Лабораторная работа №4

БИНАРНЫЕ ДЕРЕВЬЯ

Цель работы: изучить теорию и научиться программировать бинарные деревья.

Теоретические сведения

Связные списки не охватывают весь спектр возможных представлений данных. Например, с их помощью сложно описать иерархические структуры подобные каталогам и файлам или хранения информации генеалогического древа. Для этого лучше подходит модель известная как бинарные деревья. Графически бинарные деревья можно изобразить как последовательность объектов, каждый из которых может быть связан с двумя последующими (рис. 5).

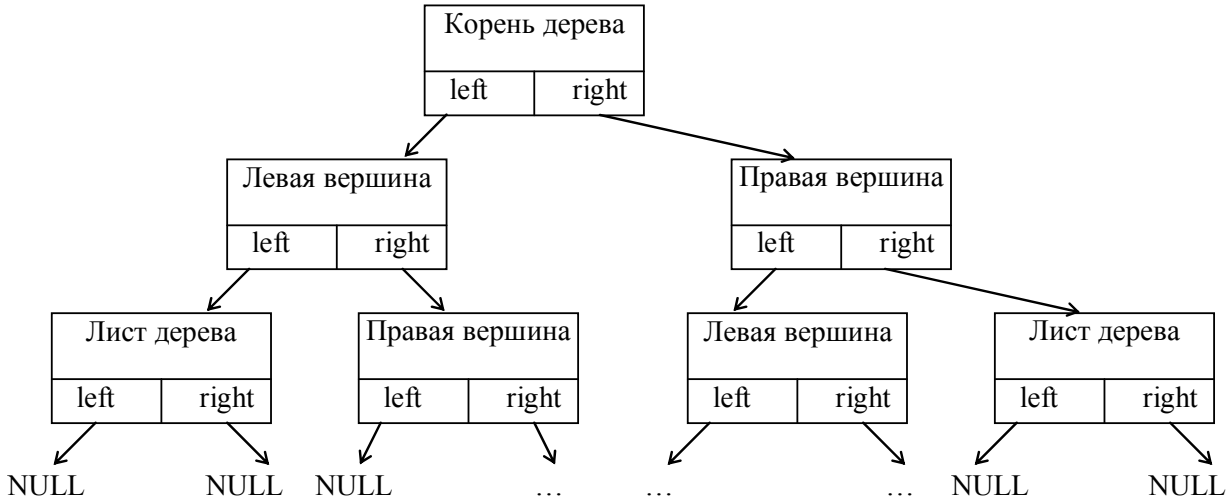

Рис. 5. Графическая интерпретация бинарного дерева

Каждый объект бинарного дерева имеет два указателя: на «левый» и «правый» вершины. Самый верхний уровень называется корнем дерева. Если указатели объекта left и right равны NULL, то он называет листом дерева.

При описании структуры каталогов с помощью бинарного дерева можно воспользоваться следующим правилом. Переход по «левым» указателям будет означать список файлов, а переход по «правым» – список каталогов. Например, для описания простой структуры (рис. 6), бинарное дерево будет иметь вид, представленный на рис. 7.

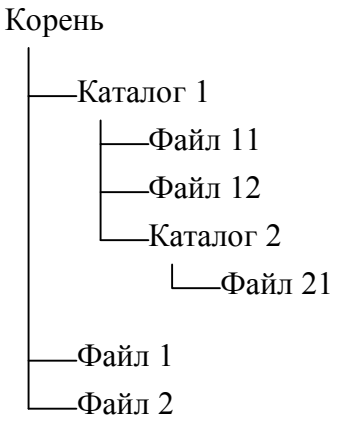

Рис. 6. Структура каталогов

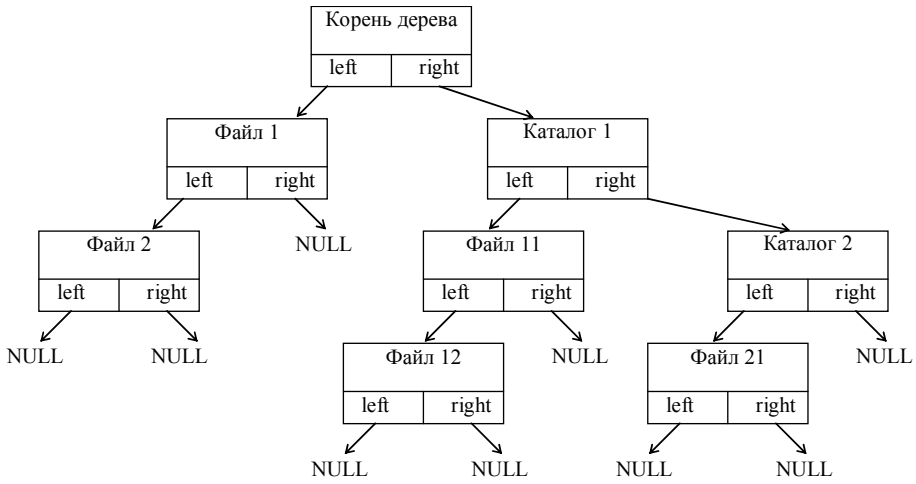

Рис. 7. Структура бинарного дерева

Объекты, из которых состоит бинарное дерево удобно представить в виде структур. Также как и в связных списках, первая структура будет описывать данные, хранящиеся в вершинах дерева, а вторая представлять связи между вершинами.

```
typedef struct tag_data {
 char name[100];
} DATA;

typedef struct tag_tree {
 DATA data;
 struct tag_tree* left, *right;
} TREE;
TREE* root = NULL;
```

Для формирования дерева введем функцию add_node(), которая в качестве аргументов будет принимать указатель на вершину дерева, к которому добавляются новые вершины, имя вершины и тип вершины: левая или правая. Кроме того, данная функция будет возвращать указатель на новую созданную вершину.

```
TREE* add_node(TREE* node,char* name, TYPE type = LEFT)
{
 TREE* new_node = (TREE *)malloc(sizeof(TREE));
 if(type == LEFT && node != NULL) node->left = new_node;
 else if(node != NULL) node->right = new_node;
 strcpy(new_node->data.name,name);
 new_node->left = NULL;
 new_node->right = NULL;
 return new_node;
}
```

Последний аргумент функции имеет тип TYPE, который удобно определить как перечисляемый тип:

```
typedef enum tag_type {RIGHT, LEFT} TYPE;
```

и определить параметр по умолчанию LEFT.

Для отображения дерева целесообразно воспользоваться рекурсивными функциями show_next(), которые вызываются из функции show_tree() следующим образом:

```
void show_next(TREE* node,int off)
{
 if(node != NULL)
 {
      for(int i=0;i < off;i++) putchar(' ');
      printf("%s\n",node->data.name);
      show_next(node->left,off);
      show_next(node->right,off+1);
 }
}
void show_tree()
{
 if(root != NULL)
 {
      printf("%s\n",root->data.name);
      show_next(root->left,0);
      show_next(root->right,1);
 }
}
```

Первый параметр функции show_next() служит для перемещения к следующим вершинам дерева, а второй – для смещения при отображении строк на экране монитора, принадлежащих разным вершинам. Рекурсия функций show_next() выполняется до тех пор, пока указатель на левую или правую вершину не станет равным NULL. В этом случае работа функции завершается, и управление передается предыдущей вызываемой функции. Таким образом, происходит отображение всего бинарного дерева.

При завершении программы необходимо удалить созданные объекты дерева. Для этого также удобно воспользоваться рекурсивными функциями. По аналогии введем рекурсивную функцию del_next(), и основную del_tree(), из которой вызывается функция del_next(). Реализация этих функций приведена ниже:

```
void del_next(TREE* node)
{
 if(node != NULL)
 {
      del_next(node->left);
      del_next(node->right);
      printf("node %s - deleted\n",node->data.name);
      free(node);
```

```
 }
}
void del_tree()
{
 if(root != NULL)
 {
      del_next(root->left);
      del_next(root->right);
      printf("node %s - deleted\n",root->data.name);
      free(root);
 }
}
```

Аргумент функции del_next() используется для перехода к следующим вершинам дерева. В самой функции выполняется проверка: если следующая вершина существует, т.е. указатель не равен NULL, то выполняется просмотр дерева сначала по левой вершине, а затем по правой. При достижении листьев дерева функции del_next() вызываются с аргументом NULL и не выполняют никаких действий, поэтому программа переходит к функции printf(), которая вывод на экран сообщение об имени удаляемой вершины, а затем вызывается функция free() для освобождения памяти, занятой данным объектом. После этого осуществляется переход к предыдущей функции del_next() и описанный процесс повторяется до тех пор, пока не будут удалены все объекты кроме корневого. Корень дерева удаляется непосредственно в функции del_tree(), после чего можно говорить об удалении всего дерева.

Использование описанных функций в функции main() реализуются следующим образом:

```
int main()
{
 root = add_node(NULL,"Root");
 TREE* current = add_node(root,"File 1",LEFT);
 current = add_node(current,"File 2",LEFT);
 current = add_node(root,"Folder 1",RIGHT);
 current = add_node(current,"File 11",LEFT);
 current = add_node(current,"File 12",LEFT);
  current = add_node(root->right,"Folder 2",RIGHT);
 current = add_node(current,"File 21",LEFT);
 show_tree();
 del_tree();
 root = NULL;
 return 0;
}
```

Данная функция сначала инициализирует глобальный указатель root как корень дерева и в качестве первого аргумента функции add_node() записывает NULL, что означает, что предыдущего объекта не существует. Затем вводится вспомогательный указатель current, с помощью которого выполняется создание двоичного дерева, описывающее файловую структуру (рис. 6). После создания

дерева вызывается функция show_tree() для его отображения на экран, затем оно удаляется и программа завершает свою работу.

Задание на лабораторную работу

1. Написать программу работы с бинарным деревом в соответствии с номером своего варианта.

Варианты заданий

Вариант	Задание на программирование бинарного дерева
1	Написать программу копирования одного бинарного дерева в другое
2	Написать программу копирования правых вершин одного бинарного дерева в другое
3	Написать программу копирования левых вершин одного бинарного дерева в другое
4	Написать программу обмена правых вершин одного бинарного дерева на соответствующие левые другого дерева
5	Написать программу замены одного бинарного дерева на другое
6	Написать программу копирования вершин, связанных с левым указателем корня дерева в соответствующие «правые» вершины
7	Написать программу обмена информацией между вершинами, связанными с правым и левым указателями корня дерева
8	Написать программу подсчета числа вершин в бинарном дереве
9	Написать программу подсчета левых вершин бинарного дерева
10	Написать программу подсчета правых вершин бинарного дерева

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером варианта, фамилией студента и группы.
2. Текст программы.
3. Результаты действия программы.
4. Выводы о полученных результатах работы программы.

Контрольные вопросы

1. Опишите структуру бинарного дерева.
2. Какой тип информации удобно представлять с помощью бинарных деревьев?
3. Объясните принцип работы рекуррентных функций для отображения и удаления элементов бинарного дерева.
4. Что является объектом бинарного дерева?
5. Что такое вершина бинарного дерева?
6. Чему равны указатели left и right вершины бинарного дерева?