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SortedLinkedList クラスを用いてプログラムを完成させなさい。

```
class SortedLinkedList{
private:
    class Node{
    public:
        string data;                // ノードの値
        int hindo;                  // 頻度
        Node *next;                 // 次のノードへのポインタ
        Node(string word="", int num=0, Node *n=NULL){ data=word; hindo=num; next=n; }      // コンストラクタ
        ~Node(){ cout << data << " is released.¥n"; }                                       // デストラクタ
    };
    Node *head;                                                     // リストの先頭
    void print(Node *pos) const;                                    // リストのpos以降のデータを出力
    Node* insertNode(Node *pos, string newword);                    // リストのpos以降にnewwordを昇順に追加
    Node* removeNode(Node *pos, int n);                             // リストのpos以降の頻度nの単語を削除
    void clearNode(Node *pos);                                      // リストのpos以降を削除
public:
    SortedLinkedList(){ head=NULL; }                                // 空リストの生成
    SortedLinkedList(string*, string*);                             // int配列の内容でリストを初期化
    ~SortedLinkedList(){ clear();}                                  // デストラクタ
    void printAll() const { print(head); }                          // リストの全データを出力
    void insert(string newword){ head=insertNode(head, newword);}   // リストにnewwordを追加
    void remove(int n){ head=removeNode(head, n);}                  // リストから頻度がnの単語を削除
    void clear(){ clearNode(head); head=NULL;}                      // リスト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
    SortedLinkedList ichain(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;    ichain.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;         ichain.remove(n);      break;
            case 'P': ichain.printAll();             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;
}
```

問題(続き)

＜実行例＞

```
comsv001% ./a.out

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)->END_OF_DATA

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

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

  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)->END_OF_DATA

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

解答

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

using namespace std;

class SortedLinkedList{
private:
    class Node{
    public:
        string data;              // ノードの値
        int hindo;                // 頻度
        Node *next;               // 次のノードへのポインタ
        Node(string word="", int num=0, Node *n=NULL){ data=word; hindo=num; next=n; }      // コンストラクタ
        ~Node(){ cout << data << " is released.¥n"; }                                       // デストラクタ
    };
    Node *head;                                                                  // リストの先頭
    void print(Node *pos) const;                                                 // リストのpos以降のデータを出力
    Node* insertNode(Node *pos, string newword);                                 // リストのpos以降にnewwordを昇順に追加
    Node* removeNode(Node *pos, int n);                                          // リストのpos以降の頻度nの単語を削除
    void clearNode(Node *pos);                                                   // リストのpos以降を削除
public:
    SortedLinkedList(){ head=NULL; }                                             // 空リストの生成
    SortedLinkedList(string*, string*);                                          // int配列の内容でリストを初期化
    ~SortedLinkedList(){ clear();}                                               // デストラクタ
    void printAll() const { print(head); }                                       // リストの全データを出力
    void insert(string newword){ head=insertNode(head, newword);}                // リストにnewwordを追加
    void remove(int n){ head=removeNode(head, n);}                               // リストから頻度がnの単語を削除
    void clear(){ clearNode(head); head=NULL;}                                   // リストから全データを削除
};

SortedLinkedList::SortedLinkedList(string *begin, string *end){
    head=NULL;                                                                   //head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、head=insertNode(head, *p)となる
    }
}
```

```
void SortedLinkedList::print(SortedLinkedList::Node *pointer) const{
    if(pointer==NULL){                                          //pointerがNULLの場合
        cout << "END_OF_DATA¥n";                                //END_OF_DATAを出力
    }else{                                                      //pointerがノードを指している場合
        cout << pointer->data << "(" << pointer->hindo << ")" << "->";   //pointerの指すノードのdataを出力
        print(pointer->next);                                   //再帰的にpointer->next以降の処理を呼び出す
    }
}

SortedLinkedList::Node* SortedLinkedList::insertNode(SortedLinkedList::Node *pointer, string newdata){
    if(pointer==NULL){                                          //末尾に追加する場合
        return new Node(newdata, 1, NULL);//ノードのメモリを1個とって、newdataを格納し、そのノードのポインタを返却
    }else if( (pointer->data)==newdata ){                       //pointerの手前にノードを追加する場合
        pointer->hindo++;
        return pointer;                     //ノードのメモリを1個とって、newdataを格納し、そのノードのポインタを返却
    }else if( (pointer->data)>newdata ){                        //pointerの手前にノードを追加する場合
        return new Node(newdata, 1, pointer);//ノードのメモリを1個とって、newdataを格納し、そのノードのポインタを返却
    }else{                                                      //追加する場所ではない場合
        pointer->next=insertNode(pointer->next, newdata);//next以降の追加する所を再帰的に探索。戻り値をnextに代入
        return pointer;                                         //追加する場所でなければ、引数のpointerを返却する
    }
}

SortedLinkedList::Node* SortedLinkedList::removeNode(SortedLinkedList::Node *pointer, int num){
    if(pointer==NULL){                                          //末尾まで到達の場合
        return NULL;                                            //NULLを返す
    }else if(pointer->hindo==num){                              //pointerの指しているノードが削除対象の場合
        Node *nextpointer=pointer->next;                        //nextを仮置きする
        delete pointer;                                         //削除対象を削除
        return removeNode(nextpointer, num);                    //next以降の削除する所を再帰的に探索。戻り値を戻す。
    }else{                                                      //削除対象ではない場合
        pointer->next=removeNode(pointer->next, num);           //next以降の削除する所を再帰的に探索。戻り値をnextに代入
        return pointer;                                         //削除対象でなければ、引数のpointerを返却する
    }
}
```

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

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
    SortedLinkedList ichain(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;      ichain.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;          ichain.remove(n);   break;
            case 'P': ichain.printAll();     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;
}
```