Home Work 7 Solutions

1.

Part a. Write an algorithm to check if a user entered string is a palindrome.

Algorithm:

Get input string and length from the user.
Set Flag to True
For I in 1 .. Length/2 loop
    If String(I) /= String(Length – I+1) then
        Display ‘Input_String not a Palindrome”
        Set Flag to False
If Flag = True then
    Display “Input String is a Palindrome”

Part b. Implement your algorithm as an Ada95 program.

```
1. ------------------------------------------------
2. -- Program to check if a user input string is a
3. -- palindrome
4. -- Programmer: Joe B
5. -- Date Last Modified : March 16, 2004
6. ------------------------------------------------
7.
8. with Ada.Text_Io;
9.
10. procedure Check_Palindrome is
11.    -- defines a string with maximum length of 80
12.    User_Input : String (1..80);
13.    -- defines a variable to hold the length from get_line
14.    User_Length : Integer;
15.    -- flag to determine if the string was a palindrome
16.    -- remains true if the string is a palindrome
17.    Flag : Boolean := True;
18. begin
19.    Ada.Text_Io.Put("Please Enter a String: ");
20.    Ada.Text_Io.Get_Line(User_Input, User_Length);
21.
22.    for I in 1 .. User_Length/2 loop
23.       if User_Input(I) /= User_Input(User_Length - I+1) then
24.          Ada.Text_Io.Put_Line("Input is not a palindrome");
25.          Flag := False;
26.          exit;
27.       end if;
28.    end loop;
29.    if Flag = True then
30.       Ada.Text_Io.Put_Line("Input string is a palindrome");
31.    end if;
32. end Check_Palindrome;
```

32 lines: No errors

2. Modify the program above to read inputs from a text file and store the reversed string in an output text file. The program should:

Main Program Algorithm:

```
Open input_file
Create output_file
While not end_of_file (input_file)
        Read input_string and length from the file.
        If Is_Palindrome(input_string) then
                Store input_string in output_file
        Else
                Reverse input_string
                Store reversed_string in output_file
Close input_file
Close output_file
```

Reverse String Algorithm:

```
Get Input_String and Length
For I  in 1 .. Length/2
        Temp := Input_String(I)
        Input_String(I) := Input_String(Length – I + 1)
        Input_String(Length-I + 1) := Temp
```

Store String Algorithm:

```
Get Input_String and Length
For I in 1 .. Length
        Put Input_String(I) in Output_File
end loop
Put new_line in Output_File
```

Part b. Implement your algorithm as an Ada95 program.

Hw_7 Package Specification Listing

GNAT 3.15p (20020523) Copyright 1992-2002 Free Software Foundation, Inc.

```
 1. ------------------------------------------------
 2. -- package specification containing subprograms
 3. -- to reverse a string and check if the string is
 4. -- a palindrome
 5. -- Specifier : Joe B
 6. -- Date Last Modified : March 16, 2004
 7. ------------------------------------------------
 8. with Ada.Text_Io;
 9. package Hw_7 is
10.    -- checks if a string is a palindrome, returns true if it is
11.    function Check_Palindrome (
```

```
12.          User_Input  : String;
13.          User_Length : Integer )
14.       return Boolean;
15.    -- reverses the elements in the string from 1 .. length
16.    procedure Reverse_String (
17.          User_Input  : in out String;
18.          User_Length : in     Integer );
19.
20.    -- stores the characters in the string from 1 .. length
21.    -- into the output file and inserts a new_line
22.    procedure Store_String (
23.          User_Input   : in     String;
24.          User_Length  : in     Integer;
25.          Storage_File : in out Ada.Text_Io.File_Type );
26.
27. end Hw_7;
28.
```

28 lines: No errors

Hw_7 Package Body

GNAT 3.15p (20020523) Copyright 1992-2002 Free Software Foundation, Inc.

```
1. ------------------------------------------------------------
2. -- Package implementation of Hw_7
3. -- Programmer : Joe B
4. -- Date Last Modified : March 16, 2004
5. ------------------------------------------------------------
6. with Ada.Text_Io;
7. package body Hw_7 is
8.    function Check_Palindrome (
9.          User_Input  : String;
10.         User_Length : Integer )
11.      return Boolean is
12.      Flag : Boolean := True;
13.    begin
14.      for I in 1 .. User_Length/2 loop
15.         if User_Input(I) /= User_Input(User_Length - I+1) then
16.            Flag := False;
17.            exit;
18.         end if;
19.      end loop;
20.      return Flag;
21.    end Check_Palindrome;
22.
23.    procedure Reverse_String (
24.          User_Input  : in out String;
25.          User_Length : in     Integer ) is
26.       Temp : Character;
27.    begin
28.      for I in 1 .. User_Length/2 loop
29.         Temp := User_Input(User_Length - I + 1);
30.         User_Input(User_Length - I + 1) := User_Input(I);
31.         User_Input(I) := Temp;
32.      end loop;
33.    end Reverse_String;
34.
```

```
35.    procedure Store_String (
36.          User_Input   : in     String;
37.          User_Length  : in     Integer;
38.          Storage_File : in out Ada.Text_Io.File_Type ) is
39.    begin
40.       for I in 1 .. User_Length loop
41.          Ada.Text_Io.Put(Storage_File,User_Input(I));
42.       end loop;
43.       Ada.Text_Io.New_Line(Storage_File);
44.    end Store_String;
45.
46.
47. end Hw_7;
```

47 lines: No errors

Main Program Listing

GNAT 3.15p (20020523) Copyright 1992-2002 Free Software Foundation, Inc.

```
 1. ------------------------------------------------
 2. -- Program to read input string from a file
 3. --    if it is a palindrome
 4. --      store in output file
 5. --    else
 6. --      reverse string and store in output file
 7. -- Programmer: Joe B
 8. -- Date Last Modified : March 16, 2004
 9. ------------------------------------------------
10.
11. with Ada.Text_Io;
12. with Hw_7;
13.
14. procedure File_Based_Palindrome is
15.
16.    Input_File : Ada.Text_Io.File_Type;
17.    Input_File_Name : String := "my_program_input.txt";
18.
19.    Output_File : Ada.Text_Io.File_Type;
20.    Output_File_Name : String := "my_program_output.txt";
21.
22.    -- variable to store the string read from input_file
23.    Input_String : String (1..80);
24.    -- variable to store the length of the input_string
25.    Input_Length : Integer;
26.
27. begin
28.    -- open input file
29.    Ada.Text_Io.Open(Input_File, Ada.Text_Io.In_File, Input_File_Name);
30.    -- create the output file
31.    Ada.Text_Io.Create(Output_File, Ada.Text_Io.Out_File, Output_File_Name);
32.
33.    -- repeat while there are strings in the file
34.    while not Ada.Text_Io.End_Of_File(Input_File) loop
35.       -- read input from input_file
36.       Ada.Text_Io.Get_Line(Input_File, Input_String, Input_Length);
```

```
37.       if (Hw_7.Check_Palindrome(Input_String, Input_Length)) then
38.          -- store the string directly if it is a palindrome
39.          Hw_7.Store_String(Input_String, Input_Length, Output_File);
40.       else
41.          -- reverse the string because it is not a palindrome
42.          Hw_7.Reverse_String(Input_String, Input_Length);
43.          -- store the string in the output file
44.          Hw_7.Store_String(Input_String, Input_Length, Output_File);
45.       end if;
46.    end loop;
47.    -- close both files
48.    Ada.Text_Io.Close(Input_File);
49.    Ada.Text_Io.Close(Output_File);
50.
51. end File_Based_Palindrome;
52.
53.
```

53 lines: No errors

3.
a. Compare and contrast stacks and queues.
Stacks
Stacks and Queues are subclasses of Linear Lists.
Stacks
A Stack is an ordered (by position, not by value) collection of data (usually homogeneous), which maintains a Last-In-First-Out order. All access to a stack is restricted to one end of the list, called the *top of stack*. Visually, picture a stack of books, coins, plates, etc. Insertion and Deletion both take place at the top of the stack. The following operations are defined for stacks:

Operation	**Description**
Initialize	Initialize internal structure; create empty stack
Push	Add new element to top of stack
Pop	Remove top element from stack
Empty	True iff stack has no elements
StackTop	Returns copy of top element of stack (without popping it)
Size	Returns number of elements in the stack

Queues

A queue is an ordered (by position, not by value) collection of data (usually homogeneous), which maintains 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. The following operations are defined for queues:

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

b. Modify the expression conversion algorithm shown in class to include unary operators.

To resolve the ambiguity between unary and binary operators which use the same symbol, two new symbols are created for use within the program:

- '+' is represented using '!'
- '-' is represented using '~'

The Is_Operator function has to be modified to include these two symbols:

```
If character = '(' | ')' | '+' | '-' | '*' | '/' | '!' | '~'
```

The key thing to remember is that unary '+' and '-' have the highest precedence. Hence the Precedence function has to be modified to account for it as follows:

```
Switch operator
        When '!' | '~' => return 4
        When '*' | '/' => return 3
        When '+' | '-' => return 2
        When '(' | ')' => return 1
```

Algorithm to Identify if operator is Unary:

```
Get the infix_string and current_position of operator
Location := current_position-1
If the unary operator is the first non-blank character in the string
        Return true
While Location >= 1 and Infix_String(Location) /= '('
        if Infix_String(Location) = ' ' then
                Location := Location -1
        elsIf Infix_String(Location) is an operator
                Return true
        else
                Return false
Return true
```

Algorithm to convert '+' or '-' into unary operators:

```
If Infix_String(I) = '+' or Infix_String(I) = '-' then
        Check if Infix_String(I) is unary.
        If it is unary
                Replace '+' with '!'
                Replace '-' with '~'
```

A similar replace operation takes place when the postfix string is being generated.

If ‘!’ is encountered, replace it with ‘+’
If ‘~’ is encountered, replace it with ‘-’

c)

Code Listing

my_postfix_converter specification partial file listing

```
63.   -- function to check if the operator is unary
64.   -- this is a new function
65.   function Is_Unary(Infix: String; Position: Integer) return Boolean;
66.
67.   -- function to convert infix to postfix
68.   -- this function is modified
69.   procedure Infix_To_Postfix(Infix : in out String; Length: in Integer; postfix : out string);
70.
```

Note that the function became a procedure because the string was being manipulated. An alternative solution is to keep the original function and create a local string variable that is used throughout the Infix_To_Postfix conversion function

my_postfix_converter package body partial file listing

GNAT 3.15p (20020523) Copyright 1992-2002 Free Software Foundation, Inc.

```
1. ------------------------------------------------------------------
2. -- Package body for My_Postfix_Converter
3. --
4. -- Implementer: Joe B
5. -- Date Last Modified : March 15, 2004
6. ------------------------------------------------------------------
7.
8. with Ada.Text_Io;
9.
10. use Ada.Text_Io;
11.
12 - 95. regular stack operations hence not listed
96.
97.    -- checks if input is an operator
98.    function Isoperator (
99.          Operator : Character )
100.      return Boolean is
101.    begin
102.       case Operator is
103.          when '~'|'!'|'*'| '/'|'+'|'-'|'('|')' =>
104.             return True;
105.          when others =>
106.             return False;
107.       end case;
108.    end Isoperator;
109.
110.    – returns precedence of the operator
```

```
111.    function Precedence (
112.        Operator : Character )
113.      return Integer is
114.    begin
115.      case Operator is
116.        when '~' | '!' =>
117.          return 4;
118.        when '*'| '/'=>
119.          return 3;
120.        when '+'|'-' =>
121.          return 2;
122.        when '('| ')' =>
123.          return 1;
124.        when others =>
125.          return -1;
126.      end case;
127.    end Precedence;
128.
129 - 150.    -- function to check balance – Not changed hence not listed
151.    -- function to check if the + or - is unary
152.    function Is_Unary (
153.        Infix    : String;
154.        Position : Integer  )
155.      return Boolean is
156.      Prev_Index : Integer;
157.    begin
158.      -- if it is the first element in the infix string, it is unary
159.      if Position = 1 then
160.        return True;
161.      end if;
162.      -- check backwards to see if it is a unary operator
163.      Prev_Index:= Position-1;
164.      while Prev_Index >=1 and Infix(Prev_Index) /= '(' loop
165.        -- if it is a blank space, ignore and check backwards
166.        if Infix(Prev_Index) = ' ' then
167.          Prev_Index := Prev_Index -1;
168.          -- if the previous element is an operator, then the + or -
169.          -- has to be a unary operator hence return true
170.        elsif Isoperator(Infix(Prev_Index)) then
171.          return True;
172.          -- if the element is an operand, then it has to be a binary operator
173.          -- hence return false
174.        else
175.          return False;
176.        end if;
177.      end loop;
178.      -- either a '(' is encountered or there are no previous elements
179.      -- therefore return true.
180.      return True;
181.
182.
183.    end Is_Unary;
184.
185.
186.    -- function to convert infix to postfix
187.    procedure Infix_To_Postfix (
188.        Infix   : in out String;
189.        Length  : in     Integer;
```

```
190.        Postfix :    out String   ) is
191.
192.     Postfix_Index  : Integer;
193.     Operator_Stack : My_Stack;
194.     Done           : Boolean;
195.     Element        : Character;
196.
197.   begin
198.     Create(Operator_Stack);
199.     Postfix_Index := 1;
200.     for I in 1.. Length loop
201.       -- if it is an operand
202.       if Isoperator(Infix(I)) = False then
203.         -- ignore blank spaces
204.         if (Infix(I) /= ' ') then
205.           Postfix(Postfix_Index):= Infix(I);
206.           Postfix_Index := Postfix_Index + 1;
207.         end if;
208.       else
209.         -- it is an operator
210.         Done:= False;
211.         loop
212.           exit when Done = True;
213.           -- if there is possible ambiguity, check if it is unary
214.           -- and change the symbol in the expression
215.           if Infix(i) = '+' or infix(I) = '-' then
216.            if Is_Unary(Infix, I) then
217.              if Infix(I) = '+' then
218.                Infix(I) := '!';
219.              else
220.                Infix(I) := '~';
221.              end if;
222.              end if;
223.           end if;
224.
225.           -- if its a ( or an empty stack then push operator onto stack
226.           if Isempty(Operator_Stack) or Infix(I) = '(' then
227.             Push(Operator_Stack, Infix(I));
228.             Done := True;
229.             -- check the precedence of the element on top of the stack and infix(I)
230.           elsif Precedence(Infix(I)) > Precedence(Stacktop(Operator_Stack)) then
231.             Push(Operator_Stack, Infix(I));
232.             Done:= True;
233.           else
234.             -- pop until an operator of smaller precedence appears on the stack
235.             Pop(Operator_Stack, Element);
236.             if Element = '(' then
237.               Done := True;
238.             else
239.               -- convert special symbols back to standard representation
240.               if Element = '~' then
241.                 Element:= '-';
242.               end if;
243.               if Element = '!' then
244.                 Element := '+';
245.               end if;
246.               Postfix(Postfix_Index):= Element;
247.               Postfix_Index := Postfix_Index + 1;
```

```
248.                 end if;
249.               end if;
250.             end loop;
251.           end if;
252.       end loop;
253.       -- pop the remaining elements into postfix
254.       while (Isempty(Operator_Stack) = False) loop
255.         Pop(Operator_Stack, Element);
256.         -- convert special symbols back to standard representation
257.         if Element = '~' then
258.           Element:= '-';
259.         end if;
260.         if Element = '!' then
261.           Element := '+';
262.         end if;
263.
264.         Postfix(Postfix_Index):= Element;
265.         Postfix_Index := Postfix_Index + 1;
266.       end loop;
267.       -- set the remaining spaces to blanks
268.       for I in Postfix_Index .. Stack_Size loop
269.         Postfix(I) := ' ';
270.       end loop;
271.    end Infix_To_Postfix;
272.
273.
274. end My_Postfix_Converter;
275.
```

275 lines: No errors