МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 14

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
|  |
| КУРСОВАЯ РАБОТА |
| по дисциплине: ОСНОВЫ ПРОГРАММИРОВАНИЯ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № |  |  |  |  |  |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург  
2020

**Оглавление:**

[Введение 3](#_Toc56340559)

[1. Постановка задачи 3](#_Toc56340560)

[2. Технические характеристики и ограничения применения 3](#_Toc56340561)

[3. Структура данных 4](#_Toc56340562)

[4. Схема алгоритма 5](#_Toc56340565)

[5. Листинг программы 14](#_Toc56340566)

[6. Тестовые примеры 29](#_Toc56340567)

[7. Инструкция по применению 31](#_Toc56340568)

[8. Спецификация программы 32](#_Toc56340569)

[Заключение 32](#_Toc56340570)

[9. Список использованной литературы 32](#_Toc56340571)

# Введение

Программный продукт предназначен для выполнения математических действий над выражениями, состоящими из полиномов. Необходимо обеспечить корректное считывание таких выражений, проверку их на правильность ввода, интерпретацию в удобном для дальнейших вычислений виде, и на основании полученных данных производить следующие действия с ними: сложение, вычитание, умножение и деление – с учётом порядка ввода выражений в скобках.

Системные требования:

* Процессор с тактовой частотой 1,6 ГГц и более;
* 1 ГБ ОЗУ;
* 2.5 Мб доступного пространства на жестком диске;
* Windows 10 с пакетом обновления SP1 и выше;
* Стандартный видеоадаптер.

# 1. Постановка задачи

Калькулятор для полиномов.

* Вычисления с полиномами в символьном виде.
* Операции сложения, вычитания, умножения и деления.
* Вычисление выражений со скобками.

# 2. Технические характеристики и ограничения применения

В программе имеется одно численное ограничение и два производных от него:

* N – ограничение на ввод символов с клавиатуры (по умолчанию 200);
* (N / 2) – ограничение на количество переменных в остатке;
* (N \* N / 4) – ограничение на количество скобок, выражений в скобках, количество переменных в выражениях в скобках, количество переменных в результате выполненных действий.

Ограничение (N / 2) взято из расчёта, что при максимально компактной записи выражения на каждую переменную отводится по 2 символа, а значит, будет считано в два раза меньше переменных, чем количество символов во введённой строке.

(N \* N / 4) = (N / 2)^2, взято из расчёта, что при выполнении операции умножения в результате может получиться количество переменных, равное произведению двух умножаемых скобок.

Для успешного считывания вводимого выражения необходимо правильно расставить скобки (каждой открывающей скобке должна соответствовать закрывающая и наоборот), проверить наличие знаков действий между скобками, а также наличие переменных или скобок между каждыми знаками, вводить только целочисленные, неотрицательные и не содержащие переменных степени, не допускать ввод лишних знаков, не предусмотренных программой. В противном случае, будет выведено сообщение об ошибке и ввод выражения следует повторить заново.

3. Структура данных

Для записи исходного введённого с клавиатуры выражения в программе определён символьный массив:

* char expr[N], длина N по умолчанию 200.

Для записи информации о скобках определена структура bracket:

* int num – численное поле для записи номера скобки;
* int ord – численное поле для записи порядка (приоритета) выполнения действий над выражением в скобках.

Для записи информации о переменных определена структура variable:

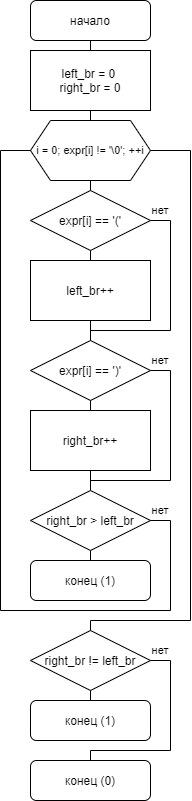
* bracket\* br – указатель на скобку, если переменная – коэффициент перед скобкой;
* float cft – дробное поле для записи коэффициента переменной;
* int pwr – целочисленное поле для записи степени переменной;
* char sgn – символьное поле для записи знака действия перед переменной.

Для выполнения дальнейших вычислений определены следующие массивы вышеописанных структур:

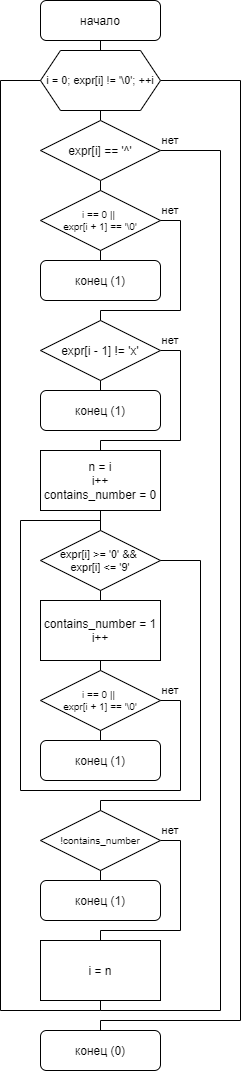
* variable res[N \* N / 4][N \* N / 4] – массив, в который считывается каждая переменная из исходного введённого с клавиатуры выражения;
* variable tmp[N \* N / 4] – массив, в который временно записываются переменные, полученные в результате выполнения действий;
* variable mod\_tmp[N / 2] – массив, в который записывается остаток от самого последнего деления;
* bracket br[N \* N / 4] – массив, в который записывается информация о скобках.

# 4. Схема алгоритма

лист 1

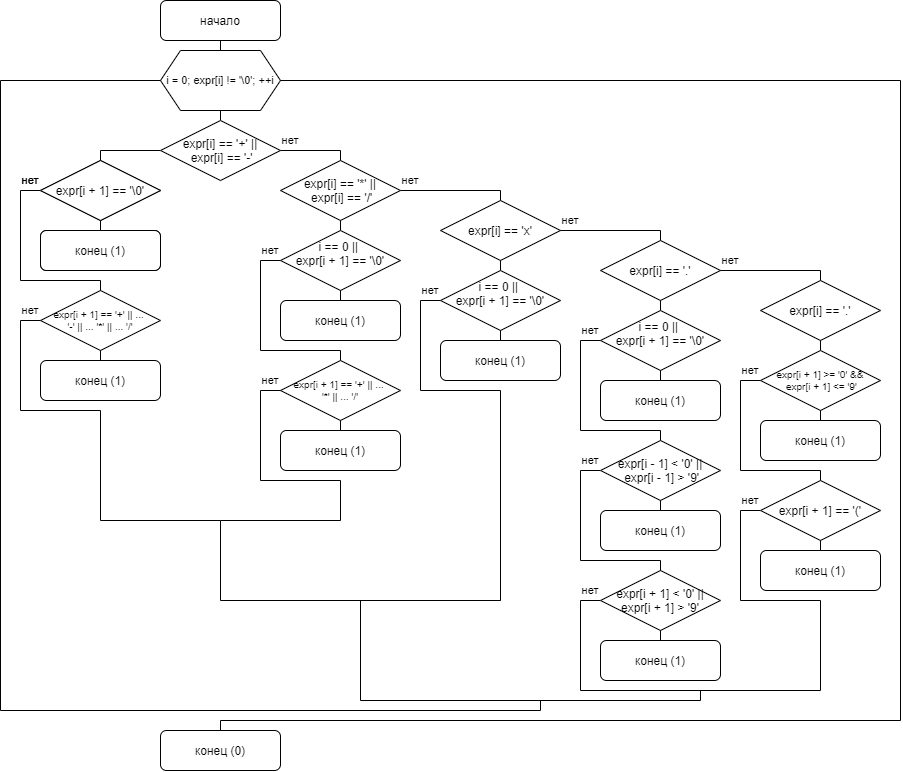


*Рисунок 1 – Алгоритм проверки ввода скобок*

лист 1

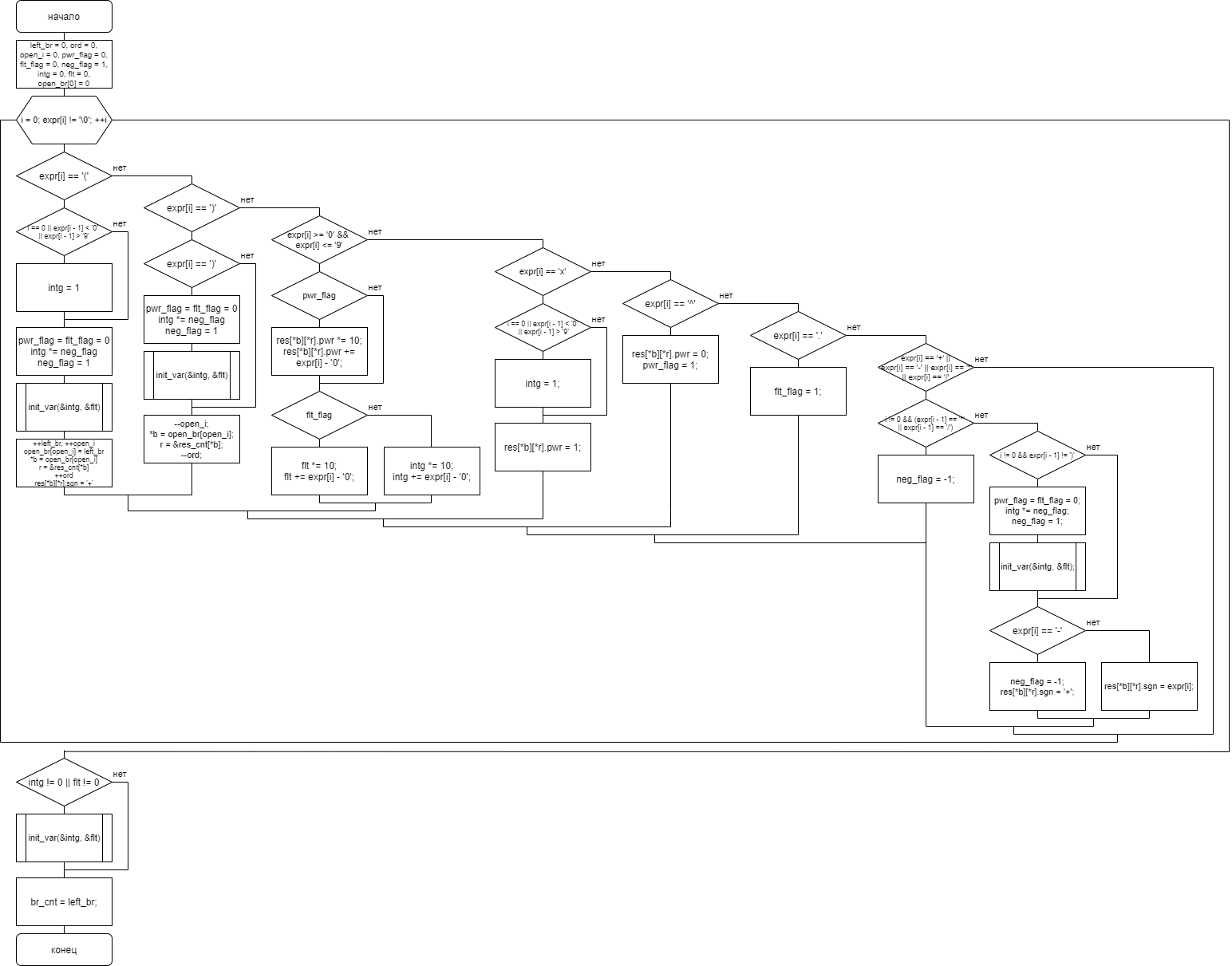
лист 1

*Рисунок 2 – Алгоритм проверки ввода степени*

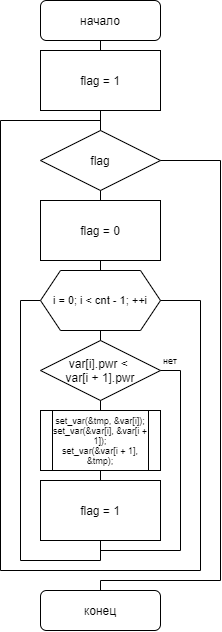


*Рисунок 3 – Алгоритм проверки ввода знаков действий*

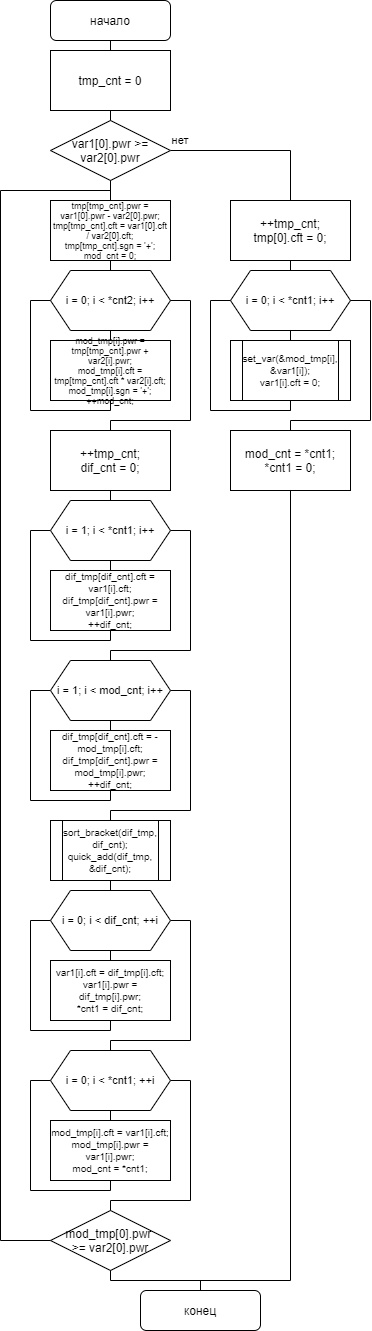
лист 1



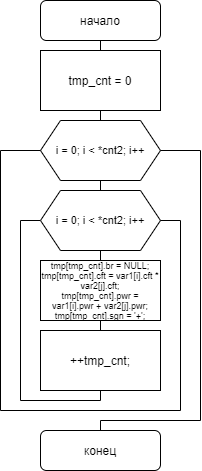
*Рисунок 4 – Алгоритм чтения введённого выражения*

лист 1

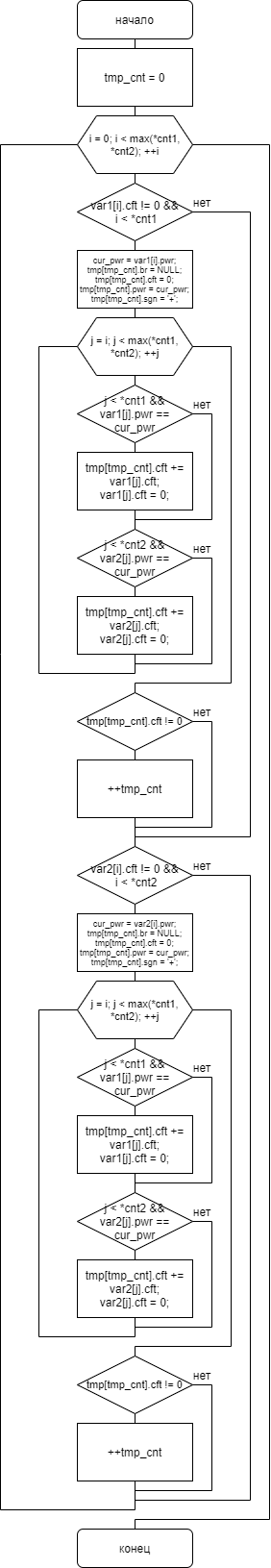
*Рисунок 5 – Алгоритм сортировки полинома*

лист 1

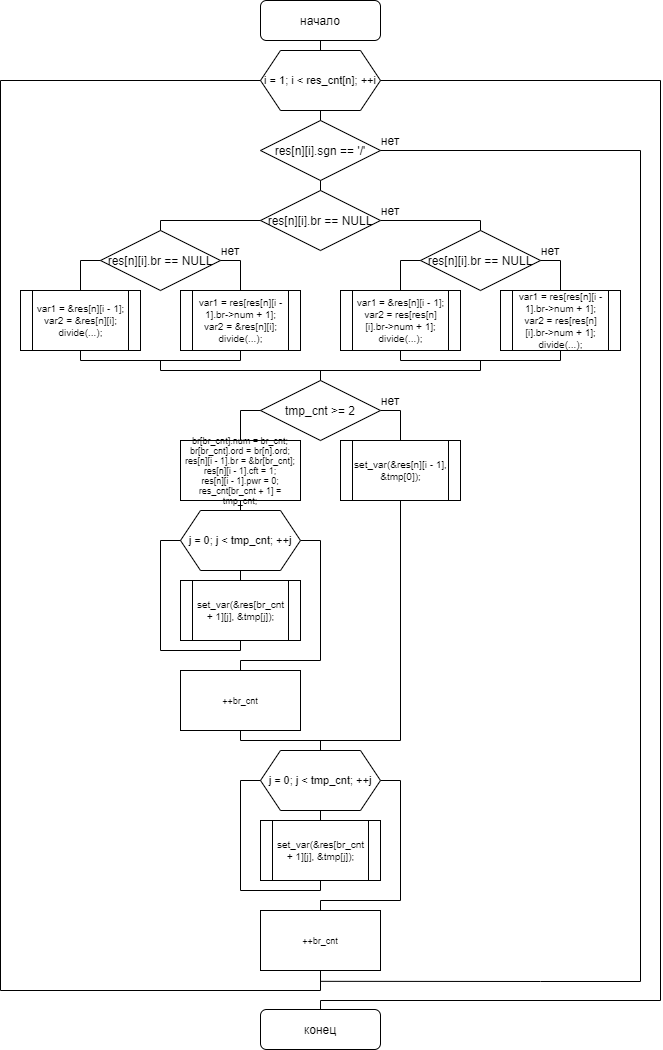
*Рисунок 6 – Алгоритм деления полиномов*

лист 1

*Рисунок 7 – Алгоритм умножения полиномов*

 лист 1

*Рисунок 8 – Алгоритм сложения полиномов*

лист 1

*Рисунок 9 – Алгоритм нахождения действия (на примере деления)*

# 5. Листинг программы

#include <stdio.h>

#include <conio.h>

#include <Windows.h>

#include <locale.h>

#define N 200

char expr[N];

typedef struct

{

int num;

int ord;

} bracket;

typedef struct

{

bracket\* br;

float cft;

int pwr;

char sgn;

} variable;

variable res[N \* N / 4][N \* N / 4];

variable tmp[N \* N / 4], mod\_tmp[N / 2];

bracket br[N \* N / 4];

int br\_cnt = 0, cur\_br = 0;

int res\_cnt[N \* N / 4], res\_br\_cnt = 0, tmp\_cnt, mod\_cnt;

int act\_cnt = 0;

void remove\_spaces()

{

for (int i = 0; expr[i] != '\0'; ++i)

{

while (expr[i] == ' ')

{

for (int j = i; expr[j] != '\0'; ++j)

{

expr[j] = expr[j + 1];

}

}

}

}

int check\_brackets()

{

int left\_br = 0, right\_br = 0;

for (int i = 0; expr[i] != '\0'; ++i)

{

if (expr[i] == '(')

left\_br++;

if (expr[i] == ')')

right\_br++;

if (right\_br > left\_br)

return 1;

}

if (right\_br != left\_br)

return 1;

return 0;

}

int check\_powers()

{

int n, contains\_number;

for (int i = 0; expr[i] != '\0'; ++i)

{

if (expr[i] == '^')

{

if (i == 0 || expr[i + 1] == '\0')

return 1;

if (expr[i - 1] != 'x')

return 1;

n = i;

i++;

contains\_number = 0;

while (expr[i] >= '0' && expr[i] <= '9')

{

contains\_number = 1;

i++;

if (expr[i] == 'x' || expr[i] == '.')

return 1;

}

if (!contains\_number)

return 1;

i = n;

}

}

return 0;

}

int check\_signs()

{

for (int i = 0; expr[i] != '\0'; ++i)

{

if (expr[i] == '+' || expr[i] == '-')

{

if (expr[i + 1] == '\0')

return 1;

if (expr[i + 1] == '+' || expr[i + 1] == '-' || expr[i + 1] == '\*' || expr[i + 1] == '/')

return 1;

}

else if (expr[i] == '\*' || expr[i] == '/')

{

if (i == 0 || expr[i + 1] == '\0')

return 1;

if (expr[i + 1] == '\*' || expr[i + 1] == '/' || expr[i + 1] == '+')

return 1;

}

else if (expr[i] == 'x')

{

if (expr[i + 1] == 'x')

return 1;

}

else if (expr[i] == '.')

{

if (i == 0)

return 1;

if (expr[i - 1] < '0' || expr[i - 1] > '9')

return 1;

if (expr[i + 1] < '0' || expr[i + 1] > '9')

return 1;

}

else if (expr[i] == ')')

{

if (expr[i + 1] >= '0' && expr[i + 1] <= '9')

return 1;

if (expr[i + 1] == '(')

return 1;

}

}

return 0;

}

int check\_characters()

{

for (int i = 0; expr[i] != '\0'; ++i)

{

if (

expr[i] != '(' &&

expr[i] != ')' &&

expr[i] != 'x' &&

expr[i] != '.' &&

expr[i] != '^' &&

expr[i] != '+' &&

expr[i] != '-' &&

expr[i] != '\*' &&

expr[i] != '/' &&

(expr[i] < '0' || expr[i] > '9')

)

return 1;

}

return 0;

}

int check\_expression()

{

remove\_spaces();

if (check\_brackets())

{

printf("\nОшибка! Неправильная постановка скобок\n");

system("pause");

return 0;

}

if (check\_powers())

{

printf("\nОшибка! Неправильное значение степени\n");

system("pause");

return 0;

}

if (check\_signs())

{

printf("\nОшибка! Неправильная постановка знаков действий\n");

system("pause");

return 0;

}

if (check\_characters())

{

printf("\nОшибка! Введены недопустимые символы\n");

system("pause");

return 0;

}

return 1;

}

void init\_var(float\* intg, float\* flt)

{

int\* b = &cur\_br, \* r = &res\_cnt[cur\_br];

if (res[\*b][\*r].sgn == '\0')

res[\*b][\*r].sgn = '+';

while (\*flt >= 1)

\*flt /= 10;

res[\*b][\*r].cft = \*intg + \*flt;

\*intg = \*flt = 0;

++(\*r);

}

void read\_expression()

{

int\* b = &cur\_br, \* r = &res\_cnt[cur\_br];

int left\_br = 0, ord = 0, open\_br[N / 2], open\_i = 0;

int pwr\_flag = 0, flt\_flag = 0, neg\_flag = 1;

float intg = 0, flt = 0;

open\_br[0] = 0;

for (int i = 0; expr[i] != '\0'; ++i)

{

if (expr[i] == '(')

{

if (i == 0 || expr[i - 1] < '0' || expr[i - 1] > '9')

{

intg = 1;

}

br[left\_br].ord = ord;

br[left\_br].num = left\_br;

res[\*b][\*r].br = &br[left\_br];

pwr\_flag = flt\_flag = 0;

intg \*= neg\_flag;

neg\_flag = 1;

init\_var(&intg, &flt);

++left\_br;

++open\_i;

open\_br[open\_i] = left\_br;

\*b = open\_br[open\_i];

r = &res\_cnt[\*b];

++ord;

res[\*b][\*r].sgn = '+';

}

else if (expr[i] == ')')

{

if (expr[i - 1] != ')')

{

pwr\_flag = flt\_flag = 0;

intg \*= neg\_flag;

neg\_flag = 1;

init\_var(&intg, &flt);

}

--open\_i;

\*b = open\_br[open\_i];

r = &res\_cnt[\*b];

--ord;

}

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

{

if (pwr\_flag)

{

res[\*b][\*r].pwr \*= 10;

res[\*b][\*r].pwr += expr[i] - '0';

}

else if (flt\_flag)

{

flt \*= 10;

flt += expr[i] - '0';

}

else

{

intg \*= 10;

intg += expr[i] - '0';

}

}

else if (expr[i] == 'x')

{

if (i == 0 || expr[i - 1] < '0' || expr[i - 1] > '9')

{

intg = 1;

}

res[\*b][\*r].pwr = 1;

}

else if (expr[i] == '^')

{

res[\*b][\*r].pwr = 0;

pwr\_flag = 1;

}

else if (expr[i] == '.')

{

flt\_flag = 1;

}

else if (expr[i] == '+' || expr[i] == '-' || expr[i] == '\*' || expr[i] == '/')

{

if (i != 0 && (expr[i - 1] == '\*' || expr[i - 1] == '/'))

{

neg\_flag = -1;

}

else

{

if (i != 0 && expr[i - 1] != ')')

{

pwr\_flag = flt\_flag = 0;

intg \*= neg\_flag;

neg\_flag = 1;

init\_var(&intg, &flt);

}

if (expr[i] == '-')

{

neg\_flag = -1;

res[\*b][\*r].sgn = '+';

}

else

res[\*b][\*r].sgn = expr[i];

}

}

}

if (intg != 0 || flt != 0)

{

intg \*= neg\_flag;

while (flt >= 1)

flt /= 10;

res[\*b][\*r].cft = intg + flt;

++(\*r);

}

br\_cnt = left\_br;

}

void output\_expression(variable\* var, int cnt)

{

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

{

if (var[i].cft == 0)

{

printf("0");

}

else

{

if (i == 0)

{

if (var[i].cft < 0)

printf("-");

}

else

{

if (var[i].cft < 0)

{

if (var[i].sgn == '+')

printf(" %c ", '-');

else

printf(" %c -", var[i].sgn);

}

else

printf(" %c ", var[i].sgn);

}

if (var[i].cft < 0)

printf("%.2f", -var[i].cft);

else

printf("%.2f", var[i].cft);

if (var[i].br != NULL)

{

printf(" (");

output\_expression(res[var[i].br->num + 1], res\_cnt[var[i].br->num + 1]);

printf(")");

}

else

{

if (var[i].pwr == 1)

printf(" x");

else if (var[i].pwr != 0)

printf(" x^%d", var[i].pwr);

}

}

}

}

void set\_var(variable\* var1, variable\* var2)

{

var1->br = var2->br;

var1->cft = var2->cft;

var1->pwr = var2->pwr;

var1->sgn = var2->sgn;

}

void sort\_bracket(variable\* var, int cnt)

{

variable tmp;

int flag = 1;

while (flag)

{

flag = 0;

for (int i = 0; i < cnt - 1; ++i)

{

if (var[i].pwr < var[i + 1].pwr)

{

set\_var(&tmp, &var[i]);

set\_var(&var[i], &var[i + 1]);

set\_var(&var[i + 1], &tmp);

flag = 1;

}

}

}

}

void quick\_add(variable\* var, int\* cnt)

{

for (int i = 0; i < \*cnt - 1; ++i)

{

if (var[i].cft == 0)

{

for (int j = i; j < \*cnt - 1; ++j)

{

set\_var(&var[j], &var[j + 1]);

}

--(\*cnt);

--i;

continue;

}

if (var[i].pwr == var[i + 1].pwr)

{

var[i].cft += var[i + 1].cft;

for (int j = i + 1; j < \*cnt - 1; ++j)

{

set\_var(&var[j], &var[j + 1]);

}

--(\*cnt);

--i;

}

}

}

void divide(variable\* var1, variable\* var2, int\* cnt1, int\* cnt2)

{

variable dif\_tmp[N \* N / 4];

int dif\_cnt;

tmp\_cnt = 0;

if (var1[0].pwr >= var2[0].pwr)

{

do

{

tmp[tmp\_cnt].pwr = var1[0].pwr - var2[0].pwr;

tmp[tmp\_cnt].cft = var1[0].cft / var2[0].cft;

tmp[tmp\_cnt].sgn = '+';

mod\_cnt = 0;

for (int i = 0; i < \*cnt2; i++)

{

mod\_tmp[i].pwr = tmp[tmp\_cnt].pwr + var2[i].pwr;

mod\_tmp[i].cft = tmp[tmp\_cnt].cft \* var2[i].cft;

mod\_tmp[i].sgn = '+';

++mod\_cnt;

}

++tmp\_cnt;

dif\_cnt = 0;

for (int i = 1; i < \*cnt1; i++)

{

dif\_tmp[dif\_cnt].cft = var1[i].cft;

dif\_tmp[dif\_cnt].pwr = var1[i].pwr;

++dif\_cnt;

}

for (int i = 1; i < mod\_cnt; i++)

{

dif\_tmp[dif\_cnt].cft = -mod\_tmp[i].cft;

dif\_tmp[dif\_cnt].pwr = mod\_tmp[i].pwr;

++dif\_cnt;

}

sort\_bracket(dif\_tmp, dif\_cnt);

quick\_add(dif\_tmp, &dif\_cnt);

for (int i = 0; i < dif\_cnt; ++i)

{

var1[i].cft = dif\_tmp[i].cft;

var1[i].pwr = dif\_tmp[i].pwr;

}

\*cnt1 = dif\_cnt;

for (int i = 0; i < \*cnt1; ++i)

{

mod\_tmp[i].cft = var1[i].cft;

mod\_tmp[i].pwr = var1[i].pwr;

}

mod\_cnt = \*cnt1;

} while (mod\_tmp[0].pwr >= var2[0].pwr);

}

else

{

++tmp\_cnt;

tmp[0].cft = 0;

for (int i = 0; i < \*cnt1; i++)

{

set\_var(&mod\_tmp[i], &var1[i]);

var1[i].cft = 0;

}

mod\_cnt = \*cnt1;

\*cnt1 = 0;

}

}

void multiply(variable\* var1, variable\* var2, int\* cnt1, int\* cnt2)

{

tmp\_cnt = 0;

for (int i = 0; i < \*cnt1; ++i)

{

for (int j = 0; j < \*cnt2; ++j)

{

tmp[tmp\_cnt].br = NULL;

tmp[tmp\_cnt].cft = var1[i].cft \* var2[j].cft;

tmp[tmp\_cnt].pwr = var1[i].pwr + var2[j].pwr;

tmp[tmp\_cnt].sgn = '+';

++tmp\_cnt;

}

}

}

void add\_sub(variable\* var1, variable\* var2, int\* cnt1, int\* cnt2)

{

int cur\_pwr;

tmp\_cnt = 0;

for (int i = 0; i < max(\*cnt1, \*cnt2); ++i)

{

if (var1[i].cft != 0 && i < \*cnt1)

{

cur\_pwr = var1[i].pwr;

tmp[tmp\_cnt].br = NULL;

tmp[tmp\_cnt].cft = 0;

tmp[tmp\_cnt].pwr = cur\_pwr;

tmp[tmp\_cnt].sgn = '+';

for (int j = i; j < max(\*cnt1, \*cnt2); ++j)

{

if (j < \*cnt1 && var1[j].pwr == cur\_pwr)

{

tmp[tmp\_cnt].cft += var1[j].cft;

var1[j].cft = 0;

}

if (j < \*cnt2 && var2[j].pwr == cur\_pwr)

{

tmp[tmp\_cnt].cft += var2[j].cft;

var2[j].cft = 0;

}

}

if (tmp[tmp\_cnt].cft != 0)

++tmp\_cnt;

}

if (var2[i].cft != 0 && i < \*cnt2)

{

cur\_pwr = var2[i].pwr;

tmp[tmp\_cnt].br = NULL;

tmp[tmp\_cnt].cft = 0;

tmp[tmp\_cnt].pwr = cur\_pwr;

tmp[tmp\_cnt].sgn = '+';

for (int j = i; j < max(\*cnt1, \*cnt2); ++j)

{

if (j < \*cnt1 && var1[j].pwr == cur\_pwr)

{

tmp[tmp\_cnt].cft += var1[j].cft;

var1[j].cft = 0;

}

if (j < \*cnt2 && var2[j].pwr == cur\_pwr)

{

tmp[tmp\_cnt].cft += var2[j].cft;

var2[j].cft = 0;

}

}

if (tmp[tmp\_cnt].cft != 0)

++tmp\_cnt;

}

}

}

void if\_div(int n)

{

variable\* var1, \* var2;

int one = 1;

for (int i = 1; i < res\_cnt[n]; ++i)

{

if (res[n][i].sgn == '/')

{

if (res[n][i].br == NULL)

{

if (res[n][i - 1].br == NULL)

{

var1 = &res[n][i - 1];

var2 = &res[n][i];

++act\_cnt;

printf("\n Действие %d:\n\n", act\_cnt);

printf(" %.2f x^%d / %.2f x^%d = ", var1->cft, var1->pwr, var2->cft, var2->pwr);

divide(var1, var2, &one, &one);

}

else

{

var1 = res[res[n][i - 1].br->num + 1];

var2 = &res[n][i];

++act\_cnt;

printf("\n Действие %d:\n\n", act\_cnt);

printf(" (");

output\_expression(var1, res\_cnt[res[n][i - 1].br->num + 1]);

printf(") / %.2f x^%d = ", var2->cft, var2->pwr);

divide(var1, var2, &res\_cnt[res[n][i - 1].br->num + 1], &one);

}

}

else

{

if (res[n][i - 1].br == NULL)

{

var1 = &res[n][i - 1];

var2 = res[res[n][i].br->num + 1];

++act\_cnt;

printf("\n Действие %d:\n\n", act\_cnt);

printf(" %.2f x^%d / (", var1->cft, var1->pwr);

output\_expression(var2, res\_cnt[res[n][i].br->num + 1]);

printf(") = ");

divide(var1, var2, &one, &res\_cnt[res[n][i].br->num + 1]);

}

else

{

var1 = res[res[n][i - 1].br->num + 1];

var2 = res[res[n][i].br->num + 1];

++act\_cnt;

printf("\n Действие %d:\n\n", act\_cnt);

printf(" (");

output\_expression(var1, res\_cnt[res[n][i - 1].br->num + 1]);

printf(") / (");

output\_expression(var2, res\_cnt[res[n][i].br->num + 1]);

printf(") = ");

divide(var1, var2, &res\_cnt[res[n][i - 1].br->num + 1], &res\_cnt[res[n][i].br->num + 1]);

}

}

if (tmp\_cnt >= 2)

{

br[br\_cnt].num = br\_cnt;

br[br\_cnt].ord = br[n].ord;

res[n][i - 1].br = &br[br\_cnt];

res[n][i - 1].cft = 1;

res[n][i - 1].pwr = 0;

res\_cnt[br\_cnt + 1] = tmp\_cnt;

for (int j = 0; j < tmp\_cnt; ++j)

{

set\_var(&res[br\_cnt + 1][j], &tmp[j]);

}

++br\_cnt;

}

else

{

set\_var(&res[n][i - 1], &tmp[0]);

}

for (int j = i + 1; j < res\_cnt[n]; ++j)

{

set\_var(&res[n][j - 1], &res[n][j]);

}

--res\_cnt[n];

--i;

output\_expression(tmp, tmp\_cnt);

printf(" (Остаток: ");

output\_expression(mod\_tmp, mod\_cnt);

printf(")\n\n");

system("pause");

}

}

}

void if\_mul(int n)

{

variable\* var1, \* var2;

int one = 1;

for (int i = 0; i < res\_cnt[n]; ++i)

{

if (res[n][i].br != NULL)

{

if (res[n][i].cft != 1)

{

var1 = &res[n][i];

var2 = res[res[n][i].br->num + 1];

++act\_cnt;

printf("\n Действие %d:\n\n", act\_cnt);

printf(" %.2f \* (", var1->cft);

output\_expression(var2, res\_cnt[res[n][i].br->num + 1]);

printf(") = ");

multiply(var1, var2, &one, &res\_cnt[res[n][i].br->num + 1]);

res[n][i].cft = 1;

for (int j = 0; j < tmp\_cnt; ++j)

{

set\_var(&var2[j], &tmp[j]);

}

output\_expression(tmp, tmp\_cnt);

printf("\n\n");

system("pause");

}

}

if (res[n][i].sgn == '\*')

{

if (res[n][i].br == NULL)

{

if (res[n][i - 1].br == NULL)

{

var1 = &res[n][i - 1];

var2 = &res[n][i];

++act\_cnt;

printf("\n Действие %d:\n\n", act\_cnt);

printf(" %.2f x^%d \* %.2f x^%d = ", var1->cft, var1->pwr, var2->cft, var2->pwr);

multiply(var1, var2, &one, &one);

}

else

{

var1 = res[res[n][i - 1].br->num + 1];

var2 = &res[n][i];

++act\_cnt;

printf("\n Действие %d:\n\n", act\_cnt);

printf(" (");

output\_expression(var1, res\_cnt[res[n][i - 1].br->num + 1]);

printf(") \* %.2f x^%d = ", var2->cft, var2->pwr);

multiply(var1, var2, &res\_cnt[res[n][i - 1].br->num + 1], &one);

}

}

else

{

if (res[n][i - 1].br == NULL)

{

var1 = &res[n][i - 1];

var2 = res[res[n][i].br->num + 1];

++act\_cnt;

printf("\n Действие %d:\n\n", act\_cnt);

printf(" %.2f x^%d \* (", var1->cft, var1->pwr);

output\_expression(var2, res\_cnt[res[n][i].br->num + 1]);

printf(") = ");

multiply(var1, var2, &one, &res\_cnt[res[n][i].br->num + 1]);

}

else

{

var1 = res[res[n][i - 1].br->num + 1];

var2 = res[res[n][i].br->num + 1];

++act\_cnt;

printf("\n Действие %d:\n\n", act\_cnt);

printf(" (");

output\_expression(var1, res\_cnt[res[n][i - 1].br->num + 1]);

printf(") \* (");

output\_expression(var2, res\_cnt[res[n][i].br->num + 1]);

printf(") = ");

multiply(var1, var2, &res\_cnt[res[n][i - 1].br->num + 1], &res\_cnt[res[n][i].br->num + 1]);

}

}

if (tmp\_cnt >= 2)

{

br[br\_cnt].num = br\_cnt;

br[br\_cnt].ord = br[n].ord;

res[n][i - 1].br = &br[br\_cnt];

res[n][i - 1].cft = 1;

res[n][i - 1].pwr = 0;

sort\_bracket(tmp, tmp\_cnt);

quick\_add(tmp, &tmp\_cnt);

res\_cnt[br\_cnt + 1] = tmp\_cnt;

for (int j = 0; j < tmp\_cnt; ++j)

{

set\_var(&res[br\_cnt + 1][j], &tmp[j]);

}

++br\_cnt;

}

else

{

set\_var(&res[n][i - 1], &tmp[0]);

}

for (int j = i + 1; j < res\_cnt[n]; ++j)

{

set\_var(&res[n][j - 1], &res[n][j]);

}

--res\_cnt[n];

--i;

output\_expression(tmp, tmp\_cnt);

printf("\n\n");

system("pause");

}

}

}

void if\_add\_sub(int n)

{

variable\* var1, \* var2;

int one = 1;

++act\_cnt;

printf("\n Действие %d:\n\n ", act\_cnt);

output\_expression(res[n], res\_cnt[n]);

printf(" = ");

for (int i = 1; i < res\_cnt[n]; ++i)

{

if (res[n][i].sgn == '+')

{

if (res[n][i].br == NULL)

{

if (res[n][i - 1].br == NULL)

{

var1 = &res[n][i - 1];

var2 = &res[n][i];

add\_sub(var1, var2, &one, &one);

}

else

{

var1 = res[res[n][i - 1].br->num + 1];

var2 = &res[n][i];

add\_sub(var1, var2, &res\_cnt[res[n][i - 1].br->num + 1], &one);

}

}

else

{

if (res[n][i - 1].br == NULL)

{

var1 = &res[n][i - 1];

var2 = res[res[n][i].br->num + 1];

add\_sub(var1, var2, &one, &res\_cnt[res[n][i].br->num + 1]);

}

else

{

var1 = res[res[n][i - 1].br->num + 1];

var2 = res[res[n][i].br->num + 1];

add\_sub(var1, var2, &res\_cnt[res[n][i - 1].br->num + 1], &res\_cnt[res[n][i].br->num + 1]);

}

}

if (tmp\_cnt >= 2)

{

for (int j = 0; j < tmp\_cnt - 2; ++j)

{

set\_var(&res[n][res\_cnt[n] - 1 - j + tmp\_cnt], &res[n][res\_cnt[n] - 1 - j]);

}

for (int j = 0; j < tmp\_cnt; ++j)

{

set\_var(&res[n][i - 1 + j], &tmp[j]);

}

res\_cnt[n] += tmp\_cnt - 2;

}

else

{

set\_var(&res[n][i - 1], &tmp[0]);

for (int j = i + 1; j < res\_cnt[n]; ++j)

{

set\_var(&res[n][j - 1], &res[n][j]);

}

--res\_cnt[n];

--i;

}

}

}

sort\_bracket(res[n], res\_cnt[n]);

quick\_add(res[n], &res\_cnt[n]);

output\_expression(res[n], res\_cnt[n]);

printf("\n\n");

system("pause");

}

void proc\_bracket(int n)

{

if\_div(n);

if\_mul(n);

if\_add\_sub(n);

}

void proc\_expression()

{

int max\_ord = 0;

for (int i = 0; i < br\_cnt; ++i)

{

if (br[i].ord > max\_ord)

++max\_ord;

}

for (int i = max\_ord; i >= 0; --i)

{

for (int j = 0; j < br\_cnt; ++j)

{

if (br[j].ord == i)

proc\_bracket(j + 1);

}

}

proc\_bracket(0);

}

void main(void)

{

setlocale(LC\_ALL, "Russian");

int correct = 0;

while (!correct)

{

memset(expr, 0, N);

system("cls");

printf("\n Введите выражение с полиномами\n\n");

printf(" - Для обозначение порядка действий допускается использование скобок: ()\n");

printf(" - Допустимое обозначение переменной: x\n");

printf(" - Допустимые действия: + - \* /\n");

printf(" - Допустимное обозначение степени переменной: ^\n");

printf(" - Максимальное количество считываемых символов: %d\n\n ", N);

gets\_s(expr, N);

correct = check\_expression();

}

read\_expression();

system("cls");

printf(" Введённое выражение:\n\n ");

output\_expression(res[0], res\_cnt[0]);

printf("\n\n");

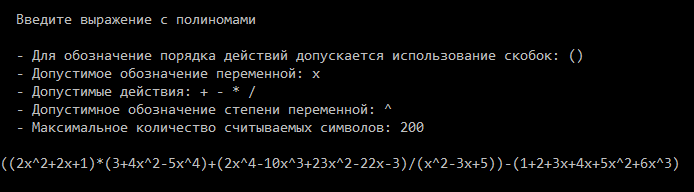
system("pause");

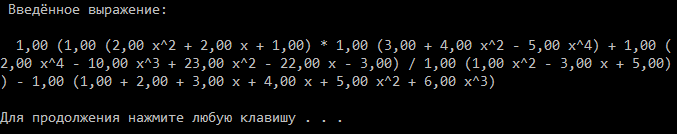
printf("\n\n");

proc\_expression();

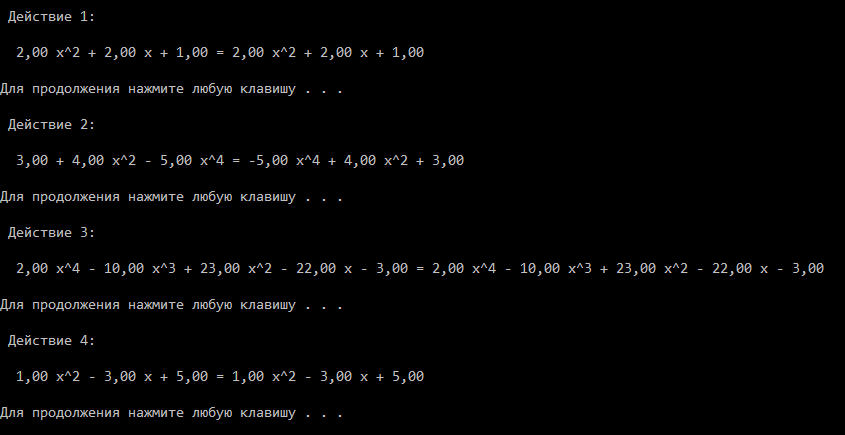
}

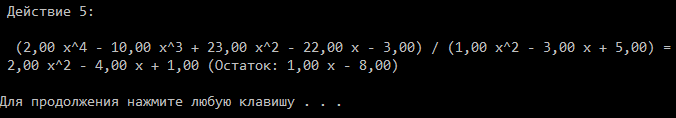
# 6. Тестовые примеры

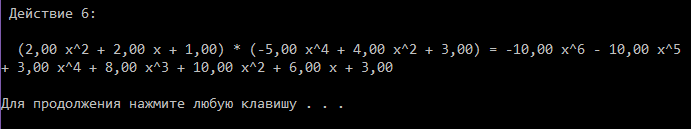
*Рисунок 10 – Ввод выражения*

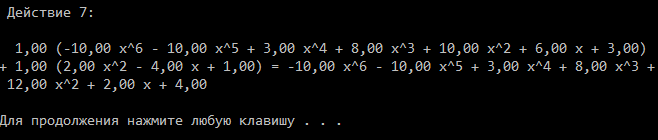


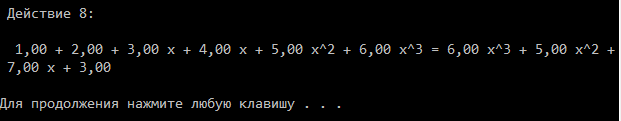
*Рисунок 11 – Интерпретация выражения*

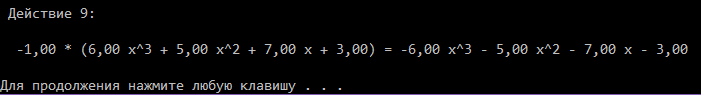
*Рисунок 12 – Действия в скобках (упорядочивание)*

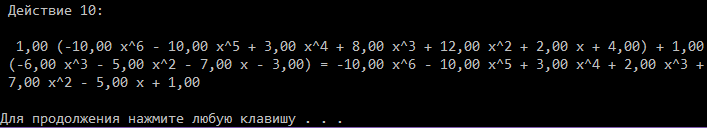
*Рисунок 13 – Деление двух полиномов*

*Рисунок 14 – Умножение двух полиномов*

*Рисунок 15 – Сложение двух полиномов*

 *Рисунок 16 – Действие в скобках (упорядочивание и сложение)*

*Рисунок 17 – Умножение скобки на коэффициент*

*Рисунок 18 – Сложение двух полиномов (последнее действие)*

# 7. Инструкция по применению

При запуске программы предлагается ввести выражение с полиномами. На экран выводятся подсказки о допустимых к вводу символах.

Чтобы произвести действия между полиномами, необходимо ввести сами полиномы в скобках, а между ними – знак действия: [+], [–], [\*], [/]. Вводимые полиномы состоят из переменных, перед которыми вводятся коэффициенты, а после, через знак степени [^] – степень коэффициента. Допускается ввод целочисленных или дробных коэффициентов (дробная часть вводится через знак [.] (точка)). Также допускается ввод коэффициента перед скобками. Нет ограничений на глубину вложения скобок. Длина вводимой строки ограничивается константой N (по умолчанию – 200). Для корректировки введённого выражения можно использовать клавиши [Backspace] для затирания символов и [←], [→] – для перемещения курсора.

После того, как необходимое выражение введено, необходимо нажать [Enter], чтобы перейти к дальнейшему выполнению действий над ним. В случае, если обнаружится неправильный ввод скобок (несоответствие открывающих и закрывающих скобок в выражении), неправильный ввод степеней (дробная или нечисловая степень), неправильная расстановка знаков (отсутствие знака между скобками, дублирование знаков друг за другом, отсутствие переменной после/перед знака/ом) или недопустимый символ, на экран будет выведено соответствующее сообщение, и ввод будет запрошен сначала.

В случае если введённое выражение корректно, произойдёт вывод введённого выражения на экран в интерпретируемой программой форме и дальнейшее выполнение действий. Каждым отдельным действием считаются действия внутри скобок, а также действия между двумя выражениями в скобках или действия между выражением в скобках и одиночной переменной. Для перехода к каждому последующему действию необходимо нажимать клавишу [Enter].

Программа завершается после выполнения последнего действия (исходное выражение со всеми раскрытыми скобками).

# 8. Спецификация программы

|  |  |  |
| --- | --- | --- |
| main.c | Исполняемый модуль |  |
| stdio.h | Библиотека |  |
| conio.h | Библиотека |  |
| Windows.h | Библиотека |  |
| locale.h | Библиотека |  |

# Заключение

Программа «Калькулятор для полиномов» производит вычисления между полиномами, обрабатывая введённое с клавиатуры выражение со скобками. Учитывается обработка действий – сложения, вычитания, умножения и деления – между двумя выражениями в скобках, а также между выражением в скобках и одиночной переменной и между двумя одиночными переменными. Также происходит сортировка выражений в скобках, сложение переменных с одинаковыми степенями и умножение выражения в скобках на численный коэффициент.

# 9. Список использованной литературы

1. Керниган Б., Ритчи Д. – Язык программирования Си. Учебное пособие. – 3-е издание, испр. СПБ.: “Невский Диалект”, 2001г.

2. Васильев А. Н. – Программирование на С в примерах и задачах. – Москва: Издательство «Э», 2017г.

3. Единая система программной документации. Гост 19.701-90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правил выполнения.