**Compiler Design**

* **Assignment – 7: Implement LALR parser**

**Implement the functionalities of LALR parser using C language**

* **CODE:**

**#include<stdio.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**#include<string.h>**

**void push(char \*,int \*,char);**

**char stacktop(char \*);**

**void isproduct(char,char);**

**int ister(char);**

**int isnter(char);**

**int isstate(char);**

**void error();**

**void isreduce(char,char);**

**char pop(char \*,int \*);**

**void printt(char \*,int \*,char [],int);**

**void rep(char [],int);**

**struct action**

**{**

**char row[6][5];**

**};**

**const struct action A[12]={**

**{"sf","emp","emp","se","emp","emp"},**

**{"emp","sg","emp","emp","emp","acc"},**

**{"emp","rc","sh","emp","rc","rc"},**

**{"emp","re","re","emp","re","re"},**

**{"sf","emp","emp","se","emp","emp"},**

**{"emp","rg","rg","emp","rg","rg"},**

**{"sf","emp","emp","se","emp","emp"},**

**{"sf","emp","emp","se","emp","emp"},**

**{"emp","sg","emp","emp","sl","emp"},**

**{"emp","rb","sh","emp","rb","rb"},**

**{"emp","rb","rd","emp","rd","rd"},**

**{"emp","rf","rf","emp","rf","rf"}**

**};**

**struct gotol**

**{**

**char r[3][4];**

**};**

**const struct gotol G[12]={**

**{"b","c","d"},**

**{"emp","emp","emp"},**

**{"emp","emp","emp"},**

**{"emp","emp","emp"},**

**{"i","c","d"},**

**{"emp","emp","emp"},**

**{"emp","j","d"},**

**{"emp","emp","k"},**

**{"emp","emp","emp"},**

**{"emp","emp","emp"},**

**};**

**char ter[6]={'i','+','\*',')','(','$'};**

**char nter[3]={'E','T','F'};**

**char states[12]={'a','b','c','d','e','f','g','h','m','j','k','l'};**

**char stack[100];**

**int top=-1;**

**char temp[10];**

**struct grammar**

**{**

**char left;**

**char right[5];**

**};**

**const struct grammar rl[6]={**

**{'E',"e+T"},**

**{'E',"T"},**

**{'T',"T\*F"},**

**{'T',"F"},**

**{'F',"(E)"},**

**{'F',"i"},**

**};**

**void main()**

**{**

**char inp[80],x,p,dl[80],y,bl='a';**

**int i=0,j,k,l,n,m,c,len;**

**printf(" Enter the input :");**

**scanf("%s",inp);**

**len=strlen(inp);**

**inp[len]='$';**

**inp[len+1]='\0';**

**push(stack,&top,bl);**

**printf("\n stack \t\t\t input");**

**printt(stack,&top,inp,i);**

**do**

**{**

**x=inp[i];**

**p=stacktop(stack);**

**isproduct(x,p);**

**if(strcmp(temp,"emp")==0)**

**error();**

**if(strcmp(temp,"acc")==0)**

**break;**

**else**

**{**

**if(temp[0]=='s')**

**{**

**push(stack,&top,inp[i]);**

**push(stack,&top,temp[1]);**

**i++;**

**}**

**else**

**{**

**if(temp[0]=='r')**

**{**

**j=isstate(temp[1]);**

**strcpy(temp,rl[j-2].right);**

**dl[0]=rl[j-2].left;**

**dl[1]='\0';**

**n=strlen(temp);**

**for(k=0;k<2\*n;k++)**

**pop(stack,&top);**

**for(m=0;dl[m]!='\0';m++)**

**push(stack,&top,dl[m]);**

**l=top;**

**y=stack[l-1];**

**isreduce(y,dl[0]);**

**for(m=0;temp[m]!='\0';m++)**

**push(stack,&top,temp[m]);**

**}**

**}**

**}**

**printt(stack,&top,inp,i);**

**}while(inp[i]!='\0');**

**if(strcmp(temp,"acc")==0)**

**printf(" \n accept the input ");**

**else**

**printf(" \n do not accept the input ");**

**getch();**

**}**

**void push(char \*s,int \*sp,char item)**

**{**

**if(\*sp==100)**

**printf(" stack is full ");**

**else**

**{**

**\*sp=\*sp+1;**

**s[\*sp]=item;**

**}**

**}**

**char stacktop(char \*s)**

**{**

**char i;**

**i=s[top];**

**return i;**

**}**

**void isproduct(char x,char p)**

**{**

**int k,l;**

**k=ister(x);**

**l=isstate(p);**

**strcpy(temp,A[l-1].row[k-1]);**

**}**

**int ister(char x)**

**{**

**int i;**

**for(i=0;i<6;i++)**

**if(x==ter[i])**

**return i+1;**

**return 0;**

**}**

**int isnter(char x)**

**{**

**int i;**

**for(i=0;i<3;i++)**

**if(x==nter[i])**

**return i+1;**

**return 0;**

**}**

**int isstate(char p)**

**{**

**int i;**

**for(i=0;i<12;i++)**

**if(p==states[i])**

**return i+1;**

**return 0;**

**}**

**void error()**

**{**

**printf(" error in the input ");**

**exit(0);**

**}**

**void isreduce(char x,char p)**

**{**

**int k,l;**

**k=isstate(x);**

**l=isnter(p);**

**strcpy(temp,G[k-1].r[l-1]);**

**}**

**char pop(char \*s,int \*sp)**

**{**

**char item;**

**if(\*sp==-1)**

**printf(" stack is empty ");**

**else**

**{**

**item=s[\*sp];**

**\*sp=\*sp-1;**

**}**

**return item;**

**}**

**void printt(char \*t,int \*p,char inp[],int i)**

**{**

**int r;**

**printf("\n");**

**for(r=0;r<=\*p;r++)**

**rep(t,r);**

**printf("\t\t\t");**

**for(r=i;inp[r]!='\0';r++)**

**printf("%c",inp[r]);**

**}**

**void rep(char t[],int r)**

**{**

**char c;**

**c=t[r];**

**switch(c)**

**{**

**case 'a': printf("0");**

**break;**

**case 'b': printf("1");**

**break;**

**case 'c': printf("2");**

**break;**

**case 'd': printf("3");**

**break;**

**case 'e': printf("4");**

**break;**

**case 'f': printf("5");**

**break;**

**case 'g': printf("6");**

**break;**

**case 'h': printf("7");**

**break;**

**case 'm': printf("8");**

**break;**

**case 'j': printf("9");**

**break;**

**case 'k': printf("10");**

**break;**

**case 'l': printf("11");**

**break;**

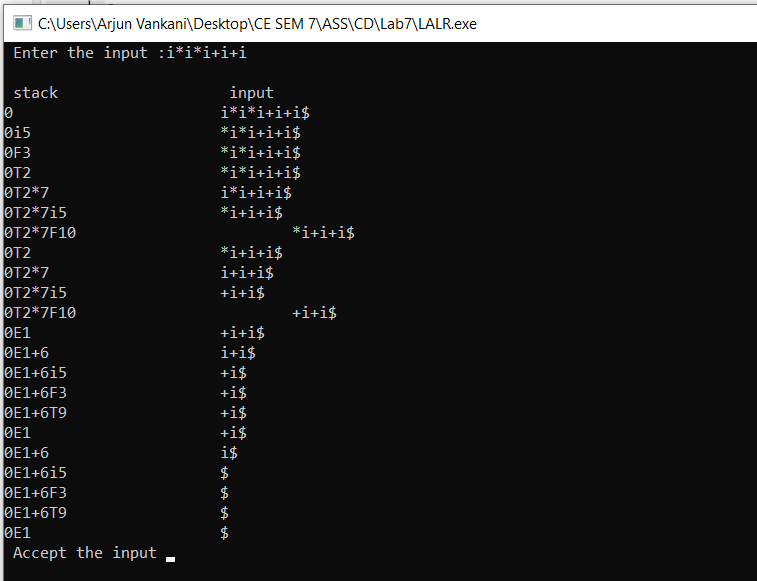
**default :printf("%c",t[r]);**

**break;**

**}**

**}**

**Output:**

****