RHODES UNIVERSITY

Computer Science 301 - 2003 - 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 is probably easier than you might at first think.
- The solution to section B does not need an enormous amount of code. But it needs 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 10h30 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 10h30 pm Jody 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 how to solve the problem below. During this time 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 `EXAM.ZIP` which contains the Java version of the Coco/R system, and its support files, and the complete sources for a solution to Practical 25.

Most of the machines in the labs will work in exactly the way they have done this semester. A few are set up to show you the configuration you can expect to find 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 `START -> Command` route as usual
- Give the command

  ```
  CONNECT  TESTxx   demo
  ```

  where `xx` is the two digit number for the machine (typically in the range `01` through `25`).
- This will then allow you to log onto the `L:` drive, where you will find the file `EXAM.ZIP` waiting for you.
- Give the command

  ```
  UNZIP   EXAM.ZIP
  ```

  to unpack the "exam kit".

From here on things should be familiar. You could, for example, log onto the `D:` or `L:` drive, use `UltraEdit` run `CMAKE Parva` … generally have hours of fun.

But note that the exam set up has *no* connection with the outside world - no web browser, 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 encouraged 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) For technical reasons the executable file `PATPARVA.EXE` has been placed in the normal path. There is a copy of this program under the name `PDTPARVA.EXE` in the exam kit, but this cannot be run off a network drive (`L:`). You can run it off a local drive (`D:`), so if you wish to take the exam kit to a private machine you should be able to execute it easily.

(f) If you take the exam kit to a private machine you will need to have Java installed, and also the .NET framework.

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 really good solutions - go for it, and good luck. Remember that tomorrow you may not bring anything into the room other than your student card, writing utensils, and especially not papers, diskettes, 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.

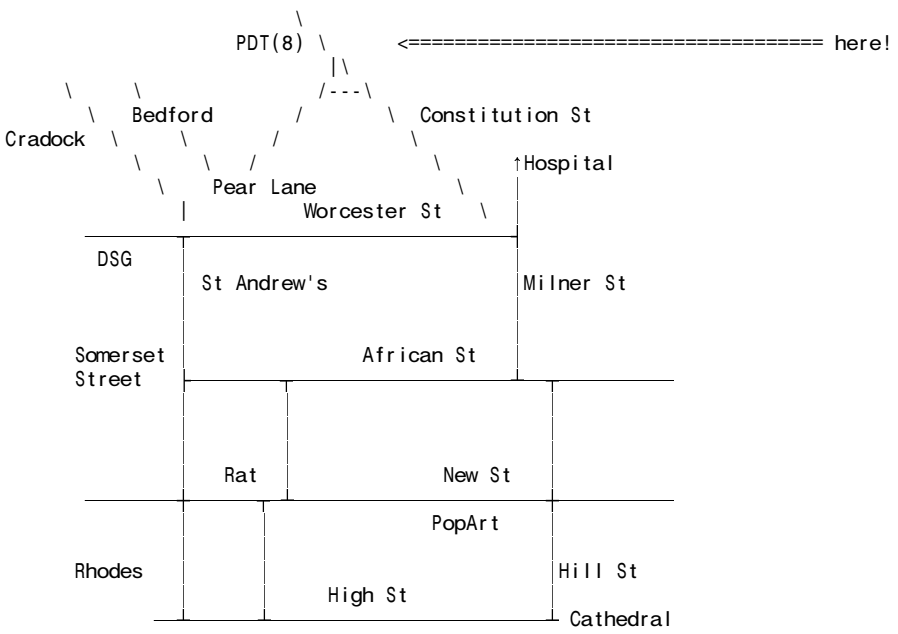
At the start of the examination session:

- You will receive ordinary answer books for the preparation of answers, and an examination paper.
- You will receive a copy of the complete listing of the Parva compiler that accompanies this handout. *You may annotate this 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 listing 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 `L:` 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 10 November. There's a 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. E-mail to p.terry@ru.ac.za. Time: from 18h30 onwards. Dress: Casual



And now for my last trick - taken exactly from tomorrow's examination paper:

Section B [80 marks]

Please note that there is no obligation to produce a machine readable solution for this section. Coco/R and other files are provided so that you can enhance, refine, or test your solution if you desire. If you choose to produce a machine readable solution, you should create a working directory, unpack EXAM.ZIP, modify any files that you like, and then copy all the files back onto an exam folder on the network.

A minor crisis has developed in the Computer Science department. This year, as an experiment, the first year students have been taught programming using the influential Parva language. This has proved eminently successful, and the students are due to write a practical examination on Monday. Today is Sunday. Earlier this morning the lecturer in charge of the class discovered, to her horror, that the existing Parva compiler makes provision only for integer and boolean data types, and not for the character type that would be required if students were asked to write code like the following:

```
void main () {
// Read a sentence and write it backwards
  char[] sentence = new char[1000];
  int i = 0;
  char ch;
  read(ch);
  while (ch != '.') {  // input loop
    sentence[i] = ch;
    i = i + 1;
    read(ch);
  }
  while (i > 0) {      // output loop
    i = i - 1;
    write(sentence[i]);
  }
}
```

or similar applications which require simple character manipulations, including the ability to mix integers and characters appropriately in expressions, detect type incompatibilities, cast between integers and characters when required, increment and decrement character variables and so on.

She had intended to set a simple problem in which the students were required to print a set of multiplication tables. When testing the solution, she made a typing error and submitted the following code to the compiler, with rather unexpected results:

```
void main () {
// Print a set of multiplication tables
  int i, j;
  for i = 1 to 10 do { // <----------------- should be j not i
    for i = 1 to 10 do write (i * j, "\t");
    write("\n");
  }
}
```

But worse is still to come! The lecturer has also been under the illusion that the key word *const* is used in the sense that it is used in C# and the word *final* is used in Java, namely as a modifier in a variable declaration that indicates that when a variable is declared it can be given a value which can then not be modified later. On checking the grammar for Parva she comes across the productions

```
Statement         = Block | ConstDeclarations | VarDeclarations | ...
ConstDeclarations = "const" OneConst { "," OneConst } ";" .
OneConst          = identifier "=" Constant .
Constant          = number | charLit | "true" | "false" | "null" .
VarDeclarations   = Type OneVar { "," OneVar } ";" .
OneVar            = identifier [ "=" Expression ] .
```

which she realizes will not correctly handle code of the form she needs, like

```
const int max = 2;
int i = 5;
const int iPlusMax = i + max;
```

In desperation she turns to the members of the third year class, imploring them to come up with a reliable new version of the Parva compiler in 24 hours that will provide the character type, treat the keyword *const* as she intends, and detect situations where attempts are made to alter a for loop control within the loop body. She offers to provide them with a few examples of the sort of code that she expects the compiler to be able to handle. For example, besides the example involving character types, she provides a commented test program:

```
void main () {
// Rather meaningless program to illustrate variable declarations
// that incorporate a const modifier
  const int i = 10;                 // acceptable
  const char terminator;            // unacceptable
  const int[] list = new int[i];    // acceptable
  int j = i;
  while (j < 100) {
    const int k = j;                // acceptable
    read(i);                        // unacceptable
    i = j;                          // unacceptable
    list[4] = 5;                    // acceptable
    j = j + list[4];                // acceptable
    list = new int[12];             // unacceptable
  }
}
```

Rise to the challenge! Produce a new compiler, and avoid a disaster for the department.

Note:

(a) In the exam kit (`EXAM.ZIP`) you will find the executables and support files for Coco/R, as used in the practical course. You will also find the complete attribute grammar and support files for the Parva language as developed by the end of your lecture and practical course.

(b) In the exam kit will be found some other example programs like those above to assist in your understanding of the requirements and your development of the system, and an executable derived from a model solution that you are free to use to compile these example programs or any others you may devise.

(c) Depending on your approach, your solution may require modifications to any or all of the grammar and support files in the exam kit.

(d) It is not particularly difficult to provide "first approximations" to these extensions and modifications. The examiners will, however, be looking for maturity in your solution and, in particular for signs that you have seen past the obvious "quick fix".

Summary of useful library classes

```
class SymSet { // simple set handling routines
  public SymSet()
  public SymSet(int[] members)
  public boolean equals(Symset s)
  public void incl(int i)
  public void excl(int i)
  public boolean contains(int i)
  public boolean isEmpty()
  public int members()
  public SymSet union(SymSet s)
  public SymSet intersection(SymSet s)
  public SymSet difference(SymSet s)
  public SymSet symDiff(SymSet s)
  public void write()
  public String toString()
} // SymSet

public class OutFile { // text file output
  public static OutFile StdOut
  public static OutFile StdErr
  public OutFile()
  public OutFile(String fileName)
  public boolean openError()
  public void write(String s)
  public void write(Object o)
  public void write(int o)
  public void write(long o)
  public void write(boolean o)
  public void write(float o)
  public void write(double o)
  public void write(char o)
  public void writeLine()
  public void writeLine(String s)
  public void writeLine(Object o)
  public void writeLine(int o)
  public void writeLine(long o)
  public void writeLine(boolean o)
  public void writeLine(float o)
  public void writeLine(double o)
  public void writeLine(char o)
  public void write(String o,  int width)
  public void write(Object o,  int width)
  public void write(int o,     int width)
  public void write(long o,    int width)
  public void write(boolean o, int width)
  public void write(float o,   int width)
  public void write(double o,  int width)
  public void write(char o,    int width)
  public void writeLine(String o,  int width)
  public void writeLine(Object o,  int width)
  public void writeLine(int o,     int width)
  public void writeLine(long o,    int width)
  public void writeLine(boolean o, int width)
  public void writeLine(float o,   int width)
  public void writeLine(double o,  int width)
  public void writeLine(char o,    int width)
  public void close()
} // OutFile

public class InFile {   // text file input
  public static InFile StdIn
  public InFile()
  public InFile(String fileName)
  public boolean openError()
  public int errorCount()
  public static boolean done()
  public void showErrors()
  public void hideErrors()
  public boolean eof()
  public boolean eol()
  public boolean error()
  public boolean noMoreData()
  public char readChar()
  public void readAgain()
  public void skipSpaces()
  public void readLn()
  public String readString()
  public String readString(int max)
```

```
    public String readLine()
    public String readWord()
    public int readInt()
    public long readLong()
    public int readShort()
    public float readFloat()
    public double readDouble()
    public boolean readBool()
    public void close()
  } // InFile
```

Strings and Characters in Java

The following rather meaningless program illustrates various of the string and character manipulation methods that are available in Java and which will be found to be useful in developing translators.

```
import java.util.*;

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

    b = Character.isLetter(c);             // true if letter
    b = Character.isDigit(c);              // true if digit
    b = Character.isLetterOrDigit(c);      // true if letter or digit
    b = Character.isWhitespace(c);         // true if white space
    b = Character.isLowerCase(c);          // true if lowercase
    b = Character.isUpperCase(c);          // true if uppercase
    c = Character.toLowerCase(c);          // equivalent lowercase
    c = Character.toUpperCase(c);          // equivalent uppercase
    s = Character.toString(c);             // convert to string
    i = s.length();                        // length of string
    b = s.equals(s1);                      // true if s == s1
    b = s.equalsIgnoreCase(s1);            // true if s == s1, case irrelevant
    i = s1.compareTo(s2);                  // i = -1, 0, 1 if s1 < = > s2
    s = s.trim();                          // remove leading/trailing whitespace
    s = s.toUpperCase();                   // equivalent uppercase string
    s = s.toLowerCase();                   // equivalent lowercase string
    char[] ca = s.toCharArray();           // create character array
    s = s1.concat(s2);                     // s1 + s2
    s = s.substring(i1);                   // substring starting at s[i1]
    s = s.substring(i1, i2);               // substring s[i1 ... i2]
    s = s.replace(c1, c2);                 // replace all c1 by c2
    c = s.charAt(i);                       // extract i-th character of s
//    s[i] = c;                            // not allowed
    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 = Integer.parseInt(s);               // convert string to integer
    i = Integer.parseInt(s, i1);           // convert string to integer, base i1
    s = Integer.toString(i);               // convert integer to string

    StringBuffer                           // build strings
      sb = new StringBuffer(),             //
      sb1 = new StringBuffer("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
    i = sb.indexOf(s1);                    // position of s1 in sb
    sb.delete(i1, i2);                     // remove sb[i1 .. i2]
    sb.replace(i1, i2, s1);                // replace sb[i1 .. i2] by s1
    s = sb.toString();                     // convert sb to real string
    c = sb.charAt(i);                      // extract sb[i]
    sb.setCharAt(i, c);                    // sb[i] = c
  }
}
```