History & Language Paradigms

1. Explain the relationship between machine language, assembly language, and high level languages.

2. (a) List a difference between imperative/procedural programming and object-oriented programming.

   (b) List a difference that functional programming has from procedural or object-oriented programming.

Basic C

3. Examine the following program:

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    int number = atoi(argv[1]);      // atoi parses int from str
    int found;
    do {
        int i;
        found = 1;
        --number;
        for (i = 2; i < number; ++i) {
            if (!(number % i)) {
                found = 0;
                break;
            }
        }
    } while (!found);
    printf("%d", number);
    return 0;
}
```

   (a) What does this program do? What is the output of `./a.out 10`? Step through the program execution if it helps.

   (b) Why can we write `if (!(number % i))` instead of `if (number % i == 0)`? (These statements are equivalent.)

(c) Rewrite the code using while loops in place of the do-while loop and the for loop.

I/O

4. Write a program that searches a file for a provided number on `stdin`. Print out any errors on `stderr`. Example:

```
$ fileSearch file.txt
> 234
found: 234
```

5. (a) The following program is intended to read a text file and outputs (as binary data) the number of characters (including new lines) in each line to a file. However, there is a problem with the code the way it is currently written. Find the problem and explain how to fix it. Note: The code is compiled and linked using GCC.

```
 1 #define _GNU_SOURCE
 2
 3 #include <stdio.h>
 4 #include <stdlib.h>
 5
 6 int main(void) {
 7     FILE* inputfile = fopen("input.txt", "r");
 8     if(!inputfile) {
 9         perror("fopen failed for inputfile:");
10         exit(EXIT_FAILURE);
11     }
12     int numchars = 0;
13     char* line = NULL;
14     FILE* outputfile = fopen("output.txt", "w");
15     if(!outputfile) {
16         perror("fopen failed for outputfile:");
17         fclose(inputfile);
18         exit(EXIT_FAILURE);
19     }
20     while((numchars = getline(line, 1024, inputfile)) != -1) {
21         fprintf(outputfile, "%d\n", numchars);
22     }
23     fclose(inputfile);
24     fclose(outputfile);
25     return EXIT_SUCCESS;
26 }
```

(b) Rewrite the code so that it outputs the data using binary streams instead.

C Preprocessor

6. Give an example of a header guard for a header file named `linkedlist.h`.

7. Given the following, what is output of this program?

```
#define THING1 40
#define THING2 32
#define THING4 i

#if THING1 < THING2
#define THING3 5
#elif THING2 < THING1
#define THING3 6
#else
#define THING3 7
#endif

int main() {
    int THING4 = THING3;
    printf("%d", i);
}
```

Threads & Processes

8. Describe the differences between programs, processes, and threads.

Makefiles

9. Consider the following makefile:

```
1  CFLAGS := -std=c99 -Wall -Wextra
2  me: me.o
3      $(CC) $(LDFLAGS) -o $@ $^ $(LDLIBS)
4  calc: calc.o real.o
5      $(CC) $(LDFLAGS) -o $@ $^ $(LDLIBS)
6  calc.o: calc.c
7      $(CC) $(CFLAGS) -c $<
8  real.o: real.c
9      $(CC) $(CFLAGS) -c $<
10 .PHONY: clean
11 clean:
12     $(RM) me calc *.o
```

   (a) Ralph is annoying. One day, when Ralph makes a particularly unreasonable demand, his friend loses it and shouts, "Why don't you make me???" Always one to take things literally, Ralph pops open his favorite Bourne-compatible shell and types: `make me`. However, he is confronted with the message: `make: 'me' is up to date`.

   Explain the meaning of this message. Which (if any) of the relevant files are now located in Ralph's directory? What numeric values did `Make` compare before outputting this message?

   (b) Ralph is childishly proud of himself, but everyone just groans and tells him to "get real". Coincidentally, Ralph has a library providing functions for working with reals, as well as a calculator program to test the functionality. Ready to win another trivial victory, Ralph types `make clean` and then `make calc`.

   List the exact sequence of commands that are executed as a result of this new invocation.

Arrays & Strings

10. (a) What is the issue with the code below? What is the result of the issue?

```
1 int i, x;
2 int arr[10];
3 for (i=0; i < 10; ++i)
4 {
5     scanf("%d", &x); // Reads a number from stdin and stores it in x
6     arr[i] = x;
7 }
8
9 for (i=10; i >= 0; --i)
10 {
11     printf("%d\n", arr[i]);
12 }
```

(b) What is the intended behavior of the program?

11. (a) If you want to store the string ``Hello, world!'' in the `str` variable in the following code, what is the minimum acceptable value of `n`? Why?

```
int n;
...
char str[n];
```

(b) What is the character literal for the null terminator?

Pointers, Structures, and Dynamic Allocation

12. Consider the following statement in a larger program:

```
int* x = (int*) malloc(20); //create an array of 20 ints
```

After testing the program, you notice that the values of $x[5], x[6], \cdots, x[19]$ keep changing unexpectedly.

(a) Why is this?

(b) What should the statement actually be?

13. (a) Given the following:

```
struct Point {
    char label;
    double x;
    double y;
};
```

What (specifically) happens when the following command is executed?

```
struct Point *newPoint = (struct Point *) malloc( sizeof(struct Point) );
```

(b) Let's add some more information:

```
typedef struct {
    struct Point p1;
    struct Point p2;
    struct Point p3;
} Triangle;
```

What (specifically) happens when the following command is executed?

```
Triangle *tri = (Triangle *) malloc( sizeof(Triangle) );
```

14. (a) What does the following function do?

```
 1 int foo(int n, int *arr, int **bestp) {
 2     int *start;
 3     int *end;
 4     int best = 0;
 5     *bestp = arr;
 6
 7     for (start = arr; start < arr + n; ++start) {
 8         for (end = start; end < arr + n && *end == *start; ++end);
 9         if ((end - start) > best) {
10             best = (end - start);
11             *bestp = start;
12         }
13         start = end - 1;
14     }
15     return best;
16 }
```

(b) Make a memory map of `foo`. Use the first value set to each variable in the map.

(c) Given the following code in `main`, write code that calls our `foo` function from above and prints the result.

```
int main(int argc, char **argv)
{
    int n, i;
    int arr[] = {1, 1, 1, 2, 2, 2, 5, 5, 5, 5};

    // Write your code here.









    return 0;
}
```

15. The following program compiles.

(a) Will the program crash at run time? If so, on which line will it crash?

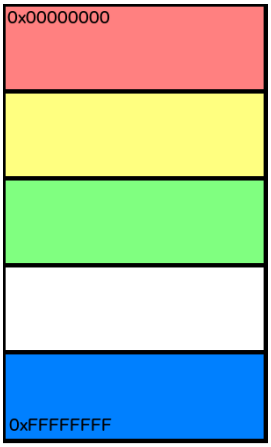
```
1 #include <stdlib.h>
2 int main(int argc, char **argv)
3 {
4     int *x = NULL;
5     int *y = NULL;
6     int *z = NULL;
7
8     x = (int *) malloc(sizeof(int) * 10);
9     y = (int *) malloc(20);
10    x = (int *) malloc(sizeof(char) * 50);
11
12    free(x);
13    free(y);
14    free(z);
15
16    return 0;
17 }
```

(b) What tool can you use to find memory leaks? What options would you use?

(c) What output would you get from part b? Is there a memory leak? (If so, where?)

Memory Layout

16. Label the sections of the program in memory and describe what is stored in each.
Your options are: `stack, data, heap, unallocated space, text`.


0x00000000
0xFFFFFFFF

Modular Design

17. For each of the following snippets of code, state whether it should be placed in the header file (`.h`) or the source file (`.c`).

(a)

```
struct Point
{
    double x;
    double y;
};
```

(b)

```
int main(int argc, char **argv)
{
    int x = run_some_function();
    printf("%d\n", x);
    return 0;
}
```

(c)

```
int do_something_and_return(int x, int y)
{
    if (y != 0)
        return x*y + (x/y);
    else
        return x*y;
}
```

(d)

```
int do_something_and_return(int x, int y);
void do_something(int x, int y);
```

Abstract Data Types

18. Generally speaking, ADTs are easier to define and work with in procedural languages like C, as opposed to object-oriented languages like Java or C#. (**True** or **False**). Explain your answer.

19. What is a *primitive data type*? Provide 3 examples of primitive data types in C.

20. Write a program that defines the structure of a queue:

- The queue ADT is described by its size (an `unsigned int`) and an array representing the queue's contents.
- The queue is built on an array of a specified size (think `#define`), the elements of which are each described by their value (any data type).
- The queue's size describes the number of occupied indices in the array; in other words, the number of elements present within the queue.
- The queue will support the following operations:
  - A function which returns an `int` representing whether or not the queue is empty, `int isEmpty(QueueADT q)`.
  - A function which adds a new element (new data) onto the end of the queue, `void enqueue(QueueADT q, void* data)`.
  - A function which removes the first element in the queue and returns it, `void* dequeue(QueueADT q)`.
  - A function which returns an `unsigned int` representing the size of the queue, `unsigned int size(QueueADT q)`.

Advanced C Features

21. (a) Define a union, `typedef`ed to the name `printableData`, that can hold any of: an seven-character long string, a `double`, or an `int`.

(b) An idiom found in some C programs is the *tagged union*.

```
typedef unsigned char tag_t;

// could also use an enum
const tag_t STRING_TAG = 0;
const tag_t DOUBLE_TAG = 1;
const tag_t INT_TAG = 2;

typedef struct {
    printableData data;
    tag_t tag;
} taggedData;
```

In addition to the actual data, the tagged union stores a tag whose value corresponds to the type of data being stored. Because of this, we can tell which of the three types of values are being stored in the `data` field.
Write a function that takes an array of `taggedData` and prints out the data each element holds, each on its own line. Also take a `count` parameter. Assume all necessary headers have been included. (*Hint:* Use a series of `if`/`else if` statements to properly print each type of data that can be stored.)

(c) Suppose that we want to be able to process this data in other ways besides just printing to standard out. We can use function pointers to write a more generic `processAll` function. Suppose the following struct is defined:

```
typedef struct {
    void (*processString)(char*);
    void (*processDouble)(double);
    void (*processInt)(int);
} processingFunctions;
```

Define a `processAll` function that takes an array of `taggedData`, a `count` parameter, and a `processingFunctions` value, and uses the functions pointed to by the struct to process each element of the array, rather than using `printf` as was used in the `printAll` function.

22. Suppose we have the following bit field representing a player in a ripoff of the Halo video game series.

```
struct player {
    unsigned int is_alive:1;
    unsigned int team_color:1;
    unsigned int weapon_type:2;
    unsigned int ammo_remaining:4;
};
```

However, you need to store these players as 8 bit `unsigned char` values. You decide to store `ammo_remaining` in the lowest 4 bits of the `unsigned char`, `weapon_type` in the next 2 bits, `team_color` in the next bit, and `is_alive` in the highest bit. Write a function that takes a `player` struct and returns its `unsigned char` representation under the aforementioned rules.

```
unsigned char serializePlayer(struct player player1) {









}
```

OS-Level I/O

23. Explain the relationship between the FD table, the open file table, and the I-node table. Which tables are in user space (a.k.a. private kernel space) and which are in system space (a.k.a. shared kernel space).

24. You create a file named `sliding.txt` with the following contents (there are newlines between each line, but not at the end of the file):

```
>>>>v
>>>>v
>>>>v
>>>>v
>>>>v
```

However, your no-good friend sneakily changes one of the lines to

```
^<<<<
```

and you don't know which line it is. Write a program that fixes the file, but only writes 5 characters (it can read the whole file, however). Do not worry about error handling. (*Hint:* You can use the `strstr` function in `<string.h>`, which takes two strings, searches for the second string inside the first, and returns the `char*` location within the first string where the second string was found. You may also need pointer arithmetic.)