**КР № 4**

Написать программу формирования ОПЗ и расчета полученного выражения. Разработать удобный интерфейс ввода исходных данных и вывода результатов. Работу программы проверить на конкретном примере.





* 1. **Создание консольного приложения**

Текст программы:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <math.h>

typedef double tData;

typedef char tPriority;

typedef struct sNode {

tData value;

tPriority Priority;

struct sNode\* next;

} tNode;

typedef struct {

tNode\* p\_top;

size\_t m\_size;

} tStack;

void Push(tStack\* s, tData value);

tData Pop(tStack\* s);

tStack\* Stack\_Create();

tPriority GetPriority(tData ch);

double CalculateExpression(char\* Str);

tStack\* Stack\_Create() {

tStack\* new\_stack = (tStack\*)malloc(sizeof(tStack));

new\_stack->m\_size = 0;

new\_stack->p\_top = NULL;

return new\_stack;

}

// Добавление элемента в стек

void Push(tStack\* s, tData value) {

tNode\* p = (tNode\*)malloc(sizeof(tNode));

p->value = value;

p->Priority = GetPriority(value);

p->next = s->p\_top;

s->p\_top = p;

s->m\_size++;

}

// Получение элемента со стека

tData Pop(tStack\* s) {

tNode\* tmp = s->p\_top;

tData tmp\_value = tmp->value;

s->p\_top = s->p\_top->next;

s->m\_size--;

free(tmp);

return tmp\_value;

}

// Получить приоритет символа

tPriority GetPriority(tData ch) {

switch ((char)ch)

{

case '(': return 1;

case '+': case '-': return 2;

case '/': case '\*': return 3;

case '^': return 4;

}

}

// Добавить символ в выходной строке

void AddToFinalStr(char\* Dest, char Source) {

int cur\_len = strlen(Dest);

\*(Dest + cur\_len) = Source;

\*(Dest + cur\_len + 1) = '\0';

}

// Присоединение элементов со стека к выходной строке

void ConcatStack(tStack\* s, char Symbol, char\* Dest) {

tPriority prior = GetPriority(Symbol);

while (s->m\_size != 0 && prior <= s->p\_top->Priority) {

AddToFinalStr(Dest, Pop(s));

}

Push(s, Symbol);

}

// Присоединение элементов со стека к выходной строке до скобки

void ConcatStackToBracket(tStack\* s, char\* Dest) {

while (s->p\_top->value != '(')

AddToFinalStr(Dest, Pop(s));

Pop(s);

}

// Присоединение оставшихся элеметов стека

void ConcatRemaining(tStack\* s, char\* Dest) {

while (s->m\_size != 0)

AddToFinalStr(Dest, Pop(s));

}

// Возвращает выражение в обратной польской записи

char\* GetRPN(char\* Expression) {

tStack\* myStack = Stack\_Create();

char CurSym;

// Создание выходной строки

char\* myExprRPN = calloc(strlen(Expression) + 1, sizeof(char));

// Пока не конец строки

while (\*Expression) {

CurSym = \*Expression++;

if (CurSym >= 'a' && CurSym <= 'z' || CurSym >= '0' && CurSym <= '9')

AddToFinalStr(myExprRPN, CurSym);

else if (CurSym == '(')

Push(myStack, CurSym);

else if (CurSym == ')')

ConcatStackToBracket(myStack, myExprRPN);

else

ConcatStack(myStack, CurSym, myExprRPN);

}

// Присоединить оставшиеся элементы со стека

ConcatRemaining(myStack, myExprRPN);

free(myStack);

return myExprRPN;

}

// Получение значения операнда

double GetValue(char Sym) {

switch (Sym)

{

case 'a': return 3.1;

case 'b': return 5.4;

case 'c': return 0.2;

case 'd': return 9.6;

case 'e': return 7.8;

}

}

// Вычисление значения выражения \*в ОПН\*

double CalculateExpression(char\* Str) {

tStack\* myStack = Stack\_Create();

char CurSym;

double tmpOpnd1, tmpOpnd2, Result;

while (\*Str) {

CurSym = \*Str++;

if (CurSym >= 'a' && CurSym <= 'z') {

Push(myStack, GetValue(CurSym));

}

else if (CurSym >= '0' && CurSym <= '9') {

// На стек кладется цифра, а не код символа

Push(myStack, CurSym - 0x30);

}

// Знаки операций

else {

tmpOpnd1 = Pop(myStack);

tmpOpnd2 = Pop(myStack);

switch (CurSym) {

case '+':

Result = tmpOpnd2 + tmpOpnd1; break;

case '-':

Result = tmpOpnd2 - tmpOpnd1; break;

case '\*':

Result = tmpOpnd2 \* tmpOpnd1; break;

case '/':

Result = tmpOpnd2 / tmpOpnd1; break;

case '^':

Result = pow(tmpOpnd2, tmpOpnd1); break;

}

Push(myStack, Result);

}

}

return Pop(myStack);

}

int main()

{

char\* Str = malloc(50 \* sizeof(char));

char\* Res;

double Result;

puts("Input infix expression");

gets(Str);

Res = GetRPN(Str);

printf("RPN: %s\n", Res);

Result = CalculateExpression(Res);

printf("Result = %lf\n", Result);

puts("Press any key to continue...");

getch();

free(Str);

free(Res);

return 0;

}

Результаты выполнения:

