Binary Search

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <math.h>

void modifiedBubbleSort(int \*arr, int n)

{

  int i, j;

  int swapped = 1;

  for (i = 0; i < n - 1 && swapped == 1; i++)

  {

    swapped = 0;

    for (j = 0; j < n - i - 1; j++)

    {

      if (arr[j] > arr[j + 1])

      {

        int temp = arr[j];

        arr[j] = arr[j + 1];

        arr[j + 1] = temp;

        swapped = 1;

      }

    }

  }

}

int BinarySearch(int \*a, int n, int data)

{

  int l = 0, r = n - 1;

  while (l < r)

  {

    int mid = (l + r) / 2;

    if (data == a[mid])

    {

      return mid;

    }

    else if (data < a[mid])

    {

      r = mid - 1;

    }

    else

    {

      l = mid + 1;

    }

  }

  return -1;

}

int binarySearchRecursive(int \*arr, int left, int right, int target)

{

  if (right >= left)

  {

    int mid = left + (right - left) / 2;

    if (arr[mid] == target)

      return mid;

    if (arr[mid] > target)

      return binarySearchRecursive(arr, left, mid - 1, target);

    return binarySearchRecursive(arr, mid + 1, right, target);

  }

  return -1;

}

int main()

{

  int n, i, data;

  printf("Enter the number of elements in the array: ");

  scanf("%d", &n);

  int \*arr = (int \*)malloc(n \* sizeof(int));

  printf("Enter the values in the array:\n");

  for (i = 0; i < n; i++)

  {

    scanf("%d", &arr[i]);

  }

  modifiedBubbleSort(arr, n);

  printf("Enter the number you want to search: ");

  scanf("%d", &data);

  int search = binarySearchRecursive(arr, 0, n - 1, data);

  if (search == -1)

  {

    printf("Not present\n");

  }

  else

  {

    printf("The number is present at position %d\n", search + 1);

  }

  return 0;

}

Infix to Postfix

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define max 100

char infix[max];

char postfix[max];

typedef struct node

{

  int data;

  struct node \*next;

} stack;

stack \*top = NULL;

void push(stack \*st, int x)

{

  stack \*nn = (stack \*)malloc(sizeof(stack));

  nn->data = x;

  if (top == NULL)

  {

    top = nn;

    nn->next = NULL;

    return;

  }

  nn->next = top;

  top = nn;

}

int isEmpty(stack \*st)

{

  if (top == NULL)

  {

    return 1;

  }

  return 0;

}

char pop(stack \*st)

{

  if (top == NULL)

  {

    printf("Stack Underflow\n");

    return -1;

  }

  stack \*temp = top;

  int n = temp->data;

  top = top->next;

  free(temp);

  return n;

}

char peek(stack \*st)

{

  if (top == NULL)

  {

    return -1;

  }

  return top->data;

}

int precedence(char c)

{

*// Determine the precedence of an operator*

  switch (c)

  {

  case '\*': *// Multiplication*

  case '/': *// Division*

  case '%': *// Remainder*

    return 2;

  case '+': *// Addition*

  case '-': *// Subtraction*

    return 1;

  default: *// No precedence*

    return 0;

  }

}

*// Function to check if a character is a space or a tab*

int space(char c)

{

*// Check if the character is a space or a tab*

  if (c == ' ' || c == '\t')

  {

    return 1; *// Return 1 if it is a space or a tab*

  }

  else

  {

    return 0; *// Return 0 if it is not a space or a tab*

  }

}

void print()

{

  int i = 0; *// Initialize index to 0*

  printf("Equivalent postfix: "); *// Print header*

  while (postfix[i]) *// Loop through postfix array*

  {

    printf("%c", postfix[i]); *// Print current character*

    i++; *// Increment index*

  }

  printf("\n"); *// Print newline*

}

void infixToPostfix(stack \*ptr)

{

  int i, j = 0;

  char n, s;

*// Loop through each character in the infix expression*

  for (i = 0; i < strlen(infix); i++)

  {

    s = infix[i];

*// If the character is not a space*

    if (!space(s))

    {

      switch (s)

      {

      case '(':

        push(ptr, s); *// Push the left parenthesis to the stack*

        break;

      case ')':

        while ((n = pop(ptr)) != '(') *// Pop and append operators to the postfix string until a left parenthesis is encountered*

        {

          postfix[j++] = n;

        }

        break;

      case '+':

      case '-':

      case '\*':

      case '/':

      case '%':

        while (!isEmpty(ptr) && precedence(peek(ptr)) >=

                                    precedence(s)) *// Pop and append operators to the postfix string until the stack is empty or the top operator has lower precedence*

        {

          postfix[j++] = pop(ptr);

        }

        push(ptr, s); *// Push the current operator to the stack*

        break;

      default:

        postfix[j++] = s; *// Append the operand to the postfix string*

      }

    }

  }

*// Pop and append the remaining operators in the stack to the postfixstring*

  while (!isEmpty(ptr))

  {

    postfix[j++] = pop(ptr);

  }

  postfix[j] = '\0'; *// Terminate the postfix string*

}

int evaluatePostfix(stack \*ptr)

{

  int i, a, b;

*// Loop through each character in the postfix string*

  for (i = 0; i < strlen(postfix); i++)

  {

*// If the character is a digit, push it onto the stack*

    if (postfix[i] >= '0' && postfix[i] <= '9')

    {

      push(ptr, postfix[i] - '0');

    }

*// If the character is an operator, pop two operands from the stack*

    else

    {

      a = pop(ptr);

      b = pop(ptr);

*// Perform the operation and push the result back onto the stack*

      switch (postfix[i])

      {

      case '+':

        push(ptr, (b + a));

        break;

      case '-':

        push(ptr, (b - a));

        break;

      case '\*':

        push(ptr, (b \* a));

        break;

      case '/':

        push(ptr, (b / a));

        break;

      case '%':

        push(ptr, (b % a));

      }

    }

  }

*// Return the final result, which is the only element left on the stack*

  int x = pop(ptr);

  return x;

}

int main()

{

  stack \*ptr = (stack \*)malloc(sizeof(stack));

  printf("Infix : ");

  gets(infix);

  infixToPostfix(ptr);

  print();

  char ans = evaluatePostfix(ptr);

  printf("\n%d\n", ans);

  return 0;

}