**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Математического обеспечения и применения ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №3**

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

**Тема: Бинарные деревья и лес**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9382 |  | Сорокумов С. В. |
| Преподаватель |  | Фирсов М. А. |

Санкт-Петербург

2020

## Цель работы.

Ознакомиться с принципами получения естественного представления леса бинарным деревом, с принципами работы векторной реализации очереди для прохода леса в ширину, векторной реализации хранения леса и бинарного дерева в памяти, а также с принципами построения леса и бинарного дерева.

## Постановка задачи.

**Вариант 7д.**

Для заданного леса с произвольным типом элементов:

- получить естественное представление леса бинарным деревом;

- вывести изображение леса и бинарного дерева;

- перечислить элементы леса в горизонтальном порядке (в ширину).

## Основные теоретические положения.

Дерево – конечное множество Т, состоящее из одного или более узлов, таких, что

а) имеется один специально обозначенный узел, называемый корнем данного дерева;

б) остальные узлы (исключая корень) содержатся в m 0 попарно не пересекающихся множествах Т1 , Т2 , ..., Тm, каждое из которых, в свою очередь, является деревом. Деревья Т1 , Т2 , ..., Тm называются поддеревьями данного дерева.

При программировании и разработке вычислительных алгоритмов удобно использовать именно такое рекурсивное определение, поскольку рекурсивность является естественной характеристикой этой структуры данных.

Бинарное дерево - конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом.

## Выполнение работы.

Сначала предлагается использовать по умолчанию строку из файла. Далее проходит проверка на открытие файла для чтения. После этого вызывается функция readFrt(&ptr[0], index, &str), в которой происходит чтение и запись выражения в массив классов Forest, использующий произвольный тип T(в текущей работе T = char). Далее функции printForest(ptr) и printBT(ptr) выводят изображения прочитанного леса и выведенного из него бинарного дерева соответственно. Далее создаем очередь Queue, состоящую из 20и элементов Forest<char>. Вызывается функция ForestForWidth(ptr, ptrQ), проходящая и попутно выводящая элементы леса в горизонтальном порядке.

**Описание функций.**

* bool askEnterBracket() – ф-ия, проводящая запрос на самостоятельное

задание выражения. Возвращает true при вводе пользователя ‘y’ или false при ‘n’.

* void error(char \*\*ptr) – ф-ия, выводящая выражение и знак <X> в

месте ошибки, т. е. куда указывает указатель на текущий символ выражения ptr. Завершает программу.

* Int readFrt(Forest<char> \*ptr, int &index, char \*\*str) - ф-ия получает на

вход указатель на текущий элемент массива forets[20], в котором предполагается хранение леса, ptr, индекс текущего элемента ptr index и указатель на текущее расположение в выражении str. Проходит по строке str, попутно синтаксически проверяя ее на правильность и записывая данные леса в ptr. Также на консоль выводятся промежуточные данные. Возвращает текущий index.

* int printForest(Forest<char> \*ptr, int index, int deep) - ф-ия получает на

вход указатель на текущий элемент массива forets[20], в котором хранится лес, ptr, индекс текущего элемента ptr index и количество совершаемых отступов при выведении элементов леса. Выводит изображение полученного леса. Возвращает число returnPoints, содержащее количество элементов массива, на которое index должен сдвинуться.

* void printBT(Forest<char> \*ptr, int index, int deep) - ф-ия получает на

вход указатель на текущий элемент массива forets[20], в котором хранится лес, ptr, индекс текущего элемента ptr index и количество совершаемых отступов при выведении элементов леса. Выводит изображение полученного бинарного дерева

* void ForestForWidth(Forest<char> \*ptr, Queue \*queue, int deep) - ф-ия

получает на вход указатель на текущий элемент массива forets[20], в котором хранится лес, ptr, указатель на очередь queue, индекс текущего элемента ptr index. Проводится проход по лесу и одновременный вывод элементов леса в горизонтальном порядке.

**Описание структур.**

template <typename T> class Forest {

public:

T root;

int sonFrt;

int broFrt;

Forest() { sonFrt = -1; broFrt = -1; }

int son() { return sonFrt; }

int bro() { return broFrt; }

T rt() { return root; }

void enterRt(T x) { root = x; }

};

T root – произвольный тип данных, содержащий корень элемента.

int sonFrt – индекс элемента, приходящегося «сыном» текущему элементу.

int broFrt – индекс элемента, приходящегося «братом» текущему элементу.

Forest() { sonFrt = -1; broFrt = -1; } – конструктор, приравнивающий индексы sonFrt и broFrt к -1.

int son() { return sonFrt; } – функция, возвращающая индекс «сына» sonFrt.

int bro() { return broFrt; } – функция, возвращающая индекс «брата» broFrt.

T rt() { return root; } – функция, возвращающая корень элемента root.

void enterRt(T x) { root = x; } – функция, получающая на вход произвольный тип x и помещающая его в корень root.

class Queue {

Forest<char> element[20];

int first;

int last;

public:

Queue() { first = 0; last = 0; }

bool isNull() {

if (first == last) return 1;

return 0;

}

void push(Forest<char> x) { element[last++] = x; }

Forest<char> pop() { return element[first++]; }

char topRt() { return element[first].root; }

};

Forest<char> element[20] – 20 элементов очереди.

int first – индекс, элемент которого является первым в очереди.

int last – индекс, элемент которого является последним в очереди.

bool isNull() {

if (first == last) return 1;

return 0;

} – функция, выводящая 1, если очередь пуста, иначе – 0.

push(Forest<char> x) { element[last++] = x; } – функция, получающая элемент массива и добавляющая его в конец очереди.

Forest<char> pop() { return element[first++]; } – функция, возвращающая первый элемент очереди и удаляющая его из нее.

char topRt() { return element[first].root; } – функция, возвращающая первый элемент очереди.

## Тестирование.

Таблица 1 – Результаты тестирования

|  |  |  |
| --- | --- | --- |
| № п/п | Входные данные | Выходные данные |
| 1 | (A) | Enumeration of forest elements in  horizontal order  A |
| 2 | (A(B)) | Enumeration of forest elements in  horizontal order  A B |
| 3 | (A(B)(C))(D)(E(F(G))) | Enumeration of forest elements in  horizontal order  A D E B C F G |
| 4 | (A-) | ERROR  (A<X>-) |
| 5 | (A(B)(C)(D)) | Enumeration of forest elements in  horizontal order  A B C D |
| 6 | (A)(B(C))(D(E(F(G)(H)))) | Enumeration of forest elements in  horizontal order  A B D C E F G H |
| 7 | (A(B)(C)(D))(E)(F(G(H))(I(J)  (K))) | Enumeration of forest elements in  horizontal order  A E F B C D G I H J K |

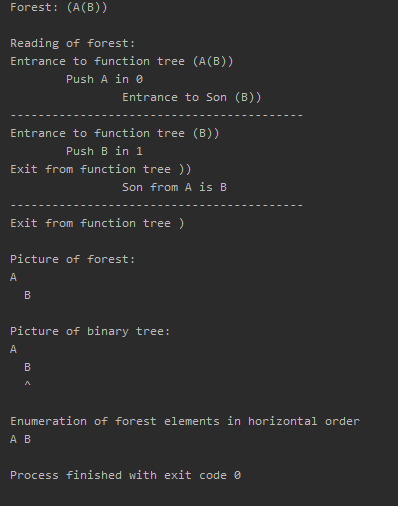


Рис. 1 – Пример работы всей программы

**Вывод.**

Была написана программа, которая для заданного леса с произвольным типом элементов: получили естественное представление леса бинарным деревом; вывели изображение леса и бинарного дерева; перечислили элементы леса в горизонтальном порядке (в ширину).

# ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

Сначала указываем имя файла, в котором код лежит в репозитории:

Название файла: main.cpp

#include<iostream>

#include<fstream>

#include<cstring>

#include<cstdio>

using namespace std;

/\*

\* Шаблонный класс леса

\*/

template <typename T> class Forest {

public:

T root;

int sonFrt;

int broFrt;

Forest() { sonFrt = -1; broFrt = -1; }

int son() { return sonFrt; }

int bro() { return broFrt; }

T rt() { return root; }

void enterRt(T x) { root = x; }

};

/\*

\* Класс очереди

\*/

class Queue {

Forest<char> element[20];

int first;

int last;

public:

Queue() { first = 0; last = 0; }

bool isNull() {

if (first == last) return true;

return false;

}

void push(Forest<char> x) { element[last++] = x; }

Forest<char> pop() { return element[first++]; }

char topRt() { return element[first].root; }

};

char \*ptrStart; //Начало выражения

bool askEnterBracket();

void error(char \*\*);

int readFrt(Forest<char> \*, int &, char \*\*);

int printForest(Forest<char> \*, int index = 0, int deep = 0);

void printBT(Forest<char> \*, int index = 0, int deep = 0);

void forestForWidth(Forest<char> \*ptr, Queue \*queue, int index = 0);

int main(int argc, char \*\*argv) {

char line[80] = "0";

char \*row = &line[0];

ptrStart = &line[0];

if(argc == 2){

/\*

\* Считывание данных из файла

\*/

setlocale(LC\_ALL, "rus");

char x[20];

strcpy(x,argv[1]);

ifstream infile(x, ios::in | ios::binary); //Открытие файла для чтения

if (!infile) { cout << "File not open for reading!\n"; exit(1); }

infile.get(line, 60);

infile.close();

} else{

/\*

\* считывание данных из консоли

\*/

std::cin >> line;

}

cout << "Forest: " << row << "\n\n";

Forest<char> forest[20];

Forest<char> \*ptr = &forest[0];

int index = 0;

cout << "Reading of forest:\n";

readFrt(&ptr[0], index, &row); //Функция чтения леса

cout << "\nPicture of forest:\n";

printForest(ptr); //Вывод изображения леса

cout << "\nPicture of binary tree:\n";

printBT(ptr); //Вывод изображения бинарного дерева

Queue queue, \*ptrQ;

ptrQ = &queue;

cout << "\nEnumeration of forest elements in horizontal order\n";

forestForWidth(ptr, ptrQ); //Проход по лесу в ширину

cout << endl;

return 0;

}

void error(char \*\*str) {//Функция, выводящая результат проверки выражения на скобку.

cout << "ERROR\n";

int x = strlen(ptrStart) - strlen(\*str);

while (\*ptrStart) {

printf("%c", \*ptrStart);

\*(++ptrStart);

if (!(--x)) //Отображается пробел, в месте, где сейчас находится указатель.

cout << "<X>";

}

cout << endl;

cout <<" <X> - Indicates the location of the error" << endl;

cout << endl;

exit(1);

}

/\*

\* Функция создания леса, путём обработки полученной строки

\*/

int readFrt(Forest<char> \*ptr, int &index, char \*\*str) {

int temp = index;

if (\*\*str == '(' && \*(\*str + 1) <= 'Z' && \*(\*str + 1) >= 'A') { // проверка текущего символа на букву

cout << "Entrance to function tree " << \*str << endl;

++\*str;

ptr[temp].enterRt(\*((\*str)++));

cout << "\tPush " << \*(\*str - 1) << " in " << temp << endl;

if (\*\*str == '(') {

/\*

\* Проверка на открывающую скобку

\* Переход на уровень ниже

\*/

cout << "\t\tEntrance to Son " << \*str << endl;

cout <<"------------------------------------------"<< endl;

ptr[temp].sonFrt = readFrt(ptr, ++index, str);

if (ptr[temp].son() != -1)

cout << "\t\tSon from " << ptr[temp].rt() << " is " << ptr[ptr[temp].son()].rt() << endl;

else cout << "\t\tSon from " << ptr[temp].rt() << " is NULL\n" << endl;

++\*str;

cout <<"------------------------------------------"<< endl;

}

if (\*\*str == ')' && \*(\*str + 1) == '(') {

/\*

\* Проверка на закрывающую скобку

\* Переход на уровень выше

\*/

cout << "\t\tEntrance to Bro " << \*str << endl;

cout <<"------------------------------------------"<< endl;

++\*str;

ptr[temp].broFrt = readFrt(ptr, ++index, str);

if (ptr[temp].bro() != -1)

cout << "\t\tBro from " << ptr[temp].rt() << " is " << ptr[ptr[temp].bro()].rt() << endl;

else cout << "\t\tBro from " << ptr[temp].rt() << " is NULL\n" << endl;

cout <<"------------------------------------------"<< endl;

}

cout << "Exit from function tree " << \*str << endl;

if (!(\*\*str == '(' || \*\*str == ')' || (\*(\*str + 1) <= 'Z' && \*(\*str + 1) >= 'A')))

/\*

\* возвращает ошибку данной строки

\*/

error(str);

return temp;

}

else error(str);

}

/\*

\* Рекурсивная функция печати леса

\*/

int printForest(Forest<char> \*ptr, int index, int deep) {

int returnPoints = 0;

for (int i = 0; i < deep; i++)

cout << ' ';

cout << ptr[index].rt() << endl;

if(ptr[index].son() != -1)

returnPoints = printForest(ptr, (index + 1), (deep + 2));

if (ptr[index].bro() != -1)

returnPoints = printForest(ptr, (index + 1 + returnPoints), deep);

return ++returnPoints;

}

/\*

\* Рекурсивная функция печати бинарного дерева

\*/

void printBT(Forest<char> \*ptr,int index, int deep) {

for (int i = 0; i < deep; i++)

cout << ' ';

cout << ptr[index].rt() << endl;

if (ptr[index].son() != -1) {

printBT(ptr, ptr[index].son(), deep + 2);

}

if (ptr[index].son() == -1 && ptr[index].bro() != -1) {

for (int i = 0; i < deep + 2; i++)

cout << ' ';

cout << '^' << endl;

}

if (ptr[index].bro() != -1) {

printBT(ptr, ptr[index].bro(), deep + 2);

}

if (ptr[index].son() != -1 && ptr[index].bro() == -1) {

for (int i = 0; i < deep + 2; i++)

cout << ' ';

cout << '^' << endl;

}

}

void forestForWidth(Forest<char> \*ptr, Queue \*queue, int index) {

cout << ptr[index].rt() << ' ';

if (ptr[index].son() != -1) {

queue->push(ptr[index]);

//cout << "Push " << ptr[index].rt() << endl;

}

if (ptr[index].bro() != -1) {

//cout << "Entrance to Bro\n"; //Закомментированные строки используются

forestForWidth(ptr, queue, ptr[index].bro());

//cout << "Exit from Bro\n"; //для вывода промежуточных результатов

}

if (!queue->isNull()) {

ptr[index] = queue->pop();

//cout << "Pop " << ptr[index].rt() << endl;

if (ptr[index].son() != -1) {

//cout << "Entrance to Son\n";

forestForWidth(ptr, queue, ptr[index].son());

//cout << "Exit from Son\n";

}

}

}