RHODES UNIVERSITY

Computer Science 301 - 2015 - Programming Language Translation

Well, here you are. Here is the free information you have all been waiting for, with some extra bits of advice:

- Don't panic. It may be easier than you might at first think.
- The problems in the examination based on the exercise posed below will need rather careful thought. I shall be looking for evidence of mature solutions, not crude hacks.
- Work in smaller, rather than larger groups. Too many conflicting ideas might be less helpful than a few carefully thought out ones.
- Do make sure you get a good night's sleep!

How to spend a Significant Special Sunday

From now until about 22h30 tonight, Computer Science 3 students have exclusive use of the Hamilton Laboratories. You are encouraged to throw out anyone else who tries to use it, but we hope that this does not need to happen. At about 22h30 we may have to rearrange things for tomorrow. If there is still a high demand we shall try to leave some computers for use until later, but by then we hope you will have a good idea of what is involved.

This year the format of the examination is similar to that of the last few years. The problem set below is only part of the story. At about 16h30 you will receive further hints as to how this problem should be solved (by then I hope you might have worked this out for yourselves, of course). You will be encouraged to study the problem further, and the hints in detail, because during the examination tomorrow you will be set further questions relating to the system - for example, asked to extend the solution you have worked on in certain ways. It is my hope that if you have really understood the material today, these questions will have solutions that come readily to mind, but of course you will have to answer these on your own.

My "24 hour exam" problems have all been designed so that everyone should be able to produce at least the basic solution, with scope given for top students to demonstrate their understanding of the subtler points of parsing, scanning and compiling. Each year I have been astounded at the quality of some of the solutions received, and I trust that this year will be no exception.

Please note that there will be no obligation to produce a machine-readable solution in the examination (in fact, doing so is quite a risky thing, if things go wrong for you, or if you cannot type quickly). The tools will be provided before the examination so that you can experiment in detail. If you feel confident, then you are free to produce a complete working solution to Section B during the examination. If you feel more confident writing out a neat detailed solution or extensive summary of the important parts of the solution, then that is quite acceptable. Many of the best solutions over the last few years have taken that form.

During the first few hours you will be able to find the following files in the usual places, or by following links from the course web page:

- All the files as were made available for the practical course.
- The file `FREE1.ZIP`, which contain the C# versions of the Coco/R system, and its support files, some grammar and skeleton files and test data for today's exercise. The kit also contains a PDF version of the Coco/R user manual (`CocoManual.pdf`) and library summaries (`Library.pdf`).

At about 16h30 a new version of the exam kit will be posted (`FREE2.ZIP`), and a further handout issued, with extra information, to help students who may be straying way off course.

Today things should be familiar. You could, for example, log onto the `D:` or `J:` drive, use `UltraEdit` or `NotePad++` and `LPRINT` to edit and print out files, use `CMAKE` … generally have hours of fun.

Note that the exam setup tomorrow will have *no* connection with the outside world - no Google, FaceBook, ftp client, telnet client, shared directories - not even a printer.

Today you may use the files and systems in any way that you wish, subject to the following restrictions: *Please*

observe these in the interests of everyone else in the class.

(a) When you have finished working, **please** delete your files from any shared drives, so that others are not tempted to come and snoop around to steal ideas from you.

(b) You are permitted to discuss the problem with one another, and with anybody not on the "prohibited" list.

(c) You are also free to consult books in the library. If you cannot find a book that you are looking for, it may well be the case that there is a copy in the Department. Feel free to ask.

(d) Please do not try to write any files onto the C: drive, for example to `C:\TEMP\`

(e) If you take the exam kit to a private machine you will need to have the .NET framework installed.

I suggest that you *do* spend some of the next 24 hours in discussion with one another, and some of the time in actually trying out your ideas. If you are well prepared for the exam you should have plenty of time in which to prepare and test your ideas - go for it, and good luck.

If you cannot unpack the file, or have trouble getting the familiar tools to work (unlikely!), you may ask me for help. You may also ask for explanation of any points in the question that you do not understand, in the same way that you are allowed to ask before the start of an ordinary examination. You are no longer allowed to ask me questions about any other part of the course. Sorry: you had your chance earlier, and I cannot allow this without risking the chance of sneak questions and suggestions being called for.

If you cannot solve the problem completely, don't panic. It has been designed so that I can recognize that students have reached varying degrees of sophistication and understanding.

How you will spend a Merry Mind-blowing Monday Morning

Before the start of the formal examination the laboratory will be unavailable. During that time

- The machines will be completely converted to a fresh exam system with no files left on directories like `D:` or `C:\TEMP`.
- The network connections will be disabled.

At the start of the examination session:

- You will receive ordinary answer books for the preparation of answers, and an examination paper.
- You will receive listings of various of the grammars and support files that you receive today. *You may annotate these during the exam to form part of your solution if you wish to submit hand-written answers to questions, and need to make reference to the code (possibly by the line numbers that are provided on the listings)*. In this case you should hand in the annotated listings with your answer book.
- You will be allocated to a computer and supplied with a `CONNECT` command for your own use. Once connected you will find an exam kit on the `J:` drive. This will contain the same Coco/R system and other files you have been given today, and in addition there will be "flat ASCII" machine readable parts of the examination paper itself, in files with names like `Q7.TXT` (Question 7). *There is no obligation to use a computer during the exam. You can answer on paper if you prefer - and yes, you can write in pencil if you prefer that - but please not in red ink.*
- At the end of the exam you will be given a chance to copy any files that you have edited or created on the D: or J: drive back to the server. This is done using a simple script and will be explained tomorrow.
- **Remember that tomorrow you may not bring anything into the room other than your student card and writing utensils, and especially not listings, disk drives, memory sticks, text books or cell phones.**

Preliminary to Section B of the examination

As by now you are well aware, the Parva language and its compiler, as found in the textbook, were designed to serve two roles - firstly, to provide a system in which small, but non-trivial, programs could be written and, secondly, and more importantly, to act as a system which could be extended in numerous ways to allow students to learn some of the techniques of language design and implementation.

The only data types found in the basic version of Parva were integers, booleans, and one-dimensional arrays of elements of those types. During the course you may have seen how the character data type can be added to this quite easily, as well as a few new statement forms.

Over the years, Parva has developed quite a cult following, especially amongst teachers who admire its simplicity. Inevitably some of these keep asking for new features ("it would be perfect", they say, "if you could just add this last little thing", unaware of the recursive nature of their suggestion!).

But one overwhelming request has just come in, a little more attractive than most. Beginners are often taught some programming logic using "Turtle Graphics", and as students at Rhodes University you must have encountered this concept at least once. The request from my contacts is for Turtle Graphics to be added to Parva - and for this to be done within the next 24 hours! Five minutes of brainstorming suggests they would be delighted if they could write little programs like:

```
// Simple Graphics Example - Rotating square - eg04

void Main() {
  InitGraphics(600, 400);
  int i = 1;
  while (i < 20) {
    Forward(100);
    TurnLeft(90); Forward(100);
    TurnLeft(90); Forward(100);
    TurnLeft(90); Forward(100);
    TurnLeft(108);
    i = i + 1;
  }
} // Main
```

Here's your chance to show off what you have learned in this course - and to get closer to getting that most prized possession - a degree from Rhodes University.

Where do you go from here?

Take a little time to read through the rest of this document carefully and make sure you understand it before you leap in and start to hack at a "solution".

In adopting the approach suggested below you can (and should) make use of the files provided in the examination kit `free1.zip`, which you will find on the course website. In particular, you will find, after unzipping this, a structure like that below - and it is recommended that you retain this directory structure

```
work\CMake.bat              Script to generate and compile the Parva compiler
work\CSharp.bat             Script to compile the C# graphics examples

work\CocoManual.pdf         Documentation
work\Library.pdf

work\Coco.exe               Coco compiler and basic frame files
work\Scanner.frame
work\Parser.frame
work\Driver.frame
work\Parva.atg              Parva grammar
work\Parva.frame            Parva frame file

work\Library.cs             Rhodes C# library

work\Parva\CodeGen.cs       Auxiliary routines for Parva compiler
work\Parva\PVM.cs
work\Parva\Table.cs

work\Parva\TurtleLib.cs     Auxiliary routines for graphics
work\Parva\GraphicsLib.cs

work\examples\eg*.cs        Simple C# example turtle graphics programs
work\examples\eg*.pav       Equivalent Parva example turtle graphics programs
```

That is, you have been provided with

- The usual frame files, and the Coco/R system;
- The files to build a Parva level 2 compiler using the Java/C# expression precedence hierarchy, that is, all the files needed to build a Parva compiler very similar to the one supplied to you during the final practical - with a few of the refinements made during that exercise.
- The file `.\Parva\GraphicsLib.cs`, familiar from a previous practical, which defines two low-level graphics classes, `GWindow` and `GraphicsWindow` (you should not modify these).
- The incomplete file `.\Parva\TurtleLib.cs`, which suggests the form of a user-level graphics class `Turtle` (which, obviously, you must modify).
- The folder `.\examples`, which contains some example programs, like the one given above - feel free to adapt or to invent your own. You may use these for testing purposes (several of them are very simple; see the attached listing of them all).

Firstly, develop the `Turtle` class (`TurtleLib.cs`) so that it can be used in a standalone C# program to draw some simple patterns and graphs as exemplified by the various small C# programs in the kit that match their equivalent Parva programs, for example:

```
// Simple Graphics Example - Rotating square - eg04

using System;

class Program {

  static void Main(string[] args) {
    Turtle.InitGraphics(600, 400);
    int i = 1;
    while (i < 60) {
      Turtle.Forward(100);
      Turtle.TurnLeft(90); Turtle.Forward(100);
      Turtle.TurnLeft(90); Turtle.Forward(100);
      Turtle.TurnLeft(90); Turtle.Forward(100);
      Turtle.TurnLeft(108);
      i = i + 1;
    }

    Console.Write("\nHit Enter to Exit!");
    Console.ReadLine();

  } // Main

} // Program
```

Of course, the `Turtle` class should make use of the supplied `GraphicsWindow` and `GWindow` classes. Remember to "keep it as simple as you can, but no simpler". A skeleton file appears on the last page of this handout - restrict yourself to implementing only the features suggested there.

Remember, from an earlier prac, that a call to `GraphicsLib.GraphicsWindow(width, height)` opens a window in which other drawing functions can take effect. This window is measured in "pixel" units, with an *origin* (0, 0) at the top left, a horizontal X axis pointing to the *right* spanning the range `(0 ... width)` and a vertical Y axis pointing *downwards*, spanning the range `(0 ... height)`. While in a later development one would want to scale these axes, initially just limit your graphs to work in pixel coordinates - convenient values for `width` and `height` would be in the range 400 to 800 (this depends on your screen resolution).

Compiling the simple example C# programs is best done by using the supplied `csharp.bat` script - for example, from the command line:

```
CSharp  Eg01            (compile)
Eg01                    (execute)
```

You may have to move the command window clear of the graphics window to be able to see them unobscured.

To close the program and the graphics window you may have to click your mouse on the command window again

and then hit the <enter> key. Slightly crude, but this suffices at this stage, as this is only an intermediate step.

Secondly, develop the Parva grammar (`Parva.atg`) further so that the Parva compiler can parse the various Parva versions of the graphics programs given in the kit. To start with, concentrate on getting the syntax correct, and ignore code generation and execution. Follow the syntactic development with static semantic checking.

When you are ready to attempt code generation, decide on what new PVM codes must be introduced to allow the PVM to access the various routines implemented in your `Turtle` class. Go on to develop the code generator and the PVM to deal with these codes. Once you are in a position to execute your compiler, try to compile and run a sample of the supplied Parva programs, for example:

```
Parva  Eg01.pav                (compile and interpret)
```

To allow you to test your compiler on programs that are not meant to be executed (merely compiled so as to check the `.COD` file, syntax or the program listing), remember that the Parva compiler recognizes a `-n` command line flag, as in

```
Parva  voter.pav  -l  -d  -n
```

which won't call on the PVM to interpret the code after it has been generated. It now also recognizes a `-g` command line flag, as in

```
Parva  Eg03.pav  -l  -d  -g
```

which will follow a successful compilation with an immediate interpretation without the usual annoying questions.

When you have done all this satisfactorily you should be able to compile and run simple complete correct graphics programs in Parva (and also reject incorrect ones). You will also have a Parva compiler unlike any developed before! If you have time, you might like to explore adding further graphics facilities to the system.

CAUTION: As with all of the "24 hour exam" problems devised over many years, this is a non-trivial exercise, and you will have to think clearly to be able to solve it. By way of assistance, at about 16h30 there will be a further handout. You will need to study all of this material carefully, as the formal examination tomorrow will require you to demonstrate understanding of what you do today. Good luck!

```
// TurtleGraphics Library for use with Parva and the PVM (C#)
// P.D. Terry, Rhodes University, 2015
// As supplied for the free information phase of the 2015 examination.
// Feel free to complete, modify and extend this as you like, without
// modifying the GraphicsLib class.  Notice that all members are static.
// The exercise and exam will not require more than one turtle object.
//
// 2015/08/13

using GraphicsLib;    // interface onto basic graphics library
using System;

  public class Turtle {

    static private
      GraphicsWindow w = null;

      // Add variables for storing the state of the (single) turtle
      //  - suit yourself


    // TurtleGraphics ================================================

    public static void InitGraphics(int width, int height) {
      w = new GraphicsWindow(width, height);

      // .... add further code as necessary

    } // Turtle.InitGraphics

    public static void CloseGraphics() {

      // .... add further parameters and code as necessary

    } // Turtle.CloseGraphics

    public static void Home() {

      // .... add further parameters and code as necessary

    } // Turtle.Home()

    public static void PenUp() {

      // .... add further parameters and code as necessary

    } // Turtle.PenUp

    public static void PenDown() {

      // .... add further parameters and code as necessary

    } // Turtle.PenDown

    public static void TurnLeft() {

      // .... add further parameters and code as necessary

    } // Turtle.TurnLeft

    public static void Forward() {

      // .... add further parameters and code as necessary

    } // Turtle.Forward

    // Line Drawing ==================================================

    public static void Line(int x1, int y1, int x2, int y2) {

      w.DrawLine(x1, y1, x2, y2);

    } // Turtle.Line

  } // class Turtle
```