**УСЛОВИЯ ЛАБОРАТОРНЫХ РАБОТ ЗА ТРЕТИЙ СЕМЕСТР**

По курсу Алгоритмы и Структуры Данных

**Лаба №1 "Задача о скобках"**

(Задача состоит из двух пунктов, но вы можете не париться и делать сразу второй)

На вход подаётся строка, состоящая из скобок. Программа должна определить правильность введённого скобочного выражения. Савкин сказал, что программа должна работать на русском языке: "Введите строку", "Строка не существует", "Строка существует" и т.п.

Пункт 1: В строке будут скобки только одного типа: или "()" , или "{}", или "[]"

Пункт 2: В строке будут все три вида скобок

Для успешной сдачи лабы оба пункта программа должна выполнять корректно (можно сделать отдельные программы на каждый пункт)

Пример входа:

()[({}())]

**Лаба №2 "Задача об арифметическом выражении"**

На вход подаётся математическое выражение. Элементы - числа. Операции - "+ - \* /". Также есть скобочки. Окончанием выражения служит "=". Программа должна вывести результат выражения

Пример ввода:

2+7\*(3/9)-5=

*Замечание:*

Программа также должна делать "проверку на дурака": нет деления на 0, все скобки стоят верно (см лабу №1) и т.п.

**Лаба №3 "Задача о простых множителях"**

На вход дается одно число х, нужно вывести все числа от 1 до х, удовлетворяющие условию:

i

где K, L, M - натуральные числа или могут быть равны 0.

**Лабы №4-12 "Методы сортировки"**

Дана последовательность чисел. Отсортировать и вывести последовательность чисел, определённым методом.

№4 Сортировка методом прочесывания

№5 Вставками

№6 Посредством выбора

№7 Шелла

№8 Поразрядная

№9 Пирамидальная (**heap sort)**

№10 Слиянием

№11 Быстрая

№12 Внешняя многофазная

========================

Дан текстовый файл с некоторым текстом на русском или английском языках произвольной длины (организовать чтение). Выбрав некоторую хеш-функцию, создать хеш-таблицу с:

Лаба №13 “с наложением”

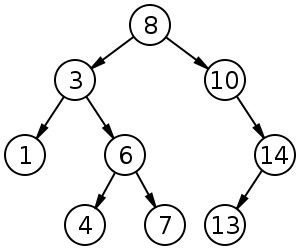
Лаба №14 “со списками”

Таблицу записать в результирующий файл.

========================

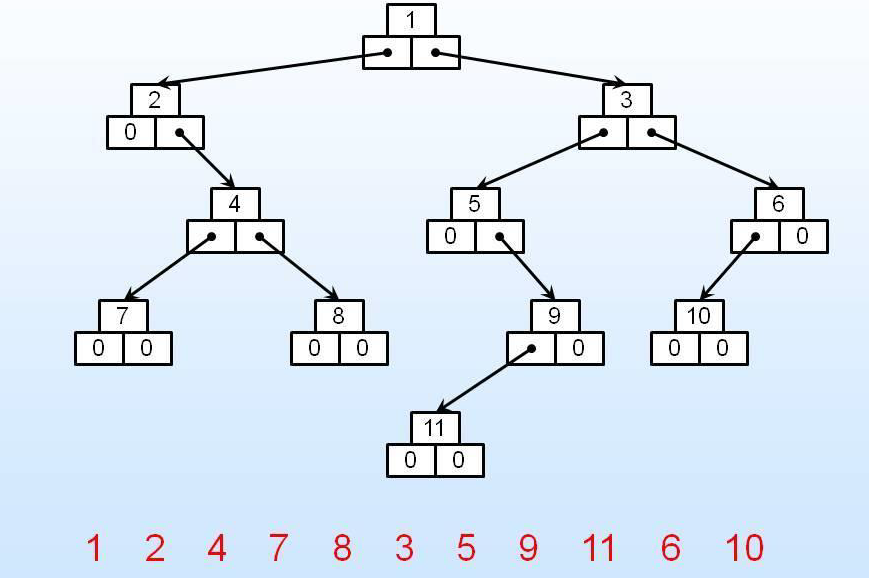
В качестве входной информации для следующих лабораторных работ вводится символьная строка (бинарное дерево) с помощью линейно-скобочной записи. Например 8 (3 (1, 6 (4,7)), 10 (, 14(13,)))

Что соответствует



Структура бинарного дерева создается с помощь динамических переменных.

Лаба №15 “Рекурсивные обходы (прямой, центральный, концевой)”



Лаба №16 “Не рекурсивный прямой обход” (реализуется с помощью стека).

В качестве выходных данных формируется строка обхода. Например:

Бинарное дерево поиска

Лаба №17 “Операции над БНП: поиск, добавление, удаление”

Дерево вводится в программу в формате линейно-скобочной записи. Затем появляется меню, в котором доступна операция добавления, удаления и поиска вершины БДП. После выполнения операции программа должна возвращаться снова в меню. При выходе их него до завершения программы на экран должно быть выведено БДН любым способом (в виде линейно-скобочной записи или в графической форме).

Комментарии к каждой строчке кода и объяснения функций:

struct Node

{

int data;

Node\* left;

Node\* right;

Node(int value) : data(value), left(nullptr), right(nullptr) {}

};

Эта структура представляет узел дерева. Она содержит данные (целое число) и указатели на левого и правого потомков.

class bnp

{

private:

Node\* root;

std::string brNot(Node\* node)

{

if (node == nullptr) { return ""; }

std::string result = std::to\_string(node->data);

if (node->left != nullptr || node->right != nullptr)

{

if (node->left != nullptr)

{

result += "(" + brNot(node->left) + ")";

}

if (node->right != nullptr)

{

result += "(" + brNot(node->right) + ")";

}

}

return result;

}

Этот класс представляет бинарное дерево поиска. Он содержит приватное поле root (корень дерева) и несколько функций для работы с деревом.

Функция brNot рекурсивно преобразует дерево в строку, используя обратную польскую запись. Она начинает с корня и добавляет его значение в результат. Затем она проверяет, есть ли у узла левый или правый потомок. Если есть, она добавляет открывающую скобку, рекурсивно вызывает brNot для потомка и добавляет закрывающую скобку. Функция возвращает полученную строку.

Node\* ins(Node\* node, int val)

{

if (node == nullptr) { return new Node(val); }

if (val < node->data) { node->left = ins(node->left, val); }

else if (val > node->data) { node->right = ins(node->right, val); }

return node;

}

Функция ins рекурсивно вставляет новый узел со значением val в дерево. Если дерево пустое, она создает новый узел и делает его корнем. Если значение меньше значения текущего узла, она рекурсивно вызывает ins для левого поддерева. Если значение больше, она рекурсивно вызывает ins для правого поддерева. Функция возвращает измененное дерево.

Node\* fMin(Node\* node)

{

while (node->left != nullptr) { node = node->left; }

return node;

}

Функция fMin находит минимальное значение в дереве. Она начинает с корня и продвигается влево до тех пор, пока не достигнет самого левого узла. Затем она возвращает этот узел.

Node\* fMax(Node\* node)

{

while (node->right != nullptr) { node = node->right; }

return node;

}

Функция fMax находит максимальное значение в дереве. Она начинает с корня и продвигается вправо до тех пор, пока не достигнет самого правого узла. Затем она возвращает этот узел.

bool searchRec(Node\* node, int value)

{

if (node == nullptr) { return false; }

if (value == node->data) { return true; }

else if (value < node->data)

{

return searchRec(node->left, value);

}

else

{

return searchRec(node->right, value);

}

}

Функция searchRec рекурсивно ищет значение value в дереве. Она начинает с корня и сравнивает значение с данными текущего узла. Если значение равно, она возвращает true. Если значение меньше, она рекурсивно вызывает searchRec для левого поддерева. Если значение больше, она рекурсивно вызывает searchRec для правого поддерева. Если достигнут конец дерева (узел равен nullptr), она возвращает false.

Node\* rem(Node\* node, int value)

{

if (node == nullptr) { return node; }

if (value < node->data)

{

node->left = rem(node->left, value);

}

else if (value > node->data)

{

node->right = rem(node->right, value);

}

else

{

if (node->left == nullptr)

{

Node\* temp = node->right;

delete node;

return temp;

}

else if (node->right == nullptr)

{

Node\* temp = node->left;

delete node;

return temp;

}

Node\* temp = fMin(node->right);

node->data = temp->data;

node->right = rem(node->right, temp->data);

}

return node;

}

Функция rem рекурсивно удаляет узел с заданным значением value из дерева. Если дерево пустое, она возвращает nullptr. Если значение меньше значения текущего узла, она рекурсивно вызывает rem для левого поддерева. Если значение больше, она рекурсивно вызывает rem для правого поддерева. Если значение равно, она проверяет, есть ли у узла левый и/или правый потомок. Если нет, она удаляет узел и возвращает соответствующий потомок. Если есть, она находит минимальное значение в правом поддереве (с помощью функции fMin), заменяет значение текущего узла на это минимальное значение, а затем рекурсивно вызывает rem для правого поддерева с удалением этого минимального значения. Функция возвращает измененное дерево.

public:

bnp() : root(nullptr) {};

void insert(int value)

{

root = ins(root, value);

}

int findMin()

{

return fMin(root)->data;

}

int findMax()

{

return fMax(root)->data;

}

void remove(int val)

{

root = rem(root, val);

}

bool search(int value)

{

return searchRec(root, value);

}

void printTree()

{

std::cout << "Tree: " << brNot(root) << std::endl;

}

};

Это публичная часть класса bnp. Она содержит конструктор, который инициализирует корень дерева nullptr, и несколько публичных функций для работы с деревом.

Функция insert вызывает приватную функцию ins для вставки нового значения в дерево.

Функция findMin вызывает приватную функцию fMin для поиска минимального значения в дереве и возвращает его.

Функция findMax вызывает приватную функцию fMax для поиска максимального значения в дереве и возвращает его.

Функция remove вызывает приватную функцию rem для удаления узла с заданным значением из дерева.

Функция search вызывает приватную функцию searchRec для поиска значения в дереве и возвращает true, если значение найдено, и false в противном случае.

Функция printTree выводит строковое представление дерева на экран.

int main()

{

bnp tree;

std::string s = "0(3(1)(6(7)(4)))(10(14)(13))";

int ind = 0;

while (ind < s.length())

{

bool flag = false;

int val = 0;

while ((s[ind] - '0') >= 0 and (s[ind] - '0') <= 9)

{

flag = true;

val \*= 10;

val += s[ind] - '0';

ind += 1;

}

if (flag)

{

tree.insert(val);

}

ind += 1;

}

std::cout << "Max: " << tree.findMax() << " min: " << tree.findMin() << " 1?: " << tree.search(1) << std::endl;

std::cout << " 1?: " << tree.search(1) << std::endl;

tree.insert(100);

tree.printTree();

tree.remove(0);

tree.printTree();

}

Это функция main, в которой создается объект класса bnp, выполняются операции вставки, поиска, удаления и вывода дерева.

Строка s содержит строковое представление дерева в обратной польской записи. Цикл while проходит по каждому символу строки и извлекает числа, добавляя их в дерево с помощью функции insert.

Затем выводятся максимальное и минимальное значения в дереве, а также результаты поиска значения 1 с помощью функции search.

Далее в дерево вставляется значение 100, выводится строковое представление дерева с помощью функции printTree, затем узел с значением 0 удаляется из дерева и снова выводится строковое представление дерева.