RHODES UNIVERSITY

November Examinations - 2009

Computer Science 301 - Paper 2

Examiners:
Prof P.D. Terry
Prof S. Berman

Time 3 hours
Marks 180
Pages 13 (please check!)

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

(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 C" - a request to develop an assembler/interpreter system for the Assembler language studied in their second year. 16 hours before the examination a complete grammar and other support files for building such a system were supplied to students, along with an appeal to study this in depth (summarized in "Section D"). 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 [100 marks]

A1 *Pragmas* and *comments* often appear in source code submitted to compilers. What property do pragmas and comments have in common, and what is the semantic difference between them? [4 marks]

A2 What do you understand by the term *short circuit semantics* as applied to the evaluation of Boolean expressions? Illustrate your answer by means of two specimens of code that might correspond to a simple Boolean expression of your choice. [6 marks]

A3 Parva programs are really very similar to Java ones, except for a few simple differences, as the following example will illustrate:

Parva: (Demo.pav)

```
void main () {                          // A Parva program has a single void method
  int year, yourAge;
  const myAge = 64;                     // Parva constants are defined like this
  read("How old are you? ", yourAge);   // Parva has a multiple argument read statement
  if ((yourAge < 0) || (yourAge > 100)) {
    write("I do not believe you!");
    halt;                               // Parva has a halt statement
  }
  bool olderThanYou = myAge > yourAge;  // Parva uses the keyword bool
  write("A claim that you are older than me is ", !olderThanYou);
}
```

Java equivalent: (Demo.java)

```
import library.*;                       // A simple Java program has a single class
                                        // and usually has to import library classes
class Demo {

  public static void main(String[] args) {  // A simple Java program has a standard main method
    int year, yourAge;
    final int myAge = 64;               // Java constants are defined like this
    IO.write("How old are you? ");      // Java will use methods from a library for I/O
    yourAge = IO.readInt();
    if ((yourAge < 0) || (yourAge > 100)) {
      IO.write("I do not believe you!");
      System.exit(0);                   // Java has the equivalent of a halt statement
    }
    boolean olderThanYou = myAge > yourAge; // Java uses the keyword boolean
    IO.write("A claim that you are older than me is ");
    IO.write(!olderThanYou);
  }

}
```

This should suggest that it must be relatively easy to develop a translator that will convert a Parva program to a Java equivalent, using a tool like Coco/R, acting on a grammar for Parva and with semantic actions that for the most part simply copy tokens from input to output, but making a few substitutions (such as

replacing `bool` by `boolean`) and additions (such as adding a standard header with an `import` directive and `class` declaration). Indeed, such a system was developed by your predecessors in this class some years ago under examination conditions.

(a) Show, by means of appropriate T diagrams the stages involved in producing an executable version of such a translator (P2J) assuming that you have an executable version of Coco/R for either C# or Java, as well as an executable version of either a C# or Java compiler. [5 marks]

(b) Also give a T diagram showing pe process by which a program like `Demo.pav` would be translated to one like `Demo.java`. [2 marks]

(c) You are (obviously) not expected to give full details of the system. However, if you can make the assumption that the Parva program to be converted is syntactically and semantically correct, do you foresee that your system would have to incorporate a symbol table handler? Justify your answer; don't just guess! [6 marks]

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

A4 Consider the following Cocol specification:

```
COMPILER Expressions

CHARACTERS
  digit  = "0123456789" .
  letter = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" .

TOKENS
  ident  = letter { letter | digit } .

IGNORE CHR(0) .. CHR(31)

PRODUCTIONS
  Expressions  = { Term "?" } EOF .
  Term         = Factor { "+" Factor | "-" Factor } .
  Factor       = ident [ "++" | "--" ] | [ "abs" ] "(" Term ")" .
END Declarations.
```

Develop a handcrafted scanner (not a parser!) that will recognize the tokens used in this grammar. Assume that the first few lines of the scanner routine are introduced as

```
static int getSym() {
  while (ch <= '  ') getChar();
```

and that the `getSym()` method will return an integer representing the token that it has recognized, as one of the values that you can denote by the names:

```
plus, minus, plusPlus, minusMinus, query, lParen, rParen, abs, ident
```

Assume that when the scanner invokes the source handling method

```
static void getChar()
```

this will assign to a static field `ch` (of the class of which these methods are members) the next available character in the source text. [16 marks]

A5 Consider the following grammar, expressed in EBNF, that describes the form of a typical university course:

```
Course       = Introduction Section { Section } Conclusion .
Introduction = "lecture" [ "handout" ] .
Section      = { "lecture" | "prac" "test" | "tutorial" | "handout" } "test" .
Conclusion   = [ "panic" ] "Examination" .
```

(a) What do you understand by the statement "two grammars are equivalent"? [2 marks]

(b) What do you understand by the statement "a grammar is ambiguous"? [2 marks]

(c) Rewrite these productions so as to produce an equivalent grammar in which no use is made of the EBNF meta-brackets { … } or […]. [8 marks]

(d) Clearly explain why the derived set of productions does not obey the LL(1) constraints? [4 marks]

(e) The original grammar happens to be ambiguous. Give an example of a sentence that demonstrates this. [2 marks]

(f) Does it follow that every grammar that is not LL(1) must be ambiguous? Why, or why not? If not, give an example of a simple non-LL(1) grammar that is not ambiguous. [5 marks]

(g) Does it follow that if a parser is constructed for this grammar that it is doomed to fail? Justify your answer. [3 marks]

(h) The University Senate have decreed that courses may no longer be offered if they do not obey various constraints. In particular, an acceptable course may not

(1) Offer fewer than 50 lectures, or subject the candidates to more than 8 tests
(2) Hold a practical until at least 4 lectures have been given
(3) Hold more practicals than tutorials

Show how the grammar may be augmented to impose these constraints. A spaced copy of the grammar appears in the free information, which you are invited to complete and hand in with your answer book. [12 marks]

A6 Generations of Parva programmers have complained about the absence of a *for* loop, and it has been decided to add this feature, using syntax suggested by Pascal, and exemplified by

```
for i = 1 to 10 write(i);
for j = i + 1 to i - 4 {
  read(i); total = total + i;
}
```

(a) Write an EBNF description that should allow you to recognize such a statement, taking care not to be over restrictive. [2 marks]

(b) What static semantic constraints must be satisfied by the various components of your production(s)? You might illustrate this by writing the Cocol attributes; alternatively just specify them in concise English. [4 marks]

(c) Critically examine and comment in detail on the suggestion that a *for* loop of the form

```
for i = start to stop {
  // something
}
```

should be regarded as (dynamically) semantically exactly equivalent to the following: [10 marks]

```
i = start;
while (i <= stop) {
  // something
  i++;
}
```

(d) Why do you suppose that language designers take the easy way out and state that "the value of the control variable of a *for* loop is to be regarded as undefined after the loop terminates?" [2 marks]

(e) If a *for* loop is implemented as in (c) - or, indeed, in any other way - problems would arise if the code indicated by "something" were to alter the value of the control variable. Do you suppose a compiler could forbid this practice? If so, how - and if not, why not? [5 marks]

Section B: Constructing an assembler [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.

Yesterday you made history when you were invited to develop a basic assembler/interpreter for the Assembler language that was used in our second year course for many years.

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 are best given by showing the actual code you need, not by giving a few sentences of explanation!

B7 The assembler currently will not allow you to have blank lines or comments before `BEG` or after `END`, as illustrated by

```
                    ; This is an introductory comment
                    ; extending over a few lines before
BEG                 ; the program code actually starts

END                 ; not quite the final comment
                    ; because there is still more to say
```

Show how to modify the system to allow this. [4 marks]

B8 The assembler currently will allow you to define a label more than once, as in

```
      BEG
A     LDI  23
      OTI
A     HLT               ; duplicated label
      END
```

Show how to modify the system to report duplication of labels as incorrect. [4 marks]

B9 The assembler currently performs no checks to ensure that all labels have been correctly defined, as in

```
BEG
LDA  A              ; A is never defined
OTI
HLT
END
```

Show in some detail how to modify the system to report on any labels that are never defined. [10 marks]

B10 Some assemblers allow you to use the mnemonics of opcodes as operands in addresses, as in

```
BEG
LDI  OTA            ; Equivalent to  LDI  27 (see table of opcodes)
ADI  PSH + POP      ; Equivalent to  ADI  16 + 17
HLT
END
```

Show how to modify the system to allow this. [6 marks]

B11 The virtual machine for this system should ensure that the values of all registers are confined to the range 0 … 255. Currently this is not the case (although for simple programs you may not have noticed!) There are several places that need attention, including the code that handles input operations and routines for incrementing and decrementing registers (there may be others as well).

Indicate the changes needed to the virtual machine emulator to handle this correctly. [8 marks]

B12 Similarly, there may be places in the assembler, code generator and table handler that are not paying attention to the same requirements as in question 11. Indicate the changes needed to improve the situation. [10 marks

B13 In a single pass assembler like this, the DS directive becomes very awkward unless the value of its address field is known at the point that assembly has reached. The following code is meaningless:

```
      BEG
      LDA  LIST        ; Equivalent to LDA 100
      HLT
LIST  DS   A           ; ??
A     DS   LIST - 12   ; ??
      END
```

Show how to modify the system to report this sort of nonsense as incorrect. [4 marks]

B14 Add an ORG directive to the assembly language that will allow you to specify the address from which future assembly will follow, as in

```
      BEG
      LDA  LIST        ; Equivalent to LDI 27 (see table of opcodes)
      BZE  ADD5        ; Equivalent to BZE 110
      HLT
      ORG  100         ; shift assembly
LIST  DS   10          ; LIST occupies bytes 100 ... 109
ADD5  ADI  5           ; ADI is in byte 110
      HLT
      END
```

Show how to modify the system to allow this. [8 marks]

B15 The assembler currently allows addresses to be formed by the simple addition and subtraction of terms, as in

```
      LDA  LIST + 1
      ADI  10 - TOTAL
```

but it does not allow the first (or maybe only) term in such an address field to be preceded by an optional minus, as in

```
      ADI  - A - B + 8
```

Show how to modify the system to allow this. [10 marks]

B16 Most assemblers will allow an EQU directive that can be used to define a label to have a particular value - useful for giving names to magic numbers as in

```
CR    EQU  13
      LDI  CR
      OTA
      LDI  LF          ; IO.writeLine()  CR/LF
      OTA
LF    EQU  10
```

Show how to modify the system to allow this. Be careful - the label defined by an EQU directive might legally have been "used" before the directive is encountered. [16 marks]

Section C

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

Candidates were reminded of the Assembler language that they had encountered in an earlier course, and were invited to create an assembler/interpreter that would generate VM code from such source.

An informal description (from the second year course) and a basic Cocol grammar for the language were provided. This grammar is reproduced below.

```
COMPILER Assem $NC
/* Describe Assembler language as used in CSC 201 */

IGNORECASE

CHARACTERS
  lf         = CHR(10) .
  cr         = CHR(13) .
  backslash  = CHR(92) .
  control    = CHR(0) .. CHR(31) .
  letter     = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" .
  digit      = "0123456789" .
  binDigit   = "01" .
  hexDigit   = digit + "ABCDEFabcdef" .
  stringCh   = ANY - '"' - control - backslash .
  charCh     = ANY - "'" - control - backslash .
  printable  = ANY - control .

TOKENS
  decNumber  = digit { digit } .
  binNumber  = binDigit { binDigit } "%" .
  hexNumber  = digit { hexDigit } ( "H" | "h" ) .
  identifier = letter { letter | digit } .
  stringLit  = '"' { stringCh | backslash printable } '"' .
  charLit    = "'" ( charCh   | backslash printable ) "'" .
  EOL        = cr lf | lf .
  comment    = ";" { printable } .

PRODUCTIONS
  Assem             = Begin StatementSequence End .
  Begin             = "BEG" CommentEOL .
  End               = "END" CommentEOL .
  CommentEOL        = [ comment ] SYNC EOL .
  StatementSequence = { Statement CommentEOL } .
  Statement         = [ Label ] [ OneByteOp | TwoByteOp Address ] .
  OneByteOp         = (   "ASR"  | "CLA"  | "CLC"  | "CLV"  | "CLX"  | "CMA"  | "CMC"
                        | "DEC"  | "DEX"  | "HLT"  | "INA"  | "INB"  | "INC"  | "INH"
                        | "INI"  | "INX"  | "NOP"  | "OTA"  | "OTB"  | "OTC"  | "OTH"
                        | "OTI"  | "POP"  | "PSH"  | "RET"  | "SHL"  | "SHR"  | "TAX" ) .
  TwoByteOp         = (   "ACI"  | "ACX"  | "ADC"  | "ADD"  | "ADI"  | "ADX"  | "ANA"
                        | "ANI"  | "ANX"  | "BCC"  | "BCS"  | "BGE"  | "BGT"  | "BLE"
                        | "BLT"  | "BNG"  | "BNZ"  | "BPZ"  | "BRN"  | "BVC"  | "BVS"
                        | "BZE"  | "CMP"  | "CPI"  | "CPX"  | "DC"   | "DS"   | "JGE"
                        | "JGT"  | "JLE"  | "JLT"  | "JSR"  | "LDA"  | "LDI"  | "LDX"
                        | "LSI"  | "LSP"  | "ORA"  | "ORI"  | "ORX"  | "SBC"  | "SBI"
                        | "SBX"  | "SCI"  | "SCX"  | "STA"  | "STX"  | "SUB" ) .
  Address           = Term { '+' Term | '-' Term } .
  Term              = Label | IntConst | StringConst | CharConst | '*' .
  Label             = identifier .
  IntConst          = decNumber | binNumber | hexNumber .
  StringConst       = stringLit .
  CharConst         = charLit .

END Assem.
```

Candidates were provided with an exam kit for Java or C#, containing files like those which they had used in the practical course, and with skeleton files for creating a tailored assembler, as well as the source code for a complete VM emulator. The kit also contained the executable of a assembler/interpreter (developed in Pascal) that would allow them to compare the code generated by this program with that produced by the system they were developing. They were also given a suite of simple, suggestive test programs. Finally, they were told that later in the day some further ideas and hints would be provided.

Section D

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

A complete assembler 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.

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 boolean 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#

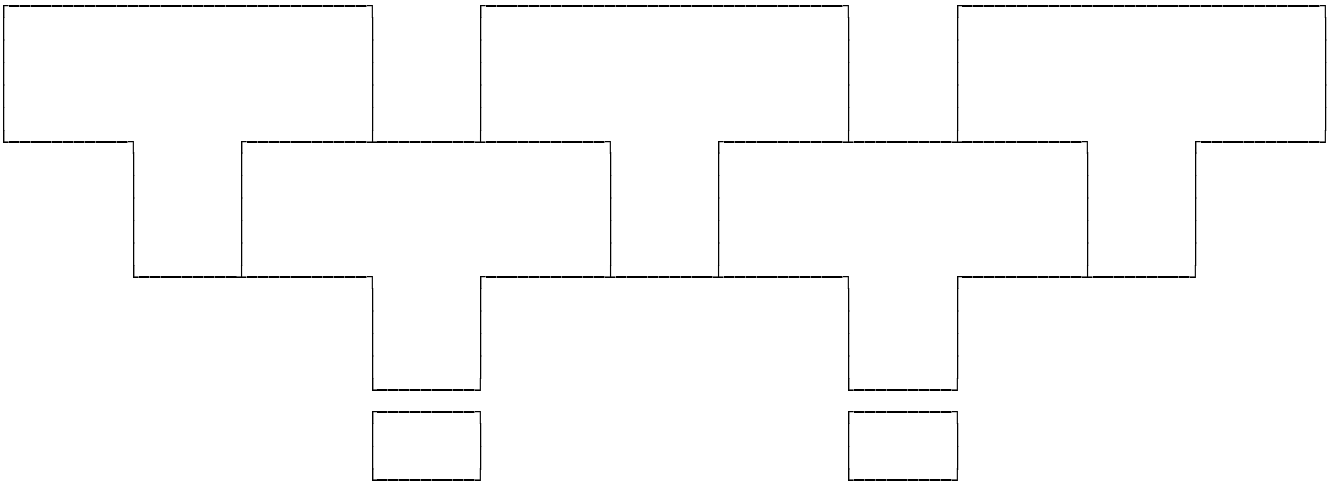
The following is the specification of useful members of a C# (2.0/3.0) list handling class.

```
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
```

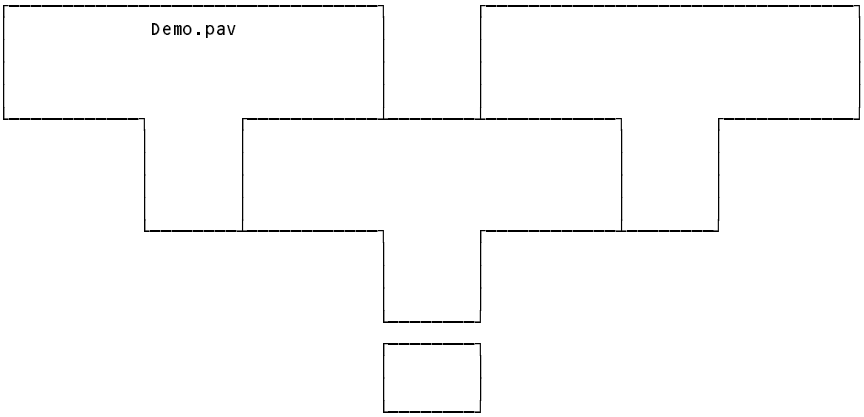
```
    public boolean Contains(E element)
    // Returns true if element is in the list

    public boolean 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 3 (Question A3) - Fill in your Student number

Developing the translator P2J using Coco/R



Using the translator



ADDENDUM 2 (Question A6) - Fill in your Student number

```
COMPILER Course



IGNORE CHR(0) .. CHR(31)

PRODUCTIONS

  Course
  = Introduction

    Section

    { Section

    } Conclusion



  .

  Introduction
  = "lecture"

    [ "handout"
    ] .


  Section
  = {   "lecture"

      | "prac"


        "test"



      | "tutorial"

      | "handout"

    }

    "test"


  .


  Conclusion
  = [ "panic"
    ]

    "Examination"
  .

END .
```