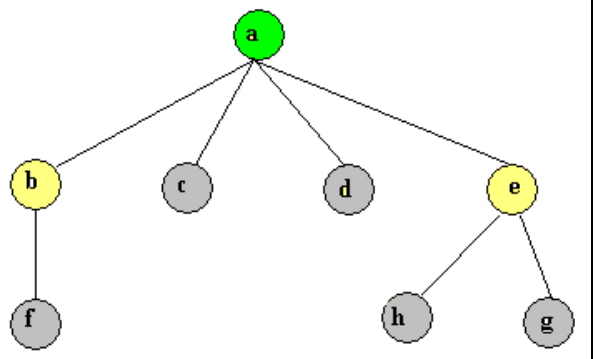
DRZEWA

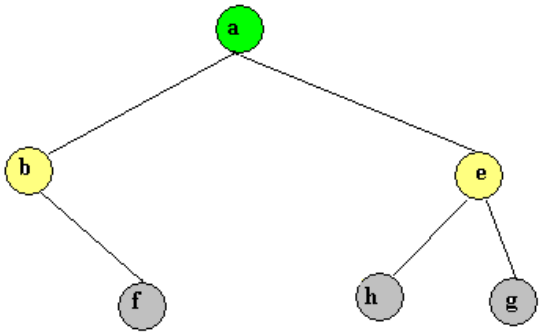
Drzewo jest zbiorem elementów zwanych wierzchołkami, spośród których jeden wyróżniony nazywany jest korzeniem. W zbiorze wierzchołków określona jest relacja bycia rodzicem, która nakłada hierarchiczną strukturę na ten zbiór.



Węzeł *a* jest korzeniem.
Węzły *b, c, d, e* są dziećmi węzła *a*
Węzeł *f* jest dzieckiem węzła *b*
Węzły *h* oraz *g są dziećmi węzła b*
Węzeł, który nie ma dzieci nazywamy liściem.
Węzły *f, c, d, h, g* są liśćmi

Drzewa binarne

Drzewo binarne to drzewo, które albo jest drzewem pustym, albo drzewem, w którym każdy węzeł posiada co najwyżej dwoje dzieci (lewe i prawe).



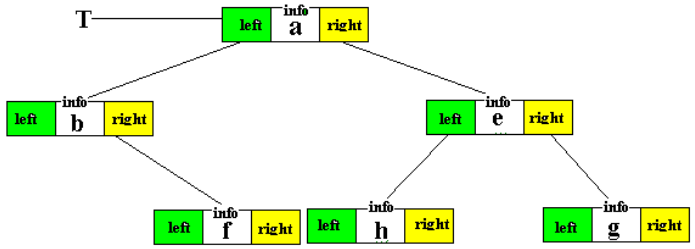
Reprezentacja wskaźnikowa drzewa binarnego:

```
typedef int Telement;

typedef struct Node
{
        Telement info;
        Node * left;
        Node * right;
};

typedef Node *PNode;
```

W tej reprezentacji drzewo jest wskaźnikiem do korzenia



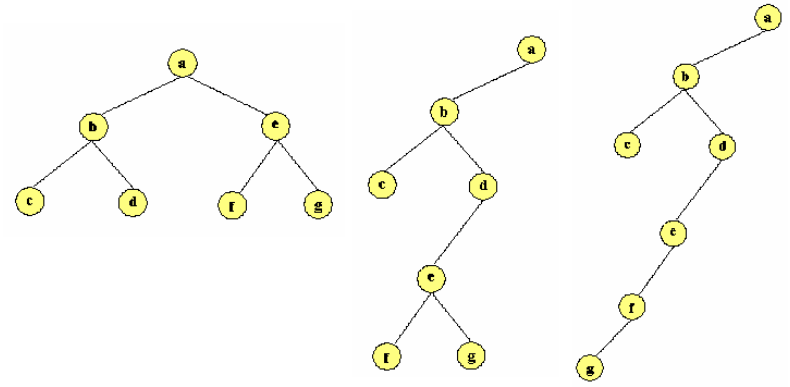

```
class BinaryTree
{
protected:
 PNode T;

public:
 BinaryTree();
 PNode PARENT(PNode n);
 PNode PARENT1(PNode n,PNode T);
 PNode LEFT_CHILD(PNode n);
 PNode RIGHT_CHILD(PNode n);
 Telement GetLabel(PNode n);
 PNode BuildTree1(int m);
 void BuildTree(int m);
 PNode ROOT();
 void MAKENULL(PNode p);
 void PreOrder(PNode t);
 void InOrder(PNode t);
 void PostOrder(PNode t);
};
```

Zad.1. Zaimplementuj metody klasy BinaryTree

1. **Konstruktor BinaryTree()** – tworzy puste drzewo (ustawia T na NULL)
2. **Operacja tworzenia drzewa binarnego:**

Załóżmy, że kolejnymi wartościami węzłów będą wartości podane przez użytkownika. Takie drzewa można zbudować na kilka sposobów.
Podając wartości a,b,c,d,e,f,g można zbudować drzewa postaci:



Będziemy chcieli, aby wysokość drzewa była jak najmniejsza.
(WYSOKOŚĆ drzewa – najdłuższa ścieżka od korzenia do liścia).
Drzewo pierwsze ma wysokość 2
Drzewo drugie ma wysokość 4
Drzewo trzecie ma wysokość 5

Sposób postępowania- Rozdzielenie tej samej liczby węzłów na prawo i na lewo dla każdego węzła w drzewie.
Funkcja **BuildTree(m) –** wywołuje BuildTree1(m) i wynik zapamiętuje w T.
Metoda pomocnicza **BuildTree1(m)** -rekurencyjna
0) jeżeli m =0 to zwróć wartość NULL
1)utwórz korzeń
2)zbuduj lewe poddrzewo z *m / 2* węzłami
3)zbuduj prawe poddrzewo z *m-(m / 2) -1* węzłami

UWAGA: poddrzewa lewe i prawe tworzymy rekurencyjnie – poprzez rekurencyjne wywołanie funkcji BuildTree1().

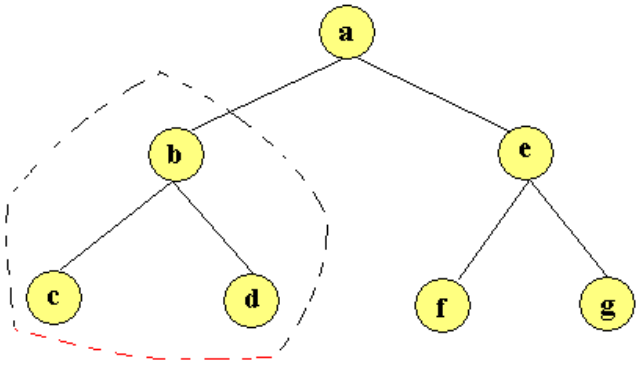
3. **GetLabel(n) –** zwraca etykietę węzła *n*
4. **ROOT() –** zwraca węzeł, który jest korzeniem lub NULL jeśli drzewo jest puste
5. **LEFT_CHILD(n) –** zwraca lewe dziecko węzła *n.*
6. **RIGHT_CHILD(n) –** zwraca prawe dziecko węzła *n.*
7. **PARENT(n) -**zwraca rodzica węzła n w drzewie T. Jeśli n jest korzeniem, to zwraca NULL. W przeciwnym wypadku wywołuje funkcję Parent1(n,T).
   Funkcja pomocnicza PARENT1(n,T) szuka rodzica węzła n w poddrzewie T.
   PARENT1 sprawdza, czy n jest lewym lub prawym dzieckiem T. Jeśli jest, zwraca T, jeśli nie, rekurencyjnie wywołuje Parent1 dla lewego i prawego poddrzewa i zwraca tą wartość, która jest różna od NULL.
8. **MAKENULL(n)-** zwalnia pamięć zajętą przez poddrzewo o korzeniu *n*
9. **Destruktor ~BinaryTree() –** wywołuje MAKENULL(T).

Istnieje wiele zadań, które wykonuje się na drzewach – najczęściej należy wykonać jakąś operację na każdym węźle drzewa.
W jakiej kolejności odwiedzać (przeglądać) węzły drzewa aby wykonać żądaną operację?

Istnieją trzy podstawowe sposoby przeglądania drzewa:

- preorder(porządek wzdłużny) – odwiedzamy korzeń przed poddrzewami
- inorder(porządek poprzeczny) – odwiedzamy lewe poddrzewo, korzeń, prawe poddrzewo
- postorder(porządek wsteczny) – odwiedzamy korzeń po poddrzewach

Preorder
Wypisujemy korzeń *a.*
Następnie traktujemy lewe poddrzewo jak drzewo i czynność powtarzamy dla tego poddrzewa – czyli wypisujemy korzeń – węzeł *b*
Następnie traktujemy lewe poddrzewo tego poddrzewa jak drzewo i wypisujemy jego korzeń – węzeł *c.* Węzeł *c* nie ma żadnych poddrzew. Gdy już mamy wypisane całe lewe poddrzewo węzła *b* to teraz traktujemy prawe poddrzewo węzła *b* jak drzewo – wypisujemy korzeń – węzeł *d*.
Gdy już mamy wypisane całe lewe poddrzewo węzła *a*, to traktujemy jego prawe poddrzewo jako drzewo i wypisujemy korzeń – węzeł *e*. itd…
Preorder: a,b,c,d,e,f,g



Zad.2.
Napisać funkcje Preorder(),Inorder() oraz Postorder wypisujące węzły drzewa w określonym porządku.

Testy:

- **Zbudować drzewo wczytując wartości:1,2,3,4,5,6,7,8,9**
- **Wypisać wartość korzenia, lewego dziecka i prawego dziecka korzenia**
- **Wypisać wartość węzła, który jest rodzicem węzła będącego prawym dzieckiem korzenia**
- **Wypisać wartość węzła, który jest rodzicem węzła będącego prawym dzieckiem lewego dziecka korzenia**
- **Wypisać węzły w porządku preorder, inorder, postorder**