ALL--算法与数据结构教学小语言*)

张乃孝　蒋凌霄
(北京大学数学科学学院信息科学系　北京100871)

ALL--A Little Language for Teaching Algorithms and Data Structures

ZHANG Nai-Xiao　JIANG Ling-Xiao
(Department of Information Science, School of Mathematical Sciences, Peking University, Beijing 100871)

Abstract Based on the idea of domain-specific languages, a little language--ALL is proposed as a response to some problems in teaching "Algorithms and Data Structures". We expect that ALL will help decouple the learning of algorithms and data structures from the learning of programming languages. The definition and implementation of the main elements of ALL are introduced in this paper.

Keywords Domain specific language, Algorithms and data structures, Programming language

1 引言

一般来说，领域语言(Domain Specific Language，又称Little Language)是特定于某个领域的需求的描述语言，它提供足够多的语言成分支持该领域中通用的各种概念，使该领域的工作者可以简洁地用它构造领域内的应用程序。设计领域语言的主要目的就是为了简化应用程序开发过程、降低开发代价，同时又保证领域概念的清晰性，使开发过程可靠性高、可维护性强[1]。

《算法与数据结构》作为计算机科学中一门重要的理论和技术基础课程，在计算机教学中有着举足轻重的地位。该课程的主要目的是使学习者较全面地理解算法与数据结构的概念，掌握各种算法与数据结构的实现方式，比较不同算法与数据结构的特点。通过教学使学习者不仅学到算法与数据结构的基本知识，同时提高用计算机解决实际问题的能力。近二十多年来，算法与数据结构在概念、内容上基本取得了共识，但是用于描述算法和数据结构的语言却多种多样：有的用实用的程序设计语言[2]；也有的用接近自然语言的伪语言描述算法与数据结构[3]。前者的优点在于描述的内容可以通过实用语言的编译器检查、编译成可执行程序；缺点在于学习者需花很大精力学习所用的语言，掌握其中很多与算法和数据结构的知识没有本质联系的内容，影响学习者对算法和数据结构中本应该独立于实用语言的概念的理解。后者正好相反，优点在于伪语言易于理解，可以叙述清晰，突出算法和数据结构的本质概念和思想，便于学习者掌握；缺点在于语法语义不严格，一般没有适应的编译器，无法实际运行检验其执行效果。

本文借助领域语言的思想和方法，设计一种用于描述算法与数据结构的小语言来帮助课程的教学。我们希望所设计的描述语言支持抽象数据类型的观点，有效地实现数据的抽象和封装；有效地描述各种数据结构之间的关系，特别是从底层结构构造高层结构的组合关系；严格、清晰地描述各种算法的实现，便于分析和比较算法的时间和空间代价；易于保证和检查算法的正确性；有利于重用已经学到的知识，支持软件的重用思想。

2 设计目标的分析

根据上面的思想，在设计中主要围绕以下5个问题考虑。

2.1 概念的清晰性

在《算法与数据结构》的教学中，每引入一种新的数据结构首先借助抽象数据类型的观点讨论这种结构所具有的行为特征，然后考虑各种具体的表示方法，并在各种表示的基础上给抽象数据类型的实现写出各种行为的实现算法，最后通过实用的问题求解例子说明这种结构的使用方法。

根据这种模式，ALL语言程序整体上主要就由struct定义、函数定义和class定义组成。各种类型数据(不含操作)的组合用struct定义包装成一个新的数据类型，外界可以将该类型的数据对象作为一个整体使用，也可以访问其内部的各个数据成员。数据类型的行为用函数定义来体现，实用的问题求解例子(如：哈夫曼算法)也由函数定义来表示。但这两种函数定义在概念上有较大差别，前者必然与特定的数据类型存在联系，可以用class定义将其与特定的数据封装在一起构成符合抽象数据类型概念(数据＋行为)的新的数据类型，并通过访问权限控制外界对该类型数据对象内部的数据、行为的访问；而后者可以在不影响对算法和数据结构概念理解的基础上独立存在，这种函数在ALL语言中表现为独立在任何类型定义之外的函数定义。ALL的组成方式与Java将所有程序元素都封装在class里的做法相比，在语言层次上更便于将数据结构的各种概念区分开来，从而可以使学习者更清晰地理解不同概念。

2.2 简洁实用的描述方式

在足够描述基本算法与数据结构的基础上，确定ALL语言的主要语言成分。

第一，包括基本的过程式语言的成分。

第二，不含指针；采用Java的变量定义和函数参数传递语义：基本类型变量是值类型，以值拷贝方式传递；非基本类型的变量都是引用类型，以引用方式传递。

*)本课题得到国家自然科学基金资助(60273001)。张乃孝　教授，博士生导师，主要研究领域为软件形式化，算法与数据结构。蒋凌霄　硕士研究生。

第三，简单的输入输出语句作为 ALL 的语言成分，复杂的 I/O 操作不作为语言成分提供。

第四，从使用方便角度，允许任何数据类型、函数先使用后定义。

2.3 程序正确性的检查

从算法与数据结构的教学角度来说，用 ALL 描述的算法与数据结构的正确性是我们关注的问题。为此我们考虑了以下几点：

第一，限定 ALL 是强类型语言，变量定义时必须有明确的类型；运算作用的对象类型必须与运算要求的类型相符；函数实参与形参类型必须相符等等。而语法语义检查器负责完成所有的语法、静态语义的检查工作。

第二，不含 Java 中虚拟函数、动态绑定等与算法和数据结构概念无本质关系的复杂语言特征，减少其对程序正确性的影响，也有利于降低语法语义检查器的复杂性。

第三，提供适当的运行时错误检查、异常处理等涉及语言动态语义的能力。为 ALL 设计了以下两种语言成分，使 ALL 的使用者可以主动地检查运行时错误：

· assert 断言：用于在程序执行当中判断程序当前状态是否满足某些条件要求。

· 函数前后置条件 pre 和 post，用于描述函数执行前后应符合的约束条件。

2.4 适度的可重用性

算法与数据结构课程中常见的是数据类型之间的组合关系，很少使用继承关系，因而 ALL 也不含类型继承机制。

但是，考虑到顺序表和链表是实现其他数据结构的基础，栈和队列也常用于许多复杂算法，出于对代码重用的考虑，ALL 语言预定义了几种模板类型：SeqList（顺序表类型）、SingleList（单链表类型）、SeqQueue（顺序队列类型）、SeqStack（顺序栈类型）、LinkQueue（链式队列类型）、LinkStack（链式栈类型）等，供用户实际使用。

2.5 灵活的可变换性

由2.2节的简单介绍，将 ALL 语言程序变换为 Java 语言程序并不困难。根据“面向模型的变换型软件开发方法”[4]，我们用 Garment[5] 定义 ALL 语言的语法和变换语义。这样就可以借助专门的语言变换器，根据不同的目标语言，修改 ALL 语言成分到目标语言的变换规则，就可以在语法、静态语义检查正确的基础上将 ALL 程序变换为目标语言程序，变换的结果可以用在目标语言的程序设计当中，也可以立即借助目标语言的编译器产生可执行代码，从而使 ALL 语言程序具有间接的可执行性。

3 主要语言成分及其变换

从语言的发展和对学习者的吸引力角度考虑，我们基本上将 ALL 语言设计成 Java 的子集。

下面我们举例说明 ALL 的部分主要语言成分，并为这些成分给出它们到 Java、C 的变换结果。为下文叙述方便约定一些记法：

· $Trans_T(ele)$ 表示 ALL 的语言成分 ele 到目标语言 T 的变换结果；Trans(ele) 表示 ele 到 Java、C 中任何一种语言的变换结果；各个变换结果之间自然地有空格间隔。

· 一个语言成分 ele 在不同上下文中可以有不同的变换结果，以形如 $Trans_T(ele)_1$ 的数字下标来区分表示不同的变换结果。

· 带底纹的斜体字都表示变换结果。

· concat(a,b,…) 表示将 a,b,…连接到一起(a,b,…之间不再有空白符)构成的变换结果(这在目标语言的一个词法元素是由几个成分联合构成的时候特别有用)。

3.1 语句

·赋值语句例如：x=y;

Trans(x=y;)≡Trans(x)=Trans(y);

· 断言语句例如：assert x!=y “error";

变换为 Java：if($Trans_{Java}$(x!=y));else {

System.out.println (“assert error : ” + $Trans_{Java}$ (“error”));System.exit(1);}

变换为 C：if($Trans_C$(x!=y));else{printf(“assert error : %s\n”,$Trans_C$(“error”));exit(1);}

3.2 函数

· 在任何类型定义之外的函数定义例如：int f(String x) {…}

变换为 Java：class $Trans_{Java}$(该函数定义所在的文件名)

{ public static int $Trans_{Java}$(f)($Trans_{Java}$(String)$Trans_{Java}$ (x)){…} }

从该变换方式可看出：此类函数定义变换成 Java 之后都是同一个类的成员函数。

变换为 C：int $Trans_C(f)_1$($Trans_C(String)_2$ $Trans_C(x)_1$) {…}

· 对上述这类函数的调用例如：f(a)

变换为 Java：$Trans_{Java}$(该函数定义所在的文件名) . $Trans_{Java}$(f)($Trans_{Java}$(a))

变换为 C：$Trans_C(f)_1$($Trans_C(a)_1$)

· class 的成员函数定义例如：class Vector{…public int length(){…}…}

变换为 Java：… public int $Trans_{Java}$(length)(){…}…

变换为 C：int concat ($Trans_C$ (Vector)$_1$, “_”, $Trans_C$ (length)$_1$)($Trans_C$(Vector)$_2$ this_LC){…}

· 对成员函数的调用例如：v.length()

变换为 Java：$Trans_{Java}$(v). $Trans_{Java}$(length)()

变换为 C：concat ($Trans_C$ (Vector)$_1$, “_”, $Trans_C$ (length)$_1$)($TransC(v)_1$)

3.3 结构、类类型

· 结构定义例如：struct LinkNode{int info; LinkNode next;}

变换为 Java：class $Trans_{Java}$(LinkNode){

public int $Trans_{Java}$(info);

public $Trans_{Java}$(LinkNode)$Trans_{Java}$(next);}

变换为 C：struct $Trans_C$(LinkNode)$_1${

int $Trans_C$(info)$_1$; $Trans_C$(LinkNode)$_2$ $Trans_C$(next)$_1$;};

· 类定义例如：class String {private char [] value; private int length;

int find(String){…} … }

类成员的缺省访问权限控制符是 public

变换为 Java：class $Trans_{Java}$(String){

private char [] $Trans_{Java}$ (value); private int $Trans_{Java}$ (length);

public int $Trans_{Java}$(find)(){…} … }

变换为 C：struct $Trans_C$(String)$_1${char* $Trans_C$(value)$_1$; int $Trans_C$(length)$_1$;};

int concat ($Trans_C$ (String)$_1$, “_”, $Trans_C$ (find)$_2$) (

$Trans_C(String)_2$ this_LC){…} …

·结构、类类型的变量定义及初始化例如:String x=new String;

变换为 Java: $Trans_{Java}$ (String) $Trans_{Java}$ (x) = new $Trans_{Java}$(String)( );

变换为 C:$Trans_C(String)_2$ $Trans_C(x)_1$=NULL;

{ $Trans_C(String)_2$ concat($Trans_C(x)_1$,"_1")=NULL;

concat($Trans_C(x)_1$,"_1")=($Trans_C(String)_2$)malloc (sizeof(struct $Trans_C(String)_1$));

$Trans_C(x)_1$= concat($Trans_C(x)_1$,"_1");}

3.4 数组

数组的定义、使用的语法语义与 Java 完全相同。例如:int [][] var=new int [10][5];

变换为 Java:int [][] $Trans_{Java}$(var)= new int [10][5];

变换为 C:int** $Trans_C(var)_1$=NULL;

{int len_LC_1 = 10, len_LC_2 = 5; int itr_LC_1, itr_LC_2;

int** concat ($Trans_C$ (var)$_1$,"_1") = (int**) malloc (sizeof(int*)*len_LC_1);

for (itr_LC_1 = 0; itr_LC_1 < len_LC_1; itr_LC_1 ++)

concat($Trans_C(var)_1$,"_1")[itr_LC_1]=(int*)malloc (sizeof(int)* len_LC_2);

$Trans_C(var)_1$=concat($Trans_C(var)_1$,"_1");}

3.5 ALL 程序

完整的 ALL 程序包括:include 声明、struct、class 定义、函数定义和程序入口点 main 函数。ALL 程序中没有全局或者静态变量,main 函数必须形如:void main(){…},这些对语言的简化也基于消除和算法与数据结构无关的语言成分的考虑。

main 函数变换为 Java:class $Trans_{Java}$(该函数定义所在的文件名){public static void main(String [] args){…} },变换为 C:void main(){…}

结束语 依据领域语言的思想设计的小语言 ALL 在用于算法与数据结构教学时有望达到概念清晰、描述简便、易于变换、间接可执行等特点。我们现在已经实现了 ALL 的语法、静态语义检查及其到 Java 的变换系统,建立了使用 ALL 进行算法和数据结构程序实习的环境,并用 ALL 语言完成了基本数据结构和算法的描述,实际检验了 ALL 语言的效果和变换系统的功能,为辅助教学打下了良好基础。但是要将 ALL 语言真正用于教学实践还需要为实习环境增加一些功能,如:变换结果的格式化输出、程序调试等,更需要的是一本使用 ALL 语言描述算法和数据结构的教材,从而充分发挥 ALL 的特点,实现第一节提出的课程教学目标。

参 考 文 献

1 Van D A, Klint P. Little Language: Little Maintenance. Journal of Software Maintenance, 1998, 10: 75~92

2 张乃孝,裘宗燕. 数据结构——C++与面向对象的途径(修订版). 北京: 高等教育出版社, 2001

3 许卓群,张乃孝,杨冬青,唐世渭. 数据结构. 北京: 高等教育出版社, 1987

4 张乃孝,郑红军,裘宗燕. 语言的抽象、封装与变换型开发方法. 软件学报, 1998, 9(7): 496~500

5 和华. Garment 定义语言方法及实例研究:[北京大学硕士学位论文]. 2000

(上接第166页)

Terminate:终止控件设运行时的实例;

Initialize:初始化控件新的设计实例;

ReadProperties:读取控件使用人员对控件属性值的设置。

(4)关闭工程时发生如下的事件:

Terminate:终止控件设计时的实例。

由上可知,控件的运行实例和设计实例是两个完全不同的实例。从运行切换到设计时,并没有发生 WriteProperties 事件将运行时的属性保存起来。这显然是合理的,因为设计时的属性值的改变对运行仍然有效;反之则不然。

为了保存控件使用人员对控件属性值的设置,VB6.0提供了一个特殊的类对象 PropertyBag。该对象有两个重要的方法:ReadProperty 和 WriteProperty,分别用于属性的读取和赋值。向导生成代码的最后部分是与设置和读取属性值有关的,主要包括 ReadProperties 和 WriteProperties 两个事件过程,在这两个过程中包含了 PropertyBag 对象对属性的读写语句。同时需要注意的是,可以对向导生成的这些内容进行相应的调整和修改。

我们知道,容器对控件的使用是非常重要的。容器的 Extender 对象提供了许多的事件,此时控件的制作者不需要作任何工作即可获得 GotFocus、LostFocus、DragOver 和 DragDrop 四个事件。

可以为控件设置缺省事件。在控件设计时,调用 VB6.0 "工具"菜单的"过程属性"菜单,在对话框中选取相应的属性,并在"高级"按钮单击后选取"省缺用户界面"即可。此时,在使用控件的代码窗口中当选取此控件对象之后,会自动形成该控件对象的省缺事件的过程框架。

最后的工作是将 ActiveX 控件编译成扩展名为.ocx 的文件,此时会在本机上自动注册。当然,在其它的机器上使用时要重新注册。

结束语 ActiveX 能被支持 OLE 标准的任何程序语言或应用系统所使用,比 Plug-in 模式更灵活、更方便。ActiveX 控件实现了软件模块最大程度的复用。

另外,对于 Web 应用来说,与传统应用的最大不同是客户端的程序主要在常用的浏览器中运行,而 ActiveX 控件在页面中的使用实现了浏览器无法实现的功能,许多任务不必再求助服务器,从而大大减小了网络和服务器的负担。ActiveX 是以构件的形式出现的,可以灵活配置,而且还可以回调服务器上其它构件的方法,在很大程度上增强了应用的兼容性、开放性和安全性。

最后要说明的是,由于 ActiveX 控件是一种构件模块,因此为基于软件总线机制的大规模软件构架技术的实现奠定了基础。

参 考 文 献

1 潘爱民,等. COM 原理与应用[M]. 北京:清华大学出版社, 2000

2 王栋. Visual BASIC 程序设计实用教程[M]. 北京:清华大学出版社, 2002.3

3 张树兵,等. Visual BASIC6.0入门与提高[M]. 北京:清华大学出版社, 2001.1

4 谭淑英,李赫男,左贵启. 服务器端的动态网站开发技术[J]. 计算机应用研究, 2002, 5: 143~149

5 陈章渊等译. 智能 CORBA [M]. 北京:电子工业出版社, 1999

6 钱力棚,等. Visual InterDev6.0网络编程技术[M]. 北京:人民邮电出版社, 2000.1

7 王映辉,冯德明. 大规模软件构架技术[M]. 北京:科学出版社, 2003.6