Neil Bryan F. Flauta CMSC 21- 1

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

Part I. True of 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. The qualifier long will further lengthen the length size or range of a double.

2. A zero value is considered false and a non-zero value is considered true.

True. In C programming language, 0 is considered false while any values other than 0 is considered true.

3. = is used for comparison whereas == is used for assignment of two quantities.

False. The "=" is supposed to be used for assignment a value to a variable name while the "==" is used to test whether the two values from each side is equivalent to each other or not.

4. The keywords cannot be used as variable names.

True. Using keywords as variable names will result to an error as each keyword has their own function in C.

5. The address operator (&) is used to tell the compiler to store and data at an address.

False. The ampersand "&" symbol followed by a variable name tells the *scanf* rather the location in the memory at which the value of a variable name will be stored.

6. Sign qualifiers can be applied to all data types.

False. Each data type has their own sign qualifiers and some data types cannot accept sign qualifiers. Signed qualifiers "signed" and "unsigned" can be used for "int" and "char" data types but are not applicable to float, double, and void data types.

7. Suppose you declared a variable a = 6, b = 7, d = 3, (((a==b) || (b>a)) && (d<a)) will generate a false.

False.
Let 1 = true and 0 = false.
(((6 == 7) || (7 > 6)) && (3 < 6))
((0 || 1) && 1)
(1 && 1)
1 (True)

8. The break statement is required in the default case of a switch selection statement.

False. The break statement in the default case of a switch selection statement is only optional if the default clause is placed as the last statement. But it is a must if each case in the switch selection statement has break statement in order to avoid confusion.

9. The expression (x > y && a < b) is true if either x > y is true or a < b is true.

False.
Let 1 = true and 0 = false; x > y is true and a < b is false.
(x > y && a < b)
(1 && 0)
0 (false)

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. Each statement will produce the same output. In the second while loop statement, the while(i) indicates that the loop will repeat until it becomes 0 and 0 is equivalent to false. This is also the same with the first while loop statement that it will repeat if it is still greater than 0.

Let i = 3
The output:

```
T minus 3
T minus 2
T minus 1
```

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

1.

```
x = 1;
while (x <= 10);
++x;
}
```

Corrected code:

```
int x = 3;

while (x <= 10) {
    ++x;
}
```

a. Indicate *data type* when declaring.

b. *Brace* instead of a semicolon.
c. Observe *proper indention* in line **(++x;)**

2.

```
for (double y = .1; y != 1.0; y += .1) {
    printf("%f\n", y);
}
```

Corrected code:

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;
}
```

Corrected Code:

```
int n = 1;

switch (n){
case 1:
    printf("The number is 1");
    break;
case 2:
    printf("The number is 2");
    break;
default:
    printf("The number is not 1 or 2");
    break;
}
```

a) Declare the *data type and value* first for the variable name “n”.

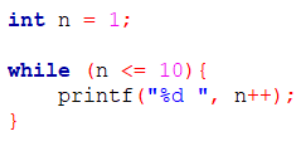
b) Put *break* statement in the first case.

4.

The following code should print the values 1 to 10.

```
n = 1;
while (n < 10) {
    printf("%d ", n++);
}
```

Corrected Code;



```
int n = 1;

while (n <= 10){
    printf("%d ", n++);
}
```

a.) Declare the data type of the variable name "n".

b) To include 10, change the operator to "<=".

Part III. Answer the following questions.

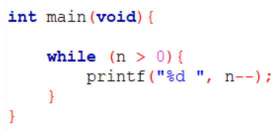
1. What happens if you attempt to access the value of an uninitialized variable? Provide some examples of "strange behaviours" that could potentially occur.

If we access the value of an initialized variable, most likely we are able to encounter errors because in any programming convention, initializing a variable is a must in order for the code to run.

Examples of code with uninitialized variable:

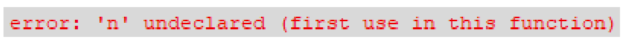
Output:

Variable n is not declared with any data type or value.



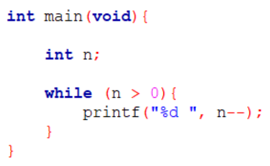
```
int main(void){

    while (n > 0){
        printf("%d ", n--);
    }
}
```

Error: 'n' undeclared.



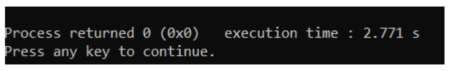
```
error: 'n' undeclared (first use in this function)
```

Variable n is of int data type with no value.



```
int main(void){

    int n;

    while (n > 0){
        printf("%d ", n--);
    }
}
```

Some compilers will usually display a warning that the variable n Is uninitialized while others will treat it as not an error and will run but only result to an undefined behavior or logical error. This will result to wrong output or no output at all.



```
Process returned 0 (0x0)   execution time : 2.771 s
Press any key to continue.
```

2. What happens if a program reaches the end of the main function without executing a return statement?

If no return statement has been executed, the terminal window will display no output but rather shows that the program did run without any displayed output. It means that there is nothing wrong with the syntax error, but only with the logic of the program or logical error.

3. Explain the difference between %i and %d.

The format specifier %i and %d is the same with the printf but only different in scanf. When using %i in the scanf, %i takes integer value as integer with decimal, hexadecimal, or octal type. On the other hand, %d takes integer value as signed decimal integer. It means that it will take both positive and negative values but inputs in hexadecimal or octal format.

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
```

The value for a 10.
The value for b is 5.
The value for c will be 5.000000.

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
```

The value for a is 12.300000
The value for b is 789.
The value for c is 45.

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

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

a. (a * b) – (c * d) + e
b. (((a / b) % c) / d)
c. (((-a) – b) + c) – (+d)
d. ((a * (-b)) / c) – d

7.

Modify the code such that the loop’s body will be empty:

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

Modified code:

```
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( "<<<<<" );
```

a. 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).

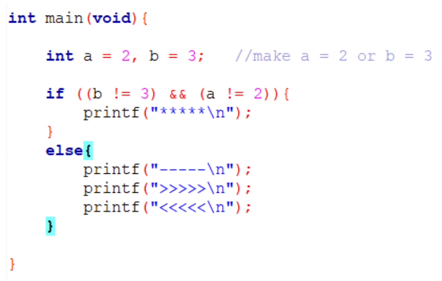
The output of the code will be “*****”.
The issues in this code are:
a) a and b are not initialized
b) The two conditions are not joined by a relational operator &&
c) The indentions are not appropriate.
d) Missing brackets in the if-else statement.

b. Modify the code such that it produces the following outputs (a = 2 and b = 3)

a.

```
*****
>>>>>
<<<<<
```

Code:

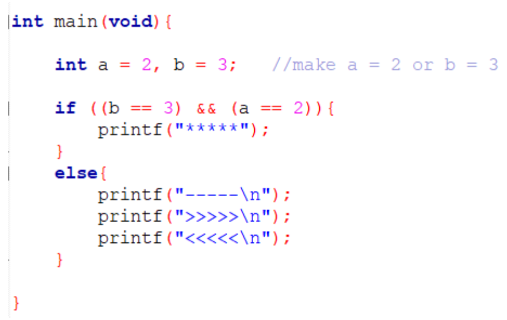


```
int main(void){

    int a = 2, b = 3;   //make a = 2 or b = 3

    if ((b != 3) && (a != 2)){
        printf("*****\n");
    }
    else{
        printf("-----\n");
        printf(">>>>>\n");
        printf("<<<<<\n");
    }

}
```

b.

```
*****
```

Code:

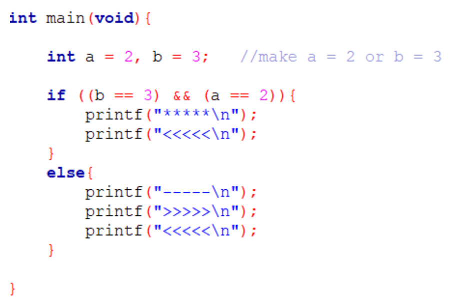


```
int main(void){

    int a = 2, b = 3;   //make a = 2 or b = 3

    if ((b == 3) && (a == 2)){
        printf("*****");
    }
    else{
        printf("-----\n");
        printf(">>>>>\n");
        printf("<<<<<\n");
    }

}
```

c.

```
*****
<<<<<
```

Code:



```
int main(void){

    int a = 2, b = 3;   //make a = 2 or b = 3

    if ((b == 3) && (a == 2)){
        printf("*****\n");
        printf("<<<<<\n");
    }
    else{
        printf("-----\n");
        printf(">>>>>\n");
        printf("<<<<<\n");
    }

}
```

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
```

```
#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__ || __f__|| __g__|| __h__){
                           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. cont = ‘y’
b. %d”
c. row++
d. column < size
e. (row == 0)
f. (row == size)
g. (column == 0)
h. (column == size)

9. printf(“\n”)
10. scanf(“%c”, &cont)
11. printf(“END”)
12. cont != ‘n’
13. scanf(%c”, &cont)

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

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

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:

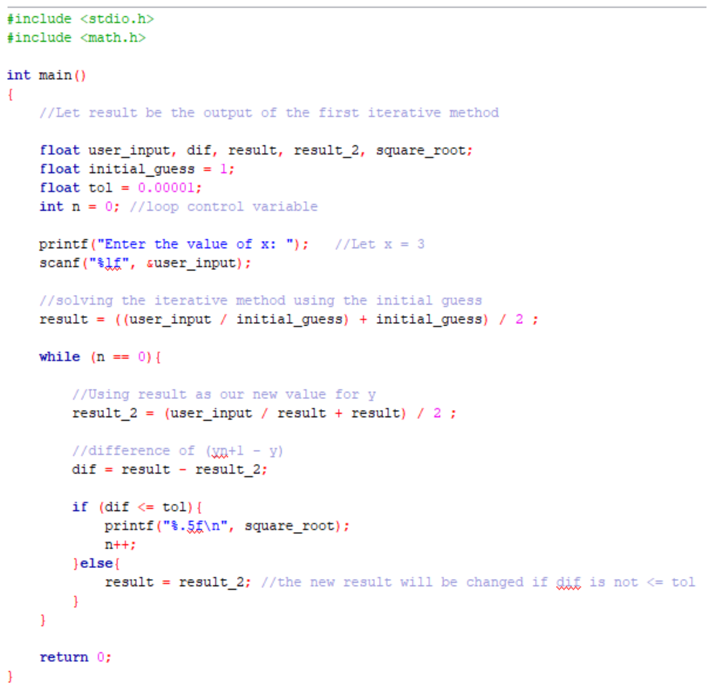
x	**y**	**x/y**	$\frac{1}{2}\left(y_n + \frac{x}{y_n}\right)$
3	1	3	2
3	2	1.5	1.75
3	1.75	1.73429	1.73214
3	1.73214	1.73196	1.73205
3	1.73205	1.73205	1.73205

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.

Screenshot of the code: (Ma’am I’m sorry I can’t finish it 😟)



```c
#include <stdio.h>
#include <math.h>

int main()
{
    //Let result be the output of the first iterative method

    float user_input, dif, result, result_2, square_root;
    float initial_guess = 1;
    float tol = 0.00001;
    int n = 0; //loop control variable

    printf("Enter the value of x: ");   //Let x = 3
    scanf("%lf", &user_input);

    //solving the iterative method using the initial guess
    result = ((user_input / initial_guess) + initial_guess) / 2 ;

    while (n == 0){

        //Using result as our new value for y
        result_2 = (user_input / result + result) / 2 ;

        //difference of (yn+1 - y)
        dif = result - result_2;

        if (dif <= tol){
            printf("%.5f\n", square_root);
            n++;
        }else{
            result = result_2; //the new result will be changed if dif is not <= tol
        }
    }

    return 0;
}
```

GitHub link: https://github.com/nbbryy/CMSC-21.git