Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **«Информатики и систем управления»**
КАФЕДРА **«»**

РАСЧЕТНО–ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ НА ТЕМУ:

«Разработка программы для вычисления математических выражений»

Студент *Т.Я.Филькин*

(Группа) (Подпись, дата) (И.О. Фамилия)

Руководитель курсовой работы

(Подпись, дата) (И.О. Фамилия)

2023 г.

Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой _____
(Индекс)

_____
_____
(И.О. Фамилия)

« ____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине _____

Студент группы _____-_____

_____

Информатика

Филькин Тимофей Ярославович
(Фамилия, имя, отчество)

Тема курсовой работы *Разработка программы* для вычисления математических выражений

Направленность КР (учебная, исследовательская, практическая, производственная, др.)
~~учебная~~ Источник тематики (кафедра, предприятие, НИР) *кафедра*

График выполнения КР: 25% к 4 нед., 50% к 9 нед., 75% к 12 нед., 100% к 15 нед.

Задание: программа разрабатывается на языке программирования Си и должна выполнять следующие функции: программа разрабатывается на языке программирования Си и должна реализовать функцию вычисления произвольного математического выражения, ~~введенного пользователем, с учетом приоритета операций.~~

Оформление курсовой работы:

Расчетно-пояснительная записка на 21 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
текст программы на языке программирования Си, слайды презентации, UML-диаграммы

Дата выдачи задания «_17_» _февраля_ 20_23_ г.

Руководитель курсовой работы

	(Подпись, дата)	(И.О. Фамилия)
Студент		*Т.Я.Филькин*
	(Подпись, дата)	(И.О. Фамилия)

ОГЛАВЛЕНИЕ

1 ВВЕДЕНИЕ

Целью выполнения курсовой работы является разработка программы для вычисления математических выражений, которая может быть использована в различных проектах, требующих наличие вычисления выражений, записанных в строку с произвольной длинной. Программа позволяет производить следующие вычисления: сложение, вычитание, умножение, деление, возведение в целую степень. Программа принимает на вход последовательность целых чисел и символы действий (операторы). Результатом работы программы является целое число.

Данная программа позволяет пользователю получить результат вычислений, избегая повторного ввода каждого действия отдельно, уменьшая при этом время выполнения любой расчетной задачи. Как следствие, пользователь может подключить данный калькулятор в другие проекты.

Одним из немаловажных моментов является возможность удобного редактирования и дополнения нужным функционалом для конкретных задач.

При использовании калькулятора уходит необходимость отдельно автоматизировать расчеты, в связи с обособленностью выполнения логики программы.

Таким образом, данная программа, реализованная на языке С, является удобным вариантом решения объемных вычислений.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Математическая основа

Числовое выражение - это набор чисел, знаков математических операций, скобок. Приоритет действий позволяет установить порядок вычислений выражения.

Существует следующая очередность операций:

1. Возведение в степень

2. Скобки
3. Умножение
4. Деление
5. Сложение
6. Вычитание

Операции умножения и деления имеют между собой один и тот же приоритет также, как и операции сложения и вычитания. В случае, когда в выражении все операции имеют одинаковый вес, вычисление происходит слева направо по очередности.

В данной программе используется хранение данных в виде стека, который работает по принципу последний вошел, первый вышел.

Указатель – переменная, хранящая в себе адрес области памяти.

Структура – область памяти, содержащая в себе данные разных типов. Для обращения к элементу структуры по указателю, используется символ “→”.

Динамический массив – множество элементов одного типа данных, располагающихся в памяти последовательно, размер которого задается произвольным числом. Для работы с таким массивом требуются функции malloc и в случае двумерного массива – realloc, которые выделяют память. Функция free освобождает выделенную ранее область памяти.

Для удобной работы со строками используется функция memmove, которая копирует заданное количество символов из одного массива в другой.

Sprintf – функция, позволяющая направить поток вывода в отдельный массив, который принимается в качестве входного значения.

Strncat – функция, которая добавляет в первый передаваемый аргумент n байт из второго аргумента.

Strlen возвращает длину строки.

Strcpy – функция, которая копируют n байт данные из строки второго аргумента в строку первого.

3 ПРАКТИЧЕСКАЯ ЧАСТЬ

Для хранения данных используется структура Stack, которая содержит в себе размер, вместимость и сами элементы, такое решение дает возможность использовать все свойства стека. Elements представлены двойными указателями, которые по своей сути являются массивом указателей. Использование capacity необходимо чтобы разгрузить операцию realloc, которая тратит много ресурса при ее вызове, она позволяет задать число элементов, которые будут добавлены без изменения размера структуры, тем самым уменьшить количество использований этой функции.

Центром программы является алгоритм сортировочной станции, придуманный Дейкстрой, позволяющий реализовывать запись математического выражения в префиксной нотации. Префиксная нотация позволяет программе выполнять вычисления выражений с большим количеством различных математических операций и реализовать присвоение приоритета операции.

Для корректной работы программы все вводимые элементы представляются в виде строк (массива символов) и к каждому числу добавляется пробел, который играет роль разделителя самого числа от знака операции. Это решение позволяет пользователю выполнять вычисления над многозначными числами в записи которых присутствует до 11 знаков.

Все обозначения математических действий определены как однозначные символы для облегчения работы по сортировке строки как последовательности чисел и знаков, что в дальнейшем будет удобным решением получения результата.

Чтобы пользователь получил конкретный результат вычислений в процессе работы программы, обратную запись строки циклом проходят, присваивая значение левой и правой операндам. Это сделано для возможности вычисления выражений типа “ (3 – 5) ” в результате которых

появляется минус перед результатом, в противном случае записывался модуль результата.

Операция возведения в степень не является ассоциативной, поэтому для ее обработки используется отдельная функция, которая указывает на очередность считывания символа действия. Скобки реализованы как дополнительные пробелы между операндами вне скобок и внутри. Это сделано для облегчения чтения и определения местоположения скобок в выражении.

Программа заканчивает считывание символов после встречи “\n”, также возможно указание длины вводимой строки.

4 ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был создана программа, способная выполнять вычисления математических выражений, содержащее произвольное количество действий, обладающих разными приоритетами.

Положительными особенностями данного кода программы являются его лаконичность, обособленность выполняемых функций, легкость чтения и понимания кода, возможностью простого добавления обработки требуемых математических операций. Отдельным моментом хочется отметить удобство считывания числа с заданным количеством символов и реализацию жесткого присвоения приоритета отдельным операциям, путем добавления в исходное выражение скобок. Программа может быть использована в более сложных проектах, где будет являться независимой частью.

Цель работы была достигнута, программа работает исправно.

5 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ ИНФОРМАЦИИ

5.1 https://www.8host.com/blog/poryadok-obrabotki-operacij-v-programmirovanii/?ysclid=lhxcqx973m366119981

5.2 https://ru.wikipedia.org/wiki/LIFO

5.3 https://prog-cpp.ru/c-string/?ysclid=lhxcu522jk90145699

5.4 https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C

5.5 https://cplusplus.com/reference/cstdio/sprintf

5.6 http://all-ht.ru/inf/prog/c/func/atoi.html?ysclid=lhxe3c0kqm651745112

5.7 strncat – функция языка Си. "Все о Hi-Tech" (all-ht.ru)

6 ПРИЛОЖЕНИЕ А. UML-ДИАГРАММЫ

Диаграмма действий

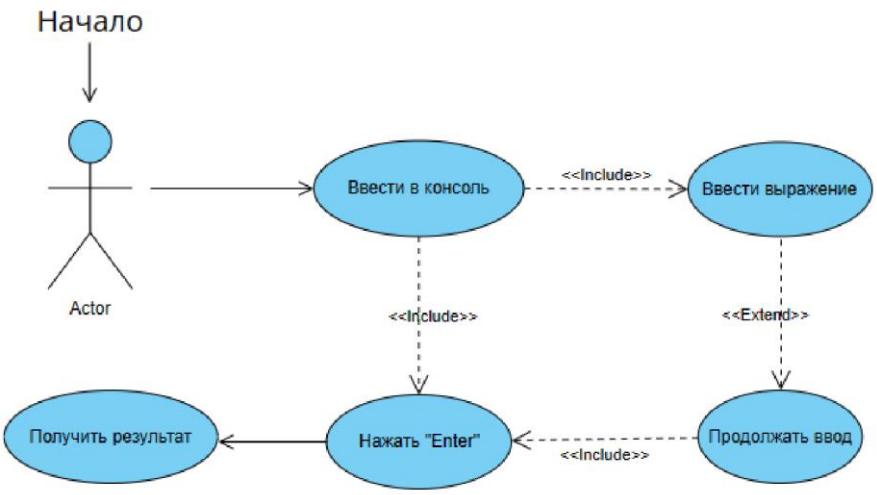


Диаграмма связей

7 ПРИЛОЖЕНИЕ Б. ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ

```
Enter the expression
 1000000+5000000+200/(6+3*5+(3+1)^(1*3))

RPN >> 1000000 5000000 +200  6 3 5 *+ 3 1 +  1 3 * ^+/+

 Result = 6000002
```

8 ПРИЛОЖЕНИЕ В. ТЕКСТ ПРОГРАММЫ

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <ctype.h>

typedef struct _Stack
{
   int size;    int capacity;
char** elements;
} Stack;

Stack* create_stack(void)
{
   Stack* stack = (Stack*)malloc(sizeof(Stack));

   stack->size = 0;    stack->capacity = 4;
   stack->elements = (char**)malloc(sizeof(char*) * stack->capacity);        return stack;
}
char* stack_pop(Stack* stack)
{
   char* r = stack->elements[0];
   memmove(stack->elements, stack->elements + 1, --stack->size * sizeof(char*));    return r;
}
char stack_pop_first_char(Stack* stack)
{
   char* r = stack->elements[0];
   memmove(stack->elements, stack->elements + 1, --stack->size * sizeof(char*));    const char rc = r[0];
   free(r);    return rc;
}
int stack_pop_number(Stack* stack)
{
   char* r = stack->elements[0];
   memmove(stack->elements, stack->elements + 1, --stack->size * sizeof(char*));    const int rn = atoi(r);    free(r);
return rn;
}
void stack_push(Stack* stack, char* value)
{
   if (stack->capacity < stack->size + 1)
   {
      stack->capacity *= 2;
      stack->elements = (char**)realloc(stack->elements, sizeof(char*) * stack->capacity);
   }
   memmove(stack->elements + 1, stack->elements, stack->size * sizeof(char*));
```

```
   stack->elements[0] = value;     stack->size++;

}
char* stack_peek(const Stack* stack, const int index)
{
   return stack->elements[index];
}
int op_precedence(const char op)
{    switch (op)    {
case '-':    case '+':
return 2;    case '*':
case '/':       return 3;
case '^':       return 4;
default:       return 1;
   }
}
int op_associativity(const char op)
{    switch (op)    {
case '^':
      return 1; // right     default:
return 0; // left
   }
}
int is_operator(const char c)
{    switch (c)    {
case '+':    case '-':
case '*':    case '/':
case '^':       return 1;
default:       return 0;
   }
}
char* str_new(const char c)
{
   char* result = (char*)malloc(sizeof(char) * 2);     result[0] = c;
result[1] = '\0';
       return result;
}
char* str_new_from_number(const int n)
{
   char* result = (char*)malloc(sizeof(char) * 11);     sprintf(result, "%d", n);

   return result;
}
char* str_append(char* str, const char c)
{
   const int len = strlen(str);
   char* result = (char*)malloc(sizeof(char) * (len + 2));     strncpy(result, str, len);
result[len] = c;     result[len + 1] = '\0';     free(str);     return result;
}
char* infix_to_rpn(const char* infix)
{
   char* result = (char*)malloc(sizeof(char) * 1024);     result[0] = '\0';

   Stack* operators = create_stack();
```

```
   const int len = strlen(infix);

   for (int i = 0; i < len; i++)
   {
      const char token = infix[i];

      if (isdigit(token))
      {
         strncat(result, infix + i, 1);
      }
      else if (is_operator(token))
      {
         strncat(result, " ", 1);

         while (operators->size > 0 &&
            stack_peek(operators, 0)[0] != '(' &&
            (op_precedence(stack_peek(operators, 0)[0]) > op_precedence(token) ||
               (op_precedence(stack_peek(operators, 0)[0]) == op_precedence(token) && op_associativity(token) ==
0)))
         {
            char ds[2];
            sprintf(ds, "%c", stack_pop_first_char(operators));              strncat(result, ds, 1);
         }
         stack_push(operators, str_new(token));
      }
      else if (token == '(')
      {
         strncat(result, " ", 1);

         stack_push(operators, str_new(token));
      }
      else if (token == ')')
      {
         strncat(result, " ", 1);

         while (stack_peek(operators, 0)[0] != '(')
         {
            char ds[2];
            sprintf(ds, "%c", stack_pop_first_char(operators));              strncat(result, ds, 1);
         }

         stack_pop_first_char(operators);
      }
   }
   while (operators->size > 0)
   {       char ds[2];
      sprintf(ds, "%c", stack_pop_first_char(operators));        strncat(result, ds, 1);
   }
   for (int i = 0; i < operators->size; i++)
stack_pop_first_char(operators);     free(operators->elements);
free(operators);
```

```
   return result;
}
int evaluate_rpn(const char* expression)
{
   int stacking_number = 0;     int r, l;
   Stack* stack = create_stack();     const int exp_len =
strlen(expression);     for (int i = 0; i < exp_len; i++)
   {
      switch (expression[i])
      {        case '+':            stacking_number = 0;            r
= stack_pop_number(stack);            l =
stack_pop_number(stack);
          stack_push(stack, str_new_from_number(l + r));            break;        case '-':
          stacking_number = 0;            r =
stack_pop_number(stack);            l =
stack_pop_number(stack);
          stack_push(stack, str_new_from_number(l - r));            break;        case '*':
          stacking_number = 0;            r =
stack_pop_number(stack);            l =
stack_pop_number(stack);
          stack_push(stack, str_new_from_number(l * r));            break;        case '/':
stacking_number = 0;            r = stack_pop_number(stack);            l =
stack_pop_number(stack);            if (r == 0) {                printf("Try again");
continue; }            else {
             stack_push(stack, str_new_from_number(l / r));
          }            break;
case '^':
          stacking_number = 0;
```

```
        r = stack_pop_number(stack);          l =
stack_pop_number(stack);
        stack_push(stack, str_new_from_number((int)pow(l, r)));          break;       case ' ':
stacking_number = 0;          break;       default:          if (stacking_number == 1)
            stack->elements[0] = str_append(stack->elements[0], expression[i]);          else          {
            stack_push(stack, str_new(expression[i]));              stacking_number = 1;
        }          break;
      }
   }
   const int result = stack_pop_number(stack);


   for (int i = 0; i < stack->size; i++)
stack_pop_first_char(stack);    free(stack->elements);
free(stack);


   return result;
}
int main(void)
{
   printf(" \n\nEnter the expression\n ");    char input[1024];
   scanf("%1023[^\n]", input);


   char* rpn = infix_to_rpn(input);    printf("\nRPN >> %s\n", rpn);
   printf("\n Result = %d\n", evaluate_rpn(rpn));


   free(rpn);


}
```

9 ПРИЛОЖЕНИЕ Г. СЛАЙДЫ ПРЕЗЕНТАЦИИ

Разработка программы для вычисления математических выражений

Подготовил студент:
Филькин Тимофей,
ИУ10-21

Введение

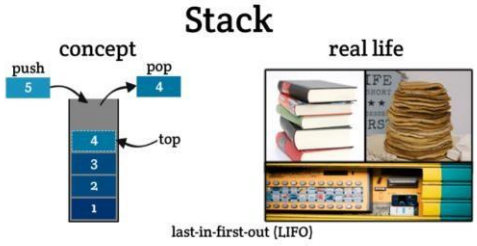
Цели работы:

- Написать программу, которая будет вычислять математические выражения произвольной длины
- Научиться работать с элементами языка Си.

Задачи:

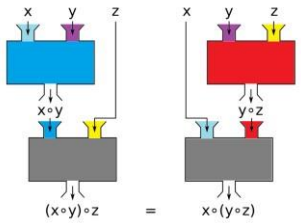
- Вспомнить основные инструменты для написания программы на языке Си.
- Изучение алгоритмических и языковых особенностей для работы со строками и представления чисел.

Принцип хранения данных



Ассоциативность

Примером неассоциативной операции является возведение в степень — результат выражения (a)^(b^c) != ((a)^b)^c



Алгоритм Дейкстры

- Пока не все токены обработаны:
- Прочитать токен.
- Если токен — число, то добавить его в очередь вывода.
- Если токен — функция, то поместить его в стек.
- Если токен — разделитель аргументов функции (например, запятая):
- Пока токен на вершине стека не открывающая скобка:
- Переложить оператор из стека в выходную очередь.
- Если стек закончился до того, как был встречен токен открывающая скобка, то в выражении пропущен разделитель аргументов функции (запятая), либо пропущена открывающая скобка.
- Если токен — оператор op1, то:
- Пока присутствует на вершине стека токен оператор op2, чей приоритет выше или равен приоритету op1, и при равенстве приоритетов op1 является левоассоциативным:
- Переложить op2 из стека в выходную очередь;
- Положить op1 в стек.
- Если токен — открывающая скобка, то положить его в стек.
- Если токен — закрывающая скобка:
- Пока токен на вершине стека не открывающая скобка
- Переложить оператор из стека в выходную очередь.
- Если стек закончился до того, как был встречен токен открывающая скобка, то в выражении пропущена скобка.
- Выкинуть открывающую скобку из стека, но не добавлять в очередь вывода.
- Если токен на вершине стека — функция, переложить её в выходную очередь.
- Если больше не осталось токенов на входе:
- Пока есть токены операторы в стеке:
- Если токен оператор на вершине стека — открывающая скобка, то в выражении пропущена скобка.
- Переложить оператор из стека в выходную очередь.
- Конец.

Представление выражения

Обратная запись выражения

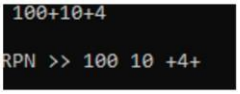
Рассмотрим выражение (8+2*5)/(1+3*2-4)

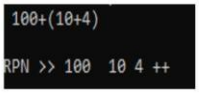
Шаг	Оставшаяся цепочка	Стек
1	8, 2,5, *+1,3,2*+4-/	8
2	2,5, *+1,3,2*+4-/	8,2
3	5*+1,3,2*+4-/	8,2,5
4	*+1,3,2*+4-/	8,10
5	+1,3,2*+4-/	18
6	1,3,2*+4-/	18,1
7	3,2*+4-/	18,1,3
8	2*+4-/	18,1,3,2
9	*+4-/	18,1,6
10	+4-/	18,7
11	4-/	18,7,4
12	-/	18,3
13	/	6

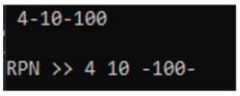
3 4 +

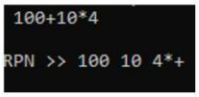
Практическая часть

Программа вычисляет выражения с использованием таких действий как умножение, возведение в степень, деление, сложение и вычитание. В случае когда выражение имеет операторы с разными приоритетами, знаки операций располагаются от самого высокого до самого низкого.









Заключение

В ходе выполнения курсовой работы был создана программа, способная выполнять вычисления математических выражений, содержащее произвольное количество действий, обладающих разными приоритетами.

Положительными особенностями данного кода программы являются его лаконичность, обособленность выполняемых функций, легкость чтения и понимания кода, возможностью простого добавления обработки требуемых математических операций. Отдельным моментом хочется отметить удобство считывания числа с заданным количеством символов и реализацию жесткого присвоения приоритета отдельным операциям, путем добавления в исходное выражение скобок. Программа может быть использована в более сложных проектах, где будет являться независимой частью.

Цель работы была достигнута, программа работает исправно.