|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** \_***ИУК «Информатика и управление»*\_\_**\_\_\_\_\_\_\_\_\_\_\_\_

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №2**

**«Создание и обработка древовидных структур данных»**

**ДИСЦИПЛИНА: «Типы и структуры данных»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-32Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_Зудин Д.В.\_\_\_\_\_)  (Подпись) (Ф.И.О.) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_Пчелинцева Н.И.\_\_\_)  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |
| Калуга, 2022 г.  **Цель**: формирование практических навыков создания алгоритмов обработки древовидных структур данных.  **Задачи**:  1. Изучить виды деревьев;  2. Научиться строить двоичные деревья, деревья поиска;  3. Изучить способы балансировки деревьев;  4. Познакомиться с основными алгоритмами обработки деревьев;  5. Реализовать основные алгоритмы обработки древовидных структур данных (создание, удаление, поиск, добавление и удаление элемента), а также алгоритм согласно полученному варианту.  **Вариант №3**  **Формулировка задания**  1. Разработать консольное приложение, написанное с помощью объектно-ориентированной технологии. Индивидуальное задание предусмотрено вариантом, который назначает преподаватель.  2. Приложение необходимо запускать для демонстрации из командной строки с указанием названий приложения и трех файлов:  − все входные данные (например, последовательности чисел, коэффициенты многочленов и т.д.) считать из первого файла;  − все выходные данные записать во второй файл;  − все возникшие ошибки записать в третий файл – файл ошибок.  3. Все основные сущности приложения представить в виде отдельных классов.  4. Необходимо предусмотреть пользовательское меню, содержащее набор команд всех основных операций для работы с двоичным деревом, а также команду для запуска индивидуального задания.  5. В приложении также должны быть учтены все критические ситуации, обработанные с помощью класса исключений.  **Индивидуальное задание**  Построить бинарное дерево следующего выражения: ((6 \* 3) + (8 \* 7)) \* (6 \* 5) и вывести его на экран. Написать процедуры постфиксного, инфиксного и префиксного обхода дерева и вывести соответствующие выражения.  **Листинг файла ExpTree.h**  #ifndef EXPTREE\_H  #define EXPTREE\_H  #include <string>  #include <vector>  namespace ExpressionTree  {  struct Node  {  char data;  Node\* left, \* right;  };  class ExpTree  {  private:  Node\* root = nullptr;  bool flag = false;  int countElem = 0;  std::string preorderS;  std::string postorderS;  std::string inorderS;  public:  std::string infixExp;  Node\* Create\_Node(int);  Node\* Create\_Node(Node\*, Node\*, int);  Node\* constructTree(std::string);  int getPriority(char ch);  std::string Convert\_In\_To\_Post(std::string infix);  void show\_node(Node\* T);  void preorder(Node\* T);  void preorder\_printing();  void postorder(Node\* T);  void postorder\_printing();  void inorder(Node\* T);  void inorder\_printing();  void printTree();  void delTree(Node\*);  void clearTree();  void printNode(Node\* t, int n);  int getCountElement() const;  void preorderF(Node\* T);  void inorderF(Node\* T);  void postorderF(Node\* T);  void WriteFile(std::string);  void string\_node(Node\*, std::string&);  bool IsExpCorrect(std::string infix);  bool isOperator(char ch);  bool isDigit(char c);  };  }  #endif  **Листинг файла ExpTree.cpp**  #include "ExpTree.h"  #include "MyError.h"  #include <iostream>  #include <fstream>  #include <string>  #include <limits>  #include <stack>  #include <cstdlib>  namespace ExpressionTree  {  Node\* ExpTree::Create\_Node(int info)  {  Node\* temp = new Node();  temp->left = nullptr;  temp->right = nullptr;  temp->data = info;  return temp;  };  Node\* ExpTree::Create\_Node(Node\* left, Node\* right, int info)  {  Node\* temp = new Node;  temp->left = left;  temp->right = right;  temp->data = info;  return temp;  };  Node\* ExpTree::constructTree(std::string infix)  {  infixExp = infix;  std::string postfix = Convert\_In\_To\_Post(infix);  std::stack<Node\*> s;  int i = 0;  Node\* node;  for (char c : postfix)  {  if (isOperator(c))  {  Node\* x = s.top();  s.pop();  Node\* y = s.top();  s.pop();  Node\* node = Create\_Node(y, x, c);  s.push(node);  }  else  {  s.push(Create\_Node(c));  }  countElem++;  i++;  }  root = s.top();  return root;  }  bool ExpTree::isDigit(char ch)  {  if (ch > 47 && ch < 57)  return true;  return false;  }  bool ExpTree::isOperator(char ch)  {  if (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '%' || ch == '^')  return true;  return false;  }  bool ExpTree::IsExpCorrect(std::string infix)  {  for (unsigned int i = 0; i < infix.length(); i++)  {  if (!isDigit(infix[i]) && !isOperator(infix[i]) && infix[i] != '(' && infix[i] != ')')  return false;  }  return true;  }  void ExpTree::show\_node(Node\* T)  {  std::cout << T->data << " ";  }  void ExpTree::preorder(Node\* T)  {  if (T != nullptr)  {  show\_node(T);  preorder(T->left);  preorder(T->right);  }  }  void ExpTree::preorder\_printing()  {  if (countElem != 0)  {  std::cout << "Префиксный обход" << std::endl;  preorder(root);  std::cout << std::endl;  }  else  {  std::cout << "Дерево пустое\n";  }  }  void ExpTree::postorder(Node\* T)  {  if (T != nullptr)  {  postorder(T->left);  postorder(T->right);  show\_node(T);  }  }  void ExpTree::postorder\_printing()  {  if (countElem != 0)  {  std::cout << "Постфиксный обход" << std::endl;  postorder(root);  std::cout << std::endl;  }  else  {  std::cout << "Дерево пустое\n";  }  }  void ExpTree::inorder(Node\* T)  {  if (T != NULL)  {  inorder(T->left);  show\_node(T);  inorder(T->right);  }  }  void ExpTree::inorder\_printing()  {  if (countElem != 0)  {  std::cout << "Инфиксный обход" << std::endl;  inorder(root);  std::cout << std::endl;  }  else  {  std::cout << "Дерево пустое\n";  }  }  int ExpTree::getPriority(char ch)  {  switch (ch)  {  case '^':  return 4;  case '%':  return 3;  case '/':  case '\*':  return 2;  case '+':  case '-':  return 1;  default:  return 0;  }  }  void ExpTree::delTree(Node\* tr)  {  if (tr != nullptr)  {  delTree(tr->left);  delTree(tr->right);  delete tr;  countElem = 0;  }  }  void ExpTree::clearTree()  {  delTree(root);  std::cout << "Дерево успешно удалено\n";  }  void ExpTree::printTree()  {  if (countElem != 0)  {  printNode(root, 0);  }  else  {  std::cout << "Дерево пустое\n";  }  }  void ExpTree::printNode(Node\* t, int n)  {  if (t)  {  printNode(t->right, n + 2);  for (int i = 0; i < n; i++)  {  std::cout << " ";  }  std::cout << t->data;  printNode(t->left, n + 2);  }  else  {  std::cout << std::endl;  }  }  int ExpTree::getCountElement() const  {  return countElem;  }  std::string ExpTree::Convert\_In\_To\_Post(std::string infix)  {  std::string postfix;  unsigned int counter1 = 0;  std::stack<char> st;  int postCount = 0;  char element;  while (counter1 < infix.length())  {  element = infix[counter1];  if (element == '(')  {  st.push(element);  counter1++;  continue;  }  if (element == ')')  {  while (!st.empty() && st.top() != '(')  {  postfix.push\_back(st.top());  st.pop();  }  if (!st.empty())  {  st.pop();  }  counter1++;  continue;  }  if (getPriority(element) == 0)  {  postfix.push\_back(element);  }  else  {  if (st.empty())  {  st.push(element);  }  else  {  while (!st.empty() && st.top() != '(' &&  getPriority(element) <= getPriority(st.top()))  {  postfix.push\_back(st.top());  st.pop();  }  st.push(element);  }  }  counter1++;  }  while (!st.empty())  {  postfix.push\_back(st.top());  st.pop();  }  return postfix;  }  void ExpTree::WriteFile(std::string nameFile)  {  std::ofstream out;  out.open(nameFile, std::ios::out);  preorderF(root);  inorderF(root);  postorderF(root);  if (out.is\_open())  {  if (!flag)  {  out << preorderS << std::endl;  out << inorderS << std::endl;  out << postorderS << std::endl;  out.close();  std::cout << "Запись в файл завершена!" << std::endl;  flag = true;  }  }  else  {  throw MyError{ "File didn't open" };  }  }  void ExpTree::string\_node(Node\* T, std::string& line)  {  line.push\_back(T->data);  }  void ExpTree::preorderF(Node\* T)  {  if (T != nullptr)  {  string\_node(T, preorderS);  preorderF(T->left);  preorderF(T->right);  }  }  void ExpTree::inorderF(Node\* T)  {  if (T != nullptr)  {  inorderF(T->left);  string\_node(T, inorderS);  inorderF(T->right);  }  }  void ExpTree::postorderF(Node\* T)  {  if (T != nullptr)  {  postorderF(T->left);  postorderF(T->right);  string\_node(T, postorderS);  }  }  }  **Листинг файла MyError.h**  #ifndef MYERROR\_H  #define MYERROR\_H  #include <string>  #include <string\_view>  namespace ExpressionTree  {  class MyError  {  public:  MyError(std::string error);  static std::string m\_file;  const char\* getError() const;  private:  void logging();  std::string m\_error;  };  }  #endif // MYERROR\_H  **Листинг файла MyError.cpp**  #define \_CRT\_SECURE\_NO\_WARNINGS  #include "MyError.h"  #include <iostream>  #include <fstream>  #include <chrono>  namespace ExpressionTree  {  const char\* MyError::getError() const  {  return m\_error.c\_str();  }  MyError::MyError(std::string error)  {  m\_error = error;  logging();  }  void MyError::logging()  {  std::fstream file;  auto now = std::chrono::system\_clock::now();  std::time\_t end\_time = std::chrono::system\_clock::to\_time\_t(now);  file.open(m\_file, std::ios::app);  file << "WARNING: " << m\_error.c\_str() << "|" << std::ctime(&end\_time);  file.close();  }  }  **Листинг файла CMenuItem.h**  #ifndef MYMENU\_CPP\_CMENUITEM\_H  #define MYMENU\_CPP\_CMENUITEM\_H  #include <string>  namespace ExpressionTree  {  class CMenuItem  {  public:  typedef int(\*Func)();  CMenuItem(std::string, Func);  Func m\_func{};  std::string m\_item\_name{};  std::string getName();  void print();  int run();  };  }  #endif //MYMENU\_CPP\_CMENUITEM\_H  **Листинг файла CMenuItem.cpp**  #include "CMenuItem.h"  #include <iostream>  namespace ExpressionTree  {  CMenuItem::CMenuItem(std::string item\_name, Func func) : m\_item\_name(item\_name), m\_func(func) {}  std::string CMenuItem::getName()  {  return m\_item\_name;  }  void CMenuItem::print()  {  std::cout << m\_item\_name;  }  int CMenuItem::run()  {  return m\_func();  }  }  **Листинг файла CMenu.h**  #ifndef MYMENU\_CMENU\_H  #define MYMENU\_CMENU\_H  #include "CMenuItem.h"  #include <cstddef>  namespace ExpressionTree  {  class CMenu  {  public:  CMenu();  CMenu(std::string, CMenuItem\*, size\_t);  int getSelect() const;  bool getRunning();  void setRunning(bool);  std::string getTitle();  size\_t getCount() const;  CMenuItem\* getItems();  void print();  void printTitle();  int runCommand();  private:  int m\_select{ -1 };  size\_t m\_count{};  bool m\_running{ true };  std::string m\_title{};  CMenuItem\* m\_items{};  };  }  #endif //MYMENU\_CMENU\_H  **Листинг файла CMenu.cpp**  #include "CMenu.h"  #include "MyError.h"  #include <iostream>  namespace ExpressionTree  {  CMenu::CMenu() {}  CMenu::CMenu(std::string title, CMenuItem\* items, size\_t count) : m\_title(title), m\_items(items), m\_count(count) {}  int CMenu::getSelect() const  {  return m\_select;  }  bool CMenu::getRunning()  {  return m\_running;  }  void CMenu::setRunning(bool \_running)  {  m\_running = \_running;  }  size\_t CMenu::getCount() const  {  return m\_count;  }  std::string CMenu::getTitle()  {  return m\_title;  }  CMenuItem\* CMenu::getItems()  {  return m\_items;  }  void CMenu::print()  {  for (size\_t i{}; i < m\_count - 1; ++i)  {  std::cout << i + 1 << ". ";  m\_items[i].print();  std::cout << std::endl;  }  std::cout << "0. ";  m\_items[m\_count - 1].print();  std::cout << std::endl;  }  void CMenu::printTitle()  {  system("cls");  std::cout << "\t" << m\_title << std::endl;  }  int CMenu::runCommand()  {  printTitle();  print();  std::cout << "\n Select >> ";  std::string SelectInput;  bool flag = true;  std::cin >> SelectInput;  for (int i{ 0 }; i < SelectInput.size(); i++)  {  if (!(SelectInput[i] >= '0' && SelectInput[i] <= '9'))  {  flag = false;  }  }  if (flag)  {  m\_select = std::stoi(SelectInput);  }  else  {  system("cls");  throw MyError{ "Wrong input. Enter only number." };  system("pause");  return 1;  }  if (m\_select == 0)  {  return m\_items[m\_count - 1].run();  }  else  {  if ((m\_select > m\_count - 1) || (m\_select < 0))  {  system("cls");  throw MyError{ "Wrong input. Enter correct number of menu." };  system("pause");  return 1;  }  else  {  system("cls");  return m\_items[m\_select - 1].run();  }  }  }  }  **Листинг файла main.cpp**  #include <iostream>  #include <cmath>  #include "CMenu.h"  #include "CMenuItem.h"  #include "ExpTree.h"  #include "MyError.h"  #include <fstream>  std::string ExpressionTree::MyError::m\_file = std::string();  using namespace ExpressionTree;  ExpTree tree;  std::fstream inp;  std::fstream out;  std::fstream excp;  float task\_result = 0;  int argc2;  char\*\* argv2;  #pragma region  int PrintTree()  {  system("cls");  if (tree.getCountElement() != 0)  {  std::cout << tree.infixExp;  tree.printTree();  }  else  {  std::cout << "Дерево пустое\n";  }  system("pause");  return 1;  }  int PreOrder()  {  system("cls");  tree.preorder\_printing();  system("pause");  return 2;  }  int InOrder()  {  system("cls");  tree.inorder\_printing();  system("pause");  return 3;  }  int PostOrder()  {  system("cls");  tree.postorder\_printing();  system("pause");  return 4;  }  int ClearTree()  {  system("cls");  if (tree.getCountElement() != 0)  {  tree.clearTree();  }  else  {  std::cout << "Дерево пустое\n";  }  system("pause");  return 5;  }  int PrintFile()  {  system("cls");  if (tree.getCountElement() != 0)  {  if (argc2 >= 3)  {  tree.WriteFile(argv2[2]);  }  else  {  tree.WriteFile("output.txt");  }  }  else  {  std::cout << "Дерево пустое\n";  }  system("pause");  return 6;  }  int Exit()  {  std::cout << std::endl << "Выход из программы" << std::endl;  return 0;  }  #pragma endregion  const int items\_number = 7;  void run(CMenu menu)  {  try  {  while (menu.runCommand()) {}  }  catch (const MyError& exception)  {  std::cout << "Error: " << exception.getError() << std::endl;  system("pause");  system("cls");  run(menu);  }  }  void ReadFile(std::string nameFile = "input.txt")  {  std::ifstream in;  std::string line;  char data;  if (tree.getCountElement() != 0)  {  tree.clearTree();  }  in.open(nameFile, std::ios::in);  if (!in.is\_open())  {  throw MyError{ "File didn't open" };  }  else  {  while (in >> data && !in.eof())  {  line.push\_back(data);  }  in.close();  std::cout << "Данные были импортированы из файла" << std::endl;  }  if (tree.IsExpCorrect(line))  {  tree.constructTree(line);  }  else  {  throw MyError{ "Incorrect expression" };  }  }  int main(int argc, char\* argv[])  {  setlocale(LC\_ALL, "Russian");  argc2 = argc;  argv2 = argv;  try  {  if (argc >= 4)  {  MyError::m\_file = std::string(argv[3]);  }  else  {  MyError::m\_file = std::string("exceptions.txt");  }  if (argc >= 2)  {  ReadFile(argv[1]);  system("pause");  }  else  {  ReadFile();  std::cout << "Используются файлы по умолчанию input.txt, output.txt, exceptions.txt\n";  system("pause");  }  }  catch (const MyError& exception)  {  std::cout << "Error: " << exception.getError() << std::endl;  system("pause");  system("cls");  }  CMenuItem items[items\_number]{  CMenuItem{"Распечатать дерево", PrintTree},  CMenuItem{"Префиксная запись", PreOrder},  CMenuItem{"Инфиксная запись", InOrder},  CMenuItem{"Постфиксная запись", PostOrder},  CMenuItem{"Удаление дерева", ClearTree},  CMenuItem{"Вывод в файл", PrintFile},  CMenuItem{"Выход", Exit}  };  CMenu menu("Дерево выражения", items, items\_number);  run(menu);  return 0;  }  **Результат выполнения программы для задания**  1. Запуск из консоли  DTaS\_Lab2V3 1.txt 2.txt 3.txt  Данные были импортированы из файла  Для продолжения нажмите любую клавишу . . .  2. Меню  Дерево выражения  1. Распечатать дерево  2. Префиксная запись  3. Инфиксная запись  4. Постфиксная запись  5. Удаление дерева  6. Вывод в файл  0. Выход  3. Вывод дерева на экран    4. Прямой обход  Префиксный обход  \* + \* 6 3 \* 8 7 \* 6 5  Для продолжения нажмите любую клавишу . . .  5. Симметричный обход  Инфиксный обход  6 \* 3 + 8 \* 7 \* 6 \* 5  Для продолжения нажмите любую клавишу . . .  6. Обратный обход  Постфиксный обход  6 3 \* 8 7 \* + 6 5 \* \*  Для продолжения нажмите любую клавишу . . .  7. Вывод обходов в файл  Запись в файл завершена!  Для продолжения нажмите любую клавишу . . .  8. Выходной файл после вывода обходов  \*+\*63\*87\*65  6\*3+8\*7\*6\*5  63\*87\*+65\*\*  9. Удаление дерева  Дерево успешно удалено  Для продолжения нажмите любую клавишу . . .  Дерево пустое  Для продолжения нажмите любую клавишу . . .  10. Запись ошибок  DTaS\_Lab2V3 1.txt 2.txt 3.txt  Данные были импортированы из файла  Error: Incorrect expression  Для продолжения нажмите любую клавишу . . .  11. Файл ошибок  WARNING: Incorrect expression|Sat Dec 17 14:22:04 2022  12. Выход из программы  Select >> 0  Выход из программы  **Выводы:**  В ходе работы были сформированы практические навыки создания алгоритмов обработки древовидных структур данных. | | |