Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Лабораторная работа № 5

по дисциплине ОАиП

Вариант 8

Выполнил студент гр. 150502: Альхимович Н.Г.

Проверил:         Дулько П.А.

Минск 2022

Задание:

С помощью бинарного дерева вычислить арифметическое выражение. В арифметическом выражении используются операции +, -, \*, / и (,).

Листинг кода:

#include <stdio.h>

#include <stdlib.h>

struct tree

{

char symbol;

struct tree \*left, \*right;

};

struct stack

{

struct tree \*node;

struct stack \*next;

};

struct tree \*buildtree(struct tree \*, char \*, int \*); //прототип функции построения дерева по выражению

int evaluate(struct tree \*); //прототип функции вычисления выражения через бинарное дерево

void push(struct stack \*\*, struct tree \*); //прототип функции, заносящей узел дерева в стек

struct tree \*pop(struct stack \*\*); //прототип функции, удаляющей текущую вершину стека

int main()

{

struct tree \*root = NULL;

char \*str;

int result, flag = 0;

printf("Введите арифметическое выражение, заключив его в круглые скобки: ");

if((str = (char \*) calloc(50, sizeof(char))) == NULL)

{

printf("\nПамять не выделена");

return -1;

}

fgets(str, 50, stdin);

while(!(\*str))

{

printf("\nАрифметическое выражение не введено. Повторите попытку");

fflush(stdin);

fgets(str, 50, stdin);

}

if((root = (struct tree \*) malloc(sizeof(struct tree))) == NULL)

{

printf("\nПамять не выделена");

exit(-1);

}

root->left = root->right = NULL;

if(\*(str+0) != '(') //если выражение не заключено в скобки

{

printf("\nПроверьте правильность введенного выражения\n\n");

return 0;

}

else buildtree(root, str, &flag);

if(flag) printf("\nПроверьте правильность введенного выражения\n\n"); //если в выражении найдены посторонние символы

else

{

result = evaluate(root);

printf("\nРезультат вычислений: %d\n\n", result);

}

free(str);

return 0;

}

struct tree \*buildtree(struct tree \*root, char \*str, int \*flag)

{

int i = 0;

struct tree \*current, \*temp;

struct stack \*tos = NULL;

if(!root)

{

printf("\nДерево пустое\n");

exit(-1);

}

else

{

current = root; //рабочий указатель указывает на корень дерева

while(\*(str+i) != '\n') //пока не достигнут конец выражения

{

switch(\*(str+i)) //анализ очередного символа

{

case '(':

if((current->left = (struct tree \*) calloc(1, sizeof(struct tree))) == NULL) //добавление левого потомка

{

printf("\nПамять не выделена");

exit(-1);

}

push(&tos, current); //запоминание родительского узла

current = current->left; //перемещение рабочего указателя на левого потомка

break;

case '+': case '-': case '\*': case '/':

current->symbol = \*(str+i); //оператор заносится в текущий узел

if((current->right = (struct tree \*) calloc(1, sizeof(struct tree))) == NULL) //добавление правого потомка

{

printf("\nПамять не выделена");

exit(-1);

}

push(&tos, current);

current = current->right; //перемещение рабочего указателя на правого потомка

break;

case '0': case '1': case '2': case '3': case '4': case '5': case '6': case '7': case '8': case '9':

current->symbol = \*(str+i); //операнд заносится в текущий узел

temp = pop(&tos); //нахождение родительского узла

current = temp; //перемещение к нему

break;

case ')': current = pop(&tos); break; //перемещение к родительскому узлу

default:

\*flag = 1; //в случае, если обнаружен посторонний символ

break;

}

i++;

}

}

return root;

}

void push(struct stack \*\*tos, struct tree \*current)

{

struct stack \*temp = \*tos;

if(!(\*tos = (struct stack \*) calloc(1, sizeof(struct stack)))) //указатель на новый элемент стека

{

printf("\nПамять не выделена");

return;

}

(\*tos)->node = current; //занесение текущего узла в стек

(\*tos)->next = temp; //новый элемент стека указывает на вершину стека

temp = \*tos; //новый элемент становится вершиной стека

}

struct tree \*pop(struct stack \*\*tos)

{

struct stack \*temp1, temp;

if(\*tos == NULL) return NULL; //если стек пуст, возвращается NULL

else

{

temp1 = \*tos; //запоминается указатель на вершину стека

temp = \*\*tos; // запоминается содержимое вершины стека

\*tos = (\*tos)->next; //рабочий указатель передвигается "вниз" по стеку

free(temp1); //удаляется элемент с вершины стека

return temp.node; //возвращается нужный родительский узел

}

}

int evaluate(struct tree \*current)

{

struct tree \*lchild, \*rchild;

char operator;

lchild = current->left;

rchild = current->right;

if((lchild != NULL) && (rchild != NULL)) //если текущий узел - оператор (не является листом)

{

operator = current->symbol;

switch(operator)

{

case '+': return evaluate(lchild) + evaluate(rchild);

case '-': return evaluate(lchild) - evaluate(rchild);

case '\*': return evaluate(lchild) \* evaluate(rchild);

case '/': return evaluate(lchild) / evaluate(rchild);

default:

printf("\nПроизошла ошибка\n");

return -1;

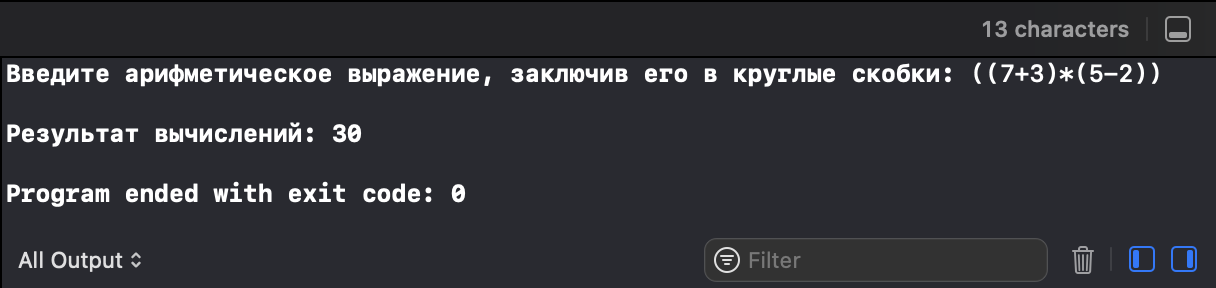
}

}

else return (current->symbol) - '0'; //если текущий узел - лист, функция возвращает его значение

}

Примеры работы программы:



В случае, если в выражении присутствуют «посторонние символы»:

