Деревья с произвольным числом сыновей у каждого узла

Наряду с двоичными деревьями в программировании рассматриваются и произвольные деревья, каждая вершина которых может иметь произвольное число сыновей (см. рис.1).

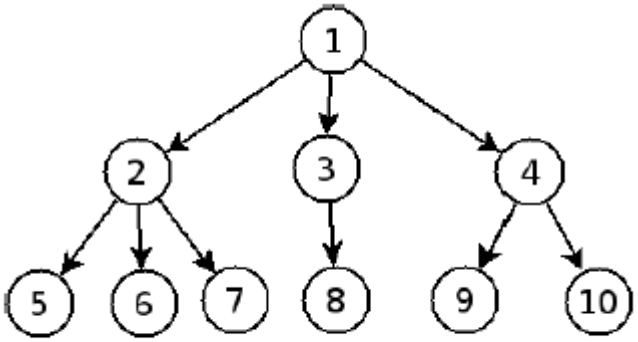


Рис. 1

Такие деревья обычно представляются теми же по структуре узлами, что и двоичные деревья, но смысл указателей меняется. Теперь один указатель у узла указывает на его первого (самого левого) сына, он называется указатель вниз, а другой – на соседнего брата справа, и он называется указатель вправо.

Если рисовать узлы дерева как прямоугольники, внутри которых содержатся поля, то дерево на рисунке 1 будет представлено структурой на рисунке 2.

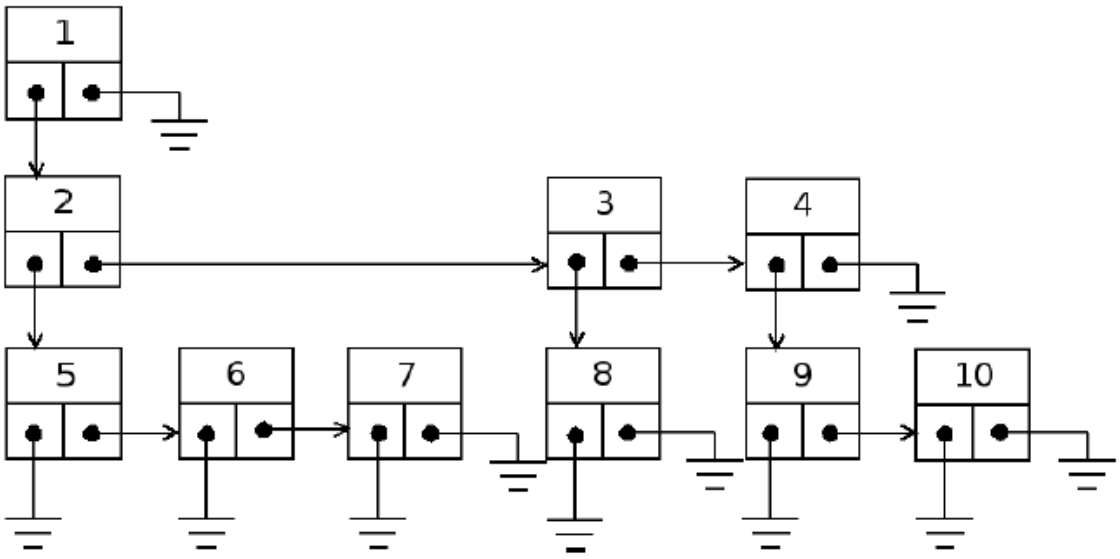


Рис. 2

Таким образом, набор братьев представлен по-существу связным списком; отличие от связного списка состоит в наличии дополнительного указателя вниз.

Такой способ трактовки указателей, содержащихся в узле, позволяет представлять не только деревья, но и леса, т. е. наборы деревьев (их корни считаются братьями).

Напишем структуру, описывающую узел дерева:

```
    template<class T>
    struct ANode
    {
    T data;
    ANode<T> *down, *right;
    ANode(T dd, ANode<T> *d = nullptr, ANode<T> *r = nullptr) :
data(dd), down(d), right(r) {}
    };
```

Замечание. Здесь узел обозначен ANode (A –arbitrary, поскольку число сыновей произвольно).

Дерево с произвольным числом потомков у каждого узла удобно печатать в том виде, в котором Windows показывает дерево папок и файлов в программах с графическим интерфейсом. Приведем функцию f_print, которая печатает такие деревья:

```
    template <class T>
    void f_print (ANode<T> *p, int d = 0)  // d - смещение
    {
         if (p == nullptr) return;
         for (int i = 0; i<d; i++)
              cout << ' ';
         cout << p->data << endl;
         f_print(p->down, d + 3);
         f_print(p->right, d);
    }
```

Создадим рассмотренное выше дерево (рис. 1) при помощи операции new и конструктора узла:

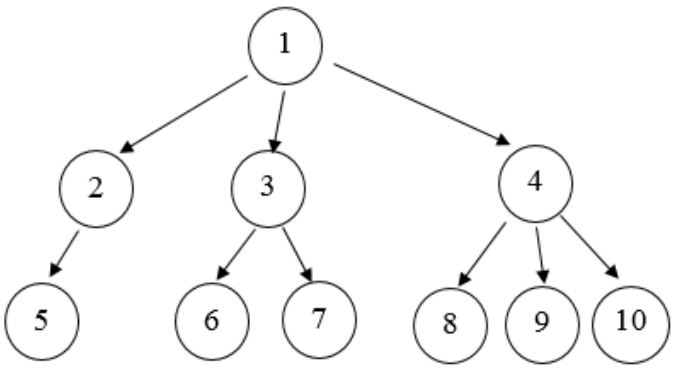
```
ANode <int> * p10 = new ANode <int> (10),
     *p9 = new ANode <int> (9, nullptr, p10),
     *p8 = new ANode <int> (8),
     *p7 = new ANode <int> (7),
     *p6 = new ANode <int> (6, nullptr, p7),
     *p5 = new ANode <int> (5, nullptr, p6),
     *p4 = new ANode <int> (4, p9),
     *p3 = new ANode <int> (3, p8, p4),
     *p2 = new ANode <int> (2, p5, p3),
     *p1 = new ANode <int> (1, p2);
```

Если применить эту функцию f_print к этому дереву (к указателю p1), будет напечатано следующее:

```
1
   2
      5
      6
      7
   3
      8
   4
      9
      10
```

Полный текст данной программы

Задание 1. Напишите программу, которая создает и печатает следующее дерево:



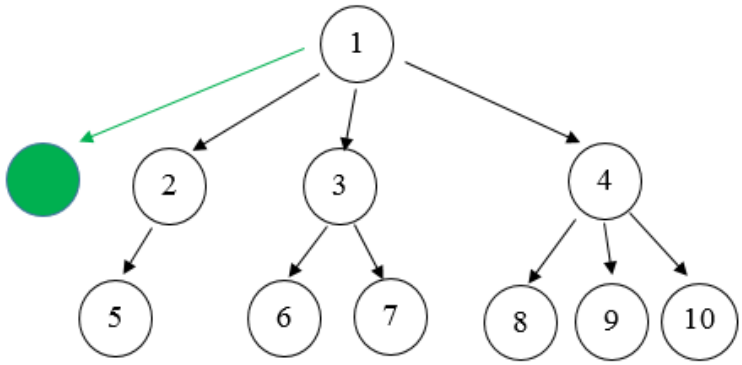
Пример 1. Функция count, возвращает число сыновей корня:

```
Template <class T>
int count (ANode<T> *p)
{
     int c = 0;
     if (p == nullptr) return c;
     p = p->down;
     while (p != nullptr)
     {
          c++;
          p = p->right;
     }
     return c;
}
```

Задание 2. Написать функцию, возвращающую число внуков корня.

Задание 3. Написать функцию, возвращающую указатель на первого сына корня с данными d1, у которого есть сын с данными d2 (d1 и d2 – параметры функции).

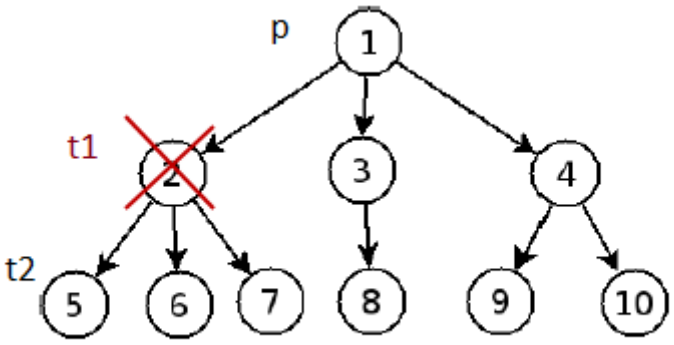
Пример 2. Функция add_first, добавляет в дерево нового сына корня (без потомков) в начало. После вставки новый сын корня должен быть первым.




```
template <class T>
void add_first (ANode<T> * p, T d)  //  d -  данные узла
{
     ANode<T> *t;
     t = new ANode<T> (d, nullptr, p->down);
     p->down = t;
}
```

Задание 4. Написать функцию, добавляющую нового сына корня (без потомков) в конец. После вставки новый сын корня должен быть последним.

Пример 3. Функция del_first, удаляет первого сына корня (его сыновья, будем предполагать, что они есть, становятся сыновьями корня перед бывшим вторым сыном корня).




```
template<class T>
void del_first(ANode<T> * p)
{
     ANode<T> *t, *t1, *t2;
     t1 = p->down;
     t2 = t1->down;
     t = t2;
     while (t->right != 0)
          t = t->right;

     t->right = t1->right;
     p->down = t2;
     delete t1;
}
```

Задание 5. Написать функцию, производящую один шаг уплощения (удаление всех сыновей корня, а не только первого, как в предыдущем примере; внуки корня становятся его сыновьями, причем их порядок сохраняется).

До этого момента мы использовали структуру ANode и функции, принимающие этот тип в качестве параметра. Далее, добавим структуру ATree, описывающую рассматриваемые деревья:

```
template<class T>
struct ATree
{
     ANode <T> * root;   // корень дерева
     ATree(ANode<T>*p) : root(p) {}
                        // конструктор по указателю на узел

      void print()   // метод печати
      {
          f_print(root);  // вызывает рекурсивную функцию
      }
};
```

Полный текст программы, использующей структуру ATree для дерева на рис.1

Задание 6. Используя структуру ATree, переделайте предыдущие задания (1-5), превратив функции в соответствующие методы.

Задание 7. Написать метод, надстраивающий еще один уровень. Указатель на корень после этого указывает на новый узел, самым левым сыном которого является старый корень. Кроме него, у нового корня есть еще 3 бездетных сыновей. Данные нового корня и его новых сыновей передаются в качестве параметров.

Задание 8. Написать метод, возвращающий указатель на узел, полученный из корня последовательными перемещениями вниз в сына с заданным номером (последовательность номеров хранится в массиве, который передается в качестве параметра).