RHODES UNIVERSITY

Computer Science 301 - 2007 - 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 final solution(s) in section B tomorrow will need rather careful thought. I am 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 Special Sunday

From now until about 22h30 tonight, Computer Science 3 students have exclusive use of the Hamilton Laboratory. 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 Chris and I will have to move some computers around and prepare things for Monday. 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 somewhat different from years gone by, so as to counter what had happened where, essentially, one solution was prepared by quite a large group and then passed around to many others, who had probably not contributed to it in any real way. As in 2006, the problem set below is only part of the story. At 16h00 you will receive further hints as to how this problem should be solved (by then 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 have been 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 files `FREE1J.ZIP` (or `FREE1C.ZIP`) which contains the Java (or C#) versions of the Coco/R system, and its support files, and an attributed grammar for a version of Parva, similar to that in the last practical, and which includes the ability to declare and use variables of the `char` type. The kit also contains a PDF version of the Coco/R user manual (`CocoManual.pdf`).

At 16h00 new versions of the exam kit will be posted (`FREE2J.ZIP` and `FREE2C.ZIP`), and a further handout issued, with extra information.

Most of the machines in the labs will work in exactly the way they have done this semester. Some are set up to show you the configuration you can expect tomorrow. It is suggested that you work at one of those machines for a short time to make sure you know what to expect. To work on these machines you proceed as follows

- Open up a DOS window following the usual route
- Give the command

  ```
  CONNECT  TESTxxx   yyyyy
  ```

  where `xx` is a three digit number for the machine (typically in the range `101` through `123`) and `yyyy` is a 5 digit password.
- This will then allow you to log onto the `J:` drive, where you will find the file `EXAM.ZIP` (and `EXAMC.ZIP`) waiting for you.
- Give a command like

  ```
  UNZIP   FREE1J.ZIP
  ```

  to unpack the "exam kit" of choice.

From here on things should be familiar. You could, for example, log onto the `D:` or `J:` drive, use `UltraEdit` use `CMAKE` and `CRUN` … generally have hours of fun. The C# system will work only on the `D:` drive, however.

But note that the exam set up has *no* connection with the outside world - no ftp client, telnet client, shared directories - not even a printer!

Today you may use the files and either the "usual" or the "exam" 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 any files from the D: drive, 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: directory, for example to `C:\TEMP`

(e) If you take the exam kit to a private machine you will need to have Java installed (or the .NET framework or equivalent to use the C# version).

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. You have plenty of time in which to prepare and test your ideas - go for it, and good luck. **Remember that tomorrow you may not bring anything into the room other than your student card and writing utensils, and especially not listings, diskettes, memory sticks, text books or cell phones.**

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 Monday

Before the start of the formal examinations 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.

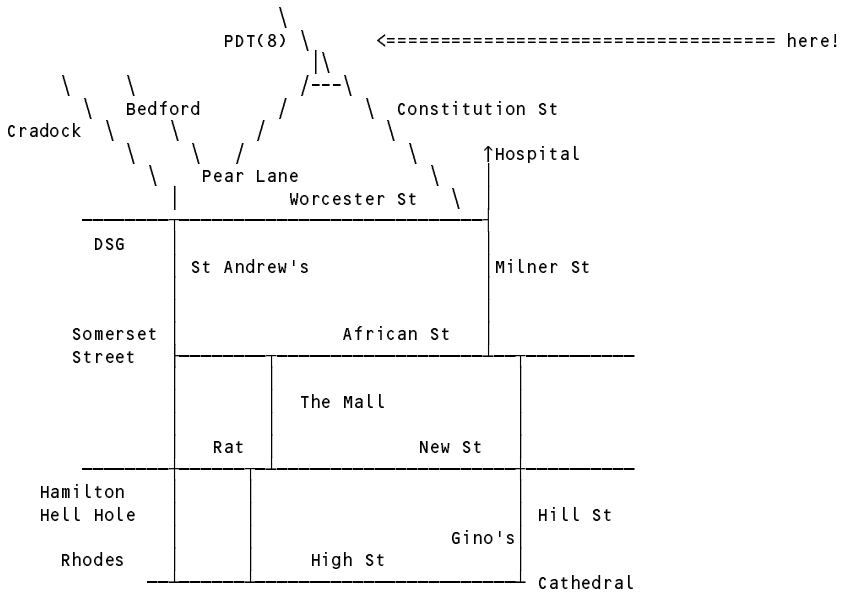
- You will receive a copy of at least the relevant portions of the Parva grammar and support files that you received at 16h00. *You may annotate these during the exam to form part of your solution if you wish to submit a hand-written answer to Section B and need to make reference to the code (possibly by line number)*. 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*.

- 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: drive back to the server. This will be explained tomorrow.

Cessation of Hostilities Party

As previously mentioned, Sally and I would like to invite you to an informal end-of-course party at our house on 16 November. There's another copy of the map below to help you find your way there. It would help if you could let me know whether you are coming so that we can borrow enough glasses, plates etc. *Please post the reply slip into the hand-in box during the course of the day*. Time: from 18h30 onwards. Dress: Casual



Section B [80 marks]

Free information and initial problem to be solved:

It was in 1988, at the second of a series of conferences held by a group charged with developing a rigorous standard for the definition of Modula-2, that one Susan Eisenbach made a telling observation: "Modula-2 would be the perfect language if we could add just one more feature. The difficulty is that nobody can decide what that one feature should be". She was right. Language designers cannot resist the temptation to add extension after extension to their brainchild.

So it is with Parva.

The Parva language that you have got to know so well currently has three basic data types - `integer`, `boolean` and `character`. Although it makes provision for the use of string constants in `read` and `write`

statement lists:

```
void main() { $C+  // eg00.pav
  int i;
  read("Supply a value for i" , i);
  write("The value you supplied was ", i);
}
```

it does not provide for the declaration and use of `string` variables:

```
void main() { $C+  // eg02.pav
  string yourName, myName;
  myName = "Pat";
  read("What is your name? ", yourName);
  write(myName, " is pleased to meet ", yourName);
}
```

As the first step in this examination you are invited to extend a version of the Parva compiler to incorporate a `string` type.

Implementations of languages that support a `string` type usually come with a complete support library of useful functions. Within the time constraints of the examination it will not be possible to provide a full implementation of the `string` type, but your system should be able to do the following at least:

(a) Declare variables and arrays of type `string`
(b) Assign string literals to such variables, and values of string variables to other string variables
(c) Read and write values for strings
(d) Compare two strings for equality or inequality
(e) Provide a function for converting a string to UPPERCASE
(f) Perform any necessary semantic and constraint checking on strings.

Here is a variation on the program given earlier, showing the use of some of these features:

```
void main() { $C+  // eg06.pav
  string yourName, myName, theBoff;
  myName = "Pat";
  theBoff = myName;
  read("What is your name? ", yourName);
  write(myName, " is pleased to meet ", yourName);
  if (myName != yourName)
    writeLine(" (Our names are different)");
  if (upper(theBoff) == upper(yourName))
    writeLine("(Our uppercased names are the same)");
}
```

Hints:

(a) The examination kit includes all the files needed to build a working Parva compiler similar to the one used in your last practical exercise (it does not have all the extensions you were asked to make in that practical, and you need not bother to try to add all of those extensions again). As usual, there is a Java version and an equivalent C# version. You may use either version.

(b) It also includes various simple test programs of the sort outlined above (in files named `egXX.pav`)

(c) The original Parva compiler stores string constants character by character in high memory, as you should remember. However, it is suggested that you do not try to extend that idea. Rather create a "string pool" dynamically, using the `ArrayList` class. Store string values in this pool, and use the index into the pool as the "value" of a string. A synopsis of the `ArrayList` class is given below.

(d) Later in the day - at 16h00 - we shall release more information, to help those of you who may not have completed the exercise to do so. Section B of the examination tomorrow will include a set of unseen questions probing your understanding of the system.

(e) Rest assured that you will not be expected to reproduce a complete Parva compiler from memory under examination conditions, but you may be asked to make some additions or improvements to the system developed today. You will not be asked to make the exact extensions of the last practical again.

(f) Remember Einstein's Advice: "Keep it as simple as you can but no simpler" and Terry's Corollary: "For every apparently complex programming problem there is an elegant solution waiting to be discovered".

Simple list handling in Java

The following is the specification of useful members of a Java generic list handling class useful in developing translators as discussed in this course.

```
import java.util.*;

class ArrayList
// Class for constructing a list of objects

  public ArrayList <type> ()
  // Empty list constructor

  public void add(Object o)
  // Appends o to end of list

  public void add(int index, Object o)
  // Inserts o at position index

  public Object get(int index)
  // Retrieves an object from position index

  public Object set(int index, Object o)
  // Stores an object o at position index

  public void clear()
  // Clears all elements from list

  public int size()
  // Returns number of elements in list

  public boolean isEmpty()
  // Returns true if list is empty

  public boolean contains(Object o)
  // Returns true if o is in the list

  public boolean indexOf(Object o)
  // Returns position of o in the list

  public Object remove(int index)
  // Removes the object at position index

} // ArrayList
```

Here is a simple Java program using this class:

```
import java.util.*;
import Library.*;

class ListDemo {

  public static void main(String[] args) {
    ArrayList<String> strList = new ArrayList<String> ();
    for (int i = 0; i < 10; i++)
      strList.add( IO.readWord() );
    IO.writeLine(strList.size() + " items in the list");
    for (int j = strList.size() - 1; j >= 0; j--) {
      IO.write(strList.get(j).toUpperCase() );
      if ( strList.get(j).equals("Pat") ) IO.write(" the rotter");
      IO.writeLine();
    }
  }
}
```

Simple list handling in C#

The following is the specification of useful members of a C# list handling class.

```
using System.Collections.Generic;

class List  // Note the change of name from ArrayList in the non-generic form
// Class for constructing a list of objects

  public List <type>()
  // Empty list constructor

  public int Add(object o)
  // Appends o to end of list

  public object this [int index] {set; get; }
  // Inserts or retrieves an object in position index
  // list[index] = object;  object = list[index]

  public void Clear()
  // Clears all elements from list

  public int Count { get; }
  // Returns number of elements in list

  public boolean Contains(object o)
  // Returns true if o is in the list

  public boolean IndexOf(object o)
  // Returns position of o in the list

  public void Remove(object o)
  // Removes object o from list

  public void RemoveAt(int index)
  // Removes the object at position index

} // ArrayList
```

Here is the equivalent simple C# program to the one given above:

```
using System.Collections.Generic;
using Library;

class ListDemo {

  public static void Main(string[] args) {
    List<string>  strList = new List<string> ();
    for (int i = 0; i < 10; i++)
      strList.Add( IO.ReadWord() );
    IO.WriteLine(strList.Count + " items in the list");
    for (int j = strList.Count - 1; j >= 0; j--) {
      IO.Write( (strList[j]).ToUpper() );
      if (strList[j].Equals("Pat") ) IO.Write(" the rotter");
      IO.WriteLine();
    }
  }
}
```