

CMSC 21 First Long Examination
April 22, 2022 @ 9am - 1pm

Part I. True or False. Justify your answer.

1. A long double can be used if range of a double is not enough to accommodate a real number.
➤ **True. Since double is only 8 bytes while long double is 10 bytes.**
2. A zero value is considered false and a non-zero value is considered true.
➤ **True. In a Boolean function, when the value is 1, it is considered True and if the value is 0, it is considered False.**
3. = is used for comparison whereas == is used for the assignment of two quantities.
➤ **False. "=" is used for assigning values to variables and "==" is used for comparison.**
4. The keywords cannot be used as variable names.
➤ **True. We use keywords to define syntax, they have a specific meaning and are already reserved.**
5. The address operator (&) is used to tell the compiler to store and data at an address.
➤ **True. It is used when we are storing variables. For example, if we want to store the value 1 to a certain variable from the scanf function, we use &.**
6. Sign qualifiers can be applied to all data types.
➤ **False. It can only be used for int and char type.**
7. Suppose you declared a variable a = 6, b = 7, d = 3, (((a==b) || (b>a)) && (d<a)) will generate a false.
➤ **False. It will generate true. The "(b>a) && (d<a)" conditions are met so that means that it is true. For the "or" part which is the "(a==b)" the answer is false since a and b are not equal. So now we have True or False and when it comes to "or" at least 1 should be True for it to be True. Therefore, the code generates true.**
8. The break statement is required in the default case of a switch selection statement
➤ **False. A break statement is not required anymore because that will be automatically printed when other conditions are not met.**
9. The expression (x > y && a < b) is true if either x > y is true or a < b is true.
➤ **False. Since it uses "&&" it means that both x > y and a < b should be true.**
10. while(i > 0) printf("T minus %d \n", i--) will produce a similar output with while (i) printf("T minus %d \n", i--)
➤ **True. Whether we use i>0 or only i, it will still produce similar output. Things will only change if we use i++ since it will provide an infinite output.**

Part II. Find the errors in the following program. Indicate the possible correction.

1. `x = 1;`

`while (x <= 10);`

`++x;`

`}`

- **While part should not end with “;”**
- **The “}” has no partner.**
- **For the solution, while part should end with “{”. We can also add printf to show the x values.**

2. `for (double y = .1; y != 1.0; y += .1) {`

`printf("%f\n", y);`

`}`

- **The “double y” will cause an error. To fix the problem we can remove the double inside the for loop and use it to declare the y.**
- **double y;**
for (y = .1; y != 1.0; y += .1) {
printf("%f\n", y);
}

3. `switch (n) {`

`case 1:`

`printf("The number is 1");`

`case 2:`

`printf ("The number is 2");`

`break;`

`default:`

`printf ("The number is not 1 or 2");`

`break;`

`}`

- **Case 1 does not have a break function. This will cause the program to continue printing up the next statements. For example, if case 1 is met, it will print “The number is 1 The number is 2”. It will not stop to “The number is 1” since the break function is not there. So, it is important to add a break function after case 1.**
- **Not really an error since the program will still run correctly but we can remove the break function in the default statement. Automatically, it will print the statement is the other conditions are not met.**

4. The following code should print the values 1 to 10.

`n = 1;`

`while (n < 10) {`

`printf("%d ", n++);`

`}`

- **Instead of (n<10), we can change it to (n<=10) because, in that way, 10 will also be printed. Now we can print the values 1 to 10.**

Part III. Answer the following questions.

1. What happens if you attempt to access the value of an uninitialized variable? Provide some examples of “strange behaviours” that could potentially occur.
 - **Since it has no value assigned to it, the program can produce a different and wrong output. It may work or run but it can lead to inconsistent results because of an uninitialized variable.**
2. What happens if a program reaches the end of the main function without executing a return statement?
 - **Nothing will happen, the program can still execute without an error. The return statement is only a signal that the program was executed correctly.**
3. Explain the difference between %i and %d.
 - **They differ when used in scanf. %d assumes base 10, it takes negative and positive integer values but should be in decimal form. %i on the other hand, detects the base, it takes the integer values with hexadecimal, octal, or decimal type.**
 - <https://www.geeksforgeeks.org/difference-d-format-specifier-c-language/>

4. Assuming that we have variables:
a, b – int
c – float
what will be the values of a, b, c after the following scanf call?

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

if the user enters

10.3 5 5.6

- **a= 10.3**
- **b= 100.3000005**
- **c= 5**

5. Assuming that we have variables:

a, b – float
c – int

what will be the values of a, b, c after the following scanf call?

```
scanf("%f%d%g", &a, &c, &b);
```

if the user enters

12.3 45.6 789

- **a= 12.3**
- **b= 2.00000010708058111.79084e-307**
- **c= 45.6**

6. How would the C compiler interpret the following expressions? Add parenthesis. (Hint: operator precedence)

- **a * b - c * d + e**
- **a / b % c / d**
- **- a - b + c - + d**
- **a * - b / c - d**

- **(a * b) - (c * d) + e**
→ C will be multiplied by D. Their answers will be added to E. After the Addition, the product of A and B will be minus it.
- **(a / b) % (c / d)**
→ A and B will be divided as well as the C and D. The answers that we will get are going to be the values that we will use for the modular division.
- **(- a - b + c -) + d**
→
- **a * (- b / c) - d**
-b/c will be divided and will be multiplied by a. Then, it will be minus D.

7. Modify the code such that the loop's body will be empty:

```
for (i = 0; j > 0; i++)
    j /= 2;
```

- **for (i = 0; j > 0; i++) {**
 j /= 2;
 }

Part IV. Coding Applications

8. Given a snippet of this code,

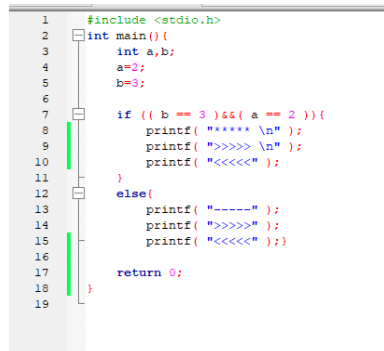
```
if ( b == 3 )
if ( a == 2 )
printf( "*****" );
else
printf( "----" );
printf( ">>>>" );
printf( "<<<<" );
```

- What will the output assuming that a = 2 and b = 3? What is the issue with the code above? How can you improve the readability? Implement in item 8b (next problem).
 - **It produces an output, "*****>>>><<<<". The issue is the proper use of "{}" and the code is hard to understand.**
- Modify the code such that it produces the following outputs (a = 2 and b = 3)

a. *****

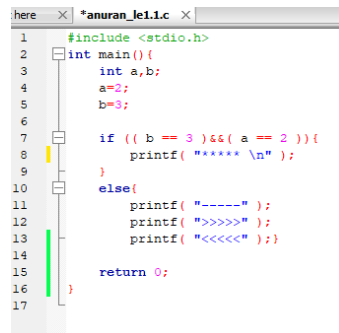
>>>>>

<<<<<



```
1 #include <stdio.h>
2 int main(){
3     int a,b;
4     a=2;
5     b=3;
6
7     if (( b == 3 )&&( a == 2 )){
8         printf( "***** \n" );
9         printf( ">>>> \n" );
10        printf( "<<<<" );
11    }
12    else{
13        printf( "----" );
14        printf( ">>>>" );
15        printf( "<<<<" );
16
17    return 0;
18 }
```

b. *****



```
here x anuran_1c1.1.c x
1 #include <stdio.h>
2 int main(){
3     int a,b;
4     a=2;
5     b=3;
6
7     if (( b == 3 )&&( a == 2 )){
8         printf( "***** \n" );
9     }
10    else{
11        printf( "----" );
12        printf( ">>>>" );
13        printf( "<<<<" );
14
15    return 0;
16 }
```

C. *****

<<<<<



```
Start here x anuran_je1.1.c x
1  #include <stdio.h>
2  int main(){
3      int a,b;
4      a=2;
5      b=3;
6
7      if (( b == 3 ) && ( a == 2 )){
8          printf( "***** \n" );
9          printf( "<<<<< " );
10     }
11     else{
12         printf( "-----" );
13         printf( ">>>>>" );
14         printf( "<<<<<" );
15     }
16     return 0;
17 }
18
```

9. We want to write a program that asks the user to enter the side of a square and displays a square out of asterisks. After printing the square, the program will ask the user to print another square. For example:

```
Enter square size: 4
*****
*   *
*   *
*   *
*****
Print another square? Enter y or n: x
Not a valid choice.
Print another square? Enter y/n: y
Enter square size:3
***
*  *
*  *
***
Print another square? Enter y or n: n
END
```

Fill in the blanks to complete the program:

```
#include <stdio.h>

int main(){
    int row, column = 0;
    int size = 0 ;
    char cont = 'y';

    while(____a____){
        printf("Enter square size:");
        scanf("____b____", &size);

        for( row = 0 ;row < size ;____c____){
            for(column = 0 ;____d____;column++){
if (____e____||
                printf("*");
            }else{
                printf(" ");
            }
        }

        ____9____

        printf("Print another square? Enter y or n: ");
        ____10____

        if (cont == 'n'){
            ____11____
        }else if (____12____){
            printf("Not a valid choice. \n");

            printf("Print another square? Enter y/n: ");
            ____13____
        }
    }

    return 0;
}
```

- A. size <= 0
b. %d
c. row++
d. column < size
e. row = 0
f. row = size-1
g. column = 0
h. column = size-1
9. printf("\n");
10. scanf("%d", &cont)
11. return 0;
12. cont == 'x'
13. scanf("%d", &cont)

GITHUB LINK: <https://github.com/supeerngln/CMSC21.git>

10. The square root of a positive number can be approximated by the following iterative method

1 x

$$y_{n+1} = \frac{1}{2} \left(y_n + \frac{x}{y_n} \right)$$

Where x is the number entered by the user and y_{n+1} is the next guess for the square root of x , computed using its old value y_n and x . Since an initial guess is required, we set $y_0 = 1$.

See the example below:

x	y	x/y	$\frac{1}{2}(y_n + \frac{x}{y_n})$
3	1	3	2
3	2	1.5	1.75
3	1.75	1.73429	1.73214
3	1.73214	1.73196	1.73205
3	1.73205	1.73205	1.73205

Use a loop to iterate until the absolute value of $y_{n+1} - y$ is less than or equal to the tolerance, given by the variable $tol = 0.00001$.

Use the `fabs` function to find the absolute value of a `double`, from the `<math.h>` header.

Prompt the user to enter x and display the final approximation.