RHODES UNIVERSITY

November Examinations - 2013

Computer Science 301 - Paper 1

Programming Language Translation

Examiners:
Prof P.D. Terry
Prof P. Blignaut

Time 4 hours
Marks 180
Pages 14 (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 C" - a request to use Coco/R to generate C# code that could be incorporated within a complete program for analysing the examination results of undergraduates, using a description of administrative actions to be taken (like exclusion) based on various criteria. Some 16 hours before the examination a complete grammar for a simple version of this system and other support files for building it 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 systems, access to a computer, and machine readable copies of the questions.)

Section A: Conventional questions [90 marks]

QUESTION A1 **[6 marks]**

A1. (a) What distinguishes a *context free grammar* from a *context sensitive grammar*? [2 marks]

(b) The syntax of many programming languages is described by a context free grammar, and yet there are properties of most programming languages that are context sensitive. Mention one such property, and indicate briefly how this context sensitivity is handled in practical compilers. [4 marks]

QUESTION A2 **[6 marks]**

A2. What distinguishes a *concrete syntax tree* from an *abstract syntax tree*? Illustrate your answer by drawing both forms of tree corresponding to the simple Java statement [6 marks]

```
a = ( b + c ) ;
```

QUESTION A3 **[4 marks]**

A3. Describe briefly but clearly what you understand by the following two properties of variables as they are used in block structured imperative programming languages - firstly, their *scope* and, secondly, their *existence*. [4 marks]

QUESTION A4 **[8 marks]**

A4. (a) What distinguishes a *native code compiler* from an *interpretive compiler*? [2 marks]

(b) Suggest (with some explanation) one property of native code compilation that is claimed to offer an advantage over interpretive compilation, and also one property of interpretive compilation that is claimed to offer an advantage over native code compilation. [2 marks]

(c) What do you understand by the technique known by the acronym JIT, and what systems known to you incorporate this technique? [4 marks]

QUESTION A5 **[30 marks]**

A5. *(Cocol and recursive descent parsing)* The following familiar Cocol grammar describes a set of EBNF productions (of the form found in the PRODUCTIONS section of the grammar itself).

```
COMPILER EBNF $CN
/* Describe a set of EBNF productions
   P.D. Terry, Rhodes University, 2013 */

CHARACTERS
  eol         = CHR(10) .
  space       = CHR(32) .
  control     = CHR(0) .. CHR(31) .
  letter      = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" .
  digit       = "0123456789" .
  lowline     = "_" .
  printable   = ANY - control .
  nonSpace    = printable - space .
  noquote1    = ANY - "'" - control .
  noquote2    = ANY - '"' - control .

TOKENS
  nonterminal = letter { letter | lowline | digit } .
  terminal    = "'" noquote1 { noquote1 } "'" | '"' noquote2 { noquote2 } '"' .

IGNORE control
```

```
PRODUCTIONS
  EBNF        = { Production } EOF .
  Production  = nonterminal "=" Expression  "." .
  Expression  = Term { "|" Term } .
  Term        = Factor { Factor } .
  Factor      =   nonterminal | terminal | "(" Expression ")"
                | "[" Expression "]" | "{" Expression "}" .
END EBNF.
```

(a) In the original notation known as BNF, productions took a form exemplified by

```
<nonterminal> ::= ε  |  terminal1  | ( <another nonterminal> | terminal2 ) + <something>
```

The notation allowed the use of an explicit ε. [] brackets and {} braces were not used (although () parentheses were allowed). Non-terminals and terminals were distinguished by the presence or absence of < > angle brackets, and a production was terminated at the end of a line.

Describe a set of BNF productions using Cocol (simply modify the grammar above). Use `"eps"` to represent ε and BNF as the goal symbol. [15 marks]

(b) Assume that you have available a suitable scanner method called `getSym` that can recognize the terminals of this new grammar BNF and classify them appropriately as members of the following enumeration

```
EOFSym, noSym, EOLSym, termSym, nontermSym, definedBySym,
epsilonSym, barSym, lParenSym, rParenSym
```

Develop a hand-crafted recursive descent parser for recognizing a set of such BNF productions (not the original EBNF production set given earlier), that is based on your description in (a). *(Your parser can take drastic action if an error is detected. Simply call methods like* `accept` *and* `abort` *to produce appropriate error messages and then terminate parsing. You are not required to write any code to implement the* `getSym`, `accept` *or* `abort` *methods.)* [15 marks]

QUESTION A6 [16 marks]

A6. *(Grammars)* By now you should be familiar with RPN or "Reverse Polish Notation" as a notation that can describe expressions without the need for parentheses. The notation eliminates parentheses by using "postfix" operators after the operands. To evaluate such expressions one uses a stack architecture, such as formed the basis of the PVM machine studied in the course. Examples of RPN expressions are:

```
3 4 +                - equivalent to   3 + 4
3 4 5 + *            - equivalent to   3 * (4 + 5)
```

In many cases an operator is taken to be "binary" - applied to the two preceding operands - but the notation is sometimes extended to incorporate "unary" operators - applied to one preceding operand:

```
4 sqrt               - equivalent to   sqrt(4)
5 -                  - equivalent to   -5
```

Here are two attempts to write grammars describing an RPN expression:

```
(G1)    RPN       =    RPN RPN binOp
                     | RPN unaryOp
                     | number .
        binOp     =   "+" | "-" | "*" | "/" .
        unaryOp   =   "-" | "sqrt" .
```

and

```
(G2)    RPN      =   number REST .
        REST     =   [ number REST binOp REST | unaryOp ].
        binOp    =   "+" | "-" | "*" | "/" .
        unaryOp  =   "-" | "sqrt" .
```

(a) Give a precise definition of what you understand by the concept of an *ambiguous grammar*. [2 marks]

(b) Give a precise definition of what you understand by the concept of *equivalent grammars*. [2 marks]

(c) Using the expression

```
15   6   -   -
```

as an example, and by drawing appropriate parse trees, demonstrate that both of the grammars above are ambiguous. [6 marks]

(d) Analyse each of these grammars to check whether they conform to the LL(1) conditions, explaining quite clearly (if they do not!) where the rules are broken. [6 marks]

QUESTION A7 **[20 marks]**

A7. *(Code generation)* A BYTE (Bright Young Terribly Eager) student has been nibbling away at writing extensions to her first Parva compiler, while learning the Python language at the same time. She has been impressed by a Python feature that allows one to write multiple assignments into a single statement, as exemplified by

```
A, B = X + Y, 3 * List[6];
A, B = B, A;                // exchange A and B
A, B = X + Y, 3, 5;         // incorrect
```

which she correctly realises can be described by the context-free production

```
Assignment = Designator { "," Designator } "=" Expression { "," Expression } ";"
```

The Parva attributed grammar that she has been given deals with single, simple integer assignments only:

```
Assignment                    (. DesType des; .)
=  Designator<out des>        (. if (des.entry.kind != Entry.Var)
                                   SemError("invalid assignment"); .)
   "="
   Expression                 (. CodeGen.assign(); .)
   ";" .
```

where the `CodeGen.assign()` method generates an opcode that is matched in the interpreter by the following `case` arm:

```
case PVM.sto:
  //   store value at top of stack on address at second on stack
  mem[mem[cpu.sp + 1]] = mem[cpu.sp];

  //   bump stack pointer
  cpu.sp = cpu.sp + 2;
  break;
```

Suggest, in as much detail as time will allow, how the `Assignment` production and the interpreter would need to be changed to support this language extension. Give the Cocol and C#/Java code that would be needed, not a simple hand-waving argument like "add some more opcodes". [20 marks]

Section B [90 marks]

Your answers to the following questions should, whenever possible, take the form of actual code, and not simply vague discussion.

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 EXAMJ.ZIP (Java) or EXAMC.ZIP (C#), modify any files that you choose, and then finally copy all the files back onto an exam folder on the network.

Save often, to guard against the catastrophe of a power outage.

There is a very simple script for saving all the files again, which will be explained to you by the invigilators.

QUESTION B8 **[3 marks]**

We have used the operators `and, or, not, =` and `/=` to denote operations which in C# or Java are denoted by `&&, ||, !, ==` and `!=`. This is reasonable; the suggestions might be more in line with what Deans readily understand. How would the system be changed to allow *either* version of each of these operators to be used in the criteria list (for the benefit of the Dean of Commerce, who just happens to be a programmer).

QUESTION B9 **[2 marks]**

Other than simply calling it a "compiler" or "translator", how would you classify the system that is built for translating specifications to code (for example: pretty-printer, assembler, decompiler, native code compiler, interpretive compiler, *etc* ...)?

QUESTION B10 **[2 marks]**

It was suggested in the preliminary discussion that you could write your system in Java, although the rest of the larger system into which your generated method would eventually be injected was originally written in C#. Presumably a Java implementation could be arranged to generate C# code. What is the term used to describe translators that generate code for a system other than the one in which they are themselves implemented?

QUESTION B11 **[10 marks]**

The `Table` class supplied as a possibility in the exam kit is weak in several respects. In particular it does not check that any identifier can be added to the table only once, or that the actions have tag numbers that are distinct. Suggest how it could be improved in this respect (give code modifications, not vague descriptions).

QUESTION B12 **[3 marks]**

It might be thought helpful to allow a Dean to use synonyms (words with the same meaning) to define criteria. Suppose, for example, that we wanted to be able to use any of the words `Lowest, Lowest_MarkYear` or `Worst` to refer to a student's lowest mark for the year. How could this feature be added to your system (give code if possible)?

QUESTION B13 **[10 marks]**

The system as provided implies that a set of criteria applies to a single Faculty. There is no check made that a student being analysed is, in fact, registered in that Faculty. Modify the code to detect such errors, and add suitable warning messages to the output log file if they occur.

QUESTION B14 **[15 marks]**

(Use your imagination) Supppose a Dean writes a set of criteria like the following

```
FirstYear:
  Exclude :      Passes < 2;
                 Lowest_Mark < 15.
  Congratulate : Passes = Courses.
```

In this case a student who had taken only one subject, and passed it at 50% would be marked for both congratulation and exclusion, which is kind of silly. In this case the silliness is easy to spot, but in general it might be far more difficult to do so. Can you think of any mechanism that the Ancient Skilled Programmer might recommend that the Deans might use so as to allow the system to detect and report nonsensical combinations of actions, perhaps by generating a suitable entry to the log file. (Give appropriate code where possible.)

QUESTION B15 **[10 marks]**

Is it really necessary to use semicolons between alternative criteria and to terminate a set of alternatives with a period (full stop)? Deans might regard these as confusing punctuation and prefer something simpler such as

```
FirstYear:
  Exclude :      Passes < 2
                 Lowest_Mark < 15
  Congratulate : Passes = Courses
```

If punctuation is unnecessary, how would the attributed grammar be changed to accommodate specifications like these.

From a compiler writer's perspective, discuss whether there is any merit in insisting that punctuation be used.

QUESTION B16 **[15 marks]**

In practice, Faculties administer several degrees - for example at Rhodes the Commerce Faculty administers BCom, BEcon and BBS. The criteria used by Deans often apply to some degrees, but not all.

Suppose we wished to extend the system to allow input exemplified by

```
Humanities
Degrees: BA, BSS:
  FirstYear:
    Exclude: Passes < 2.
  SecondYear:
    Exclude: Full_Creds_So_Far < 3.
Degrees: BFA, BJrn, BMus:
  FirstYear:
    Exclude: Passes < 3.
AllDegrees:
  SecondYear:
    Exclude: Full_Creds_SoFar < 6.
```

That is, where some degrees have one common set of criteria, while some other degrees have a different common set. In terms of the unattributed grammar this might be achieved by altering the `Converter` production to read:

```
Converter    = Faculty
               { "AllDegrees" ":" | "Degrees" ":" Degree { "," Degree } ":" } )
                 {   "FirstYear"  ":" CriteriaList
                   | "SecondYear" ":" CriteriaList
                   | "ThirdYear"  ":" CriteriaList
                   | "FourthYear" ":" CriteriaList
                   | "AnyYear"    ":" CriteriaList
                 }
               } .
```

Complete the remaining changes that would be needed to allow this extension.

QUESTION B17 **[20 marks]**

The system as it stands allows Deans to write criteria like

```
SecondYear:
  Weak    : Passes < 3.
  Exclude : Weak or Average < 30.
```

where one criterion might depend upon another one. It also attempts to warn a Dean who might have written those particular criteria in the reverse order:

```
SecondYear:
  Exclude : Weak or Average < 30.
  Weak    : Passes < 3.
```

(To understand why such a check might be useful, consider a student who has taken five subjects and failed them all at 45%.)

Here is another, more complicated example:

```
FirstYear :
  Weak : Average < 40.
SecondYear:
  Exclude : Weak or Average < 30.
  Weak : Passes < 3.
```

(a) What well-known feature of many computer languages is being called into play here? [2 Marks]

(b) In a study of translators a distinction is drawn between *static semantics* and *dynamic semantics*. Does the strategy suggested in the grammar supplied to you qualify to be termed a static semantic check or a dynamic semantic check? Justify your answer. [4 Marks]

(c) Does the suggested strategy actually work? Explain your reasoning whether you believe it does or does not (don't simply guess!) If not, can you suggest how it might be improved, and discuss whether your improvements would always solve the problem in situations like these. If not, why not? (We are looking for insight here!) [14 Marks]

END OF EXAMINATION QUESTIONS

Section C

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

Candidates were provided with some basic ideas, and asked to develop a version of a system that could transform specifications like the following:

```
Humanities
  FirstYear :
    Exclude : Full_Creds_So_Far < 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 :
              Full_Creds_So_Far /= 10.0 and Years_Here > 3.
  AnyYear :
    Congratulate :
              Weighted_Average > 90.0 and Fails = 0.
```

into C# or Java code 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 the outcome, or to draw attention to problems
    s.actions = new IntSet();
    if s.faculty.Equals("Humanities") {
      if (s.academicYear == 1) {
        if (s.fullCreditsSoFar < 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.fullCreditsSoFar != 10.0 && s.yearsOfStudy > 3)
          s.actions.Incl(21);
      }
      if (s.wgtAverage > 90.0 && s.failsThisYear == 0)
        s.actions.Incl(11);
    }
  } // ActionSetBuilder.BuildSet

} // ActionSetBuilder
```

They were provided with an exam kit for Java or C#, containing the Coco/R system, along with a suite of simple, suggestive test specifications. 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 grammar for a rudimentary solution to the exercise posed earlier in the day 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; few 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.

A context-free basic Cocol description of the converter appears below:

```
COMPILER Converter  $CN
/* Converter for generating a C# or Java method, transforming a list of criteria for marking up
   student exam results, from a simple specification, into equivalent C# or Java code.  Context
   free grammar only shown here; in the examination an attributed version was supplied to students,
   P.D. Terry, Rhodes University, 2013 */

IGNORECASE

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

TOKENS
  integer      = digit { digit } | digit { digit } CONTEXT (".") .
  double       = digit { digit } "." digit { digit }  .
  identifier   = letter { letter | digit | "_" ( letter | digit ) } .

IGNORE CHR(0) .. CHR(31)

PRODUCTIONS
  Converter    = Faculty
                 {   "FirstYear"  ":" CriteriaList
                   | "SecondYear" ":" CriteriaList
                   | "ThirdYear"  ":" CriteriaList
                   | "FourthYear" ":" CriteriaList
                   | "AnyYear"    ":" CriteriaList
                 } .
  Faculty      = Identifier .
  Identifier   = identifier .

  CriteriaList = { Action ":" Condition { ";" Condition } SYNC "." } .
  Action       = Identifier .

  Condition    = Expression .
  Expression   = AndExp { "or" AndExp } .
  AndExp       = EqlExp { "and" EqlExp } .
  EqlExp       = RelExp { EqlOp RelExp } .
  RelExp       = AddExp [ RelOp AddExp ] .
  AddExp       = MulExp { AddOp MulExp } .
  MulExp       = Factor { MulOp Factor } .
  Factor       = Primary | ( "+" | "-" | "not" ) Factor .
  Primary      = Designator | Double | Integer | "(" Expression ")" .

  Designator   = identifier .
  Integer      = integer .
  Double       = double .

  AddOp        = "+" | "-" .
  MulOp        = "*" | "/" .
  EqlOp        = "=" | "/=" .
  RelOp        = "<" | "<=" | ">" | ">=" .

END Converter.
```

Free information

Summary of useful library classes

The following summarizes the simple set handling and I/O classes that have been useful in the development of applications using the Coco/R compiler generator.

```
class IntSet  { // simple set handling routines - There are matching versions for C#
  public IntSet()
  public IntSet(int ... members)
  public Object clone()
  public IntSet copy() {
  public boolean equals(Symset s)
  public void incl(int i)
  public void excl(int i)
  public boolean contains(int i)
  public boolean isEmpty()
  public int members()
  public IntSet union(IntSet s)
  public IntSet intersection(IntSet s)
  public IntSet difference(IntSet s)
  public IntSet symDiff(IntSet s)
  public void write()
  public String toString()
} // IntSet

public class OutFile {  // text file output - There are matching versions for C#
  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 - There are matching versions for C#
  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
```