リスト構造(Linked List)
再帰版

世木寛之

SortedLinkedListを再帰で実装する

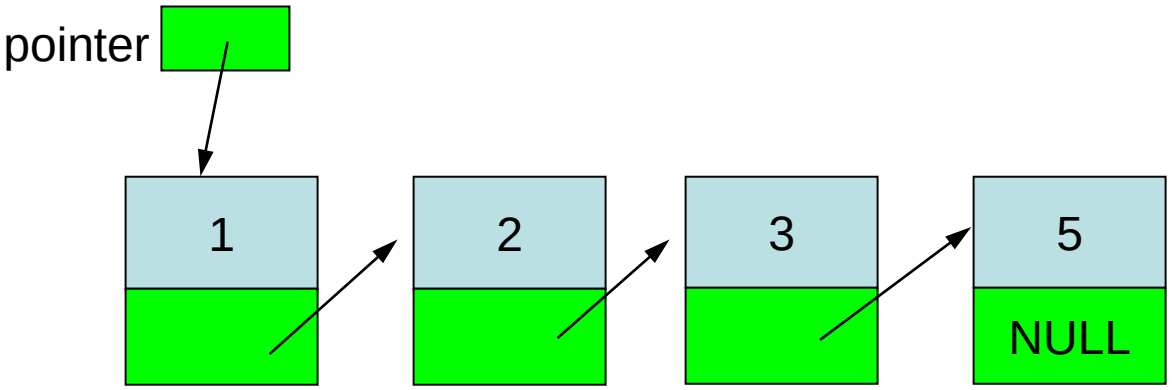
メンバ関数printAll()、size()、insert()、remove()
clear()の動作を再帰関数で定義する。

```
class SortedLinkedList{
private:
    class Node{
    public:
        int  data;                                                  // ノードの値
        Node *next;                                                 // 次のノードへのポインタ
        Node(int num=0, Node *n=NULL){ data = 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
    int  count(Node *pos) const;                    // リストのpos以降のデータ数を返却
    Node* insertNode(Node *pos, int newdata);       // リストのpos以降にnewdataを昇順に追加
    Node* removeNode(Node *pos, int deldata);       // リストのpos以降のdeldata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(int*, int*);                   // int配列の内容でリストを初期化
    ~SortedLinkedList(){ clear();}                  // デストラクタ
    bool empty() const { return head==NULL; }       // リストが空か判定
    void printAll() const { print(head); }          // リストの全データを出力
    int size() const { return count(head); }        // リストの全データ数を返却
    void insert(int newdata){ head=insertNode(head, newdata);}    // リストにnewdataを追加
    void remove(int deldata){ head=removeNode(head, deldata);} // リストからdeldataを削除
    void clear(){ clearNode(head); head=NULL;}      // リストから全データを削除
};
```

Nodeを処理する再帰関数群を追加

printAll用の再帰関数printの例

呼ばれたprint(pointer)は、pointerの指しているデータを表示し、

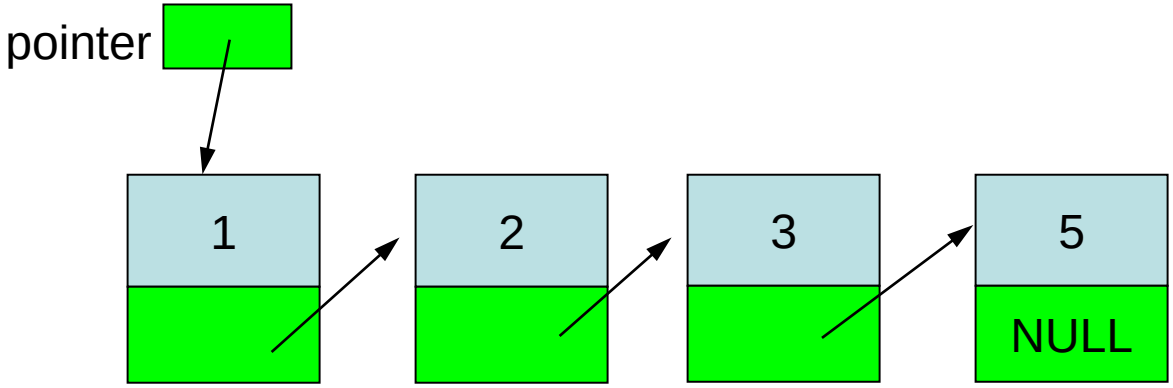


その後ろ（pointer->nextが指しているリスト構造）の表示は、
print(pointer->next)に任せれば良い。

```
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{
        cout << pointer->data << "-->"; //pointerの指すノードのdataを出力
        print(pointer->next);               //続きは別のprint()に任せる(=再帰)
    }
}
```

size用の再帰関数countの例

呼ばれたcount(pointer)には、pointerの指しているノードと

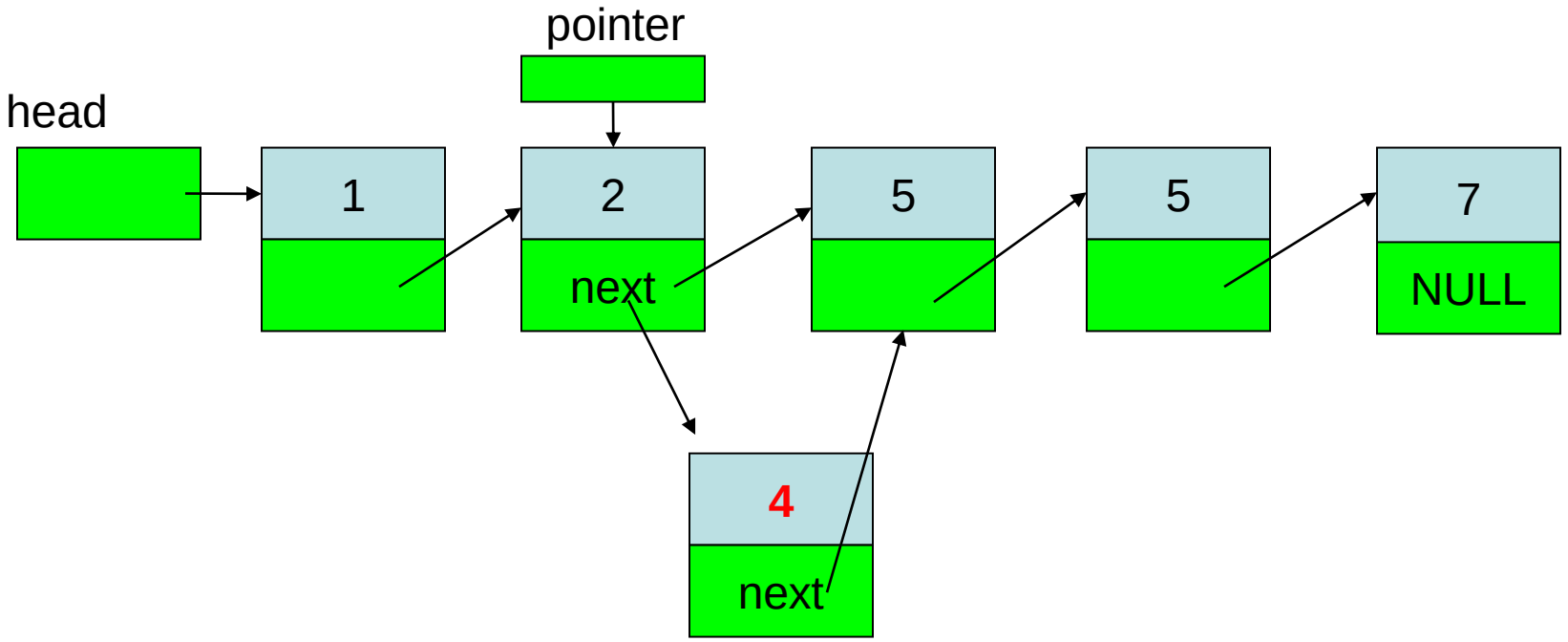


その後ろ（pointer->nextが指しているリスト構造）のノードの個数である
count(pointer->next)の和を返す

```
// next以降のノードの総数に、1を加算したものをリターンする
int SortedLinkedList::count(SortedLinkedList::Node *pointer) const{
    if(pointer==NULL){                          //pointerがNULLの場合
        return 0;                               //ノードがないので0を返却
    }else{                                      //ノードがある場合
        return 1+count(pointer->next);          //自分のノードの個数(=1)に、
    }                                           // next以降のcountの戻り値を加算した値を返却
}
```

ノードの追加

<1, 2, 5, 5, 7 > → <1, 2, **4**, 5, 5, 7>



① 引数pointerの指している位置が追加する場所ではない場合、
再帰的にnext以降を呼び出し、戻り値をpointer->nextに格納する。
さらに、pointerをリターンする。

② 引数pointerの手前に追加する場合、新しくノードを作りそのnextがpointerになるようにする。
新しく作成したノードのポインタをリターンする。

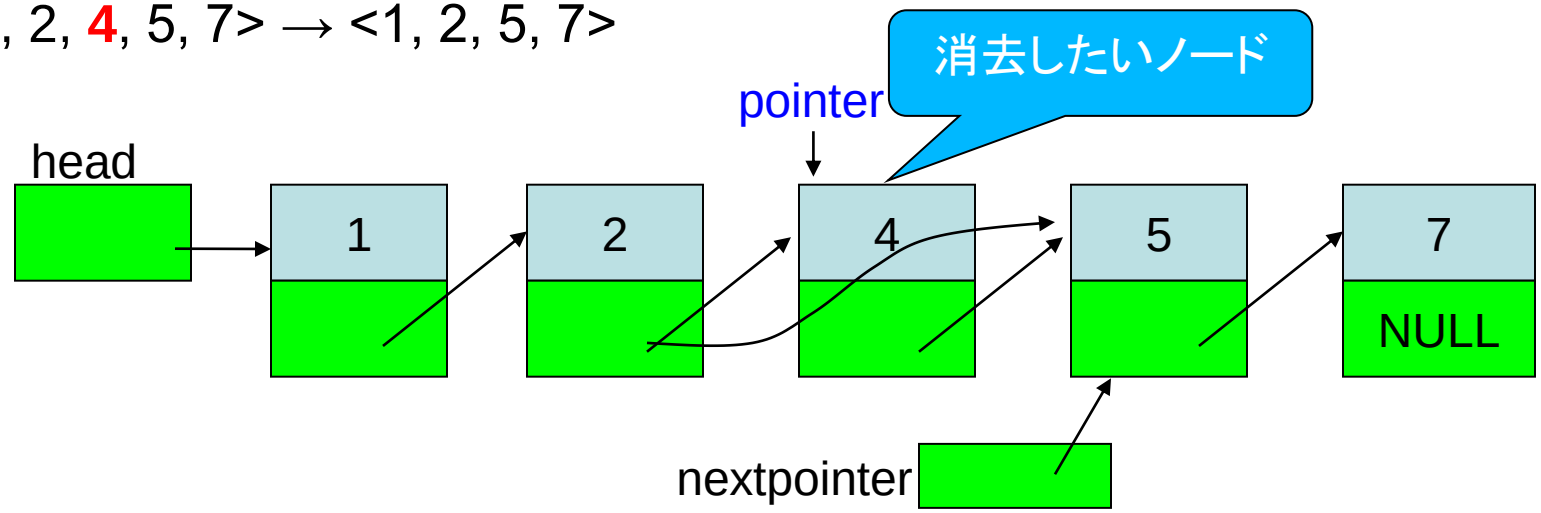
ノードの追加

```
SortedLinkedList::Node* SortedLinkedList::insertNode(SortedLinkedList::Node *pointer, int newdata){
   if(pointer==NULL){                          //末尾に追加する場合
      return new Node(newdata, 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
      return new Node(newdata, 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
   }
}

// コンストラクタ
SortedLinkedList::SortedLinkedList(int *begin, int *end){
    head=NULL;                                 //headをNULLで初期化
    for(int *p=begin; p !=end; p++){  //int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)となり、
    }                                           //上記insertNodeの処理に入る
}
```

ノードの削除

<1, 2, **4**, 5, 7> → <1, 2, 5, 7>



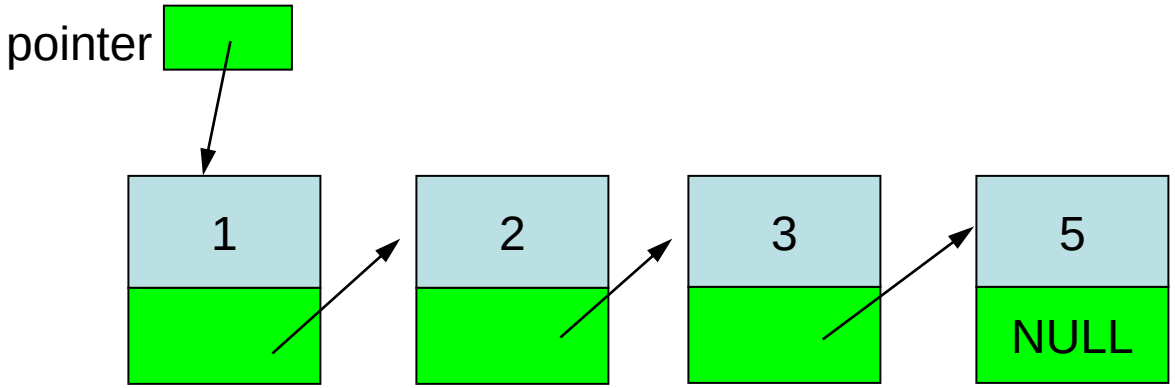
① 引数pointerの指しているノードが削除対象ではない場合、再帰的にnext以降を呼び出し、戻り値をpointer->nextに格納する。さらに、pointerをリターンする。
② 引数pointerの指しているノードを削除する場合、そのノードのnextを仮置きし、削除する。さらに、pointer->nextをリターンする。
③ 削除対象がない場合には何もしない。

ノードの削除

```
SortedLinkedList::Node* SortedLinkedList::removeNode(SortedLinkedList::Node *pointer, int deldata){
    if(pointer==NULL){                                          //末尾まで到達の場合
        return NULL;                                            //NULLを返す
    }else if(pointer->data==deldata){                           //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 nextpointer;                                     //nextpointerを返す
    }else{                                                      //削除対象ではない場合
        pointer->next=removeNode(pointer->next, deldata); //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
    }
}
```

全データ削除

pointer以降に繋がっているノードをすべて削除する




```
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
    }
}
```

再帰の場合のmain関数(再帰でない場合と変わらない)

```
int main(){
    int a[]={ 0, 1, 2, 5, 3, 3, 5, 7, 8, 8};            // -1は初期データの終端を表す
    int n;   char select;                                // n:リストに入れる非負の整数   select:メニュー項目の文字
    SortedLinkedList ichain(a, a+10);                    // int配列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, S:Size, P:Print, C:Clear, 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 >> n;   ichain.insert(n);          break;
            case 'R':                                    // リストから指定ノードを削除
            case 'r': cout << "Input a data-->"; cin >> n;   ichain.remove(n);         break;
            case 'S':                                    // リストの長さを表示
            case 's': cout << "Length=" << ichain.size() << "¥n";       break;
            case 'P':                                    // リストの全データを表示
            case 'p': ichain.printAll();    break;
            case 'C':                                    // リストの全データを削除
            case 'c': ichain.clear();       break;
            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

【実行例】
Menu[I:Insert, R:Remove, S:Size, P:Print, C:Clear, Q:Quit]
  Select I/R/S/P/C/Q-->P
0-->1-->2-->3-->3-->5-->5-->7-->8-->8-->END_OF_DATA

  Select I/R/S/P/C/Q-->S
Length=10

  Select I/R/S/P/C/Q-->R
Input a data-->3
3 is deleted.

  Select I/R/S/P/C/Q-->R
Input a data-->0
0 is deleted.

  Select I/R/S/P/C/Q-->S
Length=8

  Select I/R/S/P/C/Q-->p
1-->2-->3-->5-->5-->7-->8-->8-->END_OF_DATA

【右側に続く】

  Select I/R/S/P/C/Q-->c
8 is released.
8 is released.
7 is released.
5 is released.
5 is released.
3 is released.
2 is released.
1 is released.

  Select I/R/S/P/C/Q-->s
Length=0

  Select I/R/S/P/C/Q-->q
comsv01%

本日はここまでです
お疲れさまでした

質問があれば、下記までお願いします
11号館2階1212号室