RHODES UNIVERSITY

Computer Science 301 - 2009 - Programming Language Translation

Well, here you are. Here is the free information you have all been waiting for, with some extra bits of advice:

- Don't panic. It may be easier than you might at first think.
- The final solution(s) in Section B tomorrow will need rather careful thought. I am looking for evidence of mature solutions and insight, not crude hacks.
- Work in smaller, rather than larger groups. Too many conflicting ideas might be less helpful than a few carefully thought out ones.
- Do make sure you get a good night's sleep!

How to spend a Special Sunday

From now until about 22h30 tonight, Computer Science 3 students have exclusive use of the Hamilton Laboratory. You are encouraged to throw out anyone else who tries to use it, but we hope that this does not need to happen. At about 22h30 Chris, Dameon and I will have to move some computers around and prepare things for Monday. If there is still a high demand we shall try to leave some computers for use until later, but by then we hope you will have a good idea of what is involved.

This year the format of the examination is somewhat different from years gone by, so as to counter what had happened where, essentially, one solution was prepared by quite a large group and then passed around to many others, who had probably not contributed to it in any real way. As in 2007, the problem set below is only part of the story. At 16h30 you will receive further hints as to how this problem should be solved (by then you might have worked this out for yourselves, of course). You will be encouraged to study the problem further, and the hints in detail, because during the examination tomorrow you will be set further questions relating to the system - for example, asked to extend the solution you have worked on in certain ways. It is my hope that if you have really understood the material today, these questions will have solutions that come readily to mind, but of course you will have to answer these on your own!

My "24 hour exam" problems have all been designed so that everyone should have been able to produce at least the basic solution, with scope given for top students to demonstrate their understanding of the subtler points of parsing, scanning and compiling. Each year I have been astounded at the quality of some of the solutions received, and I trust that this year will be no exception.

Please note that there will be no obligation to produce a machine-readable solution in the examination (in fact doing so is quite a risky thing, if things go wrong for you, or if you cannot type quickly). The tools will be provided before the examination so that you can experiment in detail. If you feel confident, then you are free to produce a complete working solution to Section B during the examination. If you feel more confident writing out a neat detailed solution or extensive summary of the important parts of the solution, then that is quite acceptable. Many of the best solutions over the last few years have taken that form.

During the first few hours you will be able to find the following files in the usual places, or by following links from the course web page:

- All the files as were made available for the practical course.
- The files `FREE1J.ZIP` and `FREE1C.ZIP`, which contain the Java and C# versions of the Coco/R system, and its support files, a basic grammar and skeleton files and test data for today's exercise. The kit also contains a PDF version of the Coco/R user manual (`CocoManual.pdf`).

At 16h30 new versions of the exam kit will be posted (`FREE2J.ZIP` and `FREE2C.ZIP`), and a further handout issued, with extra information.

Most of the machines in the labs will work in exactly the way they have done this semester. Some are set up to show you the configuration you can expect tomorrow. It is suggested that you work at one of those machines for a short time to make sure you know what to expect. To work on these machines you proceed as follows

- Open up a DOS window following the usual route

- Give the command

  ```
  CONNECT  TESTxxx   yyyyy
  ```

  where `xx` is a three digit number for the machine (typically in the range `101` through `123`) and `yyyy` is a 5 digit password.

- This will then allow you to log onto the `J:` drive, where you will find the files `FREE1J.ZIP` (and `FREE1C.ZIP`) waiting for you.

- Give a command like

  ```
  UNZIP   FREE1J.ZIP
  ```

  to unpack the "exam kit" of choice.

From here on things should be familiar. You could, for example, log onto the `D:` or `J:` drive, use `UltraEdit` use `CMAKE` and `CRUN` … generally have hours of fun. The C# system will work only on the `D:` drive, however.

But note that the exam set up has *no* connection with the outside world - no ftp client, telnet client, shared directories - not even a printer!

Today you may use the files and either the "usual" or the "exam" systems in any way that you wish, subject to the following restrictions: *Please observe these in the interests of everyone else in the class*.

(a) When you have finished working, **please** delete any files from the D: drive, so that others are not tempted to come and snoop around to steal ideas from you.

(b) You are permitted to discuss the problem with one another, and with anybody not on the "prohibited" list.

(c) You are also free to consult books in the library. If you cannot find a book that you are looking for, it may well be the case that there is a copy in the Department. Feel free to ask.

(d) Please do not try to write any files onto the C: directory, for example to `C:\TEMP`

(e) If you take the exam kit to a private machine you will need to have Java installed (or the .NET framework or equivalent to use the C# version).

I suggest that you *do* spend some of the next 24 hours in discussion with one another, and some of the time in actually trying out your ideas. You have plenty of time in which to prepare and test your ideas - go for it, and good luck. **Remember that tomorrow you may not bring anything into the room other than your student card and writing utensils, and especially not listings, diskettes, memory sticks, text books or cell phones.**

If you cannot unpack the file, or have trouble getting the familiar tools to work (unlikely!), you may ask me for help. You may also ask for explanation of any points in the question that you do not understand, in the same way that you are allowed to ask before the start of an ordinary examination. You are no longer allowed to ask me questions about any other part of the course. Sorry; you had your chance earlier, and I cannot allow this without risking the chance of sneak questions and suggestions being called for.

If you cannot solve the problem completely, don't panic. It has been designed so that I can recognize that students have reached varying degrees of sophistication and understanding.

How you will spend a Merry Monday

Before the start of the formal examinations the laboratory will be unavailable. During that time

- The machines will be completely converted to a fresh exam system, with no files left on directories like `D:` or `C:\TEMP` .

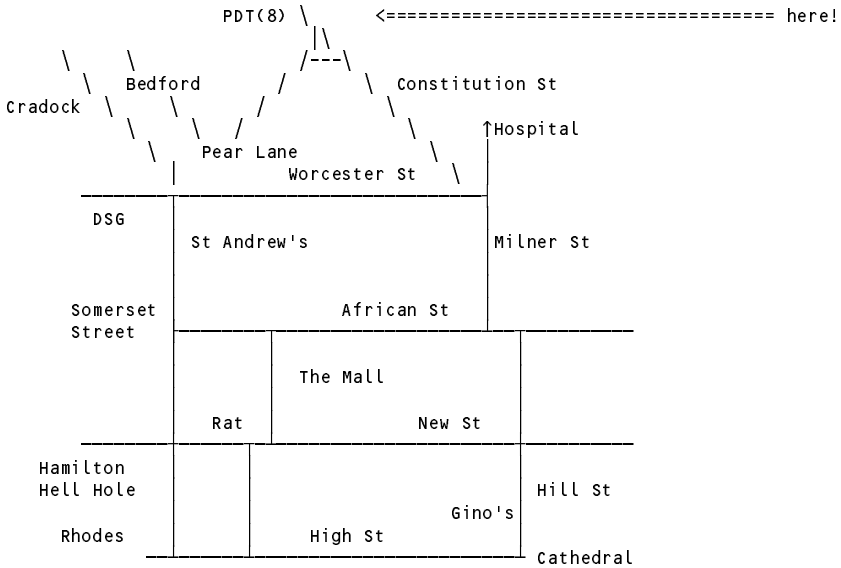
- The network connections will be disabled.

At the start of the examination session:

- You will receive ordinary answer books for the preparation of answers, and an examination paper.
- You will receive a copy of at least the relevant portions of the grammar and support files that you received at 16h30. *You may annotate these during the exam to form part of your solution if you wish to submit a hand-written answer to Section B and need to make reference to the code (possibly by line number).* In this case you should hand in the annotated listings with your answer book.
- You will be allocated to a computer and supplied with a `CONNECT` command for your own use. Once connected you will find an exam kit on the `J:` drive. This will contain the same Coco/R system and other files you have been given today, and in addition there will be "flat ASCII" machine readable parts of the examination paper itself, in files with names like `Q7.TXT` (Question 7). *There is no obligation to use a computer during the exam. You can answer on paper if you prefer - and yes, you can write in pencil if you prefer that.*
- At the end of the exam you will be given a chance to copy any files that you have edited or created on the D: drive back to the server. This will be explained tomorrow.

Cessation of Hostilities Party

As previously mentioned, Sally and I would like to invite you to an informal end-of-course party at our house on 10 November. There's another copy of the map below to help you find your way there. It would help if you could let me know whether you are coming so that we can borrow enough glasses, plates etc. *Please post the reply slip into the hand-in box during the course of the day.* Time: from 18h30 onwards. Dress: Casual



Section B [80 marks]

It was inevitable …

After many years of apparently bug-free use, the Department of Computer Science has found that there is a bug in the software that has been used to teach Assembler programming techniques to second year students. The original assembler/interpreter system was developed in Pascal. Tragically, it has become increasingly difficult to find people familiar with Pascal programming, which has led to a request to produce a new version of the system developed in Java or C#, rather than trying to modify the original Pascal source code. There is some urgency in this regard, as a bug-free version of the system is required at 24 hours notice for use in an examination.

A document exists that describes the Assembler language informally, and a copy of this has been provided, as has the following attempt at writing an EBNF description of the syntax of programs written in this popular language.

```
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.
```

Very fortunately, a group of students have recently become experts in the use of compiler-writing tools, and as one of that group, an appeal is being made to you to start from a description of the Assembler language and to craft an assembler/interpreter system in the next few hours. It is known that you will have used the original system yourself, and should remember the Assembler language sufficiently well to enable you to understand and write small programs in it. As luck will have it, all is not lost - a collection of correct programs developed in this language has come to light for you to use in testing your new system, so you may not have to write much, if any, Assembler code yourself! If you would like reminding (and for the benefit of future readers of this examination paper) here is an example of an Assembler program:

```
          BEG             ; Hollywood.ASM - place 100 stars on the screen
                          ; Nicky Greenwall, 2009

          LDI   100       ; CPU.A = 100;
          TAX             ; CPU.X = 100;
          LDI   '*'       ; CPU.A = '*';
   LOOP                   ; do {
          OTA             ;   IO.writeChar('*');
          DEX             ;   CPU.X--;
          BNZ   LOOP      ; } while (CPU.X != 0);
          HLT             ; System.exit(0);
          END
```

Hints:

(a) The examination kit includes all the files needed to build a Coco-defined Parva compiler similar to the ones seen in your recent practical exercises. These are provided for reference only, and in case you wish to refresh your memory of how various techniques might be applied.

(b) The kits contain updated libraries, an outline grammar and support files for the assembler, as well as a working implementation of the virtual machine. Hence you will easily be able to make an assembler/interpreter after you have developed the grammar, code generator and table handler further. As usual, there is a Java version of the kit and an equivalent C# version. You may use either version, and build the assembler with the command

```
CMAKE Assem
```

(c) You are advised to study the outline grammar and support files before you rush in to code, and to think carefully about the form the symbol table and its entries will take. In particular, productions in the `ASSEM.ATG` file as suggested above may need some re-factoring before the actions can sensibly be added.

(d) It will suffice to use the generic `ArrayList` (Java) or `List` (C#) class as the basis of the symbol table. A synopsis of this class can be found below, and will be supplied again during the examination proper.

(e) The kit also includes various simple test programs of the sort you wrote in CSC 201 (in files named `XXXX.ASM`) along with the corresponding generated code (as 256 numbers in files named `XXXX.CODE`, to distinguish them from files named `XXXX.COD` which will be produced by the assembler system itself). A listing of some of these source files has been provided in a separate handout.

(f) The example `.ASM` programs in the kit range from very short, simple ones, to quite advanced ones. You are well advised to start with the very simple ones that are only a few statements long, and to develop your system in an incremental way - get one program to assemble correctly before moving on to the next one.

(g) The simple batch file `ASSEM.BAT` has been provided in the Java kits to make testing easier - after making the system, a command like `ASSEM XXXX.ASM` may be used in place of `CRUN Assem XXXX.ASM`. C# users will already know to give the command `Assem XXXX.ASM` in place of using `CRUN`.

(h) A version of the CSC 201 assembler - essentially the one you used in CSC 201 - has been provided in the kit and can be executed by giving a command like

```
CSC201  XXXX.ASM
```

The differences between `CSC201` and the assembler you used in your second year are

(1) Both strings and character constants in the original version were demarcated by single quotes:

```
Tyrant  DC  'Pat Terry'
Period  DC  '.'
```

but in the version in the kit you may use either single or double quotes. Escape sequences like `\t` are not recognized by `CSC201` - but this is of no consequence.

(2) Assembly of a `.ASM` file produces a matching `.CODE` text file containing the 256 values of the bytes that were loaded into the virtual memory, immediately before interpretation commences.

(i) Your assembler should use single and double quotes to distinguish between character literals and string literals, as was the case for the Parva compiler. Although the grammar as supplied incorporates the `IGNORECASE` directive, this will apply only to key words. If you wish to make your assembler truly case-insensitive for labels as well, apply the `toUpperCase()` method (Java) or `ToUpper()` method (C#) to `token.val` at an appropriate point.

(j) You may be tempted to try simply to use the `Label` class from the Parva compiler to deal with labels in the assembler. While both the Parva compiler and this assembler must handle forward and backward references, they are, perhaps, slightly more complicated in the assembler situation.

(k) The .COD file produced by the Coco-generated system should, of course, be identical to the .CODE file produced by CSC201 from the same source, so one way of testing your system is to assemble a source program with CSC201 as well as with your system, and then compare the .COD and .CODE files. Note that this would not be an exhaustive test, but if the files are different this will signify that something must be wrong.

(l) Both the ASSEM and CSC201 commands allow you to add a further parameter -A to suppress interpretation and stop after assembly, which you might find useful:

```
d:\exam> csc201 TEST.ASM -A
d:\exam> assem  TEST.ASM -A
```

(m) Comparing two text files may be done in several ways. One is to use the FC command, in a sequence like

```
d:\exam> csc201 TEST.ASM -A
d:\exam> assem  TEST.ASM -A
d:\exam> fc     TEST1.CODE  TEST1.COD
FC: No differences encountered
```

However, to make this process even simpler, a batch file TEST.BAT has been provided in the kit. The command

```
d:\exam> TEST XXXX
```

(note the absence of the .ASM extension) will attempt to assemble XXX.ASM with both assemblers and to compare the resulting XXX.COD and XXXX.CODE files. You might also like to redirect the output to a file for later viewing:

```
d:\exam> TEST XXXX >JUNKFILE
```

(n) Later in the day - at 16h30 - we shall release more information, to help those of you who may not have completed the exercise to do so. Section B of the examination tomorrow may include a set of unseen questions that probe your understanding of the system.

(o) Rest assured that you will not be expected to reproduce a complete assembler system from memory under examination conditions, but you may be asked to make some additions or improvements to the system developed today.

(p) Remember Einstein's Advice: "Keep it as simple as you can but no simpler" and Terry's Corollary: "For every apparently complex programming problem there is an elegant solution waiting to be discovered".

Good luck!

Free information

Summary of useful library classes

The following summarizes some of the available simple I/O classes in Java (The C# ones are equivalent, but for small differences in MethodNames and methodNames). Note that the input methods allow you to specify a base - these methods have only been added to the library very recently. The IO library has static versions of the methods in the InFile and OutFile classes, but these should be familiar to you.

```
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 long    readLong(int radix)
  public short   readShort()
  public short   readShort(int radix)
  public byte    readByte()
  public byte    readByte(int radix)
  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
```