問題

入力された英単語(すべて小文字と考えてよい)を、アルファベット昇順にリスト構造で保持するプログラムを作成したい。但し、リストにある単語と同じ単語が入力された場合には、各ノードのデータメンバである単語の出現回数を増やすことで対応する。また、指定した出現回数の単語がすべて削除できるようにしたい。このため、以下のBinTreeクラスを用いてプログラムを完成させなさい。

問題(続き)

```
class BinTree {
private:
    class Node {                    // 内部クラス
    public:
        string data;                // ノードの値(単語の名前)
        int count;                  // 単語の出現回数
        Node *left;                 // 左のnextを指すポインタ
        Node *right;                // 右のnextを指すポインタ
        Node(string a="",int d=1,Node *b=NULL, Node *c=NULL){  //コンストラクタ
            data=a; count=d; left=b; right=c;
        }
        ~Node(){ cout << data << " is released.¥n"; }                  // デストラクタ
        void printNode(){ cout << data << "[" << count << "] "; }      // データの出力
    };
    Node *root;                                                        // 二分木の一番上のノードを指すポインタ
    void traverse(Node *rp);                                           // 二分木rpを出力
    Node* addNode(Node *rp, Node *node);                               // 二分木rpにnodeを追加
    Node* delNode(Node *rp, int x);                                    // 二分木rpから出現回数がxの単語を全て削除
    void clearNode(Node *rp);                                          // 二分木rp以降を削除
public:
    BinTree( ){ root=NULL;}                                            // 引数なしコンストラクタ
    BinTree(string*, string*);                                         // 引数ありコンストラクタ
    ~BinTree(){ clear();}                                              // デストラクタ
    void printTree(){ traverse(root); }                                // 二分木全体をアルファベット順に表示
    void insert(string x){                                             // 二分木にデータxを追加
        Node *np=new Node(x);                                          // データxを持つノードを作成
        root=addNode(root, np);                                        // 二分木rootにノードを追加
    }
    void remove(int x) { root = delNode(root, x); }                    // 二分木から出現回数がxの単語を全て削除
    void clear(){ clearNode(root); root=NULL;}                         // 二分木から全データを削除
};
```

問題(続き)

mainは以下のプログラムを使用してください。

```
int main(int argc, char *argv[]){

    string a[]={ "apple", "apple", "banana", "peach", "banana", "peach", "banana", "peach", "melon", "melon", "lemon", "orange",
      "watermelon"};
    string newword;                                          // リストに追加する単語
    int n;                                                   // 頻度がnの場合削除する
    char select;                                             // select:メニュー項目の文字
    BinTree bt(a, a+13);                                     // 配列aと同じ要素を持つリストを作る

    // メニューを表示して対応する処理を行う
    cout << "¥nMenu[I:Insert, R:Remove, P:Print, Q:Quit]";
    while( (cout << "¥n  Select I/R/S/P/C/Q-->") && (cin >> select) ){
        switch(select){
            case 'I':                                        // リストへ新規ノードの追加
            case 'i': cout << "Input a data-->"; cin >> newword;    bt.insert(newword);    break;
            case 'R':                                        // リストから指定ノードを削除
            case 'r': cout << "Remove a data-->"; cin >> n;         bt.remove(n);          break;
            case 'P': bt.printTree(); cout << "¥n";  break;  // リストの全データを表示
            case 'Q':                                        // プログラムを終了
            case 'q': break;
            default: continue;
        }
        if( (select=='Q') || (select=='q') ){ break;}
    }

    return 0;
}
```

実行例

```
comsv001% ./a.out
apple is released.
banana is released.
peach is released.
banana is released.
peach is released.
melon is released.

Menu[I:Insert, R:Remove, P:Print, Q:Quit]
  Select I/R/S/P/C/Q-->P
apple[2] banana[3] lemon[1] melon[2] orange[1] peach[3] watermelon[1]

  Select I/R/S/P/C/Q-->I
Input a data-->apple
apple is released.

  Select I/R/S/P/C/Q-->P
apple[3] banana[3] lemon[1] melon[2] orange[1] peach[3] watermelon[1]

  Select I/R/S/P/C/Q-->R
Remove a data-->3
apple is released.
banana is released.
peach is released.

  Select I/R/S/P/C/Q-->P
lemon[1] melon[2] orange[1] watermelon[1]

  Select I/R/S/P/C/Q-->Q
lemon is released.
orange is released.
melon is released.
watermelon is released.
comsv001%
```

解答

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>

using namespace std;

class BinTree {
private:
    class Node {                // 内部クラス
    public:
        string data;            // ノードの値(単語の名前)
        int count;              // 単語の出現回数
        Node *left;             // 左のnextを指すポインタ
        Node *right;            // 右のnextを指すポインタ
        Node(string a="",int d=1,Node *b=NULL, Node *c=NULL){  //コンストラクタ
            data=a; count=d; left=b; right=c;
        }
        ~Node(){ cout << data << " is released.¥n"; }                  // デストラクタ
        void printNode(){ cout << data << "[" << count << "] "; }      // データの出力
    };
    Node *root;                                                        // 二分木の一番上のノードを指すポインタ
    void traverse(Node *rp);                                           // 二分木rpを出力
    Node* addNode(Node *rp, Node *node);                               // 二分木rpにnodeを追加
    Node* delNode(Node *rp, int x);                                    // 二分木rpから出現回数がxの単語を全て削除
    void clearNode(Node *rp);                                          // 二分木rp以降を削除
public:
    BinTree( ){ root=NULL;}                                            // 引数なしコンストラクタ
    BinTree(string*, string*);                                         // 引数ありコンストラクタ
    ~BinTree(){ clear();}                                              // デストラクタ
    void printTree(){ traverse(root); }                                // 二分木全体をアルファベット順に表示
    void insert(string x){                                             // 二分木にデータxを追加
        Node *np=new Node(x);                                          // データxを持つノードを作成
        root=addNode(root, np);                                        // 二分木rootにノードを追加
    }
    void remove(int x) { root = delNode(root, x); }                    // 二分木から出現回数がxの単語を全て削除
    void clear(){ clearNode(root); root=NULL;}                         // 二分木から全データを削除
};
```

```
BinTree::BinTree(string *begin, string *end){
    root=NULL;                                  //rootをNULLで初期化
    for(string *p=begin; p !=end; p++){         //stringのポインタpがbeginからend-1まで動く
        insert(*p);                             //insertをコールすると、root=addNode(root, np)となる
    }
}

void BinTree::traverse(Node *rp) {
    if (rp == NULL){                            // rpがNULLである場合
        return;                                 // 何もせず戻る
    }else{                                      // rpがNULLでない場合
        traverse(rp->left);                     // 先に左を再帰呼び出し
        rp->printNode();                        // 自分のノードのデータを出力
        traverse(rp->right);                    // 後で右のノードを再帰呼び出し
        return;
    }
}

BinTree::Node* BinTree::addNode(Node *rp, Node *node) {
    if(rp==NULL){                               // rpがNULLである場合
        return node;                            // nodeを戻す
    }else if(node==NULL){                       // nodeがNULLである場合
        return rp;                              // rpを戻す
    }else{                                      // rpもnodeもNULLでない場合
        if (node->data < rp->data) {            // rp->dataよりも小さい場合には、rpの左側の処理
            rp->left=addNode(rp->left, node);   // 左の木にnodeを追加し、戻り値をrp->leftに格納
        } else if (node->data > rp->data){      // rp->dataよりも大きい場合には、rpの右側の処理
            rp->right=addNode(rp->right, node); // 右の木にnodeを追加し、戻り値をrp->rightに格納
        } else {                                // rp->dataと一致
            rp->count++;                        // 出現回数を1増やし
            delete node;                        // 新しく作成したノードは削除してしまう
        }
        return rp;                              // 追加する場所でない場合は、そのままrpを戻す
    }
}
```

```
BinTree::Node* BinTree::delNode(Node *rp, int x) {
    if(rp==NULL){                          // rpがNULLである場合
       return NULL;                        // NULLを戻す
    }else if(x == rp->count) {             // rpの指しているノードが削除対象の場合
       Node *lf=rp->left;                  // 左を仮置き
       Node *rt=rp->right;                 // 右を仮置き
       delete rp;                          // 削除対象を削除
       lf=delNode(lf, x);                  // delNodeを再帰呼び出し、戻り値を左側に格納
       rt=delNode(rt, x);                  // delNodeを再帰呼び出し、戻り値を右側に格納
       return addNode(rt, lf);             // 右側の二分木に左側の二分木を追加した戻り値を戻す
    }else{                                 // rpの指しているノードが削除対象でない場合
       rp->left=delNode(rp->left, x);      // delNodeを再帰呼び出し、戻り値を左側に格納
       rp->right=delNode(rp->right, x);    // delNodeを再帰呼び出し、戻り値を右側に格納
       return rp;                          // 削除対象でない場合は、そのままrpを戻す
    }
}

void BinTree::clearNode(Node *rp){
    if(rp==NULL){                          //末尾まで到達の場合、何もしない
        return;
    }else{                                 //ノードがある場合
        clearNode(rp->left);               //left以降を削除
        clearNode(rp->right);              //right以降を削除
        delete rp;                         //ノードを削除
        return;
    }
}
```

```
int main(int argc, char *argv[]){

    string a[]={"apple", "apple", "banana", "peach", "banana", "peach", "banana", "peach", "melon", "melon", "lemon",
      "orange", "watermelon"};
    string newword;                                                    // リストに追加する単語
    int n;                                                             // 頻度がnの場合削除する
    char select;                                                       // select:メニュー項目の文字
    BinTree bt(a, a+13);                                               // 配列aと同じ要素を持つリストを作る

    // メニューを表示して対応する処理を行う
    cout << "¥nMenu[I:Insert, R:Remove, P:Print, Q:Quit]";
    while( (cout << "¥n  Select I/R/S/P/C/Q-->") && (cin >> select) ){
        switch(select){
            case 'I':                                                  // リストへ新規ノードの追加
            case 'i': cout << "Input a data-->"; cin >> newword;       bt.insert(newword);          break;
            case 'R':                                                  // リストから指定ノードを削除
            case 'r': cout << "Remove a data-->"; cin >> n;            bt.remove(n);       break;
            case 'P': bt.printTree();       cout << "¥n";  break;      // リストの全データを表示
            case 'Q':                                                  // プログラムを終了
            case 'q': break;
            default: continue;
        }
        if( (select=='Q') || (select=='q') ){ break;}
    }

    return 0;
}
```