Цель работы.

Получение практических навыков работы со стеками и очередями; изучение обратной и прямой польской нотации; проведение сравнительного анализа этих структур данных.

Основные теоретические положения.

Стек – это частный случай однонаправленного списка, добавление элементов в который и выборка из которого выполняется с одного конца, называемого вершиной стека. Другие операции со стеком не определены. При выборке элемент исключается из стека. Говорят, что стек реализует принцип обслуживания LIFO (последним пришел – первым ушел).

Наверное, последнее может быть непонятно, поэтому поясню на цифрах.

Вводим: 1,2,3,4,5

На выходе 5,4,3,2,1

Стек — это такой линейный список, в котором и добавление новых и удаление существующих элементов возможно только с головного элемента.

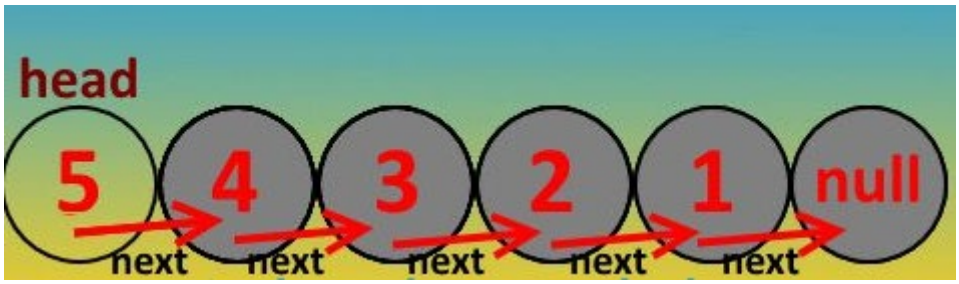

Рисунок 1.

При каждом новом добавлении элемента в стек мы должны запомнить адрес текущего элемента, после чего сместить голову, и указателем указать на тот запомненный адрес.

Для начала нам нужно создать структуру, которая будет являться нашей «ячейкой»

```
struct test { //Структура с названием test
	int Data; //Какие-то данные
	test *next;//Указатель типа test на следующий элемент
};
```

После того как у нас задана «Ячейка», перейдем к созданию функций.

Функция создания «Стека»/добавления элемента в «Стек»

При добавлении элемента у нас возникнет две ситуации:

1. Стек пуст, и нужно создать его

2. Стек уже есть и нужно лишь добавить в него новый элемент

```
void push(test **top, int D) { //функция которая принимает указатль на
вершину стека и переменную которая будет записываться в ячейку
	test *q; //Создаем новый указатель q типа структуры test. По сути это и
есть наш новый элемент
	q = new comp(); //выделяем память для нового элемента
	q->Data = D; //Записываем необходимое число  в Data элемента
	if (top == NULL) { //Если вершины нет, то есть стек пустой
		*top = q; //вершиной стека будет новый элемент
	}
	else //если стек не пустой
	{
		q->next = *top; //Проводим связь от нового элемента, к вершине.
		*top = q; //Обозначаем, что вершиной теперь является новый элемент
	}
}
```

Основные операции над стеками:

1) чтение верхнего элемента;

2) добавление нового элемента;

3) удаление существующего элемента.

Очередь — это структура данных (как было сказано выше), которая построена по принципу LILO (last in — last out: последним пришел — последним вышел). В C++ уже есть готовый STL контейнер — queue.

В очереди, если вы добавите элемент, который вошел самый первый, то он выйдет тоже самым первым. Получается, если вы добавите 4 элемента, то первый добавленный элемент выйдет первым.

Чтобы понять принцип работы очереди вы можете представить себе магазинную очередь. И вы стоите посреди нее, чтобы вы оказались напротив кассы, сначала понадобится всех впереди стоящих людей обслужить. А вот для последнего человека в очереди нужно, чтобы кассир обслужил всех людей кроме него самого.

Если для стека в момент добавления или удаления элемента допустимо задействование лишь его вершины, то касательно очереди эти две операции должны быть применены так, как это регламентировано в определении этой структуры данных, т. е. добавление – в конец, удаление – из начала.

Выделяют три способа программной реализации очереди. Первый основан на базе массива (возможна реализация на базе динамического массива), второй – на базе указателей (связного списка). Третьей с помощью класса объектно-ориентированного программирования. Первый способ – статический, так как очередь представляется в виде простого статического массива, второй – динамический.

Очередь в программировании используется, как и в реальной жизни, когда нужно совершить какие-то действия в порядке их поступления, выполнив их последовательно. Примером может служить организация событий в Windows. Когда пользователь оказывает какое-то действие на приложение, то в приложении не вызывается соответствующая процедура (ведь в этот момент приложение может совершать другие действия), а ему присылается сообщение, содержащее информацию о совершенном действии, это

сообщение ставится в очередь, и только когда будут обработаны сообщения, пришедшие ранее, приложение выполнит необходимое действие.

Постановка задачи.

Необходимо написать программу, которая выполняет следующее:

1. Реализует преобразование введенного выражения (если используются переменные, то пользователь должен их инициализировать). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и * с файла.

2. Реализует вычисления простого выражения и выражения, записанного в прямой и обратной польских нотациях (на выбор пользователя). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и с файла.

3. Генерирует несколько (на выбор пользователя) вариантов проверочной работы по польской нотации (обратной).

Выполнение работы.

Основное меню программы.

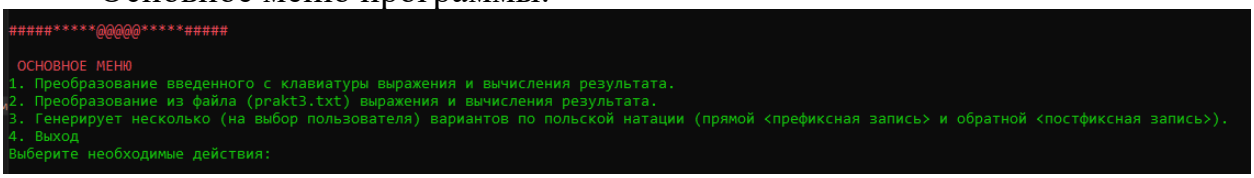

Рисунок 1. Основное меню.

Реализует преобразование введенного выражения (если используются переменные, то пользователь должен их инициализировать). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и * с файла.

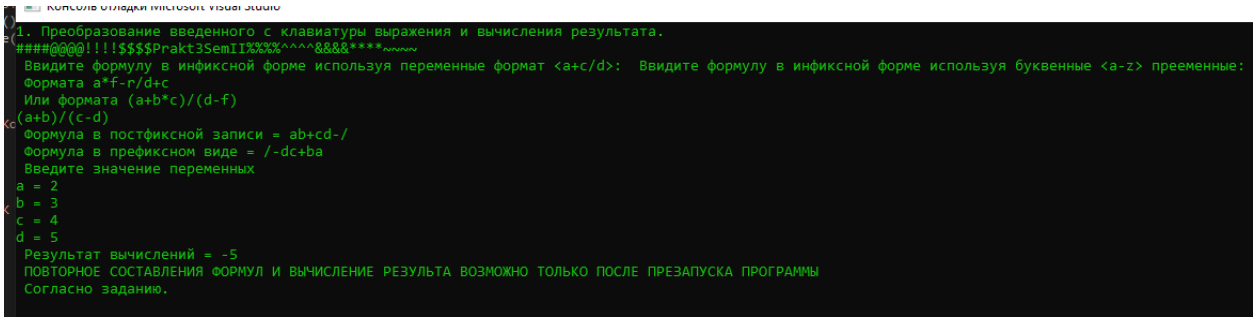

Рисунок 2. Ввод выражения с клавиатуры.

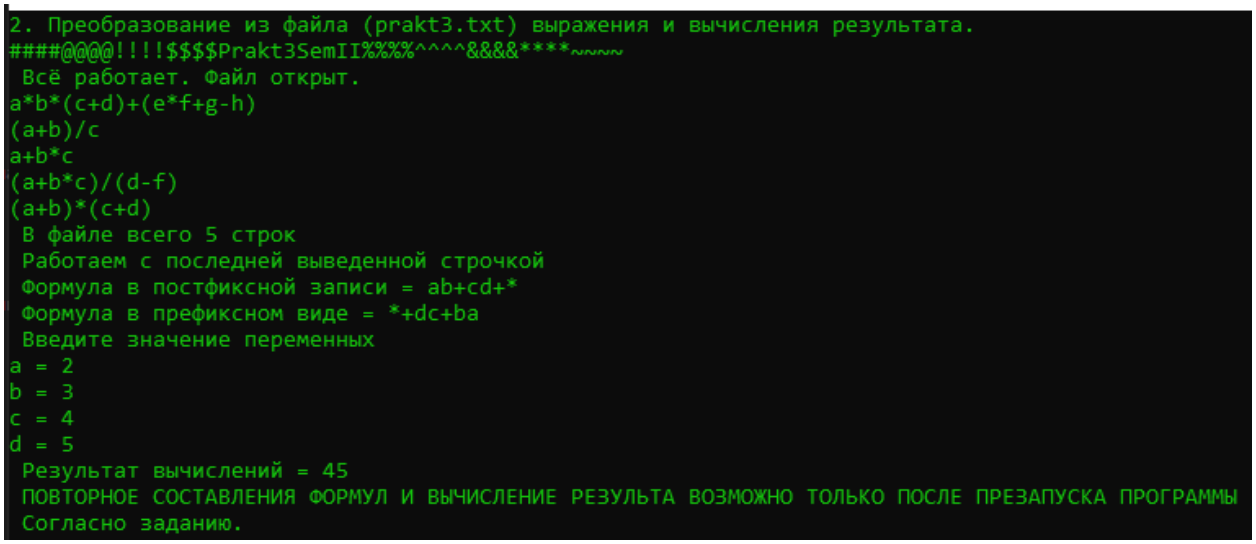

Рисунок 3. Вывод выражения из файла.

2. Реализует вычисления простого выражения и выражения, записанного в прямой и обратной польских нотациях (на выбор пользователя). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и с файла.

Второе задание объединено с первым не вижу смысла делать скрины так как они представлены выше.

3. Генерирует несколько (на выбор пользователя) вариантов проверочной работы по польской нотации (обратной).

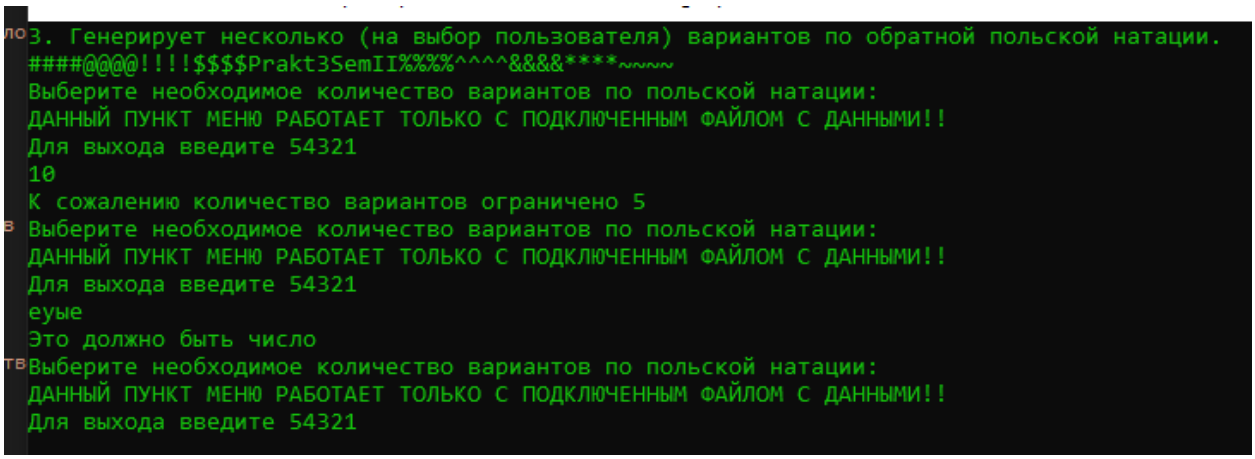

Рисунок 4. Предварительное меню.

Третье задание требует наличие подключенного к программе файла поэтому имеет смысл оговорить дополнительные условия.

Результат работы программы выводится в файл out.txt пример файла есть https://github.com/Aleksey-app/1-Kurs-2-term в разделе 3 Практическая работа C++.

Выводы.

Стек и Очередь оба являются не примитивными структурами данных. Основные различия между стеком и очередью заключаются в том, что в стеке используется метод LIFO (последний пришел первым вышел) для доступа и добавления элементов данных, тогда как в очереди используется метод FIFO (первый пришел первым вышел) для доступа и добавления элементов данных. С другой стороны, в стеке открыт только один конец для перемещения и извлечения элементов данных. В очереди имеются открытые оба конца для постановки в очередь и удаления из очереди элементов данных. Стек и очередь — это структуры данных, используемые для хранения элементов данных, и фактически они основаны на каком-то реальном эквиваленте. Например, стопка — это стопка компакт-дисков, где

вы можете извлечь и вставить компакт-диск через верх стопки компакт-дисков. Аналогично, очередь представляет собой очередь для билетов в театр, где человек, стоящий на первом месте, т.е. Сначала будет обслужен фронт очереди, а прибывающий новый человек появится в задней части очереди (задний конец очереди).

ПРИЛОЖЕНИЕ А

ПОЛНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <string> //отличная библиотека работы со строками
#include <windows.h> // разобрался это библиотека подключает функционал ОС Windows
#include <fstream> // читать и писать файлы
using namespace std;

struct stck // структура стека на основе односвязного списка
{
	char y;
	stck *next;
} *sip;
int menu(int xz) {
	int qwe = 0;
	HANDLE O = GetStdHandle(STD_OUTPUT_HANDLE);
	SetConsoleTextAttribute(O, FOREGROUND_RED | FOREGROUND_INTENSITY);
	cout << "\n##########*****@@@@@*****#####\n";
	cout << "\n ОСНОВНОЕ МЕНЮ\n";
	SetConsoleTextAttribute(O, FOREGROUND_GREEN | FOREGROUND_INTENSITY);
	cout << "1. Преобразование введенного с клавиатуры выражения и вычисления результата.\n";
	cout << "2. Преобразование из файла (prakt3.txt) выражения и вычисления результата.\n";
	cout << "3. Генерирует несколько (на выбор пользователя) вариантов по польской натации
(прямой <префиксная запись> и обратной <постфиксная запись>).\n";
	cout << "4. Выход \n";
	while (true)
	{
		cout << "Выберите необходимые действия:" << endl;
		cin >> qwe;
		if (cin.fail())
		{
			cout << "Это должно быть число" << endl;
			cin.clear();
			cin.ignore(32767, '\n');
			continue;
		}
		if (qwe <= 0)
		{
			cout << " Значение должно быть положительным" << endl;
			continue;
		}
		if (qwe > 4)
		{
			cout << " Возможное значение <В диапозоне от 1-4>" << endl;
			continue;
		}
		else break;
	}
	system("cls");
	return qwe;
}
int znac(char a) // приоритет операций
{
	switch (a)
	{
	case '*': case '/': return 3;
	case '-': case '+': return 2;
	case '(':return 1;
	}
	return 0;
```

```
}
stck *inSpi(stck *q, char out) // входящий стек лексем
{
	stck *z = new stck;
	z->y = out;
	z->next = q;
	return z;
}
stck *outSpi(stck *q, char& x)
{
	stck *z = q;
	x = q->y;
	q = q->next;
	delete z;
	return q;
}
double shet(char *str) // имитация стека опрераций
{
	int i;
	char ss, ss1, ss2, ss3 = 'z' + 1;
	double aa1, aa2, res = 0, mas[200];
	cout << " Введите значение переменных" << endl;
	for (i = 0; str[i] != '\0'; ++i)
	{
		ss = str[i];
		if (ss >= 'a' && ss <= 'z')
		{
			while (str[i] != '\0')
			{
				cout << ss << " = ";
				cin >> mas[int(ss)];
				if (cin.fail())
				{
					cout << "Это должно быть число" << endl;
					cout << "Ввидите числовое значение <в диапозоне от -543 до
345>" << endl;
					cin.clear();
					cin.ignore(32767, '\n');
					continue;
				}
				if (mas[int(ss)] > 345 || mas[int(ss)] < -543)
				{
					cout << "Ввидите числовое значение <в диапозоне от -543 до
345>" << endl;
					cin.clear();
					cin.ignore(32767, '\n');
					continue;
				}
				else break;
			}
		}
	}
	for (i = 0; str[i] != '\0'; ++i)
	{
		ss = str[i];
		if (!(ss == '+' || ss == '-' || ss == '*' || ss == '/'))
		{
			sip = inSpi(sip, ss);
		}
		else
		{
			sip = outSpi(sip, ss2);
			sip = outSpi(sip, ss1);
			aa2 = mas[int(ss2)];
			aa1 = mas[int(ss1)];
			switch (ss)
			{
			case '+':res = aa1 + aa2; break;
			case '-':res = aa1 - aa2; break;
			case '*':res = aa1 * aa2; break;
			case '/':res = aa1 / aa2; break;
			}
			mas[int(ss3)] = res;
			sip = inSpi(sip, ss3);
			ss3++;
		}
	}
	return res;
}
```

```
int shetSt(int strok) //функция подсчёта строк
{
	ifstream file("prakt3.txt");
	if (!file)
	{
		cout << " Error!!! \n";
	}
	else
	{
		while (true)
		{
			string v;
			getline(file, v);
			if (!file.eof())
				strok++;
			else
				break;
		}
		cin.get();
	}
	file.close();
	return strok;
}

char proverka()
{
	char zu;
	cin >> zu;
	if (zu >= 'a'&& zu <= 'z')
	{
		return zu;
		return proverka();
	}
	if (zu == '(' || zu == ')')
	{
		return zu;
	}
	if (zu == '+' || zu == '-' || zu == '*' || zu == '/')
	{
		return zu, proverka();
	}
	else
	{
		cout << "Ошибка ввода!" << endl;
		return 0;
	}
	return '\0';
}
int main()
{
	int xz = 0;
	setlocale(LC_ALL, "Russian");
	int lab = menu(xz);
	if (lab == 1) {
		cout << "1. Преобразование введенного с клавиатуры выражения и вычисления
результата.\n";
		cout << "####@@@@!!!!$$$$Prakt3SemII%%%%^^^^&&&&****~~~~" << endl;
		cout << " Ввидите формулу в инфиксной форме используя переменные формат <a+c/d>:
";
		cout << " Ввидите формулу в инфиксной форме используя буквенные <a-z> пременные:
" << endl;
		cout << " Формата a*f-r/d+c" << endl;
		cout << " Или формата (a+b*c)/(d-f)" << endl;
		stck *zu, *tmp = NULL;
		char a;
		char in[41], out[41];
		int ind = 0, i = 0;
		cin >> in;
		while (in[ind] != '\0')
		{
			if (in[ind] >= 'a'&& in[ind] <= 'z')
			{
				out[i++] = in[ind];
			}
			if (in[ind] == '(')
			{
				tmp = inSpi(tmp, in[ind]);
			}
			if (in[ind] == ')')
```

```
                        {
                                while ((tmp->y) != '(')
                                {
                                        tmp = outSpi(tmp, a);
                                        out[i++] = a;
                                        if (!tmp)
                                        {
                                                a = '\0';
                                                out[i++] = a;
                                        }
                                }
                                zu = tmp;
                                tmp = tmp->next;
                                delete zu;
                        }
                        if (in[ind] == '+' || in[ind] == '-' || in[ind] == '*' || in[ind] == '/')
                        {
                                while (tmp != NULL && znac(tmp->y) >= znac(in[ind]))
                                {
                                        tmp = outSpi(tmp, a);
                                        out[i++] = a;
                                }
                                tmp = inSpi(tmp, in[ind]);
                        }
                        ind++;
                }
                while (tmp != NULL)
                {
                        tmp = outSpi(tmp, a);
                        out[i++] = a;
                }
                out[i] = '\0';
                cout << " Формула в постфиксной записи = " << out << endl;
                string prefix = out;
                reverse(prefix.begin(), prefix.end());
                cout << " Формула в префиксном виде = " << prefix << endl;
                cout << " Результат вычислений = " << shet(out) << endl;
                cout << " ПОВТОРНОЕ СОСТАВЛЕНИЯ ФОРМУЛ И ВЫЧИСЛЕНИЕ РЕЗУЛЬТА ВОЗМОЖНО ТОЛЬКО ПОСЛЕ
ПРЕЗАПУСКА ПРОГРАММЫ" << endl;
                cout << " Согласно заданию." << endl;
                return 0;
        }
        if (lab == 2) {
                cout << "2. Преобразование из файла (prakt3.txt) выражения и вычисления
результата.\n";
                cout << "####@@@@!!!!$$$$Prakt3SemII%%%%^^^^&&&&****~~~~" << endl;
                int statist = 0, wid = 1;
                statist = shetSt(wid);
                char in[41];
                ifstream run;
                run.open("prakt3.txt");
                if (!run)
                {
                        cout << " Ошибка файл не загружен или отсутствует." << endl;
                        return 0;
                }
                else
                {
                        cout << " Всё работает. Файл открыт." << endl;
                }
                string kid;
                while (!run.eof())
                {
                        kid = "";
                        getline(run, kid);
                        cout << kid << endl;
                }
                int ux = 0;
                for (int i = 0; i < kid.length(); i++)
                {
                        in[i] = kid[ux++];
                }
                cout << " В файле всего " << statist << " строк " << endl;
                cout << " Работаем с последней выведенной строчкой" << endl;
                run.close();
                stck *ruu, *tmp = NULL;
                char a;
                char out[41];
                int  i = 0, ind = 0;
```

```
		while (in[ind] != '\0')
		{
			if (in[ind] >= 'a'&& in[ind] <= 'z')
			{
				out[i++] = in[ind];
			}
			if (in[ind] == '(')
			{
				tmp = inSpi(tmp, in[ind]);
			}
			if (in[ind] == ')')
			{
				while ((tmp->y) != '(')
				{
					tmp = outSpi(tmp, a);
					out[i++] = a;
					if (!tmp)
					{
						a = '\0';
						out[i++] = a;
					}
				}
				ruu = tmp;
				tmp = tmp->next;
				delete ruu;
			}
			if (in[ind] == '+' || in[ind] == '-' || in[ind] == '*' || in[ind] == '/')
			{
				while (tmp != NULL && znac(tmp->y) >= znac(in[ind]))
				{
					tmp = outSpi(tmp, a);
					out[i++] = a;
				}
				tmp = inSpi(tmp, in[ind]);
			}
			ind++;
		}
		while (tmp != NULL)
		{
			tmp = outSpi(tmp, a);
			out[i++] = a;
		}
		out[i] = '\0';
		cout << " Формула в постфиксной записи = " << out << endl;
		string prefix = out;
		reverse(prefix.begin(), prefix.end());
		cout << " Формула в префиксном виде = " << prefix << endl;
		cout << " Результат вычислений = " << shet(out) << endl;
		cout << " ПОВТОРНОЕ СОСТАВЛЕНИЯ ФОРМУЛ И ВЫЧИСЛЕНИЕ РЕЗУЛЬТА ВОЗМОЖНО ТОЛЬКО ПОСЛЕ
ПРЕЗАПУСКА ПРОГРАММЫ" << endl;
		cout << " Согласно заданию." << endl;
		return 0;
	}
	if (lab == 3) {
		cout << "3. Генерирует несколько (на выбор пользователя) вариантов по обратной
польской натации.\n";
		cout << "####@@@@!!!!$$$$Prakt3SemII%%%%^^^^&&&&****~~~~" << endl;
		char in[41], out[41];
		int y, strok = 1, t = 0;
		t = shetSt(strok);
		while (true)
		{
			cout << "Выберите необходимое количество вариантов по польской натации:" <<
endl;
			cout << "ДАННЫЙ ПУНКТ МЕНЮ РАБОТАЕТ ТОЛЬКО С ПОДКЛЮЧЕННЫМ ФАЙЛОМ С
ДАННЫМИ!!" << endl;
			cout << "Для выхода введите 54321" << endl;
			cin >> y;
			if (y == 54321)
			{
				cin.clear();
				cin.ignore(32767, '\n');
				break;
			}
			if (cin.fail())
			{
				cout << "Это должно быть число" << endl;
				cin.clear();
				cin.ignore(32767, '\n');
```

```
                        continue;
                }
                if (y <= 0)
                {
                        cout << "Количество вариантов не может быть равен nullus – никакой
или быть отрицательным " << endl;
                        continue;
                }
                if (y > t)
                {
                        cout << "К сожалению количество вариантов ограничено " << t <<
endl;
                        continue;
                }
                else break;
        }
        system("cls");
        ifstream s;
        s.open("prakt3.txt");
        if (!s)
        {
                cout << " Ошибка файл не загружен или отсутствует." << endl;
                return 0;
        }
        else
        {
                cout << " Всё работает. Файл открыт." << endl;
        }
        string tot;
        for (int i = 0; i < y; i++)
        {
                getline(s, tot);
                s.close();
                int ux = 0;
                for (int lop = 0; lop < tot.length(); lop++)
                {
                        in[lop] = tot[ux++];
                }
                stck *malyn, *tmp = NULL;
                char a;
                int  pov = 0, ind = 0;
                while (in[ind] != '\0')
                {
                        if (in[ind] >= 'a'&& in[ind] <= 'z')
                        {
                                out[pov++] = in[ind];
                        }
                        if (in[ind] == '(')
                        {
                                tmp = inSpi(tmp, in[ind]);
                        }
                        if (in[ind] == ')')
                        {
                                while ((tmp->y) != '(')
                                {
                                        tmp = outSpi(tmp, a);
                                        out[pov++] = a;
                                        if (!tmp)
                                        {
                                                a = '\0';
                                                out[pov++] = a;
                                        }
                                }
                                malyn = tmp;
                                tmp = tmp->next;
                                delete malyn;
                        }
                if (in[ind] == '+' || in[ind] == '-' || in[ind] == '*' || in[ind] == '/')
                        {
                                while (tmp != NULL && znac(tmp->y) >= znac(in[ind]))
                                {
                                        tmp = outSpi(tmp, a);
                                        out[pov++] = a;
                                }
                                tmp = inSpi(tmp, in[ind]);
                        }
                        ind++;
                }
                while (tmp != NULL)
```

```
                        {
                                tmp = outSpi(tmp, a);
                                out[pov++] = a;
                        }
                        out[pov] = '\0';

                        ofstream d;
                        d.open("out.txt", ios::app);
                        if (!d)
                        {
                                cout << "Файл не открыт!!!\n";
                        }
                        else
                        {
                                cout << "Всё работает. Файл открыт в режиме дозаписи!\n";
                        }
                        d << "\n";
                        d << out;
                        d.close();
                }
                return 0;
        }
        if (lab == 4) {
                cout << "\n GEME OVER \n";
                return 0;
        }
        return 0;
}
```