RHODES UNIVERSITY

Computer Science 301 - 2017 - 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!

This year the format of the compiler 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 any 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 complete working system in the examination (in fact, trying to do 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 complete working solutions 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.

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 you should have a good idea of what is involved.

Today you will be able to find the following files in the usual places:

- All the files as were made available for the practicals, tests, solutions, past papers.
- The file `FREE1.ZIP`, which contains 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 off course.

So today things should be familiar. You could, for example, log onto the `D:` or `J:` drive, use `NotePad++` and `LPRINT` to edit and print out files, use `CMAKE` to build Parva … 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 have prepared properly for the exam you should have plenty of time in which to implement 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 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 a machine readable examination paper in MS Word and ASCII formats.
- You will receive an examination paper, with spaced questions, and you are encouraged to write your answers on the exam paper itself, or sedit the MS-Word file.
- 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 any case you must hand back all listings with your exam paper.
- *There is no obligation to use a computer during the exam. You can answer on the examination 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 the supplementary examination

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.

Currently, the Parva language that you have got to know so well 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).

(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 heap" dynamically, using the `List` class. Store string values in this heap, and use the index into the heap as the "value" of a string variable or constant.

(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 C#

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

```
using System.Collections.Generic;

class List<E>
// Class for constructing a list of elements of type E

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

  public void Add(E element)
  // Appends element to end of list

  public void Insert(int index, E element)
  // Inserts element at position index

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

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

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

  public boolean Contains(E element)
  // Returns true if element is in the list

  public boolean IndexOf(E element)
  // Returns position of element in the list

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

  public void Remove(E element)
  // Removes element from list

} // List
```

Here is a simple C# program using this class

```
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();
    }
  } // Main
} // ListDemo
```

Simple string and character handling in C#

The following rather meaningless program illustrates various of the string and character manipulation methods that are available in C# and are useful in developing translators.

```
using System.Text;   // for StringBuilder
using System;        // for Char

class demo {
  public static void Main(string[] args) {
    char c, c1, c2;
    bool b, b1, b2;
    string s, s1, s2;
    int i, i1, i2;

    b = Char.IsLetter(c);                    // true if letter
    b = Char.IsDigit(c);                     // true if digit
    b = Char.IsLetterOrDigit(c);             // true if letter or digit
    b = Char.IsWhiteSpace(c);                // true if white space
    b = Char.IsLower(c);                     // true if lowercase
    b = Char.IsUpper(c);                     // true if uppercase
    c = Char.ToLower(c);                     // equivalent lowercase
    c = Char.ToUpper(c);                     // equivalent uppercase
    s = c.ToString();                        // convert to string
    i = s.Length;                            // length of string
    s = s.Trim();                            // remove leading/trailing whitespace
    s = s.ToUpper();                         // equivalent uppercase string
    s = s.ToLower();                         // equivalent lowercase string
    s = String.Concat(s1, s2);               // s1 + s2
    s = s.Substring(i1);                     // substring starting at s[i1]
    s = s.Substring(i1, i2);                 // substring s[i1 ... i1+i2-1] (i2 is length)
    s = s.Remove(i1, i2);                    // remove i2 chars from s[i1]
    s = s.Replace(c1, c2);                   // replace all c1 by c2
    s = s.Replace(s1, s2);                   // replace all s1 by s2
    c = s[i];                                // extract i-th character of s
//    s[i] = c;                              // not allowed
    b = s.Equals(s1);                        // true if s == s1
    b = String.Equals(s1, s2);               // true if s1 == s2
    i = String.Compare(s1, s2);              // i = -1, 0, 1 if s1 < = > s2
    i = String.Compare(s1, s2, true);        // i = -1, 0, 1 if s1 < = > s2, ignoring case
    i = s.IndexOf(c);                        // position of c in s[0 ...
    i = s.IndexOf(c, i1);                    // position of c in s[i1 ...
    i = s.IndexOf(s1);                       // position of s1 in s[0 ...
    i = s.IndexOf(s1, i1);                   // position of s1 in s[i1 ...
    i = s.LastIndexOf(c);                    // last position of c in s
    i = s.LastIndexOf(c, i1);                // last position of c in s, <= i1
    i = s.LastIndexOf(s1);                   // last position of s1 in s
    i = s.LastIndexOf(s1, i1);               // last position of s1 in s, <= i1
    i = Convert.ToInt32(s);                  // convert string to integer
    i = Convert.ToInt32(s, i1);              // convert string to integer, base i1
    s = Convert.ToString(i);                 // convert integer to string

    StringBuilder                            // build strings
      sb = new StringBuilder(),              //
      sb1 = new StringBuilder("original");   //
    sb.Append(c);                            // append c to end of sb
    sb.Append(s);                            // append s to end of sb
    sb.Insert(i, c);                         // insert c in position i
    sb.Insert(i, s);                         // insert s in position i
    b = sb.Equals(sb1);                      // true if sb == sb1
    i = sb.Length;                           // length of sb
    sb.Remove(i1, i2);                       // remove i2 chars from sb[i1]
    sb.Replace(c1, c2);                      // replace all c1 by c2
    sb.Replace(s1, s2);                      // replace all s1 by s2
    s = sb.ToString();                       // convert sb to real string
    c = sb[i];                               // extract sb[i]
    sb[i] = c;                               // sb[i] = c

  } // Main
} // demo
```