Western Washington University
Computer Science Department

CSCI 141 Computer Programming I
Fall 2011

Laboratory Exercise 2

Objectives

1. Practice in use of the procedures from packages Ada.Text_IO and Ada.Integer_Text_IO.

2. Practice in use of selection and interaction constructs of the Ada language.

3. Practice in producing zipped tar files in Linux.

Submitting Your Work

In this lab exercise you will create 2 or 3 Ada programs. At the end of the lab exercise you will find instructions on how to put those files into a single zipped (compressed) tar (tape archive) file. Save your program files as the zipped tar file Wnnnnnnnn.tar.gz (where Wnnnnnnnn is your WWU W-number) and submit the file via the **Lab Exercise 2 Submission** item on the course web site. You must submit your file by 4:00pm on Friday, October 7.

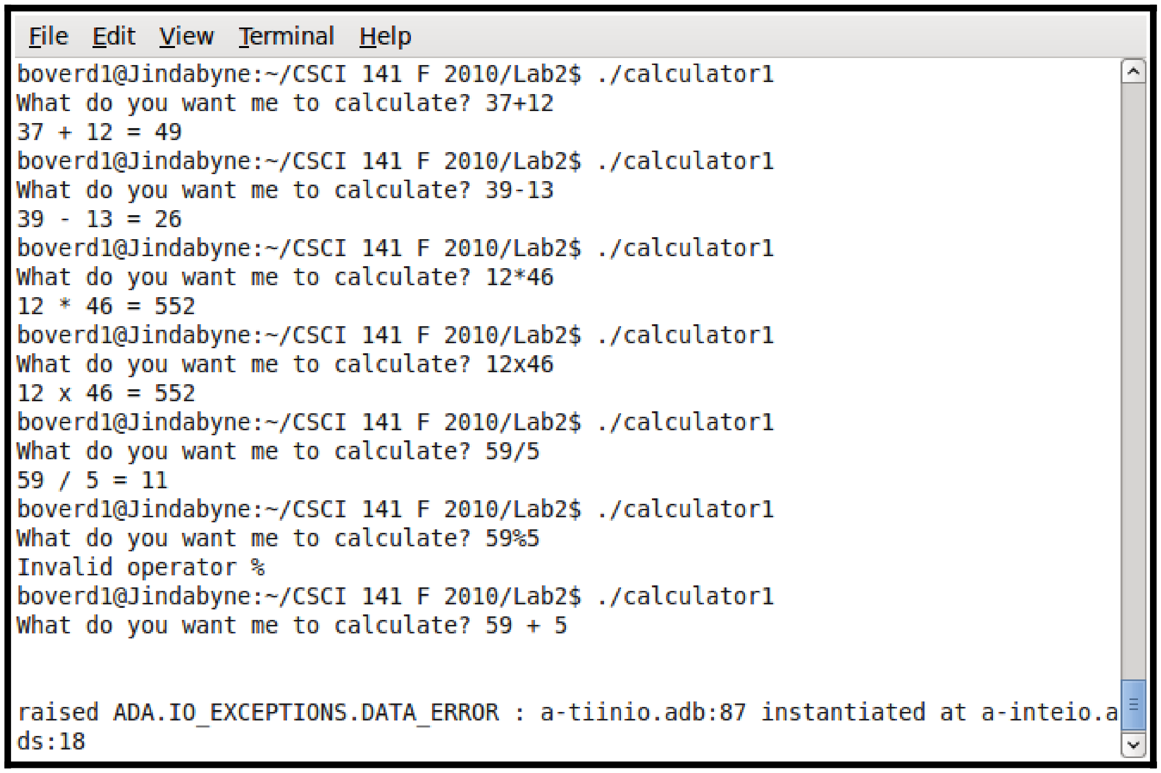
The Exercise

In this lab exercise, you will develop a calculator in the Ada language. You will develop 2 or 3 versions of the calculator, making each version more tolerant of user errors and more versatile than the previous version.

This lab exercise is to be completed in the Linux operating system. If your computer is currently running Windows, restart it in Ubuntu Linux.

Calculator 1

1. The first version of the calculator will accept as input from the user an integer number, an operator and another integer number and will then display the result. The valid operators are '+' for addition, '-' for subtraction, 'x' or '*' for multiplication and '/' for integer division. A working calculator is shown in the screen capture below.



```
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator1
What do you want me to calculate? 37+12
37 + 12 = 49
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator1
What do you want me to calculate? 39-13
39 - 13 = 26
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator1
What do you want me to calculate? 12*46
12 * 46 = 552
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator1
What do you want me to calculate? 12x46
12 x 46 = 552
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator1
What do you want me to calculate? 59/5
59 / 5 = 11
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator1
What do you want me to calculate? 59%5
Invalid operator %
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator1
What do you want me to calculate? 59 + 5


raised ADA.IO_EXCEPTIONS.DATA_ERROR : a-tiinio.adb:87 instantiated at a-inteio.a
ds:18
```

For this program you will need 5 variables: three integers, one character and one boolean. In this description I will use the names `First`, `Second` and `Result` for the integers, `Operator` for the character and `Valid` for the boolean, but you may choose whatever legal names you want.

You need packages `Ada.Text_IO` and `Ada.Integer_Text_IO` for the input and output.

Call your procedure `calculator1` and save the file as `calculator1.adb`.

2. Here is the algorithm for calculator1, expressed in *structured English*.

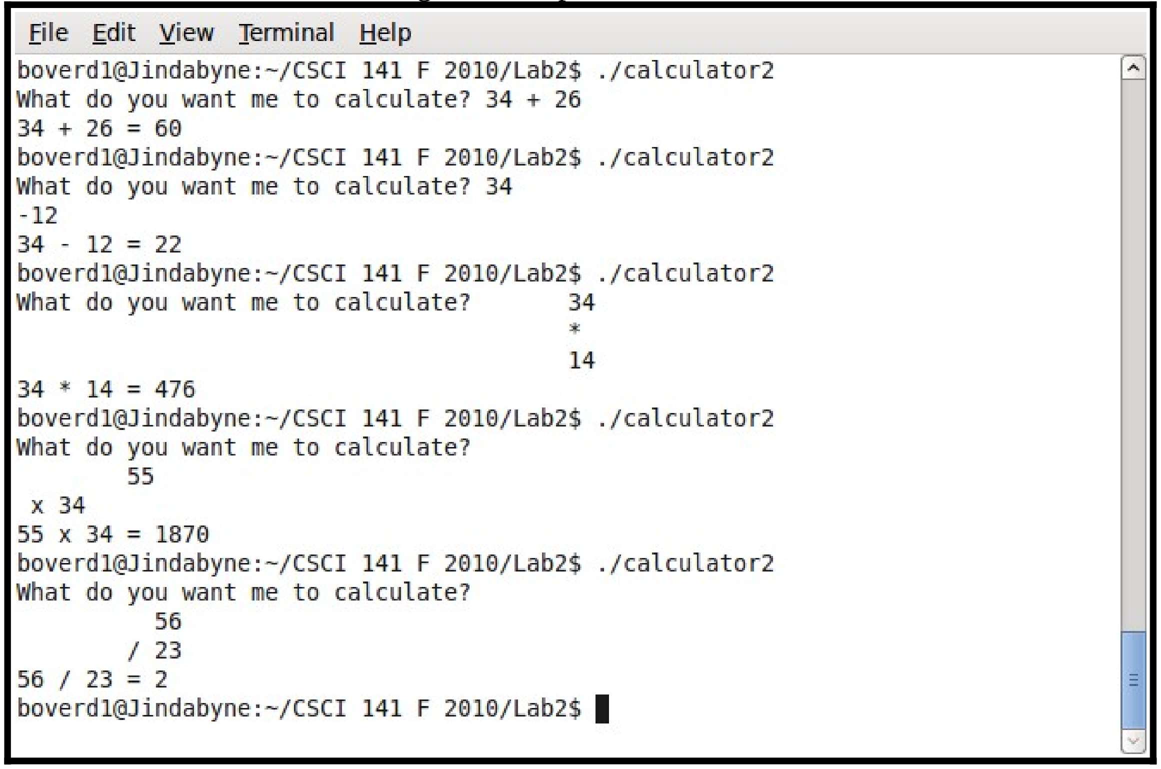
```
Set Valid to true
Display the prompt “What do you want me to calculate? “
Get the first integer
Get the operator
Get the second integer
If the operator is '+'
      Set the result to the first integer plus the second integer
If the operator is '-'
      Set the result to the first integer minus the second integer
If the operator is '*' or 'x'
      Set the result to the first integer multiplied by the second
      integer
If the operator is '/'
```

```
        Set the result to the first integer divided by the second
        integer
  If the operator is neither '+', '-', '*', 'x' nor '/'
        Display the message “Invalid operator “, followed by the
        operator and end the line
        Set Valid to false
  If Valid is true
        Display the first integer, space, the operator, space, the
        second integer, space, '=', space and then the result, all on
        one line and then end the line
```

3. Write the Ada procedure to implement this algorithm, compile it and test it. Be sure to follow the coding standards shown at the end of this lab description.

Calculator 2

Calculator1 does not tolerate spaces before the operator. `Ada.Integer_Text_IO.Get` is smart enough to skip over spaces, tabs and new lines before the integer number, but we need to make the calculator handle spaces, tabs or new lines before the operator. This is shown in the following screen capture.



```
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator2
What do you want me to calculate? 34 + 26
34 + 26 = 60
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator2
What do you want me to calculate? 34
-12
34 - 12 = 22
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator2
What do you want me to calculate?      34
                                        *
                                       14
34 * 14 = 476
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator2
What do you want me to calculate?
        55
 x 34
55 x 34 = 1870
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator2
What do you want me to calculate?
          56
        / 23
56 / 23 = 2
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$
```

Save your file `calculator1.adb` as `calculator2.adb`, so you can modify `calculator2.adb` without messing up `calculator1.adb`.

Change the name of the procedure to `calculator2`.

Within the declaration area of procedure `calculator2`, write a new procedure `Skip_spaces`, which will skip any whitespace (spaces, tabs, new lines). To do this we can use the procedure `Look_ahead` from `Ada.Text_IO`. `Look-ahead` peeps at the next character but leaves it for the next call to `Get`. It also detects the end of line. The procedure has two parameters: a character (the procedure sets this parameter to the next character in the input) and a boolean which the procedure sets to true if it detects the end of line before any other character.

Within procedure `Skip_spaces`, you need two variables: a character and a boolean. I will use the names `Next` for the character and `EndLine` for the boolean, but you may use whatever legal names you like.

Here is the algorithm for `Skip_spaces`, expressed in *structured English:*

```
Call Look_ahead with parameters Next and EndLine
While Next is a space or tab or EndLine is true
  if EndLine is true
    Skip the rest of the line
  else
    Get Next
  Call Look_ahead with parameters Next and EndLine
```

To skip the rest of the line, use the `Skip_line` procedure from `Ada.Text_IO`.

The tab character is defined in Ada under the name ASCII.ht.

Include a call to `Skip_spaces` in procedure `calculator2` just before you get the operator.

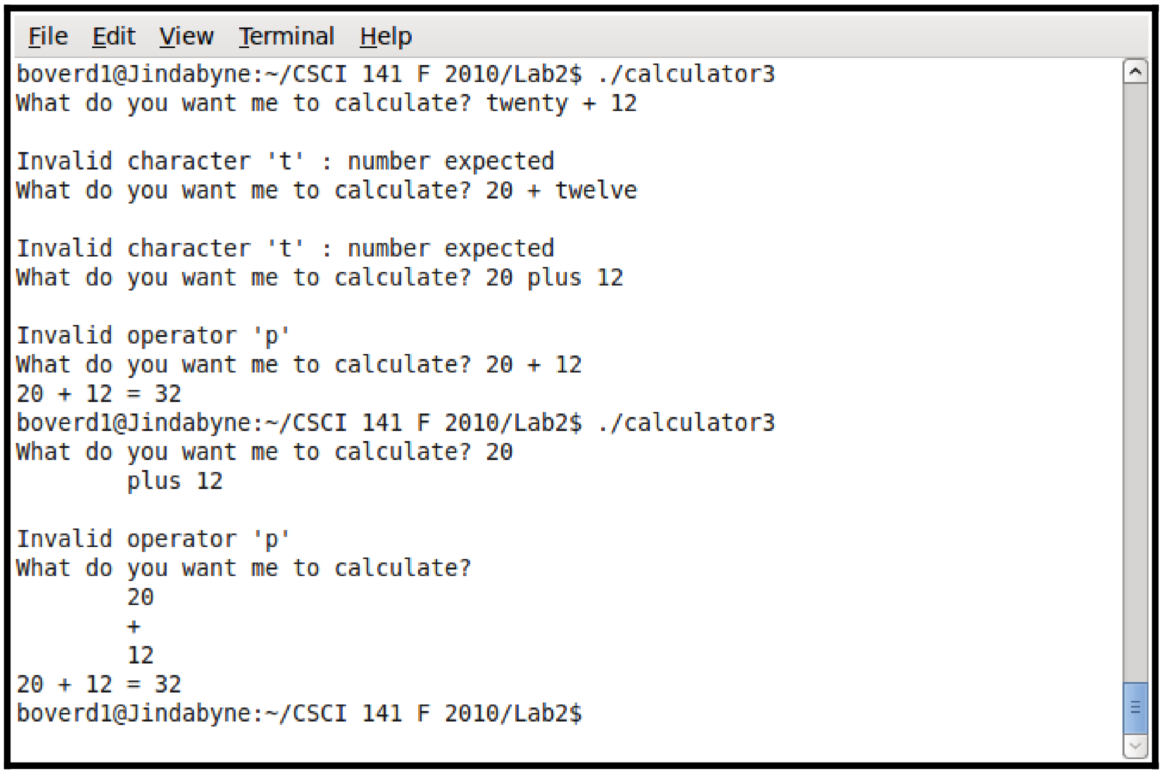
Compile `calculator2.adb` and test it. You should be able to include spaces, tabs and new lines anywhere in the input without causing the program to crash or report an invalid operator.

Be sure to follow the coding standards shown at the end of this lab description.

This is sufficient for full points for the lab exercise. If you have the time and interest, keep going with the third version of the calculator. Otherwise, you may skip to Step 6 “Saving your files in a zipped tar file”

Calculator 3

4. The calculator is still intolerant of user errors when typing the integer number. If the user types anything other than digit when the program expects to read a number, the program crashes. Our next step is to make the calculator handle user errors and make a fresh request for input, without crashing. This is shown in the following screen capture. Notice that errors when inputting the integer numbers and errors in the operator are handled in the same way – the program asks the user to try again.



```
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator3
What do you want me to calculate? twenty + 12

Invalid character 't' : number expected
What do you want me to calculate? 20 + twelve

Invalid character 't' : number expected
What do you want me to calculate? 20 plus 12

Invalid operator 'p'
What do you want me to calculate? 20 + 12
20 + 12 = 32
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$ ./calculator3
What do you want me to calculate? 20
        plus 12

Invalid operator 'p'
What do you want me to calculate?
        20
        +
        12
20 + 12 = 32
boverd1@Jindabyne:~/CSCI 141 F 2010/Lab2$
```

Save your file `calculator2.adb` as `calculator3.adb`, so you can modify `calculator3.adb` without messing up `calculator2.adb`.

5. To make the program detect and handle errors in the input, we will use two new procedures.

   Procedure `Look_for_digit` will check the next non-whitespace character. If that character is not a digit, it will set the variable `Valid` to false and skip the rest of the input.

   Procedure `Look_for_operator` will check the next non-whitespace character. If that character is not a valid operator, it will set the variable `Valid` to false and skip the rest of the input.

   Both these procedures will use the `Look_ahead` and `Skip_line` procedures from `Ada.Text_IO`.

   We will use a loop at the start of procedure `calculator3` to keep asking for input until it receives an integer number, a valid operator and an integer number.

   Within each procedure `Look_for_digit` and `Look_for_operator`, you need two variables: a character and a boolean. I will use the names `Next` for the character and `EndLine` for the boolean, but you may use whatever legal names you like.

   Here is the algorithm for `Look_for_digit`, expressed in *structured English:*

```
Call procedure Skip_spaces to skip whitespace
Call Look_ahead with parameters Next and EndLine
If Next is not a digit
```

```
  Display a message saying that the character is invalid and a number
  is expected
  Set Valid to false
  Call Skip_line to skip the rest of the line
```

Here is the algorithm for `Look_for_operator`, expressed in *structured English:*

```
Call procedure Skip_spaces to skip whitespace
Call Look_ahead with parameters Next and EndLine
If Next is not a valid operator
  Display a message saying that the operator is invalid
  Set Valid to false
  Call Skip_line to skip the rest of the line
```

Change the body of procedure `calculator3` to match the following algorithm:

```
Set Valid to false
while not Valid
  Display the prompt “What do you want me to calculate? “
  Set Valid to true
  Call Look_for_digit
  if Valid
    Get the first integer
    Call Look_for_operator
    if Valid
      Get the operator
      Call Look_for_digit
      If Valid
        Get the second integer
If the operator is '+'
  Set the result to the first integer plus the second integer
If the operator is '-'
  Set the result to the first integer minus the second integer
If the operator is '*' or 'x'
  Set the result to the first integer multiplied by the second
  integer
If the operator is '/'
  Set the result to the first integer divided by the second integer
  Display the first integer, space, the operator, space, the second
  integer, space, '=', space and then the result, all on one line and
  then end the line
```

Compile `calculator3.adb` and test it. You should be able to include spaces, tabs and new lines anywhere in the input without causing the program to crash or report an invalid operator. You should be able to type in non-digits when a number is expected and the program will display an error message and ask again for input, without crashing.

Saving your files in a zipped tar file

6. You only submit the .adb files. First you need to bundle them up into a single tar file. The term “tar” is an abbreviation of “tape archive” and goes back to the days when people would save a back-up copy of their files on magnetic tape. Nowadays, with the price of large disk drives so low, nobody uses tape anymore, but the concept of tar files survives.

   Use the command:

   ```
   tar -cf Wnnnnnnnn.tar calculator*.adb
   ```

   (where Wnnnnnnnn is your W-number).

   The -cf specifies two options for the tar command: 'c' means create and 'f' means that the name of the resulting tar file comes next in the command.

   Following the name of the tar file, we list the files to be included in the tar file. In this case, we want the files `calculator1.adb`, `calculator2.adb` and (if you went that far with the exercise) `calculator3.adb`. We could list them individually as:

   ```
   tar -cf Wnnnnnnnn.tar calculator1.adb calculator2.adb calculator3.adb
   ```

   but it is easier, shorter and less error-prone to use the wild-card character '*', as in the command

   ```
   tar -cf Wnnnnnnnn.tar calculator*.adb
   ```

   This will include every file in the current directory whose name starts with “calculator”, ends with “.adb” and has zero or more characters in between.

7. If you now use the command `ls` you should now see the file Wnnnnnnnn.tar in your directory.

8. If you use the command

   ```
   tar -tf Wnnnnnnnn.tar
   ```

   (where Wnnnnnnnn is your W-number), it will list the files within the tar file.

9. Now compress the tar file using the gzip program:

   ```
   gzip Wnnnnnnnn.tar
   ```

   By using the `ls` command again, you should see the file Wnnnnnnnn.tar.gz in your directory. This is the file that you need to submit through the **Lab Exercise 2 Submission** link in the moodle web site.

Don’t forget to submit your zipped tar file!

Coding Standards

1. Use meaningful names that give the reader a clue as to the purpose of the thing being named.

2. Use named constants instead of repeated use of the same numeric constant.

3. Use comments at the start of the program to identify the purpose of the program, the author and the date written.

4. Use comments at the start of each procedure to describe the purpose of the procedure and the purpose of each parameter to the procedure.

5. Use comments at the start of each section of the program to explain what that part of the program does.

6. Use consistent indentation:

   - The declarations within a procedure must be indented from the **procedure** and **begin** reserved words. The body of the procedure must be indented from the **begin** and **end** reserved words. Example of procedure indentation:

   ```
   procedure DoStuff is
        Count : integer;
        Valid : boolean;
   begin
        Count := 0;
        Valid := false;
   end DoStuff;
   ```

   - The statements within the then-part, each elsif-part and the else-part of an if statement must be indented from the reserved words if, elsif, else and end.

   ```
   if Count > 4 and not Valid then
        Result := 0;
        Valid := true:
   elsif Count > 0 then
        result := 4;
   else
        Valid := false;
   end if;
   ```