RHODES UNIVERSITY

November Examinations - 2010

Computer Science 301 - Paper 2

Examiners:
Prof P.D. Terry
Prof D.G. Kourie

Time 3 hours
Marks 180
Pages 16 (please check!)

Answer all questions. Answers may be written in any medium except red ink.

A word of advice: The influential mathematician R.W. Hamming very aptly and succinctly professed that "the purpose of computing is insight, not numbers".

Several of the questions in this paper are designed to probe your insight - your depth of understanding of the important principles that you have studied in this course. If, as we hope, you have gained such insight, you should find that the answers to many questions take only a few lines of explanation. Please don't write long-winded answers - as Einstein put it "Keep it as simple as you can, but no simpler".

Good luck!

(For the benefit of future readers of this paper, various free information was made available to the students 24 hours before the formal examination. This included an augmented version of "Section D" - a request to extend the Parva compiler/interpreter system studied in the course to incorporate enumeration types. 16 hours before the examination a complete grammar for a working system and other support files for building this system were supplied to students, along with an appeal to study this in depth (summarized in "Section C"). During the examination, candidates were given machine executable versions of the Coco/R compiler generator, the files needed to build the basic system, access to a computer, and machine readable copies of the questions.)

Section A: Short questions [110 marks]

QUESTION A1 **[1 + 3 + 4 + 4 = 12 marks]**

(a) What is the essential difference between native-code compilation and interpretive compilation?

(b) In the practicals of this course you have used several compilers, in particular BCC and CL (C++ compilers), javac and jikes (Java compilers), FPC (Pascal), JPI (Modula-2), X2C, JAD, Parva and Coco. Which of these are interpretive compilers (+1 for each correct answer, -1 for each wrong answer)?

(c) Interpretive compilation suffers from the disadvantage that the object code runs more slowly than code produced by native code compilation. Why should this be, and what techniques are used by compiler writers to minimize the effect?

(d) If interpretive compilation suffers in this way, what makes it attractive to compiler writers? Suggest two situations where interpretive compilation can be put to very good use.

QUESTION A2 **[2 + 2 + 3 + 5 = 12 marks]**

In the practicals of this course you experimented with a decompiler named JAD.

(a) What distinguishes a "compiler" from a "decompiler"?

(b) Draw a T-diagram that captures the essence of the JAD decompiler.

(c) The developers of JAD may well have used Java in the development process. Given that they had a Java compiler like javac that could run on the JVM, draw T-diagrams illustrating how the final version of JAD (which you ran on the JVM) might have been produced.

(d) Using further T-diagrams, suggest and explain how the developers of JAD might have performed a self-consistency test of their product before releasing it to users.

A set of blank T-diagrams is provided in the free information, which you can complete and submit with your answer book.

QUESTION A3 **[1 + 8 + 2 + 1 = 12 marks]**

Scot MacHine has been set the task of developing a compiler for Canntaireachd. He wishes to allow the source code the possibility of incorporating comments of the sort familiar from the C family of languages:

```
/* comment text */
```

Since he planned to use Coco/R as a development tool he could simply have included the directive

```
COMMENTS FROM "/*" to "*/"
```

but his supervisor requested that he explore the use of the PRAGMAS facility instead. So Scot came up with the idea of writing

```
CHARACTERS
  inComment = ANY - "*" - "/" .
PRAGMAS
  Comment = "/*" { inComment } "*/" .
```

However, the regular expression forming part of this directive has some shortcomings.

(a) Here are some examples of strings that might be valid comments. One of them, however, cannot be a comment. Which is it?

```
(1)   /* comment */
(2)   /* multiply a * b */
(3)   /* divide a / b */
(4)   /* ***** */
(5)   /*********/
(6)   /* ** /* // */
(7)   /* // */ ** */
(8)   /*a*b*/
(9)   /* **a/b* ** */
```

(b) When it was pointed out to Scot that the text within a comment can validly include / and * characters, he came up with the refinement

```
CHARACTERS
  inComment = ANY - "*" - "/" .
PRAGMAS
  Comment = "/*" { inComment | "/" | "*" inComment } { "*" } "*/" .
```

Casual testing suggested to him that the problem was solved. However, this more complicated regular expression will still not handle some of the valid comments in (a). Identify two of these, and suggest how the regular expression might be improved still further.

(c) What practical advantage might there be in using the PRAGMAS facility of Coco/R rather than the COMMENTS facility when developing a compiler?

(d) In what applications is the Canntaireachd language useful?

QUESTION A4 **[8 + 6 + 4 = 18 marks]**

Formally, a grammar G is a quadruple $\{ N, T, S, P \}$ with the four components

(a) N - a finite set of **non-terminal** symbols,
(b) T - a finite set of **terminal** symbols,
(c) S - a special **goal** or **start** or **distinguished** symbol,
(d) P - a finite set of **production rules** or, simply, **productions**.

where a production relates to a pair of strings, say α and β, specifying how one may be transformed into the other:

$$\alpha \rightarrow \beta \text{ where } \alpha \in (N \cup T)^* N (N \cup T)^* \ , \ \beta \in (N \cup T)^*$$

and formally we can define a language $L(G)$ produced by a grammar G by the relation

$$L(G) = \{ \sigma \mid \sigma \in T^* ; S \Rightarrow^* \sigma \}$$

(a) In terms of this notation, express **precisely** (*that is to say, mathematically; we do not want a long essay or English description*) what you understand by

(a) A context-free grammar

(b) A sentential form

(c) FIRST(A) where $A \in N$

(d) FOLLOW(A) where $A \in N$

(b) What constraints must be imposed upon a context-free grammar if it is to be classified as an "LL(1) grammar"?

(c) The syntax of many programming languages can be accurately described by a context-free grammar, yet most programming languages incorporate context-sensitive features. Give an example of a context-sensitive feature of the Parva language studied in this course. How does the Parva compiler handle this feature correctly?

QUESTION A5 **[12 + 8 + 2 + 16 + 2 + 4 = 44 marks]**

The contents page of a very useful textbook (if you can find a copy) begins as follows:

```
All you need to know to be able to pass your compiler examination.

                            by

                         Pat Terry.

Chapter 1  Bribery is unlikely to succeed.

Chapter 2  Understand the phases of compilation.
    2.1    Lexical and syntactic analysis are easily confused
    2.2    Constraint analysis involves the concept of type
    2.3    Code generation for the PVM is a breeze

Chapter 3  Get clued up on grammars.
    3.1    Terminals
    3.2    Sentences and sentential forms
    3.3    Productions
    3.4    EBNF and Cocol
    3.5    Ambiguity is bad news
```

The following Cocol grammar attempts to describe this contents page (and others like it - there may be many chapters and many subsections, of course, and some components are optional):

```
COMPILER Contents
/* Describe the contents pages of a book
   P.D. Terry, Rhodes University, 2010 */

CHARACTERS
  uLetter = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" .
  lLetter = "abcdefghijklmnopqrstuvwxyz" .
  letter  = uLetter + lLetter .
  digit   = "0123456789" .

TOKENS
  word    = letter { letter } .
  number  = digit { digit } .
  section = digit { digit } "." digit { digit } .

IGNORE CHR(0) .. CHR(31)

PRODUCTIONS // Version 1
  Contents   = [ Title [ "by" Author [ Date ] ] ] { Chapter } .
  Title      = word { word } "." .
  Author     = word { word } "." .
  Date       = "(" number ")" .
  Chapter    = "Chapter" number Title { Subsection } .
  Subsection = section word { word } .
END Contents.
```

(a) Compute the FIRST and FOLLOW sets for each of the non-terminals of this grammar.

(b) Is this an LL(1) grammar? Justify your answer, don't just guess!

(c) The above grammar allows the *Contents* to be completely empty. Comment on the following attempt to change it so that (i) if the *Title* appears the list of chapters is optional but (ii) if the *Title* is absent there must be at least one *Chapter*.

```
PRODUCTIONS // Version 1 modified
  Contents   =   [ Title [ "by" Author [ Date ] ] ] Chapter { Chapter }
               | Title [ "by" Author [ Date ] ] { Chapter } .
```

(d) How would you add actions to the *original grammar* (Version 1) above so as to develop a system that

could tell you the title of the chapter with the greatest number of subsections and also issue a warning if the contents turns out to be completely empty?

(A spaced version of the grammar appears in the free information, which you can complete and hand in with your answer book.)

(e) Here are two other possibilities for the set of productions for this system:

```
PRODUCTIONS // Version 2
  Contents   = [ Sentence [ "by" Sentence [ "(" number ")" ] ] ] { Chapter } .
  Sentence   = Words "." .
  Words      = word { word } .
  Chapter    = "Chapter" number Sentence { Subsection } .
  Subsection = section Words .
END Contents.

PRODUCTIONS // Version 3
  Contents   = [ Words "." [ "by" Words "." Date ] ] { Chapter } EOF .
  Chapter    = "Chapter" number Words "." { section Words } .
  Words      = word { word } .
  Date       = [ "(" number ")" ] .
END Contents.
```

If I were to claim that these grammars are "equivalent" I would be using the word "equivalent" in a special way. What is meant by the statement "two grammars are equivalent"?

(f) Even though the sets of productions are equivalent, a developer might have reasons for preferring one set over the others. Which of these sets do you consider to be the "best", and why?

QUESTION A6 **[12 marks]**

Consider the description of the contents of a book in Question A5, using the productions in version 1.

Assume that you have `accept` and `abort` routines like those you used in this course, and a scanner `getSym()` that can recognise tokens that might be described by the enumeration

```
EOFSym, noSym, wordSym, numberSym, sectionSym, bySym, periodSym, chapterSym, lparenSym, rparenSym
```

How would you complete the parser routines below? There is no need to incorporate the refinement suggested in (c) of Question A5 - simply show the syntax analysis. A spaced copy of this system appears in the free information, which you are invited to complete and hand in with your answer book.

```
static void Contents () {
// Contents = [ Title [ "by" Author [ Date ] ] ] { Chapter } .

}

static void Title () {
// Title = word { word } "." .

}

static void Author () {
// Author = word { word } "." .

}

static void Date () {
// Date = "(" number ")" .

}

static void Chapter () {
// Chapter = "Chapter" number Title { Subsection } .

}

static void Subsection () {
// Subsection = section word { word } .

}
```

Section B: Enumeration types in Parva [70 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.

Yesterday you made history when you were invited to modify the Parva compiler to allow users to introduce "enumeration types".

Later in the day you were provided with a sample solution to that challenge, and the files needed to build that system have been provided to you again today.

Since the basic system was built under great pressure, some subtle points were conveniently overlooked, and some features were omitted. So continue now to answer the following unseen questions that will test your ability to refine the system further. As a hint, many of these refinements only require you to add or modify a few lines of code as it is currently written. Your answers should be given by showing the actual code you need, not by giving a long-winded description in English!

As usual, a suite of simple test programs has been supplied for your use.

QUESTION B7 **[6 marks]**

The compiler only handled *if* statements without *else* clauses. That is very restrictive! Extend the system to allow *else* clauses.

QUESTION B8 **[6 marks]**

Allow for the definition of an enumerated type (optionally) to include the declaration of variables of that type, as in

```
enum Villains { Tom, Dick, Harry } ringLeader, fallGuy = Dick;
```

QUESTION B9 **[4 marks]**

Allow enumeration values to be used directly as indices in array access.

```
enum Villains { Tom, Dick, Harry };
int[] ages = new int[3];
age[Tom] = 12;
```

QUESTION B10 **[10 marks]**

The system as provided does no checking that the results of casting operations remain "in bounds". For example, code like

```
enum Villains { Tom, Dick, Harry };
int myAge = 65;
Villains ringleader = cast(Villains, myAge);
```

should generate a run-time error. Why does it not do so, and how can you improve on this?

Hint: The PVM implementation you were supplied yesterday and today has some useful opcodes that have not previously been used.

QUESTION B11 **[10 marks]**

Similarly, code like

```
enum Villains { Tom, Dick, Harry };
Villains ringleader = Harry, scab = Tom;
ringLeader++;
--scab;
```

should generate an error. Why does it not do so, and how can you improve on this?

QUESTION B12 **[6 marks]**

Allow for functions `max()` and `min()` in expressions, whose purpose is to allow you to write code like

```
enum Villains { Tom, Dick, Harry };
Villains fallGuy;
for fallGuy = min(Villains) to max(Villains) write(fallGuy);
```

QUESTION B13 **[24 marks]**

Casting in the form we have suggested is messy. Modify the system so that it can be done with code like

```
Villains fallGuy = Villains(2);   // Harry
int  i = int(fallGuy);            // 2
char c = char(65);                // A
```

Once again, ensure that casting operations check that the values remain "in bounds".

QUESTION B14 **[4 marks]**

It would surely be preferable to have a statement like

```
for fallGuy = Tom to Harry write(fallGuy);
```

produce the output `Tom Dick Harry` rather than `0 1 2`. This turns out to be harder than it looks (so don't waste time here in trying it). Why should it be difficult?

END OF EXAMINATION QUESTIONS

Section C

(Summary of free information made available to the students 16 hours before the formal examination.)

A complete system incorporating the features they had been asked to implement was supplied to candidates in a later version of the examination kit. They were encouraged to study it in depth and warned that questions in Section B would probe this understanding. No hints were given as to what to expect, other than that they might be called on to comment on the solution, and perhaps to make some modifications and extensions. They were also encouraged to spend some time thinking how any other ideas they had during the earlier part of the day would need modification to fit in with the solution kit presented to them. The system as supplied at this point was deliberately naive in some respects, in order to lay the ground for the unseen questions of the next day.

Section D

(Summary of free information made available to the students 24 hours before the formal examination.)

Candidates were provided with an exam kit for Java or C#, containing files for developing a Parva compiler like that which they had used in the practical course.

They were also given a description of enumeration types, and a suite of simple, suggestive test programs and asked to extend the compiler to handle these. Finally, they were told that later in the day some further ideas and hints would be provided.

An example of a Parva program with enumerations is given here for reference:

```
void main () { // enumtest.pav
// Illustrate some simple enumeration types in extended Parva
// Some valid declarations

  enum DAYS    { Mon, Tues, Wed, Thurs, Fri, Sat, Sun };
  enum WORKERS { BlueCollar, WhiteCollar, Manager, Boss };
  enum DEGREE  { BSc, BA, BCom, MSc, PhD };
  enum FRUIT   { Orange, Pear, Banana, Grape };

  const pay = 100;

  DAYS yesterday, today, tomorrow;
  WORKERS[] staff = new WORKERS[12];
  int[] payPacket;
  int i;
  bool rich;
  FRUIT juice = Orange;
  DEGREE popular = BSc;

// Some potentially sensible statements
  today = Tues;
  yesterday = Mon;                                      // That follows!
  if (today < yesterday) write("Compiler error");       // Should not occur
  today++;                                              // Working past midnight?
  if (today != Wed) write("another compiler error");

  int totalPay = 0;
  for (today = Mon; today <= Fri; today++)
    totalPay = totalPay + pay;
  for today = Sat to Sun
    totalPay = totalPay + 2 * pay;

  rich = staff[i] > Manager;
  yesterday = cast(DAYS, (int) today - 1);
  tomorrow = cast(DAYS, (int) today + 1);

// Some possible meaningless statements - be careful
  enum COLOURS { Red, Orange, Green };  // Is this valid?
  juice = cast(FRUIT, Pear);            // Is this valid?
  juice = cast(FRUIT, popular);         // Is this valid?
  Sun++;                                // Cannot increment a constant
  today = Sun; yesterday = today - 1;   // Sounds reasonable?
  if (today == 4)                       // Invalid comparison - incompatibility
    staff[1] = rich;                    // Invalid assignment - incompatibility
  Manager = Boss;                       // Cannot assign to a constant
  payPacket[Boss] = 1000;               // Is this a valid subscript expression?
  payPacket[tomorrow] = 100000;         // Is this a valid subscript expression?
}
```

Free information

Summary of useful library classes

The following summarizes some of the most useful aspects of the available simple I/O classes.

```
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(byte o)
  public void write(short 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(byte o)
  public void writeLine(short 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(byte o,    int width)
  public void write(short 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(byte o,    int width)
  public void writeLine(short 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 int readInt(int radix)
  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 code illustrates various of the string and character manipulation methods that are available in Java and which are useful in developing translators.

```
import java.util.*;

    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 ... i2-1]
    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 (Java 1.4)
      sb = new StringBuffer(),                //
      sb1 = new StringBuffer("original");     //
    StringBuilder                             // build strings (Jaba 1.5 and 1.6)
      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
    i = sb.indexOf(s1);                       // position of s1 in sb
    sb.delete(i1, i2);                        // remove sb[i1 .. i2-1]
    sb.deleteCharAt(i1);                      // remove sb[i1]
    sb.replace(i1, i2, s1);                   // replace sb[i1 .. i2-1] by s1
    s = sb.toString();                        // convert sb to real string
    c = sb.charAt(i);                         // extract sb[i]
    sb.setCharAt(i, c);                       // sb[i] = c

    StringTokenizer                           // tokenize strings
      st = new StringTokenizer(s, ".,");      // delimiters are . and ,
      st = new StringTokenizer(s, ".,", true); // delimiters are also tokens
      while (st.hasMoreTokens())              // process successive tokens
        process(st.nextToken());

    String[]                                  // tokenize strings
      tokens = s.split(".;");                 // delimiters are defined by a regexp
    for (i = 0; i < tokens.length; i++)       // process successive tokens
      process(tokens[i]);
```

Strings and Characters in C#

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

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

    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
    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
    s = s.Trim();                            // remove leading/trailing whitespace
    s = s.ToUpper();                         // equivalent uppercase string
    s = s.ToLower();                         // equivalent lowercase string
    char[] ca = s.ToCharArray();             // create character array
    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
    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

    char[] delim = new char[] {'a', 'b'};
    string[]  tokens;                        // tokenize strings
    tokens = s.Split(delim);                 // delimiters are a and b
    tokens = s.Split('.' ,':', '@');         // delimiters are . : and @
    tokens = s.Split(new char[] {'+', '-'});  // delimiters are + -?
    for (int i = 0; i < tokens.Length; i++)  // process successive tokens
      Process(tokens[i]);
  }
}
```

Simple list handling in Java

The following is the specification of useful members of a Java (1.5/1.6) list handling class

```
import java.util.*;

class ArrayList
// Class for constructing a list of elements of type E

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

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

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

  public E get(int index)
  // Retrieves an element from position index

  public E set(int index, E element)
  // Stores an element 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(E element)
  // Returns true if element is in the list

  public int indexOf(E element)
  // Returns position of element in the list

  public E remove(int index)
  // Removes the element at position index

} // ArrayList
```

Simple list handling in C#

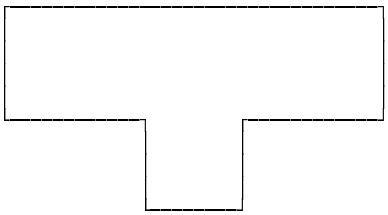
The following is the specification of useful members of a C# (2.0/3.0) list handling class.

```
using System.Collections.Generic;

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

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

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

  public element this [int index] {set; get; }
  // Inserts or retrieves an element in 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 int IndexOf(E element)
  // Returns position of element in the list

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

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

} // List
```

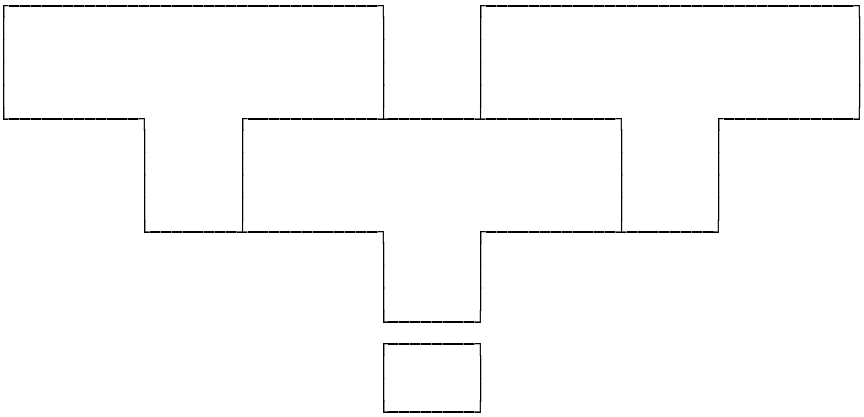
ADDENDUM 1 (Question A2)

Student number | | | | | | | | eg | 6 | 3 | T | 0 | 8 | 4 | 4 |

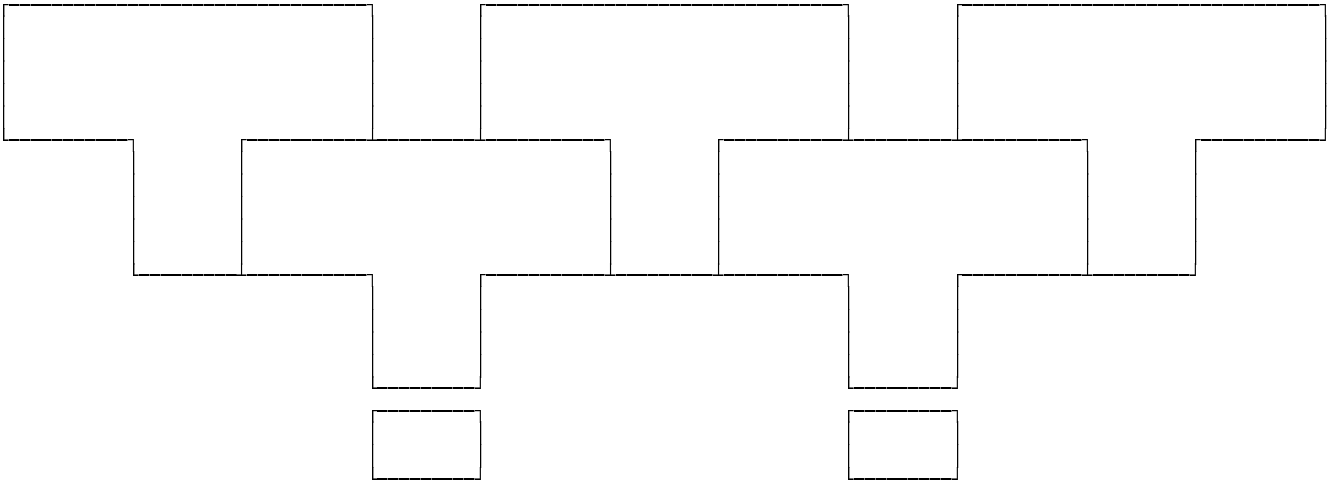
The JAD decompiler



Developing the JAD decompiler using javac



Self-consistency checking of the JAD decompiler:



ADDENDUM 2 (Question A5)

Student number | | | | | | | | eg | 6 | 3 | T | 0 | 8 | 4 | 4 |

```
COMPILER Contents


CHARACTERS
  ...

TOKENS
  ...

PRODUCTIONS // Version 1
  Contents
  = [ Title
      [ "by" Author
        [ Date
        ]
      ]
    ] { Chapter
    }
  .

  Title
  = word
    { word
    } "."
  .

  Author
  = word
    { word
    } "."
  .

  Date
  = "("
    number
    ")"
  .

  Chapter
  = "Chapter"
  number
  Title
  { Subsection
  }
  .

  Subsection
  = section
  word
  { word
  }
  .


END Contents.
```

ADDENDUM 3 (Question A6)

Student number | | | | | | | | eg | 6 | 3 | T | 0 | 8 | 4 | 4 |

```
static void Contents () {
// Contents   = [ Title [ "by" Author [ Date ] ] ] { Chapter } .









}

static void Title () {
// Title = word { word } "." .




}

static void Author () {
// Author = word { word } "." .




}

static void Date () {
// Date = "(" number ")" .




}

static void Chapter () {
// Chapter = "Chapter" number Title { Subsection } .







}

static void Subsection () {
// Subsection = section word { word } .




}
```

Examples for testing the enumeration type extensions to Parva

```
$D+ // Turn diagnostic mode on for testing the compiler - t15.pav
void main () { // Declarations with casting (bad)
  enum MyType { a, b, c };
  MyType x = cast(MyType, 12), y = 5;
}

$D+ // Turn diagnostic mode on for testing the compiler - t16.pav
void main () { // Casting between types, range errors
  enum MyType { a, b, c };
  MyType x, y;
  x = cast(MyType, -2);
  x = MyType(4);
}

$D+ // Turn diagnostic mode on for testing the compiler - t17.pav
void main () { // Casting between types, range errors
  enum Villains { Tom, Dick, Harry } stooge;
  enum Instruments { BassDrummer, SideDrummer, Piper } [] PipeBand = new Instruments[12];
  PipeBand[0] = Piper;
  stooge = Dick;
  write(PipeBand[0], stooge);
  int myAge = 65;
  Villains ringleader = cast(Villains, myAge);
}

$D+ // Turn diagnostic mode on for testing the compiler - t18.pav
void main () { // Increment enumeration - range error
  enum Villains { Tom, Dick, Harry };
  Villains ringLeader = Harry;
  ringLeader++;
}

$D+ // Turn diagnostic mode on for testing the compiler - t19.pav
void main () { // max/min function
  enum Villains { Tom, Dick, Harry };
  Villains fallGuy;
  for fallGuy = min(Villains) to max(Villains) {
    write(fallGuy);
  }
}

$D+ // Turn diagnostic mode on for testing the compiler - t20.pav
void main () { // enumerations as indices
  enum Villains { Tom, Dick, Harry };
  Villains fallGuy;
  Villains[] gang = new Villains[3];
  for fallGuy = min(Villains) to max(Villains) {
    gang[fallGuy] = fallGuy;
    write(gang[fallGuy]);
  }
}

$D+ // Turn diagnostic mode on for testing the compiler - t21.av
void main () { // new style casting
  enum Villains { Tom, Dick, Harry };
  Villains fallGuy = Villains(2);   // Harry
  int  i = int(fallGuy);            // 2
  char c = char(65);                // A
  write(c, fallGuy, i);
}

$D+ // Turn diagnostic mode on for testing the compiler - t22.pav
void main () { // can we write Tom Dick Harry instead of 0 1 2?
  enum Villains { Tom, Dick, Harry };
  Villains fallGuy;
  for fallGuy = Tom to Harry write(fallGuy);
}

$D+ // Turn diagnostic mode on for testing the compiler - t23.pav
void main () { // range checking - does it work?
  enum Villains { Tom, Dick, Harry };
  Villains fallGuy = Tom;
  while (fallGuy <= Harry) {
    write(fallGuy);  ++fallGuy;
  }
  for (fallGuy = Tom; fallGuy <= Harry; fallGuy++) write(fallGuy);
  for fallGuy = Tom to Harry write(fallGuy);
}
```