

データ構造入門及び演習

11回目: リスト構造

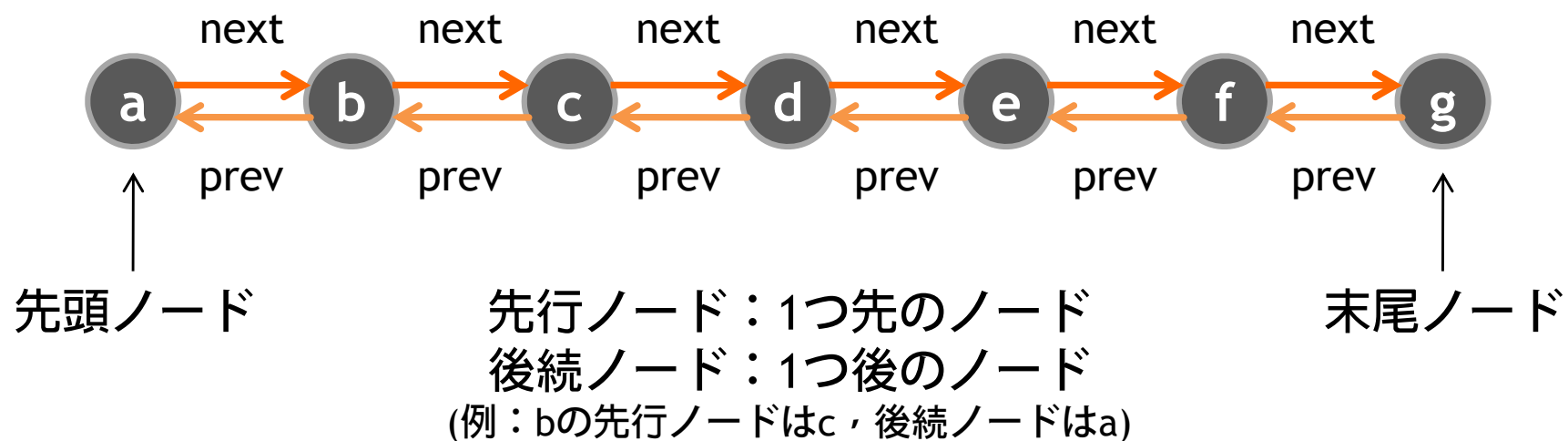
2014/06/27

担当: 見越 大樹

61号館304号室

リスト構造 (List)

- リストとは？
 - データが順序づけられて並んだデータ構造
 - データ間の「**つながり情報**」を持つ
 - 単方向リスト: 先のつながり情報のみを持つ(片方向でのみ探索可能)
 - 双方向リスト: 先と後のつながり情報を持つ(双方向で探索可能)
 - リスト上の個々のデータを**ノード(node)**, または**要素**と呼ぶ



リスト構造 (List)

- リストの応用分野:
 - データベースのデータ記録形式等に使われる
 - データ量が多いほどデータベースの更新(挿入, 追加, 削除)に高速な処理が必要
- リストの例:
 1. 配列を使用したリスト:
 - 配列番号(インデックス)が次のノードへのつながり情報となる
 - つながり情報の変更に時間がかかる
 2. 構造体を使用したリスト:
 - 構造体のメンバ変数に次のノードへのつながり情報(ポインタ)を持たせる
 - つながり情報の変更が簡単かつ高速

配列を使用したリスト

- 配列を使用して, リストを作成
 - 配列番号+1が次のノード

- 使用例:

```
int a[10] = {1, 3, 4, 7, 8, 10, 12, -1, -1, -1};
```

-1はデータが無いことを表す



出来上がるリスト



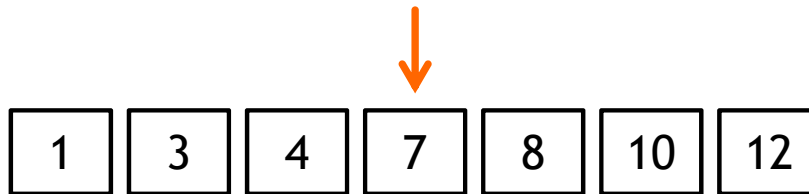
リスト(配列) の表示プログラム(ShowList)

```
void ShowList( int data[] )  
{  
    printf("リスト:");  
  
    for( int i=0;i<MAX;i++ ){  
        if( data[i] != -1 ){  
            printf( "%4d", data[i] );  
        }  
        else  
            break;  
    }  
  
    printf( "¥n" );  
}
```

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
1	3	4	7	8	10	12	-1	-1	-1

配列を使用したリスト(データの削除)

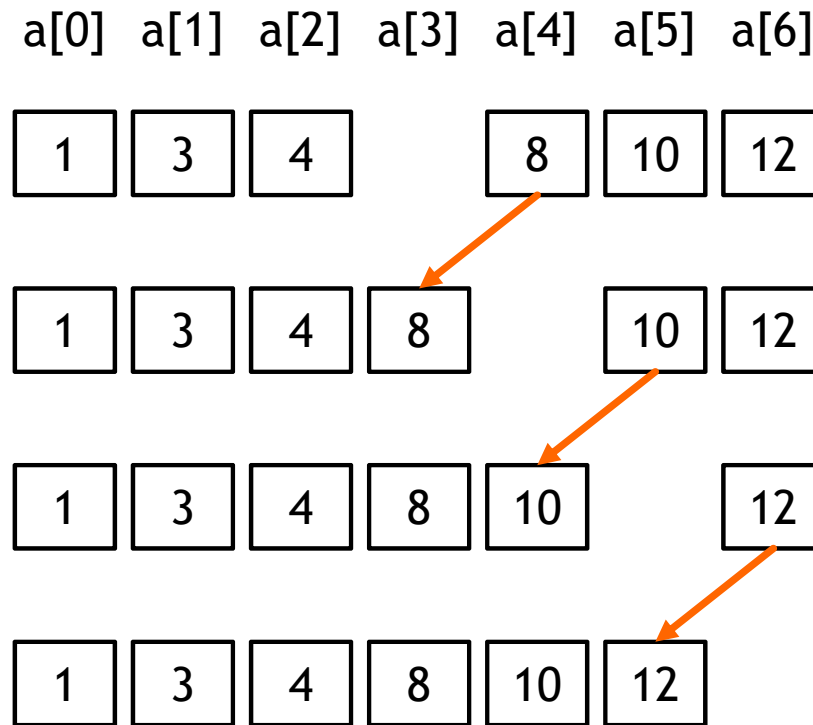
これを削除する



①削除する

②左に詰める

配列を使用したリスト(データの削除)



$a[3]=a[4]$

$a[4]=a[5]$

$a[5]=a[6]$

左から順につめる

配列を使用したリスト(データの削除)

上書きすることで、データを削除

a[0] a[1] a[2] a[3] a[4] a[5] a[6]

1	3	4	7	8	10	12
---	---	---	---	---	----	----

1	3	4	8	8	10	12
---	---	---	---	---	----	----

1	3	4	8	10	10	12
---	---	---	---	----	----	----

1	3	4	8	10	12	12
---	---	---	---	----	----	----

1	3	4	8	10	12	-1
---	---	---	---	----	----	----

a[3]=a[4]

a[4]=a[5]

a[5]=a[6]

a[6]=-1

* -1はデータがないことを示す

配列を使用したリスト(データの削除)


- 配列からのデータの削除ができない場合

削除位置にデータが存在しない

例) $a[7]$ を削除したい

・-1が格納されているので, 削除の必要がない

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$	$a[7]$	$a[8]$	$a[9]$
1	3	4	7	8	10	12	-1	-1	-1



リスト(配列)からの削除プログラム(Main)

```
#include <stdio.h>
#define MAX 15

void ShowList( int[] );
int InsertNode( int, int, int[] );
int DeleteNode( int, int[] );

void main()
{
    int data[] = { 2,4,6,8,10,12,14,16,18,
                  -1,-1,-1,-1,-1,-1 };
    int no, insdata, id;

    ShowList( data ); // リスト表示

    printf("削除位置:");          // 削除位置の指定
    scanf("%d", &no);

    id = DeleteNode( no, data ); // 削除
```

```
if( id == -1 )
    printf( "削除に失敗しました¥n¥n" );
else{
    printf( "削除したデータは%dです¥n", id );
    ShowList( data );          // リスト表示
}
}
```

リスト(配列)からの削除プログラム(DeleteNode)

```

int DeleteNode( int no, int data[] ) { // no : 削除位置
    int n, i, tmp;

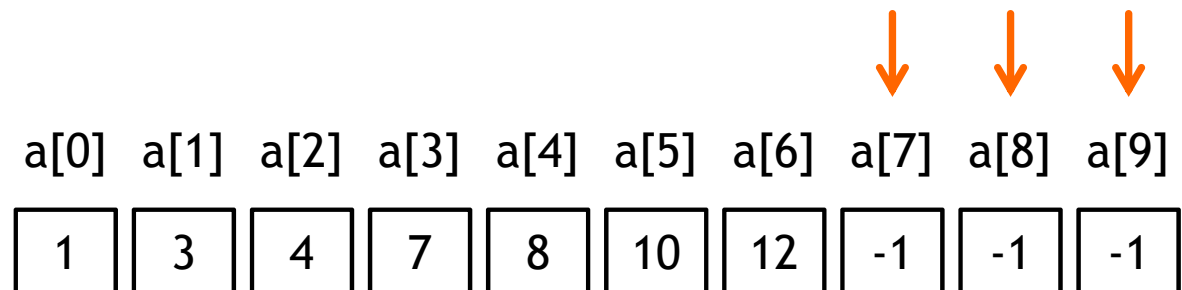
    for( n=0;n<MAX;n++ ){ // データ個数を数える
        if( data[n] == -1 ) break;
    }
    // 削除位置にデータがなければ, 削除できない
    if( no < 0 || n <= no ) return -1;
    // それ以外は削除できる
    else{
        tmp = data[no]; // 削除するデータをtmpに保存する

        // no+1~nまでデータを左につめる
        for( i=no+1;i<n;i++ )
            data[i-1] = data[i];
        // data[n-1]を無効ノードとする
        data[n-1] = -1;

        // 削除したデータをリターンする
        return tmp;
    }
}

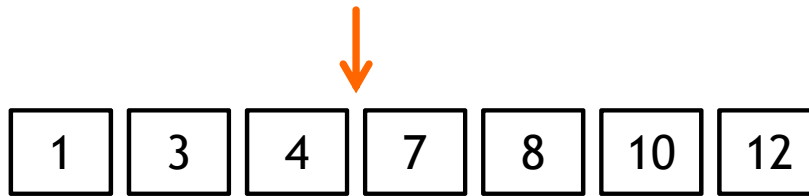
```

削除できない場合
no=7, 8, 9



配列を使用したリスト(データの挿入)

ここに5を挿入する



①右にずらす



②挿入する

配列を使用したリスト(データの挿入)



右から順にずらす

挿入する

配列を使用したリスト(データの挿入)

a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7]

1	3	4	7	8	10	12	-1
---	---	---	---	---	----	----	----

* -1はデータがないことを示す

1	3	4	7	8	10	12	12
---	---	---	---	---	----	----	----

$a[7] = a[6]$

1	3	4	7	8	10	10	12
---	---	---	---	---	----	----	----

$a[6] = a[5]$

1	3	4	7	8	8	10	12
---	---	---	---	---	---	----	----

$a[5] = a[4]$

1	3	4	7	7	8	10	12
---	---	---	---	---	---	----	----

$a[4] = a[3]$

挿入する

1	3	4	5	7	8	10	12
---	---	---	---	---	---	----	----

$a[3] = \text{insert_data}$
 $= 5$

右から順にずらす

配列を使用したリスト(データの挿入)

- 配列へのデータの挿入ができない場合

①配列が満杯

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
1	3	4	7	8	10	12	13	15	18

- ②挿入位置が不正

a[8]にデータを挿入
・左端からデータが連続しない



a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
1	3	4	7	8	10	12	-1	-1	-1

リスト(配列)への挿入プログラム(Main)

```
#include <stdio.h>
#define MAX 15

void ShowList( int[] );
int InsertNode( int, int, int[] );
int DeleteNode( int, int[] );

void main()
{
    int data[] = { 2,4,6,8,10,12,14,16,18,
                  -1,-1,-1,-1,-1,-1 };
    int no, insdata, id;

    ShowList( data ); // リスト表示

    printf("挿入位置:"); // 挿入位置の指定
    scanf("%d", &no);
    printf("挿入データ:"); // 挿入データの指定
    scanf("%d", &insdata);
```

```
    id = InsertNode( no, insdata, data ); // 挿入
    if( id == 0 ){
        printf( "挿入に成功しました¥n¥n" );
        ShowList( data ); // リスト表示
    }
    else if( id == -1 )
        printf( "挿入に失敗しました¥n¥n", id );
}
```


リスト(配列)への挿入プログラム(InsertNode)

```
int InsertNode( int no, int insdata, int data[] ){
```

```
    // データ個数数える
```

```
    int n;
```

```
    for( n=0;n<MAX;n++ ){
```

```
        if( data[n] == -1 ) break;
```

```
    }
```

```
    // ①配列が満杯なら, 挿入できない
```

```
    if( n == MAX ) return -1;
```

```
    // ②挿入位置の左側にデータが無い(不正な場所)
```

```
    else if( n < no )
```

```
        return -1;
```

```
    // それ以外は挿入できる
```

```
    else{
```

```
        for( int i=n-1;i>=no;i-- ){
```

```
            data[i+1] = data[i];
```

```
        }
```

```
        data[no] = insdata;
```

```
        return 0;
```

```
    }
```

```
}
```

①配列が満杯で, 挿入できない

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
1	3	4	7	8	10	12	13	15	18

②挿入位置の左側にデータが無いいため
挿入できない

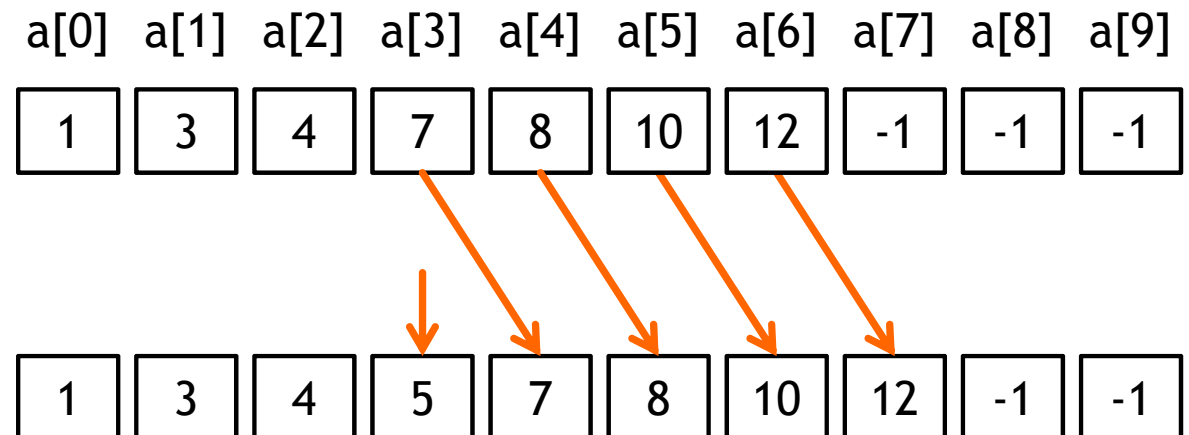
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
1	3	4	7	8	10	12	-1	-1	-1



リスト(配列)への挿入プログラム(InsertNode)

```
int InsertNode( int no, int insdata, int data[] ){  
    // データ個数数える  
    int n;  
    for( n=0;n<MAX;n++ ){  
        if( data[n] == -1 ) break;  
    }  
  
    // ①配列が満杯なら, 挿入できない  
    if( n == MAX ) return -1;  
    // ②挿入位置の左側にデータが無い(不正な場所)  
    else if( n < no )  
        return -1;  
    // それ以外は挿入できる  
    else{  
        for( int i=n-1;i>=no;i-- ){  
            data[i+1] = data[i];  
        }  
        data[no] = insdata;  
        return 0;  
    }  
}
```

挿入できる場合



配列を使用したリスト まとめ

- 大量のデータを使用可能な配列を使用してリストを実現
- リスト(配列)のデータ表示・データ挿入・データ削除
- 配列の添字をつなぎ情報として利用
- データシフト(データのずらし)を行う必要がある
- データ量が多い場合はシフトに処理時間がかかる

リスト構造 (List)

- リストの例:

1. 配列を使用したリスト:

- 配列番号(インデックス)が次のノードへのつながり情報となる
- つながり情報の変更にかかる時間がかかる

2. 構造体を使用したリスト:

- 構造体のメンバ変数に次のノードへのつながり情報(ポインタ)を持たせる
- つながり情報の変更が簡単かつ高速

構造体の宣言（復習）

1. 構造体テンプレートの宣言

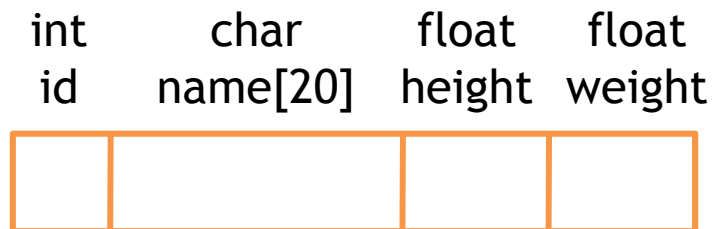
- どのような型の変数や配列を1つにまとめるかを決定する

```

struct student {
    int id;
    char name[20];
    float height;
    float weight;
}; ← セミコロン
  
```

構造体テンプレート名(タグ)

メンバ
(構成要素)



typedef (復習)

- typedef: 型の名前を定義する

struct student → typedef → MyStudent

```
struct student {  
    int id;  
    char name[20];  
    float height;  
    float weight;  
};  
  
struct student person[5];
```

```
typedef struct student {  
    int id;  
    char name[20];  
    float height;  
    float weight;  
} MyStudent; ← 新しい型の名前  
  
MyStudent person[5];
```

↑ 省略可

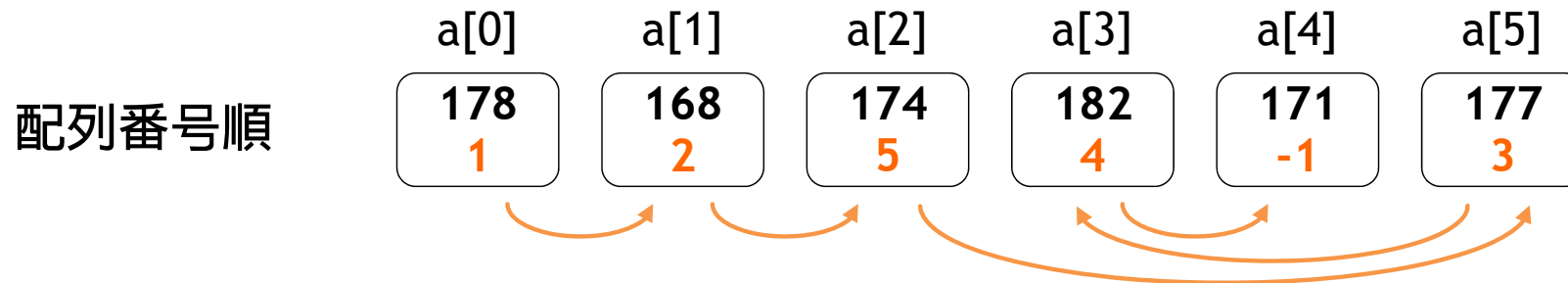
構造体を使用したリスト

- 構造体を使用して、配列の個々の要素に次の要素への「**つながり情報**」を持たせたデータ構造を定義する
- 簡単化のため、先頭のデータは必ず配列の添え字0の場所にあるものとする
- 使用例:

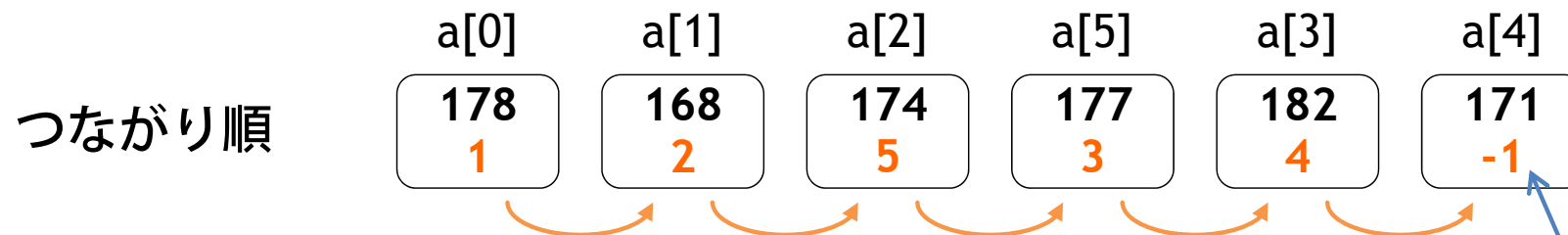
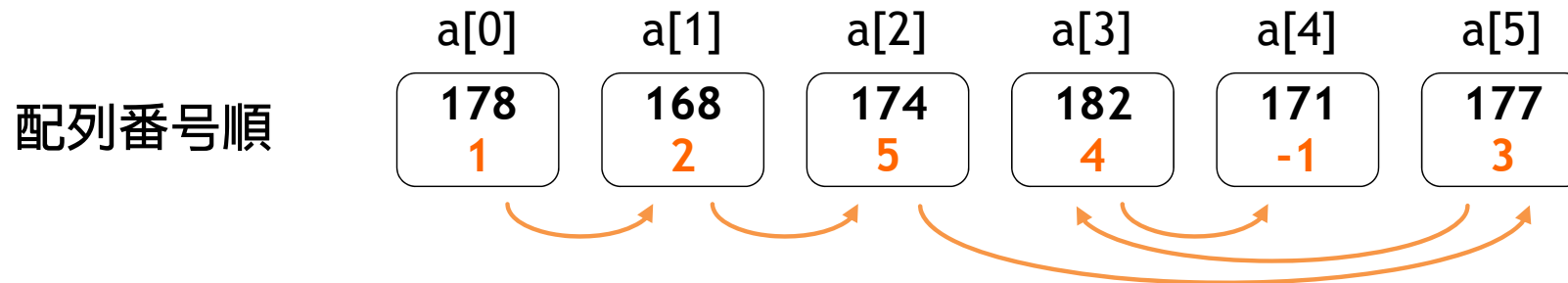
```
// 宣言
typedef struct {
    int  IntData      データ
    int  NextIndex;   つながり情報
} MyList;

MyList a[10];
```

構造体を使用したリスト



構造体を使用したリスト



末尾には-1が入る

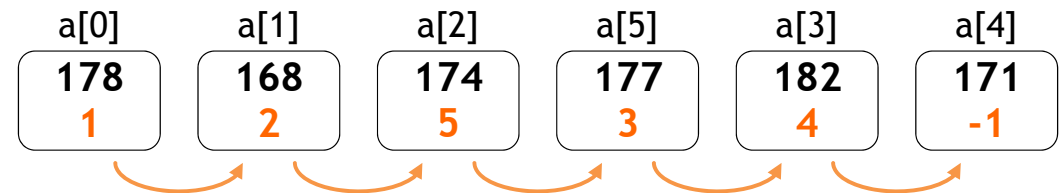
リストの表示 ShowList



```
void ShowList(MyList data[]){ // data[] : 表示するリスト
    int i = 0; // 現在の場所 (配列の添え字)
    while(1){
        // データとつながり情報を表示
        printf("(%d %d)\n", data[i].IntData, data[i].NextIndex);

        // 現在の場所を更新
        i = data[i].NextIndex;

        // 次の場所が無い (末尾に到達)
        if(i == -1) break;
    }
    printf("\n");
}
```



```
// 宣言
typedef struct {
    int IntData          データ
    int NextIndex;       つながり情報
} MyList;
```

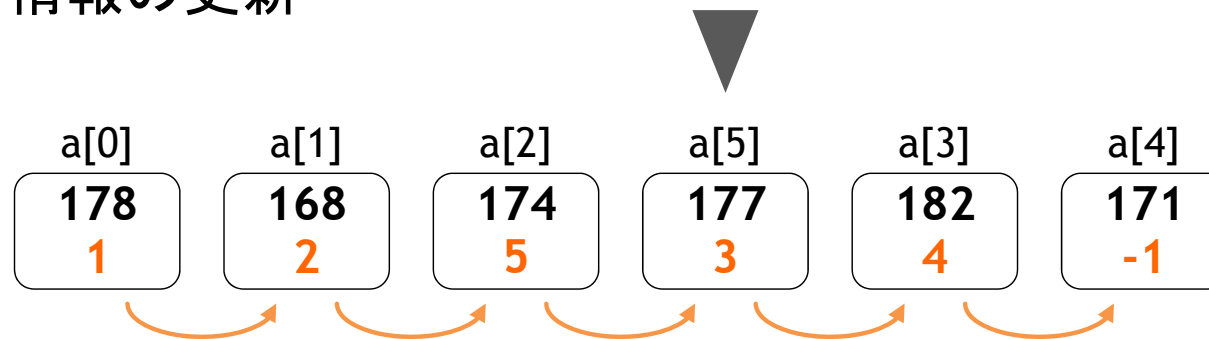
データの削除方法



- 手順: 177 (4番目)を削除する場合

1. つながり情報の更新

削除前

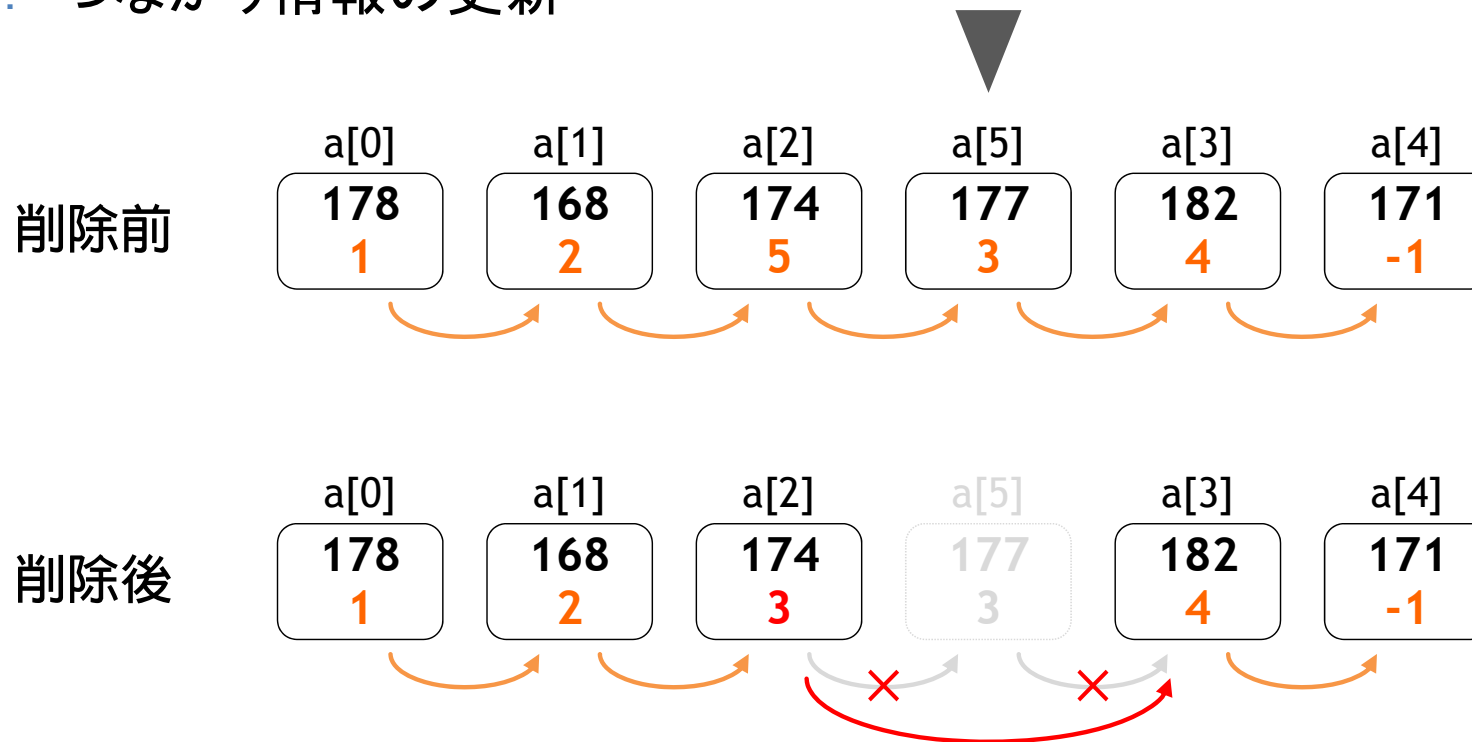


データの削除方法



- 手順: 177 (4番目)を削除する場合

1. つながり情報の更新



注意：リストからは削除されるが、物理的には削除されない！

リストからの削除 (Delete)



```
int Delete (int index, MyList data[]){
    int n = 0;
    int i = 0;
    int pre = -1;

    // 削除対象範囲外
    if(index <= 1) return -1;
```

削除対象リスト
削除するデータの
場所

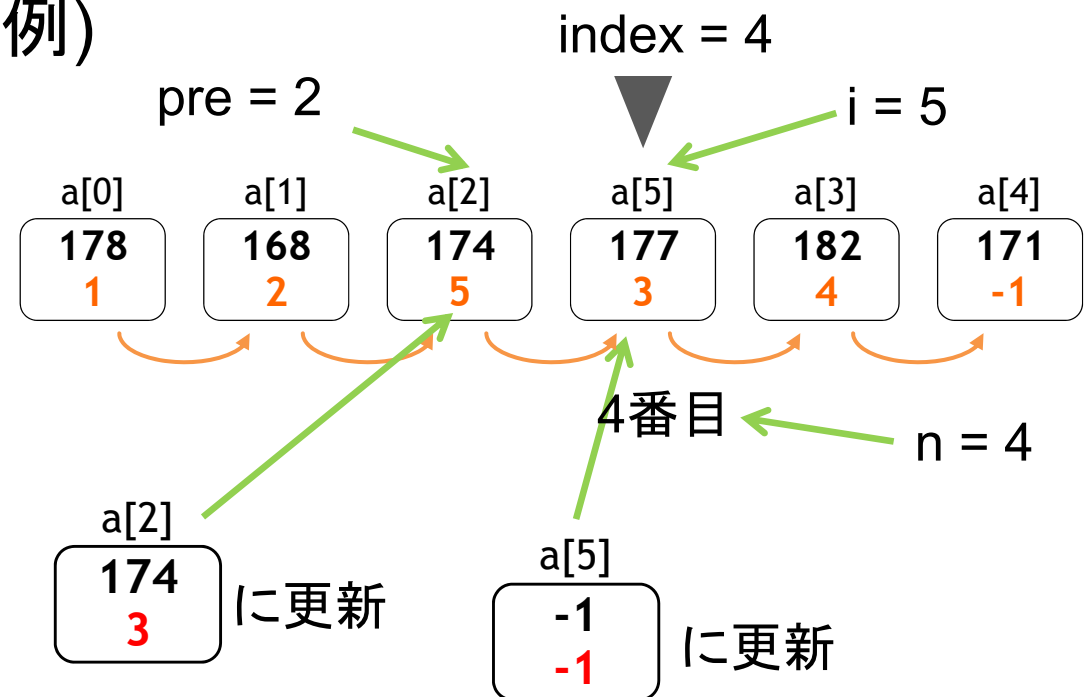
```
while(1){
    if(data[i].IntData != -1){
        n++; // 先頭から何番目か
        // 削除場所に到達(データを削除)
        if(n == index){
            data[pre].NextIndex =
                data[i].NextIndex;
            data[i].IntData = -1;
            data[i].NextIndex = -1;
            return 0;
        }
    }
}
```

```
pre = i;
i = data[i].NextIndex;
```

// 末尾まで到達したが、削除データが無い
if(i == -1) return -1;

```
}
}
```

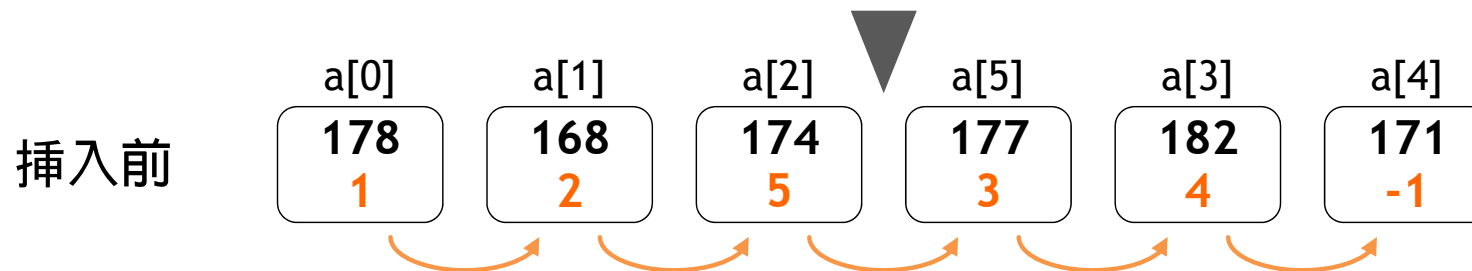
例)



データの挿入方法



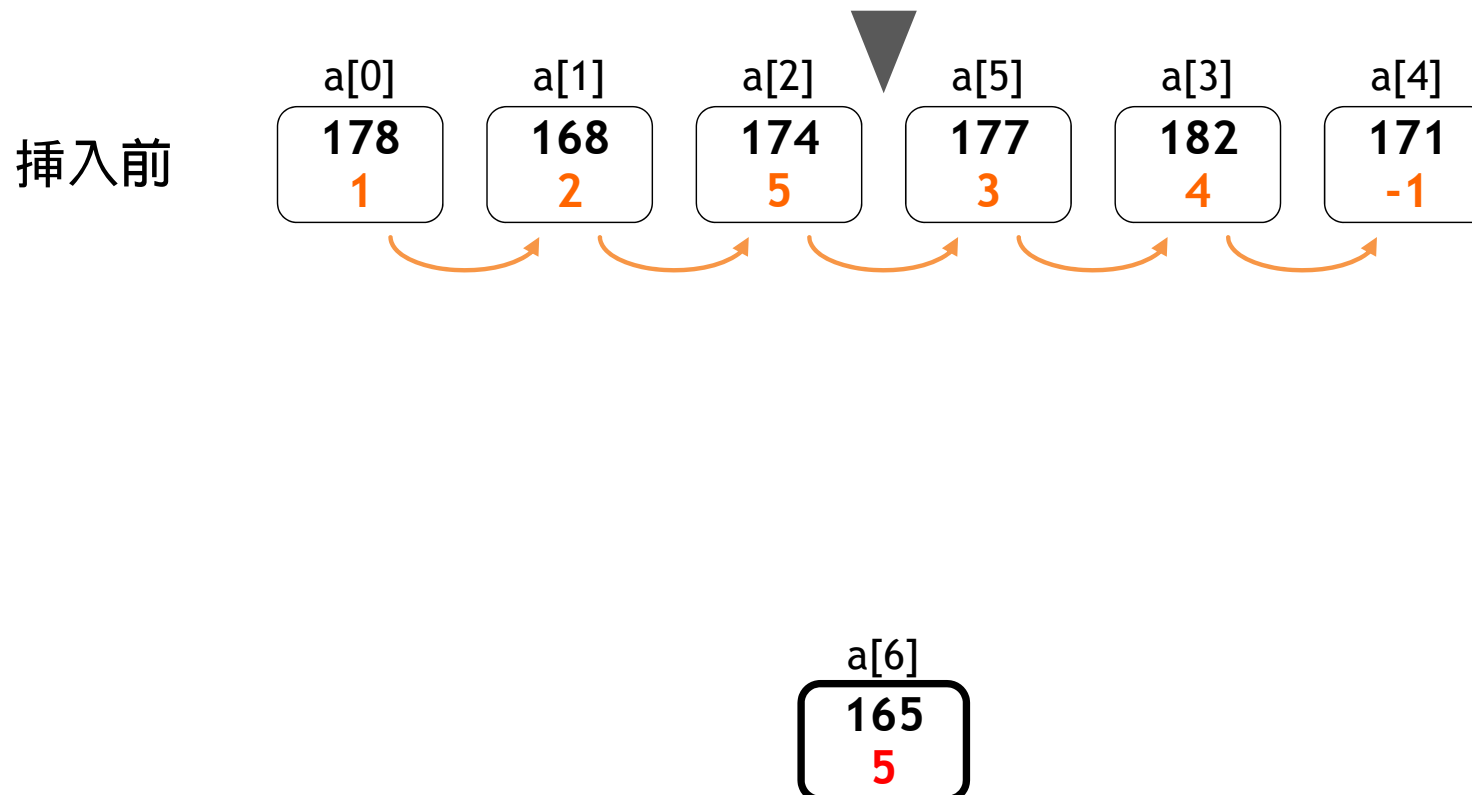
- 手順: 174と177の間(4番目)に165を挿入する場合





データの挿入方法

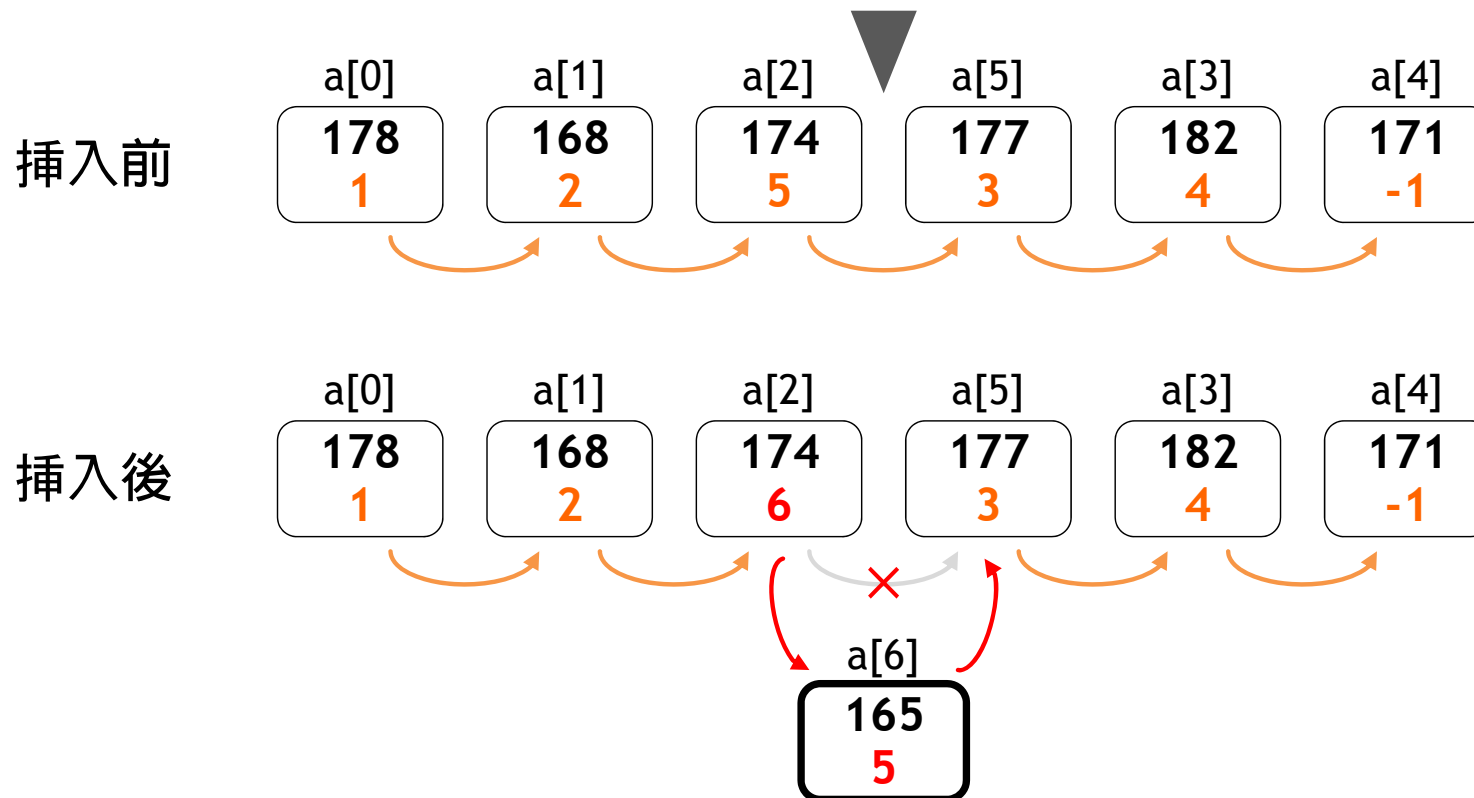
- 手順: 174と177の間(4番目)に165を挿入する場合
 - 未使用の配列にデータを格納





データの挿入方法

- 手順: 174と177の間(4番目)に165を挿入する場合
 - 未使用の配列にデータを格納
 - つながり情報の更新



リストへの挿入 Insert

```

int Insert(int index, int ins_data, MyList data[]){
// index : 挿入場所, ins_data : 挿入データ, data : リスト
  int blank = 0; // 配列の空いている場所
  int n = 0;
  int i = 0, pre = -1;

  //空いている箇所を探す
  for(blank = 0; blank < MAX; blank++){
    if(data[blank].IntData == -1) break;

  // 削除対象範囲外 or 配列が満杯で挿入不可能
  if(index <= 1 || blank == MAX) return -1;

  while(1){
    if(data[i].IntData != -1){
      n++; // データの挿入場所まで移動
      if(n == index){// データを挿入
        data[blank].IntData = ins_data;
        data[blank].NextIndex =
          data[pre].NextIndex;
        data[pre].NextIndex = blank;
        return 0;
      }
    }
  }
}

```

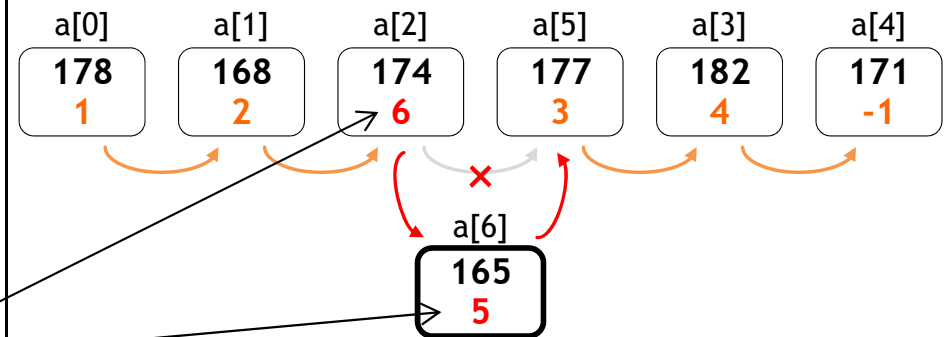


```

}
}
pre = i;
i = data[i].NextIndex;

// 末尾に到達したが,
// 挿入場所に辿り着かなかった
if(current == -1){
  return -1;
}
}
}

```



つながり情報の更新

pre = 2, blank = 6

構造体を使用したリスト まとめ

- 構造体を使用してつながり情報を含むリストを実現
- リスト(構造体)のデータ表示・データ挿入・データ削除
 - つながり情報を利用
 - データシフトを行う必要がない！
 - 処理が高速

リストからの削除(main)

```
#include <stdio.h>
```

```
#define MAX 15
```

```
typedef struct {
    int IntData;
    int NextIndex;
}MyList;
```

```
void ShowList(MyList data[]);
int Delete (int index, MyList data[]);
```

```
void main(){
    MyList data[MAX];
    int i, id, result;
```

```
//空のリストを作成
```

```
for(i = 0; i < MAX; i++){
    data[i].IntData = -1;
    data[i].NextIndex = -1;
}
```

```
// リストの初期値設定
```

```
data[0].IntData = 12, data[0].NextIndex = 3;
data[1].IntData = 23, data[1].NextIndex = 2;
data[2].IntData = 55, data[2].NextIndex = -1;
data[3].IntData = 30, data[3].NextIndex = 4;
data[4].IntData = 80, data[4].NextIndex = 1;
```

```
// リスト表示
```

```
ShowList(data);
```

```
// 削除位置指定
```

```
printf("削除位置 : ");
scanf("%d", &id);
printf("¥n");
```

```
// リストから削除
```

```
result = Delete(id, data);
```

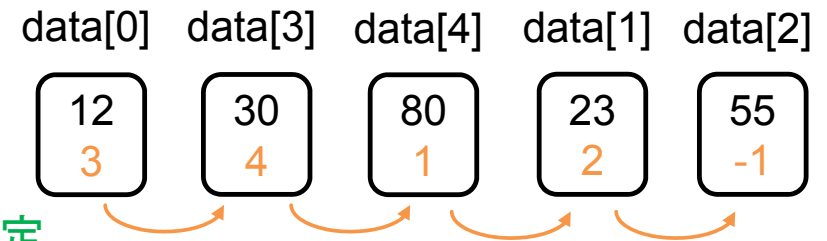
```
// リスト表示
```

```
ShowList(data);
```

```
// 結果表示
```

```
if(result == -1) printf("失敗しました¥n");
else             printf("成功しました¥n");
```

```
}
```



リストへの挿入(main)

```
#include <stdio.h>
#define MAX 15
```

```
typedef struct {
    int IntData;
    int NextIndex;
}MyList;
```

```
void ShowList(MyList data[]);
int Insert(int index, int ins_data, MyList data[]);
```

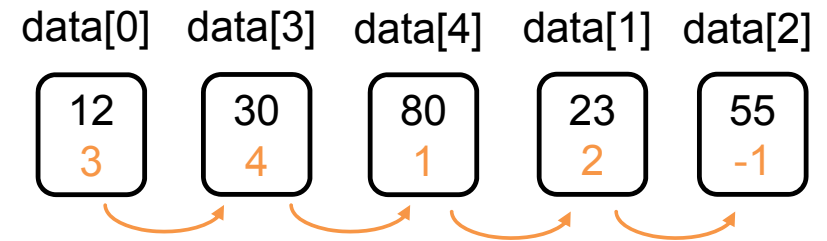
```
void main(){
    MyList data[MAX];
    int i, id, ins_data, result;
```

//空のリストを作成

.....(前ページの空のリストを作成と同じ)

// リストの初期値設定

.....(前ページの初期値設定と同じ)



// リスト表示

```
ShowList(data);
```

// 挿入位置指定

```
printf("挿入位置 : ");
scanf("%d", &id);
printf("¥n");
```

// 挿入データ指定

```
printf("挿入データ : ");
scanf("%d", &ins_data);
printf("¥n");
```

// リストへ挿入

```
result = Insert(id, ins_data, data);
```

// リスト表示

```
ShowList(data);
```

// 結果表示

```
if(result == -1) printf("失敗しました¥n");
else             printf("成功しました¥n");
```

```
}
```