FIT2102 Tutorial Worksheet - WEEK 1

MARKING RUBRIC AND SOLUTIONS

Process:

- *At the start of class, visit each student in turn*
- *Check that they made a moodle submission (you don't have to test it)*
- *have them walk you (quickly) through what they have done.*
- *spend no more than 2-3 minutes with each and*
- *If they need more help, come back and see them later or refer them to me (my consultation time is Tuesday 10-12 as per the Moodle table).*
- *Give a Mark out of 10 based on your 2-3 minute interview with each student.*
- *Be generous, give points for effort!*

Example grades:
10: All questions attempted, student demonstrates a good understanding of the material
9 - 6: Student has skipped questions entirely, deduct a mark for each question for which they have nothing to show

Below 6:
Student has not attempted most questions, and has little understanding of the material. Give them generous sympathy marks for whatever they have done (we don't want them to stop coming to tutes entirely) and tell them to put in more effort next time.

If they have not made a Moodle submission maximum mark is 5 for whatever they are able to demonstrate to you.

Solutions are in red below.

Each week we will complete a set of exercises that complement the material covered in lectures. Complete the activities below either in class, or if you run out of time then at home.

Complete the exercises below, compress the VS project directory into a single zip file, and submit it on Moodle.

You also will need to show your code and be able to adequately explain it to your tutor at the start of the following week's tute in order to receive a mark for the tutorial participation.

MASM Assembler and C

Learning Outcomes:

- Explain the need for abstraction from machine instructions to high-level languages
- Explain how assembly instructions work with registers and memory to perform computation
- Explain how assembly programs are structured into subroutines through jumps
- Create a basic x86 assembly program to perform a computation

Tasks:

1. If you have not already done so, then please install Visual Studio 2015 following these instructions.

2. Download the MASM starter project. Unzip it onto your local drive (do not use a shared drive) and open in VS.

3. Make sure it compiles and runs. If the program runs successfully the final message in the Debug output will be: "...exited with code 5 (0x5)".

FYI: Working through the problems at Project Euler is a fantastic way to learn a new language. The problems start off easy and gradually get more difficult. If you create an account on the site, when you enter the correct solution to a problem it will give you access to forum where people discuss the solutions to that problem in lots of different programming languages.

4. Project Euler Problem 1 reads:

   If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

   Find the sum of all the multiples of 3 or 5 below 1000.

   Go ahead and create a program based on the starter project, that solves this problem in MASM. To test whether a number is divisible by 3 or 5, create a procedure called IsDivisibleBy that checks if eax is evenly divisible by ebx using the div operator.

   In developing your code, make sure to try out the Visual Studio debugger to step through

the assembler line by line. See the values of registers change and set up watches for the variables you define in the .data block.

There are several different solutions below:

```
.386
.model flat,stdcall
.stack 4096
ExitProcess PROTO, dwExitCode:DWORD

.data
sum dword 0
max dword 999
.code

;-----------------------------------------------------
; sum [x|x<-[1..999],(x `mod` 3==0) || (x `mod` 5==0)]
;-----------------------------------------------------
IterateAndTest PROC
                mov     ecx,max ; ecx is our counter, start it at 999
L1:             mov eax,ecx ; loop starts, a=c
                mov ebx,3   ; b=3
                mov edx,0   ; zero d
                div ebx             ; a=a/b, remainder in d
                cmp edx,0;  ; if d==0 goto IS3OR5
                je IS3OR5
                mov eax,ecx ; a=c
                mov ebx,5   ; b=5
                mov edx,0   ; zero d
                div ebx             ; a=a/b, remainder in d
                cmp edx,0           ; if d==0 goto DONE
                jne DONE
IS3OR5:  add sum,ecx ; sum+=c
DONE:    loop L1            ; decrement ecx and go to L1
                ret
IterateAndTest ENDP

;-----------------------------------------------------
; sum [3,6..999] + sum [5,10..999] - sum [15,30..999]
;-----------------------------------------------------
SumOfSums PROC
        mov eax,0
loop3:
        add eax,3
        add sum,eax
        cmp eax,max
        jl      loop3

        mov eax,0
loop5:
        add eax,5
        cmp eax,max
        jg      endloop5
        add sum,eax
        jmp loop5
endloop5:
```

```
        mov eax,0
loop15:
        add eax,15
        cmp eax,max
        jg      endloop15
        sub     sum,eax
        jmp loop15
endloop15:
        ret
SumOfSums ENDP

;-----------------------------------------------------
; is eax evenly divisible by ebx?
; return 0 in eax if true
;-----------------------------------------------------
IsDivisibleBy PROC
                push edx  ; store edx
                xor edx,edx ; zero edx
                div ebx             ; eax=eax/ebx, remainder in edx
                mov eax,edx ; result in eax (by convention)
                pop edx             ; restore edx
                ret
IsDivisibleBy ENDP

;-----------------------------------------------------
; sum [x|x<-[1..999],(x `mod` 3==0) || (x `mod` 5==0)]
;-----------------------------------------------------
IterateAndTestWithSubroutine PROC
                mov     ecx,max ; ecx is our counter, start it at 999
L1:             mov eax,ecx ; loop starts, a=c
                mov ebx,3   ; b=3
                call IsDivisibleBy
                cmp eax,0;  ; if b evenly divisible by a then goto IS3OR5
                je IS3OR5
                mov eax,ecx ; a=c
                mov ebx,5   ; b=5
                call IsDivisibleBy
                cmp eax,0           ; if d==0 goto DONE
                jne DONE
IS3OR5:  add sum,ecx ; sum+=c
DONE:    loop L1            ; decrement ecx and go to L1
                ret
IterateAndTestWithSubroutine ENDP

main PROC
                call SumOfSums ; IterateAndTestWithSubroutine
                INVOKE ExitProcess,sum
main ENDP
END main
```

5. Right-click on your .asm file in the VS Solution Explorer and select “Exclude from Project”. Right-click on the project file and choose “Add->New Item” and create a new

C++ file. Recode your solution to Project Euler Problem 1 in a simple vanilla C function, and create a main function that invokes it and returns the result.

```
unsigned EulerProblem1(const unsigned n) {
	unsigned sum = 0;
	for (unsigned i = 0; i < n; i++) {
		if (i % 3 == 0 || i % 5 == 0) {
			sum += i;
		}
	}
	return sum;
}
int main() {
	return EulerProblem1(1000);
}
```

6. View and use the debugger to step through the disassembly for your C program. Observe how the stack frame is set up, compare your disassembly to that listed in the course notes. Copy paste the disassembly into a text document and document line-by-line as in the course notes. Be ready to explain this code to your tutor at the start of next week's tute.

See course notes page 12

7. Compile the above code in release mode and look at the disassembly carefully in the debugger. What are the differences?

It might inline the function
There may be less useless boiler plate
The code may look completely different because of optimisations

8. Write a simple recursive C function to compute the sum of positive integers up to a given *n*.

```
int sumInts(int n) {
	return 0;
	return n + sumInts(n - 1);
}
```

9. What's the biggest *n* you can specify before the stack overflow crash occurs (make sure you are running in debug mode)?

Depends!! For me it was 995210

10. If the function is not already tail-recursive then rearrange it to make it so (the recursive call must be the very last operation in the function). The tail-recursive version will still crash in debug mode for large n, but try it again in Release mode. What happens? Why?

```
int sumIntsTailRec(int a, int n) {
        if (n == 0)
                return a;
        return sumIntsTailRec(a + n, n - 1);
}
```

In release mode the compiler recognises it's tail recursive and replaces it with a loop so the stack never overflows.