TALLER

NATALIA ISABEL HERNANDEZ NAVEROS

CORPORACIÓN DE ESTUDIOS

TECNOLÓGICOS DEL NORTE DEL VALLE

TÉCNICO PROFESIONAL EN PROGRAMACIÓN

DE APLICACIONES INFORMÁTICAS

CARTAGO 2017

**Infijo a Posfijo**

1. (x + y) \* (z ^ (s + t) + u) - v

(x y +) \* (z ^ (s t +) + u) - v

(x y +) \* (z s t + ^) + u) - v

((x y +) \* (z s t + ^ u +)) - v

x y + z s t + ^ u + \* - v

**x y + z s t + ^ u + \* v –**

1. a + ((x + y) \* (w + v)) ^ (i + j)

a + ((x y +) \* (w v +)) ^ (i + j)

a + ((x y + w v + \*) ^ (i + j))

a + (x y + w v + \* i j + ^)

**a x y + w v + \* i j + ^ +**

1. (a + b) \* (c + d)

(a b +) \* (c d +)

**a b + c d \***

1. ((a + b) - c)

((a b +) - c)

**a b + c –**

1. (a + b) \* (c - d) \* e \* f

((a b +) \* (c d -) \* e \* f)

(((a b + c d - \*) \* e) \* f)

((a b + c d - \* e \*) \* f)

**a b + c d - \* e \* f \***

1. (a + b) \* (c / (d - e) + f) - g

(a b +) \* ((c / (d e -)) + f) - g

(a b +) \* ((c d e - /) + f) - g

((a b +) \* (c d e - / f +)) - g

((a b + c d e - f + \*) - g)

**a b + c d e - / f + \* g -**

1. ((a ^ b) ^ c)

(a b ^) ^ c

**a b ^ c ^**

1. a ^ b \* c - d + e / f / (g + h)

((((a ^ b) \* c) - d) + ((e / f) / (g h +)))

((((a b ^) \* c) - d) + ((e f /) / (g h +)))

((((a b ^ c \*) - d) + ((e f /) / (g h +)))

(((a b ^ c \*) - d) + (e f / g h + /))

((a b ^ c \* d -) + (e f / g h + /))

**a b ^ c \* d - e f / g h + / +**

1. a + (((b - c) \* (d - e) + f) / g) ^ (h - j)

a + (((b c -) \* (d e -) + f) / g) ^ (h j -)

a + ((b c - d e - \* f) / g) ^ (h j -)

a + ((b c - d e - \* f g /) ^ (h j -))

(a + (b c - d e - \* f g /)) ^ (h j -)

((a b c - d e - \* f g / +) ^ (h j -))

**a b c - d e - \* f g / + h j - ^**

**Infijo a Prefijo**

1. (x + y) \* (z ^ (s + t) + u) - v

(x + y) \* (((z ^ (s + t)) + u) - v

(+ x y) \* (((^ z + s t) + u) - v

(+ x y) \* (+ ^ z + s t u) - v

(+ x y) \* (- + ^ z + s t u v)

**\* + x y - + ^ z + + s t u v**

b. a + ((x + y) \* (w + v)) ^ (i + j)

(a + ((x + y) \* (w + v))) ^ (i + j)

a + ((+ x y) \* (+ w v)) ^ (+ i j)

(a + (\* + x y + w v)) ^ (+ i j)

((+ a \* + x y + w v) ^ (+ i j))

**^ + a \* + x y + w v + i j**

1. a + b) \* (c + d)

(+ a b) \* (+ c d)

**\* + a b + c d**

d. ((a + b) - c)

((+ a b) - c)

**- + a b c**

e. (a + b) \* (c - d) \* e \* f

((+ a b) \* (- c d)) \* e \* f

((\* + a b - c d) \* e) \* f

(\* \* + a b - c d e) \* f

**\* \* \* + a b - c d e f**

f. (a + b) \* (c / (d - e) + f) - g

(+ a b) \* ((c / (- d e)) + f) - g

(+ a b) \* ((/ c - d e) + f) - g

((+ a b) \* (+ / c - d e f)) - g

((\* + a b + / c - d e f) - g)

**- \* + a b + / c - d e f g**

g. ((a ^ b) ^ c)

((^ a b) ^ c)

**^ ^ a b c**

h. a ^ b \* c - d + e / f / (g + h)

(((^ a b) \* c) - d) + (e / (f / (g + h)))

((\* ^ a b c) - d) + (e / (f / (g + h)))

(- \* ^ a b c d) + (e / (f / (g + h)))

(- \* ^ a b c d) + (e / (f / (+ g h)))

(- \* ^ a b c d) + (e / (/ f + g h))

(- \* ^ a b c d) + (/ / e f + g h)

**+ - \* ^ a b c d // e f + g h**

i. a + (((b - c) \* (d - e) + f) / g) ^ (h - j)

a + ((((- b c) \* (- d e)) + f) / g) ^ (- h j)

a + (((\* - b c - d e) + f) / g) ^ (- h j)

a + ((+ \* - b c - d e f) / g) ^ (- h j)

(a + (/ + \* - b c - d e f g)) ^ (- h j)

(+ a / + \* - b c - d e f g) ^ (- h j)

**^ + a / + \* - b c - d e f g - h j**

**Prefijo a Infijo**

1. + - ^ x y z \* i \* \* k l m

**((x ^ y) - z) + (i \* ((k \* l) \* m))**

1. + + l - \* ^ m n o / + p q \* r s t

**(l + ((m ^ n) \* o)) + ((p + q) - (r \* s) / t)**

**Posfijo a Infijo**

1. p q r s t - + ^ u v \* -

**(p \* (q ^ (r + (s + t)))) - (u \* v)**

1. l m - n + o p q - + ^

**((l - m) + n) ^ (o + (p - q))**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  void init\_pila();  void vacia\_pila();  void ins\_pila();  void retirar\_pm();  void menu();  int main(){  menu();  return 0;  }  void menu(){  int n;  printf("\*\*\*\*\*\*\*\*\*\*");  printf("\n\*\*\*\*\*\*\*\*\*\*");  printf("\n\*\*\*\*\*\*\*\*\*");  printf("\n\*\*\*\*\*\*\*\*\*\*");  printf("\nIngrese un numero: ");  scanf("%d",&n);  system("cls");  switch(n){  case 0:  exit(0);  break;  case 1:  printf("\*\*\*\*\*\*\*\*\*");  init\_pila();  break;  case 2:  printf("\*\*\*\*\*\*\*\*\*\*\*\*");  vacia\_pila();  break;  case 3:  printf("\*\*\*\*\*\*\*\*");  ins\_pila();  break;  case 4:  printf("\*\*\*\*\*\*\*\*\*\*");  retirar\_pm();  break;  default:  printf("\*\*\*\*\*\*\*\*");  }  }  void init\_pila(){  int i,j;  scanf("%d, %d",i,j);  printf("%d + %d = %d",i,j,i+j);  }  void vacia\_pila(){  }  void ins\_pila(){  }  void retirar\_pm(){  } |

1. ¿Qué es programación funcional?

Los lenguajes de programación funcionales, son aquellos lenguajes donde las variables no tienen estado no hay cambios en éstas a lo largo del tiempo y son inmutables no pueden cambiarse los valores a lo largo de la ejecución. Además, los programas se estructuran componiendo expresiones que se evalúan como funciones. Dentro de los lenguajes funcionales tenemos Lisp, Scheme, Clojure, Haskell, OCaml y Standard ML, entre otros. Estos lenguajes están diversidad de tipificación, donde se encuentran lenguajes dinámicos, estáticos y estáticos fuertes.

En los lenguajes funcionales las instrucciones cíclicas como for, while y do-while no existen. Todo se procesa usando recursividad y funciones de alto orden. Esto se debe a los fundamentos matemáticos de la mayoría de los lenguajes funcionales, principalmente con bases en el sistema formal diseñado por Alonzo Church para definir cómputos y estudiar las aplicaciones de las funciones llamado Cálculo Lambda. En este sistema formal se puede expresar recursividad en las funciones, y entre otras cosas interesantes, se pueden expresar combinadores funciones sin variables libres el Combinador de Punto Fijo o Y Combinator, que expresa recursividad sin hacer llamadas recursivas. En el Cálculo Lambda existen tres transformaciones esenciales, la conversión α, la reducción β y la conversión η. En la conversión α se sustituyen los nombres de las variables para dar más claridad a la aplicación de las funciones, por ejemplo, evitando duplicados en sus nombres. En la reducción β se traza el llamado de las funciones sustituyendo las funciones por sus expresiones resultantes. Finalmente, en las conversiones η se busca las equivalencias de trazado de funciones sustituyéndolas por sus equivalentes. Estas transformaciones también pueden ser aplicadas en los lenguajes funcionales o en su mayoría dando lugar lenguajes que cuentan con una gran expresividad y consistencia. (Diaz, 2012)

1. ¿Qué es DrRackert?

Racket es un lenguaje de programación de amplio espectro de la familia de Lisp y Scheme. Es multiparadigma, así como de propósito general. Uno de sus principales objetivos tras su diseño es posibilitar la creación de nuevos lenguajes o dialectos. El lenguaje es usado en una variedad de entornos tales como scripting, enseñanza en ingeniería informática o la investigación.

La plataforma nos ofrece la herramienta DrRacket, un entorno de desarrollo integrado programado en Racket, que nos facilitará la tarea de programar en Racket. También nos ofrece raco, una herramienta para la línea de comandos que nos permitirá instalar paquetes o compilar librerías. (Wikipedia, s.f.)

1. Como implementar y funciona una pila.

Una Pila es una lista de elementos caracterizada porque las operaciones de Inserción y Eliminación de elementos se realizan solamente en un extremo de la estructura. En consecuencia, los elementos de una Pila se eliminan en el orden inverso al que se insertaron. Debido a esta característica, se le conoce como estructura LIFO (Last In First Out). (Codebotic, 2015)

|  |
| --- |
| /\*  Autor: Joel Fernandez  Ejercicio: Implementacion de una Pila usando Clases  IDE: Codeblocks  Web: http://codebotic.blogspot.com  \*/  #include<iostream>  #include<cstdlib>  using namespace std;  struct nodo{  int nro;  struct nodo \*sgte;  };  typedef struct nodo \*Puntero;  class Pila{  public:  Pila(void);  void Apilar(int );  int Desapilar(void );  void Cima(void);  bool PilaVacia(void);  void MostrarPila(void);  void DestruirPila(void);  private:  Puntero cima;  };  Pila::Pila(void){  cima=NULL;  }  bool Pila::PilaVacia(void){  if(cima==NULL)  return true;  else  return false;  }  void Pila::Apilar(int x){  Puntero p\_aux;  p\_aux=new(struct nodo);  p\_aux->nro=x;  p\_aux->sgte=cima;  cima=p\_aux;  }  void Pila::Cima(){  int x;  if(cima==NULL)  cout<<"\n\n\tPila Vacia...!";  else {  x=cima->nro;  cout<<"\n\tLa Cima es :"<<x<<endl;  }  }  int Pila::Desapilar(void){  int x;  Puntero p\_aux;  if(cima==NULL)  cout<<"\n\n\tPila Vacia...!!";  else{  p\_aux=cima;  x=p\_aux->nro;  cima=cima->sgte;  delete(p\_aux);  }  return x;  }  void Pila::MostrarPila(void){  Puntero p\_aux;  p\_aux=cima;  while(p\_aux!=NULL){  cout<<"\t "<<p\_aux->nro<<endl;  p\_aux=p\_aux->sgte;  }  }  void Pila::DestruirPila(void){  Puntero p\_aux;  while(cima!=NULL){  p\_aux=cima;  cima=cima->sgte;  delete(p\_aux);  }  }  void menu(void)  {  cout<<"\t -------------------------------------------\n";  cout<<"\t| IMPLEMENTACION DE UNA PILA |\n";  cout<<"\t|-------------------------------------------|\n";  cout<<" \t| |"<<endl;  cout<<" \t| 1. APILAR |"<<endl;  cout<<" \t| 2. DESAPILAR |"<<endl;  cout<<" \t| 3. MOSTRAR PILA |"<<endl;  cout<<" \t| 4. DESTRUIR PILA |"<<endl;  cout<<" \t| 5. MOSTRAR CIMA |"<<endl;  cout<<" \t| 6. SALIR |"<<endl;  cout<<" \t| |"<<endl;  cout<<"\t -------------------------------------------\n";  cout<<"\t Ingrese opcion: ";  }  int main(void ){  system("color 0a");  Pila pila;  int x;  int op;  do  {  menu(); cin>> op;  switch(op)  {  case 1: cout<< "\n\t INGRESE NUMERO A APILAR: "; cin>> x;  pila.Apilar(x);  cout<<"\n\n\t\tNumero " << x << " apilado...\n\n";  break;  case 2:  if(pila.PilaVacia()==true)  cout<<"\n\n\tPila Vacia....";  else{  x = pila.Desapilar( );  cout<<"\n\n\tNumero "<<x<<" desapilado\n";  }  break;  case 3:  cout << "\n\n\t MOSTRANDO PILA\n\n";  if(pila.PilaVacia()!=true)  pila.MostrarPila( );  else  cout<<"\n\n\tPila vacia..!"<<endl;  break;  case 4:  pila.DestruirPila( );  cout<<"\n\n\t\tPila eliminada...\n\n";  break;  case 5:  pila.Cima();  break;  }  cout<<endl<<endl;  system("pause"); system("cls");  }while(op!=6);  return 0;  } |

|  |
| --- |
| #include <iostream>  #include <cstring>  #include <stack>  #include <algorithm>  #define flag '#'    using namespace std;    bool isOperator(char c)  {      if(c=='+' || c=='-' || c=='\*' || c=='/' || c=='^' || c=='$')      return true;      else      return false;  }    int main()  {      stack<char> stk;      char postfix[30], prefix[30];      int j=0,len;      cout<<"Input a postfix expression: ";      cin>>postfix;      len = strlen(postfix);      for(int i=len-1;i>=0;i--)      {          if(isOperator(postfix[i]))          stk.push(postfix[i]);          else          {              prefix[j++] = postfix[i];              while(!stk.empty() && stk.top()==flag)              {                  stk.pop();                  prefix[j++] = stk.top();                  stk.pop();              }              stk.push(flag);            }      }      prefix[j] = 0;      reverse(prefix, prefix + len);      cout<<"The prefix expression is: "<<prefix;      return 0;  } |
| #include <iostream>  #include <string>  #include <fstream>    using namespace std;    bool isOperator(char symbol)          //definition of isOperator  {      if( (symbol == '\*') || (symbol == '+')  || (symbol == '/')  || (symbol == '-' ) )                          return true;      else                          return false;  }    class Stack {     public:     int MaxStack;     int EmptyStack;     char\* items;     int top;         Stack(int);     ~Stack();     int TOP();     void push(char);     char pop();     int empty();     int full();    };  Stack::Stack(int size) {     MaxStack = size;     EmptyStack = -1;     top = EmptyStack;     items = new char[MaxStack];  }    Stack::~Stack() {delete[] items;}  int Stack::TOP(){  return items[top];  }  void Stack::push(char c) {     items[++top] = c;  }    char Stack::pop() {     return items[top--];  }    int Stack::full()  {     return top + 1 == MaxStack;  }    int Stack::empty()  {     return top == EmptyStack;  }        int main()  {    cout <<"WELCOME TO PROJECT 4 PROGRAM"<<endl;      // basic components    string line;    string outputLine;      //stacks      Stack stack(20);    char c;        ifstream myfile ("Prefix.in");    if (myfile.is\_open())    {      while ( myfile.good() )      {        getline(myfile,line);        cout << line << endl;        }    }      else cout << "Unable to open file";    //flags, operator method  while (line.length()!=0)    //while not end of blank line              {                     Stack flags(20);                     Stack operators(20);                for(int i=0; i < line.size(); i ++)  //reiterate through ‘line’                          {                            char symbol = line [i];                            if( isOperator(symbol))    //if the symbol is an operator                              {                                  operators.push(symbol);    //push operator on stack                                   flags.push(0);  //push associated flag on stack                                    }                          if((symbol != ' ') && !isOperator(symbol))     //then it’s a operand                              {                                outputLine += symbol;                //append                               while(flags.TOP())       //while top flag is ON on stack                                {                                  outputLine += operators.TOP() ;  //append the associated op                                  operators.pop();   //remove operator from stack                                  flags.pop();     //remove flag from stack                                }                                  flags.pop();                 //set next flag ON                                  flags.push(1);                              }                          }//end of for                            if(!operators.empty() || (!flags.empty()) )                          {                                      cout << "SOMETHING WENT WRONG. Prob incorrect input" << endl;                          }                          else                          {                                      cout << line << endl;                                      cout << outputLine << endl;                          }              } //end of while eof        ofstream outfile ("Postfix.out");    if (outfile.is\_open())    {      outfile << outputLine << endl;      outfile << line << endl;        myfile.close();    }    else cout << "Unable to open file";    myfile.close();    //outFile.close();    return 0;  } |