C Programming lab: Control Flow Quiz

Name: ______________________________

Email: ______________________________

Roll no: ______________________________

Concept: *verifying code*

1. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Does this logic compute the correct result?

   ```
   int sum(int a, int b)
       {
       int total = 0;
       for (;a<b;a++)
           total = total + a;
       return total;
       }
   ```

   (a) No
   (b) Yes

2. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Does this logic compute the correct result?

   ```
   int sum(int a, int b)
       {
       int total = 0;
       for (;a<=b;a++)
           total = total + a;
       return total;
       }
   ```

   (a) No
   (b) Yes

3. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Does this logic compute the correct result?

   ```
   int sum(int a, int b)
       {
       int total = 0,i;
       for (i=a;i<=b;i++)
           total = total + i;
       return total;
       }
   ```

   (a) Yes
   (b) No

4. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*). Does this logic compute the correct result?

```
int sum(int a, int b)
    {
    int total = 0,i;
    for (i=a;i<b+1;i++)
        total = total + i;
    return total;
    }
```

(a) Yes
(b) No

Concept: *implementing loops*

5. Implement this loop: Product of the numbers from 1 to x, inclusive.
6. Implement this loop: Product of the numbers from a to b, inclusive.
7. Implement this loop: sum of the numbers from a to b, inclusive.
8. Implement this loop: sum of the numbers from 1 to x, inclusive.
9. Implement this loop: determine a^b by updating an accumulator via multiplication by a, b times.
10. Implement this loop: count the number of prime numbers from a to b inclusive. Here is a naive implementation of an *isPrime* logic, if you want to test your loop. We will assume that math.h header file is included. The option of math-library (-lm) is also assumed for compilation.

```
int isPrime(int n)
    {
    for (i=2;i<=(int)pow(n,0.5))
        if (n % i == 0)
            return 0;
    return 1;
    }
```

11. Implement this loop: count the number of numbers that are divisible by 2 or 3 from a to b inclusive.

Command line arguments: Verifying code

12. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*) using command line arguments. Does this logic compute the correct result?

```
int main(int argc, char *argv[])
    {
    int a, total = 0;
    for (a=atoi(argv[1])+1;a<atoi(argv[2]);a++)
        total = total + a;
    return 1;
    }
```

(a) No
(b) Yes

13. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*) using command line arguments. Does this logic compute the correct result?

```
int main(int argc, char *argv[])
    {
    int a, total = 0;
    for (a=atoi(argv[1]);a<=atoi(argv[2]);a++)
        total = total + a;
    return 1;
    }
```

(a) No

(b) Yes

14. Consider the problem statement: *sum the numbers from* a *to* b (*inclusive*) using command line arguments. Does this logic compute the correct result?

```
int main(int argc, char *argv[])
    {
    int a, total = 0;
    for (a=atoi(argv[1])-1;a<atoi(argv[2]);a++)
        total = total + a;
    return 1;
    }
```

(a) No

(b) Yes

15. Implement this loop: total up the product of the numbers from 1 to x where 'x' is provided as command line arguments (argv[1]), inclusive. (Show complete program)

16. Implement this loop: total up the product of the numbers from a to b, inclusive where 'a' and 'b' are provided as command line arguments (argv[1], argv[2]). (Show complete program)

17. Implement this loop: total up the sum of the numbers from a to b, inclusive. where 'a' and 'b' are provided as command line arguments. (Show complete program)

18. Implement this loop: total up the sum of the numbers from 1 to x, inclusive. where 'x' is provided in the command line argument. (Show complete program)

Activity problems

19. Implement a calculator program that takes three command line arguments (excluding the program file name) and performs a simple calculation as shown below:

```
./calc 2 + 3

./calc 5 / 2

./calc 5 - 6

./calc 4 \* 6
```

(Show complete program)

20. Implement a program which will take three arguments (intial final step) as shown below and prints the numbers over step-size. Use loop in this task. For example:

```
./loop 3 10 2
should show the following output.
3 5 7 9
```

(show complete program)

21. Implement a program which does eight modes of arithmatic logic unit (ALU) operations: multiplexing, demultiplexing, encoding, decoding, direct-memory-access, random-memory-access, updating-cache, and recent-memory-search. All eight modes are controlled by eight bits. Design an ALU such that when user specifies an option in command line argument, the program should indicate the specific operation. The users can specify multiple operations by passing the bits in the command line argument. In the multiple operations scheme, we will assume that sequential operations are performed in the ALU. Use bitwise operations to display the operations selected by the user. Passing a Zero (0) option, should display entire menu of ALU as shown below:

```
 ./alu 0
Select menu
0x01: multiplexer
0x02: demultiplexer
0x04: encoding
0x08: decoding
0x10: dma
0x20: rma
0x40: updating-cache
0x80: recent-memory-search
```

Passing 1 option, should display "multiplexer" as shown below:

```
./alu 1
Multiplexer selected
```

Passing 2 option, should display "Demultiplexer" as shown below:

```
./alu 2
Demultiplexer selected
```

Passing 3 option, should display two operations as shown below:

```
./alu 3
Demultiplexer selected
Multiplexer selected
```

Passing 255 option, should display all the operations.