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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья

Студентка гр. 7381 _____ Алясова А.Н.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2018

Цель работы.

Ознакомление с парадигмой объектно-ориентированного программирования. Получение навыков реализации рекурсивных функций и использования динамической реализации бинарного дерева на языке программирования C++.

Постановка задачи.

Разработать программу для выполнения задания с использованием функций языка C++.

Задание варианта 8д:

Рассматриваются бинарные деревья с элементами типа char. Заданы перечисления узлов некоторого дерева b в порядке ЛКП и ЛПК. Требуется:

1) восстановить дерево b и вывести его изображение;
2) перечислить узлы дерева b в порядке КЛП.

Основные теоретические положения.

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

Определим скобочное представление бинарного дерева (БД):

< БД > ::= < пусто > | < непустое БД >,

< пусто > ::= Λ,

< непустое БД > ::= (< корень > < БД > < БД >).

Здесь пустое дерево имеет специальное обозначение Λ.

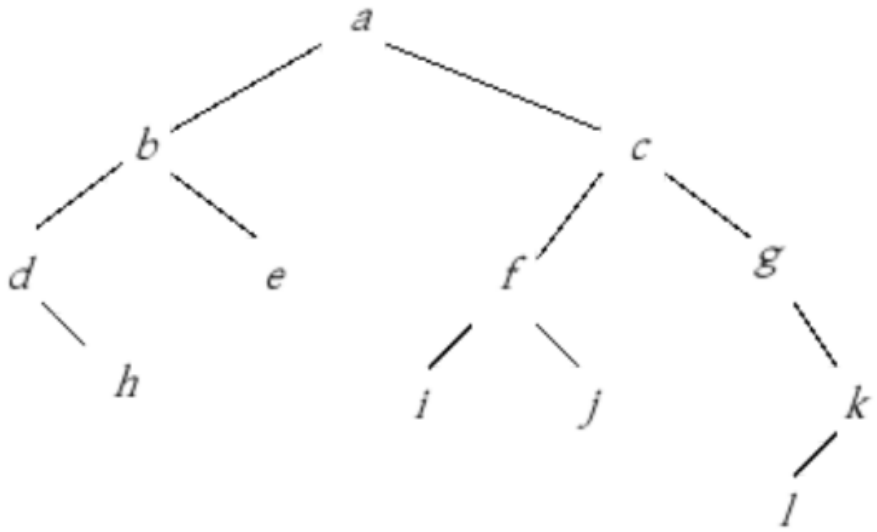


Рисунок 1 - Бинарное дерево

Например, бинарное дерево, изображенное на рис. 1, имеет скобочное представление : (a (b (d Λ (h Λ Λ)) (e Λ Λ)) (c (f (i Λ Λ) (j Λ Λ)) (g Λ (k (l Λ Λ) Λ)))).

Можно упростить скобочную запись бинарного дерева, исключив «лишние» знаки Λ по правилам:

1) (< корень > < непустое БД > Λ) = (< корень > < непустое БД >),

2) (< корень > Λ Λ) = (< корень >).

Тогда, например, скобочная запись бинарного дерева, изображенного на рис. 3.4, будет иметь вид (a (b (d Λ (h)) (e)) (c (f (i) (j)) (g Λ (k (l))))).

Многие алгоритмы работы с бинарными деревьями основаны на последовательной (в определенном порядке) обработке узлов дерева. В этом случае говорят об обходе (прохождении) бинарного дерева. Такой обход порождает определенный порядок перечисления узлов бинарного дерева. Будем именовать их в зависимости от того порядка, в котором при этом посещаются корень дерева и узлы левого и правого поддеревьев. Приведем рекурсивные процедуры КЛП-, ЛКП- и ЛПК-обходов, прямо соответствующие их рекурсивным определениям (операция обработки узла обозначена как «посетить (узел)»):

procedure *обходКЛП* (*b*: *BTree*);

```
{прямой}
begin
  if not Null (b) then
  begin
    посетить (Root (b));
    обходКЛП (Left (b));
    обходКЛП (Right (b));
  end
end{обходКЛП};
```

```
procedure обходЛКП (b: BTree);
{обратный}
begin
  if not Null (b) then
  begin
    обходЛКП (Left (b));
    посетить (Root (b));
    обходЛКП (Right (b));
  end
end{обходЛКП};
```

```
procedure обходЛПК (b: BTree);
{концевой}
begin
  if not Null (b) then
  begin
    обходЛПК (Left (b));
    обходЛПК (Right (b));
    посетить ( Root (b));
  end
end{обходЛПК}
```

Пример.

Дано бинарное дерево:

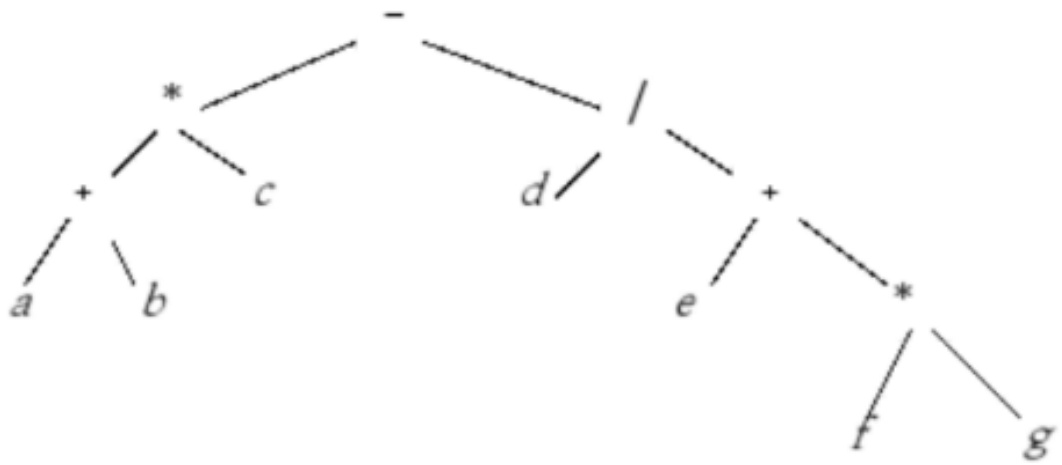


Рисунок 2 - Бинарное дерево

Тогда три варианта обхода этого дерева порождают три известные формы записи арифметического выражения:

1) КЛП – префиксную запись

** + a b c / d + e * f g*;

2) ЛКП – инфиксную запись

*a + b * c d / e + f * g*;

3) ЛПК – постфиксную запись

*a b + c * d e f g * + /.*

Ссылочная реализация бинарного дерева в связанной памяти основана на представлении типа *BT* (*Elem*) рекурсивными типами *BinT* и *Node*:

type	
BinT = ^*Node*;	{представление бинарного дерева}
Node = **record**	{узел: }
Info: *Elem*;	{− содержимое}
LSub: *BinT*;	{− левое поддерево}
RSub: *BinT*	{− правое поддерево}
end {*Node*}	

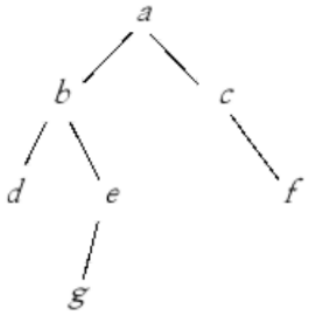
Здесь каждый узел дерева рассматривается как корень соответствующего поддерева, и этому поддереву сопоставляется запись из трех полей: поле *Info* хранит значение корня (типа *Elem*), а поля *LSub* и *RSub* − указатели на левое и правое поддеревья. Пустому дереву сопоставляется константа *NilBT* (на абстрактном уровне обозначаемая ранее как Λ). На рис. 3. а, б изображены бинарное дерево и его представление в ссылочной реализации.

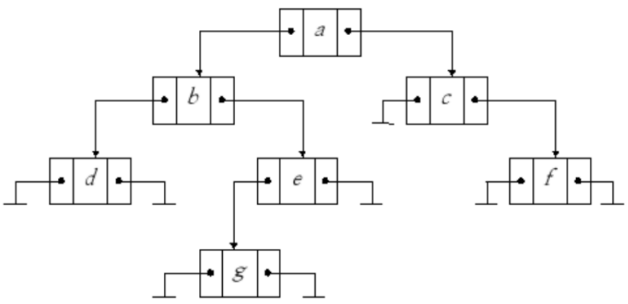
Спецификация программы.

Данная программа предназначена для построения изображения бинарного дерева по его ЛПК- и ЛКП-обходам, а также для вывода его КЛП-обхода.

Описание функций.

1) void menu(); - функция, предназначенная для вывода меню в консоль.
2) bool isNull(binTree); - функция, проверяющая, является ли данное binTree пустым.
3) base RootBT(binTree); - функция, выдающая текущий узел бинарного дерева binTree
4) void destroy(binTree&); - функция, разрущающая бинарное дерево binTree
5) binTree to_create(string LPK, string LKP); - функция, анализирующая строки LPK и LKP класса string и строящая по ним бинарное дерево и





выдающая указатель на это дерево. Она также выдает ошибку, если строки не удовлетворяют ЛПК- и ЛКП-обходам одного и того же дерева.

6) void printKLP(binTree b); - печать КЛП-представления бинарного дерева binTree.
7) int find_symb(string LPK, char word); - функция, ищущая в строке LPK символ word и возвращающая расстояние от начала строки до этого символа.
8) void print_tree(binTree elem, int l); - функция, предназначенная для печати дерева;l – текущий уровень дерева.
9) void if_error(string LPK, string LKP); - функция, выдающая ошибку, строки LPK и LKP класса string или имеют разную длину, или имеют разные относительно друг друга символы.

Структура данных

```
typedef char base; // тип узла

struct node {
        base info;//узел
        node *lt;//указатель на левого сына
        node *rt;//указатель на правого сына
        // constructor
        node(base elem) { info = elem; lt = NULL; rt = NULL; }
        };
```

Описание алгоритма.

Для решения этой задачи используется динамическое представление бинарного дерева. Изначально создается пустое бинарное дерево. Из считанных из файла или консоли ЛПК- или ЛКП-обходов программа посимвольно берет элементы из этих строк и проверяет, являются ли они равными. Если

они таковыми являются, значит, это крайний узел слева, следовательно, программа или создает его, а предыдущее дерево делает левым сыном созданного дерева, или вставляет в корень текущего дерева (если предыдущее дерево пусто). Если нет, то программа вставляет символ, являющийся текущим в ЛКП-обходе, либо в корень текущего дерева (если предыдущее дерево пусто), либо в созданный корень и присоединяет слева предыдущее поддерево. Помимо этого, при неравных символах она высчитывает расстояние от последнего обработанного символа строки ЛКП до этого же символа в строке ЛПК (символы, входящие в это расстояние по определению ЛПК находятся в правом поддереве), и, в зависимости от этого, выводит ошибку либо создает правое поддерево. Процесс продолжается до тех пор, пока не будет достигнут конец строк.

Тестирование.

№	Входные данные	Результат
1	LPK:dlkcbnhzmsa LKP:dclkbanhsmz	Full tree: z m s h n a b k l c d KLP: abcdklshnmz
2	LPK:cdbkmha	Full tree:

	LKP:cbdakhm	m h k a d b c KLP: abcdhkm
3	LPK:cdbxmh LKP:clsvm	Error.LPK and LKP have different sizes
4	LPK:avxd LKP:azxd	Error.LPK and LKP have different symbols
5	LPK: edbfzsgca LKP: debafcszg	Full tree: g z s c f a b e d KLP: abdecfgsz
6	Enter LPK:abcd Enter LKP:abcd	Full tree: d c b a KLP: dcba
7	Enter LPK:abxckds	Full tree:

	Enter LKP:sdkcxba	a b x c k d s KLP: sdkcxba
8	LPK:змгктжлшв LKP:мзгвкшжтл	Full tree: л т ж ш к в г з м KLP: вгмзшклжт
9	LPK:смы2лвт5 LKP:вт5ымс2л	Error!
10	LPK:xzcak2pum LKP:cakxzump2	Error!

Пример работы программы.

Enter LPK:вгжлаб17зк

Enter LKP:влгжка1бз7

----------Intermediate data----------

The knot [в] is processed.

Current subtree:

в

The knot [л] is processed.

Current subtree:

л

  в

The knot [г] is processed.

Current subtree:

г

The knot [ж] is processed.

Current subtree:

ж

  г

Current(united) subtree:

  ж

    г

л

  в

The knot [к] is processed.

Current subtree:

к

    ж

      г

  л

    в

The knot [a] is processed.

Current subtree:

a

The knot [1] is processed.

Current subtree:

1

a

The knot [б] is processed.

Current(united) subtree:

б

1

a

The knot [з] is processed.

Current subtree:

з

б

1

a

The knot [7] is processed.

Current(united) subtree:

7

з

б

1

a

Current(united) subtree:

7

з

б

1

a

к

ж

г

л

в

```
----------------------------------------
Full tree:
    7
  з
      б
    1
      а
к
    ж
      г
  л
    в
KLP: клвжгз1аб7
```

Выводы.

В ходе лабораторной работы была написана, отлажена и протестирована программа построения изображения бинарного дерева по его ЛПК- и ЛКП-обходам, а также вывода КЛП-обхода с использованием ссылочной реализации бинарного дерева. В программе также использовалось умение написания синтаксически и семантически корректных функций.