МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Бинарные деревья

Студент гр. 9303 _____ Лойконен М.Р.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Изучить нелинейную структуру данных бинарное дерево и способы его хранения. Научиться работать с данными, заданными в виде бинарного дерева.

Задание.

Вариант 11в.

С помощью построения дерева-формулы t преобразовать заданную формулу f из инфиксной формы в префиксную (перечисление узлов t в порядке КЛП) и в постфиксную (перечисление в порядке ЛПК). Если в дереве-формуле t терминалами являются только цифры, то вычислить (как целое число) значение дерева-формулы t.

формула::= терминал | формула знак формула

знак::= + | - | *

терминал::= 0 | 1 | ... | 9 | a | b | ... | z

Основные теоретические положения.

Дерево – конечное множество T, 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 T1, T2, ..., Tm, каждое из которых, в свою очередь, является деревом. Деревья T1, T2, ..., Tm называются поддеревьями данного дерева.

Каждый узел дерева является корнем некоторого поддерева. В том случае, когда множество поддеревьев такого корня пусто, этот узел называется концевым узлом, или листом. Уровень узла определяется рекурсивно следующим образом:

1) корень имеет уровень 1;

2) другие узлы имеют уровень, на единицу больший их уровня в содержащем их поддереве этого корня.

Говорят, что каждый корень является отцом корней своих поддеревьев и что последние являются сыновьями своего отца и братьями между собой.

Говорят также, что узел n – предок узла m (а узел m – потомок узла n), если n – либо отец m, либо отец некоторого предка m.

Если в определении дерева существен порядок перечисления поддеревьев T1, T2, ..., Tm , то дерево называют упорядоченным и говорят о «первом» (T1), «втором» (T2) и т. д. поддеревьях данного корня.

Лес – это множество (обычно упорядоченное), состоящее из некоторого (быть может, равного нулю) числа непересекающихся деревьев.

Используя понятие леса, пункт б в определении дерева можно было бы сформулировать так: узлы дерева, за исключением корня, образуют лес.

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.

Для реализации бинарного дерева был разработан шаблонный класс `BinTree` и структура узла дерева `Node`.

`Node` содержит поля: `T data` — поле, хранящее данные, `int left` — индекс левого поддерева, `int right` — индекс правого поддерева.

В классе `BinTree` есть поля `Node* vec` — массив, содержащий все элементы дерева и `size` — максимальный размер массива.

`void resize()` - метод, увеличивающий максимальный размер массива `vec`.

`void addElem(T data, int ind)` — добавляет элемент дерева в массив.

`void prefix(int i)` — вывод дерева в порядке КЛП.

`void postfix(int i)` — вывод дерева в порядке ЛПК.

Для чтения дерева в инфиксной записи используется функция `void readBinTree(string& infix, int start, int end, int ind, BinTree<char>* bt)`.

В случае, когда все листья бинарного дерева являются цифрами, программа считает результат исходного выражения с помощью функции `int result(BinTree<char>* bt, int ind, bool& err)`.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

В ходе выполнения работы была изучена структура бинарного дерева. Разработан класс `BinTree` на основе массива. Была написана программа, считывающая инфиксную запись бинарного дерева и выводящая его в префиксной (обход дерева в порядке КЛП) и постфиксной форме (обход дерева в порядке ЛПК).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <cctype>

using namespace std;

template <typename T>
class BinTree{
private:
     struct Node{
          T data;
          int left = 0;
          int right = 0;
     };
     int size;
public:
     Node* vec;
     BinTree(){
          vec = new Node [200];
          size = 200;
     }

     void resize(){
          int newSize = size + 200;
          Node* tmp = new Node [newSize];
          for (int i = 0; i<size; i++)
               tmp[i] = vec[i];
          delete [] vec;
          size = newSize;
          vec = tmp;
     }

     void addElem(T data, int ind){
          if (ind*2+2 == size)
               this->resize();
          vec[ind].data = data;
          vec[ind].left = ind*2 + 1;
          vec[ind].right = ind*2 + 2;
}
```

```
          void prefix(int i){
               if (vec[i].left || vec[i].right){
                    cout << vec[i].data << " ";
                    prefix(vec[i].left);
                    prefix(vec[i].right);
               }
          }

          void postfix(int i){
               if (vec[i].left || vec[i].right){
                    postfix(vec[i].left);
                    postfix(vec[i].right);
                    cout << vec[i].data << " ";
               }
          }

          ~BinTree(){
               delete [] vec;
          }
     };

     void readBinTree(string& infix, int start, int end, int ind,
BinTree<char>* bt){
          int depth = 0;
          if (start == end){
               bt->addElem(infix[start], ind);
               return;
          }
          for (int i = start; i<=end; i++){
               if (infix[i] == '(')
                    depth++;
               if (infix[i] == ')')
                    depth--;
               if (depth == 1 && (infix[i] == '+' || infix[i] ==
'-' || infix[i] == '*')){
                    bt->addElem(infix[i], ind);
                    readBinTree(infix,  start+1,  i-1,  ind*2+1,
bt);
                    readBinTree(infix, i+1, end-1, ind*2+2, bt);
               }
          }
     }
```

```
int result(BinTree<char>* bt, int ind, bool& err){
     if (bt->vec[ind].data == '+'){
          return   result(bt,   bt->vec[ind].left,   err)   +
result(bt, bt->vec[ind].right, err);
     }
     if (bt->vec[ind].data == '-'){
          return   result(bt,   bt->vec[ind].left,   err)   -
result(bt, bt->vec[ind].right, err);
     }
     if (bt->vec[ind].data == '*'){
          return   result(bt,   bt->vec[ind].left,   err)   *
result(bt, bt->vec[ind].right, err);
     }
     if (isdigit(bt->vec[ind].data))
          return bt->vec[ind].data - '0';
     else{
          err = true;
          return 0;
     }
}

int main(){
     string infix;
     ifstream fin("input.txt");
     while(!fin.eof()){
          BinTree <char>* bt = new BinTree <char>;
          fin >> infix;
          cout << "input: " << infix << '\n';
          readBinTree(infix, 0, infix.length()-1, 0, bt);
          cout << "prefix: ";
          bt->prefix(0);
          cout << '\n';
          cout << "postfix: ";
          bt->postfix(0);
          bool err = false;
          int res = result(bt, 0, err);
          if (!err)
               cout << '\n' << "result = " << res;
          cout << "\n\n";
          delete bt;
     }
     fin.close();
     return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(((a+b)*c)-(d+(e+(f*g))))	input: (((a+b)*c)-(d+(e+(f*g)))) prefix: - * + a b c + d + e * f g postfix: a b + c * d e f g * + + -	Тест пройден.
2.	(((a-b)+2)*3)	input: (((a-b)+2)*3) prefix: * + - a b 2 3 postfix: a b - 2 + 3 *	Тест пройден.
3.	((((a+b)+c)+d)+(k*p))	input: ((((a+b)+c)+d)+(k*p)) prefix: + + + + a b c d * k p postfix: a b + c + d + k p * +	Тест пройден.
4.	((3*5)-(9*(1+0)))	input: ((3*5)-(9*(1+0))) prefix: - * 3 5 * 9 + 1 0 postfix: 3 5 * 9 1 0 + * - result = 6	Тест пройден.
5.	((((5-6)*7)+9)+((4+5)-(2*4)))	input: ((((5-6)*7)+9)+((4+5)-(2*4))) prefix: + + * - 5 6 7 9 - + 4 5 * 2 4 postfix: 5 6 - 7 * 9 + 4 5 + 2 4 * - + result = 3	Тест пройден.
6.	(((4-9)+1)*((0*2)+7))	input: (((4-9)+1)*((0*2)+7)) prefix: * + - 4 9 1 + * 0 2 7 postfix: 4 9 - 1 + 0 2 * 7 + * result = -28	Тест пройден.
7.	(((a*(b-c))+(d-e))*(f-((k+(p-m))*n)))	input: (((a*(b-c))+(d-e))*(f-((k+(p-m))*n))) prefix: * + * a - b c - d e - f * + k - p m n postfix: a b c - * d e - + f k p m - + n * - *	Тест пройден.

8.	((f+(g*e))-(t-(p*m)))	input: ((f+(g*e))-(t-(p*m))) prefix: - + f * g e - t * p m postfix: f g e * + t p m * - -	Тест пройден.
9.	(((1-8)+(3-6))*(7*9))	input: (((1-8)+(3-6))*(7*9)) prefix: * + - 1 8 - 3 6 * 7 9 postfix: 1 8 - 3 6 - + 7 9 * * result = -630	Тест пройден.