

# Final Exam - Due 07/22/2020 @ 11:59 PM

NAME: Susan Sapkota

## Basics

### Question 1

Tell us the sizes of the following data types in C:

| Data Type | Size (in bytes) |
|-----------|-----------------|
| char      | 1 byte          |
| short     | 2 bytes         |
| long      | 8 bytes         |

### Question 2

Say we have a c file called **test.c**, what is the Linux terminal command we would use to compile it and create an executable called **a.out**.

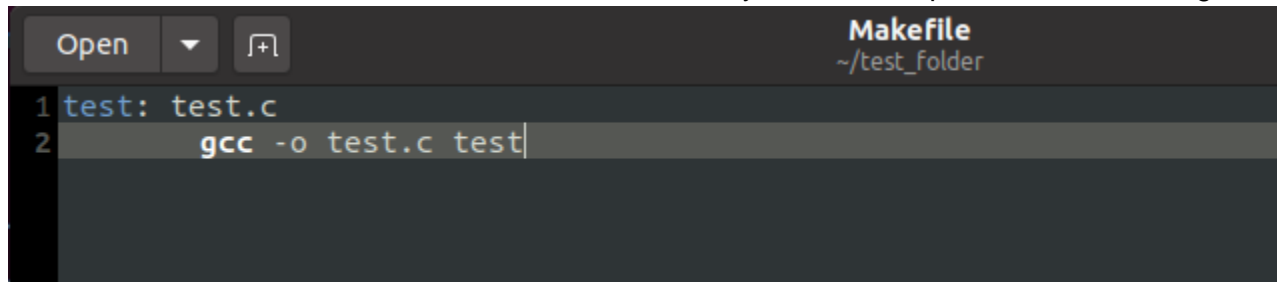
**Your Answer Here:** I use gcc command to create executable file.

gcc test.c will create executable file

then we can look on output using ./a.out

### Question 3

Below is a screenshot of a Makefile I created, I made a syntax error, explain what I did wrong and how I can fix it.



```
1 test: test.c
2 gcc -o test.c test
```

**Your Answer Here:** I think you reversed the order of c file and executable file. The code will return undefined reference to main.

The correct code of the above make file is :-

gcc test test.c

#### Question 4

Below is a screenshot of a c file called **do\_op.c**, what will the output be when I run this program? Use the space below to work through the code.

```
1 #include <stdio.h>
2
3
4 void main() {
5
6
7     int num1 = 5;
8
9     for (int k = 0; k < 25; k++) {
10
11         if (k % 5 == 0) {
12             num1 = num1 + 1;
13         }
14     }
15
16     printf("%d \n", num1++);
17
18 }
```

Your Answer Here: initially, num1=5, when I go inside the loop % sign means remainder or mod  
For k=0, 0 mod 5 is equal to zero so it will execute if statement add 1 to num1 becoming num1=6  
Similarly k=1,2,3,4,6,7,8,9,11,12,13,14,16,17,18,19,21,22,23,24 it will not execute if statement because k mod 5 is not equal to zero  
If statement execute only when k=0,5,10,15,20 each time it will add 1 to num1.  
So final output :- 10

# Binary, Hex, and Bit Manipulation

## Question 1

Convert the following decimal numbers into binary and hexadecimal: (I understand that you can just Google these conversions, but please do this on paper using the conversion techniques we learned! You **MUST SHOW ALL** work to get credit)

| Decimal Number | Binary Representation (use 16-bits for each number) | Hexadecimal Representation |
|----------------|---|----------------------------|
| 18             | 10010   | 12                         |
| 45             | 101101  | 2D                         |
| 800            | 1100100000  | 320                        |

Question: → 1

Decimal into Binary (?)<sub>2</sub> & Hexadecimal (?)<sub>16</sub>.

1. 18

we need to go from left to right.

$2^5$     $2^4$     $2^3$     $2^2$     $2^1$     $2^0$   
 32   16   8   4   2   1

we cannot use 32. since  $32 > 18$ .  
 so, we can make  $18 = 16 + 2$ .

so, taking 1 on place of 16 & 2.

$$(18)_{10} = (10010)_2$$

$$\begin{array}{r} 18 \\ -16 \\ \hline 2 \end{array}$$

Hexadecimal. we write in 4 bit combination for binary or simply do similar to binary.

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

$16^2$     $16^1$     $16^0$   
 256   16   1

Here  $18 < 256$ .

$$(18)_{10} = (12)_{16}$$

or Binary to Hex

$$\frac{(00010010)_2}{1 \quad 2} = (12)_{16}$$

2. 45

$2^6$     $2^5$     $2^4$     $2^3$     $2^2$     $2^1$     $2^0$   
 64   32   16   8   4   2   1  
 1   0   1   1   0   1

$$(45)_{10} = (101101)_2$$

grouping into 4 bit.

$$\frac{(00101101)_2}{2 \quad 13(0)} = (2D)_{16}$$

64 > 45.

$$\begin{array}{r} 45 \\ -32 \\ \hline 13 \end{array}$$

$13 < 16$ .

$$\begin{array}{r} 13 \\ -8 \\ \hline 5 \end{array}$$

$$\begin{array}{r} 5 \\ -4 \\ \hline 1 \end{array}$$



Question 1

~~18 into ba~~

2.800.

$$\begin{array}{cccccccccccc} 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1024 & 512 & 256 & 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\ (1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0)_{\text{2}} \end{array}$$

$$800 - 512 = 288$$

$$288 - 256 = 32$$

$$(800)_{10} = (1100100000)_2$$

grouping into 4-bit.

$$\left( \underbrace{0011}_3 \underbrace{0010}_2 \underbrace{0000}_0 \right)_2 = (320)_{16}$$

## Question 2

In what kind of situation(s) would we want to use the hexadecimal number system?

**Your Answer Here:** we can define location in memory using two hexadecimal digit compared to eight digit (binary). Hexadecimal is compact than binary, and decimal can be used for large binary number as it is easy to convert from Hexa to binary.

## Question 3

Is it possible to multiply a number by 8 by only using a shifting operator? If so, explain how? (hint: think about this with binary numbers, write a few examples down below).

**Your Answer Here:** we can use left shift ( $\ll$ ) to multiply the bit pattern with  $2^k$

Suppose number =  $N$

$$N \ll 1 = N \cdot (2^1) = 2N$$

$$N \ll 2 = N \cdot (2^2) = 4N$$

$$N \ll 3 = N \cdot (2^3) = 8N$$

## Question 4

Perform the following bitwise operations (steps 1-3):

1. XOR 21 (00010101) with 9 (00001001)
2. AND the result of **step 1** with 15 (00001111)
3. OR the result of **step 2** with 1 (00000001)

After these operations, what decimal number do we have?

**Your Answer Here:** after step1, I have 28

: after step2, I have 12

: after step3, I have 13. I have attached all the calculation on the picture below.

Question 4

1. XOR 21 (00010101) with 9 (00001001).

$$\begin{array}{r} 00010101 \\ (XOR) \ 00001001 \\ \hline 00011100 \end{array}$$

$(00011100)_2 = (28)_{10}$   
16/8/4

| A | B | A XOR B |
|---|---|---------|
| 0 | 0 | 0       |
| 0 | 0 | 0       |
| 0 | 0 | 0       |
| 1 | 1 | 0       |
| 0 | 0 | 0       |
| 1 | 0 | 1       |
| 1 | 0 | 1       |

STEP-2.  $00011100$  AND with 15 (00001111).

$$\begin{array}{r} 00011100 \\ (AND) \ 00001111 \\ \hline 00001100 \end{array}$$

$(00001100)_2 = 12$   
8/4

| A | B | A AND B |
|---|---|---------|
| 0 | 0 | 0       |
| 0 | 0 | 0       |
| 0 | 0 | 0       |
| 1 | 0 | 0       |
| 0 | 1 | 0       |
| 1 | 1 | 1       |
| 1 | 1 | 1       |

STEP-3.  $(00001100)_2$  OR with 1 (00000001)

$$\begin{array}{r} 00001100 \\ (OR) \ 00000001 \\ \hline 00001101 \end{array}$$

$(00001101)_2 = (13)_{10}$   
8/4/1

| A | B | A OR B |
|---|---|--------|
| 0 | 0 | 0      |
| 0 | 0 | 0      |
| 0 | 0 | 0      |
| 1 | 0 | 1      |
| 0 | 0 | 0      |
| 1 | 0 | 1      |
| 1 | 1 | 1      |

# Functions

## Question 1

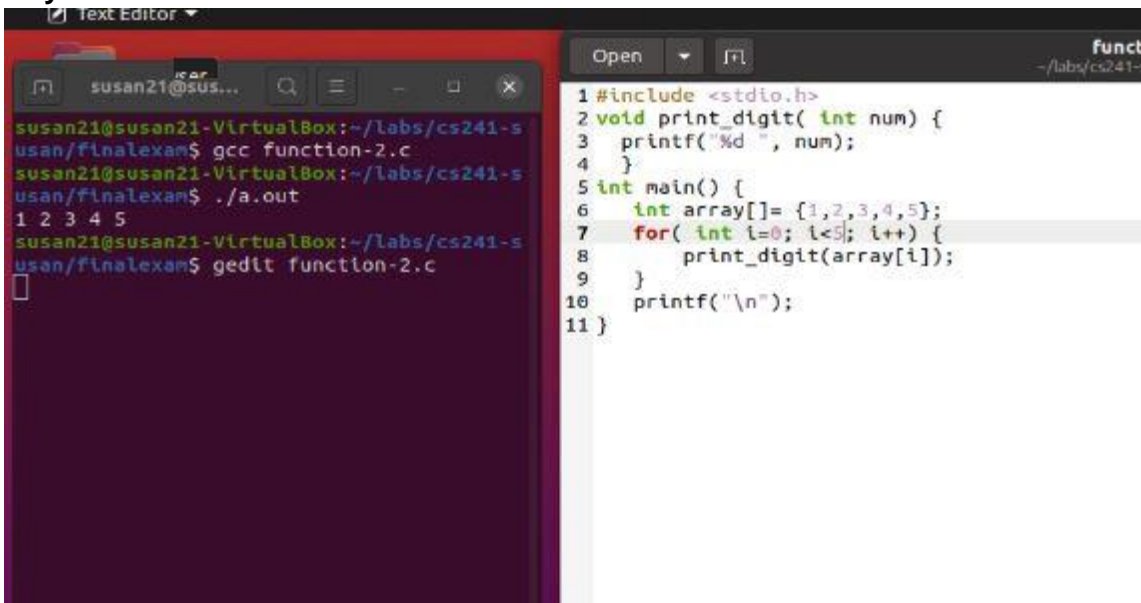
Explain the difference between call by value and call by reference.

**Your Answer Here:** call by reference modify original value but call by value keeps the original values. In the call by reference, we pass variable address into function but in call by value we pass copy of the variable. We use pointer for call by reference to store address of variable whereas in call by value we pass copy of variable itself. Call by reference copies address of argument into formal whereas in call by value, copies of value is passed into formal. If we call by value, then two value will be created in different location for actual and formal but in call by reference we use just address so no different location will be used. If we make changes then it won't be passed outside the function in call by value but will be passed outside function.

## Question 2

Is it possible to pass an array to a function using call by value? Explain why or why not.

**Your Answer Here:** I think it is possible to pass an array to a function using call by value. I kind of did two simple program to prove it. The first one use element of array to call function and other one directly use full array in function.

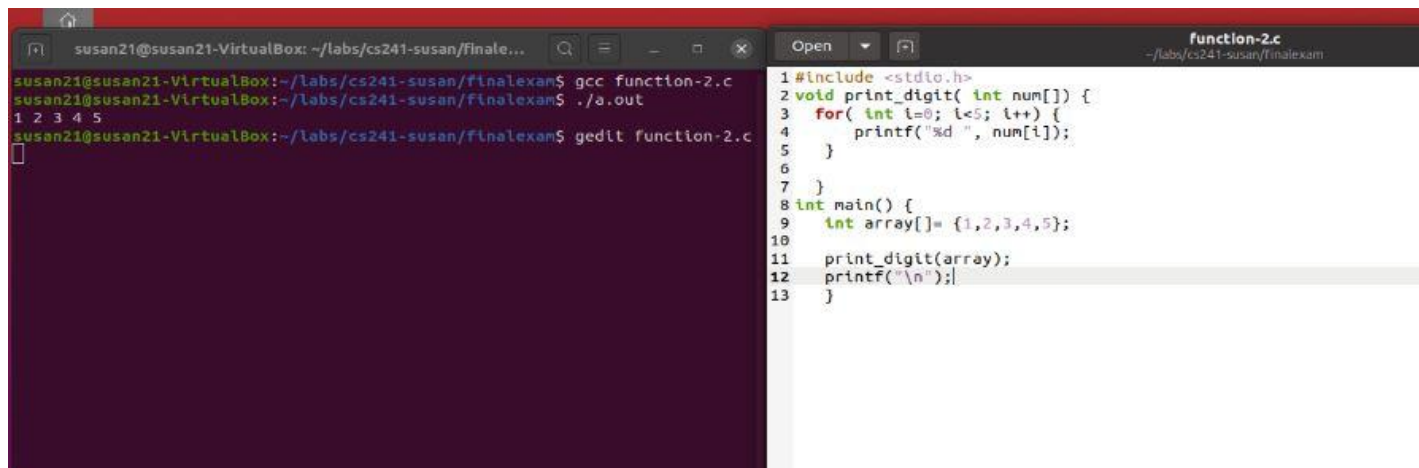


The image shows a terminal window on the left and a text editor on the right. The terminal window displays the following commands and output:

```
susan21@susan21-VirtualBox: ~/labs/cs241-s
usan/finalexan$ gcc function-2.c
susan21@susan21-VirtualBox: ~/labs/cs241-s
usan/finalexan$ ./a.out
1 2 3 4 5
susan21@susan21-VirtualBox: ~/labs/cs241-s
usan/finalexan$ gedit function-2.c
```

The text editor shows the following C code:

```
1 #include <stdio.h>
2 void print_digit( int num) {
3     printf("%d ", num);
4 }
5 int main() {
6     int array[] = {1,2,3,4,5};
7     for( int i=0; i<5; i++) {
8         print_digit(array[i]);
9     }
10    printf("\n");
11 }
```



The image shows a screenshot of a virtual machine environment. On the left is a terminal window with the following commands and output:

```
susan21@susan21-VirtualBox: ~/labs/cs241-susan/finalexam$ gcc function-2.c
susan21@susan21-VirtualBox: ~/labs/cs241-susan/finalexam$ ./a.out
1 2 3 4 5
susan21@susan21-VirtualBox: ~/labs/cs241-susan/finalexam$ gedit function-2.c
```

On the right is a code editor window titled "function-2.c" showing the following C code:

```
1 #include <stdio.h>
2 void print_digit( int num[]) {
3     for( int i=0; i<5; i++) {
4         printf("%d ", num[i]);
5     }
6 }
7
8 int main() {
9     int array[] = {1,2,3,4,5};
10
11     print_digit(array);
12     printf("\n");
13 }
```

In c, array can be considered as pointer so, it is easier to do calling by reference. The only difference will be I can changer pointer direction.

### Question 3

Below you will see two screenshots (option 1 and option 2), which of these is valid C code that will compile and run?

```
1 #include <stdio.h>
2
3 int do_op(int x) {
4     for (int k = 0; k < 25; k++) {
5         if (k % 5 == 0) {
6             x = x + 1;
7         }
8     }
9     return x++;
10 }
11
12 void main() {
13     int num1 = 5;
14     printf("%d \n", do_op(num1));
15 }
```

\*Option 1

```
1 #include <stdio.h>
2
3 int do_op(int x);
4
5 void main() {
6     int num1 = 5;
7     printf("%d \n", do_op(num1));
8 }
9
10 int do_op(int x) {
11     for (int k = 0; k < 25; k++) {
12         if (k % 5 == 0) {
13             x = x + 1;
14         }
15     }
16     return x++;
17 }
```

\*Option 2

Choose the correct option here, delete the 2 options that are incorrect so that you are only left with one:

Both. Both the code compile and run only difference between 1 and 2 is 2 has declare the function at the beginning because the function is below void main and in 1, we don't have to declare the function because function is before the void main.



# Structs

## Question 1

What is a struct and why is it useful?

**Your Answer Here:** struct is user defined data type which helps us to store combination of different data types like char, float, integer and so on. We have to define the data type in array before using it and can only store same kind of data type. Struct has similar use to array but they can store combination of data type to keep record of necessary things. For example, we want to keep track of all citizen and their information like name, age, date of birth, criminal record, marriage status, gender, ethnicity and so on then we can use struct to keep record of all information.

# Pointers

## Question 1

How can we use pointers to write more efficient code in terms of speed and memory usage? (hint: think about your lab 4 results)

**Your Answer Here:** we can use pointer to write more efficient code in terms of speed and memory usage because by using pointer, we manipulate data by manipulating pointer address which is direct link to memory location. Pointer assigns and release the memory as we did in the lab 4 the second code using the pointer and also pointer can dynamically allocate memory. In the lab4, we kind of did the reverse row method the first was hard to code, so many things involve like making temporary string and clearing out before copying temporary string to line which will take extra time rather than pointer which access just address.

## Question 2

Below you will see a screenshot of code that uses the **strtok** function, briefly explain how this function works. Your explanation must include details about how the function parses the data and how it uses a pointer to do this.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 void main() {
5
6     char my_string[50] = "Data_1, Data_2, Data_3";
7
8     char *token;
9     token = strtok(my_string, ", ");
10
11     while (token != NULL) {
12
13         printf("The token is -> %s \n", token);
14         token = strtok(NULL, ", ");
15     }
16
17 }
```

Your Answer Here: strtok splits given string into series of token according to given delimiters. Here in the code above, the Delimiter is “, “. The first line code char my\_string declare string with information of size 50. Then char\* token define pointer to use on the code. The line 9 use strtok function to get first token. The line 11 loop use token as pointer to check for delimiter. If the token is not null while looping then it will print the statement (line 13) then line 14 is going through other token if it is there inside the loop. So, the output will be

The token is -> Data\_1  
The token is -> Data\_2  
The token is -> Data\_3

# Malloc

## Question 1

In this class and on labs 6 & 7 we saw how we can use malloc to dynamically allocate space, explain what is meant by dynamically allocating space and how it is useful.

**Your Answer Here:** Dynamic Memory allocation is process in which size of data structure is varying during program processing. For example, we store 10 element creating array of 10 element and after few step if we feel we need to store extra 5 element then we can change the length of array to 15 using dynamic memory allocation process. It is useful as we can allocate and free memory according to our need. The malloc function that we uses in lab used to allocate a block of memory dynamically which reserve memory size and returns null pointer to memory location which can be assigned to pointer. We don't need to have idea how much particular structure is occupying in memory space.

# Linked Lists

## Question 1

**Part A.-** Name and explain 2 advantages that a linked list has over a traditional array.

1. Dynamic size:- we can change the size of data structure according to programing need at run time. This means we don't have to worry about size of linked list we can allocate and deallocate memory during runtime according to our purpose.
2. insertion/Deletion:- in case of array, if we want to delete or add then we have to shift element which is kind of tedious task and require more code and expensive. Insertion or Deletion can be done in linked list by using pointer address which is much more efficient.

**Part B.-** Name and explain 2 advantages that a traditional array has over a linked list

1. memory:- we need to have more memory to store linked list than array. Extra storage is required for each element in linked list in order to create space for the pointer. But in array we don't need extra space to pointer to reference to next node.

2. Randomness:- in linked list, we have to access data in sequential order. This means we cannot access 5<sup>th</sup> node without going from 1<sup>st</sup>, 2<sup>nd</sup> and so on. So, we cannot randomly choose any node which can be done array. Each element in array can be accessed in less time than linked list. According to book, we cannot perform binary search using linked list.