CSCI 345
Assignment 1
Due: Midnight, Friday, April 19

Introduction

The purpose of this assignment is to give you an opportunity to work with objects in Ada.

Problem Statement

I am providing an implementation of a Ropes type that was discussed in class on Tuesday, April 9. I am also providing a test suite that can be used to test the changes that you will be making to the provided Ropes type. I also demonstrated the test suite on Tuesday.

There are three parts to this assignment. The first two parts require you to modify the provided implementation. Part three asks you to discuss how you might make another change.

Assignment

1. On moodle I have placed a folder labeled "Assignment 1 Data". In this folder you will find:
    - Ropes.zip: A zip file containing the source code for my implementation of the Ropes type.
    - BoehmRopes.pdf: A paper discussing the Ropes concept as originally implemented.
2. In the zip file you will find all the source files for the Ropes implementation, organized as follows:
    - ropes.ads, ropes.adb -- the parent package containing the Rope and Rope_Impl types
    - ropes-string_impl.ads, ropes-string_impl.adb -- a child package of Ropes containing the String_Impl type
    - ropes-concat_impl.ads, ropes-concat_impl.adb -- a child package containing the Concat_Impl type
    - ropes-test_utils.ads, ropes-test_utils.adb -- a child package of Ropes that provides access to Rope and Rope_Impl internals. This package is used by the test software.
    - ropes.gpr -- a gnatmake project file for this project
    - aunit/ -- a directory containing the AUnit package that is used by the testing software
    - test/ -- a directory containing the tests for the Ropes package. The main program test-ropes.adb can be found in this directory
    - obj/ -- an empty directory that is used by the ropes project. All of the ".o" and ".ali" files created by the compiler will go here.

3. You must build the provided software and run "test_ropes -p0" successfully before you proceed with any modifications of the software. See additional information, below, on how to do this.

Assignment Part 1

The Ropes implementation contains a String_Impl type that is used hold a String of any non-zero length. This can be quite inefficient if a Rope is created by concatenating characters one (or a few) at a time.

You are to change the String_Impl type to an abstract type and add two new types, Small_String_Impl, and Large_String_Impl, which are derived from String_Impl. Large_String_Impl will work the same as the current String_Impl. containing a pointer to an allocated String. The allocated String must have a length greater than the constant Max_Small_String. If the String length is less than or equal to Max_Small_String, it must be contained in a Small_String_Impl.

Instead of pointing to a String, a Small_String_Impl will contain a String of length Max_Small_String. The contents of the string for the String_Impl will be stored in the first Length characters of the String embedded in the Small_String_Impl. All of the Rope_Impl operations will need to be redone to do the right thing for Small_String_Impl.

You must also modify the file ropes-test_utils.adb so the functions Is_Small_String_Impl and Is_Large_String_Impl operate correctly. You can use the function Is_String_Impl as a model for how to do this.

You can make this change by only modifying the files ropes-string_impl.ads, ropes-string_impl.adb, and ropes-test_utils.adb.

Once you have made the required changes, make sure to build and test the software. All of the part 1 tests should pass. Make sure to do this before proceeding to the next part of the assignment.

Make sure to read the Notes section, below.

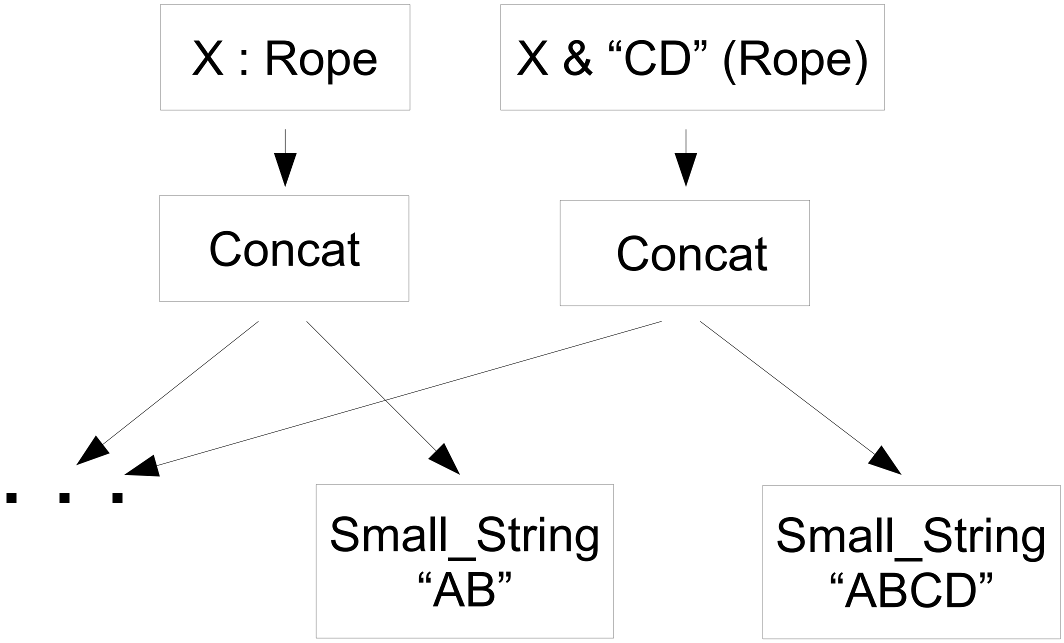
Assignment Part 2

Even after having done Part 1, building a String one character at a time will result in a very large tree. The solution to this problem is to avoid creating a lot of concatenated Small_String_Impls. As a move in this direction, you are to implement a change that replaces a Small_String_Impl with another Small_String_Impl when the original Small_String_Impl is a left-most (right-most) Rope_Impl and is concatenated on the left (right) with a short string such that the concatenation will still fit in a Small_String_Impl. (See the diagram below for an explanation.)

In this diagram, the right-most Rope_Impl of X is a Small_String_Impl containing the String "AB". The Rope X is concatenated on the right with the String "CD". As shown in the picture, the new Rope shares the left part of the Concat_Impl with X and has a new Small_String_Impl on the right which contains the String "ABCD". The Rope_Impl on the left must have its Ref_Count incremented since there are now two references to it where there was only one before.

Here is a set of steps for implementing this change:

1. In the private part of ropes.ads, add two "&" methods, "&"(access Rope_Impl, String) : Rope_Impl_Access and vice versa to ropes.ads. Both these are abstract.



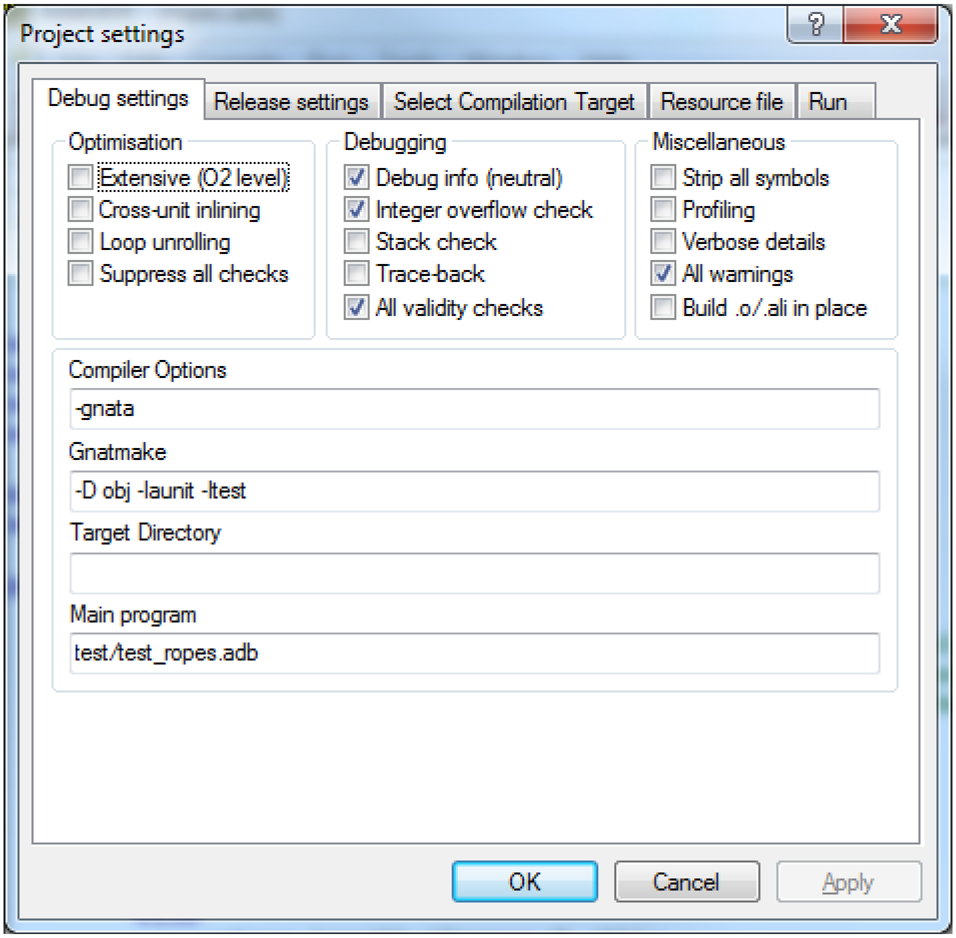
2. In ropes.adb, modify the two existing methods "&"(Rope, String) : Rope and vice versa to make use of the two new methods you added to Ropes.ads in (1).
3. In concat_impl.ads and string_impl.ads add specs for methods overriding the two abstract operations you added in (1). For each of the two new operations, you'll need an overriding method in concat_impl.ads and two overriding methods in string_impl.ads. (That's a total of six overriding methods.)
4. In concat_impl.adb add the bodies for the overriding specs you added to concat_impl.ads in (3). These operations should work by creating a new Concat_Impl that contains the original sub-Rope_Impl on one side and the result of concatenating of the String and the other sub-Rope_Impl on the other side. (See the diagram, above.)
5. In string_impl.adb add the bodies for the overriding specs you added in (3). These operations should do one of two things:

   (a) If you are concatenating a String with a Small_String_Impl and the result will fit in a Small_String_Impl, create a new Small_String_Impl with the concatenation and return that Small_String_Impl.

   (b) If you are concatenating with a Large_String_Impl or the concatenation won't fit in a Small_String_Impl (essentially option (a) won't work), the make a Small_String_Impl or Large_String_Impl that holds the String and a Concat_Impl that concatenates the existing String_Impl with the new String_Impl.

6. Make sure you get the reference counting right or test_ropes will complain at you for memory problems.

Once you have completed the change, you need to run the test suite to ensure that the change works.



Optimisation
□ Extensive (O2 level)
□ Cross-unit inlining
□ Loop unrolling
□ Suppress all checks

Debugging
☑ Debug info (neutral)
☑ Integer overflow check
□ Stack check
□ Trace-back
☑ All validity checks

Miscellaneous
□ Strip all symbols
□ Profiling
□ Verbose details
☑ All warnings
□ Build .o/.ali in place

Compiler Options
-gnata

Gnatmake
-D obj -Iaunit -Itest

Target Directory

Main program
test/test_ropes.adb

Assignment Part 3

For this part of the assignment, do not do any coding. You simply need to describe how you might go about accomplishing the following:

It would be nice if you could build a Rope by simply adding the new characters to an already existing Small_String_Impl, as opposed to creating a parallel tree as shown in the diagram. Assume that we are going to add two new operations: Append(Rope, String) and Prepend(Rope, String). For example, given `X : Rope`, `Append (X, “CD”);` would be logically equivalent to `X := X & “CD”;` We would like this operation to directly modify the Rope X if this is feasible. You should note that this is feasible as long as the modification does not modify any other Ropes that might share structure with X.

Answer the following question: Based on your implementation in parts 1 and 2, how would you go about doing this and determining that it is safe to do so?

Note: You cannot assume that just because the Ref_Count on a given Small_String_Impl is 1 that that Impl is not shared higher up in the tree.

Building the Software

You probably don't want to use Adagide to build this assignment. (I provide some instructions below for trying to do this.) There are two approaches that I know work:

Option one: use your favorite text editor (I use Notepad++ on Windows and TextWrangler on Macs) to edit the files. Open a command line window and, making sure that your current directory is the one containing ropes.gpr, execute the command "gnatmake -Propes". This will attempt to build the project. You can use "gnatmake -Propes -f" to do a "clean" build.

Option two: use the GPS IDE that comes with the gnat distribution. Open ropes.gpr using GPS. You can now use GPS to edit the project and build the project.

Option three: using Adagide, open one of the ropes files (e.g. ropes.ads). Then on the Tools menu choose "Project settings in current directory". In the resulting dialog use the settings shown in the screen shot, above.

Note that the vertical bars in the Gnatmake box are capital Is. Adagide will now behave approximately correctly. I tested this and believe that the "Build" command will work properly. The "Compile" command will work but leaves files in different places than gnatmake.

Testing the Software

The project will build the a test_ropes executable program. the comments at the beginning of test_ropes.adb describe how to run the program. Here are some samples:

```
AdaRope chris$ ./test_ropes
OK Null_Rope:Test_Length
OK Null_Rope:Test_To_String
[136 OKs deleted]
OK Concat_Rope_Int:Test_Correct_Left_Impl_Int-Lrg_Right_Concat
OK Concat_Rope_Int:Test_Correct_Right_Impl_Int-Lrg_Right_Concat

Total Tests Run:   140
Successful Tests:  140
Failed Assertions: 0
Unexpected Errors: 0
Cumulative Time: 0.008560 sec. seconds

AdaRope chris$ ./test_ropes -p0
OK Null_Rope:Test_Length
[54 OKs deleted]
OK Concat_Rope_Int:Test_Correct_Right_Impl_Int-Base

Total Tests Run:   56
Successful Tests:  56
Failed Assertions: 0
Unexpected Errors: 0
Cumulative Time: 0.004432 sec. seconds

AdaRope chris$ ./test_ropes -p1
OK Null_Rope:Test_Length
[132 OKs deleted]
OK Concat_Rope_Int:Test_Correct_Right_Impl_Int-Short_Right

Total Tests Run:   134
Successful Tests:  134
Failed Assertions: 0
Unexpected Errors: 0
Cumulative Time: 0.009379 sec. Seconds

AdaRope chris$ ./test_ropes -p2 -m String_Rope:Test_Length-Small_Min
Inc_Ref_Count [SMALL_STRING_IMPL@4298136416 A] to 1
```

```
Adjust [Rope@4296267712 [SMALL_STRING_IMPL@4298136416]]
Inc_Ref_Count [SMALL_STRING_IMPL@4298136416 A] to 2
Finalize [Rope@140734799801376 [SMALL_STRING_IMPL@4298136416]]
Dec_Ref_Count [SMALL_STRING_IMPL@4298136416 A] to 1
Adjust [Rope@140734799802176 [SMALL_STRING_IMPL@4298136416]]
Inc_Ref_Count [SMALL_STRING_IMPL@4298136416 A] to 2
Finalize [Rope@4296267712 [SMALL_STRING_IMPL@4298136416]]
Dec_Ref_Count [SMALL_STRING_IMPL@4298136416 A] to 1
Finalize [Rope@140734799802176 [SMALL_STRING_IMPL@4298136416]]
Dec_Ref_Count [SMALL_STRING_IMPL@4298136416 A] to 0

OK String_Rope:Test_Length-Small_Min

Total Tests Run:   1
Successful Tests:  1
Failed Assertions: 0
Unexpected Errors: 0
Cumulative Time: 0.000233 sec. seconds
Finalize [Rope@4296218592 null]
```

Note that using the "-m" option will generate a lot of output. It should only be used when you restrict the tests that are run to only a couple of tests.

Notes

1. For parts 1 and 2 you will probably find that you need to convert a String of length less than eight to a string of length eight (by padding the extra characters.) The cleanest way I have found to do this using the Head function in the library package Ada.Strings.Fixed. Here's a URL: http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-4-3.html#I5210. Check paragraph number 98 for a description of the function. (Yes, I warned you away from the Language Reference Manual (LRM). But, one of the good uses is the descriptions of the contents of the standard library.

What to Submit

This assignment is due by midnight, January 29. Make a zip file of the source that you changed. Don't include any source you didn't change or any of the .o, .ali, or executable files. In the zip file, also include a text file that shows the results of running test_ropes with the version of the program you are submitting. Submit the zip file on moodle. Also submit a document (.odt, .doc, .docx, .pdf, or .txt) that contains your answer for part 3 of the assignment.

Grading

Grading for this lab is as follows:

- Part 1 works 40 points
- Part 2 works 45 points
- Answer to Part3 15 points

I will be reviewing the changes you make to the code. I would appreciate it if you make an effort to make your code easy to read and understand. I will mark you down if I have problems reviewing your changes. You can use my code as an example of what I'm expecting.

Please make sure that your program compiles and runs. If it does not, your grade will suffer a major deduction. As I mentioned on the first day of class, a program that is late and mostly works will get a better grade than one that does not compile.

Total: 100 points. Partial credit will be given.