RHODES UNIVERSITY

Computer Science 301 - 2013 - 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 Terrifying Tuesday

From now until about 22h30 tonight, Computer Science 3 students have exclusive use of one 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 Chris and I 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.

Once again, 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 recent years (since 2006), the problem set below is only part of the story. At 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 files `FREE1J.ZIP` and `FREE1C.ZIP`, which contain the Java and 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`).

At about 16h30 new versions of the exam kit will be posted (`FREE2J.ZIP` and `FREE2C.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 printout files, use `CMAKE` and `CRUN` … 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 Java installed, or have 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.

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 Wonderful Wednesday

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.

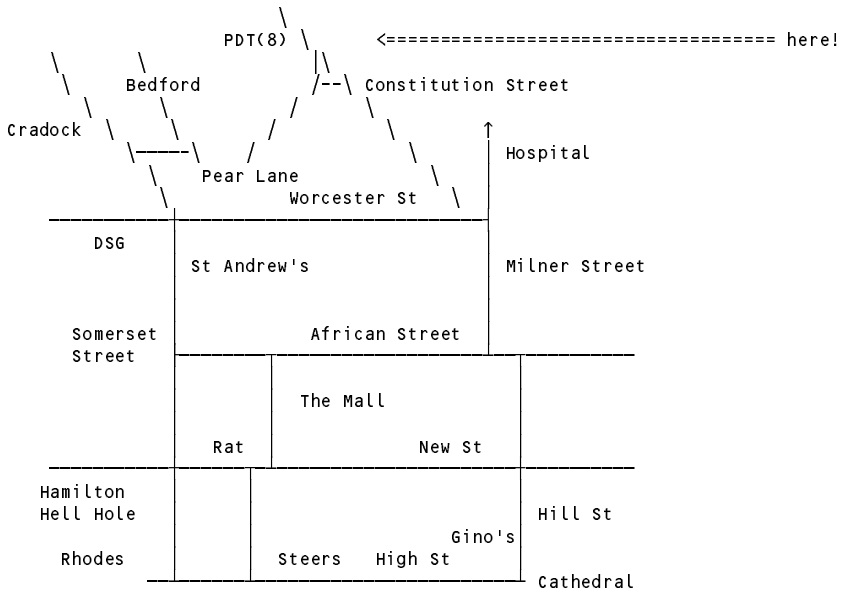
At the start of the examination session:

- You will receive ordinary answer books for the preparation of answers, and an examination paper.
- You will receive listings of various 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: 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.**

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 Wednesday evening (after the Compilers paper). There's a magnificent ASCII-art 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



(See also `http://www.cs.ru.ac.za/courses/CSc301/Translators/map.htm`)

Preliminary to the examination

As you may have noticed, there are currently some 5000+ students writing examinations at this University. As you can imagine, some considerable effort is required of the academic staff to assess the some 25000+ examination scripts that will be generated by these students - something that has to be done under extreme pressure to enable the results to be released in time to spoil the holidays of some students and delight the parents of some others. What you may not know is that a crucial part of this process - and one that has to be accomplished over a period of only two or three days immediately prior to the release of results - is an analysis by the Deans of the Faculties of the collated results, so that students who are to be excluded, congratulated, awarded degrees, put on probation and so on can be identified and their results annotated with these decisions.

Over the years Yours Truly has been instrumental in designing computer programs to help with this "Dean's Analysis". For some years this has taken the preparation of a huge text file of summaries for each student, on the lines of that illustrated below. These files can run to over 800 pages of closely printed small text.

Fayled-Badleigh, HE 10L9876 BSCS 3 (3) 39.0 Pts 14 s/crs (Fails G7)

```
   INF 301 1 4207301 Information Systems 301           2B  61
   INF 302 2 4207302 Information Systems 302           2B  65
   CSC 201 1 5101201 Computer Science 201              F2  38
   CSC 202 2 5101202 Computer Science 202              DPR
   MAT 1C1 1 540101A Mathematics 1C1                   F1  48      F1 or F1S ?
                  Passed 2/5 courses outright.  Average mark  42.00  Weighted  42.0
   +++++++++ OVER HALF THE COURSES FAILED +++++
                  Full firsts 0, Full upper seconds 0, Full credits 1, Full F1 0, AEG 0, Supps 0, DPR 1
   +++++++++ On probation - READMIT AP: MUST COMPLETE BSCS 2 IN 2012

   Has ACC 101 2A  ACC 102 3   CSC 101 3   CSC 102 3   ECO 1   ACR ECO 101 3
   Has INF 2   ACR INF 202 2A  MAN 101 2B  MAN 102 2B  PHY 1E2 3   STA 1D  3

     1 Acc  ** CSc  ** Eco  **         Man  ** St1D  * PhyE  *  (10)
     2                        Inf  **                          ( 2)
     3                        Inf  **                          ( 2)
```

More recently the program that produces these summaries has also tried to assess or predict a *set of actions* for each student that the Deans will recommend to their Faculty Boards for approval. In principle this is not too difficult - after all, students who are deserving of congratulations are easily identified, as are those who, sadly, have failed too many courses to be allowed to return.

In practice, one has a maintenance nightmare. Hard-coding the criteria that apply (differently) to each degree, each year of study, each faculty, into a program that now runs to many thousand lines of C# code, demands the attention of a grey haired retired programmer, who may not always be available for last minute, dangerous hacks to the code. So this ASP (Ancient Skilled Programmer) has recommended that the Deans be asked to draw up a specification of the actions that should be applied to the students in their Faculty, on the lines of the following example (found in the file `criteria.1` - more examples are to be found in your "kit"):

```
Humanities
  FirstYear :
    Exclude : Total_Credits < 2 and (Average < 40.0 or DPR >=2);
              Lowest_Mark < 10.
    Merit_List :
              Average > 75 and Lowest_Mark > 60 and Year_Firsts >= 2 and Fails = 0.
  ThirdYear :
    AP_Complete_In_One :
              Total_Credits < 10.0 and Years_Here > 3.
  AnyYear :
    Congratulate :
              Weighted_Average > 90.0 and Fails = 0.
```

This specification can act as input to a converter program that will read it and construct a C# method (or an equivalent Java version, which would be fairly similar) on the lines of the following:

```
class ActionSetBuilder {

  public static void BuildSet(Student s, OutFile logFile) {
  // Build up a set of numeric tags for Student s based on an analysis of his or her record
  // logFile may be used to record problems, results, anomalies (none demonstrated here).
    s.actions = new IntSet();
    if (s.academicYear == 1) {
      if (s.fullTotalCredits < 2 && (s.rawAverage < 40.0 || s.DPRthisYear >= 2))
        s.actions.Incl(10);
      if (s.lowestMark < 10)
        s.actions.Incl(10);
      if (s.rawAverage > 75 && s.lowestMark > 60 && s.yearFirstsThisYear >= 2 && s.failsThisYear == 0)
        s.actions.Incl(16);
    }
    if (s.academicYear == 3) {
      if (s.fullTotalCredits < 10.0 && s.yearsOfStudy > 3)
        s.actions.Incl(21);
    }
    {
      if (s.weightedAverage > 90.0 && s.failsThisYear == 0)
        s.actions.Incl(11);
    }

  } // ActionSetBuilder.BuildSet

} // Class ActionSetBuilder
```

The method thus generated can then easily be compiled along with the rest of the source of the RAP (Result Analyser Program). If the criteria are augmented, or changed in some other way, then the revised specification file can easily be "recompiled" into C# code and then the complete RAP recompiled with the new method. Note that the Deans who design the criteria specifications do not have to be C# or Java programmers - they simply have to be able to write specifications in a simple, though rigid form, using identifiers that they would find published in a guide, with examples like the one just given. These identifiers, as can be seen from the example, correspond closely to fields in an object `s` of the `Student` class that will be populated from the university database as the the RAP is executed, passing each in turn to `ActionSetBuilder.BuildSet(s, logFile)`.

The ASP can't complete this on his own in the time available. Only some 24 hours remain until the deadline, and so he has made you an Offer You Can't Refuse to join the team to help him.

It is suggested that, as an MYSP (Much Younger Skilled Programmer), you proceed as follows:

- 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"!

- Develop a Cocol grammar `Converter.atg` that can describe the specification files containing the lists of criteria for a faculty. Refine the skeleton in the kit. You might even try it out in "syntax only" mode.

- Refine the `Table` class in the kit to give one that will allow you to read and construct a table of the identifiers that may be used in writing a specification, as well as provide you with a method for returning a table `Entry` based on an identifier's name as a search key. For ease of later modifications, the system uses the identifiers in this table, rather than hard coded "key words". However, the concept of a student's academic year is key to the analysis, so regard the words `FirstYear`, `SecondYear`, `ThirdYear`, `FourthYear` and `AnyYear` as keywords.

- Develop the `Converter.atg` grammar to incorporate semantic checking - for example checking that the identifiers used in the criteria specifications are contextually of the correct kind and type.

- When you have done this satisfactorily you should be able to parse correct specification files (and also reject incorrect ones).

- When you are happy with that, press on to develop the actions in the grammar to allow for the generation of the C# (or Java) code that will define the class `ActionSetBuilder` and the crucial method `BuildSet`, which can then be included in a simple supplied RAP that can analyse a supplied data file of student results.

- **Keep it as simple as you can, but no simpler!** - there is enough to do without getting over fanciful. In particular do not spend a lot of effort in "pretty printing" the `BuildSet` method - just ensure that the `ActionSetBuilder` class that you construct can be correctly compiled by the C# (or Java) compiler when it is integrated into the final system. code.

In adopting this approach you can (and should) make use of the files provided in the examination kit `free1j.zip` (Java version) or `free1c.zip` (C# version) which you will find on the course website. In particular, you will find

- The usual frame files, and the Coco/R system;

- Some simple example criteria descriptions like the one above that you can use for testing purposes;

- The first part of the `Converter.atg` file, with suitable CHARACTERS and TOKENS sections.

- The files to build a Parva level 1 compiler using the Java/C# expression precedence hierarchy, most of which should be familiar, and from which you might be able to steal some ideas and even some code. **Just to be quite explicit, this project does NOT require you to generate or interpret PVM code so don't even think of it!**

- Text files `humStudents.txt`, `sciStudents.txt`, `comStudents.txt` containing some representative (but fictitious) students' records, which can be read by the `Student.ReadStudent()` method and analysed by your RAP. `allSStudent.txt` is a single file with all these students in it.

- The complete source of a `Student` class, which includes methods for reading a student record from an input file like those just mentioned and for printing a student record to an output file. You are free to add further methods and fields to this class, although it has been designed so that this should not be necessary. You probably should not be tempted to play with polymorphism and inheritance (keep it simple). This class has the following sort of interface (not all data fields are shown - you can see them all in the full sourcekit).

```
public class Student {

   public string  faculty;                // Faculty for this student
   public string  name;                   // 30 character student name and initials
   public string  studentNumber;          // 7 character "student number" eg 63TO844 (genuine!)
   ...

   // overall progress for this and previous years

   public double  fullPrevious;           // Full credit score for all previous years of study
   public double  fullCurrent;            // Full credit score for current year of study
   public double  fullTotalCredits;       // Total number of full credits earned towards the degree so far
```

```
    // for the current year of study

    public int     coursesThisYear;         // Number of courses for which student is registered
    public int     semCreditsThisYear;      // Semester credits earned during current year
    public int     yearCreditsThisYear;     // Full year credits earned during current year
    public int     passesThisYear;          // Number of courses passed this year
    public int     firstsThisYear;          // Number of courses passed this year at 75% and above
    public int     semFirstsThisYear;       // Number of semester courses passed at 75% and above
    public int     yearFirstsThisYear;      // Number of full courses passed at 75% and above
    public int     suppsThisYear;           // Number of courses failed, but supplementary exam recommended
    ...
    public IntSet  defined = new IntSet();  // A set of integers corresponding to actions that might be applied
                                            // to this student
    public IntSet  actions = new IntSet();  // A set of integers corresponding to actions that are recommended
                                            // to the Dean when RAP deals with this student
    public string  actionString;            // Summary of recommendation actions as a string

    public Student () {
    // Empty constructor for testbed applications
    ...
    } // Student Constructor ()

    public static void ReadStudent (InFile achievementsFile, Student s) {
    // Developed for testbed applications - read the details of a single student s from the achievementsFile.
    // Each record occupies a single (long) line of text.  In a real application the student record would,
    // of course, be constructed after processing a much bigger file of raw results.
    //
    // This method will allow you to fake the analysis of a representative set of student results, one at
    // a time.   Some sample "achievements" files are supplied in the exam kit.
    //
    ...
    } // Student.ReadStudent(InFile, s)

    public static void WriteStudent(OutFile output, Student s) {
    // Reflect the data for student s on an output text file - for testing applications
    ...
    } // Student.WriteStudent

    public static void WriteLabelledStudent(OutFile output, Student s) {
    // Reflect the data for student s on an output text file - for testing applications
    // (labelled for ease of lookup).
    ...
    } // Student.WriteLabelledStudent

  } // class Student
```

- The complete source code for a simple RAP (except for the `ActionSetBuilder` class, of course).

- The skeleton source code for a `Table` class, with the following structure, which you will have to develop.

```
    public class Types {
    // Identifier (and expression) types.
    // The usual named "magic numbers" approach.
      public const int
        noType     =  0,
        intType    =  1,
        boolType   =  2,
        doubleType =  3,
        stringType =  4;
    } // end Types

    public class Kinds {
    // Identifier kinds
      public const int
        noKind     = 0,
        faculty    = 1,
        degree     = 2,
        field      = 3,
        action     = 4;
    } // end Kinds

    public class Entry {
    // Entries in the symbol table are objects of this class.
    // Fields are all declared public for simplicity.
    //
      public bool    declared;          // true for all except sentinel entry
      public string  name;
      public string  codedName = "";
      public int     type = Types.noType;
```

```
  public int     kind = Kinds.noKind;

  // Not all of the following are relavant to each kind of entry, but rather
  // than spawn a whole lot of subtypes, we simply introduce all of them.

  public int     minYears     = 0;
  public double  minScore     = 10.0;
  public int     actionNumber = 99;
} // end Entry

public class Table {
// The table might include a dummy sentinel entry to handle undeclared
// entries in a simple familiar way.

  public static void Insert(Entry entry) {
  // Adds entry to symbol table
     // ...
  } // Table.Insert

  public static Entry Find(string name) {
  // Searches table for an entry matching name.  If found then return that
  // entry; otherwise return the sentinel entry (marked as not declared).
    // ...
    return null;  // dummy to be replaced
    // ...
  } // Table.Find

  public static void InitTable(InFile tableFile) {
  // Clears table, sets up sentinel entry, and fills the rest of the table from tableFile
    Entry sentinelEntry = new Entry();
    sentinelEntry.name = "";
    sentinelEntry.type = Types.noType;
    sentinelEntry.kind = Kinds.field;
    sentinelEntry.declared = false;
    Insert(sentinelEntry);
    // ...
  } // Table.InitTable

} // end Table
```

- A text file `Table.txt`, from each line of which a single entry in the `Table` can be constructed. The `Table` class in the kit already includes a method for reading a line from this file and creating such an entry. An extract from the file is shown below (the full file is available in the source kit).

```
# Dictionary of identifiers used in the examination results analyser prototype
#
# P.D. Terry, Rhodes University, 2013
# First release  2013/09/30
# Latest release 2013/10/25
# Lines starting with # in column 1 are treated as comments
# Completely blank lines are not permitted
# Lines are fixed format
#
# Each of the identifiers is associated with a Kind - one of
#
#   (Faculty, Degree, Field, Action)
#
# and has an associated Type (just as identifiers in most programming
# languages are of kinds like
#
#   (Constant, Variable, Function, Class)
#
# and have a type as one of (int, bool, char, double, string)).
#
# Firstly, we have a list of Faculty names
# (Width in parentheses, _ denotes a single space separator)
#
# Kind (14)  _Name (28)                 _Type (9) _
Faculty       Humanities                 string    H
Faculty       Science                    string    S
  ...
# Next we have a list of degree codes.  Each degree code is followed by
# the minimum number of years it should take to get the degree, and the
# score of credits that need to be earned, measured in "full credits".
# In many cases completing a degree is more loosely defined in terms of a
# score of "semester credits" that must be obtained.  Naturally one
# "full credit" for, say, ENG 3 contributes the equivalent of the two
# "semester credits" for, say, CSC 301 + CSC 302.
#
```

```
# Kind (14)  _Name (28)                 _Type (9) _Years  MinScore
#
Degree        BA                        string    3      10.0
Degree        BSS                       string    3      10.0
  ...
# Next comes a list of field names.  The first of these (in the
# second column) gives names that the Dean could use in criteria like
#
#          Ac_Yr = 1 and Passes < 2
#          Year_Firsts > 2 and Weighted_Average > 69.5
#
# which might be criteria for exclusion or congratulation, respectively.
#
# The third column gives the type of the field in the Student object and the fourth
# column gives the name of that field as it would be found in the corresponding C#
# or Java code for a Student object s.  For the examples above this code would be
#
#          s.academicYear == 1 && s.passesThisYear < 2
#          s.yearFirstsThisYear > 2 && s.weightedAverage > 69.5
#
# Kind (14)  _Name (28)                 _Type (9) _s.Field
#
Field         Ac_Yr                     int       s.academicYear
Field         Passes                    int       s.passesThisYear
Field         Year_Firsts               int       s.yearFirstsThisYear
Field         Weighted_Average          double    s.weightedAverage
  ...
# Finally we have a list of possible actions that the program should select
# for a student, based on conditions of the sort demonstrated previously.
# None of these is pre-selected.  The distinct numbers in the third column are
# to be included in a set of the actions applicable to a given student.
#
Action        Exclude                   bool      10
Action        Congratulate              bool      11
Action        Probation                 bool      12
  ...
Action        Weak                      bool      25
#
Action        INCONSISTENCY             bool      0
```

- There are some scripts to make development easy. Firstly,

  `CMAKE converter`

  will run Coco/R to create the `Converter, Scanner` and `Parser` sources and then invoke the C# or Java compiler to build the `Converter` program in the usual way. Secondly,

  `CONVERTER  criteriaFile` for example `CONVERTER  criteria.0`

  will run your converter program against the criteria file to create the `ActionSetBuilder` classs, the key component of which is the `BuildSet` method, which you should look at in an editor at least. Thirdly,

  `GoRAP  fac` for example `GoRAP  hum` or `GoRAP sci`

  will compile the test program `RAP` along with the `Library, ActionSetBuilder` and `Student` classes and then, if successful, run this program against the `facStudent.txt` to produce the summary of the suggested actions for affected students in the files `fac.summary.txt` and `fac.summary.log`, Lastly

  `FullTest Criteria.0  all`

  for example, will build the converter, compile it, build the RAP program and run it from scratch.

That should keep you busy for a few hours… In fact some of you may find that you RAP around the clock!

Have fun, and good luck!

(Although the actual system used by the Deans was written in C#, this whole exercise could all be done in Java or in C#, and you are free to experiment in either language).