# yubocao2 - HW0

---

### Comments

4-10 should be struct Person** friends; 4-11 didn't allocate space for friends before use;

### Timestamp

1/25/2018 0:43:37

### 1. Write a program that uses write() to print out "Hi! My name is ".

```
int main() {
	write(1, "Hi! My name is Yubo Cao", 23);
	return 0;
}
```

Grade: **100.0%**

### 2. Write a program that uses write() to print out a triangle of height n to Standard Error

```
void write_triangle(int n){
	int i, j;
	for (i = 1; i <= n; i++) {
		for (j = 1; j <= i; j++) {
			write(2, "*", 1);
		}
		write(2, "\n", 1);
	}
}
```

Grade: **100.0%**

### 3. Take your program from "Hello World" and have it write to a file called "hello world.txt" (without the quotes).

```
int main() {
	int files = open("hello_world.txt", O_CREAT | O_RDWR | O_TRUNC, S_IRUSR |
	    S_IWUSR | S_IRGRP | S_IROTH);
	write(files, "Hi! My name is Yubo Cao", 23);
	close(files);
	return 0;
}
```

Grade: **100.0%**

### 4. Take your program from "Writing to files" and replace it with printf()

```
int main() {
	close(STDOUT_FILENO);
```

```
        int files = open("hello_world.txt", O_CREAT | O_RDWR | O_TRUNC, S_IRUSR |
            S_IWUSR | S_IRGRP | S_IROTH);
        printf("Hi! My name is Yubo Cao\n");
        close(files);
        return 0;
}
```

Grade: **100.0%**

### 5. Name some differences from write() and printf()

```
The write() function is a system call and will only write a sequence of bytes,
    while the printf() function can write data in many different formats. Also,
    write() is much cheaper than printf(), so printf() has a buffer which makes
    printf() only calls write() to write when the buffer is full or we explicitly
    tell it to or it finishes a line.
```

Grade: **100.0%**

### 1. How many bits are there in a byte?

```
At least 8
```

Grade: **100.0%**

### 2. How many bytes is a char?

```
1
```

Grade: **100.0%**

### 3. Tell me how many bytes the following are on your machine: int, double, float, long, long long

```
int 4
double 8
float 4
long 4
long long 8
```

Grade: **100.0%**

### 4. On a machine with 8 byte integers (refer to code snippet below)

```
0x7fbd9d50
```

Grade: **100.0%**

### 5. What is data[3] equivalent to in C?

```
*(data + 3)
```

Grade: **100.0%**

**6. Why does this segfault (refer to code snippet below)?**

```
The constant "hello" stored in read-only memory so we can not change it.
```

Grade: **100.0%**

**7. What does sizeof("Hello\0World") return?**

```
12
```

**8. What does strlen("Hello\0World") return?**

```
5
```

**9. Give an example of X such that sizeof(X) is 3**

```
"ab"
```

Grade: **100.0%**

**10. Give an example of Y such at sizeof(Y) might be 4 or 8 depending on the machine.**

```
long
```

Grade: **100.0%**

**1. Name me two ways to find the length of argv**

```
1.argc or 2.loop through argv[] until we get a NULL pointer
```

Grade: **100.0%**

**2. What is argv[0]**

```
The execution name of the program
```

Grade: **100.0%**

**3. Where are the pointers to environment variables stored?**

```
At the top of the stack
```

Grade: **100.0%**

**4. On a machine where pointers are 8 bytes (refer to the code snippet)**

```
sizeof(ptr) = 8 and sizeof(array) = 6, the first one is a pointer, and the size of
    a pointer is 8 bytes while the second one is an array with 6 chars(a '\0' at
    the end) so the sizeof(array) is 6
```

Grade: **100.0%**

**5. What datastucture is managing the lifetime of automatic variables?**

```
Stack
```

Grade: **100.0%**

**1. If I want to use data after the lifetime of the function it was created in, then where should I put it and how do I put it there?**

```
We can put it in the heap using malloc()
```

Grade: **100.0%**

**2. What are the differences between heap and stack memory?**

```
Stack is used for static memory allocation and Heap for dynamic memory allocation.
    The stack memory is managed by the compiler while the heap memory is managed by
     the coder to allocate or free thus data in the heap can have a lifetime of the
     whole process.
```

Grade: **100.0%**

**3. Are there other kinds of memory in a process?**

```
Yes, text segment
```

Grade: **100.0%**

**4. Fill in the blank. In a good C program: "For every malloc there is a ___".**

```
free
```

**5. Name one reason malloc can fail.**

```
The program may use up all of the heap memory
```

Grade: **100.0%**

**6. Name some differences between time() and ctime()**

```
time() returns the time since 00:00:00 UTC, Jan 1, 1970 measured in seconds, while
    ctime() converts it to human readable version
```

Grade: **100.0%**

**7. What is wrong with this code snippet?**

```
Double free the same pointer
```

Grade: **100.0%**

**8. What is wrong with this code snippet?**

```
Use the memory that has been already freed
```

Grade: **100.0%**

**9. How can one avoid the previous 2 mistakes?**

```
We can avoid these two mistakes by setting the pointer to NULL once we free it
```

Grade: **100.0%**

**10. Create a struct that represents a Person and typedef, so that "struct Person" can be replaced with a single word. A person should contain the following information: name, age, friends (pointer to an array of pointers to People).**

```
struct Person {
  char* name;
  int age;
  struct Person** friends[];
};
typedef struct Person person_t;
```

Grade: **50.0%**

**11. Now make two people "Agent Smith" and "Sonny Moore" on the heap who are 128 and 256 years old respectively and are friends with each other.**

```
person_t* agentSmith = (person_t*) malloc(sizeof(person_t));
person_t* sonnyMoore = (person_t*) malloc(sizeof(person_t));
agentSmith->name = "Agent Smith";
sonnyMoore->name = "Sonny Moore";
agentSmith->age = 128;
sonnyMoore->age = 256;
agentSmith->friends[0] = sonnyMoore;
sonnyMoore->friends[0] = agentSmith;
free(agentSmith);
free(sonnyMoore);
```

Grade: **50.0%**

**12. ‘create()‘ should take a name and age. The name should be copied onto the heap. Use malloc to reserve sufficient memory for everyone having up to ten friends. Be sure initialize all fields (why?).**

```
person_t* create(char* name, int age) {
  person_t* temp = (person_t*) malloc(sizeof(person_t));
  temp->name = strdup(name);
  temp->age = age;
  temp->friends = (person_t**) malloc(sizeof(person_t*) * 10);
  return temp;
```

```
}
We need to initialize all the fields so that the variables in these fields will be
    set to 0 instead of unpredictable data
```

Grade: **100.0%**

**13. ‘destroy()‘ should free up not only the memory of the person struct, but also free all of its attributes that are stored on the heap. Destroying one person should not destroy any others.**

```
void destroy(person_t* temp) {
  memset(temp->friends, 0, 10 * sizeof(person_t*));
  free(temp->friends);
  free(temp->name);
  memset(temp, 0, sizeof(person_t));
  free(temp);
}
```

Grade: **100.0%**

**1. What functions can be used for getting characters for stdin and writing them to stdout?**

```
To get characters from stdin we use getchar(), while to write them to stdout we use
     putchar()
```

Grade: **100.0%**

**2. Name one issue with gets()**

```
gets() cannot tell weather the input is too long which can cause buffer overflow
    and thus is not safe
```

Grade: **100.0%**

**3. Write code that parses a the string "Hello 5 World" and initializes 3 variables to ("Hello", 5, "World") respectively.**

```
char* data = "Hello 5 World";
char buffer1[20];
char buffer2[20];
int times = 0;
int result = sscanf(data, "%s %d %s", buffer1, &times, buffer2);
```

Grade: **100.0%**

**4. What does one need to define before using getline()?**

```
#define _GNU_SOURCE
```

Grade: **100.0%**

**5. Write a C program to print out the content of a file line by line using getline()**

```
int main(){
  char* buffer = NULL;
  size_t capacity = 0;
  File *temp = fopen("filename.txt", "r");
  while (getline(&buffer, &capacity, temp) > 0) {
    printf("%s", buffer);
  }
  free(buffer);
  return 0;
}
```

Grade: **100.0%**

**1. What compiler flag is used to generate a debug build?**

```
-g
```

Grade: **100.0%**

**2. You modify the Makefile to generate debug builds and type make again. Explain why this is insufficient to generate a new build.**

```
Makefile reads the time stamp to decide if changes are made and the file needs to
    be recompiled. If no source code is changed, no new build will be generated. So
     to generate a new build we need to make clean first.
```

Grade: **100.0%**

**3. Are tabs or spaces used in Makefiles?**

```
tabs
```

Grade: **100.0%**

**4. What does 'svn commit' do? What's a revision number?**

```
The svn commit sends changes from your working copy to the repository. The revision
     number is a number to inform you the time you commit a Subversion repository.
    When first build a repository the revision number is 0 and each successive
    commit increase the revision number by 1
```

Grade: **100.0%**

**5. What does 'svn log' show you?**

```
The svn log display commit log messages
```

Grade: **100.0%**

# Final grade: 0.97674418599999999%