|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 4** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Нелинейные структуры данных. Бинарное дерево.»** | |
|  | |
| Выполнил студент группы ИКБО-06-21 | Школьник Т.О. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г.. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2022

# **Цель работы**

Получение умений и навыков разработки и реализаций операций над структурой данных бинарное дерево.

# **Постановка задачи**

1. Разработать программу, в соответствии с требованиями варианта. Выполнить реализацию средствами ООП

Требования к заданиям:

1. Реализовать операции общие для вариантов с 8 по 15
   1. Создать дерево выражений в соответствии с вводимым выражением. Структура узла дерева включает: информационная часть узла – символьного типа: знак операции +, -, \* или цифра; указатель на левое и указатель на правое поддерево. В дереве выражения операнды выражений находятся в листьях дерева.
   2. Исходное выражение имеет формат:

<формула>::=цифра|<формула><знак операции><формула>

Примеры: 5; 1+2; 1+2+3\*4-5/6.

Отобразить дерево на экране, повернув его против часовой стрелки.

1. Реализовать операции варианта.
2. Разработать программу, демонстрирующую выполнение всех операций.
3. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Условие индивидуального варианта:

|  |  |  |
| --- | --- | --- |
| 8 | Символьное значение | Вывод дерева выражений по ширине.  Вернуть самый левый лист дерева.  Вычислить значение выражения |

# **3. Решение**

Дерево — это иерархическая структура данных, являющаяся ациклическим связным графом. Дерево состоит из узлов, которые реализуются в качестве структур (или классов) и содержат 3 поля: информационную часть (значение, которое узел хранит), указатель на левого потомка, указатель на правого потомка.

Узел, находящийся на самом верхнем уровне (не являющийся чьим- либо потомком) называется корнем. Узлы, не имеющие потомков (в программной реализации – указатели на левого и правого потомков имеют значение nullptr) называются листьями.

Для создания дерева была создана структура, реализующая один узел дерева (содержание структуры было указано выше):

|  |
| --- |
| struct TreeNode {  char value;  TreeNode\* leftChild;  TreeNode\* rightChild;  }; |

Для реализации задач упражнения были реализованы следующие функции:

* + - 1. Функция 1: Создание дерева

|  |
| --- |
| // функция создаёт 1 узел дерева  TreeNode\* create(char value) {  TreeNode\* node = new TreeNode;  node->value = value;  node->rightChild = nullptr;  node->leftChild = nullptr;  return node;  } |

* + - 1. Функция 2: перевод математического выражения в обратную польскую запись

|  |
| --- |
| // функция переводит выражение в вид обратной польской нотации  string convertToRPN(string expression) {  istringstream ss(expression);  string element;  stack<string> stack;  string rpn = "";  map<string, int> prioritet = {  {"(", 1},  {")", 1},  {"+", 2},  {"-", 2},  {"\*", 3},  {"/", 3}  };  while (ss >> element) {  bool isOperationKeyword = false;  for (auto& tmp : prioritet) {  if (tmp.first == element)  isOperationKeyword = true;  }  if (!isOperationKeyword) {  rpn += " " + element;  } else {  if (element == "(") {  stack.push(element);  } else if (element == ")") {  while (!stack.empty() && stack.top() != "(") {  rpn += " " + stack.top();  stack.pop();  }  stack.pop();  } else {  while (!stack.empty() && prioritet[stack.top()] >= prioritet[element]) {  rpn += " " + stack.top();  stack.pop();  }  stack.push(element);  }  }  }  while (!stack.empty()) {  rpn += " " + stack.top();  stack.pop();  }  return rpn;  } |

* + - 1. Функция 3: создание дерева выражения из обратной польской записи

|  |
| --- |
| // функция переводит выражение из обратной польской нотации и строит дерево  TreeNode\* convertFromRPN(string rpnExpression) {  stack<TreeNode\*> stackTree;  TreeNode\* treeNode;  for (int i = 0; i < rpnExpression.size(); i++) {  if (isdigit(rpnExpression[i])) {  treeNode = create(rpnExpression[i]);  stackTree.push(treeNode);  } else if (  rpnExpression[i] == '\*' or  rpnExpression[i] == '-' or  rpnExpression[i] == '+' or  rpnExpression[i] == '/') {  treeNode = create(rpnExpression[i]);  treeNode->rightChild = stackTree.top();  stackTree.pop();  treeNode->leftChild = stackTree.top();  stackTree.pop();  stackTree.push(treeNode);  }  }  return stackTree.top();  } |

* + - 1. Функция 4: вывод самого левого узла дерева

|  |
| --- |
| // функция ищет самого левого ребёнка(лист) дерева  char findMostLeftNode(TreeNode\* rootNode) {  if (rootNode->leftChild == nullptr)  return rootNode->value;  return findMostLeftNode(rootNode->leftChild);  } |

|  |
| --- |
| // функция выводит дерево повёрнутым на 90 градусов против ч. стр.  void printTree(TreeNode\* treeNode, int layer) {  if (treeNode != nullptr) {  printTree(treeNode->rightChild, layer + 1);  for (int i = 1; i <= layer; i++)  cout << " ";  cout << treeNode->value << endl;  printTree(treeNode->leftChild, layer + 1);  }  } |

1. Функция 5: вывод дерева, с разворотом против часовой стрелки

|  |
| --- |
| // функция считает значение введённого выражения  double calculateResult(TreeNode\* node) {  if (isdigit(node->value)){  return double(int(node->value) - 48);  }  switch (node->value) {  case '+': return double(calculateResult(node->leftChild)) + double(calculateResult(node->rightChild));  case '-': return double(calculateResult(node->leftChild)) - double(calculateResult(node->rightChild));  case '\*': return double(calculateResult(node->leftChild)) \* double(calculateResult(node->rightChild));  case '/': return double(calculateResult(node->leftChild)) / double(calculateResult(node->rightChild));  default: return 0.0;  }  } |

1. Функция 6: вычисление значения введённого выражения
2. Функция 7: вывод дерева выражения в ширину

|  |
| --- |
| // функция выводит дерево в ширину  void printTreeInWidth(TreeNode\* currentNode) {  if (currentNode->leftChild != nullptr) {  printTreeInWidth(currentNode->leftChild);  }  cout << "Value: " << currentNode->value << endl;  if (currentNode->rightChild != nullptr) {  printTreeInWidth(currentNode->rightChild);  }  } |

# **4. Тестирование**

Для удобства обращения с практической работой была реализовано консольное меню - пользователю представляется меню из 5 команд, каждая из которых отвечает за каждое задание практической работы (Рисунок 1).

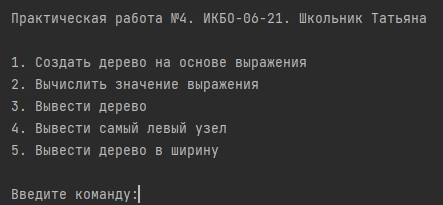


Рисунок 1. Интерфейс практической работы

Перейдем к тестированию программы

Функции 1 и 3 – создание дерева и его вывод в консоль (Рисунок 2)

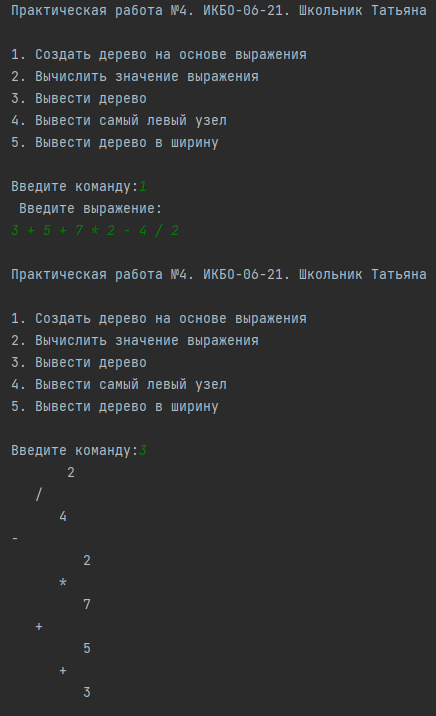


Рисунок 2. Тестирование функций создания дерева на основе выражения и вывода дерева в консоль

Функция 2 – подсчет значения выражения (Рисунок 3)

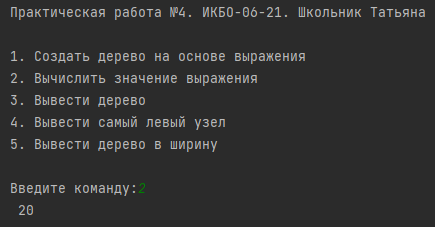


Рисунок 3. Тестирование функции подсчета значения выражения

Функция 4 – вывод самого левого узла дерева(Рисунок 4)

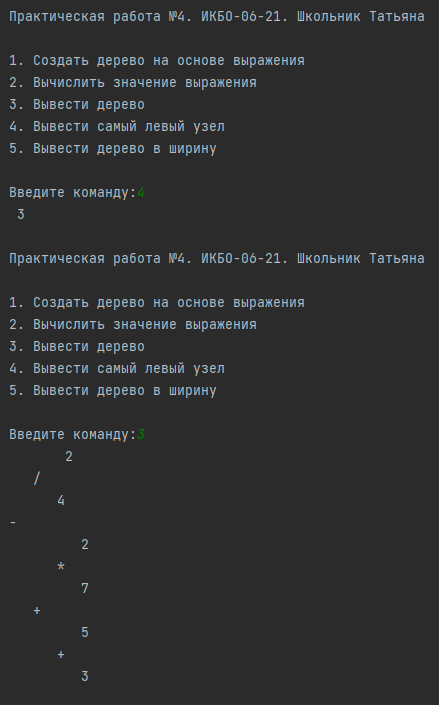


Рисунок 4. Тестирование функции вывода самого левого узла

Функция 5 – вывод дерева в ширину (Рисунок 5)

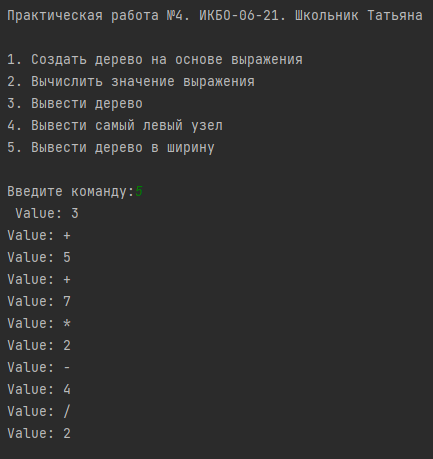


Рисунок 5. Тестирование функции вывода дерева в ширину

Из результатов выполнения программы видно, что программа работает корректно, выполняет условие индивидуального и общего варианта.

# **5. Вывод**

В результате выполнения работы были освоены методы работы с бинарными деревьями

# **6. Исходный код программы**

Код файла tree\_node.h

|  |
| --- |
| //  // Created by t.shkolnik  //  #include <iostream>  #include <string>  #include <stack>  #include <map>  #include <sstream>  using namespace std;  struct TreeNode {  char value;  TreeNode\* leftChild;  TreeNode\* rightChild;  };  // функция создаёт 1 узел дерева  TreeNode\* create(char value) {  TreeNode\* node = new TreeNode;  node->value = value;  node->rightChild = nullptr;  node->leftChild = nullptr;  return node;  }  // функция переводит выражение в вид обратной польской нотации  string convertToRPN(string expression) {  istringstream ss(expression);  string element;  stack<string> stack;  string rpn = "";  map<string, int> prioritet = {  {"(", 1},  {")", 1},  {"+", 2},  {"-", 2},  {"\*", 3},  {"/", 3}  };  while (ss >> element) {  bool isOperationKeyword = false;  for (auto& tmp : prioritet) {  if (tmp.first == element)  isOperationKeyword = true;  }  if (!isOperationKeyword) {  rpn += " " + element;  } else {  if (element == "(") {  stack.push(element);  } else if (element == ")") {  while (!stack.empty() && stack.top() != "(") {  rpn += " " + stack.top();  stack.pop();  }  stack.pop();  } else {  while (!stack.empty() && prioritet[stack.top()] >= prioritet[element]) {  rpn += " " + stack.top();  stack.pop();  }  stack.push(element);  }  }  }  while (!stack.empty()) {  rpn += " " + stack.top();  stack.pop();  }  return rpn;  }  // функция переводит выражение из обратной польской нотации и строит дерево  TreeNode\* convertFromRPN(string rpnExpression) {  stack<TreeNode\*> stackTree;  TreeNode\* treeNode;  for (int i = 0; i < rpnExpression.size(); i++) {  if (isdigit(rpnExpression[i])) {  treeNode = create(rpnExpression[i]);  stackTree.push(treeNode);  } else if (  rpnExpression[i] == '\*' or  rpnExpression[i] == '-' or  rpnExpression[i] == '+' or  rpnExpression[i] == '/') {  treeNode = create(rpnExpression[i]);  treeNode->rightChild = stackTree.top();  stackTree.pop();  treeNode->leftChild = stackTree.top();  stackTree.pop();  stackTree.push(treeNode);  }  }  return stackTree.top();  }  // функция ищет самого левого ребёнка(лист) дерева  char findMostLeftNode(TreeNode\* rootNode) {  if (rootNode->leftChild == nullptr)  return rootNode->value;  return findMostLeftNode(rootNode->leftChild);  }  // функция выводит дерево повёрнутым на 90 градусов против ч. стр.  void printTree(TreeNode\* treeNode, int layer) {  if (treeNode != nullptr) {  printTree(treeNode->rightChild, layer + 1);  for (int i = 1; i <= layer; i++)  cout << " ";  cout << treeNode->value << endl;  printTree(treeNode->leftChild, layer + 1);  }  }  // функция считает значение введённого выражения  double calculateResult(TreeNode\* node) {  if (isdigit(node->value)){  return double(int(node->value) - 48);  }  switch (node->value) {  case '+': return double(calculateResult(node->leftChild)) + double(calculateResult(node->rightChild));  case '-': return double(calculateResult(node->leftChild)) - double(calculateResult(node->rightChild));  case '\*': return double(calculateResult(node->leftChild)) \* double(calculateResult(node->rightChild));  case '/': return double(calculateResult(node->leftChild)) / double(calculateResult(node->rightChild));  default: return 0.0;  }  }  // функция выводит дерево в ширину  void printTreeInWidth(TreeNode\* currentNode) {  if (currentNode->leftChild != nullptr) {  printTreeInWidth(currentNode->leftChild);  }  cout << "Value: " << currentNode->value << endl;  if (currentNode->rightChild != nullptr) {  printTreeInWidth(currentNode->rightChild);  }  } |

Код файла main.cpp

|  |
| --- |
| //  // Created by t.shkolnik  //  #include "tree\_node.h"  //Вводная функция, выбор задания  void onboarding() {  cout << endl;  cout << "Практическая работа №4. ИКБО-06-21. Школьник Татьяна" << endl;  cout << endl;  cout << "1. Создать дерево на основе выражения" << endl;  cout << "2. Вычислить значение выражения" << endl;  cout << "3. Вывести дерево" << endl;  cout << "4. Вывести самый левый узел" << endl;  cout << "5. Вывести дерево в ширину" << endl;  cout << endl;  cout << "Введите команду: ";  }  TreeNode\* task1() {  string expression;  cout << "Введите выражение: " << endl;  cin.get();  getline(cin, expression);  expression = convertToRPN(expression);  return convertFromRPN(expression);  }  void task2(TreeNode\* tree) {  if (tree == nullptr) {  tree = task1();  }  cout << calculateResult(tree) << endl;  }  void task3(TreeNode\* tree) {  if (tree == nullptr) {  tree = task1();  }  printTree(tree, 0);  }  void task4(TreeNode\* tree) {  if (tree == nullptr) {  tree = task1();  }  cout << findMostLeftNode(tree) << endl;  }  void task5(TreeNode\* tree) {  if (tree == nullptr) {  tree = task1();  }  printTreeInWidth(tree);  }  int main() {  setlocale(LC\_ALL, "rus");  TreeNode\* tree = nullptr;  int menu = 1;  while (menu != 0) {  onboarding();  cin >> menu;  switch (menu) {  case 1: tree = task1(); break;  case 2: task2(tree); break;  case 3: task3(tree); break;  case 4: task4(tree); break;  case 5: task5(tree); break;  }  }  system("pause");  return 0;  } |