RHODES UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

EXAMINATION: JUNE 2014

Computer Science 301

PAPER 2 - Translators

Internal Examiner: Prof P.D. Terry

External Examiner: Prof P. Blignaut

MARKS: 180
DURATION: 4 hours

GENERAL INSTRUCTIONS TO CANDIDATES

---

1. This paper consists of 13 pages and 14 questions. Please ensure that you have a complete paper.

2. You will be provided with a separate document that indicates the hand in procedure for the examination.

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 develop class definitions for Parva. Some 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.)

PLEASE DO NOT TURN OVER THIS PAGE UNTIL TOLD TO DO SO

Section A: Conventional questions [95 marks]

QUESTION A1 **[12 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? [10]

QUESTION A2 **[8 marks]**

A2. Explain clearly and simply what you understand by the terms syntax, static semantics, and dynamic semantics, and what distinguishes one from another. [8]

QUESTION A3 **[10 marks]**

A3. In the practical sessions you should have used our Parva to Java translator. This was developed in C# from an attributed grammar for Parva written in Cocol. The Coco/R system used this grammar to generate C# source code that was compiled by a software development kit for C# to yield an EXE file that was circulated to your class. Draw T-diagrams showing the process used to produce this system, and go on to draw T-diagrams showing how you managed to take a program `SIEVE.PAV`, convert it to Java and then run it on the PC using the Java compiler `javac.class` and a JVM hosted on the PC system as your system 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.

QUESTION A4 **[17 marks]**

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. [6]

(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]

QUESTION A5 **[28 marks]**

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. [8]

(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 IntSet
  FirstDeclarations = new IntSet(                              ),
  FirstDeclList     = new IntSet(                              ),
  FirstType         = new IntSet(                              ),
  FirstOneDecl      = new IntSet(                              ),
  FirstDirect       = new IntSet(                              ),
  FirstParams       = new IntSet(                              ),
  FirstOneParam     = new IntSet(                              );

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 ] .


}
```

QUESTION A6 **[20 marks]**

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
12M1234  MacDonald       Female H  50.00
1201234  O'Malley        Male   C  78.6
11S1234  Smith-Jenkins   Male   S  55.67
81W4251  Bradshaw        Female S  99
09F4567  Van Wyk Smith   Female C  48
01M1234  MacHine         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]

Section B: Parva with simple classes [85 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 EXAMx.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 an extended Parva compiler which would accept simple class type definitions, and allow programs to declare and use variables of those types.

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, all but the last of which should require no changes to the code generator or interpreter.

QUESTION B7 **[9 marks]**

B8. The semantic checking in the supplied system is incomplete. For example, the compiler will accept code like the following without complaint:

```
class Wrong {
  int x;
  bool x;
};

Wrong silly;
silly.z = silly.z;
```

Indicate how and where such oversights might be corrected.

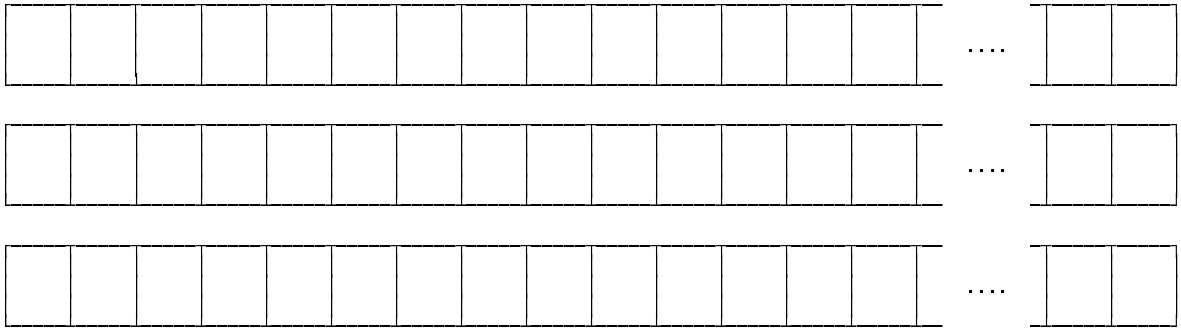
QUESTION B8 **[12 marks]**

B8. The following is a complete, if slightly pointless, program. For each iteration of the *while* loop, illustrate the state of the *stack* and *heap* immediately before the `i++;` assignment at point (a) is executed:

There are copies of these charts on page 13 for you to fill in and hand in.

```
void main() { // B8.pav
  class One {
    int x,y;
  };

  int i = 0;
  One[] list = new One[3];
  while (i <= 3) {
    int j = i + 1;
    list[i] = new One();
    list[i].x = j;
    list[i].y = 2 * j;
  // ----------------------- point (a)
    i++;
  } // while
  // point (b)
} // main
```



QUESTION B9 **[14 marks]**

B9. For the program in the previous question, draw a diagram showing the major components of the symbol table as it would exist at point (b), after completing *compilation* of the *while* loop.

QUESTION B10 **[8 marks]**

B10. Modify the compiler so that objects of a class may be declared immediately following the class definition as exemplified by

```
class One {
  int x, y;
} a, b = new One(), c = b, d = null;
```

QUESTION B11 **[8 marks]**

B11. The compiler will accept the definition of a class with no fields at all, such as

```
class Empty {
  // no field list
};
```

This might appear strange, and even stranger when it is pointed out that one can compile and execute a statement like

```
Empty e, f = e;
```

However, although compilation of a similar statement

```
Empty e = new Empty(), f = e;
```

will raise no alarm bells, the run-time execution of this statement would result in an error.

Presumably this reflects a blemish in language design or implementation. Suggest steps that could be taken to improve the language specification and/or the compilation process.

QUESTION B12 **[8 marks]**

B12. The following program illustrates a classic way to construct and display a simple stack:

```
void main() {  // B12.pav

  class Node {
     Node link;        // point (a)
     int value;
  };

  Node list = null;
  int data;
  read(data);

  while (data != 0) {
    Node next = new Node();
    next.value = data;
    next.link = list;
    list = next;
    read(data);
  }

  while (list != null) {
    writeLine(list.value);
    list = list.link;
  }

} // main
```

Unfortunately the compiler will not accept it, as reference is made to `Node` at point (a) before the definition of this class is completed. However, a trivial change to the compiler will overcome this problem. Show the change, and then comment on how it is that this violation of "must define before use" can work when others cannot as easily be made to work on a single pass, if at all.

QUESTION B13 **[11 marks]**

B13. Consider another simple, if rather awkward, program:

```
void main() {  // B13.pav

 class TestResult {
   int mark;
   bool passed;
 };

 int n = 0;

 while (true) {
   TestResult t = new TestResult();
   read("Supply mark (negative exits ", t.mark);
   if (t.mark < 0) break;
   t.passed = t.mark >= 50;
   n++;
 }
 writeLine(n, " candidates wrote the test");
} // main
```

(a) Comment on the run-time behaviour of this program, in particular with respect to its use of the heap [4].

(b) How, if at all, would this behaviour alter if the program were rewritten as follows [4]

```
void main() {  // B13a.pav

 class TestResult {
   int mark;
   bool passed;
 };

 TestResult t = new TestResult();
 int n = 0;

 while (true) {
   read("Supply mark (negative exits ", t.mark);
   if (t.mark < 0) break;
   t.passed = t.mark >= 50;
   n++;
 }
 writeLine(n, " candidates wrote the test");
} // main
```

(c) how, if at all, would the behaviour differ from what you could expect were an equivalent program to be written in C# or Java? What, in particular, do C# and Java provide that your Parva implementation does not provide? [3]

QUESTION B14 **[15 marks]**

B14. Consider the program below. Explain why the output should be "false false" and not "false true" [3].

```
void main() {  // B14.pav

 class One {
   int x, y;
 } a = new One(); b = new One();

 a.x = 1;   a.y = 2;
 b.x = a.x; b.y = 3;

 writeLine(a == b);
 b.y = a.y;
 writeLine(a == b);
} // main
```

Now go on to modify the language and implementation to introduce a function that allows you to compare two objects in the way a user might expect, along the following lines: [12]

```
void main() {  // B14a.pav

 class One {
   int x, y;
 } a = new One(); b = new One();

 a.x = 1;   a.y = 2;
 b.x = a.x; b.y = 3;

 writeLine(isEqual(a, b));  // false
 b.y = a.y;
 writeLine(isEqual(a, b));  // true
} // main
```

Hint: feel free to modify the code generator and interpreter to accomplish this task.

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 the basic ideas for adding simple class definitions to the Parva language and were invited to modify a supplied Parva implementation to accommodate these.

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. 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.

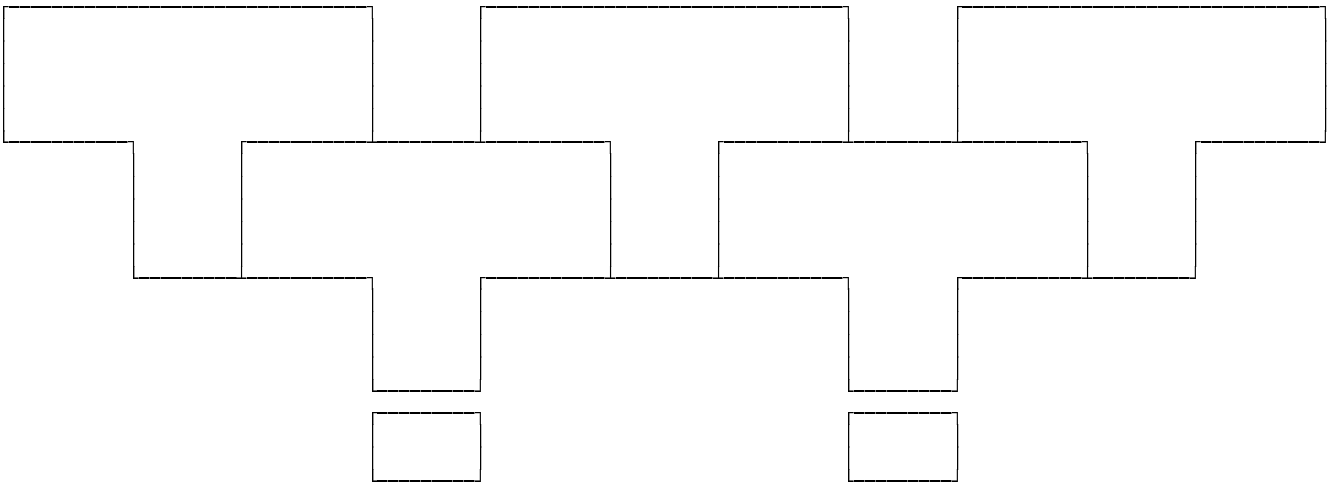
Section D

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

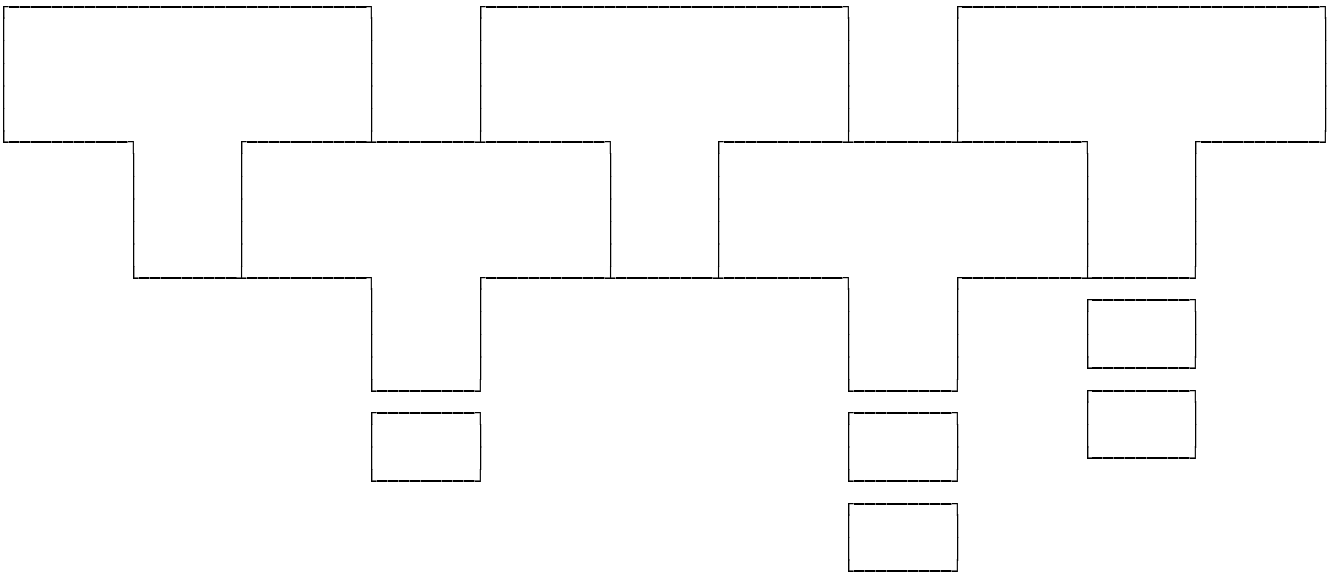
A complete Parva system incorporating the features they had been asked to implement was supplied to candidates in a later version of the examination kit. They were encouraged to study it in depth and warned that questions in Section B would probe this understanding; few hints were given as to what to 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 naïve in some respects, in order to lay the ground for the unseen questions of the following day.

A3. T Diagrams for explaining the Parva to Java system. Student Number

Developing `ParvaToJava` itself



Using `ParvaToJava`



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 IntSet
  FirstDeclarations = new IntSet(                                    ),

  FirstDeclList     = new IntSet(                                    ),

  FirstType         = new IntSet(                                    ),

  FirstOneDecl      = new IntSet(                                    ),

  FirstDirect       = new IntSet(                                    ),

  FirstParams       = new IntSet(                                    ),

  FirstOneParam     = new IntSet(                                    );

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 ] .









   }
```

B8. Run time stack and heap for the following program at point (a) : Student number:

```
void main() { // B8.pav
  class One {
    int x,y;
  };

  int i = 0;
  One[] list = new One[3];
  while (i <= 3) {
    int j = i + 1;
    list[i] = new One();
    list[i].x = j;
    list[i].y = 2 * j;
  //                      point (a)
    i++;
  } // while
  // point (b)
} // main
```