16.unified
Introduction to Computer Programming

Examination
11/18/05

Professor I. Kristina Lundqvist,
Lecturers Heidi Perry and Gustaf Naeser
Fall 2005

You have 55 minutes to take this examination. Do not begin until you are instructed to do so. This is a closed book examination. No external materials are permitted, including calculators or other electronic devices. All answers must be written in the examination paper. This examination should consist of 11 pages (including this cover sheet). Count the number of pages and immediately report any discrepancy. Should you need to do so, you may continue your answer on the back pages. Put your name at the bottom of each page of this exam.

<table>
<thead>
<tr>
<th>Question 1 True/False (10)</th>
<th>Question 2 Read Ada Code (20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 3: Recursion (10)</td>
<td>Question 4: Arrays (15)</td>
</tr>
<tr>
<td>Question 5: Number Conversions (15)</td>
<td>Question 6: Computer Architecture (10)</td>
</tr>
<tr>
<td>Question 7: Rover (20)</td>
<td></td>
</tr>
</tbody>
</table>

Page 1 of 11
Name________________________________
### Problem 1

**True or False (10p)**

(1p each)

For each statement, indicate if it is true (T) or false (F).

<table>
<thead>
<tr>
<th>Statement</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>The strong typing of Ada95 prevents users from confusing variables of different types.</td>
<td>T</td>
</tr>
<tr>
<td>Iteration and recursion solves the same kind of problem: repeated execution of statements.</td>
<td>T</td>
</tr>
<tr>
<td>A string in Ada95 can be treated as records of characters.</td>
<td>F</td>
</tr>
<tr>
<td>Variable declarations implicitly set default values of the declared variable.</td>
<td>F</td>
</tr>
<tr>
<td>Variables used as actual parameters must have the same name as the name of the subprogram’s formal parameters.</td>
<td>F</td>
</tr>
<tr>
<td>All kinds of loops can be described using for loops.</td>
<td>F</td>
</tr>
<tr>
<td>A function that returns two values must be implemented as a procedure.</td>
<td>F</td>
</tr>
<tr>
<td>Distinct datatypes, e.g. type My_Type is new Integer, are for creating types that should not be confused with other types using the same symbols (possible values).</td>
<td>T</td>
</tr>
<tr>
<td>Subtypes are assignment compatible with the base type.</td>
<td>T</td>
</tr>
<tr>
<td>You can store both positive and negative numbers using 2-complement</td>
<td>T</td>
</tr>
</tbody>
</table>

**Comments**

1. Among many things
2. Recursion loops code according to the programmer’s design
3. A string is an array of characters
4. Variables must be given values (the declaration just allocates the memory)
5. No, the formal parameters are separate from the actual ones
6. The general loop statement is the “loop”
7. Functions can only return a single value whereas procedures can have several out parameters.
8. True
9. Subtypes are (Subtype_Var := Basetype_Var_or_Value is allowed)
10. True
Problem 2  

**Reading Ada Code (20p)**

Our programmer friend Joe D has written the program `main.adb`. Predict what the output of Joe’s program will be at commented lines A through F.

```ada
with Ada.Integer_Text_Io;  
use Ada.Integer_Text_Io;  

procedure Main is  
  A : Integer := 2;  
  B : Integer := 5;  
  C : Integer := 4;  

  function F1 (X: Integer; Y: Integer) return Integer is  
    C : Integer;  
  begin  
    C := X + Y;  
    return C;  
  end;  

  procedure F2 (X: in out Integer; Y: in Integer) is  
    A := X;  
    X := A * Y;  
  begin  
    B := A * B;  
    A := F1(B, C);  
    B := B + C;  
    A := A - C;  
    C := F1(A, B);  
    for C in 1..A loop  
      B := B + 1;  
    end loop;  
  begin  
    B := A * B;  
    A := F1(B, C);  
    B := B + C;  
  end;  

begin  
  B := A * B;  
  A := F1(B, C);  
  for C in 1..A loop  
    B := B + 1;  
  end loop;  
  Put(A);  
  B := B + C;  
  A := A - C;  
  C := F1(A, B);  
  Put(C);  
  for C in 1..A loop  
    B := B + 1;  
  end loop;  
  Put(A);  
  B := B + C;  
  A := A - C;  
  C := F1(A, B);  
  Put(C);  
  for C in 1..A loop  
    B := B + 1;  
  end loop;  
  Put(A);  
  B := B + C;  
end Main;
```

<table>
<thead>
<tr>
<th>Part A</th>
<th>Part B</th>
<th>Part C</th>
<th>Part D</th>
<th>Part E</th>
<th>Part F</th>
<th>Part G</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>14</td>
<td>56</td>
<td>14</td>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
</tbody>
</table>
Problem 3

Recursion (10p)

What is wrong with the following recursive function and how should it be corrected? (Hint, look at the results from the two calls to it.)

Indicate the change that should be done and give a short motivation/explanation of no more than two lines.

recursiveaddition.adb

```ada
with Ada.Integer_Text_IO;
use Ada.Integer_Text_IO;

procedure RecursiveAddition is
  function Add_Every_Other_From_Zero(N : Integer) return Integer is
    begin
      if (N = 0) then
        return 0;
      else
        return N + Add_Every_Other_From_Zero(N-2);
      end if;
    end Add_Every_Other_From_Zero;
  begin
    Put(Add_Every_Other_From_Zero(8));
    Put(Add_Every_Other_From_Zero(7));
  end RecursiveAddition;
```

The problem with the function is that it doesn’t handle odd numbers. (Calling it with 8 is ok, it will take away 2 until it reaches the base case, 0, where the recursion will terminate, but calling it with an odd number, e.g., 7 will take away 2 until the base case is reached… but the case will never occur! Starting with an odd numbers will eventually reach 1 from which 2 will be taken away ending up with -1, from which 2 will be taken away from which 2 will be taken away… Infinite recursion!)

There are several possible solutions.

1. Fix the base case, e.g.,
   “if (N <= 0) then” or “if (N = 0 or N = -1) then” or similar

2. Make the initial number odd (kind of solves the problem…)
   “return N + Add_Every_Other_From_Zero(N-2);” is changed to
   “if (N mod 2 = 0) then
      return N + Add_Every_Other_From_Zero(N-2);
    else
      return N + Add_Every_Other_From_Zero(N-1);
    end if;”
   or similar
**Problem 4**  
*Arrays and Records (15p)*

a. Consider the program `arrayinit.adb`.

```plaintext
arrayinit.adb

with Ada.Integer_Text_IO;
use Ada.Integer_Text_IO;

procedure ArrayInit is
  type TenArray is array (1..10) of Integer;
  Chunk : TenArray;
begin
  for I in TenArray'Range loop
    Put(Chunk(I));
  end loop;
end ArrayInit;
```

What warnings or errors can be expected during compilation of the program? (3p)

`arrayinit.adb:6:04: warning: “Chunk” is never assigned a value or anything indicating that there is a problem with the use of a variable which has not been initialized.`

What will the output be when running the program? (2p)

The memory where the array is allocated contains “random” values when the array is allocated (declared) and these values will be printed, i.e., garbage.

Write code to correct the problem and indicate where the code should go. (5p)

Anything that initializes the contents of the array prior to the “Put” is ok.

Examples:

At the declaration of the “Chunk” variable:

“Chunk : TenArray := (0,0,0,0,0,0,0,0,0,0);” or
“Chunk : TenArray := (others => 0)” or similar

Before the loop:

“for I in TenArray’Range loop
  Chunk(I) := 0;
end loop;”

Inside the loop, making the loop:

“for I in TenArray’Range loop
  Chunk(I) := 0;
  Put(Chunk(I));
  end loop;”

or anything similar.

Any value can be used (instead of the “0”).
b. Consider the program in nutsandbolts.adb. (5 p)

```
with Ada.Text_Io, Ada.Integer_Text_Io;
use Ada.Text_Io, Ada.Integer_Text_Io;

procedure NutsAndBolts is
  type Description is
    record
      L    : Integer;
      H    : Integer;
      W    : Integer;
      Cost : Float;
    end record;

  type Sets is array (1..30) of Description;
  type Part is (Bolt, Nut, Cap);
  type Catalog is array (Part'range) of Sets;

  type Warehouse is
    record
      Bolts_R_Us: Catalog;
      Hold_Tighter : Catalog;
    end record;

  Dist_Center : Warehouse;

begin
  -- Set length (L) of the 10th Nut in the Bolts_R_Us Catalog to 10 HERE.

end NutsAndBolts;
```

How would you go about setting the length (L) of the 10th Nut in the Bolts_R_Us Catalog to 10? Write the statement for the assignment in the code above. (10p)

```
Dist_Center.Bolts_R_Us(Nut)(10).L := 10;
```
Problem 5  

Number Conversions (15 P)

a. Match the decimal and hexadecimal numeric values on the left to their equivalent binary numeric values on the right. (12p)

```
EB₁₆  10001000₂
1₁₀   11101011₂
-120₁₀ (2’s complement)  00001101₂
-119₁₀ (1’s complement)   11110011₂
78₁₆ (2’s complement)      10001000₂
243₁₀                          10001100₂
3F₁₆                           11101101₂
```

Note that the bit patterns for -120 (2C) and -119 (1C) are the same (10001000)

b. Convert the following floating point number to decimal notation. Note: The number has been stored using scientific notation, i.e., the leftmost 1 has been removed. (3p)

111011001100 with 1 signbit, 4 bits exponent (excess4) and 7 bits mantissa =

```
1 1101 11001100
sign  exponent  mantissa
```

The sign bit is set so the number is negative.

Yes, excess4 allows for large numbers when there are 16 possible values (4 bits). To get the number of steps the float point should be moved 4 should be removed from the “1101” => “1001” = 9 steps.

The one removed from the mantissa “”(when stored using scientific notation) should be reinserted => 1.11001100, move float point nine positions => 1110011000 = 920

So,
**Problem 6**  

*Computer Architecture (10 P)*

Assume you have an 8-bit computer that contains two registers (R1 and R2) to contain data the CPU will use for arithmetic operations. This computer is designed to process the following legal Opcodes:

- 000 = stop
- 001 = load data stored at indicated address into R1
- 010 = load data stored at indicated address into R2
- 011 = store R1 into address location given
- 100 = store R2 into address location given
- 101 = add R1 to R2 (results stored in R2)

The three most significant bits contain the op-code. The remaining 5 bits contain the address of any data that is required by the operation.

**a.** The machine level program that is written below is equivalent to the Ada statement of $Y := X + Y + Z$; where $X$, $Y$, and $Z$ are stored in addresses $1010$, $1110$ and $1210$ respectively. Fill in the meaning of each of the statements. The first one has already been done for you.

<table>
<thead>
<tr>
<th>OpCode</th>
<th>Address</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>01011</td>
<td>(1110)  Load data stored at address 1110 (the # Y) into R2 (=0001001)</td>
</tr>
<tr>
<td>001</td>
<td>01010</td>
<td>(1010)  Load data stored at address 1010 (the # X) into R1 (=0000011)</td>
</tr>
<tr>
<td>101</td>
<td>00000</td>
<td>Add R1 and R2, store in R2 (=00011100)</td>
</tr>
<tr>
<td>001</td>
<td>01100</td>
<td>(1210)  Load data stored at address 1210 (the # Z) into R1 (=00010000)</td>
</tr>
<tr>
<td>101</td>
<td>00000</td>
<td>Add R1 and R2, store in R2 (=00011100)</td>
</tr>
<tr>
<td>100</td>
<td>01011</td>
<td>Store data inside R2 at address 1110 (the # Y)</td>
</tr>
<tr>
<td>000</td>
<td>00000</td>
<td>stop</td>
</tr>
</tbody>
</table>

**b.** Given the contents of memory BEFORE the execution of the program, fill in the contents of memory AFTER the above machine instruction program has executed:

<table>
<thead>
<tr>
<th>Address</th>
<th>Data (memory contents)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001010 (1010)</td>
<td>00000011 X</td>
</tr>
<tr>
<td>00001011 (1110)</td>
<td>00001001 Y</td>
</tr>
<tr>
<td>00001100 (1210)</td>
<td>00010000 Z</td>
</tr>
</tbody>
</table>

**BEFORE**

<table>
<thead>
<tr>
<th>Address</th>
<th>Data (memory contents)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001010 (1010)</td>
<td>00000011 (not changed)</td>
</tr>
<tr>
<td>00001011 (1110)</td>
<td>00011100 (the sum)</td>
</tr>
<tr>
<td>00001100 (1210)</td>
<td>00010000 (not changed)</td>
</tr>
</tbody>
</table>

**AFTER**
Problem 7

The Rover (20 P)

Student Ida Know has written the following lego program for the Mars Rover you built for this class. She did not comment it well at all. And some of the procedure names are boring and do not reflect the actual purpose of the code.

a. Comment her code in the areas that are outlined. Describe why a particular section of code exists rather than just describing what it does. (6p)

```plaintext
with Lego;
use Lego;

procedure Q4 is

Left_Wheel  : constant Output_Port := Output_A;
Right_Wheel : constant Output_Port := Output_C;
Left_Rot    : constant Sensor_Port := Sensor_1;
Right_Rot   : constant Sensor_Port := Sensor_3;
Light       : constant Sensor_Port := Sensor_2;

procedure P1 is
begin
--
  Output_Power(
    Output => Left_Wheel,
    Power  => Power_High);
  Output_Power(
    Output => Right_Wheel,
    Power  => Power_High);
--
  Config_Sensor(
    Sensor => Left_Rot,
    Config => Config_Rotation);
  Config_Sensor(
    Sensor => Right_Rot,
    Config => Config_Rotation);
--
  Config_Sensor(
    Sensor => Light,
    Config => Config_Light);

  Output_Power(Left_Wheel,7);
  Output_Power(Right_Wheel,7);
  Clear_Sensor(Left_Rot);
  Clear_Sensor(Right_Rot);
end P1;

procedure Drive_Foward(Clicks:Integer) is
begin
--
  Clear_Sensor(Left_Rot);
  Clear_Sensor(Right_Rot);
  Output_On_Reverse(Left_Wheel);
  Output_On_Reverse(Right_Wheel);
```

- Initialize the power to the motors.
- Set the sensors to be treated as rotation sensors (which they are).
- The sensor on the Light port is configured as a light sensor.
while (\text{abs(Get\_Sensor\_Value(Left\_Rot))<Clicks}) \ loop
    \text{Wait(10);} \\
end \ loop;

\text{Output\_Off(Left\_Wheel);} \\
\text{Output\_Off(Right\_Wheel);} \\
end \ Drive\_Forward;

\text{procedure P2 is} \\
begin \\
\text{---} \\
\text{Output\_On\_Forward(Left\_Wheel);} \\
\text{Output\_On\_Reverse(Right\_Wheel);} \\
\text{Wait(1000);} \\
\text{---} \\
\text{Output\_Off(Left\_Wheel);} \\
\text{Output\_Off(Right\_Wheel);} \\
end P2;

Value : \text{Integer;} \\
begin \\
\text{---} \\
\text{P1;} \\
\text{Select\_Display(Display\_Sensor\_2);} \\
\text{loop} \\
\text{Drive\_Forward(1000);} \\
\text{Wait(100);} \\
\text{Value := Get\_Sensor\_Value(Light);} \\
\text{P2;} \\
\text{Wait(100);} \\
\text{Value := Get\_Sensor\_Value(Light);} \\
end \ loop; \\
end.

b. Rename the procedures (P1, P2) so they have more appropriate names that reflect what the procedure does for the lego rover (4 p)

\text{P1}__\text{Initialize\_Rover / Setup\_Rover / similar…} \\
\text{P2}__\text{Turn\_Rover(_Left) or similar…}
c. I enjoyed the Lego portion of Unified C&P (please answer honestly). (2p)
   A. I agree
   B. I disagree
   C. Choose me and you get a zero
   D. I don’t know/I don’t understand.

   c. Short answer: Why should you call the lego procedure Clear_Sensor for the rotation sensor before doing a new rover maneuver? (2p)

If the sensor is not cleared the value it currently contains will be included in the maneuver! Another possible answer is the counter for the sensor will eventually overflow and reset to 0.

e. The Lego Rover ...(circle all that apply) (3p)

   1) runs Mission Critical Software.
   2) software is real time.
   3) is an embedded system.
   4) is an example of a Von Neuman Architecture.
   5) is used on the Pathfinder mission.
   6) is certified by NASA.
   7) has a separate floating point processor.

f. List 3 features of the standard Ada95 language that you cannot use with Ada Mindstorms and the Lego Rover (3p)

   1. ________________________________________________________________
   2. ________________________________________________________________
   3. ________________________________________________________________

There are numerous features that cannot be used, e.g., functions, more than 32+16 variables, multidimensional arrays, array initialization, floats, user defined types, most libraries, packages, etc.