RHODES UNIVERSITY
Where leaders learn

Computer Science 301

Programming Language Translation

Practical 1: week beginning 3 August 2020

This prac is due for submission through the RUConnected submission link by 2 pm on your next practical day.

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 C#, 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 C# are available 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 C#.

To hand in (30 marks)

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

- Electronic copies of your source code for tasks 3 [5 marks], 6 [10 marks], and 7 [10 marks].
- The completed *prac1_handin* after executing tasks 1-4 [5 marks].

Note, that not all your tasks will be marked each week. The tutors will mark some of the tasks and I will mark one other. The tasks marked by the tutors are labelled as such, whereas the task I will mark will not be known to you. For this practical, tutors will mark code for Task 3 and the prac1_handin only. I will mark either task 6 or task 7.

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 another student's work as your own. 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 allowed -- even encouraged -- to work and study with other students, but if you do this you are asked to acknowledge that you have done so with suitable comments typed into *all* listings.

Note however, that "working with" someone does not imply that you hand in the same solution. The interpretation used in this course (and which determines whether a submission is plagiarised or not) is that you may discuss how you would approach the problem, but not actually work together on one coded solution. You are still expected to hand in your own solution at all times.

You are expected to be familiar with the University Policy on Plagiarism, which you can consult at:

http://www.ru.ac.za/media/rhodesuniversity/content/institutionalplanning/documents/Plagiarism.pdf

---

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 short cut).
- You might also need a Visual Studio Developer Command Prompt window in which to execute the C# compiler. This won't be necessary if you set up all your environment paths correctly.
- Please DO NOT use or hand in Visual Studio PROJECT files for any of your code. Code submissions must be single files and not projects as all code will be tested from the command line.

Copies of software for home use

For this prac, you will be given URLs from where to download additional software, or some set up files and executables will be included in the prac kit provided on RUConnected. In future pracs you will mainly use C# only, and the prac kits will contain all the software you need.

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 `pracNN.zip` on RUConnected. Download `prac1.zip` as needed for this week and extract the sources when you need them, into your own directory/folder.

[**NOTE for those working in the Hamilton labs**: In the past there has occasionally been a problem with running applications generated by the C# compiler if these are stored on the network drives. If you have difficulties in this regard, for those parts of the practical that involve the use of C#, work from the local D: drive instead. 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 j:\xxxxx\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 C/C++

The kit includes C and C++ versions of the Sieve programs. Compile these using the Bloodshed Dev-C++ compiler. A setup file for installing this compiler is available on RUConnected in the Resources folder or it may be downloaded from: https://en.softonic.com/download/bloodshed-dev-c/windows/post-download

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)

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

To run the C program type:

`sievec` or `sievec.exe`

To run the C++ program type:

```
a.out
```

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 -- 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?

Something more challenging -- find out how large a prime number the program can really handle, given a limit of 32000 on the size of the Boolean array. What is the significance of this limit? How many prime numbers can you find smaller than 20000? *Hint*: you should find that funny things happen when the requested "largest number" N gets too large, although it may not immediately be apparent. Think hard about this one!

Task 2 See C#

To use C# on your own laptop/computer you will need to install various software. Please see the Appendix on how to do this. If you are using the Hamilton lab machines, no installation is needed.

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

```
     csharp SieveCS.cs
```

[NOTE for those running on their own laptop: if the above command does not work, it may be easier to open a “Developer Command Prompt for VS” and run the csharp command from there.

Also note that csharp.bat is a batch script file that comes in the prac kit. So you need to issue the csharp command from the same directory where you have unzipped the prac kit.]

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? What limit is there now to the largest prime you can find?

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 and 4). [5 marks]

Task 3 The new language on the block -- Parva [5 marks]

In the Software Resources folder on the RUConnected course page you will find a description of Parva, a toy language very similar to C, and the language for variations on which we shall develop a compiler and interpreter later in the course. The main difference between Parva and C/Java/C# 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`

Some of the Parva code you have been given has some deliberate errors, so you will have to find and correct these. All in a jolly afternoon's work! *Hand in a listing of your final corrected* `SievePAV.pav`.

Task 4 High level translators

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-brewed system that translates Parva programs into C#. The system is called `Parva2ToCSharp`. 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 C# using a command of the form:

```
Parva2ToCSharp SievePAV.pav
Parva2ToCSharp FiboPAV.pav
Parva2ToCSharp EmptyPAV.pav

Parva2ToCSharp HanoiIter.pav
Parva2ToCSharp HanoiRec.pav
```

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

```
csharp SievePAVp2c.cs
```

and executed with the usual commands of the form:

```
SievePAVp2c
```

Take note of, and comment on, such things as the kind of C# 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.

Another program in the kit (`voter.pav`) is a fairly intuitive program that determines who is eligible to vote. This has an intentional weakness. See if you can spot it!

Run the Parva compiler directly:

```
Parva voter.pav
```

Then try translating the program to C# and compiling and running that:

```
Parva2ToCSharp voter.pav
csharp voterp2c.cs
voterp2c
```

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

Task 5 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.

[NOTE if using your own laptop: running these commands requires the following to be installed:

.NET Framework 3.5 (includes .NET 2.0 and 3.0)

]

`ildasm`	a decompiler that creates CIL assembler source from a .NET assembly
`ilasm`	an assembler that creates a .NET assembly from CIL assembler source
`peverify`	a tool for verifying .NET assemblies

Try out the following experiments or others like them:

(a) Compile `SieveCS.cs` and then disassemble it

```
csharp SieveCS.cs
disassemble SieveCS
```

(calls `ildasm` from a batch file, produces `Sieve.cil`)

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

(b) Reassemble `SieveCS.cil`

```
reassemble SieveCS
```

(calls `ilasm` from a batch file, produces new `Sieve.exe`)

and try to execute the resulting class file

```
SieveCS
```

(c) Be malicious! Corrupt `SieveCS.cil` -- 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?

(d) Experiment with the .NET verifier after step (b) and again after step (c)

```
NetVerify SieveCS
```

(calls `peverify` from a batch file)

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 get beguiled 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 6 and 7, it is important that you learn to use the C# IO libraries `InFile, OutFile` and `IO` and set libraries `Intset` and `SymSet` given in the `Library.cs` file. 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 will be substantially lower.

Task 6 Another look at the infamous Sieve in C# [10 marks]

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 C# 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 C# code on RUConnected.

Task 7 Time for some more code – Goldbach's conjecture in C# [10 marks]

Goldbach's conjecture is that every even number greater than 2 can be expressed as the sum of two prime numbers. Write a program that examines every even integer `N` from 4 to `Limit`, attempting to find a pair of prime numbers (A, B) such that `N = A + B`. If successful the program should write N, A and B; otherwise it should write a message indicating that the conjecture has been disproved. This might be done in various ways. Since the hidden agenda is to familiarize you with the use of a class for manipulating "sets", you must use the variation on the sieve method suggested by the code you have already seen: create a "set" of prime numbers first in an object of the IntSet class, and then use this set intelligently to check the conjecture.

Submit your C# code on RUConnected.

Demonstration program showing use of InFile, OutFile and IntSet classes

This code is to be found in the file `SampleIO.cs` in the prac kit.

```
// Program to demonstrate Infile, OutFile and IntSet classes

// P.D. Terry, Rhodes University, 2016

using Library;
using System;

class SampleIO {

  public static void Main(string[] args) {
  //                                 check that arguments have been supplied
    if (args.Length != 2) {
      Console.WriteLine("missing args");
      System.Environment.Exit(1);
    }
  //                                 attempt to open data file
    InFile data = new InFile(args[0]);
    if (data.OpenError()) {
      Console.WriteLine("cannot open " + args[0]);
      System.Environment.Exit(1);
    }
  //                                 attempt to open results file
    OutFile results = new OutFile(args[1]);
    if (results.OpenError()) {
      Console.WriteLine("cannot open " + args[1]);
      System.Environment.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
```

APPENDIX

Installing C# on your own computer

(NB. It is crucial that you get C# running on your computer as this is what we will be using for the rest of the course)

1. Download and install Visual Studio Community Edition
   https://visualstudio.microsoft.com/vs/community/

2. Check that Microsoft .NET framework 3.5 or later has also been installed (This is a windows feature that can be turned on)
   https://www.microsoft.com/en-za/download/details.aspx?id=21

3. Ensure that the .NET framework version is in your environment path (although if this is not the case and you do not know how to add it you can always just open a VS Developer Command Prompt Window, and you will be able to run the csharp compiler from there).