13-10-2021

Pila estática

Practica: 4

Materia: Estructura de datos

Sección: D01.

Código: 216584703

Carrera: Ingeniería en computación.

Nombre alumno: Padilla Pérez Jorge Daray

Nombre profesor: Julio Esteban Valdes Lopez

Introducción

La verdad no he podido acabarla me falta todavía pensar bien como funcionaria, además de que no se exactamente que se tiene que hacer, si se tiene que hacer lo de torres de hanoi, lo que cambiar las notaciones o ambas etc.

Pantallazos

Me falta todavia falla esperare a que pueda funcionar porque todavia no entiendo muy bien para explicar los pantallazos, nomas porque se sube hoy la practica pero espero acabarlo en los siguientes dias.

Conclusión

Me falta acabarla todavía no funciona.

Código fuente

```
1   #include <iostream>
2   #include <string.h>
3   #define TAMMAX 500
4
5   using namespace std;
6
7   void menu();
8
9   typedef char tipo_dato;
10
11  struct pila{
12   tipo_dato datos[TAMMAX];
13   void inicializa();
14   bool vacia();
15   bool llena();
16   int tope;
17   void push(tipo_dato elem, int tope);
18   void pop(int tope);
19   int top(int tope);
20   char infijo_posfijo(char ei[]);
21   int jerarquia_operadores();
22   bool Esoperador();
23  //void imprimir_torres(tipo_dato numd);
24   //void hanoi(struct pila,struct pilaDest,struct pilaAux,int numdiscos);
25   pila(){
26   inicializa();
27   }
28   };
```

```
29
30   void pila::inicializa(){
31   tope = -1;
32   }
33
34  bool pila::vacia(){
35   if (tope == -1)
36   {
37   return true;
38   }
39   else{
40   return false;
41   }
42  }
43
44   bool pila::llena(){
45   if (tope == TAMMAX -1 ){
46   return true;
47   }
48   else{
49   return false;
50  }
51   }
52
53   //Ingresar los discos a la torre
54   void pila::push(tipo_dato elem, int tope){
55   if (llena()){
56   cout<<"desbordamiento de datos"<<endl;
57   system("pause");
58   return;
```

```
59  }
60   else{
61   tope = tope + 1;
62   datos[tope] = elem;
63   }
64   }
65
66   //Sacar los discos a la torre
67   void pila::pop(int tope){
68   if (vacia()){
69  cout<<"Insuficiencia de datos"<<endl;
70   system("pause");
71   return;
72  }
73   else{
74   tope = tope - 1;
75   }
76  }
77
78   int pila::top(int tope){
79   if (vacia()){
80   cout<<"Insuficiencia de datos"<<endl;
81  system("pause");
82  return false;
83   }
84   else{
85   return datos[tope];
86   }
87   }
88
```

```
89   int pila::jerarquia_operadores(){
90   if (datos == "+"||datos == "-"){
91      return 1;
92   }
93   if (datos == "*"||datos == "/"){
94      return 2;
95   }
96   if (datos == "^"){
97      return 3;
98   }
99   else{
100     return 0;
101  }
102  }
103
104  bool pila::Esoperador(){
105  if (datos=="+" || datos=="-" || datos=="*" || datos=="/" || datos=="^" ||
106 datos=="(" || datos==")"){
107     return true;
108  }
109  else{
110     return false;
111  }
112  }
113
114  char pila::infijo_posfijo(char ei[]){
115      char ep[TAMMAX];
116      struct pila mi_pila;
117     int i = 0,j=0;
118             while(ei[i] != '\0')
```

```
119             {
120                 if(ei[i] >= 'a' && ei[i] <= 'z' || ei[i]>='A' && ei[i] <=
121 'Z')
122                 {
123                     ep[j] = ei[i];
124                     i++;
125                     j++;
126                 }
127                 else
128                     if(ei[i]=='(')
129                     {
130                         mi_pila.push(ei[i], tope);
131                         i++;
132                     }
133                 else
134                      if(ei[i]==')')
135                      {
136                         while( mi_pila.top(tope) != '(')
137                         {
138                             ep[j]= tope;
139                             j++;
140                         }
141                             if(mi_pila.top(tope) == '(')
142                                {
143                                    mi_pila.pop(tope);
144                                }
145                             i++;
146                 }
147
148                     else{
```

```
                        if(ei[i]=='+' || ei[i]=='-' || ei[i]=='*' ||
ei[i]=='/' || datos=="^")
                        {
                            if(ei[i]=='+' || ei[i] == '-')
                            {
                                while(!vacia() && mi_pila.top(tope) != '(')
                                    {
                                        ep[j]=tope;
                                        j++;
                                    }
                                mi_pila.push(ei[i],tope);
                                i++;
                            }
                            else if (ei[i]=='*' || ei[i]=='/')
                            {

                                    while(!vacia() && mi_pila.top(tope) !=
'(' && (mi_pila.top(tope)=='*'|| mi_pila.top(tope)=='/'))
                                    {
                                        ep[j]=(tope);
                                        j++;
                                    }

                                    mi_pila.push(ei[i],tope);
                                    i++;
                            }
                            else
                            {
                                if(ei[i]=='^')
                                 {
```

```
                                    while(!vacia() && mi_pila.top(tope) !=
'(' )
                                    {
                                        ep[j]=(tope);
                                        j++;
                                    }

                                    mi_pila.push(ei[i],tope);
                                    i++;
                                }
                            }


                        }

            }
            }
            while(!vacia())
            {
                ep[j]= tope;
                j++;
            }
            return ep[j];
        }


/*void pila::hanoi(struct pila,struct pilaDest,struct pilaAux,int numdiscos){

if (numdiscos == 0){
return;
```

```
209  }
210   else{
211   hanoi (pilaOrig, pilaAux, pilaDest, numdiscos - 1);
212
213   push (top (pilaOrig), pilaDest);
214   pop (pilaOrig);
215
216   hanoi (pilaAux, pilaDest, pilaOrig, numdiscos - 1);
217   }
218   }
219
220  void pila::imprimir_torres(tipo_dato numd){
221  int i,j;
222      for (i=1;i<=numd;i++){
223          for (j=1;j<=numd-i;j++){
224           cout<<" ";
225       }
226          for(j=1;j<=2*i-1;j++){
227              cout<<"*";
228          }cout<<endl;
229      }
230  }*/
231
232  struct pila mi_pila;
233   int main()
234  {
235   int opc=0;
236   do{
237   system("cls");
238   menu();
```

```
239    cout<<"Continuar 1 salir 2:"<<endl;cin>>opc;
240    }while(opc!=2);
241    system("pause>>cls");
242    return 0;
243    }
244
245    void menu(){
246        char expresion_infix[TAMMAX];
247       char expresion_postfix[TAMMAX];
248       cout << " Longitud maxima de la expresion: "<< TAMMAX -1 << "\n" << endl;
249       cout << "\nEscribe tu expresion en infix:" << endl;
250       cin>> expresion_infix;
251       mi_pila.infijo_posfijo(expresion_infix); //
252       cout << "\nLa cadena infija \n"  << expresion_infix << endl;
253       cout << "\nConvertida en postfix es: \n"  << expresion_postfix << endl;
254   /* int numdiscos;
255    cout<<"******\tTORRE DE HANOI\t******"<<endl;
256
257    cout<<"\nIngrese la cantidad de disco(s): "<<endl;cin>>numdiscos;
258    if (numdiscos<=0){
259       cout<<"Ingrese un numero entero positivo arriba de 0"<<endl;
260       system("pause");
261       return;
262    }
263    mi_pila.imprimir_torres(numdiscos);*/
264
265
266   cout<<("\n\t");
267    system("pause");
268    }
```