Midterm Study Overview

This is what you need to know for the midterm

1. C++ Programming
    a. know integer data types (int, uint8_t, uint16_t, uint32_t, uint64_t, same for int_8_t, ...)
    b. Know the 5 integer operations: + - * / %
    c. Know order of operations and type promotion
    int a = 2 / 3;
    int b = 7 / 4;
    uint16_t c = 32767;
    c += 3; // know overflow. You don't have to memorize -32768..32767 but I would like it if you did. Remember on the test we will list somewhere the relevant numbers. You do have to know that the negative one is one more than the positive.
    uint32_t d = 9.0 / 1.0; // know that integer truncates → 0.9 → 0.0
    uint32_t e = 1 / 2.0 * 3; // know that when there are two different types, the "smaller" promotes. In this case 1→ 1.0, and the operation is done in double precision 1.0/2.0 which is 0.5, then multiply by 3 = 1.5, then truncate (1).
    d. Be able to write the following constructs
        i. for loop

```
for (int i = 0; i < n; i++)
  cout << i;

for (;;)
  cout << "forever";
for (uint16_t i = 0; i < 100000; i++)
  cout << "forever"; // understand why this is forever
```

        ii. while loop

```
while (true)
  cout << "forever";

int i = 0;
while (i < 10) {
  cout << i;
  i++;
}

int i = 10;
while (i < 10) { // understand that a while loop can execute zero times
  cout << i;
```

```
    i++;
  }
```

iii. if

```
  if (a < 2) {
    cout << a;
  }
```

iv. if … else

```
  if (a < 2)
    cout << a;
  else
    a += 3;
```

e. Variables

Variables declared in a function are on the stack (or in a register if the compiler is good). They are not initialized by default and so are random unless initialized.

Variables declared outside a function are global and visible everywhere in the program, even in a different file. We learned .global in assembler. Same concept.

f. Functions.

Know how to pass values by value (the normal way, copy) and by reference.

```
void f(int x) {
  cout << x;
  x *= 2; // makes no difference, this is our copy
}

void g(int& r) {
  cout << r;
  r *= 2; // changes whichever variable is passed in
}
```

g. Arrays

A single block of contiguous memory
Only a single type
First element has position 0
Last element has position size - 1

```
int x[10]; // uninitialized if in a function
```

```
int x[10] = {3}; // first element = 3, all the rest are 0
int x[] = {5, 4, 3}; // size is 3 elements, x[0] = 5, x[1] = 4, x[2] = 3;
```

2. C++ Skills
   You should be prepared to write code similar to what we built in lab as a team. Some are a bit hard, so I am listing out the ones you should know. Note that this does not mean the test will include this code, but that you should be able to write functions to return answers like these.

1. sum(a,b)	return the sum of the integers from a to b inclusive
01. prod(a,b)	return the product of the integers from a to b inclusive
01. sumsq(a,b)	return the sum $a^2+(a+1)^2+...+b^2$
01. isPrime(a)	return true if a is prime, false otherwise
01. countPrimes(a,b)	return the number of primes found between a and b inclusive
01. gcd(a,b)	return the greatest common divisor of a and b. For example gcd(12,18) = 6, gcd(15,13)=1
01. hypot(a,b)	Given two sides of a triangle, compute the third side
01. trigIdentity(x)	We know that $\sin^2x + \cos^2x = 1$. Compute it for an actual x and see what it really returns on a computer. (hint, might not be 1 for all values.)
01. diffsq(a, b)	return $a^2 - b^2$
01. mean(a,b)	return the average of a and b
01. mean(a,b,c)	return the average of a,b, and c
01. min(a, b)	Return the smaller of the two, a and b.
01. isEven(a)	Return whether or not a is even
01. max(a,b)	Return the larger of the two numbers.
01. mean(x,n)	Return the mean of an array (x) of doubles of size (length) n.

01. max(x,n)	Return the maximum value contained in array (x) of size n.
01. min(x,n)	Return the minimum value contained in array (x) of size n.
01. prod(x,n)	Return the product of all the values contained in array (x) of size n.
01. sum(x,n)	Return the sum of all the values contained in array (x) of size n.

3. ARM Assembler
   Know the following features of the assembler we are using

```
.global xxx     specifying that a symbol is global
xxx:            defining a label


mov     r0, #245
mvn     r1, #186
ldr     r0, =0xFF4080c2

add     r0, #2          @ immediate mode arithmetic instruction
sub     r0, r1, r2      @ 3-operand instruction
and     r2,r3,r1        @ bitwise AND
orr     r2,r3,r1        @ inclusive or (OR)
eor     r2,r3,r1        @ exclusive or (XOR)
lsl     r3, #5          @ the number can be 0 to 31
lsr     r3, #12         @ the number can be 0 to 31

cmp     r0,r1
cmp     r0, #21         @immediate mode compare
bcc                     @ branch, where the condition is gt/ge/le/lt/ne/eq ....
bl                      @ branch and link (call subroutine) know what happens
                        @ with the PC (program counter) and link register LR
bx lr                   @ return from subroutine. Know what happens to PC and LR
```

4. ARM Fastcall convention
   a. For subroutines, the convention is that the first parameter is in R0, the second in R1, etc for integer parameters or pointers to int.

b. Subroutines are allowed to do whatever they want to r0, r1, r2, r3.

c. For registers r4 and above, before you can use them you must store the current value. Know how to use the stack.

d.
```
push    {r4}      @know what happens to the stack pointer sp
mov     r4, #28 @ do whatever you want with r4 once it is saved
pop     {r4}      @know what happens to the stack pointer sp
```

5. Be prepared to write simple loops in assembler like you did in lab (pass in R0, count up to r0, or down down to 0.

6.
```
ldr  r0, x
ldrb r1, [r0]
add r0, #1      @ advance r0 pointer by 1 byte

ldrb r2, [r0]   @ load r2 with the byte at memory location r0
add r2, #1
strb r2, [r0]   @store the value of r2 which is one bigger than it was before.
```