RHODES UNIVERSITY

November Examinations - 2008

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 a Forth-like assembler for the PVM. 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 [100 marks]

A1. (a) What distinguishes a "compiler" from an "assembler"? [2]

(b) How would you explain and distinguish the terms "self-resident compiler", "cross-compiler" and "self-compiling compiler" to a student taking a course in compiler constuction for the first time, and how could you lead them to believe that a self-compiling compiler can ever be a reality? [8]

A2. Explain clearly and simply what you understand by the terms syntax, static semantics, and dynamic semantics, and what distinguishes one from another. [6]

A3. In the practical sessions you should have used the Extacy Modula-2 to C translator. This was developed in Russia by a team who used the JPI Modula-2 compiler available for the PC. The demonstration system we downloaded from the Internet came with the file `XC.EXE`, and a few other modules written in Modula-2 (but not the source of the `XC.EXE` executable itself). Draw T-diagrams showing the process the Russians must have used to produce this system, and go on to draw T-diagrams showing how you managed to take a program `SIEVE.MOD` and run it on the PC using the Borland C++ system as your compiler of choice. [10]

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

A4. (a) What would you understand by a statement from a group of very unhappy students who came to you and told you that they had discovered that their carefully thought out grammar had turned out to be "ambiguous"? [2]

(b) Why can an LL(1) conformant grammar never be ambiguous? (Explain carefully!) [3]

(c) Does it then follow that if a grammar is not ambiguous it must automatically be LL(1) conformant? Support your argument by giving some simple example grammars. [5]

(d) Discuss the anomaly in programming language design that gives rise to the so-called "dangling else" problem. Show how one could easily design a language that does not suffer from this problem. Also explain why the problem is in any case less severe than it might at first appear. [6]

A5. The Cocol grammar below attempts to describe declarations in a simple C-like language:

```
COMPILER Declarations

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

TOKENS
  ident  = letter { letter | digit | "_" ( letter | digit ) } .
```

```
COMMENTS FROM "/*" TO "*/"

IGNORE CHR(0) .. CHR(31)

PRODUCTIONS
  Declarations = { DecList } EOF .
  DecList      = Type OneDecl { "," OneDecl } ";"  .
  Type         = "int" | "void" | "bool" | "char" .
  OneDecl      = "*" OneDecl | Direct .
  Direct       = ( ident | "(" OneDecl ")" ) [ Params ] .
  Params       = "(" [ OneParam { "," OneParam } ] ")" .
  OneParam     = Type [ OneDecl ] .
END Declarations.
```

(a) Is this an LL(1) compliant grammar? Explain your reasoning in some detail. [6]

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

```
EOFSym, noSym, identSym, intSym, voidSym, boolSym, charSym, commaSym, lparenSym,
rparenSym, semicolonSym, starSym;
```

How would you complete the parser routines below? [20]

A spaced copy of this system appears in the free information, which you are invited to complete and hand in with your answer book.

```
static SymSet
  FirstDeclarations = new SymSet(                              ),
  FirstDeclList     = new SymSet(                              ),
  FirstType         = new SymSet(                              ),
  FirstOneDecl      = new SymSet(                              ),
  FirstDirect       = new SymSet(                              ),
  FirstParams       = new SymSet(                              ),
  FirstOneParam     = new SymSet(                              );

static void Declarations () {
// Declarations = { DecList } EOF .

}

static void DecList () {
// DecList = Type OneDecl { "," OneDecl } ";"  .

}

static void Type () {
// Type = "int" | "void" | "bool" | "char" .

}

static void OneDecl () {
// OneDecl = "*" OneDecl | Direct .

}

static void Direct () {
// Direct = ( ident | "(" OneDecl ")" ) [ Params ] .

}

static void Params () {
// Params = "(" [ OneParam { "," OneParam } ] ")" .

}

static void OneParam () {
// OneParam  = Type [ OneDecl ] .


}
```

A6. The examination results for a class of students are to be supplied in a file which gives, for each student, their student number, surname, gender, faculty, and mark obtained - a typical extract might read

```
63T0844  Terry           Male   S  85
06M1234  MacDonald       Female H  50.00
0601234  O'Malley        Male   C  78.6
06S1234  Smith-Jenkins   Male   S  55.67
81W4251  Bradshaw        Female S  99
02F4567  Van Wyk Smith   Female C  48
01M1234  McGregor        Male   L  10.30
```

Student numbers are of a standard format - two digits giving the year of first registration (02 denotes 2002, 95 denotes 1995 and so on), followed by the initial letter of the surname at the time of registration and then by a sequence of four digits. Names begin with a capital letter, and composite surnames like "Van Wyk Smith" and ones with embedded apostrophes and hyphens are also permissible. The faculty is denoted by a single letter with an obvious relationship to our faculties of Science, Humanities, Commerce, Law, Education and Pharmacy.

(a) Show how an attributed Cocol grammar could be used to parse such a file and determine the number of students in the list who first registered more than three years ago. A skeleton grammar file is provided in the free information, which you could complete and submit with your answer book. [15]

(b) Clearly this problem can be solved very easily without the use of a tool like Coco/R. Are there any advantages to be gained from using Coco in simple applications of this sort, and if so, what might these advantages be? [5]

A7. Suppose that you are developing a recursive descent parser for a language *L* described by a grammar *G* known to conform to the LL(1) restrictions. As has often been claimed, writing such a parser is very easy if one can assume that the users of the language *L* will never make coding errors, but this is wishful thinking! Write a brief essay on how one might modify the parser for a non-terminal *A* of the grammar *G* so as to incorporate simple but effective error recovery. [12]

Section B [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 for generating PVM code from assembler source inspired by the Forth language.

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. Continue now to answer the following unseen questions.

B8. At present the system will allow a user to redeclare another variable or constant using an existing name, or even to define a variable or constant that has the same name as a word like `NEGATE`. While this might be thought to be useful, it is also dangerous. How would the system need to be modified to prevent such behaviour? [5]

B9. The language Forth had a number of other words that were of use in writing programs that manipulated the stack efficiently. Among these were DROP, which simply discarded the top element from the stack, SWAP, which had the effect of exchanging the two items at the top end of the stack, and OVER, which had the effect of pushing another copy of the item just below the top of stack onto the stack. Show how these new words could be added to the system, by introducing new opcodes to the PVM and corresponding entries to the standard dictionary. [15]

Hint: As an example of using OVER, if the stack originally looked like this:

				130	-45	126	234

<--- stack grows SP

then after executing OVER it should look like

			-45	130	-45	126	234

<--- stack grows SP

B10. Given the introduction of these words, explain what the effect would be of the following sequence of code: [10]

```
A @ B @
OVER OVER < IF SWAP ENDIF DROP
.
```

B11. Show how one could add an `ELSE` clause to the `IF ... ENDIF` construction, thus permitting one to write code of the form

```
IF  "correct"  ELSE  "incorrect" ENDIF
```

to display an appropriate message based on the value currently on the top of the stack. [10]

B12. At present, comparisons of a value with zero are easily achieved with code like

```
A @     ( push value of variable A )
0       ( push zero )
==      ( compare )
```

Show how, without modifying the opcode set of the PVM, one could introduce a new word to the dictionary

```
0==
```

which would allow a programmer to write, instead: [10]

```
A @ 0==
```

B13. The system as supplied to you is dangerous, in that no checks have been introduced to ensure that control structures like `IF ... ENDIF` and `REPEAT ... UNTIL` are correctly balanced. Consider, for example, what would happen were the assembler to be presented with code like

```
UNTIL C @ 0 == REPEAT C ?
```

Suggest how such errors could be detected and reported. [10]

B14. A very powerful feature in Forth is the ability for the user to define new words of his or her own choice. This is achieved by using so called "colon definitions". A construction like

```
: EVALUATE 0 >= IF "positive" pos ++ ELSE "negative" neg ++ ENDIF ;
```

defines the word `EVALUATE` to have the effect of examining the value at the top of the stack, displaying an appropriate message, and then incrementing a relevant counter. This word could then be used in code like

```
A @ EVALUATE  B @ EVALUATE
```

with the implication that each time `EVALUATE` is encountered, the corresponding sequence of words is introduced into the code. This "macro" facility uses a colon-semicolon pair to demarcate this sequence, and the name of the word so defined is supplied immediately after the colon. As a second example, the following defines a word that will allow one to examine and print the value at the top of the stack as both an integer and character, but without discarding it:

```
: PEEK DUP . DUP .C ;
```

New words need not be limited to having alphanumeric names like `EVALUATE` and `PEEK`. One could

introduce, for example

```
: 0==  0 == ;
```

as an alternative method for handling the situation described in B12.

Show how the colon-definition facility could be added to the assembler. Hint: This can be done elegantly in a relatively small number of lines of code. [20]

Section C

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

Candidates were provided with the basic ideas of a Forth-like assembly language, and were invited to create an assembler that would generate PVM code from such source.

It was suggested that they might limit themselves to a system that would do the following:

(a) handle strings, integer literals and character literals as code generators
(b) Declare named variables and constants
(c) Map simple "words" to the obviously equivalent PVM opcodes
(d) Handle IF … THEN and REPEAT … UNTIL control constructs

They were provided with an exam kit for Java or C#, containing a working Parva compiler like that which they had used in the practical course, and with skeleton files for creating a tailored assembler, including a rudimentary code generator and dictionary handler. 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; 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 encouraaged 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 available simple set handling and I/O classes.

```
class SymSet  { // simple set handling routines
  public SymSet()
  public SymSet(int ... members)
  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 SymSet union(SymSet s)
  public SymSet intersection(SymSet s)
  public SymSet difference(SymSet s)
  public SymSet symDiff(SymSet s)
  public String toString()
} // SymSet
```

```
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(int 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(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(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(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 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 program illustrates various of the string and character manipulation methods that are available in Java and which are useful in developing translators.

```
import java.util.*;

class Demo {
  public static void main(String[] args) {
    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]);

  }
}  import java.util.*;
```

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
```

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

```
import java.util.*;

class Stack<E>
// Class for constructing a stack of elements of type E

  public Stack<E>()
  // Empty stack constructor

  public void push(E element)
  // Appends element to top of stack

  public E pop()
  // Retrieves an element from top of stack

  public E peek()
  // Returns element on top of stack without removing it

  public void clear()
  // Clears all elements from stack

  public int size()
  // Returns number of elements in stack

  public boolean empty()
  // Returns true if stack is empty

  public boolean contains(E element)
  // Returns true if element is in the stack

} // Stack
```

Simple list handling in C#

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

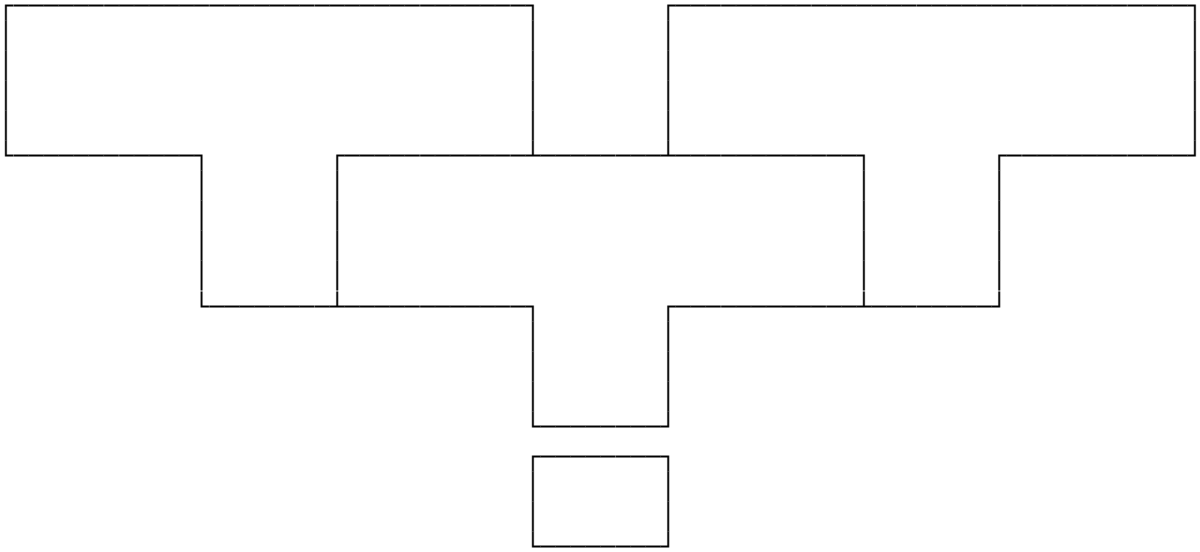
```
using System.Collections.Generic;

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

} // ArrayList
```

The following is the specification of useful members of a C# 2.0 generic stack handling class

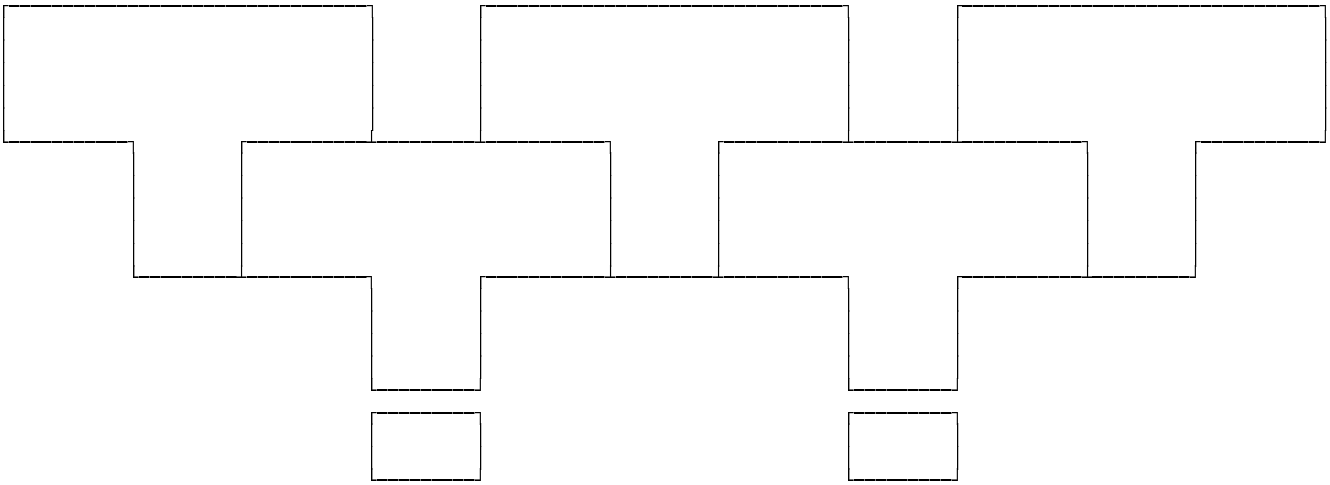
```
using System.Collections.Generic;

class Stack<E>
// Class for constructing a stack of elements of type E

  public Stack<E>()
  // Empty stack constructor

  public void Push(E element)
  // Appends element to top of stack

  public E Pop()
  // Retrieves an element from top of stack

  public E Peek()
  // Returns element on top of stack without removing it

  public void Clear()
  // Clears all elements from stack

  public int Count { get; }
  // Returns number of elements in stack

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

} // Stack
```

A3. T Diagrams for explaining the X2C system. Student Number

Developing XC itself



Using XC



A6. Cocol grammar for dealing with a set of course records Student number

```
import Library.*;

COMPILER Marks $NC





CHARACTERS
  lf            = CHR(10) .
  control       = CHR(0) .. CHR(31) .



TOKENS





IGNORE  CHR(9) + CHR(11) .. CHR(13)

PRODUCTIONS
  Marks
  =



END Marks.
```

A5. Parser for declarations in a simple C-like language: Student number:

```
static SymSet
  FirstDeclarations = new SymSet(                                          ),

  FirstDeclList     = new SymSet(                                          ),

  FirstType         = new SymSet(                                          ),

  FirstOneDecl      = new SymSet(                                          ),

  FirstDirect       = new SymSet(                                          ),

  FirstParams       = new SymSet(                                          ),

  FirstOneParam     = new SymSet(                                          );

static void Declarations () {
// Declarations = { DecList } EOF .









}

static void DecList () {
// DecList = Type OneDecl { "," OneDecl } ";"  .









}

static void Type () {
// Type = "int" | "void" | "bool" | "char" .









}
```

```
static void OneDecl () {
// OneDecl = "*" OneDecl | Direct .



}

static void Direct () {
// Direct = ( ident | "(" OneDecl ")" ) [ Params ] .



}

static void Params () {
// Params = "(" [ OneParam { "," OneParam } ] ")" .



}

static void OneParam () {
// OneParam  = Type [ OneDecl ] .



}
```