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
```