RHODES UNIVERSITY
Where leaders learn

Computer Science 301

Programming Language Translation

Practical 1: week beginning 10 July 2023

This prac is due for submission through the RUConnected submission link by 8 pm on the Wednesday before your next practical day (i.e. Wednesday 19th July 2023).

Objectives

In this practical you should aim to:

- acquaint yourselves with some command line utilities, with various editors, interpreters and compilers;
- investigate various qualities of some computer languages and their implementations, including Pascal, Java, C/C++ and Parva.
- obtain some proficiency in the use of the various library routines that will be used extensively later in the course.

The exercises for this week are not really difficult, although they may take longer than they deserve simply because you may be unfamiliar with the systems.

Copies of this handout, the Parva language report, and descriptions of the library routines for input, output, string handling and set handling in Java are available in the Software Resources folder on the RUConnected course page.

Outcomes

When you have completed this practical you should understand:

- how and where some languages are similar or dissimilar;
- how to use various command line compilers and decompilers for these languages;
- what is meant by the term "high level compiler" and how to use one;
- how to measure the relative performance of language implementations;
- the elements and limitations of programming in Parva;
- how to use I/O and set handling routines in Java.

To hand in (Total marks available = 35 marks)

This week you are required to hand in via the two links on RUConnected:

- One electronic copy per group of the completed *prac1_handin* after executing tasks 1-5 [5 marks].
- One electronic copy per group of the source code for Tasks 5 [5 marks], 8 [10 marks] and 9 [10 marks].
- Either an electronic or scanned hand-drawn copy of the T-diagrams for Task 6 [5 marks].

Note that not all your tasks may be marked each week. The tutors will mark one or two small tasks and I will mark one or two of the remaining tasks. The tasks marked by the tutors will be clearly labelled as such, whereas the tasks I will mark will not be known to you. For this practical, tutors will mark the *prac1_handin* (i.e. tasks 1-5) and the corrected code for Task 5.

Feedback for the tasks marked will be provided via RUConnected. Check carefully that your mark has been entered into the Departmental Records.

You are referred to the rules for practical submission which are clearly stated in our Departmental Handbook.

A rule not stated there, but which should be obvious, is that you are not allowed to hand in other students' work as your (or your group's) own work. Attempts to do this will result in (at best) a mark of zero and (at worst) severe disciplinary action and the loss of your DP. You are obviously allowed -- even encouraged -- to work and study with other students or student groups, but if you do this you are asked to acknowledge that you have done so with suitable comments typed into *relevant code files*. You should always include all of the names of members in your group as a comment in all code files submitted.

You are also expected to be familiar with the University Policy on Plagiarism.

Before you begin

In this practical course you will be using many simple utilities, and will usually work at the "command line" level rather than in a GUI environment. Note in particular:

- After logging on, get to the DOS command line level by executing `CMD.EXE` to open a command window if you don't already have a shortcut (it is probably worth creating a shortcut).
- Note that all code submissions for this course must be single files and not projects, as all code will be tested from the command line.

Task 0 (a trivial one)

We shall make use of zipped prac kits throughout the course; you will typically find sources for each week's prac in a file `pracN.zip` on RUConnected. Download `prac1.zip` as needed for this week and extract the sources when you need them, into your own directory/folder.

When working in the Hamilton labs, I suggest you work from the local D: drive instead of a network drive. After opening a command window, log onto the D: drive, create a working directory and unpack a copy of the prac kit there:

```
j:> d:
d:> md d:\G16B1111
d:> cd d:\G16B1111
d:> unzip prac1.zip
```

In the prac kit you will find various versions of a famous program for finding a list of prime numbers using the method known as the Sieve of Eratosthenes. You will also find various versions of a program for tabulating the Fibonacci sequence, some "empty" programs, some other bits and pieces,

including a few batch files to make some of the following tasks easier and a long list of prime numbers (`primes.txt`) for checking your own.

Task 1 The Sieve of Eratosthenes in Pascal

You may not be a Pascal expert, but in the prac kit you will find some Pascal programs, including `SIEVEPAS.PAS` that determine prime numbers using a Boolean array to form a "sieve". Study and compile these programs -- you can do this from the command line by issuing commands like:

```
FPC SIEVEPAS.pas
FPC FIBOPAS.pas
FPC EMPTYPAS.pas
```

to use the 32-bit Windows version of the Free Pascal compiler.

Make a note of the size of the executables (use the commands `DIR SIEVEPAS.EXE`, `DIR FIBOPAS.EXE` and `DIR EMPTYPAS.EXE`).

How do the sizes of the executables compare? Why do you suppose the "empty" program produces the amount of code that it does?

Note that the Sieve programs are written so that requesting one iteration displays the list of the prime numbers; requesting a large number of iterations suppresses the display, and simply "number crunches". In all cases the program reports the number of prime numbers computed. So, for example, a single iteration with an upper limit of 20 will report that there are 8 primes smaller than 20 namely, 2, 3, 5, 7, 11, 13, 17 and 19.

Prime numbers are those with no factors other than themselves and 1. But the program does not seem to be looking for factors!

Look at the code carefully. How does the algorithm work? Why is it deemed to be particularly efficient? How much (mental) arithmetic does the "computer" have to master to be able to solve the problem?

Use the table provided as prac1_handin.xls to summarise your findings on the sizes of the executables of all the compilers/program combinations tested in this prac (i.e. tasks 1, 2, 3, 4 and 5) and to answer the questions posed in the handin.xls document. [10 marks]

Task 2 And now C/C++

The kit also includes C and C++ versions of the Sieve programs. Compile these using the Embarcadero Dev-C++ compiler as well as the Borland C/C++ compiler.

You should use the following commands at the command line to compile these programs:

```
gcc SIEVEC.C         (using the Dev-C++ compiler in C mode)
bcc SIEVEC.C         (using the Borland compiler in C mode)
g++ SIEVECPP.CPP     (using the Dev-C++ compiler in C++ mode)
bcc SIEVECPP.CPP     (using the Borland compiler in C++ mode)
```

If you cannot run the Dev-C++ compiler from the command line (due to the PATH environment variable not being correctly set up), you may use the IDE to compile and run the code.

To run the C / C++ programs type:

```
sievec or sievec.exe
sievecpp or sievecpp.exe
```

Task 3 Java everywhere, but none to drink

You can compile and run the Java versions of various programs provided in the kit from the command line, for example:

```
javac sievej.java
java sievej
```

Make a note of the size of the classes produced (`sievej.class, emptyj.class` and `fiboj.class`). How do these compare with the other executables?

Task 4 See C#

You can compile the C# versions of various programs provided in the kit from the command line, for example:

```
csharp SieveCS.cs
```

If you get an error stating that csc.exe is not recognized in the normal command window, open a *Developer Command Prompt for VS 2022* window and run the C# compilations in that window instead.

Make a note of the size of the ".NET assemblies" produced (`SIEVECS.EXE, EMPTYCS.EXE` and `FIBOCS.EXE`). How do these compare with the other executables?

Task 5 The new language on the block -- Parva [5 marks]

In the “Manuals and other Documentation” folder on the RUConnected course page you will find a description of Parva, a toy language very similar to C, and the language (or subset thereof) for which we shall develop a compiler and interpreter later in the course. The main difference between Parva and C/C++/Java is that Parva is stripped down to bare essentials. Learn the Parva system by studying the language description where necessary, and trying the system out on the supplied code (`SIEVEPAV.PAV, FIBOPAV.PAV` and `EMPTYPAV.PAV`).

There are various ways to compile Parva programs. The easiest is to use a normal command-line command:

```
Parva SievePAV.pav      simple error messages
Parva -o FiboPAV.pav    slightly optimized code
Parva -l SievePAV.pav   error messages merged into listing.txt
```

Note: Some of the Parva code you have been given has some deliberate errors, so you will have to find and correct these before you can compile and run the programs. Also, the Parva compiler you have been given may not support the whole of the Parva language – if it did, we wouldn't have anything to do in this course.

Hand in the source code of your final corrected `SievePAV.pav` *file.*

Task 6 High level translators [5 marks]

It may help amplify the material we are discussing in lectures if you put some simple Parva programs through a high-level translator, and then look at and compile the generated code to see the sort of thing that happens when one performs automatic translation of a program from one high-level language to another.

We have a home-developed system that translates Parva programs into Java. The system is called `Parva2ToJava`. It is still under development -- meaning that it has some flaws that we might get you to repair in a future practical. Much of the software for this course has been designed expressly so that you can have fun improving it.

You can translate a Parva program into Java using a command of the form:

```
Parva2ToJava SievePAV.pav
Parva2ToJava FiboPAV.pav
Parva2ToJava EmptyPAV.pav
```

A Java source file is produced with an obvious name; this can then be compiled with the Java compiler by using commands of the form:

```
javac SievePAV.java
```

and executed with the usual commands of the form:

```
java SievePAV
```

Take note of, and comment on, such things as the kind of Java code that is generated (is it readable; is it anything like you might have written yourself), and of the relative ease or difficulty of using such a system.

Submit a set of T-diagrams that shows what happens when executing the `Parva2ToJava SievePAV.pav` *command only*.

Task 7 Reverse engineering – disassembly

In lectures you were told of the existence of decompilers -- programs that can take low-level code and attempt to reconstruct higher level code. There are a few utilities available for experiment.

`jad`	a decompiler that tries to construct Java source from Java class files
`javap`	a disassember that creates pseudo assembler source from a Java class file
`gnoloo`	a disassember that creates JVM assembler source from a class file
`oolong`	an assembler that creates Java class files from JVM assembler source

Try out the following experiments or others like them:

(a) After compiling Sieve.java to create Sieve.class, decompile this:

```
jad Sieve.class
```

and examine the output, which will appear in `Sieve.jad`

(b) Disassemble Sieve.class

```
javap -c Sieve >Sieve.jvm
```

and examine the output, which will appear in `Sieve.jvm`

(c) Disassemble Sieve.class (in a different way)

```
gnoloo Sieve.class
```

and examine the output, which will appear in `Sieve.j`

(d) Reassemble Sieve.j

```
oolong Sieve.j
```

and try to execute the resulting class file

```
java Sieve
```

(e) Be malicious! Corrupt `Sieve.j` - simply delete a few lines in the section that corresponds to the Main function (use lines with opcodes on them). Try to reassemble the file (as above) and to re-run it. What happens?

There is nothing to hand in for this task but you may be asked questions on what was observed in the weekly test.

Time to do some coding yourself

And now for something completely different -- and please don't use a search engine for these tasks!

Problems for this course are sometimes reputed to be hard. They only get very hard if you don't think very carefully about what you are trying to do, and they get much easier if you think hard and spend time discussing the solutions with the tutors or even the lecturer herself. Don’t be fooled by glitzy environments and think that programs just "happen" if you can guess what to click on next. Don't just go in and hack. It really does not save you any time, it just wastes it. Each of the refinements can be solved elegantly in a small number of lines of code if you think them through carefully before you start to use the editor, and I shall be looking for elegant solutions.

Remember the crucial theme of this course as introduced by the originator of this course, Prof. Terry, many years ago: "*Keep it as simple as you can, but no simpler*".

In doing tasks 8 and 9, it is important that you learn to use the Java IO libraries `InFile`, `OutFile` and `IO` and set libraries `Intset` and `SymSet` provided in the library directory of the prac kit. These libraries will be used repeatedly in this course. Please do not use other methods for doing I/O or creating sets, or spend time writing lots of exception handling code.

Note: If you do not make use of these IO and SET libraries, your marks for the next two tasks will be close to zero!

Task 8 Another look at the infamous Sieve in Java [10 marks]

You will need to become acquainted with various library classes to solve the next two tasks, which are to be done in Java, and are designed to emphasize some useful techniques.

Descriptions of the relevant I/O library routines can be found on the course website, and a simple sample program using some of the library routines can be found in the kit as the program SampleIO.java (listed below). Note the precautionary error checking.

The code you have been given for the Sieve of Eratosthenes makes use of a Boolean array. A little thought should convince you that conceptually it is using this array as a "set". In the weeks ahead we shall use the set concept repeatedly, so to get you into this frame of mind, develop a Java version of the Sieve idea using objects of the `IntSet` type, details of which can be found on the RUConnected site - and some simple examples of use can be found in the program at the end of this handout. You should find this very simple, once you get the idea.

Submit your Java code on RUConnected.

Task 9 How long will my journey take? [10 marks]

A local bus service (like those in big cities such as Makhanda) links a large number of stops. Suppose we are given the travel time from any one stop to the next (assume that this time would be the same for return journeys) and that this information is captured in a long list of times given in minutes, with the stop names (abbreviated if necessary to 8 letters) between them. For example, if we had 8 stops we might have a list like

```
College  8  Hamilton  5  GreatHal  12  Gino's  25  Mews 9  Union  12  Steers 17  Athies
```

Viv's Bus Service wants to put up a poster at each stop from which one can easily determine the total travel time between any one stop and any other one. For the data given this might look like this (see file BUSES):

	College	Hamilton	GreatHal	Ginos	Mews	Union	Steers	Athies
College	0	8	13	25	50	59	71	88
Hamilton	8	0	5	17	42	51	63	80
GreatHal	13	5	0	12	37	46	58	75
Ginos	25	17	12	0	25	34	46	63
Mews	50	42	37	25	0	9	21	38
Union	59	51	46	34	9	0	12	29
Steers	71	63	58	46	21	12	0	17
Athies	88	80	75	63	38	29	17	0

Write a program to create such a table. Use a suitable data type (ArrayList perhaps?) to store the original data - you will need a small auxiliary data class to record the successive pairs of names and

travel times - and then set up a two-dimensional matrix to contain the computed values (note that this matrix will be symmetric).

Submit your Java code on RUConnected.

Demonstration program showing use of InFile, OutFile and IntSet classes

This code is to be found in the file `SampleIO.java` in the prac kit. This code uses the `primes.txt` file (given in the kit) as input and writes the result to a file of your choosing.

Use the commands below to run `SampleIO.java`:

```
javac SampleIO.java
java SampleIO primes.txt res.txt
```

```
// Program to demonstrate Infile, OutFile and IntSet classes
// P.D. Terry, Rhodes University

   import library.*;

   class SampleIO {
     public static void main(String[] args) {
   //                                          check that arguments have been supplied
       if (args.length != 2) {
         IO.writeLine("missing args");
         System.exit(1);
       }
   //                                          attempt to open data file
       InFile data = new InFile(args[0]);
       if (data.openError()) {
         IO.writeLine("cannot open " + args[0]);
         System.exit(1);
       }
   //                                          attempt to open results file
       OutFile results = new OutFile(args[1]);
       if (results.openError()) {
         IO.writeLine("cannot open " + args[1]);
         System.exit(1);
       }
   //                                          various initializations
       int total = 0;
       IntSet mySet = new IntSet();
       IntSet smallSet = new IntSet(1, 2, 3, 4, 5);
       String smallSetStr = smallSet.toString();
   //                                          read and process data file
       int item = data.readInt();
       while (!data.noMoreData()) {
         total = total + item;
         if (item > 0) mySet.incl(item);
         item = data.readInt();
       }
   //                                          write various results to output file
       results.write("total = ");
       results.writeLine(total, 5);
       results.writeLine("unique positive numbers " + mySet.toString());
       results.writeLine("union with " + smallSetStr
                         + " = " + mySet.union(smallSet).toString());
       results.writeLine("intersection with " + smallSetStr
                         + " = " + mySet.intersection(smallSet).toString());
       results.close();
     } // main

   } // SampleIO
```