A) Multiple Choice / True False

(*Circle* the letter beside the best answer, do not fill in the blanks)

1. Unlike Java, the C language has been internationally standardized by globally recognized standards organizations including ISO, IEC, and ANSI. Over time different versions of the C standard have been issued. Which of the following is **not** a name for a version of the C standard?

   a) C99
   b) C11
   c) C73
   d) C89

2. Using the 'goto' statement in C is always considered bad.

   a) True
   b) False (smart programmers know it's ok in two specific situations, see slides)

3. Suppose you declare a variable but don't initialize it. Which of the following may result in an unpredictable/undefined variable value? Choose all that apply.

   a) A global variable with data type 'double' *(global vars are initialized to zero)*
   b) A local variable with data type 'bool'
   c) A local variable that's an array of integers size 100
   d) A variable declared with the keyword 'static' *(static vars are initialized to zero)*

4. Which statement below is true of variables declared with the keyword 'static'? Choose one.

   a) They are accessible (visible) anywhere in the same module (.c file).
   b) They are stored on the stack.
   c) They are stored in the static data area.
   d) They are accessible (visible) from any module (any .c file).
   e) None of the above statements is true.

B) Fill in the Blanks

5. What does the following C code print out?

```
char str[] = "hey";
int num = (strcmp(str, "z") >= 0 ? 42 : 22);
printf("%d\n", num);
```

___22_____

6. Suppose you're trying to compile a C source file which contains two functions named 'calculate' and 'estimate'. The compiler gives you a syntax error where the 'calculate' function calls 'estimate' (but the function call statement is correct). Describe three different ways to fix this syntax error.

   1. Move the estimate function definition before the calculate function definition.

   2. Add a prototype for the estimate function near the top of the source file.

   3. Create a header file containing the estimate function prototype, and include it.

7. Define the term Abstract Data Type. Be brief.

   An Abstract Data Type is a set of values and their associated operations. It does not specify a particular implementation.

8. List the three main steps which take place when compiling C source files to produce an executable file (.exe).

   (1) Preprocess

   (2) Compile

   (3) Link

9. Briefly describe what's wrong with each of the following input/output statements on the blank line provided. If the statement is correct then write "ok".

```
unsigned int num = 377;
long int input;
char message[] = "Data structures are cool.";

printf("The number is %d\n", num);
```
Wrong conversion spec, should be %u

```
printf("%s\n", &message);
```
& should not be used here

```
scanf("%ld", input);
```
& is needed here before 'input'

```
scanf("%25s", message);
```
ok

10. Examine the function declaration below. Under each parameter write C or N to indicate whether the data for that parameter is Copied (function can't change original value) or Not copied (function can change the original value). The first one is an example. Note: Assume 'Student' is a struct that has been previously declared.

```
void manyArgs(float val, char str[], Student st, char ch, int* arr);
```
C _N_ _C_ _C_ _N_

Note: Value types are copied when assigning using '=' or passing as a parameter. Reference types are not copied.

11. Given the following structure and variable declarations, write how to print the 'jewels' field of the variable ppl (assume the variable is properly initialized).

```
typedef struct {
    int swords;
    int jewels;
    int potions;
} ObjectCount;

typedef struct {
    char nickName[31];
    int health;
    ObjectCount* objCnt;
} Player;

Player* ppl;
```
Answer: `printf("%d\n", ppl->objCnt->jewels);`

12. Write one line of code to declare and allocate an array of 1000 unsigned 2-byte integers using malloc or calloc (your choice).

```
unsigned short int* arr = malloc(1000 * sizeof(unsigned short int));
```

13. a) Write another line of code to resize your array from Q12 so it's half the original size.

```
arr = realloc(arr, 500 * sizeof(unsigned short int));
```

b) Rewrite your code from Q12 and Q13a so that it first uses typedef to declare a new data type called **ushort** that's equivalent to the data type for unsigned 2-byte integers. Then change the rest of the code to use the new data type.

```
typedef unsigned short int ushort;
ushort* arr = malloc(1000 * sizeof(ushort));
arr = realloc(arr, 500 * sizeof(ushort));
```

C) Programming

14. Make sure you've completed all exercises in all of the course slides from week 1 to week 6. Do it without looking at my posted solutions.

15. Suppose the data type Stack has been defined as in the course slides. First declare a module variable with data type Stack. Then write a main function to:
- Initialize the Stack.
- Push two numbers on the stack.
- Pop the top-most number and print it.

```
static Stack S;
int main(int argc, char** argv)
{
    initStack();
    push(42);
    push(95);
    printf("The top number is %d\n", pop());
}
```

16. Write a C function that accepts a string (array of char) as a parameter and encodes each letter in the string using exclusive-or with 0x5A. The exclusive-or operator in C is the same as in Java: ^
    A useful property of this secret code is that you can encode again to decode a message! Test your program by writing a main function that inputs a message, encodes it and prints the result (it may look funny), then decodes it and prints the result.

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

// Encode each character in 'str' using exclusive-or.
void encode(char str[])
{
    int len = strlen(str);
    for (int index = 0; index < len; index++) {
        str[index] = str[index] ^ 0x5A;
    }
}

int main(int argc, char** argv) {
    // Input a string
    printf("Please enter a message: ");
    char msg[100];
    scanf("%99s", msg);

    printf("You entered: %s\n", msg);

    // Encode the message
    encode(msg);
    printf("The encoded message is: %s\n", msg);

    // Decode the message by encoding again
    encode(msg);
    printf("The decoded message is: %s\n", msg);
}
```

17. Write a C program to create and manage a dynamic array of doubles. Your dynamic array (pointer) and a variable to keep track of the size of the array should be module variables (static global) so they can be accessed from different functions. Write the

following functions:

`void create(void)` – Creates (allocates) the dynamic array at size 500 and fills it with zeros.

`void enlarge(void)` – Doubles the size of the dynamic array and fills the new part of the array with zeros. Should not change the existing part of the array. Don't forget to update your 'size' variable.

`void fill(double num)` – Sets all the elements of the dynamic array to 'num'.

`void delete(void)` – Deallocates the dynamic array. After deallocation you should set the dynamic array pointer to NULL, just in case.

Write a main function to test the other functions. It should create a dynamic array, fill it with the value 8.1, enlarge the array, print the contents of the array, then delete the array.

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

// Module variables
static double* arr = NULL;  // Dynamic array of doubles
static int size = 0;        // The current array size (# elements)

// Allocates the dynamic array at size 500 and fills it
// with zeros.
void create(void) {
    size = 500;
    // calloc automatically fills with zeros
    arr = calloc(size, sizeof(double));
}

// Doubles the size of the dynamic array and fills the new part of
// the array with zeros. Should not change the existing part of the
// array. Don't forget to update your 'size' variable.
void enlarge(void) {
    size *= 2;
    arr = realloc(arr, size*sizeof(double));

    // Fill the new part of the array with zeros, otherwise its
    // contents will be undefined
    for (int index = size/2; index < size; index++) {
        arr[index] = 0;
    }
```

```
}

// Sets all the elements of the dynamic array to 'num'.
void fill(double num) {
    for (int index = 0; index < size; index++) {
        arr[index] = num;
    }
}

// Deallocates the dynamic array.
void delete(void) {
    if (arr != NULL) {   // Logic to prevent "double-free"
        free(arr);
        arr = NULL;
    }
}

int main(int argc, char** argv) {
    create();
    fill(8.1);
    enlarge();

    for (int index = 0; index < size; index++) {
        printf("%f ", arr[index]);
    }

    delete();
    return 0;
}
```

18. a) Implement the **strcat** C library function. Name your function **myStrcat** and declare it as follows:

    `void myStrcat(char* dest, const char* src)`

    You should use pointer syntax to work with the strings, not array syntax. In other words the myStrcat function should not contain any square brackets – use pointer incrementing and dereferencing instead. Write a main function that passes two strings to the myStrcat function to test that it works.
    Note: If you want you can write your myStrcat function using array indexing (square brackets) to get it working, then convert it to use pointer syntax.

    b) Change your main function so that it uses **malloc** to allocate your string(s) instead

of array declarations. After this your main function should not use any square brackets either!

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

// Our version of 'strcat' implemented using pointers.
void myStrcat(char* dest, const char* src) {
    // Find the end of the destination string
    char* dptr = dest;
    while (*dptr != '\0') {
        dptr++;
    }

    // Append src to dest
    const char* sptr = src;
    while (*sptr != '\0') {
        *dptr = *sptr;
        dptr++;
        sptr++;
    }

    // Ensure that the destination string is null-terminated
    *dptr = '\0';
}

// Main function to test if myStrcat works
int main(int argc, char** argv) {
    // Allocate string size 100 using malloc
    char* combined = malloc(100 * sizeof(char));

    strcpy(combined, "string part 1");    // Put text in the string
    myStrcat(combined, "string part 2");  // Append more text

    printf("The combined string is: %s\n", combined);

    free(combined);  // Don't forget to free everything you malloc!
    return 0;
}
```