**МИНОБРНАУКИ РФ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА) “ЛЭТИ”**

**ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ИНФОРМАТИКИ**

**КАФЕДРА САПР**

**ОТЧЁТ**

**по курсовой работе   
по дисциплине «Алгоритмы и структуры данных»**

Тема: «Преобразование алгебраических формул из инфиксной в префиксную форму записи и вычисление значения выражения»

**Вариант 2**

Студент гр.9302: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кнауб К.В.

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тутуева А.В.

Санкт-Петербург

2020

# Постановка задачи

Необходимо реализовать простейшую версию калькулятора. Пользователю должен быть доступен ввод математического выражения, состоящего из чисел и арифметических знаков. Программа должна выполнить проверку корректности введенного выражения. В случае некорректного ввода необходимо вывести сообщение об ошибке с указанием позиции некорректного ввода. В противном случае выводится польская нотация введенного выражения, а также отображается результат вычисления.

# 

# Обоснование выбора используемых структур данных

Данные структуры и методы позволяют оптимально решить поставленную задачу.

struct Node

|  |  |
| --- | --- |
| **Элементы** | **Описание** |
| Node \*next; | Указатель на следующий элемент списка |
| Node \*prev; | Указатель на предыдущий элемент списка |
| string content | Данные, хранящиеся в списке |

**class duaList**

|  |  |
| --- | --- |
| **Элементы** | **Описание** |
| Node \*head; | Указатель на начало списка |
| Node \*last; | Указатель на конец списка |
| size\_t length; | Переменная, хранящая размер списка |

|  |  |  |
| --- | --- | --- |
| **Метод** | **Описание** | **Оценка временной сложности** |
| void push\_back(int) | Добавление в конец списка | O(1) |
| void push\_front(int) | Добавление в начало списка | O(1) |
| void pop\_back() | Удаление последнего элемента | O(1) |
| void pop\_front() | Удаление первого элемента | O(1) |
| void insert(string, size\_t) | Добавление элемента по индексу | O(n) |
| string at(int) | Получение элемента по индексу | O(n) |
| void remove(size\_t) | Удаление элемента по индексу | O(n) |
| size\_t get\_size() | Получение размера списка | O(1) |
| void print\_to\_console() | Вывод элементов в консоль через разделитель | O(n) |
| void clear() | Удаление всех элементов списка | O(n) |
| void set(size\_t, int) | Замена элемента по индексу на передаваемый элемент | O(n) |
| bool isEmpty() | Проверка на пустоту списка | O(1) |
| void opposite() | Переворачивание списка | O(n) |

**class calculator**

|  |  |
| --- | --- |
| **Элемент** | **Описание** |
| duoList calculate; | Список компонентов выражения |
| duoList temp; | Временный список для всех компонентов выражения |

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| string res\_func(string); | Расчет функций одного аргумента |
| string get\_result(); | Расчёт выражения в префиксной нотации |
| string print\_prefix(); | Вывод префиксной нотации |
| void component(string); | Разбиение первоначального выражения на составляющие |
| void clear(); | Очистка всех компонентов |
| void clone(); | Клонирование информации |
| void get\_normal(); | Нормализация списка компонентов |
| void get\_prefix(); | Получение префиксной нотации |

# 

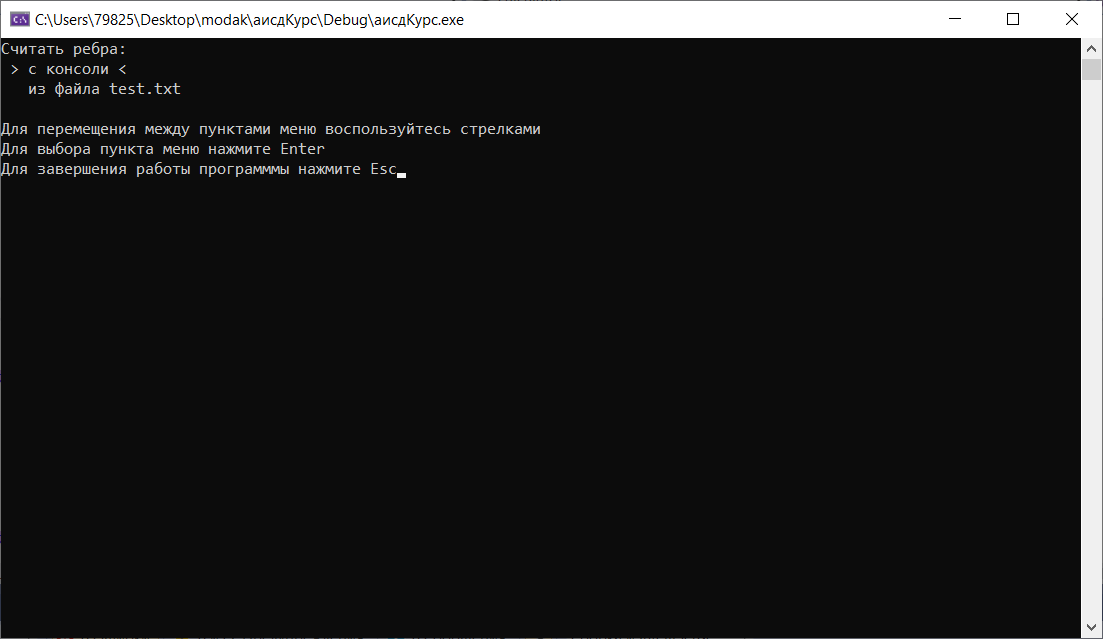
# Описание алгоритма решения

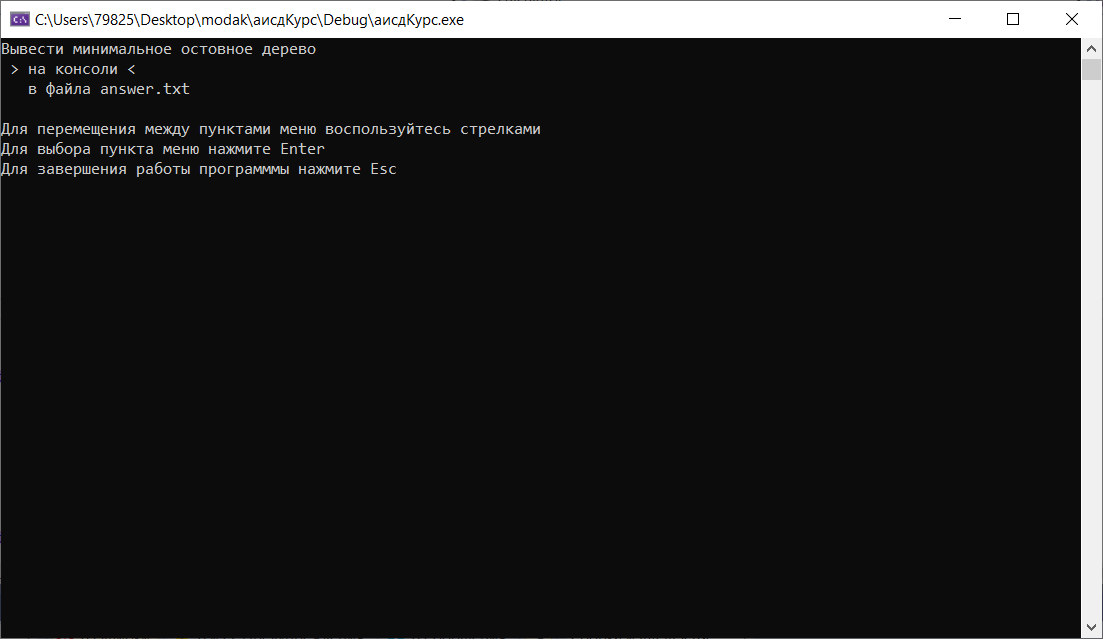
* 1. Выбираем откуда считать математическое выражение (консоль, файл).
  2. Выбираем куда выводить результат работы программы (консоль, файл).
  3. Считываем выражение.
  4. Разбиваем введенное выражение на составляющие.
  5. Обрабатываем унарный минус и функции.
  6. Приводим данное выражение в префиксный вид.
  7. Находим значение данного выражения

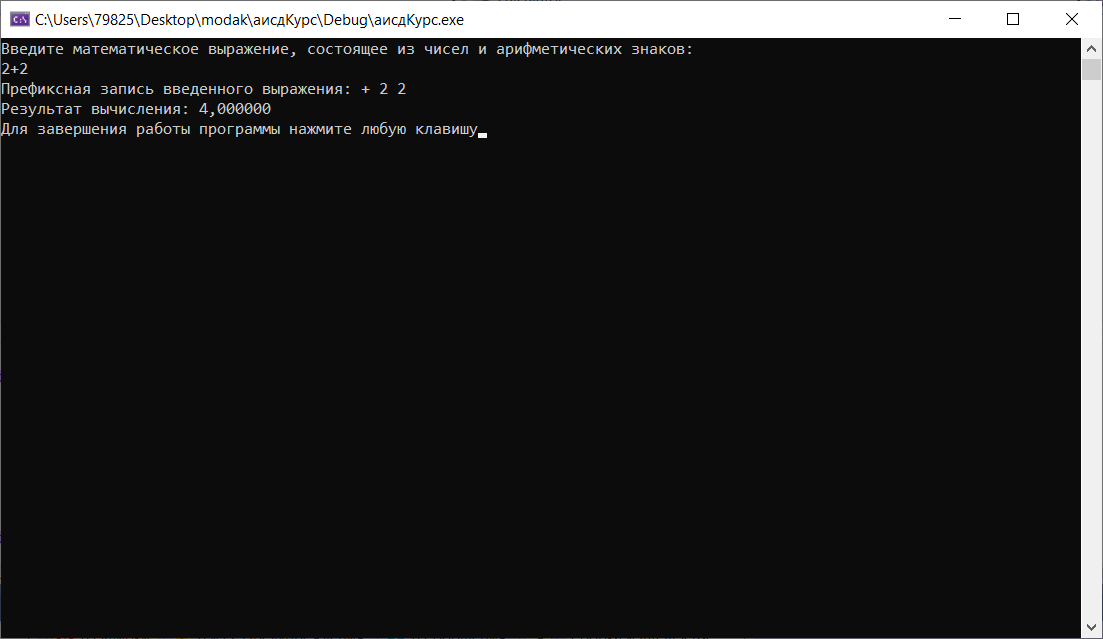
# 

# Пример работы программы

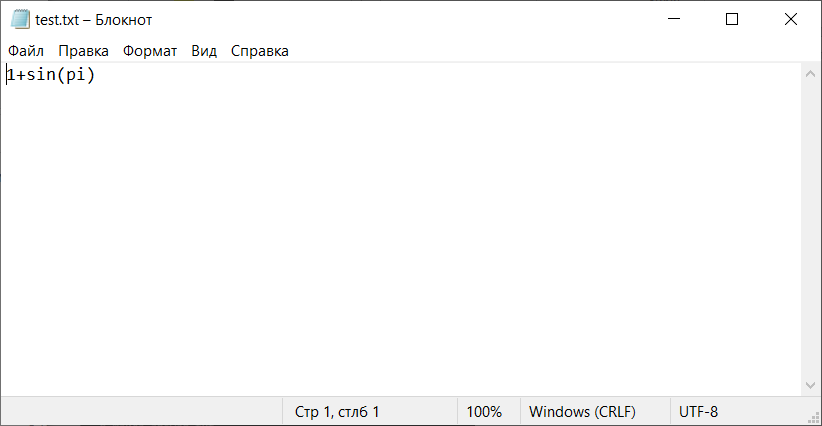
Пример 1

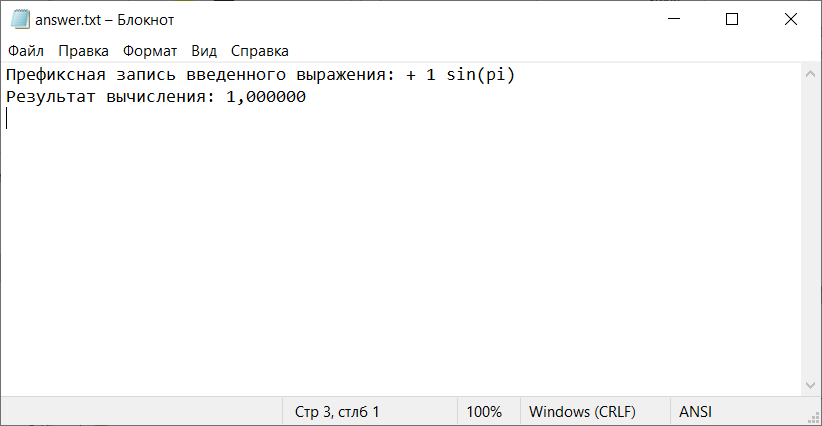




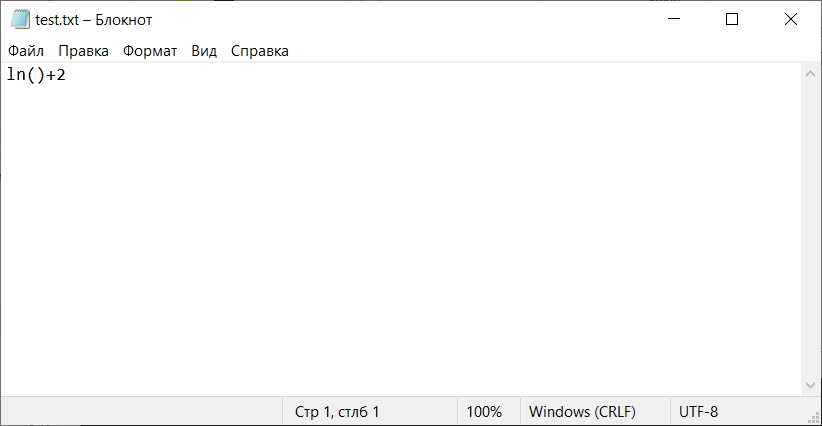


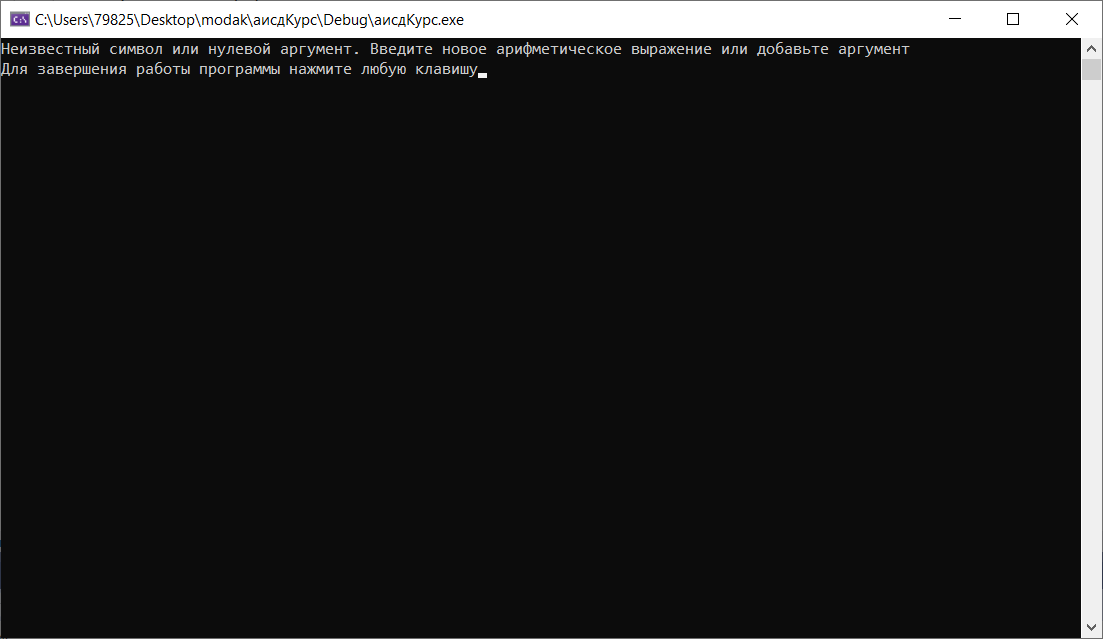
Пример 2





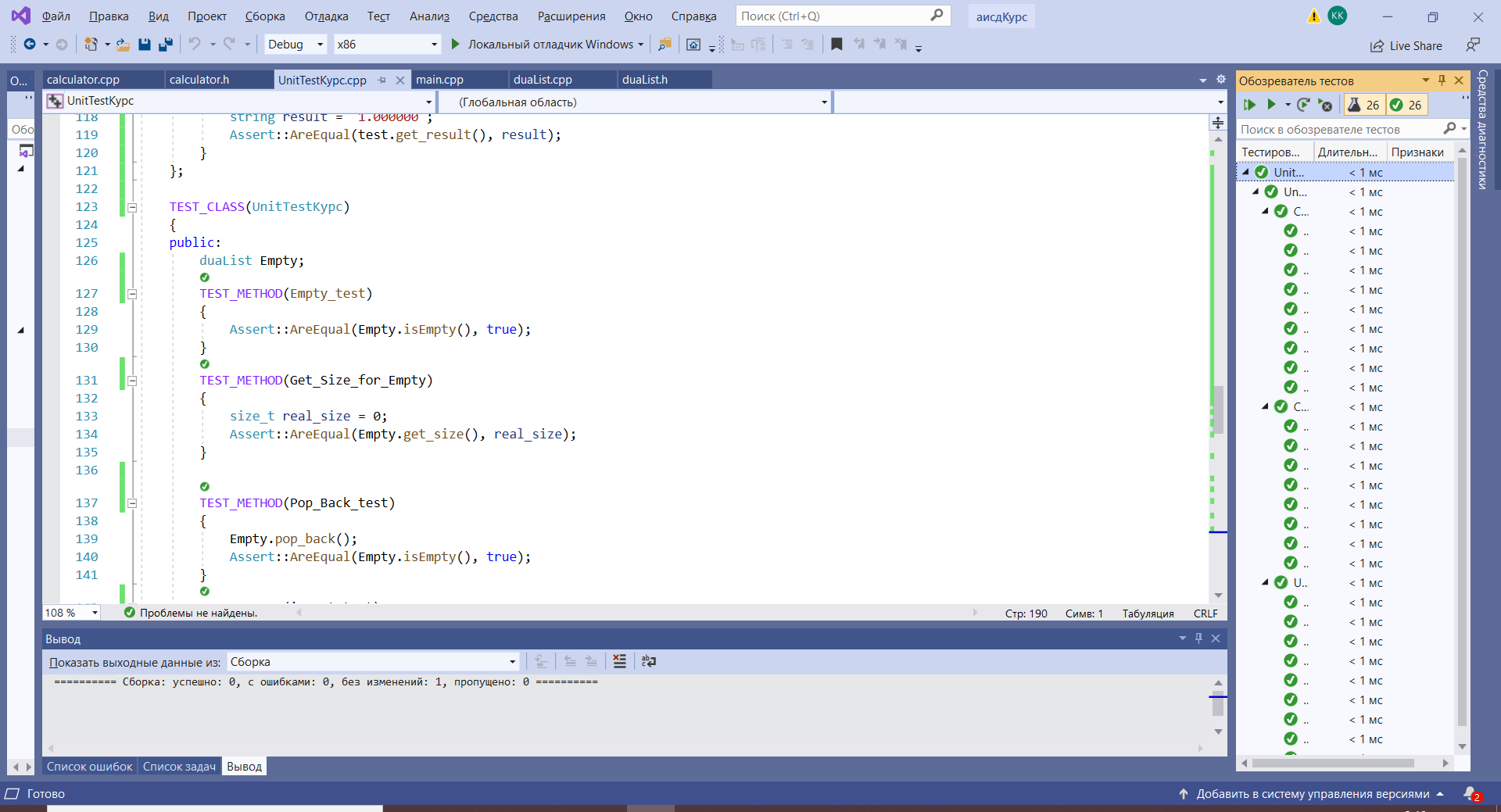
Пример 3





# Описание реализованных unit-тестов

|  |
| --- |
| **Название Unit-теста** |
| plus\_test |
| minus\_test |
| multiply\_test |
| divide\_test |
| trigan\_test |
| brackets\_test |
| mix\_test |
| unary\_minus\_test |
| unary\_sum\_test |
| sum\_test |
| difference\_test |
| composition\_test |
| division\_test |
| sum\_brackets\_test |
| trigonometry\_test |
| sqrt\_test |
| logarithm\_test |
| Empty\_test\_test |
| Get\_Size\_test |
| Pop\_Back\_test |
| insert\_test |
| remove\_test |
| Pop\_Front\_test |
| Clear\_test |
| Push\_Back\_test |
| Push\_Front\_test |



# Листинг

|  |
| --- |
| **main.cpp** |
| #include <iostream>  #include <string>  #include "calculator.h"  #include <conio.h>  #include <fstream>  using namespace std;  constexpr unsigned char Enter = 13, Esc = 27, Up = 72, Down = 80;  int main()  {  setlocale(LC\_ALL, "Rus");  unsigned char readType = 0, writeType = 0, ch = 0;  while (ch != Esc) {  std::cout << "Считать ребра:" << std::endl  << ' ' << (readType == 0 ? '>' : ' ') << " с консоли " << (readType == 0 ? '<' : ' ') << std::endl  << ' ' << (readType == 1 ? '>' : ' ') << " из файла test.txt " << (readType == 1 ? '<' : ' ') << std::endl << std::endl  << "Для перемещения между пунктами меню воспользуйтесь стрелками" << std::endl  << "Для выбора пункта меню нажмите Enter" << std::endl  << "Для завершения работы программмы нажмите Esc";  ch = \_getch();  if (ch == 224) {  ch = \_getch();  if (ch == Up || ch == Down)  readType = (readType == 0 ? 1 : 0);  }  if (ch == Enter) {  system("cls");  while (ch != Esc) {  std::cout << "Вывести минимальное остовное дерево" << std::endl  << ' ' << (writeType == 0 ? '>' : ' ') << " на консоли " << (writeType == 0 ? '<' : ' ') << std::endl  << ' ' << (writeType == 1 ? '>' : ' ') << " в файла answer.txt " << (writeType == 1 ? '<' : ' ') << std::endl << std::endl  << "Для перемещения между пунктами меню воспользуйтесь стрелками" << std::endl  << "Для выбора пункта меню нажмите Enter" << std::endl  << "Для завершения работы программмы нажмите Esc";  ch = \_getch();  if (ch == 224) {  ch = \_getch();  if (ch == Up || ch == Down)  writeType = (writeType == 0 ? 1 : 0);  }  if (ch == Enter) {  system("cls");  calculator Test;  string buf;  if (readType == 0) {  cout << "Введите математическое выражение, состоящее из чисел и арифметических знаков:" << endl;  getline(cin, buf);  }  else {  ifstream in("test.txt");  getline(in, buf);  in.close();  }  Test.make\_prefix(buf);  if (writeType == 0) {  try {  cout << "Префиксная запись введенного выражения: " << Test.print\_prefix() << endl;  cout << "Результат вычисления: " << Test.get\_result() << endl;  }  catch (const char\* error)  {  cout << error << endl;  }  }  else {  ofstream out("answer.txt");  try {  out << "Префиксная запись введенного выражения: " << Test.print\_prefix() << endl;  out << "Результат вычисления: " << Test.get\_result() << endl;  }  catch (const char\* error)  {  out << error << endl;  }  out.close();  }  std::cout << "Для завершения работы программы нажмите любую клавишу";  \_getch();  ch = Esc;  }  system("cls");  }  }  system("cls");  }  return 0;  } |
| **dualList.h** |
| #pragma once  #include <cstddef>  #include <iostream>  #include <string>  using namespace std;  struct Node  {  string data;  Node\* prev = NULL;  Node\* next = NULL;  };  class duaList  {  public:  duaList();  ~duaList();  void push\_back(string);  void push\_front(string);  void pop\_back();  void pop\_front();  void insert(string, size\_t);  string at(size\_t);  void remove(size\_t);  size\_t get\_size();  void print\_to\_console();  void clear();  void set(size\_t index, string content);  bool isEmpty();  string top();  void opposite();    private:  Node\* head;  Node\* last;  size\_t size;  }; |
| **dualList.cpp** |
| #pragma once  #include "duaList.h"  duaList::duaList()  {  head = NULL;  last = NULL;  size = 0;  }  duaList::~duaList()  {  while (head)  {  last = head->next;  delete head;  head = last;  }  }  void duaList::push\_back(string content)  {  Node\* temp = new Node;  temp->next = NULL;  temp->data = content;  if (!this->isEmpty())  {  temp->prev = last;  last->next = temp;  last = temp;  }  else {  temp->prev = NULL;  head = last = temp;  }  size++;  }  void duaList::push\_front(string content)  {  Node\* temp = new Node;  temp->prev = NULL;  temp->data = content;  if (!this->isEmpty())  {  temp->next = head;  head->prev = temp;  head = temp;  }  else  {  temp->prev = NULL;  temp->next = NULL;  head = last = temp;  }  size++;  }  void duaList::pop\_back()  {  if (size == 0)  return;  if (size == 1)  {  delete last;  last = head = NULL;  size--;  return;  }  delete last->next;  last = last->prev;  last->next = NULL;  size--;  }  void duaList::pop\_front()  {  if (size == 0)  return;  if (size == 1)  {  delete head;  head = last = NULL;  size--;  return;  }  delete head->prev;  head->prev = NULL;  head = head->next;  size--;  }  void duaList::insert(string content, size\_t index)  {  if ((size == 0 && index == 0) || index >= size)  throw "Incorrect index.";  if (index == 0)  {  this->push\_front(content);  return;  }  Node\* cursor = head;  for (size\_t i = 0; i < index; i++)  cursor = cursor->next;  Node\* temp = new Node;  temp->data = content;  temp->next = cursor;  temp->prev = cursor->prev;  cursor->prev = temp;  temp->prev->next = temp;  size++;  }  string duaList::at(size\_t index)  {  if (index >= size || index < 0)  throw "Incorrect index.";  Node\* cursor = head;  for (size\_t i = 0; i < index; i++)  cursor = cursor->next;  return cursor->data;  }  void duaList::remove(size\_t index)  {  if (index >= size || index < 0)  throw "Incorrect index.";  if (index == 0)  {  this->pop\_front();  return;  }  if (index == size - 1)  {  this->pop\_back();  return;  }  Node\* cursor = head;  for (size\_t i = 0; i < index; i++)  cursor = cursor->next;  Node\* temp = cursor;  cursor->prev->next = cursor->next;  cursor->next->prev = cursor->prev;  delete temp;  size--;  }  size\_t duaList::get\_size()  {  return size;  }  void duaList::print\_to\_console()  {  Node\* cursor = head;  if (head)  {  while (cursor->next)  {  std::cout << cursor->data << " ";  cursor = cursor->next;  }  std::cout << cursor->data;  std::cout << std::endl;  }  else throw "List is empty.";  }  void duaList::clear()  {  while (head != NULL)  this->pop\_front();  head = last = NULL;  }  void duaList::set(size\_t index, string content)  {  if (index >= size || index < 0)  throw"Incorrect index.";  Node\* cursor = head;  for (size\_t i = 0; i < index; i++)  cursor = cursor->next;  cursor->data = content;  }  bool duaList::isEmpty()  {  if (head == NULL) return true;  return false;  }  void duaList::opposite()  {  if (!head || !head->next)  return;  last = head;  Node\* temp = NULL;  Node\* cur = head;  while (cur != NULL)  {  temp = cur->prev;  cur->prev = cur->next;  cur->next = temp;  cur = cur->prev;  }  head = temp->prev;  }  string duaList::top()  {  if (head)  return head->data;  else throw "is Empty";  } |
| **calculator.h** |
| #pragma once  #include "duaList.h"  #include <string>  #include <math.h>  class calculator {  duaList calculate;  duaList temp;  void get\_normal();  void component(string);  void get\_prefix();  friend string make\_res(string input);  void clone();  void clear();  public:  void make\_prefix(string input);  string print\_prefix();  string get\_result();  };  #define pi 3.1415926535897932384626433832795  #define e 2.7182818284  using namespace std;  string make\_res(string);  string make\_operator(string, string, string); |
| **calculator.cpp** |
| #include "calculator.h"  #include "math.h"  #include <iostream>  char Const(string oper, size\_t index = 0)  {  if (oper[index] == 'p' && oper[index + 1] == 'i')  return 2;  else if (oper[index] == 'e') //e  return 1;  return 0;  }  bool Number(string str)  {  if (str[0] >= '0' && str[0] <= '9')  return true;  if (str[0] == '-' && str[1] >= '0' && str[1] <= '9')  return true;  if (Const(str))  return true;  return false;  }  bool Operator(string oper, size\_t index = 0, size\_t ctrl = 0) // + - \* / ^  {  if (ctrl == 0 && oper[1] >= '0' && oper[1] <= '9')  return false;  if (oper[index] == '+' || oper[index] == '-' || oper[index] == '\*' || oper[index] == '/' || oper[index] == '^')  return true;  return false;  }  char Func(string oper, size\_t index = 0)  {  if (oper[index] == 't' && oper[index + 1] == 'g') return 2;  else if (oper[index] == 'l' && oper[index + 1] == 'n') return 2;  else if (oper[index] == 'c' && oper[index + 1] == 'o' && oper[index + 2] == 's') return 3;  else if (oper[index] == 'c' && oper[index + 1] == 't' && oper[index + 2] == 'g') return 3;  else if (oper[index] == 'l' && oper[index + 1] == 'o' && oper[index + 2] == 'g') return 3;  else if (oper[index] == 's' && oper[index + 1] == 'i' && oper[index + 2] == 'n') return 3;  else if (oper[index] == 's' && oper[index + 1] == 'q' && oper[index + 2] == 'r' && oper[index + 3] == 't') return 4;  else if (oper[index] == 'a' && oper[index + 1] == 'r' && oper[index + 2] == 'c' && oper[index + 3] == 'c' && oper[index + 4] == 'o' && oper[index + 5] == 's') return 6;  else if (oper[index] == 'a' && oper[index + 1] == 'r' && oper[index + 2] == 'c' && oper[index + 3] == 's' && oper[index + 4] == 'i' && oper[index + 5] == 'n') return 6;  return 0;  }  void calculator::component(string input)  {  string temp;  for (size\_t i = 0; i < input.size(); i++)  {  temp.clear();  size\_t control = 0;  while (((input[i] >= 48 && input[i] <= 57) || (input[i] == ',')) && i < input.size())  temp.push\_back(input[i++]);  if (temp.size() != 0)  {  this->temp.push\_back(temp);  temp.clear();  i--;  continue;  }  if (Operator(input, i, 1))  temp.push\_back(input[i]);  if (input[i] == '(')  temp.push\_back(')');  else if (input[i] == ')')  temp.push\_back('(');  if (temp.size() != 0)  {  this->temp.push\_back(temp);  temp.clear();  continue;  }  if (input[i] == ' ')  continue;  try  {  size\_t temp\_2 = Func(input, i);  if (temp\_2)  {  for (size\_t j = i; j < i + temp\_2; j++)  temp.push\_back(input[j]);  temp\_2--;  size\_t control = 0;  do  {  temp\_2++;  if (input[i + temp\_2] == '(')  control++;  if (input[i + temp\_2] == ')')  control--;  temp.push\_back(input[i + temp\_2]);  } while (control != 0 && input.size() > i + temp\_2);  if (control != 0)  throw "";  this->temp.push\_back(temp);  temp.clear();  i += temp\_2;  continue;  }  }  catch (const char\* error)  {  cout << error << endl;  cout << "Ошибка ввода: ";  for (size\_t j = i; j < input.size(); j++)  cout << input[j];  cout << endl;  return;  }  try  {  size\_t temp\_2 = Const(input, i);  if (temp\_2)  {  for (size\_t j = i; j < i + temp\_2; j++)  temp.push\_back(input[j]);  this->temp.push\_back(temp);  temp.clear();  i += temp\_2 - 1;  continue;  }  }  catch (const char\* error)  {  cout << error << endl;  cout << "Незнакомый симол: ";  for (size\_t j = i; j < input.size(); j++)  cout << input[j];  cout << endl;  return;  }  throw "Незнакомы символ";  }  get\_normal();  this->temp.opposite();  }  string make\_res(string input)  {  string func, num;  char control = 0;  for (size\_t i = 0; i < input.size() - 1; i++)  {  if (input[i] == '(' && control == 0)  {  control++;  continue;  }  if (control)  num += input[i];  else func += input[i];  }  double result = 0, number;  if (num == "pi")  number = pi;  else if (num == "e")  number = e;  else  {  calculator temp;  temp.component(num);  temp.get\_normal();  temp.clone();  temp.calculate.opposite();  number = stod(temp.get\_result());  }  if (func == "cos") result = cos(number);  else if (func == "sin") result = sin(number);  else if (func == "tg") result = tan(number);  else if (func == "ctg") result = 1 / tan(number);  else if (func == "ln") result = log(number);  else if (func == "log") result = log10(number);  else if (func == "sqrt") result = sqrt(number);  else if (func == "arccos") result = acos(number);  else if (func == "arcsin") result = asin(number);  else throw "Unknown error";  return to\_string(result);  }  string make\_operator(string input, string number\_1, string number\_2)  {  double result = 0, num\_1, num\_2;  if (number\_1 == "pi")  num\_1 = pi;  else if (number\_1 == "e")  num\_1 = e;  else num\_1 = stod(number\_1);  if (number\_2 == "pi")  num\_2 = pi;  else if (number\_2 == "e")  num\_2 = e;  else num\_2 = stod(number\_2);  if (input == "+") result = num\_1 + num\_2;  if (input == "-") result = num\_1 - num\_2;  if (input == "\*") result = num\_1 \* num\_2;  if (input == "/") result = num\_1 / num\_2;  if (input == "^") result = pow(num\_1, num\_2);  return to\_string(result);  };  char get\_priority(string oper)  {  if (Number(oper) || Func(oper, 0) || Const(oper, 0))  return 0;  if (oper[0] == '+' || oper[0] == '-')  return 0;  if (oper[0] == '\*' || oper[0] == '/')  return 1;  return 2;  }  void calculator::get\_normal()  {  for (size\_t i = 0; i < temp.get\_size(); i++)  {  string new\_str;  if (Func(temp.at(i)))  {  calculator Temp;  string arg = temp.at(i);  for (size\_t i = Func(arg) + 1; i < arg.size() - 1; i++)  new\_str.push\_back(arg[i]);  Temp.make\_prefix(new\_str);  new\_str.clear();  for (size\_t i = 0; i < Func(arg) + 1; i++)  new\_str.push\_back(arg[i]);  new\_str += Temp.print\_prefix() + ')';  temp.set(i, new\_str);  }  else if (i == 0 && temp.at(i) == "-" && temp.at(i + 1) != ")" && !Func(temp.at(i + 1)) && !Operator(temp.at(i + 1)))  {  new\_str += temp.at(i) + temp.at(i + 1);  temp.remove(i + 1);  temp.set(i, new\_str);  }  else if (i == 0 && temp.at(i) == "-" && (temp.at(i + 1) == ")" || Func(temp.at(i + 1))))  {  temp.push\_front("0");  i++;  }  else if (temp.at(i) == ")" && i + 1 < temp.get\_size() && temp.at(i + 1) == "-" && !Func(temp.at(i + 2)))  {  new\_str += temp.at(i + 1) + temp.at(i + 2);  temp.remove(i + 1);  temp.set(i + 1, new\_str);  }  else if (temp.at(i) == ")" && i + 1 < temp.get\_size() && temp.at(i + 1) == "-" && Func(temp.at(i + 2)))  {  temp.insert("0", i + 1);  }  }  }  void calculator::get\_prefix()  {  duaList stack;  for (size\_t i = 0; i < temp.get\_size(); i++)  {  if (Number(temp.at(i)) || Func(temp.at(i)))  calculate.push\_back(temp.at(i));  else if (Operator(temp.at(i)))  {  if (stack.isEmpty())  stack.push\_front(temp.at(i));  else if (stack.top() == "(")  stack.push\_front(temp.at(i));  else if (get\_priority(temp.at(i)) >= get\_priority(stack.top()))  stack.push\_front(temp.at(i));  else  {  while (!stack.isEmpty() && stack.top() != "(" && get\_priority(temp.at(i)) < get\_priority(stack.top()))  {  calculate.push\_back(stack.top());  stack.pop\_front();  }  stack.push\_front(temp.at(i));  }  }  else if (temp.at(i) == "(")  stack.push\_front(temp.at(i));  else if (temp.at(i) == ")")  {  while (!stack.isEmpty() && stack.top() != "(")  {  calculate.push\_back(stack.top());  stack.pop\_front();  }  stack.pop\_front();  }  }  while (!stack.isEmpty())  {  calculate.push\_back(stack.top());  stack.pop\_front();  }  calculate.opposite();  }  void calculator::make\_prefix(string input)  {  try {  component(input);  get\_prefix();  temp.clear();  }  catch (const char\* error)  {  temp.clear();  }  }  string calculator::print\_prefix()  {  if (calculate.get\_size() == 0)  {  calculate.clear();  throw "Неизвестный символ или нулевой аргумент. Введите новое арифметическое выражение или добавьте аргумент";  }  string result;  for (size\_t i = 0; i < calculate.get\_size(); i++)  result += calculate.at(i) + " ";  if (result.size() != 0)  result.pop\_back();  return result;  }  string calculator::get\_result()  {  string result;  size\_t control = 0;  if (calculate.get\_size() == 0)  result = "0";  else if (calculate.get\_size() == 1)  {  try {  if (Func(calculate.at(0)))  calculate.set(0, make\_res(calculate.at(0)));  else if (Const(calculate.at(0)))  if (calculate.at(0) == "pi")  result = to\_string(pi);  else result = to\_string(e);  else if (calculate.at(0) == "-pi")  result = to\_string(-1 \* pi);  else if (calculate.at(0) == "-e")  result = to\_string(-1 \* e);  else if (Number(calculate.at(0)))  result = calculate.at(0);  else  {  calculate.print\_to\_console();  calculate.clear();  throw "Некоректная функция";  }  }  catch (const char\* error)  {  calculate.print\_to\_console();  calculate.clear();  cout << error;  throw "Неизвестная ошибка";  }  }  else if (calculate.get\_size() == 2)  {  if ((calculate.at(0) != "-" && calculate.at(0) != "+") || (Number(calculate.at(1))))  {  calculate.print\_to\_console();  calculate.clear();  throw "Лишний оператор. Добавьте операнд или уберите оператор";  }  if (Func(calculate.at(1)) && calculate.at(0) == "-")  {  calculate.set(0, to\_string(-1 \* stod(make\_res(calculate.at(1)))));  }  else if (Func(calculate.at(1)) && calculate.at(0) == "+")  {  calculate.set(0, make\_res(calculate.at(1)));  }  calculate.pop\_back();  }  else  while (calculate.get\_size() > 1)  {  control = 0;  for (size\_t i = 0; i < calculate.get\_size(); i++)  {  if (calculate.get\_size() - i >= 3 && Operator(calculate.at(i)) && Number(calculate.at(i + 1)) && Number(calculate.at(i + 2)))  {  calculate.set(i, make\_operator(calculate.at(i), calculate.at(i + 1), calculate.at(i + 2)));  calculate.remove(i + 1);  calculate.remove(i + 1);  control++;  }  else if (Func(calculate.at(i)))  {  calculate.set(i, make\_res(calculate.at(i)));  control++;  }  else if (calculate.get\_size() == 2 && Func(calculate.at(1)) && calculate.at(0) == "-")  {  control++;  calculate.set(0, to\_string(-1 \* stod(make\_res(calculate.at(1)))));  calculate.pop\_back();  }  }  if (control == 0)  {  calculate.clear();  throw "Лишний оператор или операнд. Добавьте или уберите операнд или оператор";  }  }  result = calculate.at(0);  calculate.clear();  return result;  }  void calculator::clone()  {  for (size\_t i = 0; i < temp.get\_size(); i++)  calculate.push\_back(temp.at(i));  temp.clear();  }  void calculator::clear()  {  temp.clear();  calculate.clear();  } |
| **UnitTestКурс** |
| #include "pch.h"  #include "CppUnitTest.h"  #include "C:/Users/79825/Desktop/modak/аисдКурс/duaList.cpp"  #include "C:/Users/79825/Desktop/modak/аисдКурс/calculator.cpp"  using namespace Microsoft::VisualStudio::CppUnitTestFramework;  namespace UnitTestКурс  {  TEST\_CLASS(CalculatorTest)  {  calculator Test;  TEST\_METHOD(plus)  {  string res = "+ 2 2";  Test.make\_prefix("2+2");  Assert::AreEqual(Test.print\_prefix(), res);  }  TEST\_METHOD(minus)  {  string res = "- 2 2";  Test.make\_prefix("2-2");  Assert::AreEqual(Test.print\_prefix(), res);  }  TEST\_METHOD(multiply)  {  string res = "\* 2 2";  Test.make\_prefix("2\*2");  Assert::AreEqual(Test.print\_prefix(), res);  }  TEST\_METHOD(divide)  {  string res = "/ 2 2";  Test.make\_prefix("2/2");  Assert::AreEqual(Test.print\_prefix(), res);  }  TEST\_METHOD(trigan)  {  string res = "+ sin(pi) arccos(pi)";  Test.make\_prefix("sin(pi) + arccos(pi)");  Assert::AreEqual(Test.print\_prefix(), res);  }  TEST\_METHOD(brackets)  {  string res = "arcsin(/ pi 2)";  Test.make\_prefix("arcsin(pi/2)");  Assert::AreEqual(Test.print\_prefix(), res);  }  TEST\_METHOD(mix)  {  string res = "\* + 1 sin(pi) - 1 cos(pi)";  Test.make\_prefix("(1+sin(pi))\*(1-cos(pi))");  Assert::AreEqual(Test.print\_prefix(), res);  }  TEST\_METHOD(unary\_minus)  {  string res = "- 0 -2";  Test.make\_prefix("-(-2)");  Assert::AreEqual(Test.print\_prefix(), res);  }  TEST\_METHOD(unary\_sum)  {  string res = "- 0 + 1 2";  Test.make\_prefix("-(1+2)");  Assert::AreEqual(Test.print\_prefix(), res);  }  };  TEST\_CLASS(Calculator\_Test)  {  calculator test;  TEST\_METHOD(sum)  {  test.make\_prefix("2+2");  string result = "4.000000";  Assert::AreEqual(test.get\_result(), result);  }  TEST\_METHOD(difference)  {  test.make\_prefix("2-2");  string result = "0.000000";  Assert::AreEqual(test.get\_result(), result);  }  TEST\_METHOD(composition)  {  test.make\_prefix("2\*2");  string result = "4.000000";  Assert::AreEqual(test.get\_result(), result);  }  TEST\_METHOD(division)  {  test.make\_prefix("2/2");  string result = "1.000000";  Assert::AreEqual(test.get\_result(), result);  }  TEST\_METHOD(sum\_brackets)  {  test.make\_prefix("(2-1)+(2-1)");  string result = "2.000000";  Assert::AreEqual(test.get\_result(), result);  }  TEST\_METHOD(trigonometry)  {  test.make\_prefix("cos(pi)");  string result = "-1.000000";  Assert::AreEqual(test.get\_result(), result);  }  TEST\_METHOD(sqrt)  {  test.make\_prefix("sqrt(4)");  string result = "2.000000";  Assert::AreEqual(test.get\_result(), result);  }  TEST\_METHOD(logarithm)  {  test.make\_prefix("ln(e)");  string result = "1.000000";  Assert::AreEqual(test.get\_result(), result);  }  };    TEST\_CLASS(UnitTestКурс)  {  public:  duaList Empty;  TEST\_METHOD(Empty\_test)  {  Assert::AreEqual(Empty.isEmpty(), true);  }  TEST\_METHOD(Get\_Size\_test)  {  size\_t real\_size = 0;  Assert::AreEqual(Empty.get\_size(), real\_size);  }    TEST\_METHOD(Pop\_Back\_test)  {  Empty.pop\_back();  Assert::AreEqual(Empty.isEmpty(), true);  }  TEST\_METHOD(insert\_test)  {  try  {  Empty.insert("2", 0);  }  catch (const char\* error)  {  Assert::AreEqual("Incorrect index.", error);  }  }  TEST\_METHOD(remove\_test)  {  try  {  Empty.remove(2);  }  catch (const char\* error)  {  Assert::AreEqual("Incorrect index.", error);  }  }  TEST\_METHOD(Pop\_Front\_test)  {  Empty.pop\_front();  Assert::AreEqual(Empty.isEmpty(), true);  }  TEST\_METHOD(Clear\_test)  {  Empty.clear();  size\_t real\_size = 0;  Assert::AreEqual(Empty.get\_size(), real\_size);  }  TEST\_METHOD(Push\_Back\_test)  {  Empty.push\_back("5");  string str = "5";  Assert::AreEqual(Empty.at(0), str);  Empty.pop\_back();  }  TEST\_METHOD(Push\_Front\_test)  {  Empty.push\_front("0");  string str = "0";  Assert::AreEqual(Empty.at(0), str);  }  };  } |