1 C

1.4 Word Count - Some Shell Commands

```
% ./wcount
```

Executes the file "wcount".

```
% ./wcount < wcount.c
```

Executes the file "wcount" with "wcount.c" as a standard input.

```
% ./wcount < wcount > test
```

Executes the file "wcount" with "wcount" as a standard input and "test" as a standard output.

```
% cat wcount.c | ./wcount
```

We use "cat" to display the content of "wcount.c" and insert it as a standard input after the execution of "wcount" with the help of a pipe.

```
% grep { wcount.c
```

Looks for the character "{" in the file "wcount.c" and display every line containing this character.

```
% grep { wcount.c | ./wcount
```

Same as above but we use the output of the first expression as the input of the second expression.

```
% grep -l { * | ./wcount
```

The -l option allow to print only the names of the files in which the regular expression matched. The asterisk character allow us to pass the expression in each files included in the present directory. After listing all the files, in which the '{' character appears at least once, the list is then processed by "wcount" as a standard input.

1.5 Type Conversion, Casting and ASCII

```
A 65
A 65
3.140000 3
```

On the first line, the character 'A' is first displayed as a character and then as an integer. On the second line, the integer 65 is first displayed as a character and then as an integer.

On the last line, the float "pi" is first displayed as a float and then as an integer, leaving its decimal part.

1.6 Constant, Variable, Escape Character '\', and Octal resp. Hexadecimal Digits

```
@ 64 100 40
@ 64 100 40
@ 64 100 40
```

All of the variables ('@', at and AT), are essentially of the same value simply written in different format. So it is expected for them to display the same output if converted in a equivalent format.

1.7 enum Type

```
1 0
0 1 2
```

In the first line, FALSE and TRUE have not been assigned any value, therefore FALSE is equal to 0 and TRUE is equal to 1. Hence the fact that b is equal to 1 since it has been assigned the variable TRUE.
In the second line, the variables RED, GREEN and BLUE have not been assigned any value, therefore they respectively equal 0, 1 and 2. c1 is declared as RED, taking its value of 0. c2 is assigned "c1+1", making it equal to 1. c3 is assigned BLUE, thus making it equal to 2.

1.8 Logical Expression

```
p || !q
```

p	q	Y
0	0	1
1	0	1
0	1	0
1	1	1

```
p && (p == q)
```

p	q	Y
0	0	0
1	0	0
0	1	0
1	1	1

```
p && (p = q) || (p = !q)
```

p	q	Y
0	0	1
1	0	1
0	1	0
1	1	1

1.9 Conditional Expression

```
2
```

The conditional expression, asks if x is smaller than y. If the answer is true, x is displayed, otherwise y is displayed. Since x is larger than y, the answer is false and y is displayed.

1.10 Bitwise operator

a) Let's take x as an infinite binary number with only ones. It is equivalent to the sum of an infinite number of power of two.

$$x = \sum_{n=1}^{\infty} 2^n$$

If we translate every bits to the left, we are simply adding 1 to their power.

$$x << 1 = \sum_{n=1}^{\infty} 2^{n+1}$$

$$x << 1 = \sum_{n=1}^{\infty} 2^n \cdot 2^1$$

$$x << 1 = 2 \cdot \sum_{n=1}^{\infty} 2^n$$

$$x << 1 = 2x$$

As for translating the bits to the right, we are simply subtracting 1 to their power.

$$x >> 1 = \sum_{n=1}^{\infty} 2^{n-1}$$

$$x >> 1 = \sum_{n=1}^{\infty} 2^n \cdot 2^{-1}$$

$$x >> 1 = 2^{-1} \cdot \sum_{n=1}^{\infty} 2^n$$

$$x >> 1 = 2^{-1} x$$

$$x >> 1 = x/2$$

b) Since the first bit will be gone due to the overflow, this will not result in a multiplication by 2 but rather give us an undesired result.

c) Here is the function.

```
1 unsigned long setbit(unsigned long x, int n){
2     unsigned long b = 1 << n;
3     return x | b;
4 }
```

1.11 Order of Evaluation

Compilers can interpret side effects in different ways. It might be f(n) or f(n+1) depending on the compiler.

PART 2. System

1.12 Memory

a)

Name	Value
i	8
c1	@
c2	A
s	@A

b)

Variables		Address	
Name	Value	Name	Possible
i	8	&i	0x00
c1	@	&c1	0x04
c2	A	&c2	0x08
s	@A	s	0x0C

c)

Address	Big Endian				Little Endian				Address
&i	0	0	0	8	0	0	0	8	&i
&c1	@	***	***	***	***	***	***	@	&c1
&c2	A	***	***	***	***	***	***	A	&c2
s	@	A	***	***	***	***	A	@	s
s+4	\0	***	***	***	***	***	***	\0	s+4

d)

Address	Big Endian				Little Endian				Address
0x00	0	0	0	0x08	0	0	0	8	0x00
0x04	0x40	***	***	***	***	***	***	0x40	0x04
0x08	0x41	***	***	***	***	***	***	0x41	0x08
0x0C	0x40	0x41	***	***	***	***	0x41	0x40	0x0C
0x10	0x00	***	***	***	***	***	***	0x00	0x10

1.13 scanf

a)
1. str = AA, i = 33, d = 2
2. str = 55, i = 33, d = 1
3. str = ZZ, i = 77, d = 2

b) Here is the program.

```
1 char c; int i; char str[10]; float f;
2 scanf("%c␣%i␣%s␣%f", &c, &i, str, &f);
3 printf("%c␣%i␣%s␣%f", c, i, str, f);
```