問３　２分探索木に関する次の記述を読んで，設問 1～4 に答えよ。

２分探索木とは，全てのノード N に対して，次の条件が成立している 2 分木のことである。

・N の左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。

ここで，ノードのキー値は自然数で重複しないものとする。2 分探索木の例を図 1 に示す。図中の数はキー値を表している。

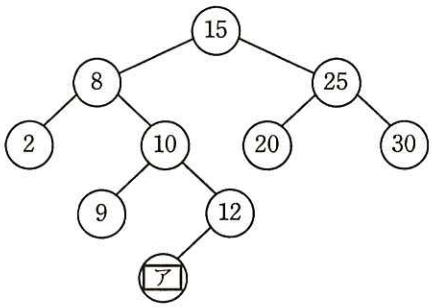

図１　２分探索木の例

２分探索木を実現するために，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	左子ノードへの参照
right	右子ノードへの参照

構造体の実体を生成するためには，次のように書く。

```
new Node(key)
```

生成した構造体への参照が戻り値となる。構造体の構成要素のうち，key は引数 key の値で初期化され，left と right は null で初期化される。

変数 p が参照するノードをノード p という。ノードを参照する変数からそのノー

ドの構成要素へのアクセスには“.”を用いる。例えば，ノード p のキー値には，p.key でアクセスできる。

なお，変数 p の値が null の場合，木は空である。

〔2 分探索木でのノードの探索〕

与えられたキー値をもつノードを探索する場合，親から子の方向へ，木を順次たどりながら探索を行う。

探索する 2 分探索木にノードがない場合は，目的のノードが見つからず，探索は失敗と判断して終了する。探索する 2 分探索木にノードがある場合は，与えられたキー値と木の根のキー値を比較し，等しければ，目的のノードが見つかったので探索は成功と判断して終了する。与えられたキー値の方が小さければ左部分木に，大きければ右部分木に移動する。移動先の部分木でも同様に探索を続ける。

この手順によって探索を行う関数 search のプログラムを図 2 に示す。このプログラムでは，探索が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。

```
//ノード p を根とする 2 分探索木から，キー値が k であるノードを探索する
function search(k, p)
    if(p と null が等しい)
        return null                  //探索失敗
    elseif(k と p.key が等しい)
        return p                     //探索成功
    elseif( イ )
        return search(k, p.left)     //左部分木を探索する
    else
        return search(k, p.right)    //右部分木を探索する
    endif
endfunction
```

図 2　関数 search のプログラム

〔2 分探索木へのノードの挿入〕

2 分探索木にノードを挿入する場合，探索と同様に，親から子の方向へ，木を順次たどりながら，適切な位置にノードを挿入する。

挿入する 2 分探索木にノードがない場合は，挿入するキー値のノードを作成する。

挿入する 2 分探索木にノードがある場合は，挿入するキー値と木の根のキー値を比較し，挿入するキー値の方が小さければ左部分木に，大きければ右部分木に移動する。移動先の部分木でも同様の処理を続ける。

この手順によって挿入を行う関数 addNode のプログラムを図 3 に示す。このプログラムでは，挿入の結果として得られた 2 分探索木の根のノードへの参照を返す。ただし，このプログラムは，挿入するキー値と同じキー値をもつノードが 2 分探索木に既に存在するときは何もしない。

```
//ノード p を根とする 2 分探索木に，キー値が k であるノードを挿入する
function addNode(k, p)
    if(p と null が等しい)
        p ← [ ウ ]                          //ノードを生成する
    elseif(k と p.key が等しくない)
        if( [ イ ] )
            p.left ← addNode(k, p.left)     //左部分木に移動し挿入を続ける
        else
            p.right ← addNode(k, p.right)   //右部分木に移動し挿入を続ける
        endif
    endif
    [ エ ]
endfunction
```

図 3　関数 addNode のプログラム

〔2 分探索木からのノードの削除〕

2 分探索木から，あるキー値をもつノードを削除する場合，次の(1)～(3)の手順を行う。

(1)　2 分探索木にノードがない場合は，何もしないで処理を終了する。

(2)　削除するキー値と木の根のキー値を比較し，削除するキー値の方が小さければ左部分木に，大きければ右部分木に移動する。移動先の部分木でも同様の処理を続ける。

(3)　削除するキー値と木の根のキー値が等しい場合，削除するキー値をもつノードを削除するため，次の(3-1)～(3-3)を実行する。

(3-1)　削除するノードが子ノードをもたない場合，そのノードを削除する。

(3-2)　削除するノードが子ノードを一つだけもつ場合，削除するノードの位置にその子ノードを置く。

(3-3) 削除するノードが左右両方に子ノードをもつ場合，削除するノードの左部分木の中で最大のキー値をもつノードを左部分木から取り除き，削除するノードの位置に置く。

この手順を使って 2 分探索木からノードの削除を行う関数 removeNode のプログラムを図 4 に示す。このプログラムでは，削除した後の 2 分探索木の根のノードへの参照を返す。ただし，このプログラムは，削除するキー値をもつノードが 2 分探索木に存在しないときは何もしない。

図 4 中の関数 extractMaxNode は，引数で指定されたノードを根とする 2 分探索木の中で最大のキー値をもつノードを木から削除し，削除されたノードへの参照を大域変数 extractedNode に設定した上で，削除した後の 2 分探索木の根のノードへの参照を返す。関数 extractMaxNode のプログラムを図 5 に示す。

```
//ノード p を根とする 2 分探索木から，キー値が k であるノードを削除する
function removeNode(k, p)
    if(p と null が等しくない)
        if(k が p.key より小さい)
            p.left ← removeNode(k, p.left)
        elseif(k が p.key より大きい)
            p.right ← removeNode(k, p.right)
        else
            if(p.left と null が等しい かつ p.right と null が等しい)
                p ← null                              //ノードを削除する
            elseif( オ と null が等しい)
                p ← p.right                           //右部分木を置く
            elseif( カ と null が等しい)
                p ← p.left                            //左部分木を置く
            else                                      //左部分木の中の最大ノードを置く
                p.left ← extractMaxNode(p.left)
                r ← extractedNode
                r.left ← p.left
                r.right ← p.right
                キ
            endif
        endif
    endif
    return p
endfunction
```

図 4　関数 removeNode のプログラム

```
//ノード p を根とする 2 分探索木から，最大のキー値をもつノードを削除し，削除された
ノードへの参照を大域変数に格納する
function extractMaxNode(p)
    if(p.right と null が等しい)
        extractedNode ← p
        p ← p.left
    else
        p.right ← extractMaxNode(p.right)
    endif
    return p
endfunction
```

図 5　関数 extractMaxNode のプログラム

〔2 分探索木の計算量〕

2 分探索木における計算量は，木の高さに依存する。図 2 の関数 search を使って n 個のノードから成る 2 分探索木を探索する場合，想定される最大の計算量は，$O($［　ク　］$)$である。木構造が完全 2 分木であれば，その計算量は最大でも $O($［　ケ　］$)$である。

設問 1　図 1 中の［　ア　］に入れる適切な数を答えよ。

設問 2　図 2～4 中の［　イ　］～［　キ　］に入れる適切な字句を答えよ。

設問 3　本文中の［　ク　］，［　ケ　］に入れる適切な字句を答えよ。

設問 4　次の順でキー値の挿入と削除を行った後でノード q を根とする 2 分探索木を答えよ。2 分探索木は，図 1 の例に倣って表現すること。

```
q ← null
q ← addNode(5, q)          //5 を挿入
q ← addNode(2, q)          //2 を挿入
q ← addNode(7, q)          //7 を挿入
q ← addNode(1, q)          //1 を挿入
q ← addNode(8, q)          //8 を挿入
q ← addNode(4, q)          //4 を挿入
q ← addNode(3, q)          //3 を挿入
q ← addNode(12, q)         //12 を挿入
q ← removeNode(5, q)       //5 を削除
q ← removeNode(7, q)       //7 を削除
```