情報構造 第三回

抽象データ型

データ型の実現

- データ型（整数，配列，構造体型など）を，個々のシステムで機械語表現する
  => データ型の実現（実装：implementation）

  しかし．．．

- 個々のシステムのビット長やレジスタに基づく制約（例えば，整数の最大値や配列の時限数など）を受けてしまう

=> 抽象データ型で解決！

今日の予定

- 抽象データ型とは
- 抽象データ型BAGの実装（C言語）
- オブジェクト指向

「抽象（abstract）」とは

- 抽象 = 抽出 = 捨象
- 抽象：事物または表象の或る側面・性質を抽（ひ・ぬ）き離して把握する心的作用。その際おのずから他の側面・性質を排除する作用が伴うが、これを捨象という。…（広辞苑）
- 対義語：具体，具象，concrete => コンクレートな音楽？

- 工学的では，ものを「作る」際に，本質（機能，性質）のみに注目し，他を忘れる（忘れても問題ない仕掛けを用意する）ことで大規模な構造物を得る =>モジュール化

参考：Music Concrete

- 日本語：具体音楽（音を作るのでなく，元々ある音を使う）

- 人や動物の声、鉄道や都市などから発せられる騒音、自然界から発せられる音、楽音、電子音、楽曲などを録音、加工し、再構成を経て創作される。

- ピエール・シェフェール: etude aux chemins de fer (1978)
  - https://www.youtube.com/watch?v=N9pOq8u6-bA

- ルイージ・ロッソ：Intonarumoris (1913)
  - https://www.youtube.com/watch?v=ReUL6KKGLGg

- ビートルズ： Revolution 9（1968）
  - https://www.youtube.com/watch?v=SNdcFPjGsm8

モジュール化

- 神戸大学MBA ビジネス・キーワード
  - モジュール化
  - https://mba.kobe-u.ac.jp/business_keyword/8000/

  - … 「モジュール化」とは、全体システムを、いくつかの下位システム（モジュール）にわけ、モジュール間のインターフェイスを標準化することによって、システム全体の構造を変革することなく．．．

  - …メリットの1つは、全体システムの変革に際して、変革を局所化することにより変革に伴う効率のロスを小さくできるところにある。…

  - …全体システムから時間的および空間的に切り離しての独立した設計を可能にする…

モジュール

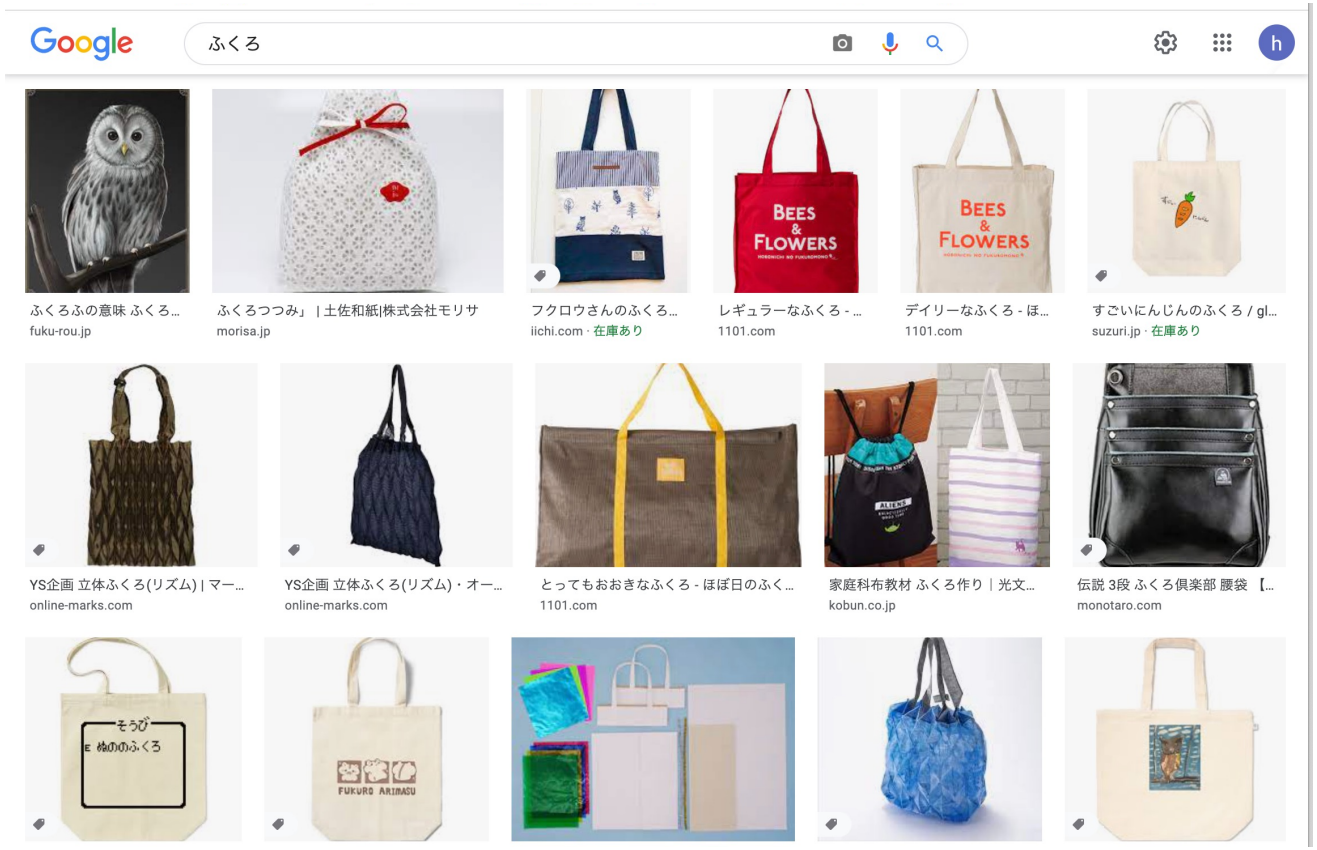
- IT用語辞典 e-words
  - https://e-words.jp/w/モジュール.html

- **モジュール**とは、機能単位、交換可能な構成部分などを意味する英単語。機器やシステムの一部を構成するひとまとまりの機能を持った部品で、システム中核部や他の部品への接合部（**インターフェース**）の仕様が明確に定義され、容易に追加や交換ができるようなもののことを指す。

- **モジュール化**とは、機器やソフトウェア、システムなどを設計する際に、全体を機能的なまとまりである**モジュール**（module）の組み合わせとして構成する手法。

BAG: バッグ，鞄，ふくろ

- ふくろ【袋・嚢】
  - 中に物を入れて、口をとじるようにした入れ物。紙・布・革などでつくる。



BAG: 多重集合（multiset）

- 集合に同じ値の元（要素）がいくつも含まれるとき、各元がそれぞれいくつ含まれるかという重複度を考え合わせた集合概念である。

- 非順序対、非順序組 (unordered tuple) ともいう。

抽象データ型（ADT: abstract data type）

- 実現の制約を取り去って，表現しようとする物の本質的な特徴を抽出したデータ値とプロセス（演算）を仕様に持つデータ型

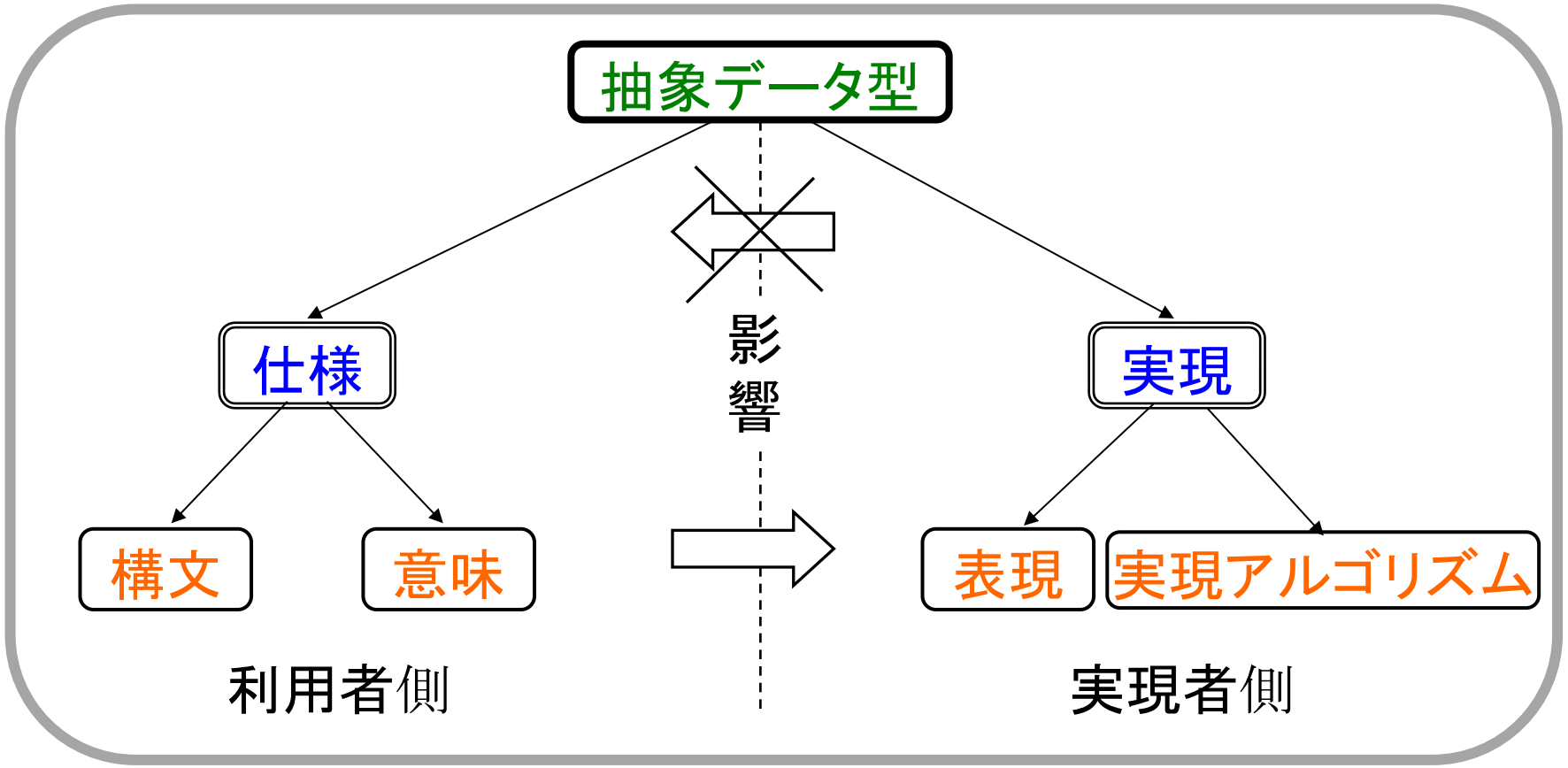
- 抽象データ型は，仕様と実現を分離して仕様に本質的特徴だけ記述する

- 実現を隠す機能 =>情報隠蔽，カプセル化

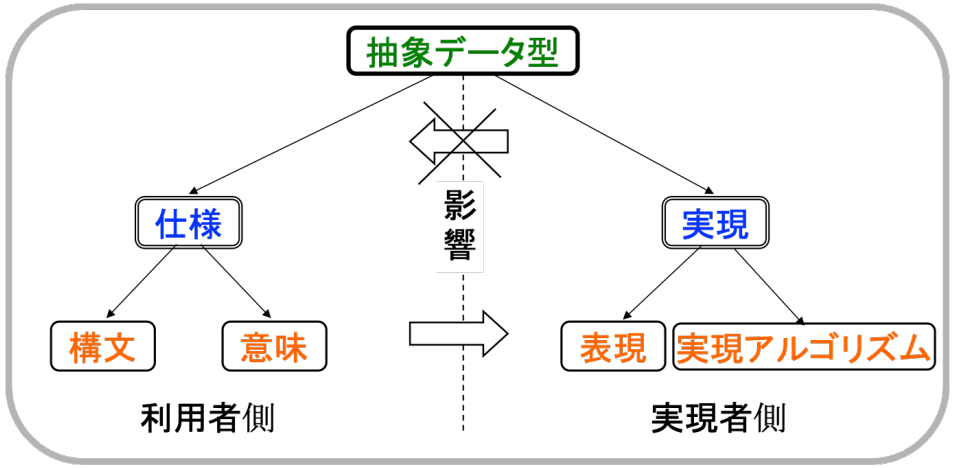
仕様と実現の分離

- 抽象データ型を支える本質的な考え:
  - 『データ型の仕様と実現とを分離すること』
- 仕様の設計の際:
  - 「利用者側から見た使い方、即ち、データ型の本質的特徴」だけを考慮
  - 「実現の様々な制約」の影響を受けないようにする。
- 利用者は、仕様だけを知っていればよい
  - 実現方法(表現、実現アルゴリズム)を知る必要はない。
- 実現は仕様を満たす必要がある
  - これを確かめることを検証(verification)という。

抽象データ型の仕様と実現



仕様（specification）

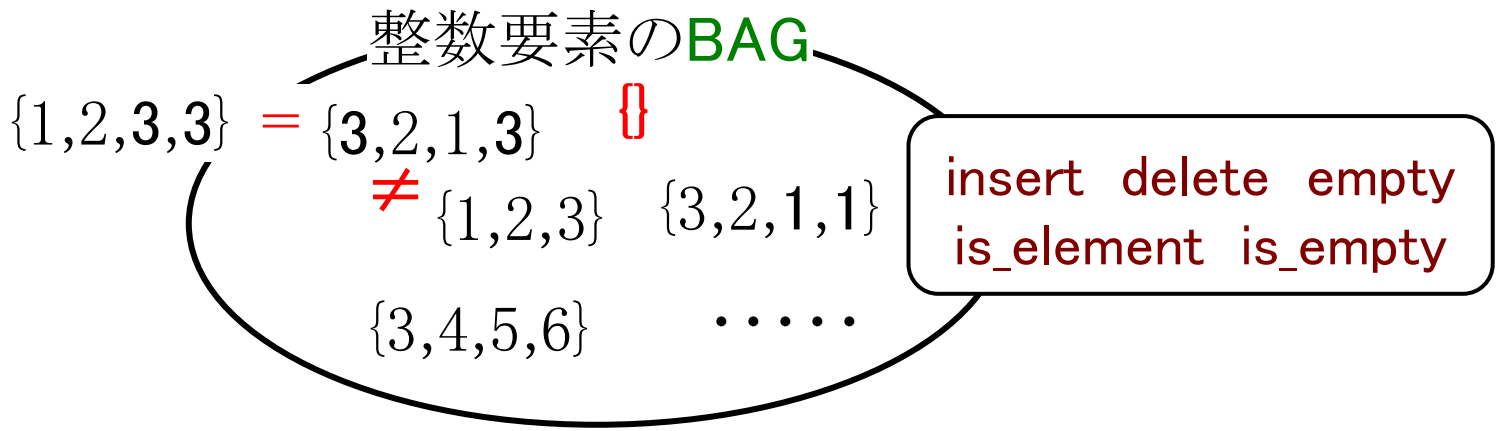


- 抽象データ型の仕様
  - 利用者側における抽象データ型の使い方
    - どのようなもの(what)かを定めるもの
  - 仕様にデータ型の本質的な特徴を記述する
  - 実現者側は、この仕様の下で実現を行う

- 仕様：構文と意味とからなる。
  - 構文(syntax)
    - データ型名、演算子、関数名、被演算子及び返り値の型、被演算子の個数と並び方を定める。
  - 意味(semantics)
    - データ値の集合、データ値の構造、演算子の振舞を定める。

【例】抽象データ型BAGの仕様

- BAG（多重集合型）とは
  - 同じ型の順序付けされていない要素の集まり
  - 同じ値の要素の重複を許す
  - 要素の個数が0のものを空のBAG（empty）という
  - 演算：
    - 要素の追加・削除の関数（insert, delete）
    - 要素の含意および要素数が0かを確かめる述語（is_element, is_empty）



【例】BAGの仕様＝構文＋意味

- 構文(syntax)を定める
  - データ型名：BAG
  - 演算子名：insert, delete, empty, is_element, is_empty
  - 被演算子の個数、並び方と型、返り値の型
    - insert: 第1引数 int型、第2引数 BAG型、返り値 BAG型
      - 宣言 BAG insert (int, BAG)
    - is_empty: 引数は BAG 型、返り値は Bool 型
      - 宣言 Bool is_empty(BAG)

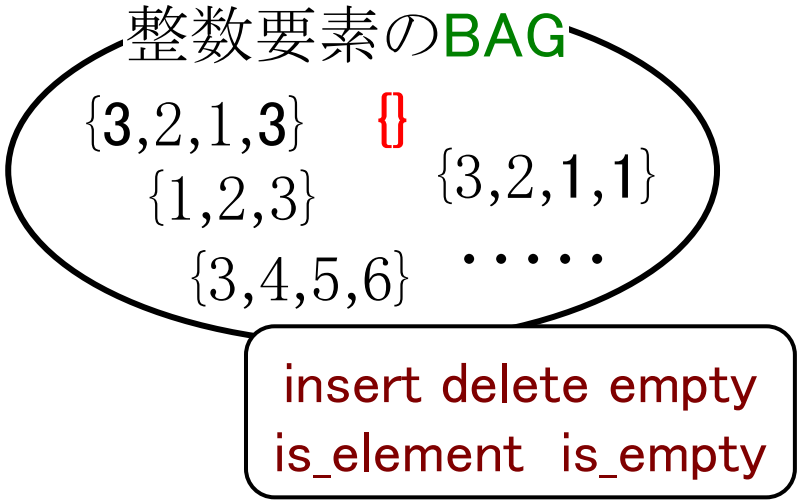
【例】BAGの仕様＝構文＋意味

- 意味(semantics)を定める
  - データ値の集合
  - データ値の構造

    {3, 2, 1, 3} ＝ {1, 2, 3, 3}
    - 「要素の並び方は問題にしない」

    {3, 2, 1, 3} ≠ {1, 2, 3}
    - 「重複要素を許す」
  - 演算子の振る舞い

    insert(2, {3, 2, 1, 3}) ＝ {2, 3, 2, 1, 3}
    - 第1引数の値を第2引数に追加した値を返す

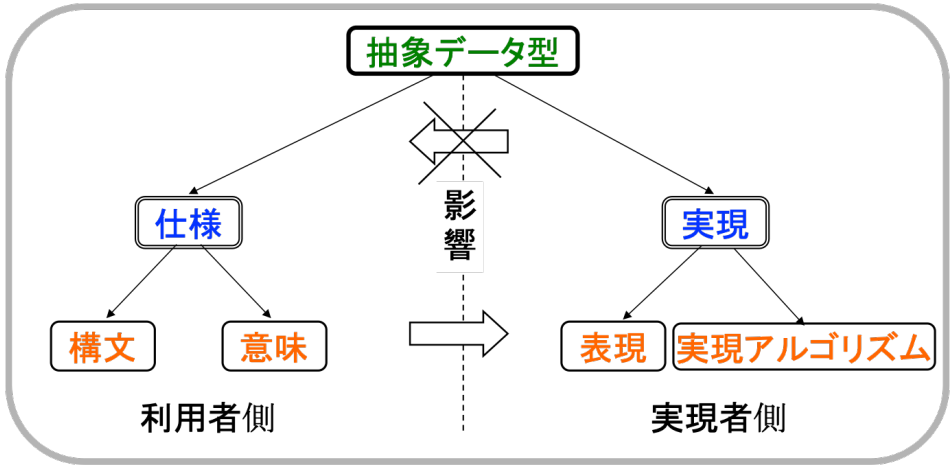


- 日本語による意味の記述・・・・・ 曖昧、不正確

  ⇒ 数学による仕様記述（形式的仕様記述）

実現（implementation）

- 抽象データ型の記述を行うプログラミング言語（記述言語）やデータ構造の表現を定め，どのように記述(how)するかを決めるもの。
- 利用者側は知る必要がない。
- 実現する計算機システム上のプログラミング言語およびアーキテクチャに依存（実現の制約）。
- JavaやOCamlなどは、データ抽象機能をもつ、すなわち、言語自身で実現の記述が行える。
- 実現は、表現と実現アルゴリズムから成る



表現（representation）

- 表現とは、抽象データ型のデータ値をこれより低いレベルのデータ型で表わすこと
  - データ構造に依存しない操作方法の提供
  - 実現（実装）方法が変わっても利用者プログラムの変更が必要ない
- 計算機システム上の記述言語のデータ型を用いて表す

実現アルゴリズム（implementation algorithm）

- 実現アルゴリズムとは、演算子の**具体的な実現**であり、通常、**記述言語の手続きや関数**で記述する。
- JavaやOCamlは、言語自身で実現を記述できる（インタフェースやモジュール）ため、実現アルゴリズムは、手続きや関数の**詳細化**(refinement)となる。

【例】抽象データ型BAGの実現

- 実現の指針
  - BAG型引数、関数値の渡し方
    引数の渡し方
      - 番地呼び
      - 値呼び

    関数値の返し方
      - ポインタで返す
      - 値で返す

◎引数を値呼びで渡し、値で返せば安全（これを採用）
⇒ BAG値全体のコピーのため、時間と領域を多く要する

- もし，引数を番地呼びで渡したら
  - 引数値をコピーする必要がないため、時間と領域が少なく済む
    ただし、引数に修正を加えると、元のBAG値が変化する。
- もし，関数値をポインタで返したら
  - 被参照変数は、ヒープ領域上か、番地呼び実引数を用いる。

【例】BAGの実現＝表現＋実現アルゴリズム
C言語のデータ型で表す

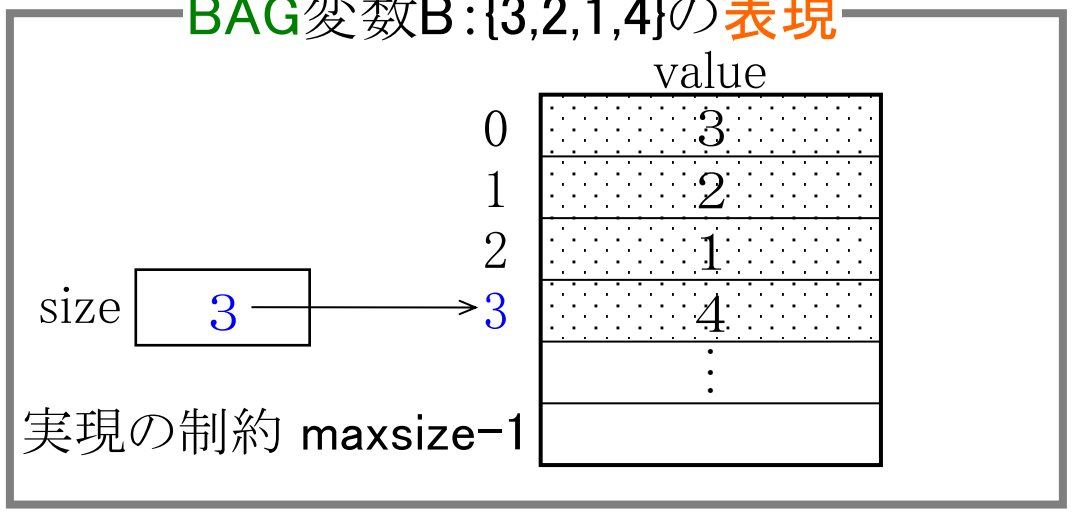
- Cの構造体型によるBAGの表現

```
#define maxsize 100

typedef struct {
    int value[maxsize];
    int size;
} BAG;

BAG B;
```

BAGの要素数の最大を表す
配列による実現の制約



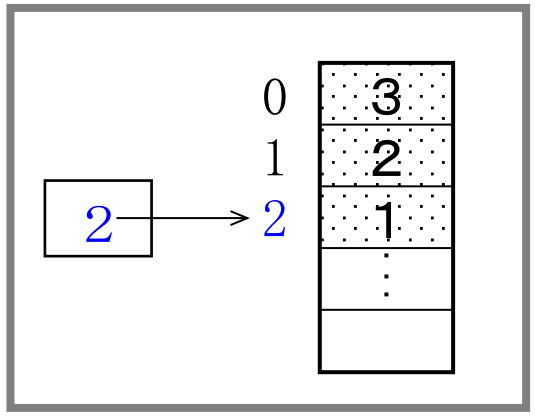
【例】BAGの実現＝表現＋実現アルゴリズム

C言語の手続きでinsertを記述

- BAG引数を値呼びで渡し、演算後の値を関数値で返す

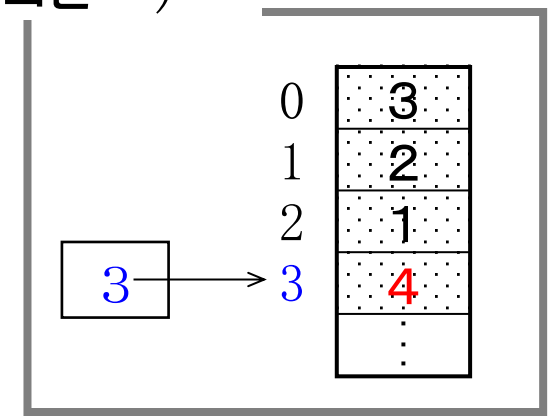
```
BAG insert ( int e, BAG B ){
    if ( B.size < maxsize-1 ) {
            B.size = B.size + 1;
            B.value[ B.size ] = e;
    } … 要素を追加
    else /*要素が一杯の場合の記述*/;
    return B;          … 関数の返り値設定（構造体は値返し可能）
}
```

局所的な仮引数Bの値:{3,2,1}



insertの関数値:{3,2,1,4}
（Bの値のコピー）

insert(4,B)



【例】BAGの実現=表現+実現アルゴリズム

C言語の手続きで補助関数searchを記述

…演算子delete、is_elementで使う補助関数

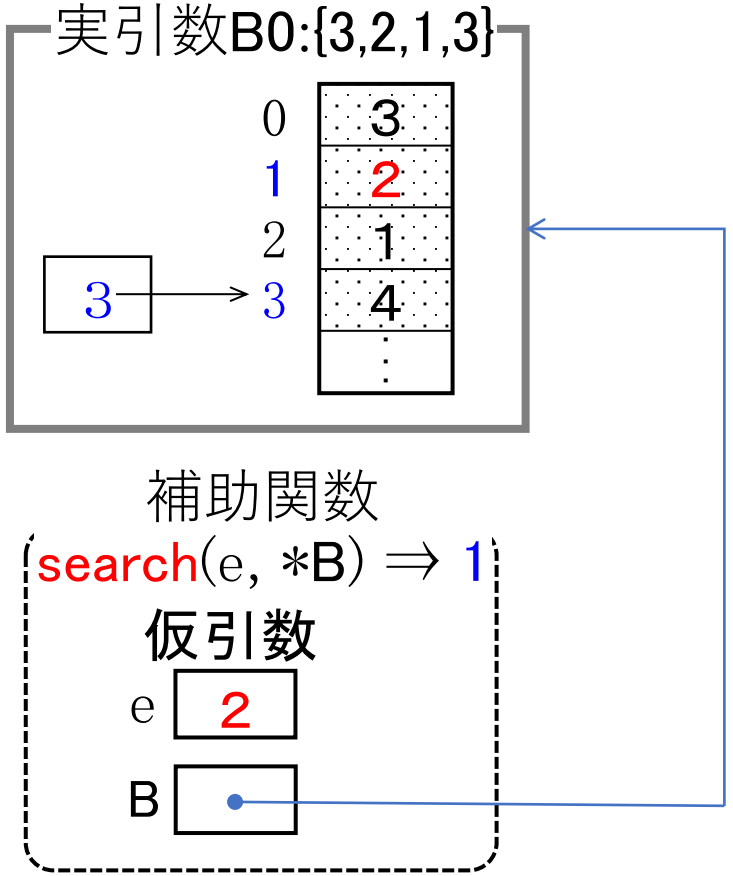
```
int search ( int e, BAG *B ){
        int i=0;
        while( (B->value[i] != e ) && (i <= B->size ) )
                i=i+1;
        if ( i<= B->size )
                return i;
        else
                return -1;
}
```

/* Bは**変更されない**ので、**番地呼び**で行い、時間と領域を**節約** */

… eがB[i]で見つかった!

… eが見つからなかった!

呼び出し側 search(2,&B0)=1



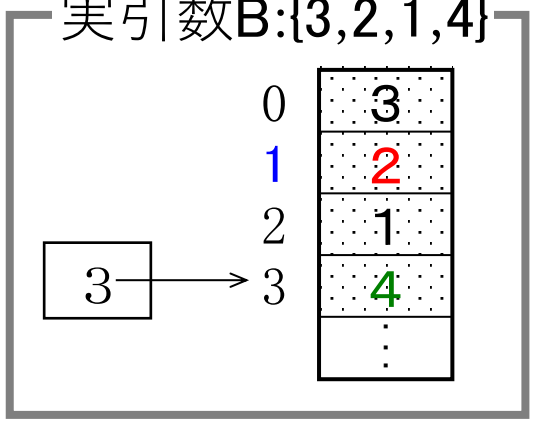
【例】BAGの実現=表現+実現アルゴリズム

C言語の手続きでdeleteを記述

値呼びと値返し

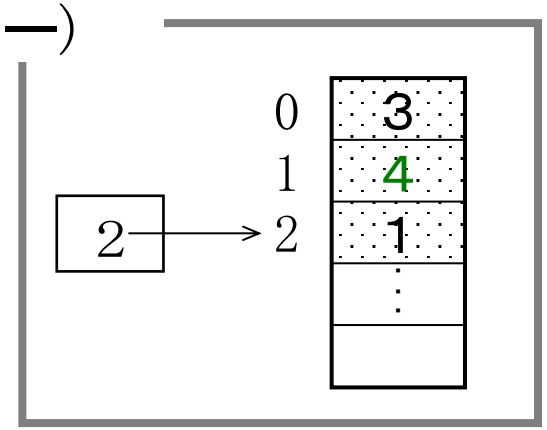
補助関数searchを使用

```
BAG delete(int e, BAG B) {
        int i;
        i = search( e, &B );
        if( i != -1 ){
                B.value[i] = B.value[ B.size ];
                B.size = B.size-1;
        }
        return B;
}
```



delete(2,B)

deleteの関数値:{3,4,1}
(Bの値のコピー)

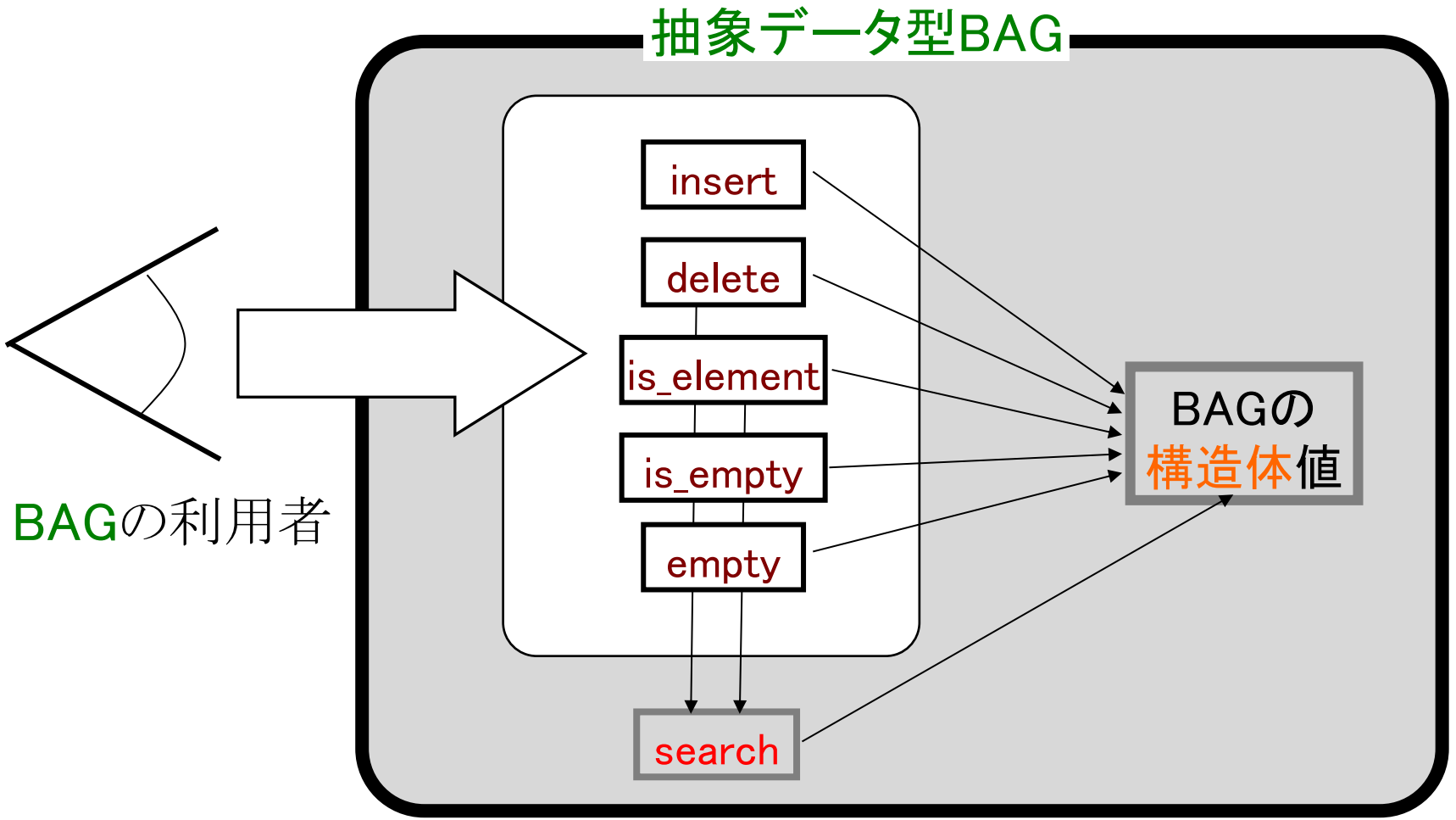


情報隠蔽

- 実現：抽象データ型BAGの実際の値
  ⇒ 構造体型等で定義
- 仕様： 抽象データ型BAGの値
  ⇒ insert, delete, is_element, is_empty, empty だけで演算

- 抽象データ型利用者は、構造体を直接読み書きできない
- 演算delete, is_elementの補助関数searchを直接使えない
  - 情報隠蔽(information hiding) … 抽象データ型
  - カプセル化(encapsulation) … オブジェクト指向
  - 保護(protection) … 代数仕様

BAGの情報隠蔽


抽象データ型BAG
insert
delete
is_element
is_empty
empty
search
BAGの
構造体値
BAGの利用者

whatとhow

- **情報隠蔽**は、**外部**（利用者）に対し、
  - これらの**5つの演算**を通して
  - 『**何を**(what)**するか**を見せるだけで、
  - **どのようにして**(how)データ型を**実現**しているかは知らせない』

- **抽象データ型**の**仕様**で**定義した性質を護る**ことができる

- 外部に5つの演算を見ることを許している
  - これを「**輸出する**(export)」という。

データ抽象機能

- **データ抽象機能**:
  - **抽象データ型**を**定義する機能**
    - **モジュール化**や**情報隠蔽**など
- OCaml, Java, Adaなどは充分なデータ抽象機能をもつ
  - カプセル化, モジュール化
- C, Pascalは、情報隠蔽の機能をもたない
  ⇒ BAGの値の**直接の読み書き**が出来てしまう！

- **プログラミング言語**の**仕様**は、
  - **構文記述**をもつが、**意味記述**はもたない！
  - 一方、**形式的仕様**は、**意味記述**をもつ

オブジェクト指向

- データを自律的なオブジェクトとしてとらえ，計算をオブジェクト間のメッセージ交換によって進むものとしてとらえる

- オブジェクト指向は，抽象データ型に動的分配やクラス間の継承などのツールを追加したものと考えることができる

- プログラミング言語 Simula 67に端を発しておりSmalltalk 言語によって広く知られるようになった.
- 今日では C++，Javaをはじめメインストリー ムの手続き型言語の多くがオブジェクト指向の機能を備える

オブジェクト指向言語の特長

- **メッセージパッシング**
  - Smalltalk（ゼロックス社パロアルト研究所）で採用
  - 定数、変数、プログラム等をすべて**オブジェクト**とし，オブジェクトに**メッセージ**を送り計算するプログラミングスタイル
  - Javaでは：
    - Obj.method(p)
  - オブジェクトObjに，メッセージ.method(p)（メソッドmethodと引数pからなる）を送る
- **動的なインスタンス生成**
  - SIMULA67[O.J.Dahl]で採用
  - **クラス**という**変数、手続き、仮引数**などをもつ**モジュール**
  - クラス名Cの**動的インスタンス**を new (C(**実引数**)) で生成する
  - newで自由に**いくつも**インスタンス生成が可能

オブジェクト指向言語の特長

- **抽象データ型**
  - データ型を**データ型名**と**演算子**とで表し、**データ型**に共通する**性質の意味**を**演算子で定義**したもの
  - 70年代に現れる**代数仕様**は、形式的に**抽象データ型**を記述するもの
- **継承**（インヘリタンス）
  - あるオブジェクトが他のオブジェクトの特性を引き継ぐこと
  - SIMULA67で、**クラス**を**別の**クラスの**部分クラス**として定義できるようになった
- **ポリモルフィズム**（多相性、多態性）
  - プログラミング言語の各要素が複数の型に属すること
  - **異なるオブジェクト**に対し**メソッド名M**のメッセージを送ると異なる反応が返ってくる
  - これは、各オブジェクトに**同じ名前**Mのメソッドが**所属できる**ためである。
- **オーバーライド**
  - **スーパークラス**で定義された**メソッド**を**サブクラス**中の**同名のメソッド**で**再定義**して、動作を**上書き**すること
  - 仕様中の動作の実現に使える。

まとめ

- 抽象データ型とは
  - 仕様に本質的特徴だけ記述する
  - **仕様で定義した性質を護る** => 情報隠蔽, カプセル化

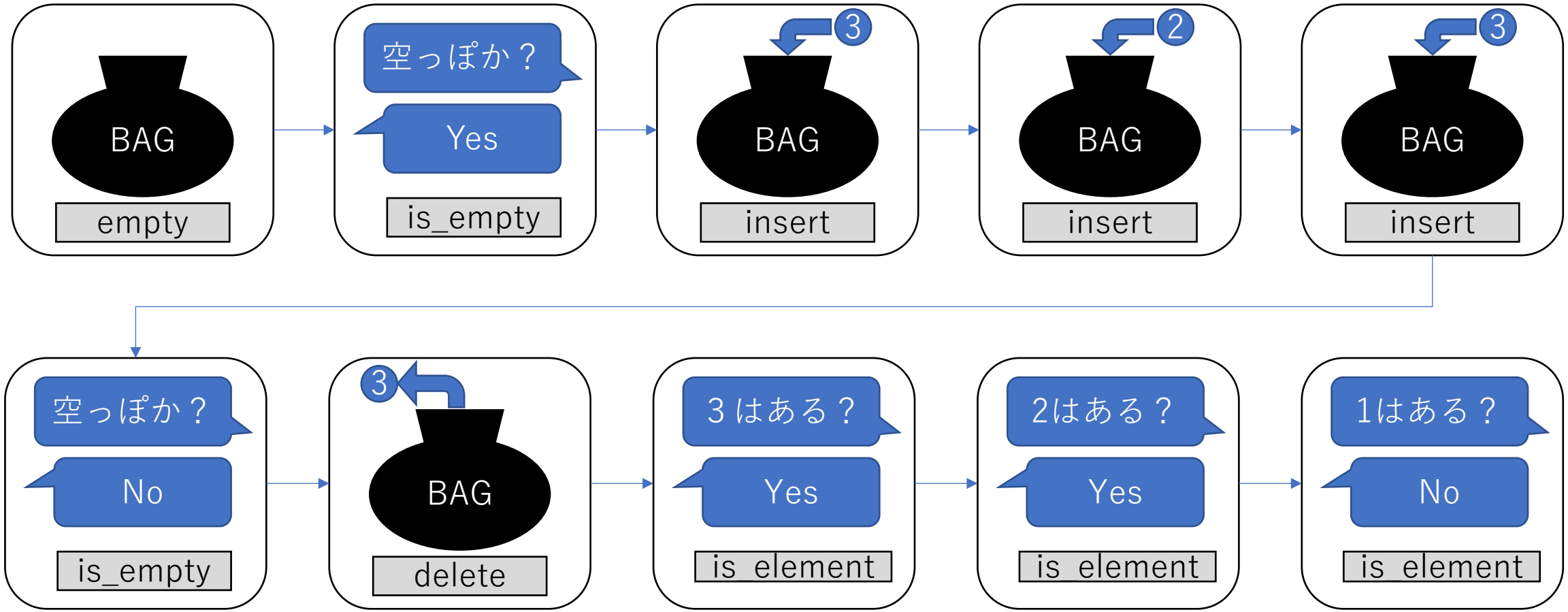
- 抽象データ型 **BAG の実装**

- **オブジェクト指向**
  - データは自律的なオブジェクト
  - 処理はオブジェクト間のメッセージ交換

宿題

- C言語でBAGの抽象型を完成させて下さい.
- BAGの構造体，insert, search, deleteは授業内容と同じでかまいません.
- empty, is_element, is_emptyの処理を考えて下さい.
- main関数内で各関数を適当に動かして下さい.
- 【注意】
  - C言語は抽象型を作成できません,
  - ここまでの実装が抽象型である事を意識してください.
  - 隠蔽機能はないので，隠蔽などは気にしないでください
- 提出はLetusで行って下さい.
- ファイル名：bag.c
- 提出期限：2023/5/8 10:30まで

BAGの抽象構造型（main関数）



空のバックを作成，○を入れる，○を取り出す，のみで操作
「空っぽか？」「○はある？」の情報だけで，BAGの中身は見せる必要はない

bag.c のひな形（この通りでなくてもOK）

```c
#include <stdio.h>
#include <stdbool.h>
#define maxsize 100

typedef struct{
    int value[maxsize];
    int size;
}BAG;

BAG insert(int e, BAG B){
    /* 処理を書く */
}

int search(int e, BAG *B){
    /* 処理を書く */
}

BAG delete(int e, BAG B){
    /* 処理を書く */
}

BAG empty(){
    /* 処理を書く */
}
```

```c
bool is_element(int e, BAG *B){
    /* 処理を書く */
}

bool is_empty(BAG *B){
    /* 処理を書く */
}


int main(void){
    BAG B = empty();;
    printf("is empty? : %d\n",is_empty(&B));
    B = insert(3, B);
    B = insert(2, B);
    B = insert(3, B);
    printf("is empty? : %d\n",is_empty(&B));;
    B = delete(3, B);
    printf("%d is element?: %d\n", 3, is_element(3, &B));
    printf("%d is element?: %d\n", 2, is_element(2, &B));
    printf("%d is element?: %d\n", 1, is_element(1, &B));
    return 0;
}
```