#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "stackFunctions.h"

int isOperand(char ch){

return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');

}

int prec(char ch){

switch (ch){

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

int infixToPostfix(char\* exp){

int i, k;

struct Stack\* stack = createStack(strlen(exp));

if(!stack)

return -1 ;

for (i = 0, k = -1; exp[i]; ++i){

if (isOperand(exp[i])) exp[++k] = exp[i];

else if (exp[i] == '(') push(stack, exp[i]);

else if (exp[i] == ')'){

while (!isEmpty(stack) && peek(stack) != '(')

exp[++k] = pop(stack);

if (!isEmpty(stack) && peek(stack) != '(')

return -1;

else

pop(stack);

}

else {

while (!isEmpty(stack) && prec(exp[i]) <= prec(peek(stack)))

exp[++k] = pop(stack);

push(stack, exp[i]);

}

}

while (!isEmpty(stack)) exp[++k] = pop(stack);

exp[++k] = '\0';

printf( "%s", exp );

printf("\n");

}

int main(){

char exp[] = "a+b\*(c/d-e)\*(f-g/h)-i";

// int t,n;

// printf("Enter the number of inputs\n");

// scanf("%d",&t);

// printf("Enter the expression\n");

// while(t--){

// scanf("%s",exp);

infixToPostfix(exp);

// }

return 0;

}

OUTPUT:

$ gcc infixToPostfix.c

$ ./a.out

abcd/e-\*fgh/-\*+i-

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

#include "stackFunctions.h"

int evaluatePostfix(char\* exp){

struct Stack\* stack = createStack(strlen(exp));

int i;

if (!stack) return -1;

for (i = 0; exp[i]; ++i){

if (isdigit(exp[i])) push(stack, exp[i] - '0');

else{

int val1 = pop(stack);

int val2 = pop(stack);

switch (exp[i]){

case '+': push(stack, val2 + val1); break;

case '-': push(stack, val2 - val1); break;

case '\*': push(stack, val2 \* val1); break;

case '/': push(stack, val2/val1); break;

}

}

}

return pop(stack);

}

int main(){

char exp[] = "462/-6+";

printf ("Value of %s is %d\n", exp, evaluatePostfix(exp));

return 0;

}

OUTPUT:

$ gcc postEval.c

$ ./a.out

Value of 462/-6+ is 7

#include <stdio.h>

#include <stdlib.h>

struct Node{

int data;

struct Node\* left;

struct Node\* right;

};

typedef struct Node\* node;

node root;

node createNewNode(int x){

node current = (node) malloc(sizeof(node));

current->data =x;

current->left = current->right = NULL;

return current;

}

node plant(node root, int x){

if(root == NULL) root = createNewNode(x);

else if(x <= root->data) root->left = plant(root->left, x);

else if(x > root->data) root->right = plant(root->right, x);

return root;

}

node minOf(node root){

while(root->left != NULL) root = root->left;

return root;

}

node deleteIt(node root, int data){

if(root == NULL) return root;

else if(data < root->data) root->left = deleteIt(root->left,data);

else if(data > root->data) root->right = deleteIt(root->right,data);

else{

if(root->right == NULL && root->left == NULL){

free(root);

root = NULL;

}

else if(root->left == NULL){

node temp = root;

root = root->right;

free(temp);

}

else if(root->right == NULL){

node temp = root;

root = root->left;

free(temp);

}

else{

node temp = minOf(root->right);

root->data = temp->data;

root->right = deleteIt(root->right,temp->data);

}

}

return root;

}

int search(node root, int x){

if(root == NULL) return 0;

else if(x == root->data) return 1;

else if(x <= root->data) return search(root->left, x);

else if(x > root->data) return search(root->right, x);

else return 0;

}

void postOrderTraversal(node root){

if (root == NULL) return;

postOrderTraversal(root->left);

postOrderTraversal(root->right);

printf("%d ", root->data);

}

void inOrderTraversal(node root){

if (root == NULL) return;

inOrderTraversal(root->left);

printf("%d ", root->data);

inOrderTraversal(root->right);

}

void preOrderTraversal(node root){

if (root == NULL) return;

printf("%d ", root->data);

preOrderTraversal(root->left);

preOrderTraversal(root->right);

}

int main(){

root = NULL;

int k,t,n,ch;

printf("Enter the number of inputs\n");

scanf("%d",&n);

printf("Enter the values\n");

while(n--){

scanf("%d",&t);

root = plant(root, t);

}

printf("you wanna search?\n if yes press 1 & the number to search\n");

scanf("%d",&k);

scanf("%d",&ch);

if(k==1){

if(search(root,ch)) printf("found\n");

else printf("not found");

}

preOrderTraversal(root);

return 0;

}

OUTPUT

$ gcc tree.c

$ ./a.out

Enter the number of inputs

4

Enter the values

5

6

1

2

you wanna search?\n if yes press 1 & the number to search

1

6

found

5 1 2 6

#include <stdio.h>

#include <stdlib.h>

void swap(int \*i, int \*j){

int\* temp;

temp = \*i;

\*i = \*j;

\*j = temp;

return;

}

void heapify(int arr[], int n, int i){

int high = i;

int l = 2\*i + 1;

int r = 2\*i + 2;

if(l < n && arr[l] > arr[high]) high = l;

if(r < n && arr[r] > arr[high]) high = r;

if(high != i){

int \*l = arr[i];

int \*r = arr[high];

swap(l, r);

heapify(arr,n,i);

}

}

void heapSort(int arr, int n){

for(int i = n-1/2; i>=0; i--) heapify(arr,n,i);

for(int i=n-1; i>=0; i--){

int \*l = arr[0];

int \*r = arr[i];

swap(l, r);

heapify(arr,n,0);

}

}

void print(int arr[], int n)

{

for (int i=0; i<n; ++i)

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

printf("\n");

}

int main(){

int arr[] = {2, 0, 31, 55, 23, 71};

int n = sizeof(arr)/sizeof(arr[0]);

heapSort(arr, n);

printf("After sorting...\n");

print(arr, n);

return 0;

}

OUTPUT:

$ gcc heap.c

$ ./a.out

After sorting…

0 2 23 32 55 71

struct Stack

{

int top;

unsigned capacity;

int\* array;

};

struct Stack\* createStack( unsigned capacity )

{

struct Stack\* stack = (struct Stack\*) malloc(sizeof(struct Stack));

if (!stack)

return NULL;

stack->top = -1;

stack->capacity = capacity;

stack->array = (int\*) malloc(stack->capacity \* sizeof(int));

if (!stack->array)

return NULL;

return stack;

}

int isEmpty(struct Stack\* stack)

{

return stack->top == -1 ;

}

char peek(struct Stack\* stack)

{

return stack->array[stack->top];

}

char pop(struct Stack\* stack)

{

if (!isEmpty(stack))

return stack->array[stack->top--] ;

return '&';

}

void push(struct Stack\* stack, char op)

{

stack->array[++stack->top] = op;

}