Coding Review (I2P 2019)

Standard Inputs of C

Review of common inputs: `scanf`, `getchar`, `gets` (`fgets`).

Basic formats

1. What is the result of the following code?

   ```
   #include <stdio.h>

   int main(void) {
       int a, b;
       scanf("%d%d", a, b);
       printf("%d %d", a, b);
       return 0;
   }
   ```

   Input:

   ```
   12 34
   ```

   Answer:

   Undefined Behavior.

   `scanf` requires **the memory address** of the variable.

2. What is the result of the following code?

   ```
   #include <stdio.h>

   int main(void) {
       int a, b;
       scanf("%1d%2d", &a, &b);
       printf("%d %d", a, b);
       return 0;
   }
   ```

   Input:

   ```
   1234 5678
   ```

Answer:

```
1 23
```

The %2d part means to read in 2 digits.

3. What is the result of the following code?

```c
#include <stdio.h>

int main(void) {
    int a, b;
    scanf("(%d,%d)", &a, &b);
    printf("%d %d", a, b);
    return 0;
}
```

Input:

```
(12,34)
```

Answer:

```
12 34
```

What if the input becomes:

```
(12, 34)
```

Answer:

```
12 34
```

If we specify what certain character to match in `scanf`, the input must input that specific character. For example, using `(` in `scanf` indicates the next character must be `(`, the call will fail if the next character is space or `\n`, ... However, using `%d`, `%f`, ... automatically ignores every whitespace characters (space, `\n`, ...) before the number digits. `%c` is an exception that does read in whitespace characters.

4. What is the result of the following code?

```
#include <stdio.h>

int main(void) {
    float f;
    double d;
    scanf("%f%f", &f, &d);
    printf("%f %f", f, d);
    return 0;
}
```

Answer:

Undefined Behavior.

The input format should be `%lf` for double variables, using `%f` will leave half of the variable's memory undefined.

Usages of `scanf("%c", ...)` and `getchar`

1. What is the result of the following code?

```
#include <stdio.h>

int main(void) {
    int i;
    char a[6];
    for (i = 0; i < 6; i++)
        scanf("%c", &a[i]);
    for (i = 0; i < 6; i++)
        printf("%d ", (int)a[i]);
    return 0;
}
```

Input:

```
A B
C
```

Hint: `'A'` is `65`, `' '` is `32`, `'\n'` is `10`.

Answer:

```
65 32 66 10 67 10
```

Using `%c` as the input format automatically reads in every character, including whitespace characters. To visualize `a`, it is actually `{'A', ' ', 'B', ' ', '\n', 'C'}`. For each variable, regardless its type, they are stored in binary format in the memory. Take `char` as example, the characters are stored by their ASCII code. It should be straightforward to see that: `'A' + ' ' is 'a'`. (`65 + 32 = 97`)

The `int` cast before printing out the character is used to extend the 1-byte variable to 4-bytes to avoid undefined behavior.

2. What is the result of the following code?

```c
#include <stdio.h>

int main(void) {
    int i;
    char a[6];
    for (i = 0; i < 6; i++)
        scanf(" %c", &a[i]);
    for (i = 0; i < 6; i++)
        printf("%d ", (int)a[i]);
    return 0;
}
```

Input:

```
AB C
DEF G
```

Hint: `'A'` is `65`, `' '` is `32`, `'\n'` is `10`.

Answer:

```
65 66 67 68 69 70
```

The space character before `%c` ignores all whitespace characters.

3. What is the result of the following code?

```c
#include <stdio.h>

int main(void) {
    int i;
    char a[6];
    for (i = 0; i < 6; i++)
```

```c
        a[i] = getchar();
    for (i = 0; i < 6; i++)
        printf("%d ", (int)a[i]);
    return 0;
}
```

Input:

```
AB C
DEF G
```

Hint: `'A'` is `65`, `' '` is `32`, `'\n'` is `10`.

Answer:

```
65 66 32 67 10 68
```

`c = getchar();` is same as `scanf("%c", &c);`.

Usages of `scanf("%s", ...)`

1. What is the result of the following code?

```c
#include <stdio.h>

int main(void) {
    int i;
    char a[5] = {-1, -1, -1, -1, -1};
    scanf("%s", a);
    for (i = 0; i < 5; i++)
        printf("%d ", (int)a[i]);
    return 0;
}
```

Input:

```
 ABC DE
FG
```

Hint: `'A'` is `65`, `' '` is `32`, `'\n'` is `10`.

Answer:

```
65 66 67 0 -1
```

The value of an array variable is its memory address. So using `scanf("%s", a)` should be enough, of course, using `scanf("%s", &a)` is also OK. Using `%s` as the input format automatically ignore whitespace characters (like `%d`). It'll continuously read input characters until a whitespace character, and then add a `'\0'` (ASCII code is 0) character indicating the termination of the string. The values after `'\0'` aren't affected.

2. What is the result of the following code?

   ```c
   #include <stdio.h>

   int main(void) {
       int i;
       char a[3];
       scanf("%s", a);
       for (i = 0; i < 3; i++)
           printf("%d ", (int)a[i]);
       return 0;
   }
   ```

   Input:

   ```
   ABC
   ```

   Hint: `'A'` is `65`, `' '` is `32`, `'\n'` is `10`.

   Answer:

   Undefined Behavior. (May result in segmentation fault)

   After reading `ABC`, `scanf` adds an additional `'\0'`. However, `a[3]` shouldn't be accessed in this case. This is the notorious Null-terminated string in C, that can easily lead to many bugs.

`gets` vs. `fgets`

1. What is the result of the following code?

   ```c
   #include <stdio.h>

   int main(void) {
       int i;
       char s[6] = {-1, -1, -1, -1, -1, -1};
       fgets(s, 6, stdin);
       for (i = 0; i < 6; i++)
   ```

```
        printf("%d ", s[i]);
    return 0;
}
```

Input:

```
ABCDEF
```

Hint: `'A'` is `65`, `' '` is `32`, `'\n'` is `10`.

Answer:

```
65 66 67 68 69 0
```

What if the input becomes:

```
ABC
```

Answer:

```
65 66 67 10 0 -1
```

`fgets` read in the ending newline character, and so are other whitespace characters, including the leading whitespaces!

2. What is the result of the following code?

```
#include <stdio.h>

int main(void) {
    int i;
    char s[6] = {-1, -1, -1, -1, -1, -1};
    gets(s);
    for (i = 0; i < 6; i++)
        printf("%d ", s[i]);
    return 0;
}
```

Input:

```
ABC
```

Hint: `'A'` is `65`, `' '` is `32`, `'\n'` is `10`.

Answer:

```
65 66 67 0 -1 -1
```

What if the input becomes:

```
ABCDEF
```

Answer:

Undefined Behavior. (May result in segmentation fault)

`gets` is like `fgets` but does not read in the newline character. `scanf("%[^\n]", s)` works the same as `gets` but the syntax isn't really easy to remember.

`scanf`'s return value

1. What is the result of the following code?

```c
#include <stdio.h>

int main(void) {
    int a, b, ret;
    a = b = -1;
    ret = scanf("%d%d", &a, &b);
    printf("%d %d %d", ret, a, b);
    return 0;
}
```

Input:

```
100 abc 1000
```

Answer:

```
```

```
1 100 -1
```

The return value of `scanf` indicates how many variables does it read successfully. Take the above code and input as example, it reads in `100` and store it into `a`, then it sees `abc`, which is not a number, so the matching fails. The value of `b` isn't modified due to the failed matching. In C, the return value of many library function indicates whether that function ran without error.

2. What is the result of the following code?

```c
#include <stdio.h>

int main(void) {
    int a, b, ret;
    a = b = -1;
    ret = scanf("(%d,%d)", &a, &b);
    printf("%d %d %d", ret, a, b);
    return 0;
}
```

Input:

```
( 1 , 2 )
```

Answer:

```
1 1 -1
```

For `,`, whitespace characters are invalid, so the match fails.

End of File `EOF`

`EOF` is defined to be `-1` in most cases. When the input ends, using `scanf`, `getchar` result in `EOF`.

For `gets` and `fgets`, they returns `NULL`, which is `0` in most cases.

To simulate `EOF`, press `Ctrl+Z` and `Enter` on Windows; press `Ctrl+D` and `Enter` on MacOS or Linux.

For some common usages:

```
while (scanf("%d", &x) != EOF) {
    // Do something.
}
while (scanf("%d%d", &a, &b) == 2) {
    // Do something.
}
while ((c = getchar()) != EOF) {
    // Do something.
}
// Change 'MAX_STRLEN' to a constant (max string length including null-terminator)
while (fgets(s, MAX_STRLEN, stdin) != NULL) {
    // Do something.
}
```

Case Study

There's often multiple ways to read a input.

If we want to read in each of the digits of a 5-digit number:

```
12345
```

And store them into 5 variables `a`, `b`, `c`, `d`, `e` respectively.

1. Math

   ```
   int n;
   scanf("%d", &n);
   a = n / 10000 % 10;
   b = n / 1000 % 10;
   c = n / 100 % 10;
   d = n / 10 % 10;
   e = n / 1 % 10;
   ```

2. Scan 1-digit at a time

   ```
   scanf("%1d%1d%1d%1d%1d", &a, &b, &c, &d, &e);
   ```

3. Scan by char

   ```
   a = getchar() - '0';
   b = getchar() - '0';
   c = getchar() - '0';
   d = getchar() - '0';
   e = getchar() - '0';
   ```

4. Scan by string

```c
char str[6];
scanf("%s", &str);
a = str[0] - '0';
b = str[1] - '0';
c = str[2] - '0';
d = str[3] - '0';
e = str[4] - '0';
```

5. And more...

For the next assignment, we'll review standard outputs of C.

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