Conceptual Questions

1. list modes of fopen to perform: read, write, read & write, append.

"r"

Opens a file for reading. The file must already exist.

"w"

Opens a file for writing. The file must already exist

"r+"

Opens a file for reading and writing. The file must already exist.

"a"

Opens a file for appending. Writes to the end of the file. Creates file if it does not exist.

2. Does dynamic memory use the stack or heap? What is the difference?

Dynamic memory allocation uses the heap. The difference between the stack and heap are that the heap is created per process. Also the heap is generally slower to manipulate. The stack on the other hand is created for individual threads, which are created by processes. When a process creates a threads, usually during a function call or some similar task, a stack is allocated of a predetermined maximum size. The heap on the other hand can have it's size changed based on its needs. The stack is also generally faster than the heap because it uses LIFO. This means that the last added entry will be the first one removed when allocated new space. This also means we only really need to keep track of one pointer, that is the top element of the stack.

3. What is a pointer, how to change its address and access the data being pointed to?

A pointer is a variable whose value is the direct memory location of another variable. Pointers can be used to store values when we don't know the exact size or type of the other value. For example, if we want to determine the size of an array after the program starts executing, we can use a pointer to a memory location where we can store a size value which can be changed later.

```
#include <stdlib.h>
#include <stdio.h>

/* prints pointer address and value */
```

```
void print_pointer(int *p) {
    printf("%p -> %d\n", (void *)p, *p);
}

int main() {
    /* make pointer and point to variable */
    int *p;
    int num = 20;
    p = &num;
    print_pointer(p);

    /* Change pointer address to another variable  */
    int num2 = 35;
    p = &num2;
    print_pointer(p);

    /* Change value at pointer address */
    *p += 1;
    print_pointer(p);
}
```

4. Explain malloc, free andd how to use malloc.

void *malloc(int num);

This functiion allocates an array of **num** bytes and leaves them uninitialized.

void free(void *address);

This function released a block of memory specified by the **address**.

We can use malloc to allocate a specific amount of memory for data. In the following example, we are using malloc to allocate 5 * sizeof(char) bytes of data. Since the size of a char is 1 byte, this is 5 bytes of data. We then store the string "Eric" in this memory and print it out. The reason we need 5 bytes rather than 4 to store "Eric" is because we need to make room for the null terminator so that we know where the string ends.

Malloc usage example

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main() {
    char *text = (char *) malloc(5 * sizeof(char));
    strcpy(text, "Eric");
    printf("Our TA is %s\n", text);
}
```

Output

```
Our TA is Eric
```

5. What is the difference between malloc and calloc?

Both **malloc** and **calloc** are used to dynamically allocate memory for storing data. However, calloc will initialize all the memory to zero while malloc simply allocates the space.

Application Questions

All code and output here. Also can be found inside the zip file.