C & OS ASSIGNMENTS

bhagavan@auranetworks.in

www.auranetworks.in

C & OS ASSIGNMENTS

I. C Programming Language

1. Compilation Stages

1.1 What are object (.o or .obj) files and what is their format?

1.2 How to compile a program that contains two or more C source files?

1.3 What is -o option for the C compiler?

1.4 What is the -c option for the C compiler?

1.5 What is a.out file and what is its format?

$ nm a.out

$ objdump a.out

1.6 Preprocessing

1.6.1 Compile below program like below and compare both the files

```
$ gcc main.c -E -o main.i

$ gvim -d main.c main.i
```

main.c

```
#define MAX 5

main()
{
	for(i = 1; i <= MAX; i++)
	{
		printf("-->%d\n", i);
	}
}
```

1.6.2 Compile below program like below and compare both the files

```
$ gcc main.c -E -o main.i

$ gvim -d main.c main.i
```

defs.h

```
#define MAX 5
struct newst
{
	int a;
	int b;
}
```

main.c

```
#include "defs.h"
main()
{
	for(i = 1; i <= MAX; i++)
	{
		printf("-->%d\n", i);
	}
}
```

1.6.3 What is the expected and actual result of the below program?

```
main()
{
	int val = 0;

	printf("-->%d\n", val);
```

```
}
```

1.7 Compilation

1.7.1 What kind of compilation error does below program gives?

main.c

```
main()
{
	int a = 10;
	float f = 5.5;
	float retval = 0;

	retval = my_multi_fun(a, f);

	printf("%f", retval);
}
```

fun.c

```
int my_multi_fun(int x, int y)
{
	return x * y;
}
```

1.7.2 What is the expected and actual result of the below program?

defs.h

```
int my_multi_fun(int x, int y);
```

fun.c

```
int my_multi_fun(int x, int y)
{
	return x * y;
}
```

main.c

```
#include "defs.h
main()
{
	int a = 10;
	float f = 5.5;
	float retval = 0;

	retval = my_multi_fun(a, f);

	printf("%f", retval);
}
```

1.8 Linking

1.8.1 What kind of error is occurs while compiling below program?

fun.c

```
int MY_multi_fun(int x, int y)
{
	return x * y;
}
```

main.c

```
main()
{
	int a = 10;

	my_multi_fun(a);
}
```

2. Storage Section & Sections

2.1 Write a program which contains four simple functions and a main function. Keep each function in a separate file total five files. Each function gets called from main and prints "I am function 1 in file 1".

2.2 Make any one of the above four functions as static function and compile. Try to explain what kind of compilation error occurs.

2.3 Declare one initialized global variable in each function of above program and print all variables in all functions

2.4 Make all the global variable of the above program as static and compile and run

2.5 Print addresses of all variables of below program and note down them in descending order

2.6 Also run below command on executable image

```
$ size a.out
```

.

```
#include "stdio.h"
#include "stdlib.h"

/*
1. Print the addresses of all variables.
Based on the address (value) arrange them
in descending order
NOTE: Also Print ptr, &ptr, fun1, fun2 and
fun3
2. List all Local variables
3. List all Global variables
*/

int   idata1=1;
char  carray1[10];
char  cname1[128] = "Aura Networks";
char  *pname1 = "Aura Networks";
int   idata2;
int   idata3 = 3;
int   idata4;
short sh1;
short sh2 = 2;
int   iarray1[10] = {1, 2};
int   iarray2[10] = {10, 20};
short sh3;
short sh4 = 4;
char  ch1 = 1;
char  ch2;
short sharray3[10] = {5, 6};
char  ch3 = 3;

void fun1();
void fun2();
void fun3();
```

```
main()
{
      int ldata1;
      int ldata2 = 2;
      int  ldata3;
      static int sdata1;
      static int sdata2 = 10;
      char  cname2[128] =
                  "Aura Networks";
      char  *pname2 =
                  "Aura Networks";

      int *ptr=malloc(10);

      printf("%p\n", ptr);
      printf("%p\n", &ptr);

      printf("%p\n", cname1);
      printf("%p\n", pname1);
      printf("%p\n", &pname1);

      printf("%p\n", cname2);
      printf("%p\n", pname2);
      printf("%p\n", &pname2);

      printf("%p\n", fun1);
      printf("%p\n", fun2);
      printf("%p\n", fun3);

      ptr = malloc(10);
      printf("%p\n", ptr);

      printf("%p\n", iarray1);
      printf("%p\n", iarray2);
      printf("%p\n", iarray3);

}

void fun1( )
{
      int ldata4;

      printf("I am funcion1\n");
}

int idata5 = 10;
```

```
void fun2( )
{
      printf("I am function 2\n");
}

void fun3()
{
      int ldata5 = 10;
      printf("I am function 3\n");
}

int idata6 = 20;
```

2.7 Automatic

2.7.1 What is automatic variable?

2.7.2 How automatic variable is different from normal variable (without storage specifier)?

2.7.3 What is the difference of the output of below programs?

```
main()
{
        auto int a = 10;
        auto int b;

        printf("-->a :%d\r\n", a);
        printf("-->b :%d\r\n", b);
}
```

```
main()
{
        int a = 10;
        int b;

        printf("-->a :%d\r\n", a);
        printf("-->b :%d\r\n", b);
}
```

2.7.4 What is the output of the below program?

Compile below programs together 'gcc main.c fun1.c'

main.c

```
main()
{
        auto int a = 10;

        printf("-->a :%d\r\n", a);

        my_function1();
        my_function1();
}
```

fun1.c

```
void my_function1(void)
{
        auto int temp = 10;

        printf("-->temp :%d\r\n", temp);

        temp++;
}
```

2.7.5 What is the output of the below program?

Compile below programs together 'gcc main.c fun1.c'

main.c

```
main()
{
        auto int a = 10;

        printf("-->a :%d\r\n", a);

        my_function1();
        my_function1();
}
```

fun1.c

```
void my_function1(void)
{
        int temp = 10;

        printf("-->temp :%d\r\n", temp);

        temp++;
}
```

2.7.6 Does below program compiles? If not, why? At what stage compilation gets stopped?

```
auto int a = 10;
auto int b;

main()
{
        printf("-->a :%d\r\n", a);
        printf("-->b :%d\r\n", b);
}
```

2.7.7 Compile below two programs together 'gcc main.c fun1.c"

a. Do they compile? If not, why?
b. At what stage 'main.c' stops compilation? And why?
c. At what stage 'fun1.c' stops compilation? And why?

main.c

```
int a = 10;
int b;

main()
{
      printf("-->a :%d\r\n", a);
      printf("-->b :%d\r\n", b);

      my_function1();
}
```

fun1.c

```
void my_function1(void)
{
      printf("-->a :%d\r\n", a);
      printf("-->b :%d\r\n", b);
}
```

2.7.8 Compile below two programs together 'gcc main.c fun1.c"

a. Do they compile? If not, why?
b. At what stage 'main.c' stops compilation? And why?
c. At what stage 'fun1.c' stops compilation? And why?

main.c

```
auto int a = 10;
int b;

main()
{
      printf("-->a :%d\r\n", a);
      printf("-->b :%d\r\n", b);

      my_function1();
}
```

fun1.c

```
void my_function1(void)
{
      printf("-->a :%d\r\n", a);
      printf("-->b :%d\r\n", b);
}
```

2.7.9 Compile below two programs together 'gcc main.c fun1.c"

a. Do they compile? If not, why?
b. At what stage 'main.c' stops compilation? And why?
c. At what stage 'fun1.c' stops compilation? And why?
d. How to fix the compilation error? And explain the reason?

defs.h

```
auto int a = 10;
#define NUMBER 5
```

main.c

```
#include "defs.h"

main()
{
```

fun1.c

```
#include "defs.h"

void my_function1(void)
{
      printf("-->MAX :%d\r\n", MAX);
      printf("-->a :%d\r\n", a);
}
```

```
      printf("-->MAX :%d\r\n", MAX);
      printf("-->a :%d\r\n", a);

      my_function1();
}
```

2.7.10 Compile below two programs together 'gcc main.c fun1.c"

e. Do they compile? If not, why?
f. At what stage 'main.c' stops compilation? And why?
g. At what stage 'fun1.c' stops compilation? And why?
h. How to fix the compilation error? And explain the reason?

fun1.c

```
int a = 20;
void my_function1(void)
{
      printf("-->a :%d\r\n", a);
}
```

main.c

```
int a = 10;
main()
{
      printf("-->a :%d\r\n", a);

      my_function1();
}
```

2.8 Extern

2.8.1 What is extern variable?

2.8.2 How extern variable is different from normal variable (without storage specifier)?

2.8.3 When do we declare the variable as 'extern'?

2.8.4 Compile below two programs together 'gcc main.c fun1.c" Do they compile? If not, why?

main.c

```
int a = 10;

main()
{
      printf("-->a :%d\r\n", a);

      my_function1();
}
```

fun1.c

```
extern int a;

void my_function1(void)
{
      printf("-->a :%d\r\n", a);
}
```

2.8.5 Compile below two programs together 'gcc main.c fun1.c"

a. Do they compile? If not, why?
b. At what stage 'main.c' stops compilation? And why?
c. At what stage 'fun1.c' stops compilation? And why?

main.c

```
main()
{
      printf("-->a :%d\r\n", a);

      my_function1();
```

fun1.c

```
extern int a;

void my_function1(void)
{
      printf("-->a :%d\r\n", a);
```

```
}
```

```
}
```

2.8.6 Compile below two programs together 'gcc main.c fun1.c"

a. Do they compile? If not, why?
b. At what stage 'main.c' stops compilation? And why?
c. At what stage 'fun1.c' stops compilation? And why?

main.c

```
int a = 10;
int b;

main()
{
      printf("-->a :%d\r\n", a);
      printf("-->b :%d\r\n", b);

      my_function1();
}
```

fun1.c

```
extern int a;

void my_function1(void)
{
      printf("-->a :%d\r\n", a);
      printf("-->b :%d\r\n", b);
}
```

2.8.7 Compile below two programs together 'gcc main.c fun1.c fun2.c"

a. Do they compile? If not, why?
b. At what stage 'main.c' stops compilation? And why?
c. At what stage 'fun1.c' stops compilation? And why?
d. How to fix the compilation error? And explain the reason?

main.c

```
int a = 10;
int b = 20;
#include "defs.h"

main()
{
      printf("-->MAX :%d\r\n", MAX);
      printf("-->a :%d\r\n", a);
      printf("-->b :%d\r\n", b);

      my_function1();
      my_function2();
}
```

defs.h

```
extern int a;
#define NUMBER 5
```

fun1.c

```
#include "defs.h"
extern int b;

void my_function1(void)
{
      printf("-->MAX :%d\r\n", MAX);
      printf("-->b :%d\r\n", b);
}
```

fun2.c

```
#include "defs.h"
extern int b = 20;

void my_function2(void)
{
      printf("-->MAX :%d\r\n", MAX);
      printf("-->a :%d\r\n", a);
      printf("-->b :%d\r\n", b);
}
```

2.8.8 Compile below two programs together 'gcc main.c fun1.c fun2.c"

a. Do they compile? If not, why?
b. At what stage 'main.c' stops compilation? And why?

c. At what stage 'fun1.c' stops compilation? And why?
d. How to fix the compilation error? And explain the reason?

main.c
```
int a = 10;
#include "defs.h"
main()
{
	printf("-->MAX :%d\r\n", MAX);
	printf("-->a :%d\r\n", a);

	my_function1();
	my_function2();
}
```

defs.h
```
extern int a;
#define NUMBER 5
```

fun1.c
```
#include "defs.h"

void my_function1(void)
{
	printf("-->MAX :%d\r\n", MAX);
	printf("-->a :%d\r\n", a);
}
```

fun2.c
```
#include "defs.h"

void my_function2(void)
{
	printf("-->MAX :%d\r\n", MAX);
	printf("-->a :%d\r\n", a);
}
```

2.8.9 Compile below two programs together 'gcc main.c fun1.c" Do they compile? If not, why?

a. Do they compile? If not, why?
b. At what stage 'main.c' stops compilation? And why?
c. At what stage 'fun1.c' stops compilation? And why?
d. How to fix the compilation error? And explain the reason?

main.c
```
int a = 10;

main()
{
	printf("-->a :%d\r\n", a);

	my_function1();
	my_function2();
}
```

fun1.c
```
void function1(void)
{
	printf("-->a :%d\r\n", a);
}
```

fun2.c
```
void my_function2(void)
{
	printf("-->a :%d\r\n", a);
}
```

2.9 Static

2.9.1 Compile below two programs together 'gcc main.c fun1.c func2.c"

a. Do they compile? If not, why?
b. At what stage 'main.c' stops compilation? And why?
c. At what stage 'fun1.c' stops compilation? And why?
d. How to fix the compilation error? And explain the reason?

main.c
```
static a = 10;

main()
{
```

fun1.c
```
static int a = 20;

void my_function1(void)
{
```

```
      printf("-->a :%d\r\n", a);

      my_function1();
      my_function2();
}
```

```
      printf("-->a :%d\r\n", a);
}
```

fun2.c

```
static int a = 30;

void my_function2(void)
{
      printf("-->a :%d\r\n", a);
}
```

2.9.2 Compile below two programs together 'gcc main.c fun1.c fun2.c". What is the value of 'a' in all functions?

main.c

```
int a = 10;

main()
{
      printf("-->a :%d\r\n", a);
      my_function1();
      my_function2();
      printf("-->a :%d\r\n", a);
}
```

fun1.c

```
static int a = 20;

void my_function1(void)
{
      printf("-->a :%d\r\n", a);
      a++;
      printf("-->a :%d\r\n", a);
}
```

fun2.c

```
static int a = 30;

void my_function2(void)
{
      printf("-->a :%d\r\n", a);
      a++;
      printf("-->a :%d\r\n", a);
}
```

2.9.3 Compile below two programs together 'gcc main.c fun1.c". What is the output of the below program?

main.c

```
int a = 10;

main()
{
      printf("-->a :%d\r\n", a);
      my_function1();
      printf("-->a :%d\r\n", a);
      my_function1();
      printf("-->a :%d\r\n", a);
}
```

fun1.c

```
void my_function1(void)
{
      static int a = 20;
      printf("-->a :%d\r\n", a);
      a++;
      printf("-->a :%d\r\n", a);
}
```

2.9.4 In above program how come the local variable in 'my_function1' is retaining its value across the function calls?

2.9.5 Compile below two programs together 'gcc main.c fun1.c"

a. Do they compile? If not, why?

b. At what stage 'main.c' stops compilation? And why?

c. At what stage 'fun1.c' stops compilation? And why?
d. How to fix the compilation error? And explain the reason?
e. What do we mean 'file scope variable'?

main.c

```
static int a = 10;

main()
{
      printf("-->a :%d\r\n", a);
      my_function1();
      printf("-->a :%d\r\n", a);
      my_function1();
      printf("-->a :%d\r\n", a);
}
```

fun1.c

```
void my_function1(void)
{
      printf("-->a :%d\r\n", a);
      a++;
      printf("-->a :%d\r\n", a);
}
```

2.9.6 What does it mean a function declared as 'static'?

2.9.7 How to declare a 'file scope function'

2.10 Register

2.10.1 What is the 'register' variable? How to declare it?

2.10.2 How register variable is different from normal variable and static variables?

2.10.3 Can we print the address of 'register' variable?

2.10.4 In what cases 'register' variable is used?

2.11 Volatile

2.11.1 What is the 'volatile' variable? How to declare it?

2.11.2 How is it different from normal variable and static variables?

2.11.3 Can we print the address of 'volatile' variable?

2.11.4 In what cases can we use 'volatile' variable?

3. File Operations

3.1 Create a file 't.txt' with below content

```
            C is the most commonly used programming      language for writing
operating systems.
Unix was the first operating system written in C. 121
Later Microsoft Windows, Mac OS X, and GNU/Linux were all written in C. Not
only is C the language of  madam  operating     systems, Madam it is the
precursor and
      Inspiration for almost all of the most popular high-level a languages
available today, Perl, PHP, 12321 and Python are all written in C
```

3.2 How to pass command line arguments to the program? Write a program which takes command line arguments and prints them.

3.3 Use and experiment with below commands

```
$ cat t.txt  //Prints the entire content of the file on to screen
$ wc t.txt   //Prints number of Characters, Words and lines in 't.txt'
```

3.4 Implement your own 'cat' command called 'acat'. This should give the same output as 'cat' command.

```
$ acat t.txt
```

3.5 Implement your own 'wc' command called 'awc'. This should give the same output as 'wc' command

```
$ awc t.txt
```

3.6 Implement your own 'cp' command called 'acp'. This should copy 't.txt' content to 'a.txt'

```
$ acp t.txt a.txt
```

3.7 Create a new command 'ancp' which copies first and last 'n' bytes from source file to destination file

```
Syntax: $ ancp <source file> <destination file> <number of bytes>

Ex: $ ancp t.txt a.txt 10  //Copy first and last 10 bytes of t.txt to a.txt
```

3.8 Implement your own 'acpcc' command. This copies 't.txt' content to 'a.txt' by toggling Upper case to lower case and vice versa.

```
$ acpcc t.txt a.txt
```

3.9 Create a new command 'cpalindromes' which counts number of palindromes in 't.txt'

```
$ cpalindromes t.txt a.txt
```

4. Remote Procedural Calls (RPC Engine) - Project on File Management

4.1 Purpose

This module enables disk-less system/device to create/store files on remote file system

4.2 Pre-Requisites – C

4.2.1 Structures & Unions

- Structures - Populating and printing values
- Structures - Size of structures, holes in a structures, Advantages and Dis-Advantages with holes
- Print base address of the structure
- Print base address of each members of the structure
- Print the sizeof a structure whose members are int, char, int (i, c, i)
- Similarly, print the sizes of different structures with different members like below
- i, i, c c, c, i s, i, c s, s, c i, s, c

4.2.2 Basic File operations - open, read, write, lseek, close

4.2.3 Knowledge of sharing files across PCs (Windows/Linux)

4.2.4 **State1.** Modify above 'acp' program such that only wrapper functions are called from main to do all file operations. i.e.

- Call 'aopen' instead of 'open'
- Call 'aread' instead of 'read'
- Call 'awrite' instead of 'write'
- Call 'alseek' instead of 'lseek'

- Call 'aclose' instead of 'close'

```
main()                          int aopen(char *filename, int flags, int mode)
{                               {
      aopen                           open  //actual open
      ...                       }
      aread
      ...
      aclose
}
```

4.2.5 **Stage2:** Modify above program such that even wrapper function 'aopen' do not call actual 'open'. Instead, it calls the one more wrapper function rpc_open. Chage all other wrapper functions to call rpc_xxxx functions

- Call 'rpc_open' instead of 'open'
- Call ' rpc_read' instead of 'read'
- Call ' rpc_write' instead of 'write'
- Call ' rpc_lseek' instead of 'lseek'
- Call ' rpc_close' instead of 'close'

main.c

```
main()
{
      aopen
      ...
      aread
      ...
      aclose
}
```

afop.c

```
int aopen(char *filename, int flags, int mode)
{
      rpc_open(…)
}
```

rpc_fop.c

```
int rpc_open(char *filename, int flags, int mode)
{
      open(…)
}
```

4.2.6 **Stage3:** Modify above program such that rpcc_open' do not call actual 'open'. Instead, it calls the one more wrapper function rpcc_fop. Chage all other wrapper functions to call the **same rpcc_fop** function

- Call 'rpcc_fop' instead of 'open'
- Call ' rpcc_ fop ' instead of 'read'
- Call ' rpcc_ fop ' instead of 'write'
- Call ' rpcc_ fop ' instead of 'lseek'
- Call ' rpcc_ fop ' instead of 'close'

main.c

```
main()
{
      aopen
      ...
      aread
      ...
      aclose
}
```

afop.c

```
int aopen(char *filename, int flags, int mode)
{
      rpc_open(…)
}
```

rpcc_fop.c

```
int rpcc_fop(…)
{
```

rpc_fop.c

```
int rpc_open(char *filename, int flags, int mode)
{
```

```
      open
}
```

```
      rpcc_fop(…)
}
```

rpcc_fop.c

```
int rpcc_fop(……)
{
      open
      ...
      read
      ...
      close

}
```

4.2.7 **Stage4:** rpcc_fop function sends a network request to server to call actual 'open' on a remote system. The request consist of all required parameters for all file operations

4.2.8 **State5:** Upon reading request server does actual file operations and sends the replay with required arguments.

5. Scope of variables

5.1 Guess what is the output of the below program?

main.c

```
int a = 10;

main()
{
      printf("-->a :%d\r\n", a);
      my_function1();
      printf("-->a :%d\r\n", a);
      my_function2();
      printf("-->a :%d\r\n", a);
      my_function1();
      printf("-->a :%d\r\n", a);
      new_fun();
      printf("-->a :%d\r\n", a);
      my_function2();
      printf("-->a :%d\r\n", a);
      new_fun();
      printf("-->a :%d\r\n", a);
      a = my_function3();
      printf("-->a :%d\r\n", a);
}
```

```
void my_function1(void)
{
      printf("-->a :%d\r\n", a);
      a++;
      printf("-->a :%d\r\n", a);
}

void my_function2(void)
{
      static int a = 20;
      printf("-->a :%d\r\n", a);
      a++;
      printf("-->a :%d\r\n", a);
}

void my_function3(void)
{
      int a = 30;
      printf("-->a :%d\r\n", a);
      a++;
      printf("-->a :%d\r\n", a);
      return a;
}
```

new_fun.c

```
static int a = 50;

int new_fun()
{
      printf("-->a :%d\r\n", a);
      a++;
```

```
                                        printf("-->a :%d\r\n", a);
                                }
```

5.2 Guess what is the output of the below program?

main.c

```
int a = 10;
main()
{
      int a = 20;
      int i = 5;
      printf("-->a :%d\r\n", a);

      if (i > 3)
      {
            int a = 30;

            printf("-->a :%d\r\n", a);
            a++;
      }
      printf("-->a :%d\r\n", a);
}
```

5.3 Define Block Scope, Function Scope, File Scope and Global scope variables with examples

5.4 Define 'static function' properties

5.5 When do we define any function as 'static function'?

6. Pointers Basics

6.1 If the value of variable ‘a’ is like below

```
int a = 0x12131415;

Print the value of ‘a’ as '0x12131415'
Print each individual byte's value of ‘a’ '0x12 0x13 0x14 0x15'
Print both short int values of ‘a’ as  '0x1213 0x1415'
Print each individual byte's value of ‘a’ in reverse order '0x15 0x14 0x13 0x12
```

6.2 Fill each byte value of 'a' to character variables ch1, ch2, ch3, ch4 and print

```
int a = 0x12131415

i.e. ch1 contains 0x12
     ch2 contains 0x13
     ch3 contains 0x14
     ch4 contains 0x15
```

6.3 Fill both short int values of ‘a’ to short int variables sh1 and sh2 and print

```
Int a = 0x12131415

i.e. sh1 contains 0x1213
     sh1 contains 0x1415
```

6.4 Swap byte values of a from 1 to 4, and 2 to 3 and print each byte value

```
int a = 0x12131415

i.e. after swapping value of ‘a’ is 0x151141312
```

6.5 Swap short int values of 'a'

```
int a = 0x12131415

i.e. after swapping value of 'a' is 0x15114131
```

7. Pointers and Arrays

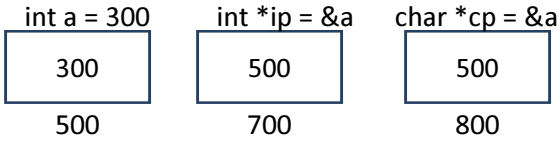
7.1 What is the minimum and maximum decimal value can be stored in in 1 'bit', 2, 3, 5, 7 and 8 bits?

7.2 What is the minimum and maximum decimal value can be stored in 1 'Byte'?

7.3 Why the range of values is different in 'char' and 'unsigned char'?

7.4 What is the minimum and maximum decimal value of if the base address of the array "str" is 500, fill the columns "TYPE" and "VALUE" against each notation.

7.5 The address and values of the variables a, *ip and *cp are like below. Please fill the columns "TYPE" and "VALUE" against each notation



Notation	TYPE	Value
a		
&a		
ip		
&ip		
*ip		
*&ip		
**&ip		
*&a		
cp		
&cp		
*cp		
*&cp		
**&cp		

7.6 If the base address of the array "str" is 500, fill the columns "TYPE" and "VALUE" against each notation.

char str[15] = "Aura Networks";

Notation	TYPE	Value
str		
&str		

str+1		
&str+1		
*&str		
*str		
**&str		

7.7 If the base address of the array "dstr" is 500, fill the columns "TYPE" and "VALUE" against each notation.

```
char dstr[5][15] = {
                        "Aura Networks",
                        "Bangalore",
                        "India"
                   };
```

Notation	TYPE	Value
dstr		
dstr[1]		
&dstr		
dstr[1][1]		
&dstr[1][1]		
dstr + 1		
&dstr + 1		
*&dstr		
**&dstr		
***&dstr		

7.8 Which of the below programs gives the same results and which one is in-correct and why?

```
main()
{
    int a = 10;
    int *p = &a;

    printf("%d\n", *p);
    *p = 20;
    printf("%d\n", *p);
}
```

```
main()
{
    int a = 10;
    int *p;

    p = &a;

    printf("%d\n", *p);
    *p = 20;
    printf("%d\n", *p);
}
```

```
main()
{
    int a = 10;
    int *p;

    *p = &a;

    printf("%d\n", *p);
    *p = 20;
    printf("%d\n", *p);
}
```

7.9 Which of the below programs gives the same results and which one is in-correct and why?

```
main()
{
  char p[20];

  strcpy(p, "ANetworks");

  printf("%s", p);
```

```
main()
{
  char arr[20];
  char *p;

  p = arr;
```

```
main()
{
  char *p = malloc(20);

  strcpy(p, "ANetworks");
  printf("%s", p);
```

```
}                                                                    }
                              strcpy(p, "ANetworks");
                              printf("%s", p);
                            }
main()
{
  char *p;

  strcpy(p, "ANetworks");
  printf("%s", p);
}
```

7.10 What is difference between malloc and calloc?

7.11 Declare pointers to different data types of below?

7.11.1 Pointer to char

7.11.2 Pointer to char array of 10 bytes

7.11.3 Pointer to char array of 10 strings and length 100 bytes each

7.11.4 Pointer to a 'constant int'

7.11.5 Constant pointer

7.11.6 Constant Pointer to a 'constant int'

7.12 Declare 'function pointers' to different function like below.

7.12.1 Function that returns int and take two arguments as int and int

7.12.2 Function that returns 'int pointer' takes int and float

7.12.3 Function that returns 'void' and takes 'void pointer' and 'int pointer'

7.12.4 Function that returns 'int pointer' and takes 'void pointer' and 'int pointer'

7.12.5 Function that returns a 'function pointer, which returns void pointer and takes as int and 'integer pointer'. And takes void pointer as an argument

8. Bit Manipulations

8.1 Write a function that determines the number of bits set to 1 in the binary representation of an integer?

```
int get_bit_count(int n);
```

8.2 How to find the number of bit swaps required converting integer A to integer B?

8.3 How do you find out if an unsigned integer is a power of 2?

```
int is_power_of_two(int n);
```

8.4 Define the below bit manipulation functions

8.4.1 Show Bits

```
int showbits(int n);
```

8.4.2 Swapping two integers using bit manipulation

```
void inplace_swap_B(int *pa, int *pb)
```

8.4.3 Swap Odd and Even Bits in an Integer

```
int swapEvenOddBits(int n)
```

8.4.4 Flipping n-th bit of an integer

```
void bit_flip(int& m, int nth)
```

8.4.5 Finding Endian'ness

```
int endian() //Returns 0 - Little, 1 - Big
```

8.4.6 Bit pattern palindrome of an integer

```
bool isPalindrome(int n)
```

8.4.7 Compute x & y using only | and ~

```
int bitAnd(int x, int y)
```

8.4.8 allOddBits return 1 if all odd numbered bits of x are set

```
int allOddBits(int x)
```

8.4.9 oddBits return an int with all odd numbered bits set

```
int oddBits(void)
```

8.4.10 ReplaceByte replaces byte n in x with c

```
int replaceByte(int x, int n, int c)
```

8.4.11 bitParity return 1 if x has an odd number of bits set

```
int bitParity(int x)
```

8.4.12 (x|y) using only & and ~

```
int bitNor(int x, int y)
```

8.4.13 Using only & and ~

```
int bitXor(int x, int y)
```

8.4.14 x != y?

```
int isNotEqual(int x, int y)
```

8.4.15 Extract byte n from x

```
int getByte(int x, int n)
```

8.4.16 Set all bits to LSB of x

```
int copyLSB(int x)
```

8.4.17 Logical right shift x by n

```
int logicalShift(int x, int n)
```

8.4.18 Compute !x without using ! operator

```
int bang(int x)
```

8.4.19 Mark least significant 1 bit

```
int leastBitPos(int x)
```

8.4.20 largest two's complement integer

```
int tmax(void)
```

8.4.21 x >= 0?

```
int isNonNegative(int x)
```

8.4.22 x > y?

```
int isGreater(int x, int y)
```

8.4.23 Does x+y overflow?

```
int addOK(int x, int y)
```

8.4.24 Absolute value

```
int abs(int x)
```

8.4.25 Does x fit in a short?

```
int fitsShort(int x)
```

8.4.26 Does x -> y in propositional logic?

```
int implication(int x, int y)
```

8.5 Is x >= 0?

```
int isNonNegative(int x)
```

8.5.1 Rotate x to the left by n bits

```
int rotateLeft(int x, int n)
```

8.5.2 Implement '!' without using '!'

```
int logicalNeg(int x)
```

8.6 HOW TO STORE A DATE (DD MM YYYY) IN UNSIGNED FOUR BYTE INTEGER?

8.7 How to store a date (DD MM YYYY) in unsigned two byte short integer?

9. Makefile

9.1 What is the purpose of make files?

9.2 What is 'command line' represent in Makefile

9.3 What is 'dependency line' represent in Makefile

9.4 How does Make utility understand the dependency files changed?

9.5 In Makefile how to write dependency for a header (.h) file

10. Structures

10.1 Declare a structure with different data types. Define object for the same and initialize/assign values and print

10.2 Can we assign structure objects (of same structure) one to another?

10.3 Define many structures consist of different type of members.

- Print size of the each structure using sizeof operator
- Print Base and each member address.
- What is the expected size and actual size of structure? Why there is a difference?

```
i    c    c    c    i    sh   sh   sh   c    i    c    c    sh   c         c
i    c    i    i    c    i    sh   c    sh   c    c    c    c    sh        c
     i         c    i    c    i    i    i    c         c    c    c         sh
                    c
```

Note: i → int, c → char, sh →short int

11. Unions

11.1 Declare a union with different data types. Define object for the same and initialize/assign values and print

11.2 Define many union consist of different type of members like below.

- Print size of the each union using sizeof operator
- Print Base and each member address.
- What is the expected size and actual size of union? Why there is a difference?

```
i    c    c    c    i    sh   sh   sh   c    i    c    c    sh   c         c
i    c    i    i    c    i    sh   c    sh   c    c    c    c    sh        c
     i         c    i    c    i    i    i    c         c    c    c         sh
                     c
```

Note: i → int, c → char, sh →short int

12. Linked List

12.1 Single Linked List. For all implementations of below functions, assume Header of linked list(s) is available globally.

12.1.1 node_count() - function that counts the number nodes in a list.

```
int node_count(void)
```

12.1.2 count() - function that counts the number of times a given int occurs in a list.

```
int count(int value)
```

12.1.3 getNth() - function returns the data value stored in the node at that index position

```
int getNth(int index)
```

12.1.4 delete_list() - Function takes a list, deallocates all of its memory and sets its head pointer to NULL.

```
int delete_list(void)
```

12.1.5 pop() - Function removes the first node and returns the data of it

```
int pop(void)
```

12.1.6 InsertNth() - Function inserts new node in N'th position

```
int InsertNth(int value)
```

12.1.7 SortedInsert() - Function insert nodes in sorted order

```
int SortedInsert(int value)
```

12.1.8 InsertSort() - Function that sorts the given list in order

```
int InsertSort(void)
```

12.1.9 Append() - Function that takes two lists, 'a' and 'b', appends 'b' onto the end of 'a', and then sets 'b' to NULL.

```
int Append(void)
```

12.1.10 FrontBackSplit() - From given a list 's', split it into two sublists 'a' for the front half, and b for the back half.

```
int FrontBackSplit(void)
```

12.1.11 RemoveDuplicates() Function which takes a list sorted in increasing order and deletes any duplicate nodes from the list. Ideally, the list should only be traversed once

```
int RemoveDuplicates(void)
```

12.1.12 MoveNode() - Instead of creating a new node and pushes node 'p' from list 'a' to list 'b'

```
int MoveNode(Node *p)
```

12.1.13 AlternatingSplit() Function that takes one list 's' and divides up its nodes 'alternatively' to make two smaller lists 'a' and 'b'

```
int AlternatingSplit(void)
```

12.1.14 ShuffleMerge() Given two lists 'a' and 'b', merge their nodes together to make one list, taking nodes alternately between the two lists.

```
int ShuffleMerge(void)
```

12.1.15 SortedMerge() Function that takes two lists 'a' and 'b', each of which is sorted in increasing order, and merges the two together into one sorted list 'c'

```
int SortedMerge(void)
```

12.1.16 MergeSort() - Split the list into two smaller lists, recursively sort those lists, and finally merge the two sorted lists together into a single sorted list.

```
int MergeSort(void)
```

12.1.17 is_lists_are_intersecting(void) – Function finds the given two lists 'a' and 'b' are intersecting?

```
int is_lists_are_intersecting(void)
```

12.1.18 Reverse() - Reverse the list

```
int Reverse(void)
```

12.1.19 RecursiveReverse() - Do reversing the list using recursion

```
int RecursiveReverse(void)
```

12.1.20 List_Sum(void) - Adding two lists 'a' and 'b' and creating a new list 'c' with sum. Ex. if list 'a' nodes are {8, 9, 4, 7}, and 'b' nodes are {7, 8, 5} then, 'c' nodes should be { 9, 7, 3, 2}

```
int List_Sum(void)
```

12.2 Double Linked list

12.2.1 Write a function which takes any node address as a single argument and deletes that from the list. It should return 0 on success, -1 in case of failure

```
int delete_node(node *ptr);
```

II. Operating System - Unix Internals

1. File Management Assignments

1.1 Bhagavan

1.2 Open the same file in two different processes (un-related). What if the file pointer/index moved in one process, does it moves the file pointer/index in second process also?

1.3 Do the same above activity but, make the process related (Parent Child relation). Open the file and create a child process by using fork().

1.4 In a program close file descriptors 0 – STDIN, 1 – STDOUT, 2 – STDERR, by calling close() system call and then try to print string by using printf. Does it print anything onto screen?

1.5 Open multiple files simultaneously (ex. 5 files) and print descriptors. We might get descriptors as 3, 4, 5, 6 and 7. Now close the descriptor 4 and open a different file, what will be the new descriptor value?

1.6 Open multiple files simultaneously (ex. 5 files) and print descriptors. We might get descriptors as 3, 4, 5, 6 and 7. Now "dup" the descriptor 6 by using dup() system call. What will be the new descriptor value that returns by the dup()?

1.7 Open multiple files simultaneously (ex. 5 files) and print descriptors. We might get descriptors as 3, 4, 5, 6 and 7. Now close the descriptor 4 and dup the descriptor 6 by using dup() system call. What will be the new descriptor value that returns by the dup()?

1.8 Open a file in write mode, close the descriptor 1 and dup() the file descriptor of the text file. Now print some strings. Does it stores the strings into text file?

1.9 How many files can be opened by a single process simultaneously?

1.10 How many file can be opened by two processes simultaneously?

1.11 Is the any system's limit to open maximum number of files?

1.12 Print "/proc/pid/limits" file content.