Министерство образования Российской Федерации  
Пензенский государственный университет  
Кафедра «Вычислительная техника»

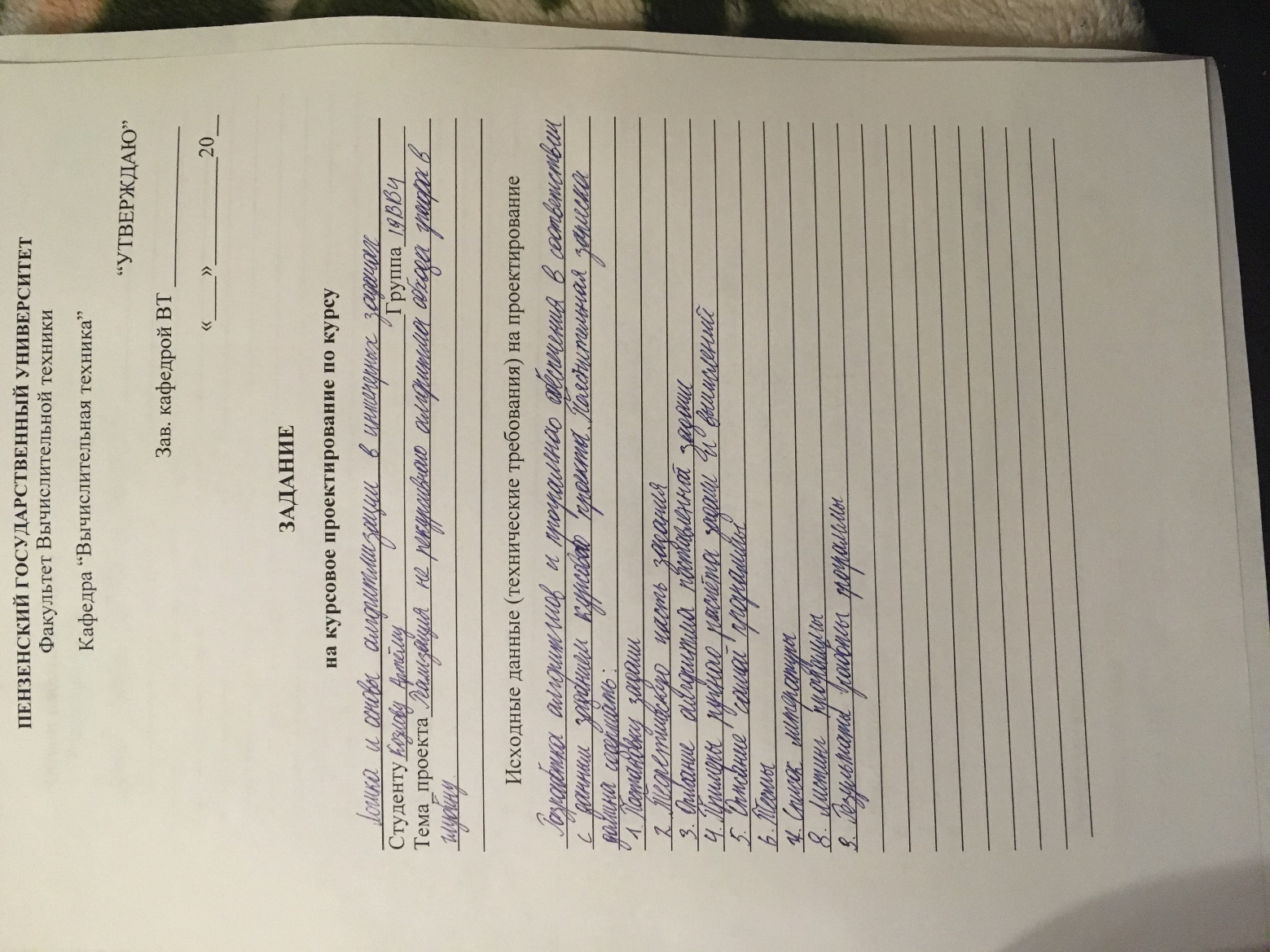
**Пояснительная записка**

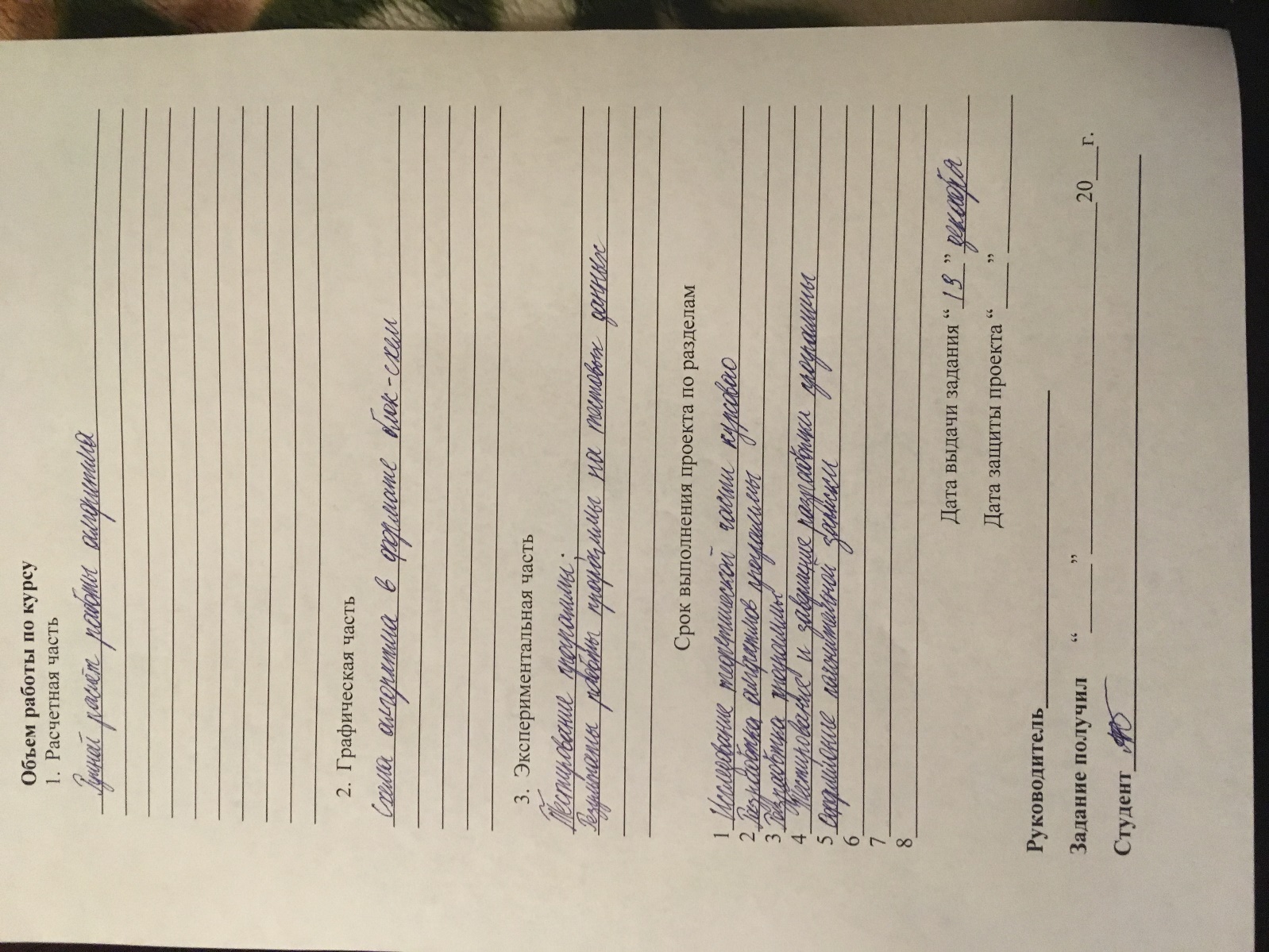
к курсовой работе  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Реализация не рекурсивного алгоритма обхода графа в глубину»

**Выполнил:**Студент группы 19ВВ4  
Козлов Артём

**Приняли:**Митрохин М.А.  
Юрова О.В.

Пенза 2020





**Содержание**

Реферат ............................................................................................................... 3

Введение............................................................................................................. 4

Постановка задачи…………………………………………………………...... 5

Теоретическая часть............................................................................................6

Описание алгоритма программы..................................................................... 7

Описание программы......................................................................................... 8

Тестирование ................................................................................................... 12

Ручной расчѐт задачи...................................................................................... 14

Заключение ...................................................................................................... 15

Список литературы ......................................................................................... 16

Приложение A. Листинг программы ............................................................ 17

**Реферат**

Отчёт 30 стр, 12 рисунков.

Цель исследования – изучение теории графов и деревьев, изучение способов нерекурсивного обхода деревьев. При сведении к итеративному алгоритму, так как дерево предполагает несколько путей обхода, часть узлов придётся "откладывать" для дальнейшей обработки, для чего будут использоваться стек или очередь.

В работе рассмотрены принципы работы алгоритма, способом нерекурсивного обхода деревьев.

**Введение**

При обходе графа в глубину, чем позже просмотрена вершина, тем раньше она используется потом, когда у текущей вершины не окажется смежных непросмотренных вершин и приходится возвращаться на шаг назад. Поэтому, используя для хранения просмотренных вершин стек, можно избавиться от рекурсии.

В качестве среды разработки была выбрана среда MicrosoftVisualStudio 2019, язык программирования – СИ.

Целью данной курсовой работы является разработка программы на языке Си, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм поиска в глубину.

1. Постановка задачи

Требуется разработать программу, которая будет определять прямой способ обхода дерева.

Программа должна работать так, чтобы пользователь вводил количество вершин в ручную, либо задавал любое другое число и выполнялся автоматический ввод. Результат программы должен вводиться на экран в консоле без ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

1. Теоретическая часть задания

Бинарное дерево — это иерархическая структура данных, в которой каждый узел имеет значение (оно же является в данном случае и ключом) и ссылки на левого и правого потомка. Узел, находящийся на самом верхнем уровне (не являющийся чьим либо потомком) называется корнем. Узлы, не имеющие потомков (оба потомка которых равны NULL) называются листьями.

В отличие от линейных структур типа односвязного списка и массива, у которых есть каноничный, прямой способ обхода, деревья можно обходить несколькими способами, в зависимости от поставленной задачи. Начиная с корня, можно применять необходимое действия (именуемое в дальнейшем "визит") как к самому узлу, так и к его левой или правой ветви. Порядок, в котором операции применяются, и будет определять способ обхода.

Наиболее простыми и понятными являются рекурсивные алгоритмы. При сведении к итеративному алгоритму, так как дерево предполагает несколько путей обхода, часть узлов придётся "откладывать" для дальнейшей обработки, для чего будут использоваться стек или очередь.

Существует способ обхода в глубину:

Прямой (pre-order)  
    Посетить корень  
    Обойти левое поддерево  
    Обойти правое поддерево

3. Описание алгоритма программы

|  |
| --- |
|  |
|  | Алгоритм прямого обхода дерева и подсчёта суммы всех его узлов  int calculateTreeSum(...) |
|  | Вызываем функцию для создания стека в переменную ps |
|  | Задаем начальное значение суммы узлов дерева. totalSum = 0 |
|  | До тех пор, пока размер стека != 0 или узел существует (!= NULL) |
|  | Если узел существует (!= NULL) |
|  | Выводим значение узла и прибавляем это значение в сумму. |
|  | Если существует правый узел |
|  | Заносим значение правого узла в стек |
|  | Переводим узел на левый подузел |
|  | Иначе |
|  | Переводим узел на последний занесенный правый узел из стека, удаляя его из стека. |
|  | После завершения обхода дерева освобождаем память, занятую стеком. |
|  | Возвращаем подсчитанное значение суммы узлов дерева totalSum. |

**4. Описание программы**

Для написания данной программы использован язык программирования Си. Язык программирования Си — универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Проект был создан в виде консольного приложения Win32 (Visual C++).

Данная программа является многомодульной , поскольку состоит из нескольких функций: Stack\* createStack(), void freeStack(Stack\*\* stack), void push(Stack\* stack, Node\* nodeItem), pop(Stack\* s)

Работа программы начинается с построения дерева из узлов:

typedef struct Node { // Структура узла дерева.

int data; // Значение узла дерева - число.

struct Node\* head; // Указатель на корневой узел.

struct Node\* left; // Указатель на левый подузел.

struct Node\* right; // Указатель на правый полузел.

} Node;

Каждый узел в себе несёт число значений узла, ссылку на левый узел, ссылку на правый узел и ссылку на корневой узел, если он имеется.

Node\* createNode() – функция, которая создаёт узел, она просто задаёт память и задаёт все значения нулями и возвращает значение созданной Node.

Структура стека, нужная для цикла перебора прохода по всему дереву, включает в себя: typedef struct Stack {

size\_t size; // Размер стека.

size\_t limit; // Лимит размера стека.

Node\*\* data; // Наполнение стека - узлы дерева.

} Stack;

Функция freeStack очищает полностью Stack и задаёт его значение нулём.

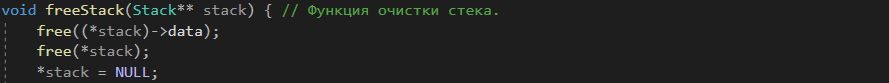


Рисунок Очистка стэка.

Функция Pop – выдача последнего узла из стека с его удалением. if (s->size == 0) { // Если стек пустой, вызов ошибки.

Далее идёт алгоритм обхода по дереву:

// Прямой обход дерева и подсчет суммы всех узлов - обход дерева в порядке: 1. Посещение корня дерева. 2. Посещение левого подузла. 3. Посещение правого подузла.

int calculateTreeSum(Node\* nodeRoot) { // Входные данные - указатель на корень верхнего узла.

Stack\* ps = createStack(); // Создание стека для обхода дерева.

int totalSum = 0;

while (ps->size != 0 || nodeRoot != NULL) { // Условие цикла - Размер стека не равен 0 И Значение корня не равно NULL (Индикация завершения обхода узла - конца узла).

if (nodeRoot != NULL) { // Если корень не равен NULL (Индикация завершения обхода узла - конца узла).

printf("Значение узла %d\n", nodeRoot->data); // Вывод значения корня - число.

totalSum += nodeRoot->data;

if (nodeRoot->right) { // Если есть правый узел - заносим его в стек.

push(ps, nodeRoot->right); // Запись в стек правого подузла.

}

nodeRoot = nodeRoot->left; // Перевод указателя на левый подузел.

}

else { // Иначе обход узла завершается.

nodeRoot = pop(ps); // В значение узла записывается указатель на последнюю вершину стека.

}

Само дерево задаётся в зависимости от режима ввода данных:

int main() {

setlocale(LC\_ALL, "Russian");

int inputDataMode; // Режим ввода входных данных. 1 - ручной ввод. Любое другое число - автоматический ввод.

Node\* inputNode;

int treeSum;

printf("Выберите режим ввода данных о дереве:\nВведите '1' для ручного ввода. Любое другое число выбирает автоматический ввод.\nРежим ввода данных:\t");

scanf\_s("%d", &inputDataMode);

if (inputDataMode == 1) {

inputNode = initNodeWithUserData();

}

Работа программы начинается с выбора режима ввода:

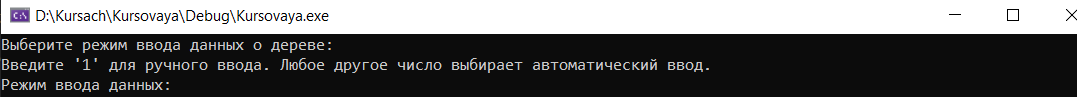


Рисунок Режим ввода данных о дереве

После происходит заполнение пользователем значений, путём ручного ввода.

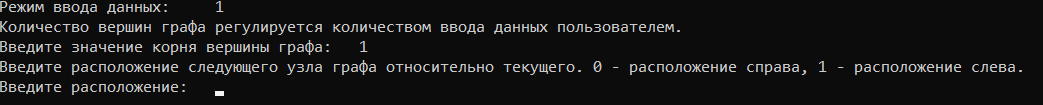


Рисунок Заполнение ручным вводом

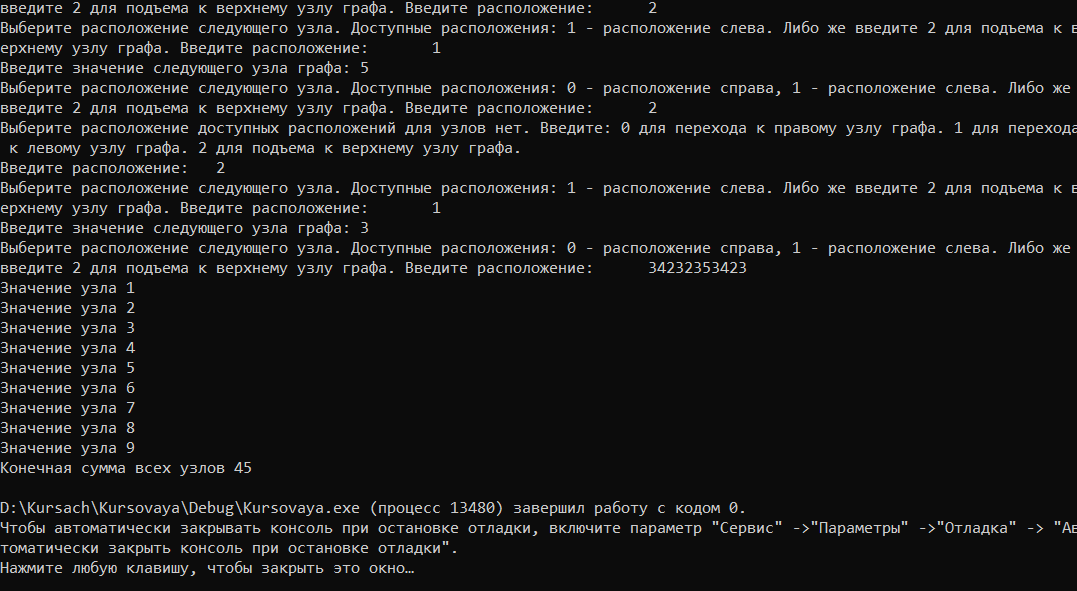
Для примера, что получится я использовал 9 узлов. 

Рисунок Работа программы.

**5. Результат программы**

Завершение программы при ручном вводе производиться вводом любых чисел.

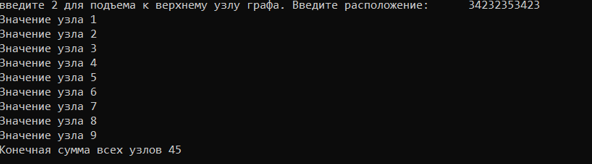


Рисунок Завершение программы при ручном вводе.

Завершение программы при автоматическом вводе производиться при вводе любого числа.

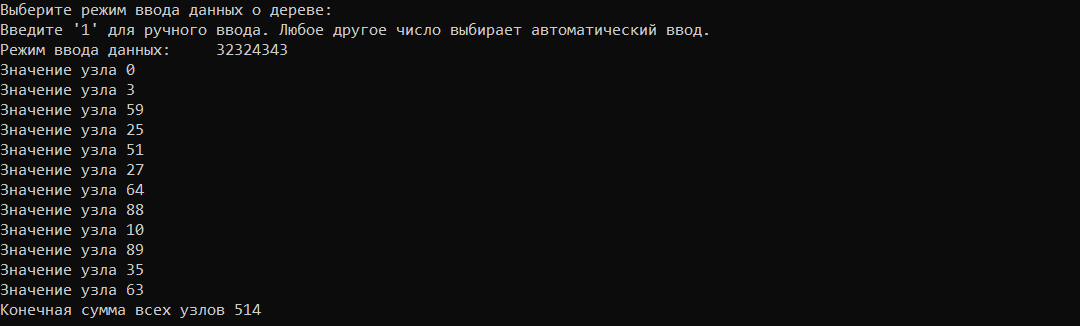


Рисунок Завершение программы при автоматическом вводе.

**6. Тестирование**

Среда разработки MicrosoftVisualStudio 2019 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, взаимодействием функции.

Запуск программы выполняется успешно:

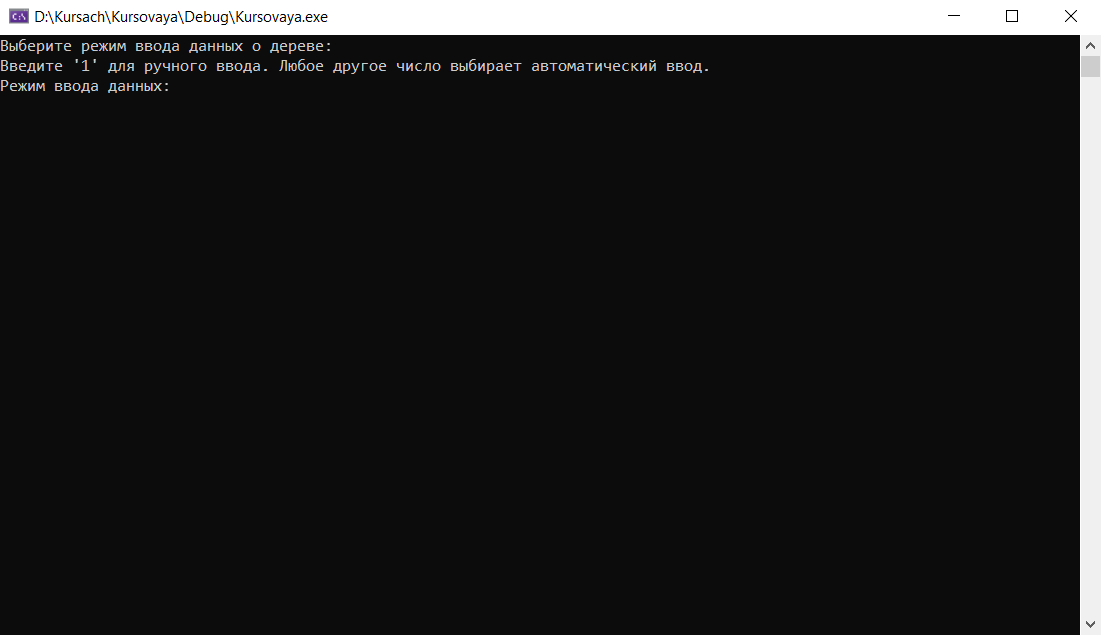


Рисунок Тестирование: Запуск программы

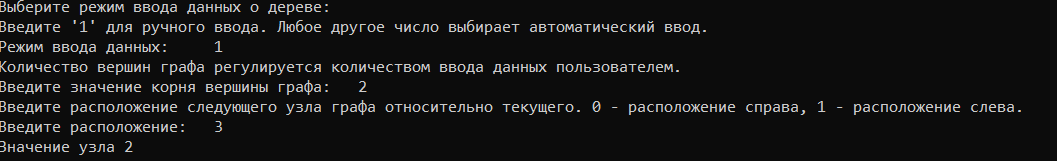
Ввод данных вручную выполняется успешно: 

Рисунок Тестирование: Ввод данных

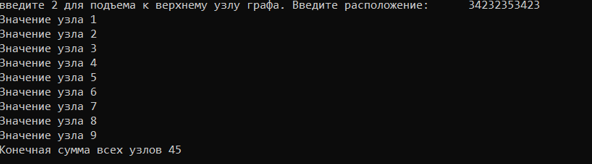
Конец программы тоже выполняется успешно: 

Рисунок Тестирование: Конец программы

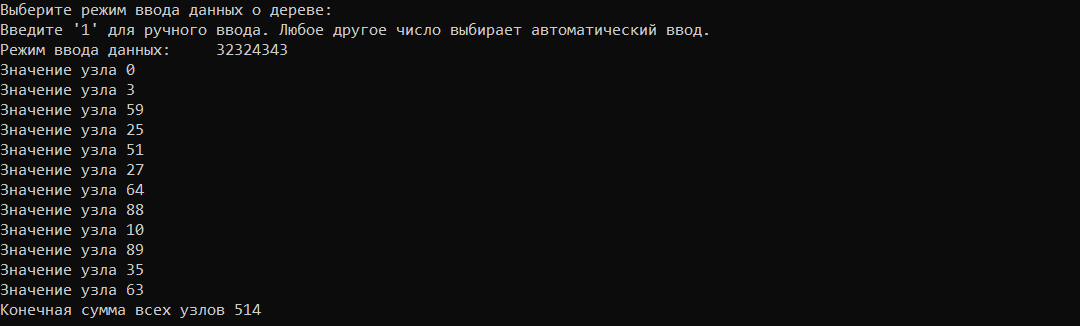
Так же успешно выполняется конец программы при автоматическом вводе данных: 

Рисунок Тестирование: Конец программы при автоматическом вводе

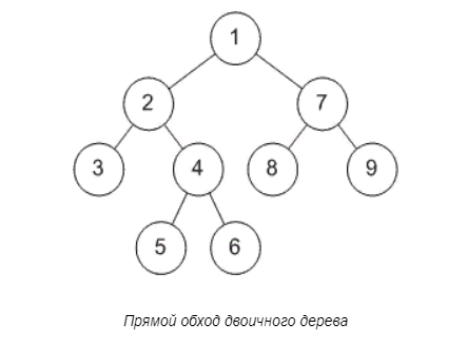
Таблица 1 – Описание поведения программы при тестировании

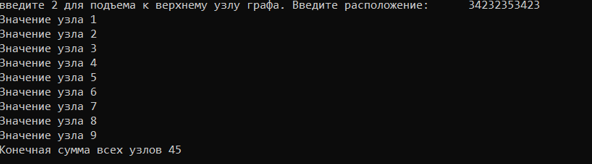
|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Успешный запуск | Верно |
| Ввод данных вручную | Не происходит никаких неполадок и ввод работает верно | Верно |
| Конец программы при ручном вводе данных | Правильный подсчёт данных | Верно |
| Ввод данных автоматически | Не происходит никаких неполадок и ввод работает правильно | Верно |
| Конец программы при автоматическом вводе данных | Правильный подсчёт данных | Верно |

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.

7. Ручной расчёт задачи

Проведём проверку программы посредством ручных вычислений.





Значение узлов по порядку обхода влево. То есть мы спускаемся влево, идёт 1 и 2, от 3 идём к 4 вводим. Потом 5 и 6, идём к 7, вводим 8 и потом переходим к 9.

1+2+3+4+5+6+7+8+9 = 45.

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая не рекурсивный алгоритм обхода в глубину в MicrosoftVisualStudio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приёмы работы обхода дерева в глубину. Углублены знания языка программирования Си.

Недостатком разработанной программы является примитивный пользовательский интерфейс и не самый удобный формат вывода результата работы программы.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

Список литературы

1. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.

2. Белецкая С.Ю. Комбинаторика. Графы. Алгоритмы: Учеб. пособие. - Воронеж: ВГТУ, 2003. -103 с.

3. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.

4. https://habr.com/ru/post/267855/

5. https://learnc.info/adt/binary\_tree\_traversal.html

**Приложение А.**

**Листинг.**

#include <stdio.h>

#include <stdlib.h>

#include <clocale> //Обязательно для функции setlocale()

#include <ctime>

#define STACK\_INIT\_SIZE 100

// Дерево будет состоять из узлов.

typedef struct Node { // Структура узла дерева.

int data; // Значение узла дерева - число.

struct Node\* head; // Указатель на корневой узел.

struct Node\* left; // Указатель на левый подузел.

struct Node\* right; // Указатель на правый полузел.

} Node;

Node\* createNode() {

Node\* tmp = (Node\*)malloc(sizeof(Node));

tmp->data = NULL;

tmp->head = NULL;

tmp->right = NULL;

tmp->left = NULL;

return tmp;

}

// Стек представляет из себя массив всех узлов дерева для обхода.

typedef struct Stack {

size\_t size; // Размер стека.

size\_t limit; // Лимит размера стека.

Node\*\* data; // Наполнение стека - узлы дерева.

} Stack;

Stack\* createStack() { // Функция создания стека.

Stack\* tmp = (Stack\*)malloc(sizeof(Stack));

tmp->limit = STACK\_INIT\_SIZE;

tmp->size = 0;

tmp->data = (Node\*\*)malloc(tmp->limit \* sizeof(Node\*));

return tmp;

}

void freeStack(Stack\*\* stack) { // Функция очистки стека.

free((\*stack)->data);

free(\*stack);

\*stack = NULL;

}

void push(Stack\* stack, Node\* nodeItem) { // Функция заполнения стека.

if (stack->size >= stack->limit) { // Если итоговое значение стека превышает лимит размера, то стек переаллоцируется под больший лимит.

stack->limit \*= 2;

stack->data = (Node\*\*)realloc(stack->data, stack->limit \* sizeof(Node\*));

}

stack->data[stack->size++] = nodeItem; // Добавление элемента в конец стека.

}

Node\* pop(Stack\* s) { // Функция выдачи последнего узла стека с его удалением.

if (s->size == 0) { // Если стек пустой, вызов ошибки.

exit(7);

}

s->size--; // Уменьшение размера стека.

return s->data[s->size]; // Возрват последнего узла стека.

}

// Прямой обход дерева и подсчет суммы всех узлов - обход дерева в порядке: 1. Посещение корня дерева. 2. Посещение левого подузла. 3. Посещение правого подузла.

int calculateTreeSum(Node\* nodeRoot) { // Входные данные - указатель на корень верхнего узла.

Stack\* ps = createStack(); // Создание стека для обхода дерева.

int totalSum = 0;

while (ps->size != 0 || nodeRoot != NULL) { // Условие цикла - Размер стека не равен 0 И Значение корня не равно NULL (Индикация завершения обхода узла - конца узла).

if (nodeRoot != NULL) { // Если корень не равен NULL (Индикация завершения обхода узла - конца узла).

printf("Значение узла %d\n", nodeRoot->data); // Вывод значения корня - число.

totalSum += nodeRoot->data;

if (nodeRoot->right) { // Если есть правый узел - заносим его в стек.

push(ps, nodeRoot->right); // Запись в стек правого подузла.

}

nodeRoot = nodeRoot->left; // Перевод указателя на левый подузел.

}

else { // Иначе обход узла завершается.

nodeRoot = pop(ps); // В значение узла записывается указатель на последнюю вершину стека.

}

}

freeStack(&ps); // По завершении обхода дерева освобождение используемого стека.

return totalSum;

}

Node\* initNodeWithUserData() {

int graphNodeData; // Значение корня графа.

int graphNodeDirection; // Расположение узла графа относительно вершины.

Node\* startNode = createNode(); // Вершина графа.

Node\* temporaryHeadNode = startNode; // Изменяемая переменная для хранения родительского узла.

Node\* temporaryNode = createNode(); // Изменяемая переменная для хранения узла.

printf("Количество вершин графа регулируется количеством ввода данных пользователем.\nВведите значение корня вершины графа:\t");

scanf\_s("%d", &graphNodeData);

startNode->data = graphNodeData;

printf("Введите расположение следующего узла графа относительно текущего. 0 - расположение справа, 1 - расположение слева.\nВведите расположение:\t");

scanf\_s("%d", &graphNodeDirection);

if (graphNodeDirection != 0 & graphNodeDirection != 1) {

return startNode;

}

printf("Введите значение следующего узла графа:\t");

scanf\_s("%d", &graphNodeData);

temporaryNode->data = graphNodeData;

if (graphNodeDirection == 0) {

temporaryHeadNode->right = temporaryNode;

}

else {

temporaryHeadNode->left = temporaryNode;

}

temporaryNode->head = temporaryHeadNode;

temporaryHeadNode = temporaryNode;

temporaryNode = createNode();

while (true) {

if (temporaryHeadNode->right == NULL & temporaryHeadNode->left == NULL) { // Если не задано значение правого и левого подузлов.

printf("Выберите расположение следующего узла. Доступные расположения: 0 - расположение справа, 1 - расположение слева. Либо же введите 2 для подъема к верхнему узлу графа. Введите расположение: \t");

scanf\_s("%d", &graphNodeDirection);

if (graphNodeDirection == 2 & temporaryHeadNode->head != NULL) { // Если выбран переход к верхнему узлу и узел доступен.

temporaryHeadNode = temporaryHeadNode->head;

continue;

}

else if (graphNodeDirection != 0 & graphNodeDirection != 1) {

break;

}

printf("Введите значение следующего узла графа:\t");

scanf\_s("%d", &graphNodeData);

temporaryNode->data = graphNodeData;

if (graphNodeDirection == 0) {

temporaryHeadNode->right = temporaryNode;

}

else {

temporaryHeadNode->left = temporaryNode;

}

}

else { // Иначе проверяем доступность подузлов по отдельности.

if (temporaryHeadNode->right == NULL) { // Если не задано значение правого подузла.

printf("Выберите расположение следующего узла. Доступные расположения: 0 - расположение справа. Либо же введите 2 для подъема к верхнему узлу графа. Введите расположение: \t");

scanf\_s("%d", &graphNodeDirection);

if (graphNodeDirection == 2 & temporaryHeadNode->head != NULL) { // Если выбран переход к верхнему узлу и узел доступен.

temporaryHeadNode = temporaryHeadNode->head;

continue;

}

else if (graphNodeDirection != 0) {

break;

}

printf("Введите значение следующего узла графа:\t");

scanf\_s("%d", &graphNodeData);

temporaryNode->data = graphNodeData;

temporaryHeadNode->right = temporaryNode;

}

else {

if (temporaryHeadNode->left == NULL) { // Если доступен левый узел.

printf("Выберите расположение следующего узла. Доступные расположения: 1 - расположение слева. Либо же введите 2 для подъема к верхнему узлу графа. Введите расположение: \t");

scanf\_s("%d", &graphNodeDirection);

if (graphNodeDirection == 2 & temporaryHeadNode->head != NULL) { // Если выбран переход к верхнему узлу и узел доступен.

temporaryHeadNode = temporaryHeadNode->head;

continue;

}

else if (graphNodeDirection != 1) {

break;

}

printf("Введите значение следующего узла графа:\t");

scanf\_s("%d", &graphNodeData);

temporaryNode->data = graphNodeData;

temporaryHeadNode->left = temporaryNode;

}

else { // Иначе предлагаем перемещение между узлами.

printf("Выберите расположение доступных расположений для узлов нет. Введите: 0 для перехода к правому узлу графа. 1 для перехода к левому узлу графа. 2 для подъема к верхнему узлу графа.\nВведите расположение: \t");

scanf\_s("%d", &graphNodeDirection);

if (graphNodeDirection == 0) {

temporaryHeadNode = temporaryHeadNode->right;

}

else if (graphNodeDirection == 1) {

temporaryHeadNode = temporaryHeadNode->left;

}

else if (graphNodeDirection == 2 & temporaryHeadNode->head != NULL) { // Если выбран переход к верхнему узлу и узел доступен.

temporaryHeadNode = temporaryHeadNode->head;

}

else {

break;

}

continue;

}

}

}

temporaryNode->head = temporaryHeadNode;

temporaryHeadNode = temporaryNode;

temporaryNode = createNode();

}

return startNode;

}

Node\* initNodeWithRandomData() {

Node\* startNode = createNode(); // Вершина графа.

srand((unsigned int)time(NULL));

Node\* subNode0 = createNode();

subNode0->data = rand() % 100;

subNode0->head = startNode;

startNode->right = subNode0;

Node\* subNode00 = createNode();

subNode00->data = rand() % 100;

subNode00->head = subNode0;

subNode0->right = subNode00;

Node\* subNode000 = createNode();

subNode000->data = rand() % 100;

subNode000->head = subNode00;

subNode00->right = subNode000;

Node\* subNode001 = createNode();

subNode001->data = rand() % 100;

subNode001->head = subNode00;

subNode00->left = subNode001;

Node\* subNode01 = createNode();

subNode01->data = rand() % 100;

subNode01->head = subNode0;

subNode0->left = subNode01;

Node\* subNode010 = createNode();

subNode010->data = rand() % 100;

subNode010->head = subNode01;

subNode01->right = subNode010;

Node\* subNode0100 = createNode();

subNode0100->data = rand() % 100;

subNode0100->head = subNode010;

subNode010->right = subNode0100;

Node\* subNode1 = createNode();

subNode1->data = rand() % 100;

subNode1->head = startNode;

startNode->left = subNode1;

Node\* subNode10 = createNode();

subNode10->data = rand() % 100;

subNode10->head = subNode1;

subNode1->right = subNode10;

Node\* subNode11 = createNode();

subNode11->data = rand() % 100;

subNode11->head = subNode1;

subNode1->left = subNode11;

Node\* subNode110 = createNode();

subNode110->data = rand() % 100;

subNode110->head = subNode11;

subNode11->right = subNode110;

return startNode;

}

int main() {

setlocale(LC\_ALL, "Russian");

int inputDataMode; // Режим ввода входных данных. 1 - ручной ввод. Любое другое число - автоматический ввод.

Node\* inputNode;

int treeSum;

printf("Выберите режим ввода данных о дереве:\nВведите '1' для ручного ввода. Любое другое число выбирает автоматический ввод.\nРежим ввода данных:\t");

scanf\_s("%d", &inputDataMode);

if (inputDataMode == 1) {

inputNode = initNodeWithUserData();

}

else {

inputNode = initNodeWithRandomData();

// RESULT NODE STRUCTURE

// inputNode

// subNode0 subNode1

// subNode00 subNode01 subNode10 subNode11

// subNode000 subNode001 subNode010 subNode110

// subNode0100

}

treeSum = calculateTreeSum(inputNode);

printf("Конечная сумма всех узлов %d\n", treeSum); // Вывод значения корня - число.

}