CS145 Midterm – February 9, 2005

For questions 1 to 12, unless otherwise indicated, please select the letter of the choice that best completes the statement or answers the question. Each question counts 4 points.

__D__ 1. Which of the following is a not a reason to use an ADT?
   A. Create new types similar to built-in types
   B. Expose the interface while hiding the implementation of the type
   C. Changing the implementation of the type without changing the use
   D. Allow users of the type to manipulate the internals of instances of the type
   E. Support reuse

__D__ 2. Which of the following Ada language features is not used to support ADTs?
   A. PRIVATE types
   B. Packages
   C. User defined exceptions
   D. Recursion

_____ 3. (Write your answer below.) Briefly, what is the difference between a PRIVATE and a LIMITED PRIVATE type?

   LIMITED PRIVATE types don’t have equality “=” or assignment “:=”.

__D__ 4. Which of the following cannot be a generic:
   A. PROCEDURE
   B. FUNCTION
   C. PACKAGE
   D. TYPE

__C__ 5. Recursion is another way to implement
   A. Sequential execution
   B. Conditional execution
   C. Iteration
   D. None of the above

For questions 6 and 7, consider the following program:

```ada
with Ada.Integer_Text_IO;
procedure DoIt is

   function Recursive( N : Natural ) return Integer is
   begin
      if N = 0 then
         return 0;
      else
         return 10 + Recursive( N - 1 );
      end if;
   end Recursive;

begin
   Ada.Integer_Text_IO.Put( Recursive( 3 ) );
End DoIt;
```
6. (Fill in the value.) What value is printed by the program?

7. (Fill in the value.) When this program is executing the stack will consist of a stack frame for DoIt and some number of stack frames for the function Recursive. What is the maximum number of frames for the function Recursive that will appear on the stack when the DoIt, as shown, above is executed?

8. Which of the following is not used for dynamic memory management?
   A. ACCESS types
   B. NEW expression
   C. PRIVATE types
   D. .ALL
   E. NULL

9. What will be printed by the following program:

   ```ada
   with Ada.Text_IO;
   procedure DoIt is
     X : Integer;
   begin
     if X = 0 then
       Ada.Text_IO.Put("True");
     else
       Ada.Text_IO.Put("False");
     end if;
   end DoIt;
   ```

   A. “True”
   B. “False”
   C. Neither “True” or “False”
   D. Either “True” or “False”

10. What will be printed by the following program:

    ```ada
    with Ada.Text_IO;
    procedure DoIt is
      type Ptr is access Integer;
      X : Ptr;
    begin
      if X = null then
        Ada.Text_IO.Put("True");
      else
        Ada.Text_IO.Put("False");
      end if;
    end DoIt;
    ```

    A. “True”
    B. “False”
    C. Neither “True” or “False”
    D. Either “True” or “False”
For questions 11 and 12, consider following program:

```pascal
procedure P is
  type IP is access Integer;
  type IPA is access all Integer;
  type IPC is access constant Integer;

  P : IP;
  PA : IPA;
  PC : IPC;

  X : Integer;
  Y : aliased Integer;
  Z : aliased constant Integer := 5;

begin
  P := Y'access; -- (1)
  PA := X'access; -- (2)
  PA := Y'access; -- (3)
  PA := Z'access; -- (4)
  PC := Y'access; -- (5)
  PC := Z'access; -- (6)
end P;
```

11. Which of assignments (1), (2), and (3) are illegal:
   A. (1)
   B. (2)
   C. (3)
   D. (1) and (2)
   E. (2) and (3)

12. Which of assignments (4), (5), and (6) are illegal:
   A. (4)
   B. (5)
   C. (6)
   D. (4) and (5)
   E. (5) and (6)
For questions 13 to 17 please answer as indicated by the question. Point values are as indicated for each question.

For question 13 to 16 consider the following proposed GENERIC specification for a sort program.

generic
    type Element_Type is private;
    type Index_Type is (<>);
    type Array_Type is array( Index_Type range <> ) of Element_Type;
procedure Sort( in out Data : Array_Type );

And the following example instantiation for the generic:

type Integer_Array is array( Integer range <> ) of Integer;
procedure Integer_Sort is new Sort(
    Element_Type => Integer;
    Index_Type => Integer;
    Array_Type => Integer_Array;
    "<" => "<");

__B__ 13. (5 points, multiple choice) Element_Type can be any type.
   A. True
   B. False

____ 14. (5 points, write your answer below.) Briefly describe the kinds of types that can be used for Index_Type.

    Any Discrete type. (An Integer or a subtype of Integer or an enumeration or subtype of an enumeration type.)

__B__ 15. (5 points, multiple choice) The type of Array_Type can be
   A. A Constrained Array
   B. An Unconstrained Array
   C. Both A and B

____ 16. (10 points, write your answer below.) The specification shown above cannot work with the proposed instantiation without an additional generic parameter. Write the additional generic parameter needed to make the above work:

    with function "<"( L, R : Element_Type ) return Boolean;
17. (25 points, write your answer below.) Given the Linked List type

type List_Node;
type List is access List_Node;
type List_Node is
  record
    Next : List;
    Data : Integer;
  end record;

Write a **recursive** function with the following specification:

```plaintext
function Found_In_List( D : Integer; L : List ) return Boolean;
-- Preconditions: D is a valid Integer
-- L is a valid (well formed) Linked List
-- Returns: True when D is in the List, that is when the Integer D is
--         equal to the Data component in some List_Node
--         False otherwise (D is not in the List, that is D is not
--         equal to the Data component in any List_Node)

function Found_In_List( D : Integer; L : List ) return Boolean is
begin
  -- First test to see if we are at the end of the list.
  -- Note: This needs to be done first, since the remainder of
  -- the program will assume that the L is not null.
  if L = null then
    -- The list is empty. The value was not found.
    return False;
  elsif D = L.Data then
    -- The value was found in the list.
    return True;
  else
    -- Return the result of searching the rest of the list.
    return Found_In_List( D, L.Next);
  end if;
end Found_In_List;
```