問３　２分探索木に関する次の記述を読んで，設問に答えよ。

２分探索木とは，木に含まれる全てのノードがキー値をもち，各ノード N が次の二つの条件を満たす２分木のことである。ここで，重複したキー値をもつノードは存在しないものとする。

・N の左側の部分木にある全てのノードのキー値は，N のキー値よりも小さい。

・N の右側の部分木にある全てのノードのキー値は，N のキー値よりも大きい。

２分探索木の例を図１に示す。図中の数字はキー値を表している。

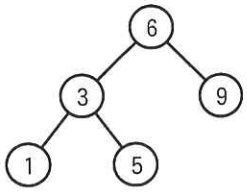

図１　２分探索木の例

２分探索木をプログラムで表現するために，ノードを表す構造体 Node を定義する。構造体 Node の構成要素を表１に示す。

表１　構造体 Node の構成要素

構成要素	説明
key	キー値
left	左側の子ノードへの参照
right	右側の子ノードへの参照

構造体 Node を新しく生成し，その構造体への参照を変数 p に代入する式を次のように書く。

```
p ← new Node(k)
```

ここで，引数 k は生成するノードのキー値であり，構成要素 key の初期値となる。構成要素 left 及び right は，参照するノードがないこと（以下，空のノードという）を表す NULL で初期化される。また，生成した p の各構成要素へのアクセスには“.”を用いる。例えば，キー値は p.key でアクセスする。

〔2 分探索木におけるノードの探索・挿入〕

キー値 k をもつノードの探索は次の手順で行う。

(1) 探索対象の 2 分探索木の根を参照する変数を t とする。

(2) t が空のノードであるかを調べる。

(2-1) t が空のノードであれば，探索失敗と判断して探索を終了する。

(2-2) t が空のノードでなければ，t のキー値 t.key と k を比較する。

・t.key = k の場合，探索成功と判断して探索を終了する。

・t.key > k の場合，t の左側の子ノードを新たな t として(2)から処理を行う。

・t.key < k の場合，t の右側の子ノードを新たな t として(2)から処理を行う。

キー値 k をもつノード K の挿入は，探索と同様の手順で根から順にたどっていき，空のノードが見つかった位置にノード K を追加することで行う。ただし，キー値 k と同じキー値をもつノードが既に 2 分探索木中に存在するときは何もしない。

これらの手順によって探索を行う関数 search のプログラムを図 2 に，挿入を行う関数 insert のプログラムを図 3 に示す。関数 search は，探索に成功した場合は見つかったノードへの参照を返し，失敗した場合は NULL を返す。関数 insert は，得られた木の根への参照を返す。

```
// t が参照するノードを根とする木から
// キー値が k であるノードを探索する
function search(t, k)
  if(t が NULL と等しい)
    return NULL
  elseif(t.key が k と等しい)
    return t
  elseif(t.key が k より大きい)
    return search(t.left, k)
  else // t.key が k より小さい場合
    return search(t.right, k)
  endif
endfunction
```

図 2 探索を行う関数 search のプログラム

```
// t が参照するノードを根とする木に
// キー値が k であるノードを挿入する
function insert(t, k)
  if(t が NULL と等しい)
    t ← new Node(k)
  elseif(t.key が k より大きい)
    t.left ← insert(t.left, k)
  elseif(t.key が k より小さい)
    t.right ← insert(t.right, k)
  endif
  return t
endfunction
```

図 3 挿入を行う関数 insert のプログラム

関数 search を用いてノードの総数が n 個の 2 分探索木を探索するとき，探索に掛かる最悪の場合の時間計算量（以下，最悪時間計算量という）は $O($ [ア] $)$ であ

る。これは葉を除く全てのノードについて左右のどちらかにだけ子ノードが存在する場合である。一方で，葉を除く全てのノードに左右両方の子ノードが存在し，また，全ての葉の深さが等しい完全な 2 分探索木であれば，最悪時間計算量は $O($ [イ] $)$となる。したがって，高速に探索するためには，なるべく左右両方の子ノードが存在するように配置して，高さができるだけ低くなるように構成した木であることが望ましい。このような木のことを平衡 2 分探索木という。

〔2 分探索木における回転操作〕

2 分探索木中のノード X と X の左側の子ノード Y について，X を Y の右側の子に，元の Y の右側の部分木を X の左側の部分木にする変形操作を右回転といい，逆の操作を左回転という。回転操作後も 2 分探索木の条件は維持される。木の回転の様子を図 4 に示す。ここで，t_1～t_3は部分木を表している。また，根から t_1～t_3の最も深いノードまでの深さを，図 4(a)では d_1～d_3，図 4(b)では d_1'～d_3'でそれぞれ表している。ここで，$d_1' = d_1 - 1$，$d_2' = d_2$，$d_3' = d_3 + 1$，が成り立つ。

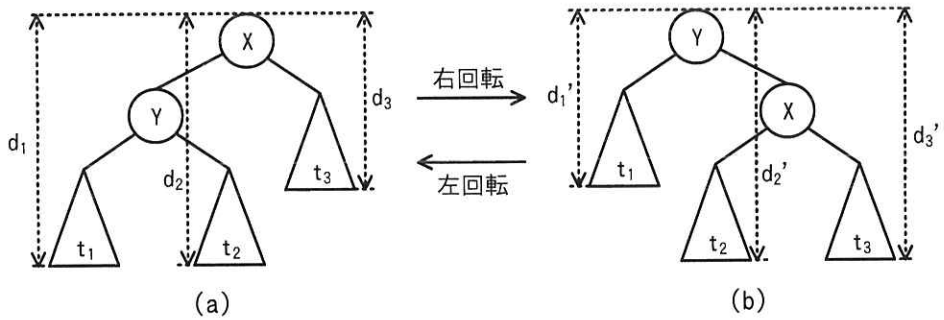

図 4　木の回転の様子

右回転を行う関数 rotateR のプログラムを図 5 に，左回転を行う関数 rotateL のプログラムを図 6 に示す。これらの関数は，回転した結果として得られた木の根への参照を返す。

```
// t が参照するノードを根とする木に対して
// 右回転を行う
function rotateR(t)
  a ← t.left
  b ← a.right
  a.right ← t
  t.left ← b
  return a
endfunction
```

図5　右回転を行う関数 rotateR のプログラム

```
// t が参照するノードを根とする木に対して
// 左回転を行う
function rotateL(t)
  a ← t.right
  b ← a.left
  a.left ← t
  t.right ← b
  return a
endfunction
```

図6　左回転を行う関数 rotateL のプログラム

〔回転操作を利用した平衡2分探索木の構成〕

全てのノードについて左右の部分木の高さの差が1以下という条件（以下，条件 Bal という）を考える。条件 Bal を満たす場合，完全ではないときでも比較的左右均等にノードが配置された木になる。

条件 Bal を満たす2分探索木 W に対して図3の関数 insert を用いてノードを挿入した2分探索木を W' とすると，ノードが挿入される位置によっては左右の部分木の高さの差が2になるノードが生じるので，W' は条件 Bal を満たさなくなることがある。その場合，挿入したノードから根まで，親をたどった各ノード T に対して順に次の手順を適用することで，条件 Bal を満たすように W' を変形することができる。

(1)　T の左側の部分木の高さが T の右側の部分木の高さより2大きい場合

T を根とする部分木に対して右回転を行う。ただし，T の左側の子ノード U について，U の右側の部分木の方が U の左側の部分木よりも高い場合は，先に U を根とする部分木に対して左回転を行う。

(2)　T の右側の部分木の高さが T の左側の部分木の高さより2大きい場合

T を根とする部分木に対して左回転を行う。ただし，T の右側の子ノード V について，V の左側の部分木の方が V の右側の部分木よりも高い場合は，先に V を根とする部分木に対して右回転を行う。

この手順(1)，(2)によって木を変形する関数 balance のプログラムを図7に，関数 balance を適用するように関数 insert を修正した関数 insertB のプログラムを図8に示す。ここで，関数 height は，引数で与えられたノードを根とする木の高さを返す関数である。関数 balance は，変形の結果として得られた木の根への参照を返す。

```
// t が参照するノードを根とする木を
// 条件 Bal を満たすように変形する
function balance(t)
  h1 ← height(t.left) - height(t.right)
  if( [ウ] )
    h2 ← [エ]
    if(h2 が 0 より大きい)
      t.left ← rotateL(t.left)
    endif
    t ← rotateR(t)
  elseif( [オ] )
    h3 ← [カ]
    if(h3 が 0 より大きい)
      t.right ← rotateR(t.right)
    endif
    t ← rotateL(t)
  endif
  return t
endfunction
```

図 7　関数 balance のプログラム

```
// t が参照するノードを根とする木に
// キー値が k であるノードを挿入する
function insertB(t, k)
  if(t が NULL と等しい)
    t ← new Node(k)
  elseif(t.key が k より大きい)
    t.left ← insertB(t.left, k)
  elseif(t.key が k より小さい)
    t.right ← insertB(t.right, k)
  endif
  t ← balance(t)    // 追加
  return t
endfunction
```

図 8　関数 insertB のプログラム

条件 Bal を満たすノードの総数が n 個の 2 分探索木に対して関数 insertB を実行した場合，挿入に掛かる最悪時間計算量は $O($ [キ] $)$ となる。

設問 1　本文中の [ア]，[イ] に入れる適切な字句を答えよ。

設問 2　〔回転操作を利用した平衡 2 分探索木の構成〕について答えよ。

(1)　図 7 中の [ウ] ～ [カ] に入れる適切な字句を答えよ。

(2)　図 1 の 2 分探索木の根を参照する変数を r としたとき，次の処理を行うことで生成される 2 分探索木を図示せよ。2 分探索木は図 1 に倣って表現すること。

```
insertB(insertB(r, 4), 8)
```

(3)　本文中の [キ] に入れる適切な字句を答えよ。なお，図 7 中の関数 height の処理時間は無視できるものとする。