Problem Set 5 Solutions

Problem 1

Part 1:

Merge Sort is a *sort* algorithm that splits the items to be sorted into two groups, *recursively* sorts each group, and *merges* them into a final, sorted sequence.

Merge – Sort

Pre-Conditions: An array of 1 or more elements
Post-Condition : A sorted array

Pseudo-Code

1. Check if LB < UB
    a. Call merge sort with
        1. Input Array
        2. LB = LB
        3. UB = (UB + LB)/2

    b. Call merge sort with
        1. Input Array
        2. LB = (UB + LB)/2 + 1
        3. UB = UB

    c. Call merge with
        1. Merge with parameters
            - Array to be sorted
            - Lower bound
            - (Lower bound + Upper bound) / 2
            - (Lower bound + Upper bound) / 2 +1
            - Upper bound

The merge sort procedure will recursively call itself until it has only single element arrays. These are inherently sorted. It will then merge the successively larger elements until the whole sorted array is produced.

Merge

Pre-Conditions: An array of 1 or more elements, Legal values of Lower_Bound_1, Upper_Bound_1, Lower_Bound_2 and Upper_Bound_2,

Post-Condition: A merged array that is sorted

Pseudo-Code

1. Check if
    a. Lower_Bound_1 < Upper_Bound_1
    b. Lower_Bound_2 < Upper_Bound_2
    c. Lower_Bound_1 < Lower_Bound_2
    d. Upper_Bound_1 < Upper_Bound_2

2. If any of the above conditions are violated,
    a. Display Error
    b. Stop Executing the program

3. Set
    a. Index_1 := Lower_Bound_1;
    b. Index_2 := Lower_Bound_2;
    c. Index := Lower_Bound_1;
    d. Temp_Array := Input_Array

4. While (Index_1 <= Upper_Bound_1) and (Index_2 < =Upper_Bound_2)
    a. If (Input_Array(Index_1) < Input_Array (Index_2)
        i. Temp_Array(Index) := Input_Array(Index_1)
        ii. Index_1 := Index_1 + 1;
        iii. Index := Index + 1;
    b. Else
        i. Temp_Array(Index) := Input_Array(Index_2)
        ii. Index_2 := Index_2 + 1;
        iii. Index := Index + 1;

5. While (Index_1 <= Upper_Bound_1)
    a. Temp_Array(Index) := Input_Array(Index_1)
    b. Index_1 := Index_1 + 1;
    c. Index := Index + 1;

6. While (Index_2 <= Upper_Bound_2)
    a. Temp_Array(Index) := Input_Array(Index_2)
    b. Index_2 := Index_2 + 1;
    c. Index := Index + 1;

7. Input_Array := Temp_Array

Part 2:
My_Array Specification

```
------------------------------------------------------------------------
-- Specification of package to perform perform operations on My_array
-- which is of type integer and has a maximum size of 10
-- Specifier: Jayakanth Srinivasan
-- Date : March 09, 2003
------------------------------------------------------------------------


package My_Array is

   My_Array_Max : constant Integer := 10;
   My_Array_Min : constant Integer := 1;

   type My_Integer_Array is array (My_Array_Min .. My_Array_Max) of Integer;

   -- procedure to create an array of My_Array_Max numbers, of type My_Integer_Array
   procedure Create (
         Output_Array : in out My_Integer_Array;
         Size         : in     Natural          );

   -- procedure to display the contents of an array
   procedure Display (
         Output_Array : in     My_Integer_Array;
         Size         : in     Natural          );

   -- procedure to merge sort the array
   procedure Merge_Sort (
         Input_Array : in out My_Integer_Array;
         Lb          : in     Integer;
         Ub          : in     Integer          );

   -- procedure to merge
   procedure Merge (
         Input_Array : in out My_Integer_Array;
         Lb_1        : in     Integer;
         Ub_1        : in     Integer;
         Lb_2        : in     Integer;
         Ub_2        : in     Integer          );

end My_Array;
```

My_Array Implementation

```
-------------------------------------------------------
-- Implementation of My_Array.Ads
-- Programmer : Jayakanth Srinivasan
-- Created : Mar 01, 2003
-- Last Modified : Mar 09, 2003
-------------------------------------------------------

with Ada.Text_Io;
with Ada.Integer_Text_Io;
```

```
use Ada.Text_Io;
use Ada.Integer_Text_Io;
-- implementation of the package specified by My_Array.Ads
package body My_Array is

   -- procedure to create the array
   -- prompt the user for "size" numbers
   -- store it in Output_Array and return to the user
   procedure Create (
         Output_Array : in out My_Integer_Array;
         Size         : in     Natural           ) is
      Number : Integer;
   begin
      if (Size > My_Array_Max) then
         Put("Size is too Large");
         New_Line;
      else

         for I in 1..Size loop
            Put("Please enter the number : ");
            Get (Number);
            Skip_Line;
            Output_Array(I) := Number;
            New_Line;
         end loop;
      end if;
   end Create;

   -- procedure to display the contents of Output_Array on the screen
   -- Accepts the output array and displays it to the user on screen
   procedure Display (
         Output_Array : in     My_Integer_Array;
         Size         :        Natural           ) is

   begin
      for I in 1.. Size loop
         Put (Output_Array(I));
         New_Line;
      end loop;

   end Display;
   -- procedure to merge sort the array
   procedure Merge_Sort (
         Input_Array : in out My_Integer_Array;
         Lb          : in     Integer;
         Ub          : in     Integer           ) is
      Mid : Integer;
   begin

      if (Lb /= Ub) then
         Mid := (Ub + Lb)/2;
         -- merge sort the first half
         Merge_Sort(Input_Array, Lb, Mid);

         -- merge sort the other half
         Merge_Sort(Input_Array, Mid+1, Ub);
```

```
      -- merge the sorted pieces
      Merge(Input_Array, Lb, Mid,Mid+1, Ub);

   end if;


end Merge_Sort;

-- procedure to merge
procedure Merge (
      Input_Array : in out My_Integer_Array;
      Lb_1        : in     Integer;
      Ub_1        : in     Integer;
      Lb_2        : in     Integer;
      Ub_2        : in     Integer          ) is

   Temp_Array : My_Integer_Array; -- temporary array to store the merge
   Index      : Integer;          -- index of temporary array;
   Index_1    : Integer;          -- index of first piece to merge
   Index_2    : Integer;          -- index of the second piece to merge
begin
   Temp_Array := Input_Array;
   -- set the index values
   Index := Lb_1;
   Index_1 := Lb_1;
   Index_2 := Lb_2;
   -- copy user array into temporary array
   --Temp_Array := Input_Array;

   loop
      -- quit comparing when you finish processing one of the two pieces
      exit when (Index_1 > Ub_1) or (Index_2 > Ub_2);
      if (Input_Array(Index_1) < Input_Array(Index_2)) then
         Temp_Array(Index) := Input_Array(Index_1);
         -- move the index pointer in the first piece to the next element
         Index_1:=Index_1 +1;
         --  move the index pointed of temp_array to next element location
         Index := Index +1;
      else
         Temp_Array(Index) := Input_Array(Index_2);
         -- move the index pointer in the second piece to the next element
         Index_2:=Index_2 +1;
         --  move the index pointed of temp_array to next element location
         Index := Index +1;
      end if;
   end loop;

   loop
      exit when (Index_1 > Ub_1) ;

      Temp_Array(Index) := Input_Array(Index_1);
      -- move the index pointer in the first piece to the next element
      Index_1:=Index_1 +1;
      --  move the index pointed of temp_array to next element location
      Index := Index +1;
   end loop;
```

```
      -- copy any remaining elements in the second piece of the merge into temp
      loop
        exit when (Index_2 > Ub_2) ;

        Temp_Array(Index) := Input_Array(Index_2);
        -- move the index pointer in the second piece to the next element
        Index_2:=Index_2 +1;
        --  move the index pointed of temp_array to next element location
        Index := Index +1;
      end loop;
      Input_Array := Temp_Array;

   end Merge;

end My_Array;
```

Merge_Sort_Test

```
-------------------------------------------------------------------------------
-- An Ada procedure to perform merge_sort on an array using the my_array package
--      Prompt the user for the number of elements 'n' in the array.
--      Create an Array of n elements
--      Display the unsorted array to the user
--      Merge sort the array
--      Display the sorted array to the user
-- Programmer : Jayakanth Srinivasan
-- Date Last Modified : Mar 10, 2003
--
-------------------------------------------------------------------------------


with My_Array;
with Ada.Integer_Text_Io;
with Ada.Text_Io;

use My_Array;
use Ada.Integer_Text_Io;
use Ada.Text_Io;

procedure Merge_Sort_Test is
   Size_Of_Array : Natural;
   Merge_Array   : My_Integer_Array;

begin
   Put("Please Enter Size of Array >=0 :");
   Get(Size_Of_Array);
   -- create an array with size_of_array elements
   New_Line;

   Create(Merge_Array, Size_Of_Array);
   -- display the unsorted array
   Put_Line("Unsorted Array is : ");
   Display(Merge_Array, Size_Of_Array);
   New_Line;
   -- carry out merge sort
```

```
   Merge_Sort(Merge_Array,1, Size_Of_Array);
   New_Line;
   -- display the sorted array
   Put_Line("Sorted Array is : ");
   Display(Merge_Array,Size_Of_Array);

end Merge_Sort_Test;
```

Problem 2

Part 1.
Queues

Queues are a subclass of Linear Lists, which maintain the First-In-First-Out order of elements. Insertion of elements is carried out at the 'Tail' of the queue and deletion is carried out at the 'Head' of the queue.

A queue is an ordered (by position, not by value) collection of data (usually homogeneous), with the following operations defined on it:

Operation	**Description**
Initialize	Initialize internal structure; create an empty queue
Enqueue	Add new element to the tail of the queue
Dequeue	Remove an element from the head of the queue
Empty	True iff the queue has no elements
Full	True iff no elements can be inserted into the queue
Size	Returns number of elements in the queue
Display	Display the contents of the Queue

An array-based queue requires us to know two values *a priori*: the type of data contained in the queue, and the size of the array. For our implementation, we will assume that the queue stores integer numbers and can store 10 numbers.

The queue itself is a structure containing three elements: **Data**, the array of data, **Head**, an index that keeps track of the first element in the queue (location where data is removed from the queue), and **Tail**, an index that keeps track of the last element in the queue (location where elements are inserted into the queue).

Initialize

Preconditions : none
Post-Conditions: Queue, Head, Tail set to 1
Pseudo-Code :

```
Set Head to 1
Set Tail to 1
Return the Queue to the user.
```

Enqueue

Preconditions : Non-Full Queue, Element to insert
Post-Conditions: Queue with the element appended to it
Pseudo-Code :

```
If Queue is Full (Tail = Size of Queue + 1) Then
        Output "Overflow, Queue is full, cannot Enqueue."
Else
        Place Element in Queue(Tail)
        Increment Tail (Tail = Tail + 1)
        Return the queue to the user.
```

Dequeue

Preconditions : Non-Empty Queue
Post-Conditions: Queue with element at Head removed, element that is dequeued
Pseudo-Code :

```
If Queue is Empty (Head = Tail) Then
        Output "Underflow, Queue is empty, cannot dequeue."
Else
        Element := Queue(Head);
        Move all the elements from head+1 to Size of Queue one step to the left
        Return Element
```

Empty

Preconditions : Queue
Post-Conditions: Determines if the queue is empty
Pseudo-Code :

```
If Head = Tail Then
        Return Empty_Queue := True
```

```
        Else
                Return Empty_Queue:= False
```

Full

Preconditions : Queue
Post-Conditions: Return True if the Queue is full
Pseudo-Code :

```
        If Tail  = Queue_Size+1 Then
                Return True
        Else
                Return False
```

Size
Preconditions : Queue
Post-Conditions: Return the number of elements in the queue
Pseudo-Code :

```
                Return (Tail - Head)
```

Display
Preconditions : Queue
Post-Conditions: Display the contents of the queue
Pseudo-Code :

```
        If head < 1 then
                Lb :=1;
        Else
                Lb := Head;

        If tail > max_queue_ size + 1 then
                Ub := max_queue_size;
        Else
                Ub := Tail;

        For I:= Lb to Ub
                Display Queue(I)
```

Part 2:
My_Queue Specification

GNAT 3.13p (20000509) Copyright 1992-2000 Free Software Foundation, Inc.

Checking: c:/docume~2/jk/desktop/16070c~1/lab4~1/my_queue.ads (source file time stamp: 2003-03-19 12:24:44)

```
1. ----------------------------------------------------------------
2. -- Package Specification for My_Queue
3. --
4. -- Specifier : Jayakanth Srinivasan
5. -- Date Last Modified : March 09, 2003
6. ----------------------------------------------------------------
7.
8. package My_Queue is
9.    Queue_Size : constant Integer := 10;
10.    type My_Queue_Array is array (1 .. Queue_Size) of Integer;
11.
12.    -- declare a stack and top index as a record
13.    type My_Record_Queue is
14.       record
15.          Queue : My_Queue_Array;
16.          Head  : Integer;
17.          Tail  : Integer;
18.       end record;
19.
20.    -- create the queue by intializing head and tail to 1;
21.    procedure Create (
22.          Input_Queue : in out My_Record_Queue );
23.
24.    -- display the queue
25.    procedure Display (
26.          Input_Queue : in My_Record_Queue );
27.
28.
29.    -- insert an element into the queue
30.    procedure Enqueue (
31.          Input_Queue : in out My_Record_Queue;
32.          Element     : in     Integer         );
33.
34.    -- delete an element from the queue
35.    procedure Dequeue (
36.          Input_Queue : in out My_Record_Queue;
37.          Element     :    out Integer         );
38.
39.    -- size of the queue
40.    function Size (
41.          Input_Queue : My_Record_Queue )
42.      return Integer;
43.
44.    -- is empty queue
45.    function Isempty (
46.          Input_Queue : My_Record_Queue )
47.      return Boolean;
48.
49.    -- is queue full
50.    function Isfull (
51.          Input_Queue : My_Record_Queue )
52.      return Boolean;
53.
54. end My_Queue;
```

54 lines: No errors

My_Queue Implementation

GNAT 3.13p (20000509) Copyright 1992-2000 Free Software Foundation, Inc.

Compiling: c:/docume~2/jk/desktop/16070c~1/lab4~1/my_queue.adb (source file time stamp: 2003-03-19 12:33:46)

```
 1. ---------------------------------------------------------------
 2. -- Package body for My_Queue
 3. --
 4. -- Implementer: Jayakanth Srinivasan
 5. -- Date Last Modified : March 9, 2003
 6. ---------------------------------------------------------------
 7.
 8. with Ada.Text_Io;
 9. with Ada.Integer_Text_Io;
10. use Ada.Integer_Text_Io;
11. use Ada.Text_Io;
12.
13. package body My_Queue is
14.
15.    -- create the queue by intializing head and tail to 1;
16.    procedure Create (
17.          Input_Queue : in out My_Record_Queue ) is
18.    begin
19.       Input_Queue.Head:=1;
20.       Input_Queue.Tail:=1;
21.    end Create;
22.
23.    -- insert an element into the queue
24.    procedure Enqueue (
25.          Input_Queue : in out My_Record_Queue;
26.          Element     : in     Integer         ) is
27.    begin
28.       if (Isfull(Input_Queue)) then
29.          Put_Line("Cannot insert into queue");
30.       else
31.          Input_Queue.Queue(Input_Queue.Tail) := Element;
32.          Input_Queue.Tail:= Input_Queue.Tail +1;
33.       end if;
34.    end Enqueue;
35.
36.
37.    -- remove an element from the queue
38.    procedure Dequeue (
39.          Input_Queue : in out My_Record_Queue;
40.          Element     :    out Integer         ) is
41.    begin
42.       if (Isempty(Input_Queue)) then
43.          Put_Line("Cannot Delete from an empty queue");
44.       else
45.          Element := Input_Queue.Queue(Input_Queue.Head);
46.          -- move all elements one step to the left
```

```
47.         for I in 2.. Size(Input_Queue) loop
48.           Input_Queue.Queue(I-1) := Input_Queue.Queue(I);
49.         end loop;
50.         Input_Queue.Tail := Input_Queue.Tail -1;
51.       end if;
52.    end Dequeue;
53.
54.
55.    -- returns the queue size
56.    function Size (
57.         Input_Queue : My_Record_Queue )
58.     return Integer is
59.    begin
60.       return(Input_Queue.Tail -Input_Queue.Head);
61.    end Size;
62.
63.
64.    -- is empty queue
65.    function Isempty (
66.         Input_Queue : My_Record_Queue )
67.     return Boolean is
68.    begin
69.       if Input_Queue.Head = Input_Queue.Tail then
70.         return True;
71.       else
72.         return False;
73.       end if;
74.    end Isempty;
75.
76.    -- is the queue full
77.    function Isfull (
78.         Input_Queue : My_Record_Queue )
79.     return Boolean is
80.    begin
81.       if Input_Queue.Tail = Queue_Size +1 then
82.         return True;
83.       else
84.         return False;
85.       end if;
86.    end Isfull;
87.
88.    -- display the contents of the queue
89.    procedure Display (
90.         Input_Queue : in My_Record_Queue ) is
91.       Lb : Integer;
92.       Ub : Integer;
93.    begin
94.        -- check lower bound of the queue
95.       if Input_Queue.Head < 1 then
96.         Lb := 1;
97.       else
98.         Lb:= Input_Queue.Head;
99.       end if;
100.       -- check upper bound of the queue
101.       if Input_Queue.Tail > Queue_Size then
102.         Ub := Queue_Size;
```

```
103.       else
104.          Ub:= Input_Queue.Tail;
105.       end if;
106.
107.       for I in lb .. Ub loop
108.          Put (
109.             Input_Queue.Queue (I) );
110.          Put(" , ");
111.       end loop;
112.       New_Line;
113.    end Display;
114.
115. end My_Queue;
```

115 lines: No errors