Coding Review (I2P 2019)

Standard Outputs of C v1.0

Review of common inputs: `printf`, `putchar`, `puts`.

Comparing to Standard Inputs, Standard Outputs are much simpler.

Basic Formats

1. What is the result of the following code and input?

```c
#include <stdio.h>

int main(void) {
    char a[5] = {'A', 'B', '\0', 'C', 'D'};
    puts(a);
    return 0;
}
```

Answer:

```
AB
```

`puts` or `printf("%s", ...)` stops when the terminating character (`'\0'`) is seen. If you forget to add a `'\0'` at the end of your string, the program might crash due to memory access violation.

2. Replace `<REPLACE_HERE>` with a string format that can get the expected output.

```c
#include <stdio.h>

int main(void) {
    printf("<REPLACE_HERE>");
    return 0;
}
```

Expected Output:

```
printf("%d\n", x);
```

Answer:

```
printf(\"%%d\\n\", x);
```

`printf` requires escape characters to print certain special characters.

- `\\` becomes `'\'`
- `\"` becomes `'"'`
- `%%` becomes `'%'`

For the example above, we can print out the expected output easily by `puts`. But if you use `printf`, it's a pain in the a**.

3. What is the result of the following code and input?

```
#include <stdio.h>

int main(void) {
    int a, b;
    a = 101;
    b = 8787887;
    printf("%8d\n", a);
    printf("%8d\n", b);
    printf("%08d\n", a);
    printf("%08d", b);
    return 0;
}
```

Answer:

```
     101
 8787887
00000101
08787887
```

This is a easy way to pad outputs with whitespaces or zeros.

4. What is the result of the following code and input?

```
#include <stdio.h>

int main(void) {
    float f;
    f = 878722e-4;
    printf("%f\n", f);
    printf("%.2f\n", f);
    printf("%.1f\n", f);
    printf("%.0f\n", f);
```

```
    printf("%.f", f);
    return 0;
}
```

Answer:

```
87.872200
87.87
87.9
88
88
```

The `xey` in float representation means $x \cdot 10^y$.

If the precision isn't specified, the default is 6 digits after decimal.

`printf` does the rounding for you.

5. Replace `<REPLACE_HERE>` with a string formats that can get the expected output.

```
#include <stdio.h>

int main(void) {
    long long x, y;
    scanf("%lld%lld", &x, &y);
    printf("<REPLACE_HERE>\n", 20, 2*(unsigned long long)x);
    printf("<REPLACE_HERE>\n", 20, 2*(unsigned long long)y);
    return 0;
}
```

Input:

```
9100000000000000000
12
```

Expected Output:

```
18200000000000000000
00000000000000000024
```

Answer:

```
%0*llu
```

- `%lld` for `long long`
- `%llu` for `unsigned long long`.

`printf("%*d", NUM, ...)` replaces `*` to `NUM`.

6. What is the result of the following code and input?

```
#include <stdio.h>

int main(void) {
    putchar('\a');
    return 0;
}
```

Answer:

Does not output any visible characters.

If your computer's sound is on, you should hear a bell ringing sound or a "beep!", or some other strange noises.

Standard I/O Review

1. `scanf("%c", ...)`, `getchar` does not ignore leading whitespace characters.
2. `gets` does not store the terminating newline character; `fgets` stores the terminating newline character (if the input is terminated by newline instead of `EOF`).
3. when reading `EOF`, `scanf`, `getchar` returns `EOF`; `gets`, `fgets` return `NULL`.
4. When reading strings, remember to save an additional space for the easily forgotten `'\0'`.
5. Strings should be null-terminated (end with `'\0'`) before outputting using `printf("%s", ...)` or `puts`.

The list above are some mistakes that I see a lot of beginners make. If you see other special usages, you can search for them online. (such as `%x`, `%#x`, `%hd`, ...)

If you forget some of the I/O formats above in your exam (such as leading zero paddings), most of them can be replaced with additional `if` statements and loops.

For the next assignment, we'll review some basic syntaxes of C.

Epilogue

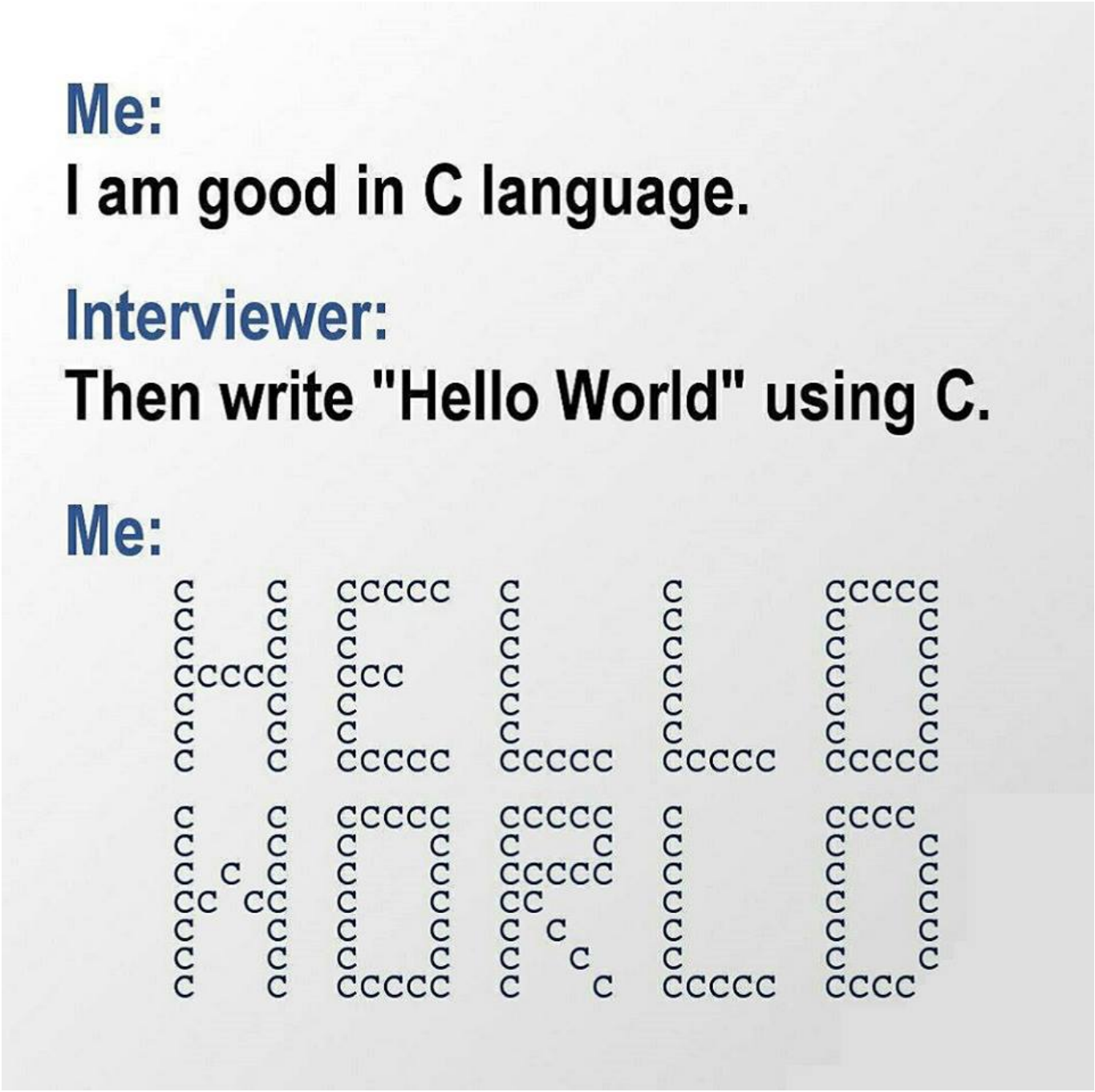

Photo Credit: Posted on Reddit

If there's any typo, please discuss on iLMS or email j3soon@gapp.nthu.edu.tw, I appreciate your help.