C++プログラミングIII

第6回 連結リスト1

本日のお話

1. リストとは
2. リストを用いたプログラム例
3. SortedLinkedListクラスの実装

リストとは

1. リストとは

- **集合**(set) = 要素(element)の集まり。
  - 例： $\{3, 5, 1, 7\}$
  - 要素の順序に意味は無く、同じ要素は複数回現れない。
- **リスト**(list) = 要素の列
  - 例： $< 1, 3, 3, 5, 5, 5, 7 >$
  - 要素の順序に意味があり、同じ要素が複数回現れてもよい

- **リストに対する操作**
  1. リストを生成する　（コンストラクタで対応　）
  2. リストが空かどうかチェックする　（メンバ関数empty()で対応　）
  3. リスト内の要素を順に辿る
  4. リストに要素を加える　（メンバ関数insert()で対応　）
  5. リストから要素を除く　（メンバ関数remove()で対応　）

本日のお話

1. リストとは
2. リストを用いたプログラム例
3. SortedLinkedListクラスの実装

リストを用いたプログラム例

```
#include <iostream>
#include <list>
using namespace std;
int main() {
	int x[] = { 0, 1, 2, 5, 3, 3, 5, 7, 8, 8 };
	int n; char select;
	list<int> ichain(x, x + 10); // 配列x[0]からx[9]までを要素としてコピーしたリストを作成（x[10]は含まない）
	ichain.sort();
	cout << endl << "Menu[I:Insert, R:Remove, S:Size, P:Print, C:Clear, Q:Quit]";
	while ((cout << endl << "Select I/R/S/P/C/Q-->") && (cin >> select)) {
		switch (select) {
			case 'I': // 新しいノードを追加
			case 'i': cout << "Input a data-->"; cin >> n; ichain.insert(ichain.begin(), n); ichain.sort(); break;
			case 'R': // 指定したノードを削除
			case 'r': cout << "Input a data-->"; cin >> n; ichain.remove(n); break;
			case 'S': // リストの要素数を表示
			case 's': cout << "Length=" << ichain.size() << endl; break;
			case 'P': // リストの全データを表示. iteratorという抽象化されたポインタでアクセスする
			case 'p': for (list<int>::iterator it = ichain.begin(); it != ichain.end(); it++) cout << *it << "->";
			cout << "END_OF_DATA" << endl; 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;
}
```

リストを用いたプログラム例

- 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
Select I / R / S / P / C / Q-- > R
Input a data-- > 0
Select I / R / S / P / C / Q-- > S
Length = 7
Select I / R / S / P / C / Q-- > p
1->2->5->5->7->8->8->END_OF_DATA
Select I / R / S / P / C / Q-- > c
Select I / R / S / P / C / Q-- > s
Length = 0
Select I / R / S / P / C / Q-- > q
%
```

本日のお話

1. リストとは
2. リストを用いたプログラム例
3. SortedLinkedListクラスの実装

Sorted Linked List

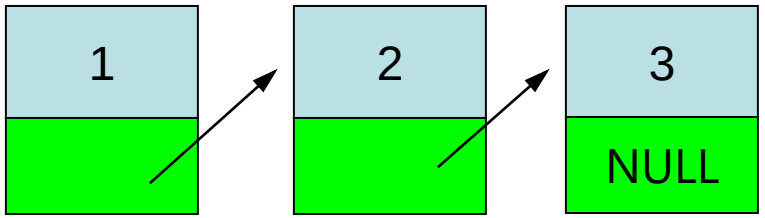
- リストの実装方法
  - 配列（array），ベクトル（vector）
  - 連結リスト（linked list）

- 配列，ベクトルによる実装
  - 問題点： 途中の要素の追加，削除が困難
- 連結リストによる実装
  - 要素をノード（node）で表現し，
    隣接するノードをリンクで連結して実装したリスト
  - 要素の追加，削除が容易

リストの実装

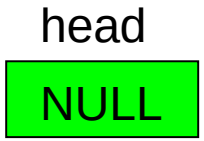
- 例
  - list: $< 1, 2, 3, 5 >$
  - linked list : $< head \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 5 >$
    (→: ポインタ　　$1, 2, 3, 5$: ノード)

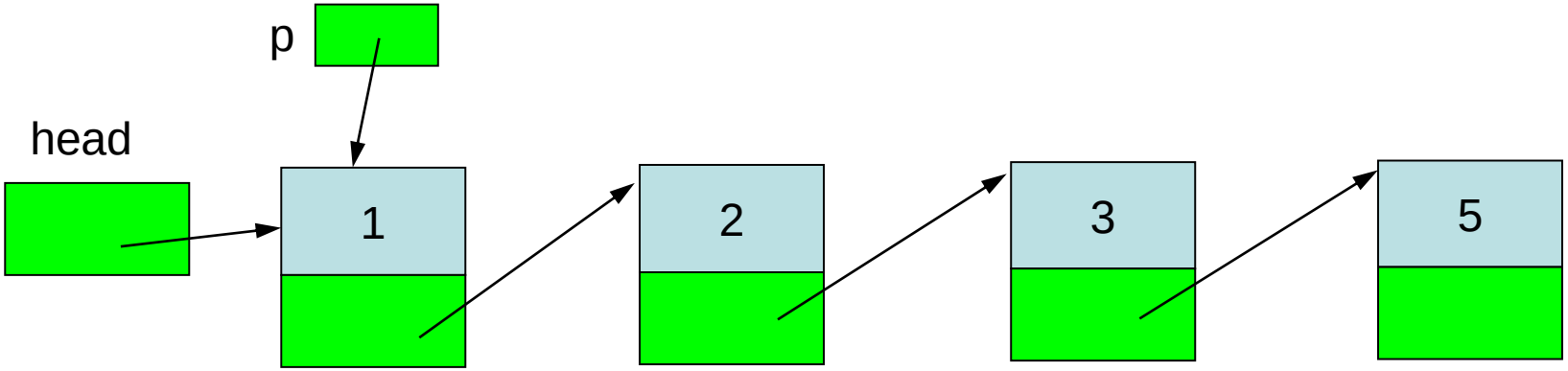
- ノードの構造
  - data: リストの要素
  - next: 次の要素ノードへのポインタ



リスト操作

1. リストの生成
   - 空リストを生成する。
   - head: リストの先頭
2. リスト内のノードをポインタで辿る

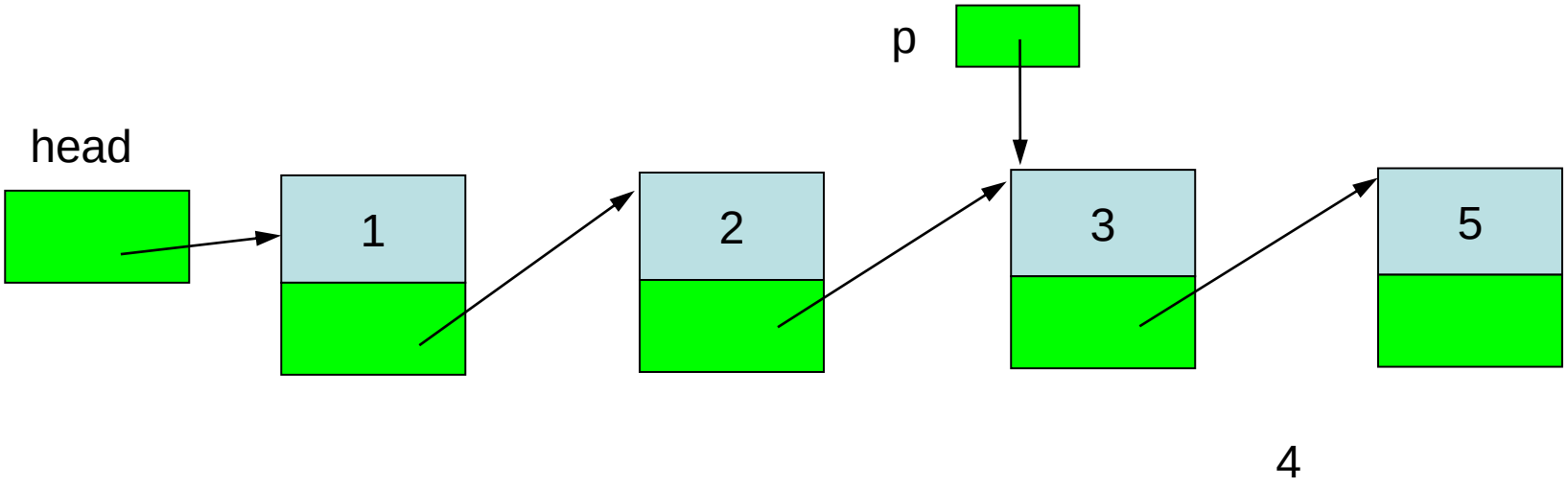




リスト操作

- ノードの追加
  1. 追加する位置の手前あるいは先まで辿る

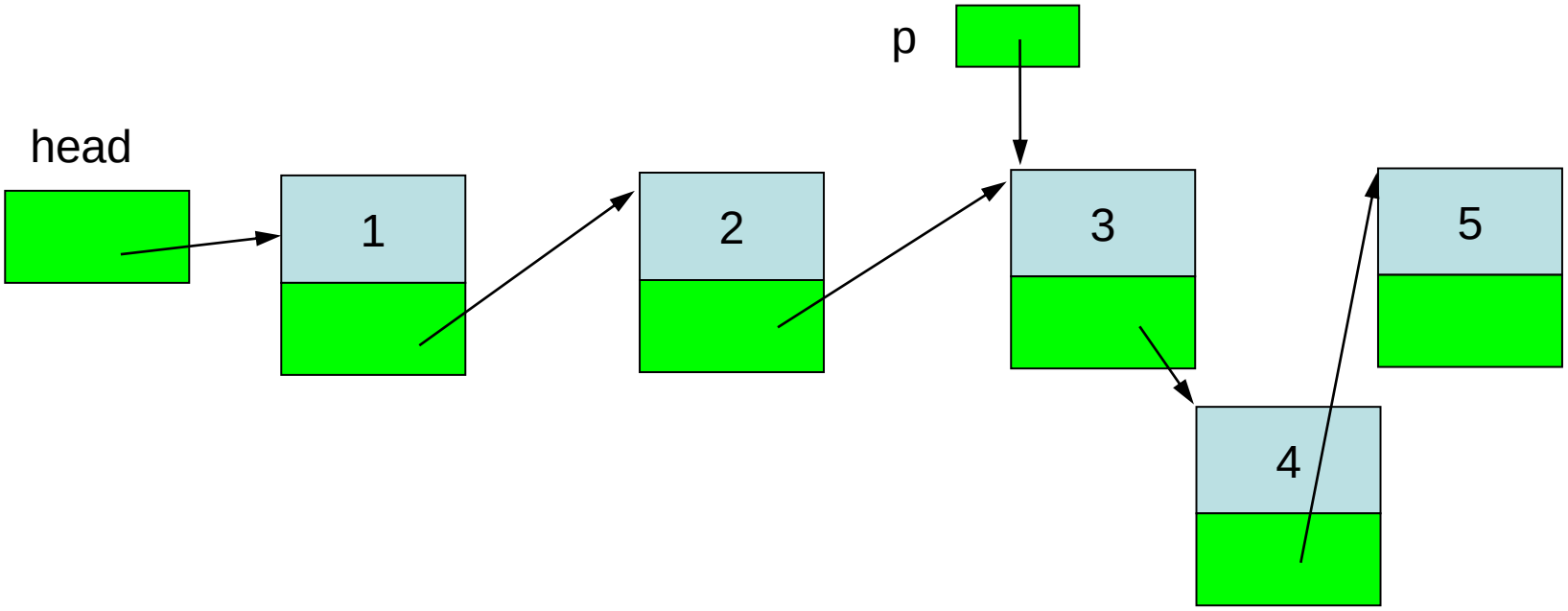
$< 1, 2, 3, 5 > \rightarrow < 1, 2, 3, \mathbf{4}, 5 >$



リスト操作

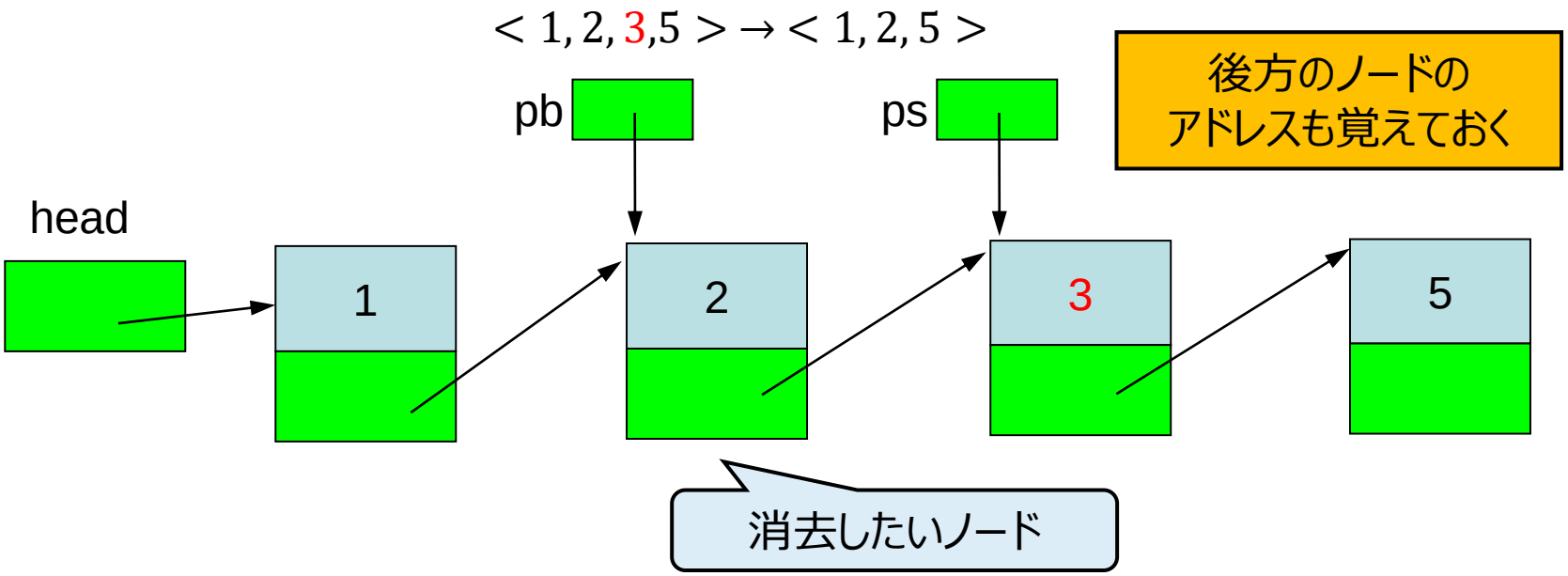
- ノードの追加
  1. 追加する位置の手前あるいは先まで辿る
  2. 追加する位置の後方のノードを追加するノードに繋ぎ，
     追加するノードを前方のノードに繋ぐ

$< 1, 2, 3, 5 > \rightarrow < 1, 2, 3, \mathbf{4}, 5 >$



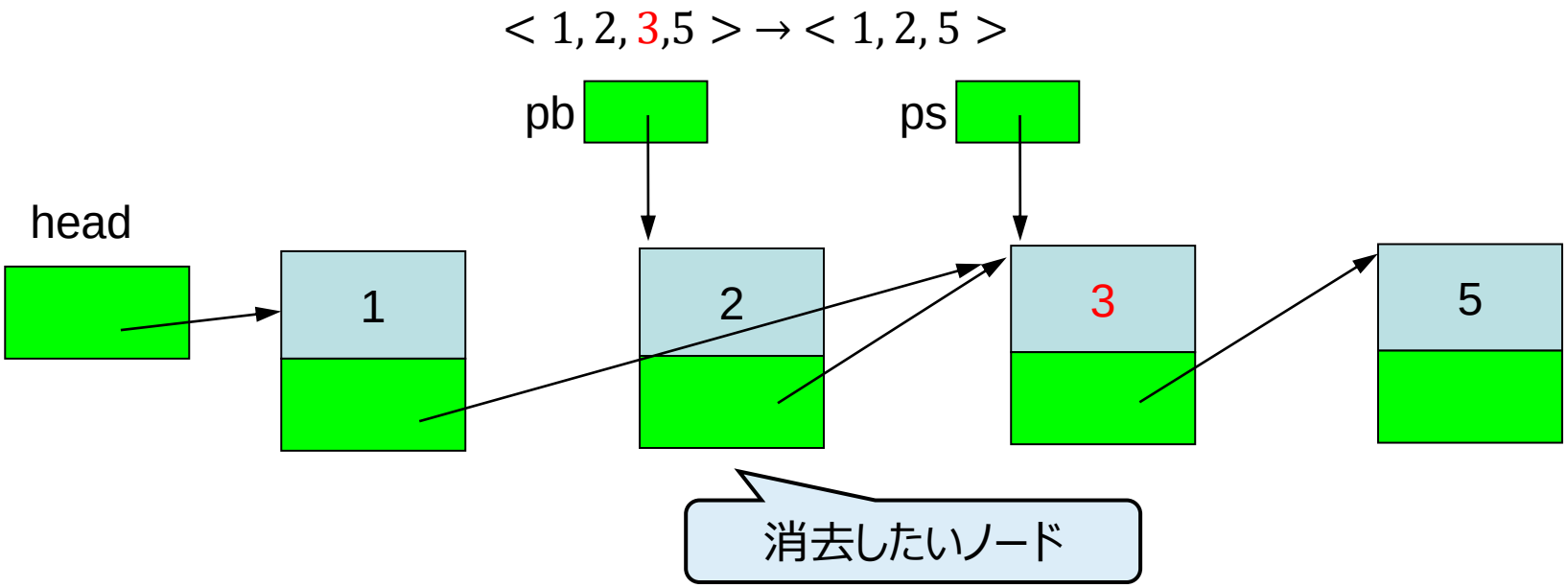
リスト操作

- ノードの削除
  1. 目的のノードまで辿る



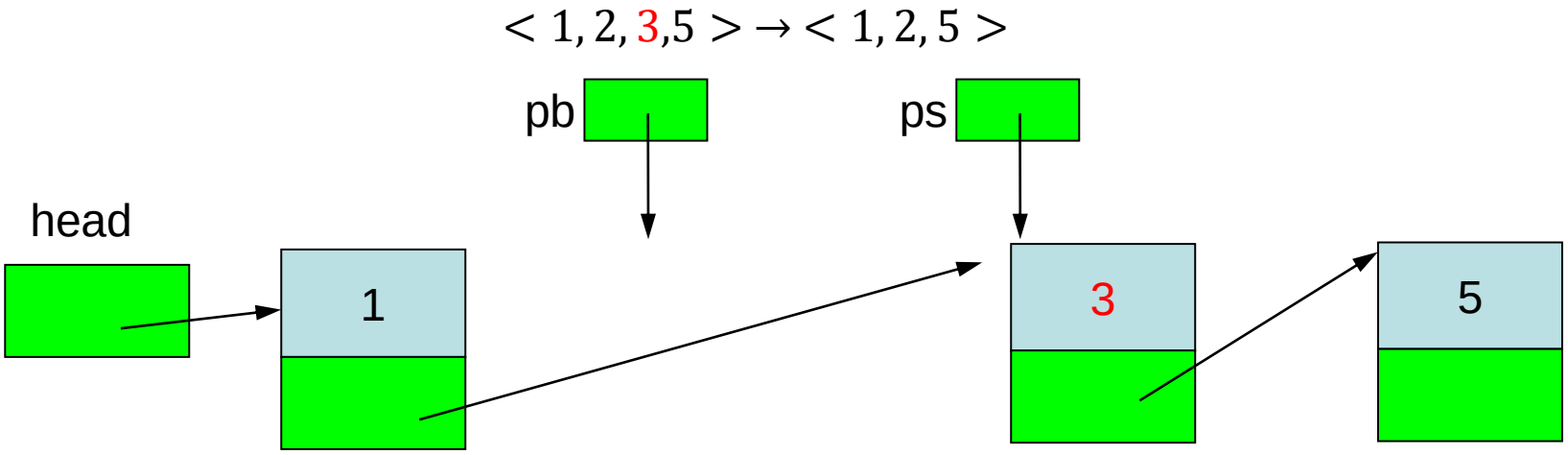
リスト操作

- ノードの削除
  1. 目的のノードまで辿る
  2. 目的のノードの後方のノードを前方のノードに繋ぐ



リスト操作

- ノードの削除
  1. 目的のノードまで辿る
  2. 目的のノードの後方のノードを前方のノードに繋ぐ
  3. 目的のノードを削除する



SortedLinkedListを用いたプログラム例

```
#include <iostream>
using namespace std;
class SortedLinkedList {
      class Node { // privateなので, SortedLinkedListのメンバ関数以外からはアクセスできない
            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* p = NULL) { data = num; next = p; }
            ~Node() { cout << data << " is released" << endl; }
      };
      Node* head;
      void push(int x) { head = new Node(x, head); }
      void pop() { Node* node = head;head = head->next;delete node;}
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(); }
      bool empty() const { return head == NULL; } // リストが空ならTrueを返す
      void clear() { while (head) pop(); }; // リストから全データを削除
      void printALL() const; // リストの全データを表示
      void insert(int n); // リストにデータnを追加
      void remove(int n); // リストからデータnを削除
      int size() const; // リストのデータ数を表示
};
```

SortedLinkedListを用いたプログラム例

```
#include <iostream>
using namespace std;
class SortedLinkedList {
        class Node { // privateなので，SortedLinkedListのメンバ関数以外からはアクセスできない
                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* p = NULL) { data = num; next = p; }
                ~Node() { cout << data << " is released" << endl; }
        };
        Node* head;
public:
        void clear() { while (head) pop(); }; // リストから全データを削除
        void printALL() const; // リストの全データを表示
        void insert(int n); // リストにデータnを追加
        void remove(int n); // リストからデータnを削除
        int size() const; // リストのデータ数を表示
};
```

SortedLinkedListの内部クラスNodeは，

1. SortedLinkedListのメンバ関数からはアクセスできる
2. mainなど他の関数からアクセスできない

という特徴を持つ

SortedLinkedListを用いたプログラム例

```
// int配列の内容でリストを初期化
SortedLinkedList::SortedLinkedList(int* begin, int* end){
      head = NULL;
      for (int* p = begin; p != end; p++) insert(*p);
}

// リストの全データを表示
void SortedLinkedList::printALL() const {
      for (Node* p = head; p; p = p->next)
            cout << p->data << "-->";
      cout << "END_OF_DATA" << endl;
}

// リストの要素数を返す
int SortedLinkedList::size() const {
      int n = 0;
      for (Node* p = head; p; p = p->next) n++;
      return n;
}
```

SortedLinkedListを用いたプログラム例

```
// リストに要素nを昇順になるように挿入する
void SortedLinkedList::insert(int n) {
    // 空もしくは先頭に挿入する場合はpush_front
    if (empty() || head->data > n) push(n);
    else { // そうでない場合


    }
}
// リストから値がnの要素の中で最も先頭にある要素を取り除く
void SortedLinkedList::remove(int n) {
    // 先頭が削除対象の場合は場合はpop_front
    if (head->data == n) pop();
    else { // そうでない場合


    }
}
```

本日はここまで

- お疲れさまでした