Practical 1

Louise Poole, Martin Simango, Matt Doherty

Task 2

Sievepas.exe – 62 797 bytes

Fibopas.exe – 61 487 bytes

Emptypas.exe – 57 334 bytes

How do the sizes of the executables compare?

Sievepas.exe is the biggest, followed by fibopas.exe and lastly emptypas.exe.

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

Simply successfully compiling a Pascal program, even if it is skeleton code with no actual coding inside, uses a minimum amount of space to create the executable. This is because all executables have ‘unseen’ metadata attached to the file that specify requirements for it to be run (which operating system it was compiled for etc.).

How does the algorithm work?

Creates an array of all the numbers up to the inputted maximum number, then starts at the number 2, ‘crosses out’ (removes) all of its multiples from the array and then prints the number (2). It then repeats with the next number in the array (which we know is a prime as all multiples for the values smaller than it have already been removed) until there are no more values to check in the array (only primes are left in the array).

Why is it deemed to be particularly efficient?

Instead of checking every single number up to the inputted maximum number and trying to prove it has no factors, this method starts with the smallest known prime and removes all multiples of it (as those aren’t primes) which makes the list of numbers it needs to check smaller. Repeating this as it progresses through the list (as we know the next number in the list is always a prime as all multiples for the values smaller than it have already been removed) makes the list it’s checking accumulatively smaller every time. This makes it very efficient.

How much (mental) arithmetic does the "computer" have to master to be able to solve the problem?

All the computer is required to compute is multiplication and removing values from an array. It’s a simple program to execute.

What is the significance of this limit?

The highest index an array can have is 32000.

How many prime numbers can you find smaller than 20000?

1900 primes (16410 was the maximum input we could enter)

Task 3

Sievec.exe – 66 560 bytes

Sievecpp – 149 504 bytes

Can you think of any reason why the differences are as you find them?

C++ is a much more complex and higher level language compared to Pascal and even simply C and so it’s compiled file could and does easily result in it requiring more memory. Simpler languages are easier to translate to lower level languages and machine code and so the resulting machine code will most likely be fairly simple too.

How large a prime number can you handle now?

Up to 32 000

How many prime numbers can you find smaller than 20000? If there is a difference, explain it.

1900. The difference is that Pascal uses 16 bit integers and so crashes as soon as any value it calculates cannot be stored in a 16 bit integer. C and C++ use 32 bit integers and so can store much larger values.

Task 4

Sievecs.exe – 35 840 bytes

Sieveset.exe – 35 840 bytes

Sieveset2.exe – 35 840 bytes

Emptycs.exe – 35 328 bytes

Fibocs.exe – 35 328 bytes

How do these compare with the other executables?

They are all notably much smaller.

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

Can test up to 32000

Task 6

TP6

Sievepas.exe – 3 248 bytes

Fibopas.exe – 2 640 bytes

Emptypas.exe – 1472 bytes

TP6O

Sievepas.exe – 3 136 bytes

Fibopas.exe – 2 640 bytes

Emptypas.exe – 1472 bytes

Differences between these compilers and FPC(Free Pascal)?

The file size produced are significantly smaller for all files. This is most likely due to the more efficient TP6 and TP6O compilers which were developed about 20 years after the Free Pascal compiler.

Task 7

Is the translated Parva code readable and is it anything like you might have written yourself?

Yes the new C# source file is readable and is extremely similar to the Parva code and very much looks like something I could have written.

Task 8

Do you suppose Parva programs need to be acceptable to the Parva compiler if they are to be acceptable to Parva2ToCSharp?

Ummm yes…

What can you learn from these exercises about using a tool of this nature?

Have we made Parva2ToCSharp "as simple as possible, but no simpler"?   
Probably…

Do we have to, or could we, make it simpler still? Do we have to make it more complex? Why -- or why not?

Task 10

(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?

Program crashed even before input

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

NetVerify SieveCS (calls peverify from a batch file)

After step c) (deleting lines in Main) gave me 1 error of a stack overflow.   
After step b) gave me 1 error with verifying the SieveCS.exe by saying “The assembly is built by a runtime newer than the currently loaded runtime, and cannot be loaded”

Task 11

Rerun the executables, do they still work?

Fibo2.exe works perfectly  
Sieve2.exe crashes just before it outputs how many primes it found.

What happens if you try to decompile an executable that was not produced from a .NET compatible compiler? Try it.

Dotpeek does not decompile the programs compiled by the BCC and FPC compilers because the compilers are unsupported