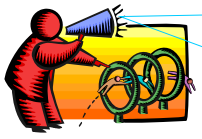
Programming Methodology
程序设计方法学

第4章　数据类型和抽象
——大型程序设计方法学基础

主讲：刘悦
yliu@staff.shu.edu.cn

回顾



- 程序正确性简介
- 程序测试
- 程序正确性证明

本章目标

?什么是类型\数据类型\数据抽象
?引入抽象的目的及意义
？学会用代数化方法设计抽象数据类型
？能够给出抽象数据类型的实现
？能够证明实现的正确性

主要内容

- 类型
- 数据类型
- 抽象数据类型
- 数据抽象及其代数规范
- 大型程序设计与抽象数据类型

内容线索

- 类型
- 数据类型
- 抽象数据类型
- 数据抽象及其代数规范
- 大型程序设计与抽象数据类型

类型

- 类型（type）是一组值的集合。
  - 例如：
    - 布尔类型由两个值TRUE和FALSE组成
    - 整数也构成一个类型

内容线索

- ✓类型
- 数据类型
- 抽象数据类型
- 数据抽象及其代数规范
- 大型程序设计与抽象数据类型

数据类型

- 数据类型（data type）是指一个类型以及定义在这个类型上的一组操作。
  - 数据类型=允许的数据+数据的操作能力
    - 例如，一个整数变量是整数类型的一个成员
  - 线性表
    - 链表（linked list）
    - 顺序表（array-based list，基于数组的线性表）
    - 因此，可以在链表或者数组之间选择一种方式来实现线性表数据类型。

为什么要引入数据类型

- 为更好地描述实际问题,更好地用程序设计方法来解决它们
  - 带来了程序的简明性和数据的可靠性
    - 数据说明部分:能够进行类型匹配检查
  - 编译系统对不同类型的数据预先分配为存储变量的值所需要的存储空间,在程序执行时就会提高执行效率,节省存储空间

关于数据类型

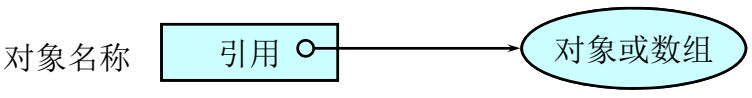
- 1、有几种数据类型
  - 不同高级语言有不同的分类
  - Java有两种数据类型
    - 原始数据类型
    - 引用数据类型
- 2、如何定义数据类型
  - 标志符
  - 表示法
  - 取值范围
  - 值表示法
- 3、如何使用
  - 可对它进行什么运算操作
  - 如何对它进行读写操作和赋值操作

Java原始数据类型

		数据类型	大小/格式	说明	范围
数值型	整型	byte	8位	字节长整型	有符号:-128到+127(-2^7到2^7-1) 无符号:0到255
		short	16位	短整型	-32768到+32767(-2^{15}到2^{15}-1)
		int	32位	整型	-2^{31}到2^{31}-1
		long	64位	长整型	-2^{63}到2^{63}-1
	浮点型	float	32位	单精度浮点	+/- 10^{39}
		double	64位	双精度浮点	+/- 10^{317}
	字符型	char	16位	单字符	
	布尔型	boolean	1位	布尔值	true或false

Java引用数据类型

- 引用类型
  - 数组
  - 类
  - 接口
  - 与原始类型比较
    - 引用类型变量的值是对由此变量代表的一个值或一组值的引用



引用类型变量包含对一个对象或数组的引用

数据类型的属性

- 属性1:类型定义了变量或表达式所能取的值的集合
- 属性2:某种类型中的值仅属于此类型
- 属性3:对一种运算操作，要求一定类型的操作数，并得到一定类型的结果数据
- 属性4:对每种类型只能进行规定的操作

实例

```
boolean p;
int i;
```

- 属性1
  - p:true/false
  - i:-2^{31}到$2^{31}-1$
- 属性2：true/false仅属于布尔类型
- 属性3、4：
  - 布尔型：条件操作符&、|、^、！ 、&&、||
  - 整型：算术操作符+ 、-、*、/、%
  - 整型，布尔型：关系操作符<、 <= 、>、>=、==、!=

内容线索

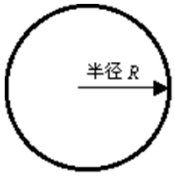
- ✓ 类型
- ✓ 数据类型
- 抽象数据类型
- 数据抽象及其代数规范
- 大型程序设计与抽象数据类型

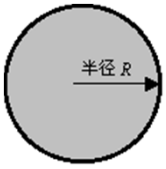
抽象数据类型

- 抽象数据类型（abstract data type，简称ADT）通常是指对数据的某种抽象，它定义数据的取值范围及其结构形式，是对数据的操作的集合。
  - 抽象数据类型是算法设计和程序设计中的重要概念。
- 抽象数据类型描述数据的构造及使用，并不注重其在程序中如何实现。
- 抽象数据类型通过一种特定的数据结构在程序的某个部分得以实现，而在设计使用抽象数据类型的那部分程序时，我们只关心这个数据类型上的操作，而不关心数据结构的具体实现。因此，在思考一个复杂的程序时，首先应考虑能否将它抽象简化，否则很难理解或者实现它。

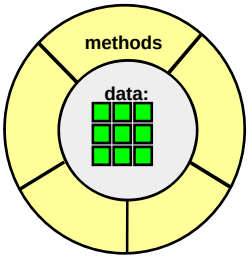
实例

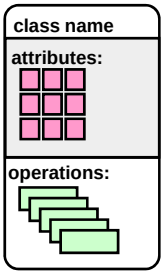
- 例如，整数的数学概念和施加到整数的运算构成一个抽象数据类型。
- 又如，圆是平面上与圆心等距离的所有点的集合。
  - 从图形显示角度看，圆的抽象数据类型包括圆心和半径；
  - 而从计量角度看，它所需要的抽象数据类型只需半径即可。
  - 如果从计量角度来给出圆的抽象数据类型，那么它的数据取值范围应为半径的取值范围，即为非负实数，而它的操作形式为确定圆的半径（赋值）；求圆的面积；求圆的周长。





抽象数据类型——类





使用抽象数据类型带来的好处…

- 使用抽象数据类型将给算法和程序设计带来很多好处：
  - 算法顶层的设计与底层的设计被隔开，使得在进行顶层设计时不必考虑它所用到的数据和运算分别如何表示和实现；反过来，在进行数据表示和运算实现等底层设计时，只要抽象数据类型定义清楚，也不必考虑它在什么场合被引用。这样做，算法和程序设计的复杂性降低了，条理性增强了。既有助于迅速开发出程序的原型，又有助于在开发过程中少出差错，保证编出的程序有较高的可靠性。
  - 算法设计与数据结构设计隔开，允许数据结构自由选择，从中比较，可优化算法和提高程序运行的效率。

…使用抽象数据类型带来的好处

- 数据模型和该模型上的运算统一在抽象数据类型中，反映了它们之间内在的互相依赖和互相制约的关系，便于空间和时间耗费的折衷，满足用户的要求。
- 由于顶层设计和底层设计被局部化，在设计中，如果出现差错，将是局部的，因而容易查找也容易纠正。在设计中常常要做的增、删、改也都是局部的，因而也都很容易进行。因此，可以肯定，用抽象数据类型表述的程序具有很好的可维护性。
- 编出来的程序自然地呈现模块化，而且，抽象的数据类型的表示和实现都可以封装起来，便于移植和重用。
- 为自顶向下逐步求精和模块化提供一种有效的途径和工具。
- 编出来的程序结构清晰，层次分明，便于程序正确性的证明和复杂性的分析。

内容线索

- ✓ **类型**
- ✓ **数据类型**
- ✓ **抽象数据类型**
- 数据抽象及其代数规范
  - 抽象数据类型的代数规范
  - 抽象数据类型的定义
  - 抽象数据类型的实现
  - 抽象数据类型实现的正确性
  - 代数规范的应用及其它
- 大型程序设计与抽象数据类型

数据抽象及其代数规范

- **分解：**将问题分成若干块，然后再求解。
- **抽象：**在任一时刻只考虑问题的某些与当时相关的重要属性，而暂且不考虑其他细节部分。
- **过程抽象：**实现从一组输入值到一组输出值的映射，其定义域和值域包括了数据抽象，其行为则由这些数据抽象的行为而定。
- **数据抽象：**针对某单一类型的操作聚集起来，与该类型一起看作一个独立的单元。而把操作的语义作为该类型的定义，这样就构成了一个数据抽象或称为抽象数据类型。
- **制定数据类型的规范方法：**代数化方法、状态机方法和抽象模型方法。
- **代数化方法的基本思想：**用各操作之间代数等式来描述一个数据类型。

抽象数据类型的代数规范

- 用一种代数规范语言对所要建立的抽象数据类型所进行的一种形式化的描述。一般包括：
  - 语法规范
    - 描述操作的类型
  - 语义规范
    - 描述操作的语义（公理）
  - 出错处理

obj 抽象数据类型名

sorts 所要定义的新类型/所要用到的原来的类型

ok-ops 抽象数据类型所能进行的操作的语法描述

每个操作的形参类型、结果类型

error-ops 对出错信息的语法描述

ok-eqn's 操作的语义描述

error-eqn's 出错信息的语义描述

Jbo

实例

- 无界栈的代数规范：P123，例1

obj stack;
sorts stack/integer, boolean;
ok-ops
  push:stack,integer → stack;
  pop:stack → stack;
  top:stack → integer;
  empty:stack → boolean;
  newstack: → stack;
error-ops
  underflow → stack;
  no-more → integer;

ok-eqn's
  pop(push(s,item))=s;
  top(push(s,item))=item;
  empty(newstack)=true;

empty(push(s,item))=false;
error-eqn's
  pop(newstack)=underflow;
  top(newstack)=no-more;
jbo

抽象程序的符号执行

```
begin
  integer result,stack s;
  push(s,5);
  push(s,3);
  pop(s);
  result:=top(s);
end
```

- 编程时用的是ok-ops和error-ops，程序执行以及如何理解程序应用的是ok-eqn's和error-eqn's
- push(s, 5)=push(newstack, 5)
- push(s, 3)=push(push(newstack, 5), 3)
- pop(s)=pop(push(push(newstack, 5), 3))
- =push(newstack, 5)
- result=top(s)=top(push(newstack, 5))=5

因为pop(newstack)=underflow，且underflow为stack类型，所以假定代数规范隐式地含有下述等式：
pop(underflow)=underflow;
push(underflow,item)=underflow;
top(underflow)=no-more;

抽象数据类型的定义

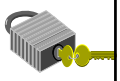
- 1）把所有该类型所能进行的操作找出来，同时把所有可能出错的信息找出来，再转换成ok-ops和error-ops
- 2）找出构造子集，定义其他操作与构造子集中操作的关系，放进ok-eqn's
  - 构造子集：该数据类型的所有示例均可仅用构造子集的操作表示出来。
- 3）确定出错信息的含义以及哪些操作可能引起出错，放进error-eqn's

同步练习

- 写出有界数组的代数规范。

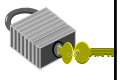
有界数组的代数规范（1）

- obj array;
- sorts array/integer;
- ok-ops
  - newarray:→array;
  - assign: integer, array, integer→ arrray;
  - read: array, integer→ integer;
  - number: array→ integer; hidden;
- error-ops
  - no-element:→integer;
  - overflow:→array;



有界数组的代数规范（2）

- ok-eqn’s
  - read (assign(val, arr, index1),index2)
    - = if index1=index2 then val
      - else read(arr,index2);
  - number(newarray)=0;
  - number(assign(val, arr, index))= index; （1分）
- error-eqn’s
  - read (newarray, index)=no-element;
  - assign(val, arr, index) = overflow
    - if index=number(arr)>100;
- jbo

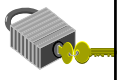


同步练习

- 写出有界栈的代数规范。

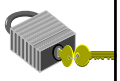
答案（1）

- obj stack;
- sorts stack/integer, boolean;
- ok-ops
  - push: stack,integer→ stack;
  - pop: stack → stack;
  - top: stack→interger;
  - empty: stack→boolean;
  - newstack:→ stack;
  - depth: stack → integer;hidden;
- error-ops
  - underflow:→ stack;
  - no-more: → integer;
  - overflow:→ stack;



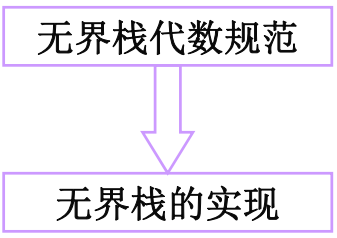
答案（2）

- ok-eqn’s
- pop(push(s,item))=s;
- top(push(s,item))=item;
- empty(newstack)=true;
- empty(push(s,item))=false;
- depth(newstack)=0;
- depth(push(s,item))=depth(s)+1;
- error-eqn’s
- pop(newstack)=underflow;
- top(newstack)=no-more;
- push(s,item)=overflow if depth(s)>=n;
- jbo



抽象数据类型的实现

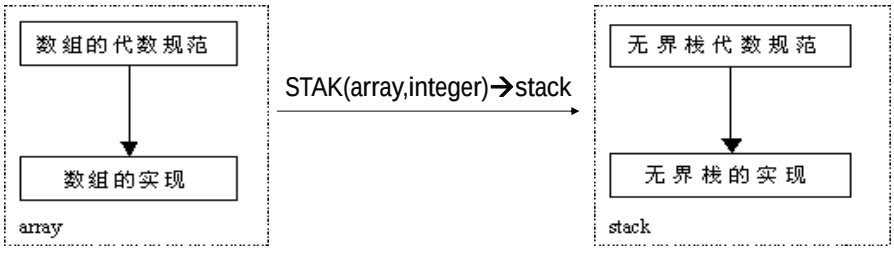
- 1）解决抽象数据类型的表示。
- 2）用选定的表示方式实现代数规范中的ok-ops部分所有操作。



无界数组的代数规范

obj array;
Sorts array/interger;
Ok-ops
  newarray:→array;
  assign: interger, array, integer→array;
  read: array, integer→integer;
Error-ops
  no-element→integer;
Ok-eqn's
  read(assign(va1,arr,index1),index2)
    =if index1=index2 then va1 else read(arr,index2);
Error-eqn's
   read(newarray,index)=no-element;
jbo

用数组来实现无界栈 …



…数组来实现无界栈 …

STAK(array,integer) → stack

对于栈的每一个操作需要定义一实现程序，实现程序由数组级的操作组成，即通过**STAK**的具体表示**(arr,index)** 在数组级的运算实现

```
Push(arr,index,arg)
  begin
    arr:=assign(arg,arr,index+1);
    index:=index+1;
End;
```

不便证明

…数组来实现无界栈

- 用单一等式来表示
  - stk= STK(arr,index)
  - Push(stk,arg)=stk'
  - stk'=STK(arr',index')
  - 由push程序知
  - arr'=assign(arg,arr,index+1);
  - index':=index+1
  - Push的实现程序可以直接由以下等式来表示
  - Push(STAK(arr,index),arg)
  - =STAK(assign(arg,arr,index+1),index+1)

栈数据类型的实现

```
implementation stackBYarray;
 representation STAK(arry,integer)→stack;
 program
   newstack=STAK(newarray,0);
   push(STAK(arr,index),arg)
     =STAK(assgin(arg,arr,index+1),index+1);
   pop( (STAK(arr,index))
     =if index=0
       then STAK(arr,-1)
       else STAK(arr,index-1);
   top(STAK(arr,index))
     =if index=0 then no-more
                 else read(arr,index);
   empty(STAK(arr,index))=(index=0);
end
```

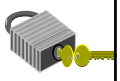
under-flow

同步练习

- 用有限数组实现有界栈。

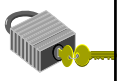
答案 (1)

- implementation stackBYarray;
- representation STAK(array, integer)→stack;
- program
- newstack=STAK(newarray,0);
- push(STAK(arr,index),arg)
- = if index=100 then STAK(arr,101)
- else
- STAK(assigh(arg,arr,index+1),index+1);



答案 (2)

- pop(STAK(arr,index))
- = if index=0 then STAK(arr,-1)
- else STAK(arr,index-1);
- top(STAK(arr,index))= read(arr,index);
- empty(STAK(arr,index))=(index=0);
- depth(STAK(arr,index))=index;
- end.



抽象数据类型实现的正确...

- 证明方法
  - 对于ok-eqn's和error-eqn's中所有的等式，将每个操作的实现代入后仍然能保证其成立。

...抽象数据类型实现的正确性...

- 证明：假设s为一正常的stack值（即非under-flow）

  1、pop(push(s,item))=s

pop(push(STAK(arr,index),item))= STAK(arr,index)

左边=**pop(STAK(assign(item,arr,index+1),index+1))**
=if index+1=0 then STAK(arr,-1)
else (STAK(assign(item,arr,index+1),index))
= STAK(assign(item,arr,index+1),index)

…抽象数据类型实现的正确性…

- 要证
- pop(push(STAK(arr,index),item))= STAK(arr,index)
- 即要证
- STAK(assign(item,arr,index+1),index)= STAK(arr,index)

引理 STAK(arr1,index1)= STAK(arr2,index2)
=(index1=index2) ∧
∀ (k:1<=k<=index1: read(arr1,k)=read(arr2,k))

即要证 (index=index) ∧ ∀（k:1<=k<=index:
read(assign(item,arr,index+1),k)=read(arr,k)) =true

read(assign(item,arr,index+1),k)=if index+1=k then item
else read(arr,k)
k<=index，所以index+1=k 显然为false

…抽象数据类型实现的正确性…

2、top(push(s,item))=item

左边= top(push(STAK(arr,index),item))
= top(STAK(assign(item,arr,index+1),index+1))
= if index+1=0 then no-more
else read((assign(item,arr,index+1),index+1))
=item=右边

…抽象数据类型实现的正确性…

3、empty(newstack)=true

左边= **empty(stack(newarray,0))**
=index=0=true=右边

4、empty(push(s,item))=false

左边= **empty(push(stack(arr,index),item))**
= empty(stack(assign(item,arr,index+1),index+1))
=false=右边

…抽象数据类型实现的正确性

5、pop(newstack)= STAK(arr,-1) ------underflow
左边=**pop(stack(newarray,0))**
=if index=0 then STAK(arr,-1)
else STAK(arr,index-1);
= STAK(arr,-1) =右边

6、top(newstack)=no-more
左边=**top(stack(newarray,0))**
=if index=0 then no-more
else read(arr,index);
= no-more =右边

代数规范的应用及其它

- 代数规范方法的应用
  - 作为设计和实现类型的一种辅助工具，通常要选择一特定可实现形式
  - 最重要的是证明实现的正确性。证明的方式纯粹是数学证明的形式化过程，容易自动地完成证明过程
  - 用于早期测试

内容线索

- ✓ **类型**
- ✓ **数据类型**
- ✓ **抽象数据类型**
- ✓ **数据抽象及其代数规范**
- 大型程序设计与抽象数据类型

C++中的抽象数据类型...

在下面的例子中，假定我们正在编写的应用程序需要大量的日期类型的数据。很明显，从数据的角度来看，一个日期类型的数据必须能访问和存储指定的年、月、日。从实现的角度来看，有很多种方法可以用来存储数据。从用户的角度来看，这个数据类型的实现应该与用户无关。例如，一个日期数据可以用三个整数来存储，每一个整数分别表示年、月、日。另外，我们也可以使用一个如 yyyymmdd 的长整型数来表示日期。假定数据 5/16/02 表示日期 5/16/2002，如果使用长整型数来存储日期数据类型，它应该被存储成长整型数 20020516。长整型数非常适合存储日期数据类型，因为用长整型存储日期的数字的顺序同日历的顺序是一致的。

但是，利用 C++内置的数据类型构造日期数据类型只解决了我们的一部分问题。我们还需要提供一系列的操作来使用这种日期类型的数据。显然，这些操作必须包括给日期类型数据赋值、进行两个日期类型数据的减运算以得到这两个日期之间包括的天数、比较两个日期数据以判断哪一天在前面、按照习惯的方式来显示日期，如显示成 12/03/01 而不是显示成 12/03/2001。

... C++中的抽象数据类型...

注意，每一种操作如何实现依赖于数据的存储方式（数据结构），并且只有开发这些操作的人对怎么实现数据的存储感兴趣。例如，采用一个长整型数来存储日期和使用三个整数分别存储年、月、日这两种存储结构中，进行两个日期的比较操作的实现是不同的。

日期类型数据的存储结构和一系列的可用于日期类型数据的操作一起定义了一种抽象的日期数据类型。一旦这种数据类型被设计出来，程序员就不需要考虑这些日期类型的数据是怎么存储的、对它的操作是怎么实现的。就像我们使用 C++内置的操作一样，只需要知道每一个操作是用来干什么的、怎么调用它。因此，我们并不真正关心两个整数相加是怎么实现的，只要知道这样做是对的。

在 C++中，抽象数据类型称为“类”。构造一个类相当简单，C++提供了所有必要工具：变量和函数。C++提供了一种机制将它们封装成一个自包含的单元。下面我们将看到这是怎么实现的。

… C++中的抽象数据类型…

```
///---类声明部分
class Date
{
   private://注意单词private后面的冒号
       int month; //数据成员
       int day; //数据成员
       int year; //数据成员
    public: //同样注意冒号
        Date(int=1,int=8,int=2007);
        //一个成员函数——构造函数
        void setdate(int, int, int);//成员函数
        void showdate();//成员函数
};
```


… C++中的抽象数据类型…

```
//---类实现部分
Date::Date(int mm, int dd, int yyyy)
{
   month=mm;
   day=dd;
   year=yyyy;
}
```

… C++中的抽象数据类型…

```
//---类实现部分
void Date::setdate(int mm, int dd, int yyyy)
{
   month=mm;
   day=dd;
   year=yyyy;
   return;
}
```


… C++中的抽象数据类型…

```
//---类实现部分
void Date::showdate( )
{
   cout<<"The date is";
   cout<<setfill('0')
           <<setw(2)<<month<<'/'
           <<setw(2)<<day<<'/'
           <<setw(2)<<year%100;
   //将年份的后两位展开;
   cout<<endl;
   return;
}
```

… C++中的抽象数据类型

```
int main()
{
  Date a,b,c(1,8,2005);//声明3个对象，初始化其中1个
  b.setdate(1,8,2006); //为数据成员赋值
  a.showdate();//显示对象a的值
  b.showdate();//显示对象b的值
  c.showdate();//显示对象c的值
  return 0;
}
```

作业3

- 1、写出有界集合的代数规范，然后予以实现。实现用有界数组。设有界集合的操作有emptyset, isempty, member, insert, delete, number(隐函数，表示元素的个数)假定界为5。
- 2、补充：写出下面抽象数据类型的代数规范并用数组实现
  - 栈，队列和集合