

# プログラミング言語論

## 第4回 抽象データ型

担当: 犬塚

1

### 今日の講義

データ抽象に関する事柄を見る。

- データ抽象とは
- 抽象データ型とは
- 抽象データ型の諸概念、利点
- ADTのバリエーション
  - 引数つきの抽象データ型
  - モジュール単位のカプセル化
- データ不变条件

2

### 抽象 abstract, abstraction

- 抽象 = **抽出** = 何かを取り出すこと  
= **捨象** = いくつかを捨て去ること
- 反対語は、**具体的**、**具象** concrete
- ものを抽象的に表すというのは、その本質(あるいは注目点)のみを残してそれ以外を捨てること。
- 工学ではものを「作る」のに、その本質(機能、性質)のみに注目して、他を忘れる(忘れても大丈夫な仕掛けを用意する)ことで大規模な構築物を得る。

3

### プログラミングでの2つの抽象

- 抽象はプログラミングで2つの側面から使われる  
= 具体的な実現方法を捨象して、機能に注目。
- **プロセスの抽象** process abstraction
- **データ(型)の抽象** data abstraction

4

## プロセスの抽象 process abstraction

- 一連の手続きを括りだして、抽象化する。  
=サブルーチン、手続き、関数
- そのルーチンの実現方法(=実装)を抽象し(気にせずに)次のことのみに注目:
  - そのルーチンの機能
  - そのルーチンの呼出し方:プロトコル、インターフェース

例 sortのルーチン

機能:リストを順番に並べる

呼出し方:sort(リスト, リストの長さ)

順番に並べるための方法は気にしない。

5

## データ抽象 data abstraction

- データの情報内容と情報内容の操作のみを気にすればよい仕掛け。
  - どのようにメモリー内で記憶されているか、
  - どのような型定義によって実現されているか、は気にしない。
- これを実現するための考え方:
  - 抽象データ型 Abstract data type
  - オブジェクト指向プログラミング  
Object-oriented programming

6

## 抽象データ型 ADT:abstract data type

- 抽象データ型は次の2組からなる:
  - データ型(その型の変数の取りうる値の範囲)
  - そのデータ型のデータに対する操作(手続き、関数)
- 抽象データ型を定めると、
  - クライアントはそのデータ型を宣言することができる。
  - 与えられた手続きによってデータを操作できる。
- ADTをサポートする言語は、
  - ADTの具体的な実現方法(=その抽象データ型の実装)を記述できる。
  - ADTの実装を隠蔽することができる。

※クライアント=ADTを利用する側のルーチン

7

## 抽象データ型の例

- スタックを扱うADT stack型を考える。
- クライアントは、stack型の変数を宣言できる。  
`var stk1: stack;`
- stack型のデータを操作する手続きがある:
  - `create(stk1);` stk1を新たなスタックとして用意し、初期化する。
  - `destroy(stk1);` stk1の使用をやめる(メモリーを解放する)
  - `empty(stk1);` stk1を空のスタックに初期化する。
  - `push(stk1,elm);` 要素elmをstk1に積む。
  - `elm := pop(stk1)` stk1に積まれている最上位の要素を取り出す。

8

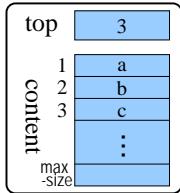
## スタック型の実現(1)

```
type stack = record
    top : integer;
    content : array[1..maxsize] of element
end;

{proc create(s : stack);
    prepare memory for s;
    s.top := 0;
}

{proc destroy(s : stack);
    release memory for s;
}

proc empty(s: stack);
    s.top := 0;
proc push(s:stack; e:element);
    if top=maxsize then error
    else s.top:=s.top+1; s.content[s.top]:=e;
func top(s:stack):element;
    if s.top=0 then error
    else s.top:=s.top-1; return s.content[s.top+1];
```

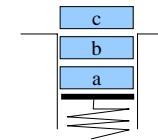
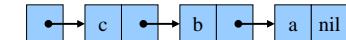


9

## スタック型の実現(2)

```
type cell = record content : element;
    next : pointer to cell
end;
stack = pointer to cell

proc create(s : stack);
    s=nil;
proc destroy(s : stack);
    var t : pointer to cell;
    while s<>nil do
        t := s.next; release(s); s:=t;
proc empty(s: stack);
    destroy(s);
proc push(s:stack; e:element);
    var t : pointer to cell;
    new(t); t.content:=e; t.next:=s;
func pop(s:stack):element;
    var t : pointer to cell; e : element;
    e:=t.content; t:=s; s:=t.next;
    release(t); return e;
```



10

## スタックの操作

- ADTとして定義されたデータは、実現法によらず操作できる必要がある。

```
var stk1 : stack
create(stk1);
push(stk1,a);
push(stk1,b);
x:=pop(stk1);
push(stk1,c);
push(stk1,d);
x:=pop(stk1);
```

11

## 基本データ型とADT

- 基本データ型は元々ADTである。

整数型、浮動小数点型、…

- 浮動小数点型が、どのようなビットパターンで実現されているか、普段は気にしない。
- 浮動小数点型のデータを操作する演算がはじめからよういされている。
- その演算を使用せず、直接ビット操作をすることは許されていない。

- 基本データ型はクライアントがその実現を気にせずに使うことができる、組み込みのADTである。

12

## 抽象データ型の利点

- カプセル化 encapsulation
- 情報隠蔽 information hiding  
クライアントに必要な情報のみにアクセスを許す。
- 実現の隠蔽 implementation hiding
- 表現独立 representation independence  
データ構造に依存しない操作方法を提供する。  
実装方法が変わってもユーザプログラムの変更が要らない。
- データ抽象は考え方であり、実際には上述の概念を提供する仕掛けを与えることが目標。

13

## カプセル化 encapsulation

クライアント側に必要な情報のみを見せる。

- 見せるもの:
  - データ型の名称
  - データ操作する手続きのインターフェース(プロトコル)  
手続きを用いる場合の引数の指定方法、返り値。
- 隠すもの:
  - データ構造
  - 手続きのルーチン: 実装(implementation)

14

## Ada の例

```
package Stack_Pack is
  -- The visible entities, or public interface
  type Stack_Type is limited private;
  Max_Size : constant := 100;
  function Empty(Stk : in Stack_Type) return Boolean;
  procedure Push(Stk : in out Stack_Type;
                 Element : in Integer);
  procedure Pop(Stk : in out Stack_Type);
  function Top(Stk : in Stack_Type) return Integer;
  -- The part that is hidden from clients
  private private はクライエントへの非公開
  type List_Type is array (1..Max_Size) of Integer;
  type Stack_Type is
    record
      List : List_Type;
      Topsub : Integer range 0..Max_Size := 0;
    end record;
end Stack_Pack;
```

クライアントへ公開

手続きのインターフェースのみ公開

非公開

15

## Ada の例(つづき)

```
with Ada.Text_IO; use Ada.Text_IO;
package body Stack_Pack is
  function Empty(Stk: in Stack_Type) return Boolean is
    begin
      return Stk.Topsum = 0;
    end Empty;

  procedure Push(Stk : in out Stack_Type;
                 Element : in Integer) is
    begin
      if Stk.Topsum >= Max_Size then
        Put_Line("ERROR - Stack overflow");
      else
        Stk.Topsub := Stk.Topsub + 1;
        Stk.List(Topsub) := Element;
      end if;
    end Push;
```

手続きの実装  
(本体)

pop, topは省略

16

# モジュール単位のカプセル化

一組の機能を実現するADT  
のパッケージ、モジュール

- **公開部分:**
  - ADTのデータ型(名称)
  - ADTについて定数、変数
  - ADTの操作手続き、関数のインターフェース
- **非公開部分:**
  - 内部で必要な他のデータ型
  - 内部で必要な変数、定数
  - 内部で必要な手続き、関数のインターフェース
  - ADTの操作手続き、関数の実装
  - 内部で必要な手続き、関数の実装

パッケージ(モジュール)単位で必要な部分のみを隠蔽する機能、  
公開する相手(モジュール)を指定する機能等がある。

17

## 例

- stack型
- maxsize
- push, pop, ...の入出力
  
- cell
- top cellやbottom cellへのポインター
- 新しいcellをつくる手続き newcell
- push, popの実装
- newcellの実装

# 引数つきのADT parameterized ADT

- 整数のスタック、文字列のスタックなどは別の型。maxsizeが異なれば、これも別の型。
- これを別々に定義せず、要素型, maxsize  
→ その要素型のmaxsizeのスタック  
というADTが想定できる：引数つきのADT
- Ada, C++などで利用できる。  
cf. C++のtemplate

18

# データの安全性とデータ不变条件

- ADTはデータの操作を**安全な操作**に限定している。
- データが安全とは、予め決められた条件を満足していること。
- データの操作中以外は、この条件は常に満足しているべき。
- これを**データ不变条件(data invariant)**という。

例 stack型のデータ不变条件:

- $0 \leq \text{top} \leq \text{maxsize}$ 。
- $\text{top} = 0$ ならば、スタックは空。
- $\text{top} = \text{maxsize}$ ならば、スタックは満杯。
- $0 < \text{top} < \text{maxsize}$ なら、top個のデータが插入順と逆に入っている。

- データ不变条件中心にADTを設計するとよい。
  - データ不变条件を決定する。
  - データの操作を決定する。ここまででADTが決まる。
  - データ構造、操作の実装を決める。
- ADTの理論的基礎づけ:公理的、代数的記述と関連

19

# まとめ

- 抽象はプログラミング(だけでなく、工学)での重要概念。
- ADTはデータの機能面に注目し、実装を隠す。
- ADT=データ型+データに対する操作
- ADTは、
  - クライアントに実装を隠蔽、プログラム開発用容易にする。
  - 実装の置き換えを容易にする。
  - 安全な操作のみをクライアントに許し、データ不变条件を守る。

20