

Chapter 7

Abstract Stacks and Lists

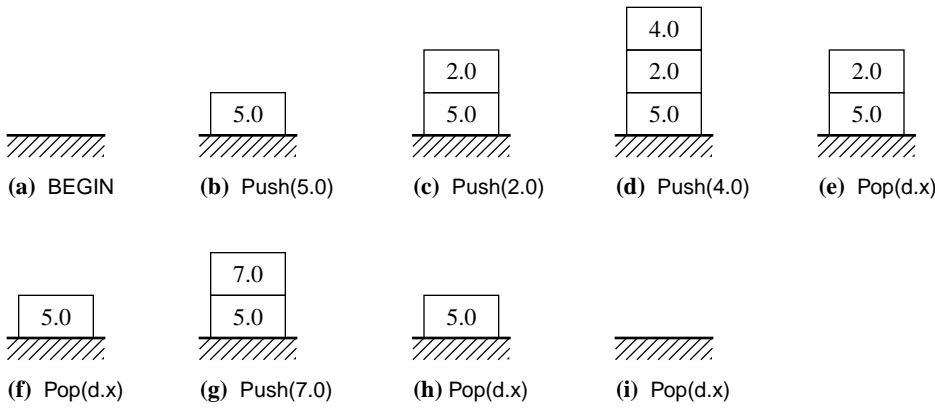


Figure 7.1
A sequence of operations on a stack.

There are three kinds of arithmetic notation:

- Infix $3 + 5$
- Prefix $+ 3 5$
- Postfix $3 5 +$

DEFINITION PboxStackADS;

CONST
 capacity = 8;

PROCEDURE Clear;
PROCEDURE NumItems (): INTEGER;
PROCEDURE Pop (OUT val: REAL);
PROCEDURE Push (val: REAL);

END PboxStackADS.

Figure 7.2

The interface of the stack abstract data structure.

CONST capacity

The maximum number of real values in the stack.

PROCEDURE Clear

Post

The stack is cleared to the empty stack.

PROCEDURE NumItems (): INTEGER

Post

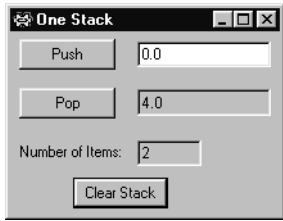
Returns the number of elements in the stack.

PROCEDURE **Pop** (OUT val: REAL)

Pre
 $0 < \text{NumItems}() \leq 20$
Post
An item is removed from the top of the stack and val gets its value.

PROCEDURE **Push** (val: REAL)

Pre
 $\text{NumItems}() < \text{capacity} \leq 20$
Post
val is pushed onto the top of the stack.

**Figure 7.3**

The dialog box that goes with the program in Figure 7.4.

```
MODULE Pbox07A;
  IMPORT Dialog, PboxStackADS;

  VAR
    d*: RECORD
      valuePushed*, valuePopped -: REAL;
      numItems -: INTEGER;
    END;

  PROCEDURE Push*;
  BEGIN
    PboxStackADS.Push(d.valuePushed);
    d.numItems := PboxStackADS.NumItems();
    Dialog.Update(d)
  END Push;

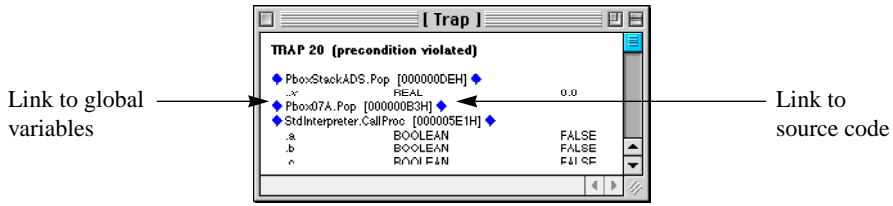
  PROCEDURE Pop*;
  BEGIN
    PboxStackADS.Pop(d.valuePopped);
    d.numItems := PboxStackADS.NumItems();
    Dialog.Update(d)
  END Pop;

  PROCEDURE ClearStack*;
  BEGIN
    PboxStackADS.Clear;
    d.valuePushed := 0.0; d.valuePopped := 0.0;
    d.numItems := 0;
    Dialog.Update(d)
  END ClearStack;

  BEGIN
    ClearStack
  END Pbox07A.
```

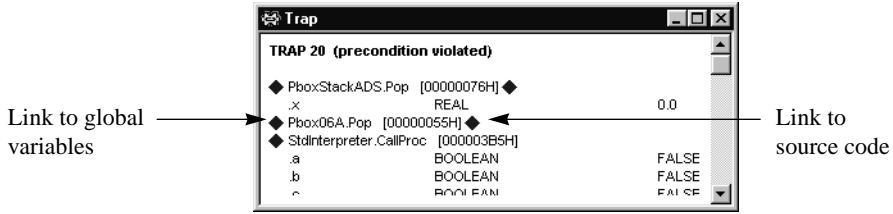
Figure 7.4

A program that uses the stack abstract data structure.

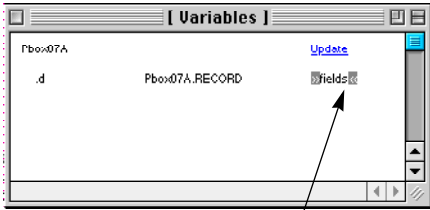


(a) MacOS

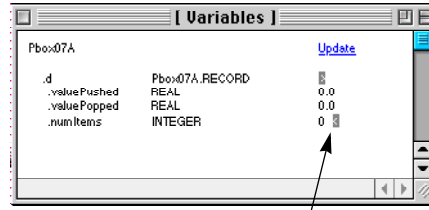
Figure 7.5
A trap window for the stack ADS program.



(b) MSWindows



Fold mark for collapsed fold

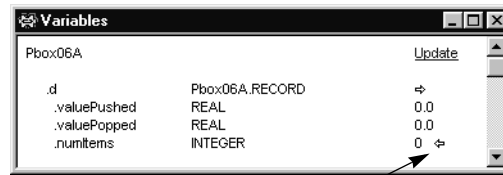


Fold mark for expanded fold

(a) MacOS



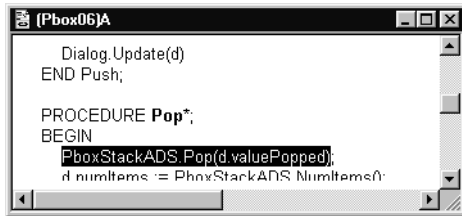
Fold mark for collapsed fold



Fold mark for expanded fold

(b) MSWindows

Figure 7.6
Global variables generated from the trap window of Figure 7.5.



```
Dialog.Update(d)
END Push;

PROCEDURE Pop*;
BEGIN
  PboxStackADS.Pop(d.valuePopped);
  d.numItems := PboxStackADS.NumItems();
```

Figure 7.7

Source code window opened by the link to the source code in Figure 7.5.

DEFINITION PboxStackADT;

CONST
 capacity = 8;

TYPE
 Stack = RECORD END;

PROCEDURE Clear (VAR s: Stack);
PROCEDURE NumItems (IN s: Stack): INTEGER;
PROCEDURE Pop (VAR s: Stack; OUT val: REAL);
PROCEDURE Push (VAR s: Stack; val: REAL);

END PboxStackADT.

Figure 7.8

The interface of the stack abstract data type.

TYPE Stack

The stack abstract data type supplied by PboxStackADT.

PROCEDURE Clear (VAR s: Stack)

Post

Stack s is cleared to the empty stack.

PROCEDURE Pop (VAR s: Stack; OUT val: REAL)

Pre

$0 < \text{NumItems}(s) \leq 20$

Post

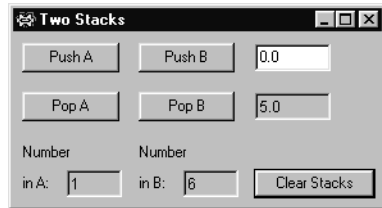
An item is removed from the top of stack s and val gets its value.

- Default Call by value
- IN Call by constant reference
- OUT Call by result
- VAR Call by reference

*The four calling modes of
Component Pascal*

- In call by value (default) and call by constant reference (IN), the procedure does *not* change the value of the actual parameter.
- In call by result (OUT) and call by reference (VAR), the procedure *does* change the value of the actual parameter.

*Changing versus not
changing the value of the
actual parameter.*

**Figure 7.9**

The dialog box for manipulating two stacks.


```
MODULE Pbox07B;
  IMPORT Dialog, PboxStackADT;

  VAR
    d*: RECORD
      valuePushed*, valuePopped-: REAL;
      numItemsA-, numItemsB-: INTEGER;
    END;
    stackA, stackB: PboxStackADT.Stack;

  PROCEDURE PushA*;
  BEGIN
    PboxStackADT.Push(stackA, d.valuePushed);
    d.numItemsA := PboxStackADT.NumItems(stackA);
    Dialog.Update(d)
  END PushA;

  PROCEDURE PushB*;
  BEGIN
    PboxStackADT.Push(stackB, d.valuePushed);
    d.numItemsB := PboxStackADT.NumItems(stackB);
    Dialog.Update(d)
  END PushB;
```

Figure 7.10

A program that uses the stack abstract data type.

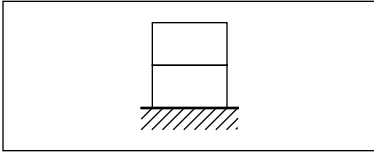
```
PROCEDURE PopA*;  
BEGIN  
    PboxStackADT.Pop(stackA, d.valuePopped);  
    d.numItemsA := PboxStackADT.NumItems(stackA);  
    Dialog.Update(d)  
END PopA;
```

```
PROCEDURE PopB*;  
BEGIN  
    PboxStackADT.Pop(stackB, d.valuePopped);  
    d.numItemsB := PboxStackADT.NumItems(stackB);  
    Dialog.Update(d)  
END PopB;
```

```
PROCEDURE ClearStacks*;  
BEGIN  
    PboxStackADT.Clear(stackA);  
    PboxStackADT.Clear(stackB);  
    d.valuePushed := 0.0; d.valuePopped := 0.0;  
    d.numItemsA := 0; d.numItemsB := 0;  
    Dialog.Update(d)  
END ClearStacks;
```

```
BEGIN  
    ClearStacks  
END Pbox07B.
```

PboxStackADS



PboxStackADT

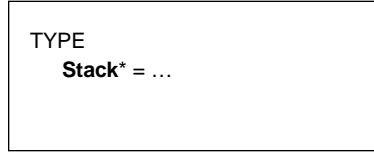


Figure 7.11

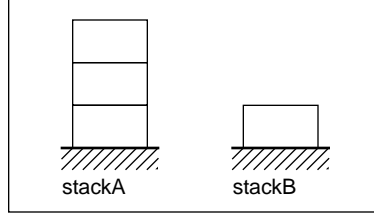
The difference between an abstract data structure and an abstract data type.

Pbox07A



(a) Abstract data structure

Pbox07B



(b) Abstract data type

0	trout
1	tuna
2	cod
3	salmon
4	
5	
6	
7	

Figure 7.12

A list that contains a maximum of eight strings.

DEFINITION PboxListADS;

```
CONST
  capacity = 8;
TYPE
  T = ARRAY 16 OF CHAR;

PROCEDURE Clear;
PROCEDURE Display;
PROCEDURE GetElementN (n: INTEGER; OUT val: T);
PROCEDURE InsertAtN (n: INTEGER; IN val: T);
PROCEDURE Length (): INTEGER;
PROCEDURE RemoveN (n: INTEGER);
PROCEDURE Search (IN srchVal: T; OUT n: INTEGER; OUT fnd: BOOLEAN);

END PboxListADS.
```

Figure 7.13

The interface of the list abstract data structure.

PROCEDURE Clear

Post

The list is initialized to the empty list.

PROCEDURE Length (): INTEGER

Post

Returns the number of elements in the list.

PROCEDURE InsertAtN (n: INTEGER; IN val: T)

Pre

 $0 \leq n \leq 20$ $\text{Length}() < \text{capacity} - 1$

Post

val is inserted at position n in the list, increasing Length() by 1.

If $n > \text{Length}()$, val is appended to the list.

0	trout
1	tuna
2	bass
3	cod
4	salmon
5	
6	
7	

(a) After inserting at position 2.

0	bass
1	trout
2	tuna
3	cod
4	salmon
5	
6	
7	

(b) After inserting at position 0.

0	trout
1	tuna
2	cod
3	salmon
4	bass
5	
6	
7	

(c) After inserting at position 7.

Figure 7.14

The list of Figure 7.12 after executing the statements of Example 7.9.

PROCEDURE **RemoveN** (n: INTEGER)

Pre

$0 \leq n < 20$

Post

If $n < \text{Length}()$, the element at position n in the list is removed.

Otherwise, the list is unchanged.

0	trout
1	tuna
2	salmon
3	
4	
5	
6	
7	

(a) After removing from position 2.

0	tuna
1	cod
2	salmon
3	
4	
5	
6	
7	

(b) After removing from position 0.

0	trout
1	tuna
2	cod
3	salmon
4	
5	
6	
7	

(c) After removing from position 7.

Figure 7.15

The list of Figure 7.12 after executing the statements of Example 7.10.

PROCEDURE **GetElementN** (n: INTEGER; OUT val: T)

Pre

$0 \leq n < 20$

$n < \text{Length}()$ 21

Post

val gets the data value of the element at position n of the list.

Note: 0 is the position of the first element in the list.

PROCEDURE **Search** (IN srchVal: T; OUT n: INTEGER; OUT fnd: BOOLEAN)

Post

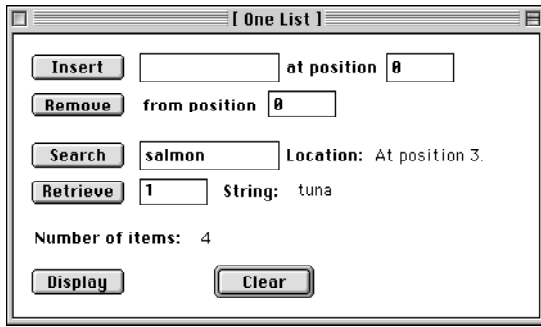
If srchVal is in the list, fnd is set to TRUE and n is set to the first position where srchVal is found.

Otherwise, fnd is set to FALSE and n is undefined.

PROCEDURE **Display**

Post

The list is output to the Log, one element per line with each element preceded by its position.

**Figure 7.16**

The dialog box for manipulating a list.

```
MODULE Pbox07C;
  IMPORT Dialog, PboxListADS, PboxStrings;

  TYPE
    String32 = ARRAY 32 OF CHAR;

  VAR
    d*: RECORD
      insertT*: PboxListADS.T; insertPosition*: INTEGER;
      removePosition*: INTEGER;
      searchT*: PboxListADS.T; searchPosition-: String32;
      retrievePosition*: INTEGER; retrieveT-: PboxListADS.T;
      numItems-: INTEGER;
    END;

  PROCEDURE InsertAt*;
  BEGIN
    PboxListADS.InsertAtN(d.insertPosition, d.insertT);
    d.numItems := PboxListADS.Length();
    Dialog.Update(d)
  END InsertAt;

  PROCEDURE RemoveFrom*;
  BEGIN
    PboxListADS.RemoveN(d.removePosition);
    d.numItems := PboxListADS.Length();
    Dialog.Update(d)
  END RemoveFrom;
```

Figure 7.17

A program that uses the list abstract data structure to implement the dialog box of Figure 7.16.

```
PROCEDURE SearchFor*;  
VAR  
  found: BOOLEAN;  
  position: INTEGER;  
BEGIN  
  PboxListADS.Search(d.searchT, position, found);  
  IF found THEN  
    PboxStrings.IntToString(position, 1, d.searchPosition);  
    d.searchPosition := "At position " + d.searchPosition + "."  
  ELSE  
    d.searchPosition := "Not in list."  
  END;  
  Dialog.Update(d)  
END SearchFor;
```

```
PROCEDURE RetrieveFrom*;  
BEGIN  
  PboxListADS.GetElementN(d.retrievePosition, d.retrieveT);  
  Dialog.Update(d)  
END RetrieveFrom;
```

```
PROCEDURE DisplayList*;  
BEGIN  
  PboxListADS.Display;  
  Dialog.Update(d)  
END DisplayList;
```

```
PROCEDURE ClearList*;  
BEGIN  
  PboxListADS.Clear;  
  d.insertT := ""; d.insertPosition := 0;  
  d.removePosition := 0;  
  d.searchT := ""; d.searchPosition := "";  
  d.retrievePosition := 0; d.retrieveT := "";  
  d.numItems := PboxListADS.Length();  
  Dialog.Update(d)  
END ClearList;
```

```
BEGIN  
  ClearList  
END Pbox07C.
```

DEFINITION PboxListADT;

CONST
 capacity = 8;

TYPE
 List = RECORD END;
 T = ARRAY 16 OF CHAR;

PROCEDURE Clear (VAR lst: List);
PROCEDURE Display (IN lst: List);
PROCEDURE GetElementN (IN lst: List; n: INTEGER; OUT val: T);
PROCEDURE InsertAtN (VAR lst: List; n: INTEGER; IN val: T);
PROCEDURE Length (IN lst: List): INTEGER;
PROCEDURE RemoveN (VAR lst: List; n: INTEGER);
PROCEDURE Search (VAR lst: List; IN srchVal: T; OUT n: INTEGER; OUT fnd: BOOLEAN);

END PboxListADT.

Figure 7.18

The interface of the list abstract data type.

PROCEDURE **InsertAtN** (n: INTEGER; IN val: T)

Pre

$0 \leq n \leq 20$

$\text{Length}() < \text{capacity} - 1$

Post

val is inserted at position n in the list, increasing $\text{Length}()$ by 1.

If $n > \text{Length}()$, val is appended to the list.

PROCEDURE **InsertAtN** (VAR lst: List; n: INTEGER; IN val: T);

Pre

$0 \leq n \leq 20$

$\text{Length}(\text{lst}) < \text{capacity} - 1$

Post

val is inserted at position n in list lst, increasing $\text{Length}(\text{lst})$ by 1.

If $n > \text{Length}(\text{lst})$, val is appended to lst.

The dialog box, titled "Two Lists", contains the following elements:

- Buttons: "Insert A", "Insert B", "Remove A", "Remove B", "Search A", "Search B", "Retrieve A", "Retrieve B", "Display A", "Display B", "Clear".
- Text input fields: "halibut" (next to "Insert B"), "2" (next to "at position"), "0" (next to "from position"), "0" (next to "String:"), "2" (next to "Number of items in A:"), "3" (next to "Number of items in B:").
- Labels: "at position", "from position", "Location:", "String:", "Number of items in A:", "Number of items in B:".

Figure 7.19

The dialog box for manipulating two lists.


```
MODULE Pbox07D;
  IMPORT Dialog, PboxListADT, PboxStrings;

  TYPE
    String32 = ARRAY 32 OF CHAR;

  VAR
    d*: RECORD
      insertT*: PboxListADT.T; insertPosition*: INTEGER;
      removePosition*: INTEGER;
      searchT*: PboxListADT.T; searchPosition-: String32;
      retrievePosition*: INTEGER; retrieveT-: PboxListADT.T;
      numItemsA-, numItemsB-: INTEGER;
    END;
    listA, listB: PboxListADT.List;

  PROCEDURE InsertAtA*;
  BEGIN
    PboxListADT.InsertAtN(listA, d.insertPosition, d.insertT);
    d.numItemsA := PboxListADT.Length(listA);
    Dialog.Update(d)
  END InsertAtA;

  PROCEDURE InsertAtB*;
  BEGIN
    PboxListADT.InsertAtN(listB, d.insertPosition, d.insertT);
    d.numItemsB := PboxListADT.Length(listB);
    Dialog.Update(d)
  END InsertAtB;
```

Figure 7.20

A program that uses the list abstract data type

```
PROCEDURE RemoveFromA*;  
BEGIN  
    PboxListADT.RemoveN(listA, d.removePosition);  
    d.numItemsA := PboxListADT.Length(listA);  
    Dialog.Update(d)  
END RemoveFromA;
```

```
PROCEDURE RemoveFromB*;  
BEGIN  
    PboxListADT.RemoveN(listB, d.removePosition);  
    d.numItemsB := PboxListADT.Length(listB);  
    Dialog.Update(d)  
END RemoveFromB;
```

```
PROCEDURE SearchForA*;  
VAR  
  found: BOOLEAN;  
  position: INTEGER;  
BEGIN  
  PboxListADT.Search(listA, d.searchT, position, found);  
  IF found THEN  
    PboxStrings.IntToString(position, 1, d.searchPosition);  
    d.searchPosition := "At position " + d.searchPosition + "."  
  ELSE  
    d.searchPosition := "Not in list."  
  END;  
  Dialog.Update(d)  
END SearchForA;
```

```
PROCEDURE SearchForB*;  
VAR  
  found: BOOLEAN;  
  position: INTEGER;  
BEGIN  
  PboxListADT.Search(listB, d.searchT, position, found);  
  IF found THEN  
    PboxStrings.IntToString(position, 1, d.searchPosition);  
    d.searchPosition := "At position " + d.searchPosition + "."  
  ELSE  
    d.searchPosition := "Not in list."  
  END;  
  Dialog.Update(d)  
END SearchForB;
```

```
PROCEDURE RetrieveFromA*;  
BEGIN  
    PboxListADT.GetElementN(listA, d.retrievePosition, d.retrieveT);  
    Dialog.Update(d)  
END RetrieveFromA;
```

```
PROCEDURE RetrieveFromB*;  
BEGIN  
    PboxListADT.GetElementN(listB, d.retrievePosition, d.retrieveT);  
    Dialog.Update(d)  
END RetrieveFromB;
```

```
PROCEDURE DisplayListA*;  
BEGIN  
    PboxListADT.Display(listA);  
END DisplayListA;
```

```
PROCEDURE DisplayListB*;  
BEGIN  
    PboxListADT.Display(listB);  
END DisplayListB;
```

```
PROCEDURE ClearLists*;  
BEGIN  
    PboxListADT.Clear(listA); PboxListADT.Clear(listB);  
    d.insertT := ""; d.insertPosition := 0;  
    d.removePosition := 0;  
    d.searchT := ""; d.searchPosition := "";  
    d.retrievePosition := 0; d.retrieveT := "";  
    d.numItemsA := 0; d.numItemsB := 0;  
    Dialog.Update(d)  
END ClearLists;
```

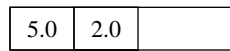
```
BEGIN  
    ClearLists  
END Pbox07D.
```



(a) BEGIN



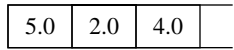
(b) Enqueue(5.0)



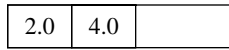
(c) Enqueue(2.0)

Figure 7.21

A sequence of operations on a queue.



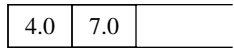
(d) Enqueue(4.0)



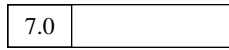
(e) Dequeue(d.x)



(f) Dequeue(d.x)



(g) Enqueue(7.0)



(h) Dequeue(d.x)



(i) Dequeue(d.x)

- IF in the client.
- ASSERT in the server.

The design-by-contract rule

- the precondition, P
- the statement, S
- the postcondition, Q

The Hoare triple

$\{P\}S\{Q\}$

PROCEDURE **RemoveN** (VAR *lst*: List; *n*: INTEGER)

Pre

$0 \leq n < 20$

Post

If $n < \text{Length}(lst)$, the element at position *n* in list *lst* is removed.

Otherwise, the list is unchanged.

$\{0 \leq n \wedge \text{Length}(lst) = \mathbf{L}\}$

lst := ?

$\{(n < \mathbf{L} \Rightarrow \text{Element at } n \text{ is removed} \wedge \text{Length}(lst) = \mathbf{L} - 1) \wedge (n \geq \mathbf{L} \Rightarrow lst \text{ is unchanged})\}$

PROCEDURE **Length** (IN *lst*: List): INTEGER

Post

Returns the number of elements in list *lst*.

$\{true\}S\{Length(lst) = \text{The length of } lst\}$

$\{R[x := E]\}x := E\{R\}$

*Formal definition of
assignment*