

Linked list specification

```
generic
type ELEMENT is private ;
package Linked is
  -- Exported type declarations
  type LIST is limited private ;
  type STATUS is range 1..10 ;
  type ITERATOR is private ;

  -- Comparison operations
  function Equals (L1, L2: LIST) return BOOLEAN ;
  function Equivalent (L1, L2: LIST) return BOOLEAN ;

  -- Access operations (Fig 20.6)
  -- Constructor operations (Fig. 20.7)
  -- I/O operations (Fig. 20.8)
  -- Iterator operations (Fig. 20.9)
private
  type LIST_ELEM;
  type LIST is access LIST_ELEM ;
  type ITERATOR is access LIST_ELEM ;
end Linked ;
```

Constructor operations

```
-- Append adds an element to the end of the list
procedure Append ( E: ELEMENT; Outlist: in out LIST ;
  Error_level: out STATUS ) ;
-- Add adds an element to the front of the list
procedure Add ( E: ELEMENT; Outlist: in out LIST ;
  Error_level: out STATUS ) ;
-- Add_before adds an element before element value E
procedure Add_before ( E: ELEMENT ; Outlist: in out LIST ;
  Error_level: out STATUS ) ;
-- Add_after adds an element after element E
procedure Add_after ( E: ELEMENT; Outlist: in out LIST ;
  Error_level: out STATUS ) ;
-- Replace replaces the element matching E1 with E2
procedure Replace ( E1, E2: ELEMENT; Outlist: in out LIST ;
  Error_level: out STATUS ) ;
-- Clear deletes all members of a list
procedure Clear ( Outlist: in out LIST ;
  Error_level: out STATUS ) ;
-- Prune removes the last element from the list
procedure Prune ( Outlist: in out LIST ;
  Error_level: out STATUS ) ;
```

Constructor operations

```
-- Prune_to deletes the list up to and including
-- the element matching E
procedure Prune_to ( E: ELEMENT; Outlist: in out LIST ;
                    Error_level: out STATUS ) ;
-- Prune_from deletes list after element matching E
procedure Prune_from( E: ELEMENT; Outlist: in out LIST ;
                      Error_level: out STATUS ) ;
-- Remove deletes the element which matches E
procedure Remove ( E: ELEMENT; Outlist: in out LIST ;
                   Error_level: out STATUS ) ;
-- Remove_before and Remove_after delete the element before
-- and after E respectively
procedure Remove_before ( E: ELEMENT; Outlist: in out LIST;
                          Error_level: out STATUS ) ;
procedure Remove_after ( E: ELEMENT; Outlist: in out LIST ;
                          Error_level: out STATUS ) ;
```

C++ linked list component

```
template <class elem> class List
{
public:
    List(); // Automatic constructor
    ~List(); // Automatic destructor
    // Basic list operations
    elem Head (error_indic &Err) ;
    int Length ( ) ;
    List <elem> Tail (error_indic &Err) ;
    // Equality operations
    friend List <elem> operator == (List <elem> L1, List <elem> L2) ;
    friend List <elem> Equivalent (List <elem> L1, List <elem> L2) ;
    // Constructor operations for linked list
    void Append (elem E, error_indic &Err) ;
    void Add (elem E, error_indic &Err) ;
    void Add_before (elem E, error_indic &Err) ;
    void Add_after (elem E, error_indic &Err) ;
    void Replace (elem E, error_indic &Err) ;
    void Clear (error_indic &Err) ;
    void Prune (error_indic &Err) ;
    void Prune_to (elem E, error_indic &Err) ;
    void Prune_from (elem E, error_indic &Err) ;
    void Remove (elem E, error_indic &Err) ;
    void Remove_before (elem E, error_indic &Err) ;
    void Remove_after (elem E, error_indic &Err) ;
```

C++ linked list component

```
// I/O functions
void Print(error_indic &Err) ;
void Write_list(char* filename, error_indic &Err) ;
void Read_list(char* filename, error_indic &Err) ;
private:
typedef struct LinkedList {
    elem val;
    LinkedList* next;
} LinkedList;

LinkedList* Listhead ; // (Internal) Pointer to start of list
};

template <class elem> class Iterator {
    friend class List <elem> ;
public:
    Iterator () ;
    ~Iterator () ;
    void Create (List <elem> L, error_indic &Err) ;
    void Go_next (error_indic &Err) ;
    elem Eval (error_indic &Err) ;
    boolean At_end () ;
private:
    LinkedList* iter ;
};
```

Portable counter component

```
Ada
package Counter is
    type T is limited private;
    procedure Inc (Cnt : in out T) ;
    procedure Dec (Cnt : in out T) ;
    procedure Copy (Cnt1: T ; Cnt2: out T) ;
    function Cequals (Cnt1, Cnt2: T) return BOOLEAN ;
private
    type T is range 0..500_000 ;
end Counter;

C++
class Counter {
public:
    Counter () ;
    void Inc () ;
    void Dec () ;
    friend Counter Copy ( Counter c1, Counter c2 ) ;
    // Overload the operator == to compare two counters
    friend Counter operator == ( Counter c1, Counter c2 ) ;
private:
    int value ;
};
```

Process management

```
package Process_manager is
  type PROCESS is private ;
  type STATUS is (READY, RUNNING, WAITING, KILLED)
  function Create return PROCESS ;
  function Kill (Process: PROCESS ) return STATUS ;
  function Get_status (P: PROCESS ) return STATUS ;
  function Wake_up (P: PROCESS ) return STATUS ;
  function Sleep (P: PROCESS ) return STATUS ;
  procedure Wait (P: PROCESS ;S: STATUS) ;
private type PROCESS is record
  PID: NATURAL ;
  State: STATUS ;
  end record ;
end Process_manager ;

class Process {
public:
  Process () ;
  P_state Kill () ;
  P_state Get_status () ;
  P_state Wake_up () ;
  P_state Sleep ();
  void Wait (P_state &status) ;
private:
  int PID ;
  P_state status ;
};
```