->next[0] = NULL; //leaf node (aka terminal)

j = t->no\_of\_child\_nodes; // t is pointing to E (the root node)

j++; // the number of child nodes = j+1

t->next[j] = n; // root node now points to a new leaf node

t->no\_of\_child\_nodes = j; //increment no\_of\_child\_nodes of root node

n->prev = t; //new node points to root node

no\_of\_terminals++;

break;

case 3:

n = new node; //creating new Factor node

n->node\_type = 'F';

n->prev = t;

j = t->no\_of\_child\_nodes;

j++;

t->next[j] = n; //T points to new node F

t->no\_of\_child\_nodes = j;

t = n;

cout << "E ---> T ---> F ---> ";

cout << in[i];

cout << "\n\n";

n = new node;

n->data = in[i];

n->next[0] = NULL;

n->prev = t;

j = t->no\_of\_child\_nodes;

j++;

t->next[j] = n; //F points to new node

t->no\_of\_child\_nodes = j;

t = n;

no\_of\_non\_terminals += 2;

no\_of\_terminals++;

break;

}

}

else if (t->node\_type == 'F') {

switch (input\_type(in[i])) {

case 1:

t = t->prev; //goes from F to T

t = t->prev; //goes form T to E

cout << "E ---> ";

cout << in[i];

cout << "\n\n";

n = new node; //create new node

n->data = in[i]; // data = current input

n->next[0] = NULL; //leaf node (aka terminal)

j = t->no\_of\_child\_nodes; // t is pointing to E (the root node)

j++; // the number of child nodes = j+1

t->next[j] = n; // root node now points to a new leaf node

t->no\_of\_child\_nodes = j; //increment no\_of\_child\_nodes of root node

n->prev = t; //new node points to root node

no\_of\_terminals++;

break;

case 2:

t = t->prev; //goes form F to T

cout << "T ---> ";

cout << in[i];

cout << "\n\n";

n = new node;

n->data = in[i];

n->next[0] = NULL;

n->prev = t;

t->no\_of\_child\_nodes = j;// A new child node of T

j = t->no\_of\_child\_nodes;

j++;

t->next[j] = n; //T points to new node

no\_of\_terminals++;

break;

case 3:

cout << "F ---> ";

cout << in[i];

cout << "\n\n";

n = new node;

n->data = in[i];

n->next[0] = NULL;

n->prev = t;

t->no\_of\_child\_nodes = j;// A new child node of T

j = t->no\_of\_child\_nodes;

j++;

t->next[j] = n; //T points to new node

no\_of\_terminals++;

break;

}

}

}

cout << "Number of Non-Terminals = ";

cout << no\_of\_non\_terminals;

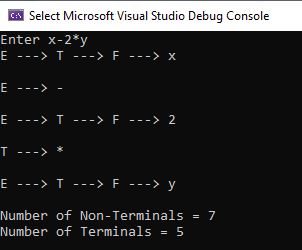
cout << "\n";

cout << "Number of Terminals = ";

cout << no\_of\_terminals;

cout << "\n";

}



## Task 5: Implement Flex (any example)

/\*\*\* Definition Section has one variable   
which can be accessed inside yylex()    
and main() \*\*\*/  
%{   
int count = 0;   
%}   
    
/\*\*\* Rule Section has three rules, first rule    
matches with capital letters, second rule   
matches with any character except newline and    
third rule does not take input after the enter\*\*\*/  
%%   
[A-Z] {printf("%s capital letter\n", yytext);   
       count++;}   
.     {printf("%s not a capital letter\n", yytext);}   
\n    {return 0;}   
%%   
    
/\*\*\* Code Section prints the number of   
capital letter present in the given input\*\*\*/  
int yywrap(){}   
int main(){   
    
// Explanation:   
// yywrap() - wraps the above rule section   
/\* yyin - takes the file pointer    
          which contains the input\*/  
/\* yylex() - this is the main flex function   
          which runs the Rule Section\*/  
// yytext is the text in the buffer   
    
// Uncomment the lines below    
// to take input from file   
// FILE \*fp;   
// char filename[50];   
// printf("Enter the filename: \n");   
// scanf("%s",filename);   
// fp = fopen(filename,"r");   
// yyin = fp;   
    
yylex();   
printf("\nNumber of Captial letters "   
      "in the given input - %d\n", count);   
    
return 0;   
}

