2013

Wavedigitech

aleem

[C INTERVIEW QUESTIONS & ANSWERS]

Ver-1.0

Q 1. Difference between Arrays and Pointers?

Arrays:

a) Array is a collection of contiguous elements of same data type.

b) It is internally treated as constant pointer to that data type.

c) size of array = index * sizeof(data type);

d) Array does not check array bounds, the base address of array is the address of first element of array.

e) incrementing array and assigning address to array is illegal.

Pointers:

a) Pointer is a variable which holds the address of another variable of same type and has a property of De-referencing.

b) Any pointer has size 4 bytes(that is equal to size of integer)

c) The pointer version of program generate less instructions compared to array version.

d) Pointers are used in dynamic memory allocation (malloc, calloc, realloc)

e) Incrementing pointer, assigning address to it is legal.

---

2. Difference between malloc and calloc with syntax?

malloc and calloc are used for dynamic memory allocation of memory in C.

malloc:

a) malloc allocates a block of memory in heap segment, it has only one argument

b) The default values are garbage in malloc allocation

c) the malloc return a void pointer to the first location of allocated memory, return NULL if fails

d) malloc allocates memory as a single contiguous block.

Syntax:

char *ptr;

ptr = (char *)malloc(SIZE * sizeof(char));

(char *) to type casting void pointer to our datatype

SIZE is a macro , it tells size

sizeof() operator used for allocating respective data of size SIZE

calloc:

a) calloc allocates a multiple blocks of memory, it has two arguments.

b) The default values are Zero in case of calloc.

c) calloc internally calls malloc and does a memset to Zero, thereby initialize all elements with Zero.

d) calloc allocates memory which may or may not be contiguous

Syntax:

char *ptr;

ptr = (char *)calloc(no_of_blocks, SIZE * sizeof(char));

no_of_blocks = to specify how many no of blocks you want.

(char *) to type casting void pointer to our datatype

SIZE is a macro , it tell size sizeof() operator used for allocating respective data of size SIZE free() used to release the memory allocated by malloc and calloc.

Q3.Explain compilation process in detail.

Compilation process

Code written in editor
| .c or .h extension
Preprocessor
|-------------------> File inclusion
|-------------------> Comment removal
|--------------------> Macro expansion
|--------------------> Conditional compilation
|
| .i file generated
Compiler
|---->Analysis phase -|---> Laxical Analyser
| |---> Syntax Analyser
| |---> Symmantic Analyser
|
|----->Synthesis |--->Intermediate code
| generator
| |---> Code optimizer
| |---> Code generator
| .s File generated
Assembler
| Converts .s file into .o file
| The output is nearest to
| executable file but not executable.
|
Linker & Loader
| It links object modules and
| produce executable file
| (a.out in unix), .exe, .com
|
Executable file

Compilation process : There are four phases of compilation.

1. Preprocessor
2. Actual compiler
3. Assembler
4. Liker and loader

1.**Preprocessor**: In this phase all the # directive will be preprocessed.

<u>There are four phases of Preprocessor.</u>

5. File inclusion: All the #include file will be included into source code.
6. Comment removal: In this phase all the comments will be removed.
7. macro expansion: In this phase all the macro will be substituted where it is called.
8. Conditional compilation: In this phase the condition

2. **Compiler**:This will take preprocessed file (.i) and generated assembly file .s file.

There are two pahses.

9. Analysis phase
10. Synthesis phase

1.**Analysis Phase:** This phase will analyze the preprocessed file.

This phase has again three sub phases:

1.1>Lexical analyzer: This will tokenize our source code in to six tokens:

a. Keyword
b. Identifiers
c. Operator
d. Constant
e. Delimiter
f. String literals

1.2>**Syntax analyzer:** This Phase will check for syntax of the codes written.

11. Symmantic analyzer: This will check the grammer of the code.

2.**Synthesis phase:** In this phase assembly code will be generated.

This phase has again three sub phases:

12. Intermediate code generator: This will generate three addressing code i.e it has maximum of three variables.
13. Code optimizer: This will optimize our code
14. Code generator: This will generate code in Assembly language.

3. **ASSEMBLER**: This phase will take assembly code and generate binary object module, near to machine instruction but not executable.

4. **LINKER & LOADER**: This will link all the object module, libraries, command line argument etc. and produce executable file. In unix usually a.out file, in widows .exe file.

Thus executable file will be generated from our source code file.

These are the various phases of compilation process.

Q4. What is Little and big Endian,Explain with an example.

Little Endian: If the hardware is built so that the lowest, least significant byte of a multi-byte scalar is stored "first", at the lowest memory address, then the hardware is said to be "little-endian"; the "little" end of the integer gets stored first, and the next bytes get stored in higher (increasing) memory locations. Little-Endian byte order is "littlest end goes first (to the littlest address)".

Machines such as the Intel/AMD x86, Digital VAX, and Digital Alpha, handle scalars in Little-Endian form.

Big Endian : If the hardware is built so that the highest, most significant byte of a multi-byte scalar is stored "first", at the lowest memory address, then the hardware is said to be "big-endian"; the "big" end of the integer gets stored first, and the next bytes get stored in higher (increasing) memory locations. Big-Endian byte order is "biggest end goes first (to the lowest address)".

Machines such as IBM mainframes, the Motorola 680x0, Sun SPARC, PowerPC, and most RISC machines, handle scalars in Big-Endian form.

Four-byte Integer Example

Consider the four-byte integer 0x44332211. The "little" end byte, the lowest or least significant byte, is 0x11, and the "big" end byte, the highest or most significant byte, is 0x44. The two memory storage patterns for the four bytes are:

Four-Byte Integer: 0x44332211

Memory
address Big-Endianbyte value Little-Endian byte value
104 11 44
103 22 33
102 33 22
101 44 11

If we display the memory dump of the number 0x44332211 stored in memory at address 101 in Big-Endian order, we see something like this:

ADDRESS: ---------- MEMORY BYTES ----------
100: 00 44 33 22 11 00 00 00 00 00 ...

If we display the memory dump of the same number 0x44332211 stored in memory at address 101 in Little-Endian order, we see something like this:

ADDRESS: ---------- MEMORY BYTES ----------
100: 00 11 22 33 44 00 00 00 00 00 ...

Q5 : What is static memory allocation and dynamic memory allocation?

Static memory allocation:
-Static memory allocation refers to allocation of memory during compile time.
-It is used when we need to allocate fixed size.
-Memory will be allocated in stack and data segment.
-It has fast memory access take less execution time.
-It deleted automatically on exit from program.
e.g : int a;
int arr[20]; - allocate 20*4 = 80 bytes of memory to arr

Dynamic memory allocation :
-memory will be allocated during run time.
-Data structures can grow and shrink to fit changing data requirements.
-use malloc and calloc to allocate memory.
-access time is more.
-memory will be allocated in heap.
-we have to free memory before exiting from program.
-We can allocate (create) additional storage whenever we need them.
-We can de-allocate (free/delete) dynamic space whenever we are done with them.

Advantage: we can always have the exact amount of space required - no more, no less. For example, with references to connect them, we can use dynamic data structures to create a chain of data structures called a linked list.
e.g :
1.
char *str;
str = (char *)malloc(SIZE * sizeof(char));
2.
void * calloc (size_t num, size_t size);
char *str;
str = (char *)calloc(SIZE * sizeof(char))

Q6.What is Pointer Variable? How are Poniter Variables initialized?

Definition: A Variable which hold the address of another variable is called poniter variable and it has a property of derefrencing.

Features of Pointers:

1. powerful but difficult to master.
2. Sminulates by call by refrence.
3.Closely rlated to strings and arrays.

Initializing Pointers:

1. double *amount = &total;
2. int section[80]
int *student = section;
is equivalent to:
int section[80];
int *student = &section[0];
3. char *string = "abcd";
4. static char *weekdays[ ] ={
"Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"
};

Q7.What is sizeof() operator ? Explain how to implement?

sizeof is a compiler time operator. It is used to calculate the size of any data type, variable or constant, measured in the number of bytes required to represent.
It is very useful in case of dynamic memory allocation.
Implementation of sizeof using macro:

```
#define SIZEOF(x) SIZEOF1(x)
#define SIZEOF1(y) ( __typeof__(y) tmp; (((char *)(&tmp + 1)) - ((char *)(&tmp));)
```

where 'x' is a variable of any type or any data type or any constant.
Here ' __typeof__ ' is a function which returns the data type of the variable. So we are making a variable 'tmp' of type which has 'y'. Now we are type casting the variable to (char *) which gives the exact number of bytes allocated by that data type or variable.

Q 8. Operation on bit wise operator (all assignment examples)

Bit wise operator: Bit wise operator operates on one or more bit pattern or binary number. It is fast, directly action supported by processor.

There are some bit wise operator listed below:

Operator

Description
OR operator (|)
If any one of the bit is set then it will result set.
AND operator (&)
If both bits are set then it will result set.
TILDE operator (~)
This will Toggle the bit.
LEFT SHIFT operator (>>)
This will shift the bits in left direction.
RIGHT SHIFT operator (<<)
This will shift the bits in right direction.

Swapping of bits of a given number:

```
int bit_swap(int s, int d, int n){
return(((n >> d) & 1) == ((n >> s) & 1)) ? n : (n ^ ((1<<s) | (1 << d)))); }
```

Copy a bit from source bit to destination bit

```
int bit_cpy(int s, int d, int snum, int dnum){
return(((1 & (snum >> s)) == (1 & (dnum >> d))) ? \
dnum : (dnum^(1 <<d )));
}
```

Toggle odd bit and even bits of a number

```
int odd_bit_toggle(int num){
return (num ^= 0xAAAAAAAA); }
```

OR

```
int odd_bit_toggle(int num){
int mask = 2;
while (mask){
num ^= mask; mask << 2; }
return num;
}
```

```
int even_bit_toggle(int num){
return(num ^= 0x55555555);
}
```

OR

```
int even_bit_toggle(int num){
int mask = 1;
while (mask){
num ^= mask;
mask << 2;
}
return num;
}
```

Set a bit of a number

```
int bit_ts(int pos, int num){
return(num |= (1 << pos));
}
```

toupper() and tolower() using bitwise operator

```
int mytolower(int c){
if(c >= 'A' && c <= 'Z')
c |= (1 << 5);
return c;
}

int mytoupper(int c){
if(c >= 'a' && c <= 'z')
c |= (1 << 5);
return c;
}
```

Check a number is Odd or Even

```
int num_chk(int num){
if(num & 1)
printf("Number is odd");
else
printf("Number is even");
return(num & 1);
}
```

Reverse bits of a number

```
unsigned int reverse_bit(unsigned int n){
int s = 0;
int d = 31;
for(i = s; s < d; s++, d--)
n= (((n >> d) & 1) == ((n >> s) & 1)) ? n : (n ^ ((1<<s) | (1 << d))));
}
```

Addition with bit wise operator

```
int add(int x, int y){
while(b = ((((a ^= b) ^ b) & b) << 1) );
return a;
}

int sub(int x, int y){
while(b = ((~((a ^= b) ^ b) & b) << 1) );
return a;
}

int mul(int x, int y){
int res = 0;
while(y != 0){
if(y != 0)
res = add(res, x);
x <<1;
y >> 1;
}
return res;
}

int div(int x, int y){
int sign = 0;
int count = 0;
if(x < 0){
x = -x;
sign++;
}
if(y < 0){
y = -y;
```

```
sign++;
}
while(x > 0){
while(y = ((~((x ^= y) ^ y) & y) << 1));
count++;
}
if(sign == 0 || sign == 2)
return(count);
else
return(-count);
}
```

Rotate n-Bit of a number

```
int rotate(int dir, int nbit, int num){
unsigned unum = num;
if( dir == 1)
unum = (unum >> nbit) | (unum << (32 – nbit));
else
unum = (unum << nbit) | (unum >> (32 – nbit));
num = unum;
return (num);
}
```

Count clear and set bit in number

```
int count_bit_clear(int num){
int c;
for(c = 0; num; num &= num - 1 )
c++;
return c;
}

int count_bit_set(int num){
int c;
for(c = 0; num; num &= num - 1 )
c++;
c = sizeof(int) – c;
return c;
}
```

Q9. What is Call by value and Call by reference?Explain with an example.

Call by Value:

If data is passed by value, the data is copied from the variable used in for example main() to a variable used by the function. So if the data passed (that is stored in the function variable) is modified inside the function, the value is only changed in the variable used inside the function.

Call by value example:

```
#include <stdio.h>

void call_by_value(int x) {
printf("Inside call_by_value x = %d before adding 10.\n", x);
x += 10;
printf("Inside call_by_value x = %d after adding 10.\n", x);
}

int main() {
int a=10;

printf("a = %d before function call_by_value.\n", a);
call_by_value(a);
printf("a = %d after function call_by_value.\n", a);
return 0;
}
```

The output of this call by value code example :

```
a = 10 before function call_by_value.
Inside call_by_value x = 10 before adding 10.
Inside call_by_value x = 20 after adding 10.
a = 10 after function call_by_value.
```

Call by Reference:

If data is passed by reference, a pointer to the data is copied instead of the actual variable as is done in a call by value. Because a pointer is copied, if the value at that pointers address is changed in the function, the value is also changed in main().

Call by Reference example:

```
#include <stdio.h>

void call_by_reference(int *y) {
printf("Inside call_by_reference y = %d before adding 10.\n", *y);
```

```
(*y) += 10;
printf("Inside call_by_reference y = %d after adding 10.\n", *y);
}

int main() {
int b=10;

printf("b = %d before function call_by_reference.\n", b);
call_by_reference(&b);
printf("b = %d after function call_by_reference.\n", b);

return 0;
}
```

The output of this call by reference :

b = 10 before function call_by_reference.
Inside call_by_reference y = 10 before adding 10.
Inside call_by_reference y = 20 after adding 10.
b = 20 after function call_by_reference.

Q 10- what is function?advantage of function over macro.

Function :- A function is a block or set of instruction that perform operation on some given input to produce some output.it provides reusibility , modularity, portability to a program.
There are two types of function
1. Built in function : printf(),scanf()
2. User defined function

eg. :
```
#include <stdio.h>
#include <conio.h>

int add(int a, int b){
int c;
c = a + b;
return c;
}

int main(){
```

```
printf("\n Enter Any 2 Numbers : "); /*built in function*/
scanf("%d %d",&a,&b); /*built in function*/
add(a, b); /*user defined function*/
printf("\n Addition is : %d",c); /*built in function*/
return 0;
}
```

Advantage of function over macro :

1. In macro all the time it get a macro it substitute the code defined by macro thus it consume more memory whereas in case of function it just de-reference the function so it consume less memory

2. Macro has less execution time as it doesn't have to jump or return whereas function has to jump and return which cause more execution time.

3. function check for parameter type checking whereas there is no such concept in macro.

---

11. What are characteristics of array in C? Explain the initialization of 2 dimensional array?

Ans: Array characterstics:
1. Array holds elements of same data type .
2. Array elements are stored in subsequent memory locations.
3. Array index by default starts from 0 and ends with index no - 1.
4. Array name indicates the address base address of the array.
5.Two dimensional array can be stored row by row in continues memory.
6. In two dimensional array column no declaration is must.

Initialization of 2 dimensional array.
int a[no-row] [no-col];

a->array name.
no-row->number of row.
no-col ->number of columns.

You can initialize like
int a[][no-col]

You can skip no_row but not no_col because compiler gets confused how much space should be allocated to column of no_row.

---

12. Explain dynamic memory allocation w.r.t. Single and two dimensional arrays?

Ans. 1 Dimensional array :

Using a single pointer to a type can be used to allocate a memory of that type and can be used to access as a single dimensional array with a pointer pointing to that memory.

Ex :

```
int *p; /* p is a pointer to an integer object */
p = (int *) malloc(sizeof(int ) * (no of elements));
```

2. Two Dimensional array :

This can be done using either a double pointer to a type or a pointer to an array.

Using a Double pointer we need to allocate an array of pointers and then we need to allocate a single dimensional array for each pointer.

Ex :

```
int **ptr;
int I;
/* creating an array of pointers */
ptr = (int **) malloc(sizeof(int ) * (no of arrays));

for (i = 0; i < no of arrays; i++) {
(ptr + I) = (int *) malloc(sizeof(int ) * (no of ele));
}
```

Using a pointer to an array we can create using the following statement

Ex :

```
int (*ptr)[no of cols];
ptr = (int (*)[no of cols]) malloc(sizeof(*ptr) * (no of rows));
```

---

13. Explain the concept of linking n loading?

Linking : In the compilation process once after the compiling has been successfully completed we get all the relocatable object modules which must be linked together to get a final executable code. This linking can be either static or dynamic.

In static linking all the static libraries and relocatable object modules will be present in the executable file.

In the case dynamic linking all the shared libraries will be loaded into the our executable once it gets loaded into the memory. The advantage of dynamic linking is that the libraries that we add will not be part of the executable file, so that file size gets reduced.

Loading :Loading is bringing the executable code into the main memory for execution. The header record is checked to verify that there correct program is been presented for loading. As each text record is read, the object code it contains is moved to the indicated address in memory. When the end record in encountered, the loader jumps to the specified address to begin the execution of the loaded program.

---

14. What are storage class and what are advantages and disadvantages of external storage class?

Ans: Storage class : Is an attribute which changes the behavior of variable,It also decides the scope and lifetime of the variable.

There are two storage class:

1. Auto
2.Static

Auto:

1. Automatic variables are local to a block and discarded on exit of the block.
2. Declarations within a block create automatic variable if no storage class specification is mentioned.
3. Variables declared register are automatic and are if possible stored in fast registers of the machine.

Static:

1. Static objects may be local to block or external to all blocks, but in either case retain their values across exit from and reentry to functions and blocks.

2. static objects are declared with keyword static.
3. Any local or global variable can be made static depending upon what the logic expects out of that variable.

4. The external static declaration is most often used for variables,but it can be used to function as well. if function is declared static, however,its name is invisible outside of that file in which it is declared.

Advantages of external storage class:

1. storage class retains the latest value.
2.The value is globally available.

Disadvantages of external storage class:

1. The storage for external variable exists even when the variable is not needed.
2. The side effect may produce surprising output

---

15. Scope and lifetime, with an example?.

Scope : It is the boundary of a variable(identifier) within which it is accessible.
Ex:

```
int main(){
int a = 10;
int i;
for(i = 0; i < 9; i++) {
for (j = 0; j < 10; j++) {
if (a[a[i] > a[j]) {
int temp ; /*Scope of temp is only within if block */
temp = a[i];
a[i] = a[j];
a[j] = temp;
}
} /*temp is inaccessible here but a can be accessed*/
}
printf("%d",temp); /*Error: Undeined symbol*/
return 0;
}
```

In this example a and j are within whole main function scope . But temp is within if block scope.

Lifetime: It is the time within which a variable(identifier) exists in memory.

Ex:

```
int main(){
fun(); /*a exists between function calls eventhough the scope of a is within fun() only*/
fun();
return 0;
}
void fun(){
static int a = 5;
printf("a = %d",a++);
}
```

In this example a is static and its lifetime is within the program but in accessible in any functions other than fun().

First function call prints 5 and 2nd call prints 6 and the value of a finally is 7.

16. Differentiate between internal static and external static variable, with an example?

Ans. A static variable declared inside a block or a function is called as an internal static variable. It will not have any linkage. Scope of such variables is with in the function(if it is declared in the function) or block(if it is declared inside a block) but, it persists its storage.

A static variable declared inside globally is called as an external static variable. It will have internal linkage in that file. Scope of such variables is with in the file and also persists its storage.

Ex :

```
static int a = 8; /* external static
scope : file
storage : till program exits
have internal linkage */
int main(){
static int c = 9; /* internal static
scope : function
storage : till program exits
will not have any linkage */ {
static int c = 4; /* internal static
scope : block
storage : till the program exits
will not have any linkage */
}
}
```

---

17. difference between arrays and strings

Ex:

```
int main(){
int a[4] = {1,2,3,4}; /*Size is specified as 4*/
char b[5] = {'a', 'b', 'c','d','\0'}; /*Null character is explicitly specified*/
char s[5] = "abcd"; /*Size is one more than the actual elements as null character Is
/*automatically appended to end*/
}
```

In case of array of characters we need to explicitly specify the null character at the end to make it as string.

As in the example array can take any datatype (int, char in this case).

String is stored only in the form of ascii values. Here ascii values of a, b, c, d and null character are stored in s. But array can store integer values float and so on.

18. Is using exit() the same as using return? If yes or no give the appropriate reason?

Ans. No, It is not same. Because exit is a function is used to exit from the program but return is a statement used to return the control from the called function to the calling function. We can also call an exit handler function while exiting. This needs the registration of that handler and this can be done using atexit() function.

Only in our program's main function using exit is same as the return this is because after returning from the main function exit function is called and also exit handler functions if any registered.

---

19. Explain C memory layout with a diagram?

Ans. C memory layout and explanation :

1. **Text Segment** : Contains The machine instruction that the CPU executes. Usually text segment become sharable so that only a single copy needs to be in memory for frequently executed the program Ex : Text editor, Compiler. Text segment is read only memory

2. **Initialized Data Segment** : Simply called data segment contains variable that are initialized in the program

3. **Uninitialized Data segment** : Called BSS (Block Started by symbol)

4. **Stack** : where automatic variables are stored, along with information that is saved each time a function is called. Each time a function is called, the address of where to return to and certain information about the caller's environment, such as some of the machine registers, are saved on the stack. The newly called function then allocates room on the stack for its automatic and temporary variables. This is how recursive functions in C can work. Each time a recursive function calls itself, a new stack frame is used, so one set of variables doesn't interfere with the variables from another instance of the function.

5. **Heap** : where dynamic memory allocation usually takes place. Historically, the heap has been located between the uninitialized data and the stack.

---

20. how recursive function works, keeping in mind about frame n stack pointer?

Ans : Recursive function works in this way :-

For each call of a subroutine (function) a Stack frame is created on the stack, Stack frame is an area on the stack where return address (address where control will be return after the completion of the function work), the argument of the function, and function local variable store. Access of the local variable is done through the frame pointer. Each frame have a frame pointer.

---

21. Explain the concept of function pointer with an example(call back function).

Ans. **Function pointer:** It is a pointer variable , which point to the address of a function.

Example : int (*fptr)(int); /* FUNCTION POINTER DECLARATION */

```
int fun () { /* FUNCTION DEFINITION */
return 20;
}
```

fptr = fun; /* ASSIGNING ADDRESS OF FUNCTION FUN */

fptr is a pointer to function fun ;

we can call this function by using any of these methods.
1> fun();
2> (*fptr)();
3> (fptr)();

Call back function: A callback is a reference to executable code, or a piece of executable code, that is passed as an argument to other code.
Consider this case where we have passed address of a function as argument .

IN FILE 2:

```
/* Function pointer fptr stores the address of the function get_value*/
void fun(int *element, int (*fptr) ){
*element = fptr();
}
```

IN FILE 1:

```
/* Call the function get_value*/
int get_value(int){
return 1;
}

int main(void){
int element;
fun(&element, get_value);
}
```

---

22. Explain the difference between the structure and union?

Ans. STRUCTURE :
It is user defined datatype which holds heterogeneous datatype and having keyword struct.

Here memory is allocated for all variables.
Each member have their own memory space.
All member of structure will be initialized .

Example :

```
struct tag {
char c;
int I;
```

```
float f;
};
int main(){
struct tag s_var = { 'a' , 1 , 1.4};
printf("%u " , &s_var);
printf("%u", &s_var.c);
printf("%u",&s_var.I);
printf("%u",&s_var.f);
}
```

UNION :It is user defined datatype which holds heterogeneous datatype and having keyword union.
Memory allocated is equal to maximum memory required by member of union.
Memory is shared by all the members.

Only first member of union will be initialized .

```
union tag {
char c;
int I;
float f;
};
int main(){
union tag s_var = { 'a'};
printf("%u " , &s_var);
printf("%u", &s_var.c);
printf("%u",&s_var.I);
printf("%u",&s_var.f);
}
```

---

23. How can I convert number to string.

```
#include < stdio.h>
#include<stdlib.h>

char *my_itoa(int num){
int i = 0; /* FOR ITERATION */
int div = 1; /* TO STORE THE DIVIDEND */
int num_copy = num;
int digit = 0; /* TO STORE THE EACH DIGIT */
```

```
int x = 0;
char *str_num = NULL; /* POINT TO THE STRING FOR STRING */

str_num = malloc(1);
if( str_num == NULL) {
printf("MEMORY CAN NOT BE CREATED\n");
return NULL;
}
if( num < 0) { /* IF IT IS A NEGATIVE NUMBER */
num = num * -1;
*(str_num + 0) = '-';
i = i + 1;

}

while( num_copy / 10 ) { /* TO KNOW THE LENGTH OF A NUMBER */
div = div * 10;
num_copy = num_copy / 10;
}
while(div != 0) {
x = num / div ;
digit = x % 10; /* ACCESS EACH BIT FROM LEFT TO RIGHT */
if((str_num = realloc(str_num, i + 1))== NULL) {
printf("MEMORY CAN NOT BE CREATED\n");
return NULL;
}
*(str_num + i ) = digit + 48 ; /* STORE EACH DIGIT IN TO STRING */
i = i + 1;
div = div / 10;
}

if((str_num = realloc(str_num, i ))== NULL) {
printf("MEMORY CAN NOT BE CREATED\n");
return NULL;
}
*(str_num + i ) = '\0';
return str_num;
}
```

Q.24. What is the benefit of using an enum rather than a #defined constant ?

Explain with an example .

1> Enumerated constants are generated by the compiler automatically , but
macros must be manually assigned values by the programmer.
2> All #defined constants are global while enum obeys the scope rules .
3> Readability enhances by using enum than using macros .
4> Macro substituted at compile time(pre processing stage) while enum is processed at run time.
Example :

```
#define RED 1
#define BLUE 2

enum colour {green,red,blue};
```

Here green is assigned default value of 0, red a value of 1, blue a value of 2;

Q. 25. What is preprocessor ? What will the preprocessor do for the program?

Ans : The preprocessor is a stage at which it modifies the program(extends the source code) according to the preprocessor directives in your source code.
The preprocessor provides different facilities :
1. file inclusion :
#include replace the current line with the contents
2. Strip comments :It removes all the comments from the source code.
3. Macro Expansion :All the macro invocations are replaced withthe text defined with the #define directives.
4. Conditional Compilation:Preprocessing directives , you can include or exclude , parts of the the program according to the condition.

```
#ifdef MAX
printf("hello");
#endif
```

5. # pragma directive

Q. 26 . Explain Stringizing operator with respect to preprocessor?

Ans : The stringizing operator is used to convert prameters passed to macro into strings .
The symbol is "#" .
The stringizing operator used with #define preprocessor directives taking a parameter .

```
#define MAKE_STRING (x) #x
```

1>Will convert whatever is passed as parameter "x" into string
2>Leading and trailing white spaces are deleted.

Example :

#define MAKE_STRING(x)
#define COMMA ,
MAKE_STRING (shankar pankaj);
0/p : “shankar pankaj”
MAKE_STRING(shankar pankaj);
o/p : “shankar pankaj”
MAKE_STRING(shankar COMMA pankaj);
o/p : “shankar COMMA pankaj”

---

27. What is the benefit of using const for declaring constants? Explain with example.

Ans.As the name implies the value of the variable can not be changed using its own name.
During the declaration of the variable we have to initialize it which can not be changed.

Example : int flag;
int *const ptr = &flag;
Consider a situation where we have to take decision basing on the value of this flag to whom the ptr points.
so ptr should point always to flag & no one else so that the task can pe properly done.

---

28. Can a variable be both const and volatile ? When should the volatile modifier be used.

Ans.Yes a variable can be const & volatile.
consider the previous case where the flag variable is checked all the time so that basing on this we have to take the decision.

int volatile * const ptr = &flag;

As ptr is const so always it point to the flag variable.
*ptr which is equal to the variable flag is read all the time to know thw status.
If we will not take the value of flag volatile then the compiler will optimize & will not read the status of flag all the time which leads to problem.
so using volatile to the value pointed by ptr we will tell the compiler that dont optimize the variable & it should be written & read all the time.

Q.29. What is ADT in data structure ?

Ans : An abstract datatype is the way we look at a data structure focusing on what it does , and ignoring how it does the job.

Example :

stack & queue : This can be implemented using an array or a linked list according to our requirements .
* As an implementor we should know how tho use it rather considering the design principle which is abstracted.

30. Difference between pointer to array and array of pointers with syntax?

Ans. POINTER TO ARRAY:
As the name implies it is a single pointer variable which holds the address of the array.
Syntax:
<type> (*ptr_name) [SIZE];
ptr_name is a pointer to 1-D array of SIZE elements to type.

As it is a pointer its size will be the size of the word length of the machine(4 byte in 32 bit machine).

int (*ptr)[20]: It is a pointer to an 1 D integer array of 20 elements.
ptr++- It goes to next 1 D array.
ARRAY OF POINTERS:
It is a block of memory location which can store address of different variables.
Syntax: <type> *<array_name> [SIZE]
array_name of size element long pointers of type.
The total size of the block will be SIZE * pointer size.

int *ptr[20] - Array of 20 pointers which holds the address of 20 integers.
As the name is constant pointer so we can't increment it.

Ques. 31 : Give real time technical example for Structure and Unions ?

Ans. : Structure: Compiler and Lexical Analyzer ,widely used with data structures
Union: Unions are useful for application involving multiple members where values are not assigned to all the members at any one time.
Semaphore:mechanism for ensuring mutual exclusion among concurrent process accessing a shared resources.

Ques. 32 : What is bit-fields, explain with an example ?

Ans. : A bit field is a set of adjacent bits with in a single implementation defined storage unit call a word .
In order to set individual flags of device drives where it needed to communicate with operating system at low level .
int data;
struct {
unsigned int ready: 1;
unsigned int enable: 1;

unsigned int received_Byte: 8;
} receiver;
data = receiver.received_Byte;
receiver.ready = 0;
receiver.enable = 1;

9 2 1 0

received_byte
Enable
ready

Signal Set
1 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 1 1 1 1 1 0 0 0 0 0 0 1 1 1 0 1
each individual bit represents offset of different signal.

Ques. 33: How to convert float to binary ?

Ans. : Floating point representation:
According to IEEE-754 the floating point representation is done as
follows:
Sign exponent(bias) mantissa-23bits
1-bit 8bit
exponent are first added by a fixed bias ,the bias is chosen to be large enough
to convert every integer in range to positive integer(to store in binary).
Example : -5.5834 * 10 (10)(pow)
= -1.625 * 2 (35)(pow)
= -1 (1)(pow) * 1.625 * 2(162 - 127)(pow)

1 1 0 1 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

Ques. 34 : When do you use register storage class specifier, explain with an example?

Register Storage Class : A program that uses register variables executes faster as compared to similar program without register variable because access to the register in CPU is faster than main memory.
When we go for Register storage class : Simply to make the execution fast
and to utilize the microprocessor's register memory we use register
variable. In some situations like :
1. for frequently used variable ,such as loop control variable.
2. If we need a local variable and nowhere in program need of that's address, on that situation we specifies that variable as register.
3. If you want to specify the storage class of formal argument then you can only use register storage class.

Limitations of that variable : If we specify as register to variable then that
have some limitations like :
1. we can't declare that objects globally,
2. we can't apply & operator on that object.

Ques. 35 : What is the difference between switch and if-else-if ladder. How it is internally working ?

Ans : if-else-if ladder and switch both are the type of selection statement ,so choose one of several control flows.
Differences in both :
1. if-else-if ladder may use arithmetic and pointer both type of expressions but in switch controlling expression must be of type arithmetic.
2. If-else-if ladder uses linear flow of control and control alters from where it expression result nonzero value after executing its sub statements but in switch case constant are the labels so control directly goes to that address.
3. In switch case controlling expression must be promoted to integral value but in if-else-if ladder it may be to any type.
4. Switch case constant uses 'break' jump statement to terminate the flow otherwise it executes statement until break statement or the end of switch's statements. But in if-else-if ladder no need of any jump statement if any one of sub statement executes then it comes out of ladder.
5. In if-else-if we use 'else' to resolve ambiguity but in switch we use default label statements.

Ques. 36 : Explain the difference between break and continue, with an example ?.:

break and continue both are the type of jumped statement. These both statement does not expect any expression with it, means no need to give any expression with it.
Differences in break and continue : Both are different in the ways
1. Break statement may appear in an iteration statement or in switch statement but continue statement may appear in only iteration statement.
2. Break statement terminate the execution of the smallest enclosing such statement(Iteration or switch statements), but it skip the sub statements below itself (of iteration statements) and continues the iteration.
Example :
In Iteration statements:

```
do {
statement1;
if (condition_1) {
statement2;
break;
} else {
statement3;
continue;
```

```
}
} while (condition_2);
```

explanation : in first iteration it will executes statement 1 then check condition_1 if it results some non zero value then- execute statement 2 and comes out of loop if condition_1 false then- it goes for else part run statement 3 and the check condition_2 if it is resulting nonzero value then loop will run again.

```
switch (expression) {
case 'constant_1' :
statement_1;
break;
case 'constant_2' :
statement_2;
break;
default:
statement_3;
}
```

Ques. 37 : Explain the operators in c with precedence ?

Ans. : Operators :

OPERATOR

Associativity

() [] -> .

L to R

! ~ ++ -- + - *(type) sizeof

R to L

* / %

L to R

+ -

L to R

<< >>

L to R

< <= > >=

L to R

== !=

L to R

&

L to R

^

L to R

|

L to R
&&
L to R
||
L to R
? :
R to L
= += -= *= /= %= &= ^= |= <<= >>=
R to L
,
L to R

. Assignment operator : =
. Arithmetic operator : + - / % *
. Increment and Decrement operator : ++ —
. Relational operator : > >= < <= ==
. Logical operator : && ||
. Bitwise operator : & | ^ ~
. ? : operator
. & and *pointer operator
. comma operator
. Dot(.) and Arrow(->) operator

operators can be :
. Unary operator - require one operand (++ — + -)
. Binary operator - require two operand (/ * +)
. Ternary operator - require three operand (? :)

Ques. 38 : what is self referential structure, give an example ?

Ans. : A structure which has member as a pointer to its structure variable.
Example :
struct tag {
int a;
struct tag *ptr;
};

Ques. 39 : What is the difference between char const *p and const char *p ?

Ans. : char const *ptr;
ptr is a pointer that points to a constant variable.

using pointer ,value of variable can't be changed
. const char *ptr;
same as - char const *ptr
. char *const ptr;
ptr is a constant pointer.
value of the variable using pointer can be changed.
Ques. 40 : What is typedef. How it is Different from #define ?
Ans. :#define is replacement which is handled during preprocessing stage.
Typedef is used to define a new name which is ignored during preprocessing.
Typedef is handled by the C compiler itself and is a an actual definition of a new type.It is used as a specifier because of syntax similarity with other data type.
Example:
#define flaot flaot*
typedef flaot * float;

in main function they declared like

float p1,p2,p3; //for #define
float ptr1,pt2,ptr3; //for typedef
then after replacement in macro and renaming by typedef
in case of #define:
float *p1,p2,p3;
in case of typedef:
float *ptr1;
float *ptr2;
float *ptr3;

GROUP – 5

---

(41)What is linking and loading. Explain in details.

Linking: Linking is the process of collecting and combining various pieces of code and data into a single file that can be loaded (copied) into memory and executed.

Linking can be performed at compile time, when the source code is translated into machine code, at load time, when the program is loaded into memory and executed by the loader, and even at run time, by application programs.

On early computer systems, linking was performed manually. On modern systems, linking is performed automatically by programs called linkers.

To build the executable, the linker must perform two main task:-

Symbol resolution

Object files define and reference symbols. The purpose of symbol resolution is to associate each symbol reference with exactly one symbol definition.

Relocation Compilers and assemblers generate code and data sections that start at address 0. The linker relocates these sections by associating a memory location with each symbol definition, and then modifying all of the references to those symbols so that they point to this memory location.

Loading Coping the loadable image into memory, connecting it with any other program already loaded and updating addresses as needed is called loading .

In Unix, the loader is the handler for the system call execve().
The Unix loader's tasks include:
1. validation (permissions, memory requirements etc.);
2. copying the program image from the disk into main memory;
3. copying the command-line arguments on the stack;
4. initializing registers (e.g., the stack pointer);
5. jumping to the program entry point (_start).

---

(42)Explain the importance of loops, with different scenarios.

Loops are very important in any programming language, they allow us to reduce the number of lines in a code, making our code more readable and efficient.
One of the main uses of loops in programs is to carry out repetitive tasks.
Many problem exist which cannot be solved without the looping construct.

---

(43)Explain the different variants of printf and scanf with prototype and examples.

The functions in the printf() family produce output according to a format, upon successful return, these functions return the number of characters printed.

printf, fprintf, sprintf, snprintf, asprintf, dprintf

printf()
This functions write output to stdout
int printf(const char * format, ...);

fprintf()
This function write output to the given output stream
int fprintf(FILE * stream, const char * format, ...);

sprintf()
write to the character string str
int sprintf(char * str, const char * format, ...);

snprintf()
write to the character string str
int snprintf(char * str, size_t size, const char * format, ...);

asprintf()
dynamically allocate a new string with malloc.
int asprintf(char **ret, const char *format, ...);

dprintf()
This Function write output to the given file descriptor.
int dprintf(int fd, const char * format, ...);

The scanf() family of functions scans input according to format, these functions return the number of input items successfully matched and assigned.
scanf, fscanf, sscanf

---

scanf()
This function reads input from the standard input stream stdin
int scanf(const char *format, ...);

fscanf()
This function reads input from the stream pointer stream
int fscanf(FILE *stream, const char *format, ...);

sscanf()
reads its input from the character string pointed to by s
int sscanf(const char *s, const char *format, ...);

---

(44)What is structure padding? Explain #pragma and #error in details.

To align the data, it may be necessary to insert some meaningless bytes between the end of the last member and the start of the next, which is called structure padding.

Structure padding occurs because the members of a structure must appear at the correct byte boundary. This is more efficient at hardware level (needs less cpu cycle to read or write variables) and in some platforms this is mandatory. To achieve this, the compiler puts in padding so that the structure members appear in the correct location

#pragma:-
The #pragma preprocessor directive allows each compiler to implement compiler-specific features that can be turned on and off with the #pragma statement.

Preprocessor Error Directive (#error):-
The #error macro allows you to make compilation fail and issue a statement that will appear in the list of compilation errors. It is most useful when combined with #if/#elif/#else to fail compilation if some condition is not true.

For example:
```
#ifndef __unix__ // __unix__ is typically defined when targetting Unix
#error "Only Unix is supported"
#endif
```

---

(45)What is the Harm in using goto.

1. lead to a very bad/unreadable/unstructrured/complicated code
2. increase the complexity of debugging and analyzing the code

---

(46)What is inline function? How it is different from macros.

Inline function is a normal function preceded with a keyword inline. When a call is made to the inline function, the actual code is substituted or expanded in the function call statement in the program.

1) Inline functions are parsed by the compiler, whereas macros are expanded by the C++ preprocessor.
2) Macros does not return value, but inline function always return.
3) inline performed strict type checking but macros does not perform type cheking.
4) Usage of macros is always replaced with actual code (i.e. they are always in lined) While its up to the compiler to decide whether to replace the inline function call to function code or keep it as function call.
5) macros dont make the part of Code Segment, but inline functions are a part of code segment.

---

(47)How variable argument lists works in c? Explain with an example.

To declare a function that accepts any number of values (arguments), in place of the last argument should place an ellipsis (which looks like …),
so for example in function – int addThemAll (int numargs,...) would tell the compiler the function should accept however many arguments that the programmer uses.

The macros used from stdarg.h to extract the values stored in the variable argument list

6. va_arg which initializes the list.
7. va_arg which returns the next argument in the list.
3. va_end which cleans up the variable argument list.
4. va_list is a variable capable of storing a variable-length argument

EXAMPLE:

```
#include <stdio.h>
#include <stdarg.h>

int addThemAll(int numargs, …) {
/* this function can accept a variable number of arguments */
/* creats a pointer that will be used to point to the first element of the variable argument list */

va_list listpointer;
int arg;
/* make listpointer point to the first argument in the list */
va_start(listpointer, numargs);

int sum = 0;
for (int i= 0; i< numargs; i++) {

/* get an argument from the va_list */
arg = va_arg(listpointer, int);
printf("the %dth arg is %d\n", I , arg);
sum += arg;
}
/* cleans up the argument list by calling va_end(); */

va_end(listpointer);
printf("the total sum is %d\n", sum);
return sum;
}

int main(){
printf("calling 'addThemAll(3, 104, 28, 46)';....\n");
addThemAll(3, 104, 28, 46);
printf("calling 'addThemAll(8, 1, 2, 3, 4, 5, 6, 7, 8)';....\n");
addThemAll(8, 1, 2, 3, 4, 5, 6, 7, 8);
}
```

(48)Explain comma operator with example.

In c, the comma operator (represented by the token ,) is a binary operator that evaluates its first operand and discards the result, and then evaluates the second operand and returns this value.

The comma operator separates expressions in a way analogous to how the semicolon terminates statements, and sequences of expressions are enclosed in parenthesis analogous to how sequences of statements are enclosed in braces: (a, b, c) is a sequence of expressions separated by commas, which evaluates to the last expression c.

The comma operator has the lowest precedence of any c operator. In a combination of commas and semicolons , semicolons have lower precedence than commas, as semicolons separates statements but commas occur with in statements.

Examples:
// comma acts separator in this line not as an operator
5. int a = 1, b = 2, c = 3, i ;

// stores b in to i.
6. i = (a, b);

// stores a into i
7. i = a, b;

// increases a by 2, then stores a + b = 3 + 2 into i.
8. i = (a += 2, a + b);

// stores a into i
9. I = a, b, c;

---

49. What is packing and unpacking in C?

Packing :
1. Packing is a method for which leads to override the compiler optimization.
2. It lead to natural boundary alignment of one or more of the data
elements contained in the structure.
3. A "packed" structure will almost always have a smaller memory than its unpacked brother.
4.This way you can access data within the packet directly by simply pointing the struct you want to the first byte of the packet. Ex:(TCP/IP packet form) .

How can we do packing:
#pragma pack (2^n)

Unpacking :
1.Its a complement of packing
2.Alignment is done according to word length of the longest data type.
3.Unpacking do the compiler optimization.

(50)Explain the assert() macro in c, with an example program

assert() macro defined in header file assert.h . When executed, if the expression is false (that is, compares equal to 0), assert() writes information about the call that failed and then calls abort(). The information it writes to stderr includes:
the source filename (the predefined macro __FILE__)
the source line number (the predefined macro __LINE__)
the source function (the predefined identifier __func__)
the text of expression that evaluated to 0

example:-

```
#include <stdio.h>
#include <assert.h>

int test_assert ( int x ){
assert( x <= 4 );
return x;
}

int main ( void ) {
int i;
for (i=0; i<=9; i++){
test_assert( i );
printf("i = %i\n", i);
}

return 0;
}
```

Output:-
i = 0 i = 1 i = 2 i = 3 i = 4
assert: assert.c:6: test_assert: Assertion `x <= 4' failed. Aborted

C – Questions and Answers

Table of Contents

1 String Functions

Function Prototype Description

1.strcpy char *strcpy (char *dest, char *src);
Copy src string into dest string.
2.strncpy char *strncpy(char *string1, char *string2, int n);
Copy first n characters of string2 to stringl .
3.strcmp int strcmp(char *string1, char *string2);
Compare string1 and string2 to determine alphabetic order.
If both strings are same returns '0' else returns difference of strings
4.strncmp int strncmp(char *string1, char *string2, int n);
Compare first n characters of two strings.
5.strlen int strlen(char *string); Determine the length of a string.
6.strcat char *strcat(char *dest, const char *src);
Concatenate string src to the string dest.
7.strncat char *strncat(char *dest, const char *src, int n);
Concatenate n chracters from string src to the string dest.
8.strchr char *strchr(char *string, int c); Find first occurrence of character c in string.
9.strrchr char *strrchr(char *string, int c); Find last occurrence of character c in string.
10.strstr char *strstr(char *string2, char string*1);
Find first occurrence of string string1 in string2.
11.strtok char *strtok(char *s, const char *delim) ;
Parse the string s into tokens using delim as delimiter.

1.1 Give an example for "strtok" ?

```
#include <stdio.h>
#include <string.h>
int main () {
	char str[] ="This, a sample string.";
	char * pch;
	printf ("Splitting string \"%s\" into tokens:\n",str);
	pch = strtok (str," ,.");
	while (pch != NULL) {
		printf ("%s\n",pch);
		pch = strtok (NULL, " ,.");
	}
	return 0;
}
```

output: Splitting
string "- This, a sample string." into tokens:
This
a
sample
string

---

2 Pointers and Arrays

2.1 Find out what can be error in below code snippet?

```
#include<stdio.h>
void f1(int a[]) {
	int *p=0;
	int i=0;
	while(i++ > 3)
		p=&a[i]; //error
	*p=0;
}

void main(){
	int a[10];
	f1(a);
}
```

ANSWER: Segmentation fault.
P is not assigned with any address as while loop does not execute (since i++ is not greater than 3). thereby p is NULL , *p will result in error. The same program executes fine if while executes (change condition in loop as while(i++ < 3))

2.2 What does the fallowing declaration mean? **(char* (*) ()) (*p[N])();**

ANSWER: Declaration of 'array of N pointers to functions returning pointers to functions returning pointers to characters'

2.3 What is the output of the below code

```
#include<stdio.h>
int main(){
	int a = sizeof(signed) + sizeof(unsigned);
	int b = sizeof(const) + sizeof(volatile);
	printf("%d\n", a + + + b);
	return 0;
}
```

ANSWER: 16
Default data type of signed, usigned, const and volatile is int.therefore,
a++ + b= 8 + 8 =16.

2.4 Q28. What is the output of the below code?

```
#include<stdio.h>
int main(){
	static char *s[] = {"ice","green","cone","please"};
	static char **ptr[] = {s+3,s+2,s+1,s};
	char ***p = ptr;
	printf("%s\n",**++p);
```

```
    printf("%s\n",**++ p);
    return 0;
}
```

ANSWER:cone
ice.
Assume the starting addr of ice, green, cone, please
(1000) (1004) (1010) (1015) > memory
Address and in **ptr[]
1015, 1010, 1004, 1000
(2000) (2004) (2008) (2012) >
memory address and in ***p[]
2000 (8888) > memory address
In **++p statement, ++p goes to work first and increments p to 2004,then the
*++p fetches the contents of 2004 i.e 1010 and **++p fetches the contents in
1010 i.e "cone" In **++
p statement the ++p operator gives the value of p 2008, *++p gives the contents of addr 2008 i.e 1004
*++ decrements the value 1004 to 1000 *++ gives the contents located at 1000 i.e "ice"

2.5 Q30. What is the output of the below code?

```
void main{
    char arr = "Smartplay";
    char ptr = "Smartplay";
    ptr++;
    arr++;
    printf("%s\n",*ptr);
    printf("%s\n",*arr);
}
```

ANSWER: arr++ will give compilation error since we can't increment base address of an array.

2.6 Q40. What is the output of the below code?

```
#include<stdio.h>
void main {
    static main;
    int x;
    x=call(main);
    printf("%d",x);
}
int call(int address){
    return(address++);
}
```

ANSWER: 1
Main is a normal variable. It is declared as a static, so value of it is 0,
and it gets incremented when 'call' function is called. x is 1

2.7 Q42. What is the output of the below code?

int a = 257;
char *ptr;
int *iptr;
iptr = ptr = &a;
printf("%d %d\n", *(char *)ptr, *(char *)(ptr+1)); // 1,1
printf("%d %d\n", *(char *)iptr, *(char *)(iptr+1)); // 1,0
ANSWER:
As 257 is stored in memory ...00000100000001 with respective to base '2'

2.8 Q45. Find the bug in the below code?
int const *a; /* pointer to a constant integer type */
a = NULL;
*a = NULL;
ANSWER:
*a = NULL is not valid as *a is constant
int *const b; /* constant pointer to an integer type */
b = NULL;
*b = NULL;
ANSWER:
b = NULL is not valid, pointed value is constant
int * const *c; /* pointer to a constant pointer of integer type*/
c = &b;
*c = NULL;
**c = NULL;
ANSWER:
*c = NULL is not valid as *c is constant

2.9 What is the output of the below program?
CODE:
```
#include<stdio.h>
void hello() { printf("hello\n"); }
int main(void) { (*****hello)(); }
```

ANSWER: helllo.
This is just a quirk of C. There's no other reason but the C standard just says that dereferencing or taking the address of a function just evaluates to a pointer to that function, and dereferencing a function pointer just evaluates back to the function pointer.

2.10 Is it possible to declare only one function pointer to the below mentioned functions:
*int float_func(float *array, int a);*
*int double_func(double *array, int b);*
ANSWER:Yes.
int (* ptr_func)(void * , int b);

2.11 What is the output of the below program?

CODE:

```
#include<stdio.h>

int add(int x,int y){
	return x+y;
}
int main(void) {
	int (*ptr_func)(int , int)= add;
	printf("%d,", sizeof(ptr_func));
	printf("%d\n", sizeof(*ptr_func));
}
```

ANSWER:

Size of function pointer is 4 bytes. Output of the program will be 4, 1.ptr_func is a pointer so the size will be 4 bytes.(*ptr_func) is a function. As the C language standard requires a diagnostic when using the sizeof operator with a function name since it's a constraint violation for the sizeof operator.

However, as an extension to the C language, GCC allows arithmetic on void pointers and function pointers, which is done by treating the size of a void or a function as 1. As a consequence, the sizeof operator will evaluate to 1 for void or a function with GCC.

2.12 Illustrate different ways of calling a function using function pointers with a code?

ANSWER:

```
int main(){
	char *s = "hello";
	void (* ptr1)(char *);
	void (* ptr2)(char *);
	ptr1= myprint();
	ptr2= &myprint();
	ptr1(s); //Implicit calling
	(*ptr1)(s); //explicit calling
	ptr2(s); //Implicit calling
	(*ptr2)(s); //explicit calling
}
```

2.13 Q52. WAP to find the number of occurrences of a character 'S' in string "Smartplay is Smart" without using comparator operators.

ANSWER:

```
void main(){
	char *str = "Smartplay is Smart";
	char *ptr = str;
	int count[255];
	count['S'] = 0;
	while( *ptr != '\0'){
		printf("char = %c\n",*ptr);
```

```
            count[*ptr]++;
            ptr++;
        }
        printf("character 'S' occurred %d times \n",count[(int)'S']);
}
```

2.14 Q58. What is the output of the below program?

CODE:

```
struct a{
        char *ptr1;
        char *ptr2;
};
struct b{
        char *ptr3;
        struct a s1;
};
struct b s2 = {"Hyderabad","Bangalore","Pune"};
printf("%s %s",++s2.ptr3 , ++s2.s1.ptr2);
```

ANSWER:
yderabad
une

2.15 Q61. What are the different ways of representing, · an array?

ANSWER:
arr[2], 2[arr], *(arr + 2), *(2 + arr)
· 2d array?
·
ANSWER:
arr[2][3]
((arr + 2) + 3)
*(arr[2] + 3)

2.16 Q62. Given int arr[4]what will the fallowing statements output?

· arr+1
· &arr+1
ANSWER: Here, 'arr' represent the base address of the 1st element of array arr. i.e.
&arr[0] So arr + 1, represent the address of arr[1], i.e &arr[1] '&arr' represent the base address of the entire array. So '&arr + 1' represent the address after the arr[4].
i.e. if arr is stores in location 1000, then printf("%p", &arr + 1); // value is 1016

2.17 Q63. Given

int arr[5] = {1,2,3,4,5};
int (*p)[5];
int (*q)[3];

tell if the operations p = arr, p = &arr, q = arr, q = &arr
ANSWER:
p = arr; // warning : assignment from incompatible pointer type
p = &arr; //no error
q = arr; //warning
q = &arr; //warning
In the above fields, it'll work.To access the elements using the pointer to array, (*p)[1] ; // value is 2, here

2.18 Q64. Given char str[20] = "hello"; what is the output of the below statements

sizeof(str);
strlen(str);
ANSWER:
sizeof(str) ; // return 20
strlen(str); // return 5

2.19 Given char char str[] = "hello"; what is the output of the below statements

sizeof(str);
strlen(str);
ANSWER:
sizeof(str); // return 6. as '\0' included in the string.
strlen(str); // return 5

2.20 Out of the two declarations below, which is correct and which is wrong?

int arr[][3] = {1,2,3,4,5,6};
int arr[3][] = {1,2,3,4,5,6};
ANSWER:
int arr[][3] = {1,2,3,4,5,6}; //OK
int arr[3][] = {1,2,3,4,5,6}; // wrong
Here, the leftside dimension is adjusted according to the no. of elements in each row(mentioned in the right dimension as 3).

2.21 Is the below declaration correct? Explain?

1. int a =1 , b =2 , c=3;
int arr[] = {a, b, c};
2. int n = 5;
int arr[n];
ANSWER:
int a =1 , b =2 , c=3;
int arr[] = {a, b, c}; //wrong
Similarly, no. of elements can't be variable
int n = 5;
int arr[n]; //wrong

2.22 Will the below code works correctly?

CODE:
```
main(){
int arr[] = {1,2,3};
show(arr, 3);
}
```

```
int show(int p[3], int n){
sizeof(p); // here size = 4bytes. i.e. size of a pointer // As it's a pointer, we can increment p
p++; //OK
}
```

ANSWER: Yes

2.23 Illustarte with a piece of code, how to copy elements of one array to another?

ANSWER: Structure can be used as below

```
struct arr{
	char array[5];
} s_arr1, s_arr2;
int main() {
	s_arr1.array[0] = s_arr1.array[1] = s_arr1.array[2] =
		s_arr1.array[3] = s_arr1.array[4] = '2';
	s_arr2 = s_arr1;
	printf(" s_arr2.array[3] = %c\n" , s_arr2.array[3]);
	return 0;
}
```

Output: s_arr2.array[3] = 2

2.24 What is the output of the below code?

```
#include<stdio.h>
int main(){
struct s1 {
	char *z;
	int i;
struct s1 *p;
};
struct s1 a[]={
	{"nagpur",1,a+1},
	{"raipur",2,a+2},
	{"hyd",2,a+2}
};
struct s1 *ptr=a;
printf("%s\n",++(ptr> z));
printf("s\n",a[(++ptr)> i].z);
}
```

ANSWER:

agpur hyd

2.25 What is the output of the below code?

CODE:

```
main(){
struct a{
char ch[7];
char *str;
```

```
};
struct b{
char*p;
int data;
};
struct c{
char *c;
struct a ss1;
struct b ss2;
};
struct c s3 ={"hello","world","universe","galaxy",10};
printf("%3s %3d %s\n" ,s3.c,++s3.ss2.data,++s3.ss1.str);
}
```

ANSWER: hello 11 niverse

2.26 What is the output of the below code?

```
main() {
        char ch[10];
        int i;
        for(i=0;i<9;i++)
                *(ch+i)=67;
        *(ch+i)='\0';
        printf("%s\n",ch);
}
```

ANSWER:
CCCCCCCCC

2.27 What is the output of the below code?

```
main(){
struct node{
            int data;
            struct node *link;
};
struct node *p,*q;
p=malloc(sizeof(struct node));
q=malloc(sizeof(struct node));
printf("%d %d\n",sizeof(p),sizeof(q));
return 0;
}
```

ANSWER:
4 4

2.28 What is the output of the below code?

```
main(){
char s[]="Smart";
printf("%d\n", *(s+strlen(s)));
return 0;
};
```

ANSWER:
0

2.29 Given int a[10], the operation a++ is allowed or not?Explain.

ANSWER:
Here a cant be altered that is a++ is not allowed because base address of array will be stored in read only section(.text section).
but elements a[0],a[1] can be altered because it will be stored in stack.

2.30 what is the efficient way to store strings. Illustrate with an example.

ANSWER:
Consider, char s[5][10];
size allocated to this will be 5*10*sizeof(char), that is 5*10*1 = 50 bytes.
if we use this to store strings. we can store 5 strings of length 10 bytes.
if we use it or not the memory will be allocated and will be wasted if not used efficiently.
ex: name would be
"kiran\0" \\6 bytes
"smart\0" \\6 bytes
"play\0" \\5 bytes
"bus\0" \\4 bytes
"car\0' \\4 bytes
so total is 25 bytes and remaining 25 bytes go wasted.
in order to make use of this efficiently we can use array of char pointers.
char *s[5];
here size of s will be 5*sizeof(pointer) that is 5*4 = 20 and this accommodate any variable length strings. so memory will be efficiently used and size of string doesn't matter, it can exceed 10 or can be less than 10.

2.31 How to access array elements using pointer to an array?

ANSWER:
Consider,
char b[10];
char (*s)[10];
here s is a pointer to an array of 10 elements holding characters. and b is an array of characters
if you want to access b[10] using array pointer s. have to map the address of b to s and can access it like a normal array.
s = b;
(*s)[0] >
b[0]
.
.
.
.
(*s)[9] >
b[9]

2.32 Given char a[10][5][3]; and assuming base address of a = 1000

1. Predict the results
&a = ?
a = ?
a[0] = ?
a[0][0] = ?
a[0][0][0] =?
ANSWER:
&a = 1000
a = 1000
a[0] = 1000
a[0][0] = 1000
a[0][0][0] =value stored in that location
reason will be &a will contain the address of entire 3 dimensional array
a will contain the address of the 2 dimensional address that is a[5][3]
a[0] will contain the address of 1 dimensional array that is a[3]
a[0][0] will contain the address of elements in 1 d array
a [0][0][0] will contain the value contained in 0th location.

2.33 Predict the results.

&a+1 =
a+1 = ?
a[0]+1 = ?
a[0][0]+1 = ?
a[0][0][0]+1 =?
ANSWER:
&a+1 = 1150
a+1 = 1015
a[0]+1 = 1003
a[0][0]+1 = 1001
a[0][0][0]+1 =value of 0th location + 1
reason is simple as i have stated above these locations contain the address
of the multidimensional
array. so the size of respective array gets
incremented

2.34 If we just want to iterate in for loop and print it, Which one is faster and why?

a. Array
b. Link List.
Ans. Arrays. As it is stored in the contiguous memory location

---

2.35 What will be the output of the program?

```
#include<stdio.h>
#define var 3
int main(){
char *cricket[var+~0]={"clarke","kallis"};
char *ptr=cricket[1+~0];
printf("%c",*++ptr);
return 0;
```

```
}
```

Ans:l

2.36 What will be the output of the program?

```
#include<stdio.h>
#include<string.h>
int main(){
	register a = 25;
	int *p;
	p=&a;
	printf("%d ",*p);
	return 0;
}
```

Ans:compile error ,address of register variable ‘a’ requested

*2.37 What is the difference between char s[] and char *s in C?*

Ans:
The difference here is that
char *s = "Hello world";
will place Hello world in the readonly
parts of the memory and making s a
pointer to that, making any writing operation on this memory illegal. While
doing:
char s[] = "Hello world";
puts the literal string in readonly
memory and copies the string to newly
allocated memory on the stack. Making
s[0] = 'J';
legal

2.38 what is dangling pointer

Ans:

```
{
char *dp = NULL;
/* ... */
{
char c;
dp = &c;
} /* c falls out of scope */
/* dp is now a dangling pointer */
}
```

2.39 program to find the endianness of a processor?

Answer:

```
unsigned int i = 0xabcdef12;
unsigned char *pc = (unsigned char *) &i;
if (*pc == 0x12) {
/* little endian */
printf(“little endian \n”);
```

```
}
else {
/* big endian */
printf("big endian \n");
}
```

2.40 What is the o/p of the follwoing program?

```
int main() {
int a[] = {10, 20, 30, 40, 50};
printf("%d\n", 2[a]);
}
```

Ans: 30
Array indexes must be always possitive. So 2[
a] will be treated as (2[a])

2.41 How can we find whether the machine is little endian or big endian?

ans:

```
int main(){
	int i=1;
	char *p=(char*)&i;
	if(*p==1)
		printf("Machine is Little endian");
	else
		printf("Macine is big endian");
	return 0;
}
```

Explanation: Little endian means LSB will be stored first and Big endian
means MSB will be stored first. In the code checking which one whether MSB
or LSB stored first and printing based on the value.

2.42 What is the o/p of the program?

```
int main(){
	int a[]={1,2,3,4,5};
	int *p[]={a,a+1,a+2,a+3,a+4};
	int **ptr=p;
	ptr++;
	printf("%d %d %d\n",ptrp,*
			ptra,**
			ptr);
	*ptr++;
	printf("%d %d %d\n",ptrp,*
			ptra,**
			ptr);
	*++ptr;
	printf("%d %d %d\n",ptrp,*
			ptra,**
			ptr);
	return 0;
```

}ans: 1 1 2
2 2 3
3 3 4
Explanation:
ptr is incrementing first, ptr comes to next position ptr is now at p+1
location so ptrp
gives 1. *ptr is a+1 *ptra
gives 1 and **ptr is value
that lies at that pointer and that is 2.
Next is *ptr++ and here ++ will gets higher priority than *, so ptr++ will
execute first and value at that point *p is a+2. ptr is now at p+2 location
so ptrp
gives 2 and **ptr is value that lies at that pointer and that is 3.
Next is *++ptr and here ++ptr will execute first and value at that point *p
is a+3. ptr is now at p+3 location so ptrp
gives 3 and **ptr is value that
lies at that pointer and that is 4.

2.43 what is the o/p the below code

```
int main(){
char *s = "hello";
*s='H';
printf("%s\n", s);
}
```

O/P = segmentation fault:
Reason :: string literals will be stored in read only part of data segment, so we cannot write.

2.44 what is the o/p the below code

```
int main(){
int *ptr=NULL;
*ptr = 1;
printf("%d\n", *ptr);
}
```

O/P = segmentation fault:
Reason :: NULL pointers can not be dereferenced.

2.45 what is thr o/p the below code

```
int main(){
char *p;
strcat(p, "abc");
printf("%s",p);
return 0;
}
```

O/P = segmentation fault:
Reason :: Memory not allocated for p.

2.46 What is a segmentation fault

A segmentation fault occurs when a program attempts to access a memory
location that it is not allowed to access, or attempts to access a memory

location in a way that is not allowed (for example, attempting to write to a readonly
location, or to overwrite part of the operating system).
Reasons for segmentation fault:
a. Working on a dangling pointer and dereferencing a null pointer.
b. Writing past the allocated area on heap.
c. Operating on an array without boundary checks.
d. Freeing a memory twice.
e. Working on Returned address of a local variable
f. Running out of memory(stack or heap)

2.47 What is a function pointer explain its usage.

prototype of a function pointer: ReturnType (*PtrToFun)(arguments if any);
A function pointer is a variable which is used to hold the starting address of a functions and the same can be used to invoke a function. It is also possible to pass address of different functions at different times thus making the function more flexible and abstract.
a.Function pointers are mainly used in callback mechanisms.
b.Also used in Functions as Arguments to Other Functions.

2.48 Write a program to implement memmove in c.

```
void * my_memmove(void *source,void *destination,int count) {
int i = 0;
char *src = (char *)source;
char *dest = (char *)destination;
void * ptr = dest;
char * src_beg = src;
char * src_end = src_beg + count;
char * dest_beg = dest;
char * dest_end = dest_beg + count;
if(src == dest) {
printf("Same string, same address\n");
return ptr;
}
printf("Detecting whether there is any ovelap\n");
if((dest_beg >= src_beg && dest_beg <= src_end) || (dest_end >= src_beg &&
dest_end <= src_end)) {
printf("\n:(
there is overlap\n");
//Copy from higher addresses to lower addresses
for( i = count ; i > 0 ; i){
*(dest + i) = *(src + i);
}
printf("dest is > %s\n",dest);
return ptr;
}
printf("src is > %s\n",src);
//No overlap, copy from lower addresses to higher addresses
```

```
for( i = 0 ; i < count ; i++) {
*(dest + i) = *(src + i);
}
printf("dest is %s\n",dest);
return ptr;
}
```

2.49 Define sizeof(array) or sizeof(int) pointer?

sizeof(x) ((&x +1) (&x)) it works for any data type

2.50 how do you pass any two dimensional array using pass by value method?

```
void func(int arrayx[][6],int rows){
}
int main(){
int array[3][6];
func(array);
}
```

2.51 column size must be specified in arg list of func like int arrayx[][6] as in prev question . **Why?**

if we want to access any value say arrayx[1][2] , compiler interpret it as arrayx + col_size *1 + 2 ... so for compiler column size is required to evaluate any addresses within the received func.

2.52 What is the o/p of the following program.

```
#include <stdio.h>
int main(){
char s[] ="angel";
s[6]='d';
printf("%s\n",s);
return 0;
}
```

Ans.angel

2.53 Print the correct output.

```
#include <stdio.h>
#define s str[]
int main(){
char s ="come sep";
printf("%s\n",s);
return 0;
}
```

Ans.Error.

expected expression before ‘]’ token. because, You use [] in the declaration to indicate that it is an array, and when indexing to get at individual array elements, but when you refer to the array as a whole you just use the name not [].

2.54 What will the o/p of the following program?

```
#include <stdio.h>
int main(){
	char s1[]="jerry";
	char s2[]="ferry";
	int i,j,k;
	i= strcmp(s1,"jerry");
	j= strcmp(s1,s2);
	k= strcmp(s1,"jerry boy");
	printf("%d %d %d \n",i,j,k);
	return 0;
}
```

Ans.0,
4, 32

2.55 what will be the o/p of the following program.? Why?

```
#include <stdio.h>
int main(){
int y=1;
if(y & (y=2))
printf("true =%d",y);
else
printf("false =%d",y);
}
```

Ans.true
=2

Explanation:here, inside the if condition first the value of y will assign to 2 (BODMAS rule), after that it will check the condition in if(y & (y=2)) i.e, if(2 & 2) condition true. so, the result is true =2.

2.56 What is the o/p of the following program.

```
#include<stdio.h>
main(){
	int a[2][2][2] = { {1,2,3,4}, {5,6,7,8} };
	int *p,*q;
	p=&a[2][2][2];
	*q=***a;
	*q1=**a[1];
	printf("%d%
			d %
			d",*p,*q1,*q);
}
```

Output : garbage value 5
1

2.57 What is the o/p of below program.

```
void main(){
char p[10]="Program",*p1;
```

```
p1=p;
while(*p1!='\0')
++*p1++;
printf("%s %s",p,p1);
}
```

Output : Qsphsbn

2.58 What is the o/p of below program.

```
void main() {
    int c[ ]={1.2, 3, 4, 5.6 , 7};
    int j,*p=c,*q=c;
    for(j=0;j<5;j++) {
        printf(" %d\t ",*c);
        ++q; }
        printf("\n");
        for(j=0;j<5;j++){
            printf(" %d ",*p);
            ++p; }
}
```

Output : 1 1 1 1 1
1 3 4 5 7

2.59 char *p="Hello", what is the value for below cases:

a. *&*p H
b. *&*p+1 I
c. *&(*p+1) error:
lvalue required as unary '&' operand
d. *&*(p+1) e
e. *&*(p+5) no
output, blank

2.60 What is the o/p of below program?

```
void main() {
int i;
float a = 5.2;
char *ptr;
ptr = (char *)&a;
for(i0; i<3;i++)
printf("%d",*ptr++);
}
```

Answer: 102 102 90

2.61 What is the o/p of below program?

```
#include<string.h>
void main( ){
printf("%d%d",sizeof("string"),strlen("string"));
}
```

Answer: 7 6

2.62 What is the o/p of below program?

```
main(){
char s[]={'a','b','c','\n','c','\0'};
char *p,*str,*str1;
p=&s[3];
str=p;
str1=s;
printf("%d",++*p + ++*str132);
}
```

Answer:
77
Explanation:
p is pointing to character '\n'. str1 is pointing to character 'a' ++*p. "p is pointing to '\n' and that is incremented by one." the ASCII value of '\n' is 10, which is then incremented to 11. The value of ++*p is 11. ++*str1, str1 is pointing to 'a' that is incremented by 1 and it becomes 'b'. ASCII value of 'b' is 98.
Now performing (11 + 98 – 32), we get 77("M");
So we get the output 77 :: "M" (ASCII is 77).

2.63 What is the o/p of below program?

```
void main( ){
char *str = "cpointer";
printf("%*.*s",10,7,str);
}
```

Answer : cpoint

2.64 What is the o/p of the following program?

```
#include <stdio.h>
struct student{
int no;
char name[10];//line 5
}st;
main(){
st.no = 10;
st.name = "smartplay";
printf("no: %d \n name:%s\n",st.no,st.name);
}
```

Ans: Compiler Error
Error: error: incompatible types when assigning to type ‘char[10]’ from type ‘char *’
Explanation:
We have to use "strcpy" instead of direct copying. or
We can pointer instead of char array.
if you replace line 5 with char *name: it will work.

2.65 What is the output of below program?

```
#include <stdio.h>
main(){
int i, j;
```

```
scanf("%d %d"+scanf("%d %d", &i, &j));
printf("%d %d", i, j);
}
```

Output:
2
2
2
Segmentation fault
Explanation: In the statement scanf("%d %d"+scanf("%d %d", &i, &j)); the first two values are read into i and j. for the third value it is a null pointer assignment. so segmentation fault occurs at run time.

2.66 What is the o/p of the below program?

```
#include <stdio.h>
main(){
	int i, j, *p;
	i = 25;
	j = 100;
	p = &i;
	printf("%f\n", j/(*p) );
}
```

Output:0.000000
Explanation:
The bit representation of 1 is not the same for float.even the size of the data types are same for 32 bit compiler the way the compiler fetches the value is different.so it prints 0.0000

2.67 What is the output of the following?

```
#include<stdio.h>
void f(char**);
main(){
char * argv[] = { "ab" ,"cd" , "ef" ,"gh", "ij" ,"kl" };
f(argv);
}
void f( char **p ){
char* t;
t= (p+= sizeof(int))[1];
printf( "%s" , t);
}
```

Ans: gh

2.68 What is the output of the following?

```
main(){
	int a[][3]={1,2,3,4,5,6};
	int (*ptr)[3]=a;
	printf("%d,%d,",(*ptr)[1], (*ptr)[2]);
	++ptr;
	printf("%d,%d",(*ptr)[1], (*ptr)[2]);
```

```
}
Ans:2,3,5,6
```

2.69 What is the output of the following?

```
main(){
int a[5]={1,2,3,4,5};
int *ptr=(int*)(&a+1);
printf("%d,%d\n",(*a+1),*(ptr1));
}
Ans:2,5
```

2.70 What is the output of the following?

```
char *gxx(){
static char xx[1024];
return xx;
}
int main(){
char *g="string";
strcpy(gxx(),g);
g=gxx();
strcpy(g,"oldstring");
printf("%s\n", gxx());
}
Answer: oldstring
```

2.71 What is the output of the following?

```
main( ){
	static int a[ ] = {0,1,2,3,4};
	int *p[ ] = {a,a+1,a+2,a+3,a+4};
	int **ptr = p;
	ptr++;
	printf("\n %d %d %d", ptrp,*ptra,**ptr);
	*ptr++;
	printf("\n %d %d %d", ptrp,*ptra,**ptr);
	*++ptr;
	printf("\n %d %d %d", ptrp,*ptra,**ptr);
	++*ptr;
	printf("\n %d %d %d", ptrp,*ptra,**ptr);
}
Answer :
1 1 1
2 2 2
3 3 3
3 4 4
```

2.72 What is the output of the following?

```
#include <string.h>
#include <stdio.h>
int main(){
```

```
const char str[80] = " www.google.com ";
const char s[2] = ".";
char *token;
/* get the first token */
token = strtok(str, s);
/* walk through other tokens */
while( token != NULL ){
printf( " %s\n", token );
token = strtok(NULL, s);
}
return(0);
}
```

OUTPUT:
www
google
com
Syntax: char *strtok(char *str, const char *delim)
str The
contents of this string are modified and broken into smaller
strings (tokens).
delim This
is the C string containing the delimiters. These may vary from
one call to another

2.73 What is the output of the following?

```
#include <stdio.h>
#include <string.h>
int mainn(){
	const char haystack[20] = "DiamondPoint";
	const char needle[10] = "Point";
	char *ret;
	ret = strstr(haystack, needle);
	printf("The substring is: %s\n", ret);
	return(0);
}
```

The substring is: Point Syntax : char *strstr(const char *haystack, const char *needle) haystack This is the main C string to be scanned. needle This is the small string to be searched within haystack string.

2.74 What is the output of the following?

```
xyz(char *t,char *s) {
while(*s!= '\0'){
		*t = *s;
		t++;
		s++;
		}
		*t = '\0';
}
int main(){
```

```
char a[] = "Banglore";
char b[10];
xyz(b,a);
printf("%s\n",b);
return 0;
}
```

output: Banglore

this is the implementation of strcpy();

2.75 Write the implementation of strlen and strcmp?

Strcmp

```
xstrcmp(char s1, char *s2){
        while(*s1 == *s2){
                if (*s1 == '\0')
                        return (0)
                                s1++;
                s2++;
        }
        return(*s1 *s2);
}
```

Strlen

```
xstrlen(char *s){
        int length = 0;
        while(*s! = '\0'){
                lenght++;
                s++;
        }
        return s;
}
```

2.76 What is the output of the following?

```
#include <string.h>
#include <stdio.h>
#include <string.h>
main(){
char *str = "smartplay";
int p1 = sizeof(str);
int p2 = strlen(str);
printf("p1 = %d\np2 = %d\n", p1, p2);
}
```

OUTPUT:

p1 = 8

p2 = 9

2.77 What is the output of the following?

```
int main(int argc, char *argv[]){
```

```
int j;
j = argv[1] + argv[2] + argv[3];
printf("%d", j);
return 0;
}
```

Answer : compilation error : invalid operands to binary + (have ‘char *’ and ‘char *’)
Reason : we can not add strings ,
j = atoi(argv[1]) + atoi(argv[2]) + atoi(argv[3]) will work if you give numeric characters as input while executing (ex : ./a.out 1 2 3 will give output as 6).

2.78 What is the output of the following?

```
int main(int argc, char *argv[]){
 int i=0;
i+=strlen(argv[1]);
while(i>0){
 printf("%c", argv[1][i]);
} return 0;
}
```

suppose if we execute like below.
./a.out one two three
Ans : eno

2.79 What is the output of the following?

```
int main(){
const int x=5;
const int *ptrx;
ptrx = &x;
*ptrx= 10;
printf("%d\n", x);
return 0;
}
```

Ans : at “*ptrx= 10;”,
Compilation error : assignment of readonly location ‘*ptrx’
solution : we can change the value of x using below way *(int *)&x = 10;

2.80 Deallocate memory without using free() in C?

Ansvoid
*realloc(void *ptr,size_t size);
if size=0 then call to realloc is equivalent to free(ptr)

2.81 What is the output of the following?

```
char *getString(){
char *str = "Sampath";
return str;
}
int main(){
```

```
printf("%s", getString());
getchar();
return 0;
}
```

Output: Sampath

The above program works because string constants are stored in Data Section (not in Stack Section). So, when getString returns *str is not lost.

2.82 What is the o/p of the below program?

```
int main(){
register int a=10;
int *p=&a;
printf("%d",*p);
return 0;
}
```

Ans: Compiler error (Register gets stored in CPU. It doesn't have any memory address to access through pointer).

2.83 What will print out?

```
main(){
char *p1="name";
char *p2;
p2=(char*)malloc(20);
memset (p2, 0, 20);
while(*p2++ = *p1++);
printf("%s\n",p2);
}
```

Answer:empty string.

Explanation:

while (*p2++ = *p1++); is equivalent to strcpy(p2, p1);

The pointer p2 value is also increasing with p1 .

*p2++ = *p1++ means copy value of *p1 to *p2 , then increment both addresses (p1,p2) by one , so that they can point to next address . So when the loop exits (ie when address p1 reaches next character to “name” ie null) p2 address also points to next location to “name” . When we try to print string with p2 as starting address , it will try to print string from location after “name” … hense it is null string ….

eg : initially p1 = 2000 (address) , p2 = 3000

*p1 has value “n” ..after 4 increments , loop exits … at that time p1 value will be 2004 , p2 =3004 … the actual result is stored in 3000 n

, 3001 a

, 3002 m

, 3003 e

…

we r trying to print from 3004 …. where no data is present … thats why its printing null .

2.84 what is the output of following

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

```
void main(){
short num[3][2]={3,6,9,12,15,18};
printf("%d %d",*(num+1)[1],**(num+2));
}
```

ans:15 15

Explanation:

*(num+1)[1]
=*(*((num+1)+1))
=*(*(num+2))
=*(num[2])
=num[2][0]
=15

And

**(num+2)
=*(num[2]+0)
=num[2][0]
=15

2.85 Is char a[3] = "abc"; legal? What does it mean?

It declares an array of size three, initialized with the three characters 'a', 'b', and 'c', without the usual terminating '\0' character. The '\0' maybe overwritten by another part of code. This array is therefore not a true C string and cannot be used with strcpy, printf %s, etc safely. But its legal.

2.86 What is the o/p of below program?

```
int mian(){
int ptr[]={1,2,3,4,5,6};
printf("%d",&ptr[3]-&ptr[0]);
return 0;
}
```

o/p:3

2.87 What is the o/p of below program?

```
void foo(int[][3]);
int main(){
int a[3][3]={
{1,2,3},
{4,5,6},
{7,8,10}
};
foo(a);
printf("%d"a[2][1]);
return 0;
}
void foo(int b[][3]){
b++;
```

2.88 What is the o/p of below program?

```
int mian(){
int ptr[]={1,2,3,4,5,6};
printf("%d",&ptr[3]-&ptr[0]);
return 0;
}
```

o/p:3

2.89 What is the o/p of below program?

```
void foo(int[][3]);
int main(){
int a[3][3]={
{1,2,3},
b[1][1]=9;
}
```

o/p:9

2.90 What is the o/p of below program?

```
int main(){
int x;
int arr[]={11,22,33,44,55,66,77,88,99};
x=(arr+2)[3];
printf("%d",x);
}
```

2.91 What is the o/p of below program?

```
int main(){
char *s = "NULL";
if(strcmp(s, NULL) )
printf("Yes");
else
printf("No");
return 0;
}
```

o/p = Segmentation fault

2.92 What is the o/p of below program?

```
int main(){
char *str = "c-pointer";
printf("%*.*s", 10, 7, str);
return 0;
}
```

o/p = c-point

2.93 What is the o/p of below program?

```
int main(){
char s[] = "c\0abcd";
char p[] = "abc\0\0";
printf("%s %s", s, p);
return 0;
}
```

o/p = c abc

2.94 What is the o/p of below program?

```
int main(){
printf("%s", "c" "Question" "Bank");
return 0;
}
```

o/p = CquestionBank

2.95 what is the size of void pointer?

Ans) The size of a void* is a platform dependent value. Typically it's value is 4 or 8 for 32 an 64 bit platforms respectively.

2.96 find length of string without using "string.h" and loops?

```
#include <stdio.h>
int main(){
char str[100];
printf("Enter a string: ");
int len = printf("%s",gets(str));
printf("\nlen = %d\n",len);
return 0;
}
```

2.97 What is the Output?

```
#define PRINT printf("smartplay1"); printf("smartplay2");
#include<stdio.h>
void main(){
int x=1;
if(x)
PRINT
else
printf("smartplay3");
}
```

Ans) Compiler Error --> error: 'else' without a previous 'if'

2.98 Extracting the word "a2b2c4d8u7"?

Output would be

aa
bb
cccc
dddddddd
uuuuuuu

Ans)

```
#include<stdio.h>
#include<stdlib.h>
int main(){
char str[100]="a2b2c4d8u7";
int i, j;
for(i=0;str[i]!='\0';i++){
```

```
if(96 < str[i] && str[i] < 123){
i++;
for(j=0;j<atoi(&str[i]);j++){
        printf("%c",str[i1]);
}
printf("\n");
}
}
printf("\n");
return 0;
}
```

3 Structures

3.1 *What is the output of the below code?*

CODE:

```
#include <stdio.h>
const enum Alpha{
x,
y=5,
z
}p=10;
int main(){
enum Alpha a,b;
a=x;
b=z;
printf("%d",a+bp);
return 0;
}
```

ANSWER: 0+610 = 4

3.2 What is the output of the below code?

CODE:

```
#include <stdio.h>
enum A{
x,y=5
enum B{
p=10,q
}varp;
}Varx;
int main(){
printf("%d %d",x,varp.q);
return 0;
}
```

ANSWER: Compilation error.
Nesting of enum constant is not possible in C and request for member ‘q’ in something not a structure or union.

3.3 What is the output of the below code?

CODE:

```
#include <stdio.h>
enum power{
        element1,
        element2=3,
        element3,
        element4
};
void main(){
int leader[element1+element4]={1,2,3,4,5};
enum power p=element3;
printf("%d",leader[p>>1+1]);
}
```

ANSWER:

we can use enum variables inside the array declaration.

leader[p>>1+1] => leader[4>>1+1] => leader[4>>2] => leader[1] => 2

3.4 How do you initialize selected structure members at declaration?

ANSWER:

use '.' operator at declaration

```
struct test{
int a;
char b;
int c;
};
// Initialising only members a and c
struct test t1 = {
.a = 10;
.c = 11;
}
```

3.5 What is the output of the below code?

CODE:

```
union x{
int a:2;
int b:3;
int c:3;
char xyz;
} ex;
int main(){
ex.xyz = 0x45;
printf("%d ,%d ,%d",ex.a,ex.b,ex.c);
}
```

ANSWER: 1,3,
3

3.6 Find the bug in the below program?

CODE:

```
struct {
int abc : 4;
}s;
int *func(){
return &s.abc;
}
```

ANSWER:

error: cannot take address of bit field abc.

In C, address of a bit field cannot be accessed, the smallest addressable unit is a 'byte'

3.7 What is the output of the below code?

CODE:

```
struct abc{
        int a:3;
        int b:3;
        int c:2;
};
void main(){
struct abc s={2,6,5};
struct abc s1={2,6,5};
printf("%d %d %d\n",s.a,s.b,s.c);
printf("%d %d %d\n",s1.a,s1.b,s1.c);
}
```

ANSWER:

2 2 1
2 2
1

3.8 How to get the offset of a member variable from the structure, If we know the only member of that structure?

CODE:

```
struct str{
int a[100];
char b;
int c;
};
```

i.e. How to get the offset of member variable c?

ANSWER:

&((struct str *)&0x0).c

taking a struct str variable at the base address 0. we can get the address of c.

NOTE: we can't assign any value at that location

3.9 What is the output of the below code?

CODE:

```
typedef struct {
unsigned int a:13;
unsigned int b:20;
} test;
int main() {
printf(" size of bitstr
= %u\n" , sizeof(test));
return 0;
}
```

ANSWER:
size of bitstr
= 8

3.10 If we declare int i, it will occupy 2 or 4 bytes depends on the compiler. If we want to allocate 1 byte for an integer variable what can we do?

ans: Can use bit fields

3.11 What is structure padding?

ans: Allocating a block or chunk of memory at a time.

3.12 If the compiler is gcc then what is the output of the following program?

```
int main(){
typedef struct abc{
int a;
char b;
float c
}abc:
abc ABC;
printf("%d",sizeof(ABC));
return 0;
}
```

ans: 12 bytes
Because in 32 bit compilers structure padding will happen. After char b 3 bytes will be free and in the next block float c will be stored. This structure padding will beused for fast access.

3.13 If the compiler is gcc then what is the output of the following program?

```
#include <stdio.h>
int main(){
typedef struct abc{
int a;
char b;
short int c;
char d;
}abc;
printf("%ld\n",sizeof(abc));
return 0;
}
```

ans: 12 bytes
Here also same structure padding will happen. After char b the short int c will leave 1 byte gap and it gets stored in the next byte because here even padding will happen, the short int gets stored in a even block(even memory location).]

3.14 what is the o/p the below code

```
typedef struct size {
unsigned int a:1;
unsigned int b:31;
unsigned int c:1;
}mystruct;
int main() {
mystruct a;
printf("%d\n", sizeof(a));
return 0;
}
```

o/p = 8

3.15 what is the o/p the below code

```
struct bitfield {
unsigned int a: 1;
unsigned int b: 1;
unsigned int c: 1;
unsigned int d: 1;
};
int main(void){
struct bitfield pipe = {
.a = 1, .b = 0,
.c = 0, .d = 0
};
printf("%d %d %d %d\n", pipe.a,
pipe.b, pipe.c, pipe.d);
printf("%p\n", &pipe.a); /* OPPS HERE */
return 0;
}
```

O/P : error: cannot take address of bitfield

3.16 why structure padding is required?

Faster memory access as all members starting addresses are aligned to a boundary corresponding to their sizes.
but again it is compiler dependent where some can do structure padding and some wont if hardware support unaligned memory access.

3.17 Output of below program with increment operator?

```
main(){
int i=5;
```

```
printf("%d\n",i=++i==6);
}
```

3.18 Output of below program - struct?

```
struct marks{
int p:3;
int c:3;
int m:3;
};
main(){
struct marks s={2,6,5};
printf("%d %d %d",s.p,s.c,s.m);
}
```

3.19 find the size of structure without using sizeof()?

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
struct MyStruct{
int i;
int j;
char k;
};

int main(){
struct MyStruct *p=0;
int size = ((char*)(p+1))-((char*)p);
printf("\nSIZE : [%d]\n\n", size);
return 0;
}
```

3.20 declare a structure of a linked list?

The right way of declaring a structure for a linked list in a C program is

```
struct node {
int value;
struct node *next;
};
typedef struct node *mynode;
```

Note that the following are NOT correct

```
typedef struct {
int value;
mynode next;
} *mynode;
```

The typedef is not defined at the point where the "next" field is declared.

```
struct node {
int value;
```

```
struct node next;
};
typedef struct node mynode;
```

We can only have pointer to structures, not the structure itself as its recursive!

4 Control Statements

4.1 What is the output of the below code?

```
#include <stdio.h>
void main(){
int const x=0;
switch(5/4/3){
case x: printf("one");
break;
case x+1:printf("two");
break;
case x+2:printf("three");
break;
default:printf("four");
}
}
```

ANSWER:
compilation error: case label does not reduce to an integer constant(ie for
x, x+1 , x+2)

4.2 What is the output of the below code?

CODE:

```
#include<stdio.h>
void main(){
        switch(0X0){
                case NULL:printf("one");
                          break;
                case '\0':printf("two");
                          break;
                case 0: printf("three");
                        break;
                default: printf("four");
        }
}
```

ANSWER:
Compilation error.
pointers are not permitted as case values (for NULL) and duplicate case
values are not allowed.

4.3 What is the output of the below code?

```
int i,j;
```

```
i = j = 0;
if(i && j++)
printf(" hello ");
printf("i = %d, j = %d\n",i,j);
```

ANSWER:
hello is not printed because if(0 && x value) which will be false. and the 'j' value is not incremented because in anding lhs to rhs execution is made and in 'if' for anding comparison first value is zero then the next value is not considered and default it will consider as false (as any thing AND with zero is always zero)
i = 0, j = 0

4.4 Put a condition in place of statement 1 so that it prints 'hello world'???

CODE:

```
#include<stdio.h>
main(){
if(statement 1... )
        printf("hello");
else
        printf("world");
}
```

ANSWER:

```
#include<stdio.h>
main(){
if(printf("hello") && 0) //'hello' will printed so printf will
return
//positive value(1)
printf("hello"); // hence 1&&0 will give 0 so 'world'
will print
else
printf("world");
}
```

4.5 What is the output of the below program?

CODE:

```
#include<stdio.h>
void main(){
     int i=10;
     static int x=i;
     if(x==i)
          printf("Equal");
     else if(x>i)
          printf("Greater than");
     else
          printf("Less than");
}
```

ANSWER:
It gives compilation Error
Error: initializer element is not constant
Static Variables are load time entity while auto variables are run time entity.
We can not initialize any load time variable by the run time variable.
In this example i is run time variable while x is load time variable.

4.6 What is the output of the below program?

CODE:

```
#include<stdio.h>
int main(){
int i=0;
for(;i++;printf("%d",i)) ;
printf("%d",i);
}
```

ANSWER:
It Prints output 1.
for the first time, when it enters the loop it checks the condition, as the i value is zero,loop terminates. I increments to 1 value as it is post increment and prints 1

4.7 Relate a,b and c (assume they are bits)

CODE:

```
if(c==0)
a=0;
if(c==1)
a=b;
```

ANSWER:
a=b&c

4.8 which part of ifelse is executed in the below code?

CODE:

```
if(1 == (unsigned)1){
...
}
else{ ...
}
```

ANSWER:
it'll execute if part. Any signed integer compared with unsigned one, that signed no. is converted to unsigned.

4.9 What is the output of the below code?

CODE:

```
float a = 7.5;
```

```
if(a == 7.5)
printf("equal");
else
printf("not equal");
```

ANSWER: not equal
because float has a size of 4 bytes and by default decimal numbers will have 8 bytes,
so the location of mantissa and exponent will be stored in 4 bytes in float, where as in default it will be expanded to 8 bytes so the precision will vary.

4.10 What will be the output of the program?

```
void main( ){
int c=3;
switch(c){
case '3': printf("Hi"); break;
case 3: printf("Hello"); break;
default: printf("How r u ?");
   }
}
```

Ans. Hello. As the 3 in first case will be taken as character.

4.11 Find out the sum of the digits of a given number in single statement?

Ans. for(sum=0; n > 0; sum+=n%10,n/=10);

4.12 What will be the output of the program?

```
#include <stdio.h>
int main(){
	int x=011,i;
	for(i=0;i<x;i++){
		printf("Start ");
		continue;
		printf("End");
	}
	return 0;
}
```

Ans:9 times start

4.13 What will be the output of the program?

```
enum {false,true};
int main(){
	int i=1;
	do{
		printf("%d\n",i);
		i++;
		if(i < 15)
			continue;
```

```
    }while(false);
    return 0;
} Ans:1
```

4.14 What is the o/p of the follwoing program?

```
#include<stdio.h>

int main(){
    int a=10;
    switch(a){
        case '1': printf("ONE\n");
                break;
        case '2': printf("TWO\n");
                break;
defa1ut: printf("NONE\n");
    }
    return 0;
}
```

Answer: It will compile.
It will not print anything.Because, there is NO 'default case' in the switch statement. And 'defa1ut' will be treated as a valid LABEL(Similar to 'goto' label)

4.15 What is the o/p of the follwoing program?

```
#include <stdio.h>
int main(){
    if("Hello World" == "Hello World"){
        printf("Yes ","No ");
    }
    printf(10+"Hello World"4);
    return 0;
}
```

~Answer: Yes World
First of all the statement inside the IF will be executed because we compare the same pointer, which returns 1
(although in some machines this might not be true, because storing similar string literals only once is not required by the ANSI C standard).
The first printf will only print “Yes”, though, because the function only takes one parameter in this case, so “No” is ignored.
The second printf will only print “World” because the pointer to the initial character was incremented by 6 bytes, i.e 104= 6

4.16 What is the o/p of below program?

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

```
int main() {
	switch(*(2 + "AB" "CD" + 1)) {
		case 'A': printf("A"); break;
		case 'B': printf("B"); break;
		case 'C': printf("C"); break;
		case 'D': printf("D"); break;
		default: printf("default"); break;
	}
}
```

~Output: C
Explanation:
*((1+"AB" "CD"+1)) => *(2+ABCD) here A =0 B = 1 C = 2 D = 3 so,
we are pointing to 2nd value i.e., C so it will go to case 'C'.

4.17 compare two float values without using "==" operator

```
#include <stdio.h>
main(){
	float f1=5.55554,f2 = 5.55554;
	if((f1f2))
		printf("Not Equal\n");
	else
		printf("Equal\n");
}
```

Output:Equal
Note: For this question we discussed one solution i.e.,
if(((f10.00001) < f2)&&((f1+0.00001)>f2))
In this solution no.of fractional values must be same otherwise it will fail.

4.18 What is the output of the following?

```
#include<stdio.h>
int main(){
	int a=1;
	switch(a){
		int b=20;
		case 1: printf("b is %d\n",b);
			break;
		default:printf("b is %d in default\n",b);
			break;
	}
	return 0;
}
```

Ans: Garbage value

4.19 What is the output of the following?

```
main(){
int i=0;
for(i=0; i<20; i++){
switch(i){
case 0: i+=5;
case 1: i+=2;
case 5: i+=5;
default: i+=4;
break;
}
printf("%d\t",i);
}
}
```

Answer: 16 21

4.20 What is the output of the following?

```
#include<stdio.h>
#define TRUE 0
#define FALSE 1
#define NULL 0
void main(){
if(TRUE) printf("TRUE");
else if(FALSE) printf("FALSE");
else printf("NULL");
```

OUTPUT: FALSE

1

is considered as FF

4.21 What is the output of the following?

```
enum actor {
        abc = 5,
        def = 2,
        ghi,
        jkl,
};
void main(){
    enum letter l =0;
    switch(l) {
        case abc: printf("abc"); break;
        case def:printf("def"); break;
        case ghi: printf("ghi"); break;
        case jkl: printf("jkl");  break;
        default: printf("default"); break;
    }
}
```

output:jkl

4.22 What is the o/p of the below program?

```
int main(){
for(printf("1");printf("0");printf("2"));
printf("Something\n");
return 0;
}
```

Ans: 120Something
20Something
20Something
.
.
infinite loop

4.23 What's the expected output for the following program and why?

```
enum {false,true};
int main() {
	int i=1;
	do
	{ printf("%d\n",i);
		i++;
		if(i < 15) {
			continue;
			printf("Hello\n");
		}
	}while(false);
	return 0;
}
```

Ans: 1
Explanation: Answer is 1 because of 'continue' statement. The continue statement passes control to the next iteration of the nearest enclosing do, for, or while statement in which it appears, bypassing any remaining statements in the do, for, or while statement body.
In the above program: do while loop will execute once. since while(false) is nothing but while(0). and hence it prints i = 1 and any statements after continue is skipped.

4.24 Does 'default' in switch need a break?

The default case does not require a break; if and only if its at the end of the switch() statement. Otherwise, even the default case requires a break;

4.25 What is more efficient? A switch() or an if() else()?

Both are equally efficient. If else is to be used for unrelated condition testing and multiple condition checks switch is preferred for code readability if the test condition resolves to an integer

---

4.26 how many additions are done in this C-code?

```
for(i=0i<31;i++)
for(j=0;j<31;j++)
for(k=0;k<31;j++){
d=d+1;
}
```

4.27 What is the o/p of below program?

```
#include <stdio.h>
extern int x;
int main(){
	do{
		do{
			printf(.%d.,x);
		}while(!-2);
	}while(0);
}
int x = 10;
```

O/P: 10

4.28 What is the o/p of below program?

```
#include <stdio.h>
int main(){
	int i = 0;
	switch(i){
		case 0:
			int aa = 10;
			printf("a\n");
			break;
		default:
			printf("a=%d\n",aa);
			printf("default\n");
	}
	return ;
}
```

O/P:error: a label can only be part of a statement and a declaration is not a statement

4.29 What is the o/p of below program?

```
#include<stdio.h>
int r();
int main(){
for(r();r();r()) {
printf("%d ",r());
}
```

```
return 0;
}
int r(){
int static num=7; // line 10
return num--;
}
```

O/P:5 2

if I remove static in line 10 then it is infinite loop

4.30 What is the o/p of below program?

```
int main(){
        switch( "c" == "c" ){
                case 0 : printf("Hello");
                          break;
                case 1: printf("World");
                         break;
                default: printf("superb");
                          break;
        }
        return 0;
}
```

o/p = World

4.31 What is the o/p of below program?

```
int main(){
        int i = 0;
        if(i ==0){
                i = ( ( 5, (i=3) ), i=1);
                printf(.%d\n., i);
        }
        else
                printf(.equal\n.);
        return 0;
}
```

o/p = 1

4.32 What is the o/p of below program?

```
#include <stdio.h>
int main(){
int a = 0;
if(a)
int b =0;
}
```

O/P: error: expected expression before ‘int’

Solution:

```
if(a)
```

```
{
int b = 0;
}
```

4.33 What is the o/p of below program?

```
#include <stdio.h>
int a,i;
int main(){
int ss[i];
scanf("%d",&i);
for(a=0;a<i;a++){
ss[a] = a;
printf("%d\n",ss[i]);
}
}
```

O/P:Garbage value

5 Macros and Bit Operations

5.1 Write macros to perform below operations on a number N whose bit at position P is operated upon.

- Set flag
- Clear flag
- Test flag
- Toggle flag

ANSWER:

```
#define SET_FLAG(N, P) ( (N) | (1U << (P)) )
#define CLR_FLAG(N, P) ( (N) & ~(1U << (P)) )
#define TST_FLAG(N, P) ( (N) & (1U << (P)) )
#define TOG_FLAG(N, P) ( (N) ^ (1U << (P)) )
```

5.2 What is the difference between typedef and #define?

ANSWER:

```
typedef int* int_p1;
int_p1 t1, t2; // t1 and t2 are both int pointers.
#define int_p2 int*
int_p2 d1, d2; /* only the first, d1 is a pointer! as preprocessor outputs
int* d1, d2 ;*/
```

Hence it is always advisable to use typedef to define new datatypes.

5.3 WAP to perform the below operations (without using loops **· set the 'n' number of bit from mth postion in a given value int "X" · get the bits from p1th postion to p2th position in a given value int "Y"**

CODE:

```
#include <stdio.h>
/*Bit positions start from 0
macro to set the 'n' number of bit from mth postion in a given value
```

```
int "X" */
#define set_n_number_of_bits(X, n,m) (X=(X | ((~(1<<
n))<<m)))
/*macro to get the bits from p1th postion to p2th position in a given
value int "Y"
Position of bits is retained as in original input*/
#define get_n_number_of_bits(Y, p2,p1) (Y=(Y & ((~(1<<( p2p1 +1)))<<p1)))
int main() {
int X, Y, origX, origY, n, m, p1, p2;
X = Y = origX = origY = 11; // ... 0000 1011
n = 3;
m = p1 = 1; // position 1
p2 = 3; // position 3
set_n_number_of_bits(X, n,m);
get_n_number_of_bits(Y, p2,p1);
printf(" 0x0%x after %d bits from bit postion 'p%d' set = 0x0%x\n"
, origX, n, m, X);
printf(" Bits from %dth postion to %dth position in 0x0%x =
0x0%x\n" , p1, p2, origY, Y);
return 0;
}
```

ANSWER:
0x0b after 3 bits from bit position 'p1' set = 0x0f
Bits from 1th position to 3th position in 0x0b = 0x0a

5.4 WAP to print value of pos1 to pos2 bits in a number?

Eg:no
: 10 0
0 0 0 ... **0 0 1 0** 1 0
Take pos1: 2, pos2: 4
CODE:

```
#include <stdio.h>
unsigned createMask(unsigned a, unsigned b){
unsigned r = 0;
unsigned i;
for (i=a; i<=b; i++)
r = r | (1 << i);
return r;
}
main(){
unsigned x;
x=createMask(2,4);
unsigned n = 10;
unsigned result = ((x & n)>>2);
printf("result:%u\n",result);
}
```

ANSWER: 2

5.5 What is the output of the below code?

CODE:

```
#define call(x) #x
void main(){
printf("%s",call(c program));
}
```

ANSWER:

c program, # is a string operator. c program is considered as a string and is displayed

5.6 WAP Given the input as ABCDEF, Expected output asEFCDAB.

ANSWER:

Method1:

One way is using Masking Technique.

Method 2:

Reverse the entire input and shift the adjacent values.

5.7 Number of swaps required to convert a number from A toB

example: A=4 and B=5. How many bit swaps are needed?

ANSWER:

```
int bit_swaps_required( int a, int b ) {
unsigned int count = 0;
for( int c = a ^ b; c != 0; c = c >> 1 ) {
count += c & 1;
}
return count;
}
```

XOR the bits and count the number of 1's.

5.8 WAP to count the number of 1's or bits set in a number.

ANSWER: One way is to right shift the number, check for last bit and if he last bit is 1 increment the counter.

Smartway:

```
int numOnesInInteger( int num ){
int numOnes = 0;
while( num != 0 ){
num = num & (num – 1);
numOnes++;
}
return numOnes;
}
```

P.S: Same above question can be put across in different way as to check for power of 2 or power of 4.Logic remains the same.

If the given number is x, then isPower = (x !=0 && !(x & (x1)))
x & (x1)
will always give you a 0 if x is a power of 2. !(x & (x1))
should give us what we want but there is one corner case. If x is 0,
then the second term alone would return true when the answer should be
false.
We need the check to make sure x is not 0.

5.9 Write a macro to rotate the bits of a number by n bits.

ANSWER:
Logic is to left shift or right shift the number as per the requirement
and OR the number with the (INT_BITS number)

```
#define INT_BITS 32
/*Function to left rotate n by d bits*/
int leftRotate(int n, unsigned int d){
/* In n<<d, last d bits are 0. To put first 3 bits of n at last, do bitwise or of n<<d with n >>(INT_BITS d)
*/
return (n << d)|(n >> (INT_BITS d));
}
/*Function to right rotate n by d bits*/
int rightRotate(int n, unsigned int d){
/* In n>>d, first d bits are 0. To put last 3 bits of at first, do bitwise or of n>>d with n <<(INT_BITS d)
*/
return (n >> d)|(n << (INT_BITS d));
}
```

5.10 Frequntly asked: Swap even and odd bits of 32 bit number.

ANSWER: We can use masking here.
Mask all the even bits by using bitwise AND between a number and 10101010 or
0Xaaaa and do a right shift.
Mask all the odd bits by using bitwise AND between a number and 01010101 or
0X5555 and do a left shift.
Finally OR the above two.

```
int swapOddEvenBits(int x) {
return (((x & 0xaaaaaaaa) >> 1) | ((x & 0x55555555) << 1));
}
```

Generalizing the formula for masking and then swaping:

```
public static int swapTwoPositions(int number, int i, int j) {
return (((number & (1 << i)) >> i) << j)| (((number & (1 << j)) >> j) << i)| ((~(1 << i)) & (~(1 << j)) &
number);
}
```

There are three parts separated by |(Bitwise OR).
The first part mask the i position and move it to j position.
The second part mask the j position and move it to i position.
The third part turns off bits i and j but keep the rest

5.11 Only one bit is different in a given two integer values of a and b.How will you find out which bit position is different?

ANSWER :

```
int find_bitpostion(int a, int b){
int temp=0;
int count =0;
temp =a^b;
for(count =0; count <sizeof(int); count++){
if( (temp >>count)&1){
	printf(" Bit position is =%d\n",count);
break;
}
}
}
```

5.12 Write a macro to set the 'n' number of bit from mth postion in a given value int "X".

ANSWER: #define set_n_number_of_bits(X, n,m) (X | ((~(1<<n))<<m))

5.13 What will be the output of the program?

```
#include <stdio.h>
void main(){
int a=12;
a=a>>3;
printf("%d",a);
}
```

Ans :2

5.14 Easy way to multiply a number by 7 ? and also for 3

Ans :

a. n*7 == (n<<3) – n

b. n*8=(n<<3)

5.15 Given the input as ABCDEF

Expected output as EFCDAB

Ans:

One way is using Masking Technique.

A=(((num&0x0000ff)<<16) | (num&0x00FF00) | ((num & 0xFF0000) >>16))

5.16 How can we make a particular bit zero?

```
int main(){
int value, position;
printf("Enter the value\n");
scanf("%d",&value);
printf("Enter the position\n");
scanf("%d",&position);
```

```
value= value & (~(1<<position));
printf("%d\n",value);
return 0;
}
```

5.17 write a program to reverse bits in a 32 bit integer with and without using loops.

With loop:

```
int mirror(int input) {
int returnval = 0;
for (int i = 0; i < 32; ++i) {
bit = input & 0x01;
returnval |= bit;
input >>= 1;
returnval <<= 1;
}
return returnval;
}
```

Without loop (using a macro):

a)

```
#define REVERSEBITS(b) (BitReverseTable[b])
static BYTE BitReverseTable[256] =
{
0x00, 0x80, 0x40, 0xc0, 0x20, 0xa0, 0x60, 0xe0,
0x10, 0x90, 0x50, 0xd0, 0x30, 0xb0, 0x70, 0xf0,
0x08, 0x88, 0x48, 0xc8, 0x28, 0xa8, 0x68, 0xe8,
0x18, 0x98, 0x58, 0xd8, 0x38, 0xb8, 0x78, 0xf8,
0x04, 0x84, 0x44, 0xc4, 0x24, 0xa4, 0x64, 0xe4,
0x14, 0x94, 0x54, 0xd4, 0x34, 0xb4, 0x74, 0xf4,
0x0c, 0x8c, 0x4c, 0xcc, 0x2c, 0xac, 0x6c, 0xec,
0x1c, 0x9c, 0x5c, 0xdc, 0x3c, 0xbc, 0x7c, 0xfc,
0x02, 0x82, 0x42, 0xc2, 0x22, 0xa2, 0x62, 0xe2,
0x12, 0x92, 0x52, 0xd2, 0x32, 0xb2, 0x72, 0xf2,
0x0a, 0x8a, 0x4a, 0xca, 0x2a, 0xaa, 0x6a, 0xea,
0x1a, 0x9a, 0x5a, 0xda, 0x3a, 0xba, 0x7a, 0xfa,
0x06, 0x86, 0x46, 0xc6, 0x26, 0xa6, 0x66, 0xe6,
0x16, 0x96, 0x56, 0xd6, 0x36, 0xb6, 0x76, 0xf6,
0x0e, 0x8e, 0x4e, 0xce, 0x2e, 0xae, 0x6e, 0xee,
0x1e, 0x9e, 0x5e, 0xde, 0x3e, 0xbe, 0x7e, 0xfe,
0x01, 0x81, 0x41, 0xc1, 0x21, 0xa1, 0x61, 0xe1,
0x11, 0x91, 0x51, 0xd1, 0x31, 0xb1, 0x71, 0xf1,
0x09, 0x89, 0x49, 0xc9, 0x29, 0xa9, 0x69, 0xe9,
0x19, 0x99, 0x59, 0xd9, 0x39, 0xb9, 0x79, 0xf9,
0x05, 0x85, 0x45, 0xc5, 0x25, 0xa5, 0x65, 0xe5,
0x15, 0x95, 0x55, 0xd5, 0x35, 0xb5, 0x75, 0xf5,
0x0d, 0x8d, 0x4d, 0xcd, 0x2d, 0xad, 0x6d, 0xed,
```

```
0x1d, 0x9d, 0x5d, 0xdd, 0x3d, 0xbd, 0x7d, 0xfd,
0x03, 0x83, 0x43, 0xc3, 0x23, 0xa3, 0x63, 0xe3,
0x13, 0x93, 0x53, 0xd3, 0x33, 0xb3, 0x73, 0xf3,
0x0b, 0x8b, 0x4b, 0xcb, 0x2b, 0xab, 0x6b, 0xeb,
0x1b, 0x9b, 0x5b, 0xdb, 0x3b, 0xbb, 0x7b, 0xfb,
0x07, 0x87, 0x47, 0xc7, 0x27, 0xa7, 0x67, 0xe7,
0x17, 0x97, 0x57, 0xd7, 0x37, 0xb7, 0x77, 0xf7,
0x0f, 0x8f, 0x4f, 0xcf, 0x2f, 0xaf, 0x6f, 0xef,
0x1f, 0x9f, 0x5f, 0xdf, 0x3f, 0xbf, 0x7f, 0xff
};
```

or

b)Without loop (using bit wise operators).

```
BYTE ReverseBits(BYTE b){
BYTE c;
c = ((b >> 1) & 0x55) | ((b << 1) & 0xaa);
c |= ((b >> 2) & 0x33) | ((b << 2) & 0xcc);
c |= ((b >> 4) & 0x0f) | ((b << 4) & 0xf0);
return(c);
}
```

5.18 write a program to extract 10 bit temperature reading (negative or positive value) from sensor registers of size byte each located at adress 0x30 and 0x31 rectively.

out of 10 bits 0x30 hold two Lsbs 10 at bit positions 7 and 8 respectively.

remaining 8 bits are from 0x31 reg that holds 92 at bit positions 7 and 0 respectively.

0x30 > b1 b0 X X X XXX 0X31

- ➢ b9 b8 b7 b6 b5 b4 b3 b2

Ans:

```
short val_temp = * ( char *)(0X31);
short val = * ( char *)(0X30);
val = (val << 2) |(val_temp);
// do sign extension logic for negative sensor readings
val = val <<6;
val = val >>6;
```

5.19 What is the o/p of the following program.

```
#include <stdio.h>
int main(){
char c=48;
int i,mask=01;
for(i=0;i<=5;i++){
printf("%c",c|mask);
mask =mask<<1;
}
printf("\n");
}
```

Ans.12480

5.20 What will the o/p of the following program?

```
#include <stdio.h>
int main(){
int a=5,b=7,
c=0,d;
d= ++a && ++b || ++c;
printf("%d %d %d %d",a,b,c,d);
}
```

Ans.6,
6,0,1

The logical operators && and || are used when evaluating two expressions to obtain a single relational result. The operator && corresponds with Boolean logical operation AND. This operation results true if both its two operands are true, and false otherwise. The following panel shows the result of operator && evaluating the expression a && b:

&& OPERATOR
a b a && b
true true true
true false false
false true false
false false false

The operator || corresponds with Boolean logical operation OR. This operation results true if either one of its two operands is true, thus being false only when both operands are false themselves. Here are the possible results of a || b:

|| OPERATOR
a b a || b
true true true
true false true
false true true
false false false

5.21 Print the o/p.

```
#include <stdio.h>
  int main(){
              if((~0) ==1)
                    printf("hello");
              else
                    printf("bye");
        }
```

Ans.bye

5.22 What is the o/p of below program?

```
#include<stdio.h>
#define max 3+2+3
```

```
#define max2 max*2
int main() {
int a;
a = max * max;
printf("%d\n",a);
printf("%d\n",max2);
return 0;
}
```

Output:
19
11
Explanation:
a = max * max; this will become 3+2+3*3+2+3 => 3+2+9+2+3 => 19
max2 = max *2; this will become 3+2+3*2 => 11

5.23 What is the output of the following?

```
#include <stdio.h>
#define f(a,b) a##b
#define g(a)
#a
#define h(a) g(a)
int main(){
printf("%s\n",h(f(1,2)));
printf("%s\n",g(f(1,2)));
return 0;
}
```

Ans: 12
f(1,2)

5.24 What is the output of the following?

```
#include<stdio.h>
#define f(g,g2) g##g2
main(){
int var12=100;
printf("%d\n",f(var,12));
}
```

Answer: 100

5.25 What is the o/p of the below program?

```
int main(){
int a, b=2, c=1, d=2, e=1;
a = b && c | d && e & c;
printf("%d,a);
return 0;
}
```

Ans: 1

5.26 The >>, <<, >>=, <<= operators

· Operators >> and << can be used to shift the bits of an operand
to the right or left a desired number of positions.
· The number of positions to be shifted can be specified as a constant, in a variable or as an expression. Bits shifted out are lost.
· For left shifts, bit positions vacated by shifting always filled with zeros. Here arithmetic and logical shifts are generally same
· For right shifts, bit positions vacated by shifting filled with zeros for unsigned data type and with copy of the highest (sign) bit for signed data type.
In C, right shifts of signed datatypes always undergo Arithmetic shift. For unsigned types, arithmetic and logical shifts are same

- The left shift operator can be used to achieve quick multiplication by a power of 2.
- The right shift operator can be used to do a quick division by power of 2 (unsigned types only).
- The operators >> and <<, dont change the operand
- The operators >>= and <=< also change the operand after doing the shift operations.

5.27 Whats short-circuiting in C expressions?

The right hand side of the expression is not evaluated if the left hand
side determines the outcome. That is if the left
hand side is true for || or false for &&, the right hand side is not
evaluated.
1 || (expression) = 1 always
0 && (expression) = 0 always
Hence expression is not evaluated

5.28 Does C have boolean variable type?

C does not have a boolean variable type. One can use ints, chars,
#defines or enums to achieve the same in C.
· #define TRUE 1
#define FALSE 0
· enum bool {false, true};
An enum may be good if the debugger shows the names of enum constants
when examining variables.

5.29 What is the difference between enumeration variables and the preprocessor #defines?

- Enumerations have Numeric values assigned automatically
- With Enumerations, the debugger display the symbolic values
- Enumerations obey block scope

5.30 whats the difference between these two?

#define ABC 5
and
const int abc = 5;

Ans: There are two main advantages of the second one over the first technique
1. The type of the constant is defined. This allows for some type checking by the compiler.
2. These constants are variables with a definite scope. The scope of a variable relates to parts of your program in which it is defined.

5.31 Write a function invert(x,p,n) that returns x with the nbits that begin at position p inverted (i.e., 1 changed into 0 and vice versa), leaving the others unchanged.

```
unsigned invert(unsigned x,int p,int n){
return (x ^ (~(~0<<n)) << (pn+ 1));
}
```

5.32 implementation of getbits?

```
unsigned getbits(unsigned x, int p, int n){
return (x >> (p+1 -n)) & ~(0 << n);
}
```

5.33 What is the o/p of below program?

```
#define call(x) #x
int main(){
printf("%s", call(c/c++) );
return 0;
}
```

o/p = c/c++

---

6 Dynamic Memory Allocation

6.1 How to allocate memory with starting address multiple of four?you can create a user defined function using internally malloc.

ANSWER:
Allocate memory using malloc with (needed memory + 4).
check whether the memory allocated is multiple of 4.If not, create user starting address as (address+(address%4))
Track both the addresses and use for memory access and free.

6.2 What is the Difference between MEMCPY and MEMMOVE?

```
char str[]="ABCDEFGH";
MEMCPY(str+2,str,4);
MEMMOVE(str+2,str,4)
```

ANSWER:
MEMCPY gives you the output ABABABGH
MEMCPY overwrites the memory.
MEMMOVE gives you the output ABABCDGH

6.3 Whether physical memory address information get update in the heap memory segment when malloc return virtual address?

ANSWER: Heap memory segment maintains one page table entry for dynamic memory allocation of calloc and malloc. Whenever memory allocate dynamically it will return virtual address to the programme. No physical address (ram) information updated in the page table entry till that virtual address get accesses or initialized.
For calloc: Calloc return address are initialized with zero (internally executing memset after allocate memory), Hence the actual physical address information update in the page table entry of calloc.
For malloc: malloc return address are not initialized with zero. Hence actual physical address information wont available in the page table entry. Whenever acces/ initialize the malloc virtual address, it will create a memory in the physical address and update the physical address information in the page table entry of malloc.

6.4 If the heap memory starts from address 2000 (i.e virtual address) and try to allocate memory by using malloc of 100 bytes. what is the return address of malloc?

ANSWER: Assume: house keeping information (i.e number of bytes allocated , this memory region is free or not,access information..etc..) takes 20 bytes.
For malloc,it will return starting address 2020.While we freeing the memory will pass address 2020.In the definition of free() function it will decrement the size of house keeping information (20 bytes) from the given address and proceed further operation.

6.5 what is memory leak

Ans : A memory leak is caused when you allocated memory, haven't yet deallocated it, and you will never be able to deallocate it because you can't access it anymore.
eg: If an exception is raised between allocation and deallocation, memory leak will occur.

6.6 Is the cast to malloc() required at all?

Ans : Before ANSI C introduced the void * generic pointer, these casts were required because older compilers used to return a char pointer.
int *myarray;
myarray = (int *)malloc(no_of_elements * sizeof(int));
But, under ANSI Standard C, these casts are no longer necessary as a void pointer can be assigned to any pointer. These casts are still required with C++, however.

---

7 Data Structures

7.1 What are the differences between Linked List and Array?

ANSWER:
Linked List:

- ❖ Linked list consists of data nodes, each pointing to the next in the list.
- ❖ Size of Linked List grows.
- ❖ No need to know the size ahead.
- ❖ Deleting a node is simple.

Array:

- ❖ An array consists of contiguous chunks memory.
- ❖ Arrays are fixed in size.
- ❖ we need to know the size ahead.
- ❖ For deleting a element we need to perform Shift operations.

7.2 WAP to implement the below operations on linked list

· **Find Middle node in a Linked List?** · **Reverse a Linked List?**
· **Swap Nth node from beginning and ending of Linked List?**

CODE:

```
#include<stdio.h>
#include<stdlib.h>
/* Link list node */
struct node{
        int data;
        struct node* next;
};

void MiddleNode(struct node *head){
     int count = 0;
     struct node *mid = head;
     while (head != NULL){
          if (count & 1)
               mid = mid>
                    next;
          ++count;
          head = head>
               next;
     }
     if (mid != NULL)
          printf("\n\nThe middle element is: %d\n\n", mid>
                    data);
}

/* Function to reverse the linked list */
void reverse(struct node** head_ref){
     struct node* prev = NULL;
     struct node* current = *head_ref;
     struct node* next;
     while (current != NULL){
          next = current>
               next;
          current>
               next = prev;
          prev = current;
          current = next;
```

```
        }
        *head_ref = prev;
}
int countNodes(struct node *s){
        int count = 0;
        while (s != NULL){
                count++;
                s = s>
                        next;
        }
        return count;
}


void swapNthNodes(struct node **head_ref, int k){
        int n = countNodes(*head_ref);
        int i;
        if (n < k)
                return;
        if (2*k 1
                        == n)
                return;
        struct node *x = *head_ref;
        struct node *x_prev = NULL;
        for (i = 1; i < k; i++){
                x_prev = x;
                x = x>
                        next;
        }
        struct node *y = *head_ref;
        struct node *y_prev = NULL;
        for (i = 1; i < nk+1; i++){
                y_prev = y;
                y = y>
                        next;
        }
        if (x_prev)
                x_prev>
                        next = y;
        if (y_prev)
                y_prev>next = x;
        struct node *temp = x>
                next;
        x>
                next = y>
                next;
```

```
	y>
		next = temp;
	if (k == 1)
		*head_ref = y;
	if (k == n)
		*head_ref = x;
}

/* Function to push a node */
void push(struct node** head_ref, int new_data){
	struct node* new_node = (struct node*) malloc(sizeof(struct node));
	new_node>
		data = new_data;
	new_node>
		next = (*head_ref);
	(*head_ref) = new_node;
}

/* Function to print linked list */
void printList(struct node *head) {
	struct node *temp = head;
	while(temp != NULL){
		printf("%d ", temp>
				data);
		temp = temp>
			next;
	}
	printf("\n");
}

int main(){
	struct node* head = NULL;
	int n = 2;
	push(&head, 20);
	push(&head, 4);
	push(&head, 15);
	push(&head, 85);
	push(&head, 23);
	printf("\n\nThe list is:");
	printList(head);
	MiddleNode(head);
	reverse(&head);
	printf("\nReversed Linked list \n");
	printList(head);
	swapNthNodes(&head, n);
	printf("\nAfter swap the %d node:",n);
```

```
    printList(head);
}
```

7.3 How to find number of elements in a non-circular list whole Last element is not null-terminated

ANSWER: Maintain 2 pointers : One to point next node and another to point to
next node's next.
When the two coincide on traversing, the end can be found. With a counter,
number of elements
can be found

7.4 WAP to check if the linked list is circular or not?

ANSWER:
1.pointers travelling at different speeds start from the head of the linked list
2.Iterate through a loop
3.If the faster pointer reaches a NULL pointer then return that list is
acyclic and not circular
4.If the faster pointer is ever equal to the slower pointer or the faster
pointer's next pointer is ever equal to the slower pointer
then return that the list is circular
*Advance the slower pointer one node
*Advance the faster pointer by 2 nodes

```
bool findCircular(Node *head){
Node *slower, * faster;
slower = head;
faster = head;
while(true) {
/* if the faster pointer encounters a NULL element*/
if( !faster || !faster>
next)
return false;
/*if faster pointer ever equals slower or faster's next
pointer is ever equal to slow then it's a circular list*/
else if (faster == slower || faster>
next == slower)
return true;
else{
// advance the pointers
slower = slower>
next;
faster = faster>
next>
next;
}
}
}
```

7.5 Given last node element address in a singly linked list, can we traverse it in reverse?

ANSWER:
Not possible

7.6 Write a program to find out the intersection of two nodes?

Ans. USING DIFFERENCE OF NODE COUNT:
1) Get count of the nodes in first list, let count be c1.
2) Get count of the nodes in second list, let count be c2.
3) Get the difference of counts d = abs(c1 – c2)
4) Now traverse the bigger list from the first node till d nodes so that from here onwards both the lists have equal no of nodes.
5) Then we can traverse both the lists in parallel till we come across a common node. (Note that getting a common node is done by comparing the address of the nodes)

```
#include<stdio.h>
#include<stdlib.h>
struct node{
int data;
struct node* next;
};
int getCount(struct node* head);
int findINode(struct node* head1, struct node* head2);
int _getIntesectionNode(int d, struct node* head1, struct node* head2);
int findINode(struct node* head1, struct node* head2)
{
	int c1 = getCount(head1);
	int c2 = getCount(head2);
	int d;
	if(c1 > c2){
		d = c1 c2;
		return _getIntesectionNode(d, head1, head2);
	}
	Else {
		d = c2 c1;
		return _getIntesectionNode(d, head2, head1);
	}
}

int _getIntesectionNode(int d, struct node* head1, struct node* head2){
	int i;
	struct node* current1 = head1;
	struct node* current2 = head2;
	for(i = 0; i < d; i++){
		if(current1 == NULL){
			return 1;
		}
		current1 = current1>
```

```
            next;
    }
    while(current1 != NULL && current2 != NULL){
        if(current1 == current2)
            return current1>
                data;
        current1= current1>
            next;
        current2= current2>
            next;
    }
    return 1;
}


int getCount(struct node* head){
    struct node* current = head;
    int count = 0;
    while (current != NULL){
        count++;
        current = current>
            next;
    }
    return count;
}

int main(){
/* assuming data is populated in both the nodes & we have the head
pointers of both the nodes */
printf("\n The node of intersection is %d \n", findINode(head1, head2));
return 0;
}
```

7.7 Write a program for Stack implementation using linked list?

Ans.

```
#include<stdio.h>
void push();
void pop();
void display();
struct node{
int data;
struct node *next;
}obj;
struct node *head = NULL;
struct node *temp = NULL;
void push(){
```

```
        printf("In push\n");
        int item;
        printf("Enter the item\n");
        scanf("%d",&item);
        temp=(struct node *) malloc (sizeof(struct node));
        temp>
                data=item;
        temp>
                next=head;
        head =temp;
}

void pop(){
        printf("In pop\n");
        if(head==NULL)
                printf("Stack is empty\n");
        else
        {
                temp=head;
                printf("The element deleted = %d\n",temp>
                                data);
                free(temp);
                head=head>
                        next;
        }
}

void display(){
        struct node *save = NULL;
        if(head == NULL)
                printf("stack is empty.\n");
        else{
                save = head;
                printf("DATA IN STACK:\n");
                while(save != NULL){
                        printf("%d & %u\n", save>
                                        data, save>
                                        next);
                        save = save>
                                next;
                }
        }
}

int main() {
push();
```

```
push();
display();
pop();
pop();
display();
return 0;
}
```

7.8 Write a C program to compare two linked lists?

Ans. Approach 1: recursion

A simple C program to accomplish the same.

```
int compare_linked_lists(struct node *q, struct node *r){
static int flag;
if((q==NULL ) && (r==NULL)){
flag=1;
}
else{
if(q==NULL || r==NULL){
flag=0;
}
if(q->data!=r->data){
flag=0;
}
else{
compare_linked_lists(q->link,r->link);
}
}
return(flag);
}
```

Approach 2: compare each node

Another way is to do it on similar lines as strcmp() compares two strings, character by character (here each node is like a character).

8 Miscellaneous

8.1 What is the output of the below code?

CODE:

```
#include <stdio.h>
void main() {
char c='\11';
printf("%d",c);
}
```

ANSWER: 9

The symbol \ will treat it as octal number Compiler will show warning and print ascii value of 8 and 9 for '\8' and '\9' respectively.

8.2 What is the output of the below code?

CODE:

```
#include <stdio.h>
int main(){
int a=10,b;
a>=5 ? b=100 : b=200;
printf ("%d\n",b);
return 0;
}
```

ANSWER: compiler error l
value required as left operand of assignment (for b).
Correct expression is b= a>=5 ? 100 : 200;

8.3 WAP to find largest of 3 numbers without using comparison operators

CODE:

```
#include<stdio.h>
void main(){
int a=33,b=100,c=90;
if(!((ab)& 0x80000000 )){
if(!((ac)& 0x80000000))
printf("a is large");
else
printf("c is large\n");
}
else{
if(!((bc)& 0x80000000))
printf(" b is large\n");
else
printf(" c is large\n");
}
}
```

ANSWER: b is large

8.4 WAP to print semicolon (;) without using it anywhere in the program

ANSWER:

```
#include<stdio.h>
void main(){
if(printf("%c",59)){
}

}
```

8.5 C program to find maximum and minimum value without

using any condition and loop

CODE:
```
#include<stdio.h>
void main(){
int a=1895,b=1570;
printf("max=%d \n min=%d\n",((a+b)+abs(ab))/2, ((a+b)abs(ab))/2);
}
```
ANSWER:
max=1895
min=1570

8.6 Swap 2 variables without using any binary operators (^,-) and condition check

CODE:
```
#include<stdio.h>
int a =36;
int f1(){
return a;
}
void main(){
int a=36,b=12;
printf("a=%d,b=%d\n",a,b);
a=b;
b=f1();
printf("after swapping a=%d,b=%d\n",a,b);
}
```
ANSWER:
a=36,b=12
after swapping
a=12,b=36

8.7 swap 2 nos in a single line logic

ANSWER:
a^=b^=a^=b;

8.8 Swap 1st and 2nd byte of an integer

ANSWER:
x= x<<8 | x>>8;

8.9 What is the output of the below code

CODE:
```
#include<stdio.h>
int main()
{
double num = 5.2;
int var = 5;
```

```
printf("%d\t", sizeof(!num));
printf("%d\t", sizeof(var = 15/2));
printf("%d\n", var);
return 0;
}
```

ANSWER: 4 4 5
Explanation: var = 15/2, gets computed and result is localized only within the sizeof opearator. Outside this printf, var retains its assigned value 5.

8.10 What is the output of the below code

CODE:

```
#include<stdio.h>
int main(){
char a = 250;
int expr;
expr = a + !a + ~a + ++a;
printf("%d\n", expr);
return 0;
}
```

ANSWER: 6
Explanation: 250 is beyond the range of signed char. Its corresponding cyclic value is 6.
!,~ and ++ have equal precedence ans Associativity is from right to left.

8.11 What is the output of the below code

CODE:

```
#include<stdio.h>
int main(){
	int c[] = {2.8, 3.4, 4, 6.7,5};
	int j, *p = c, *q = c;
	for( j = 0; j<5; j++){
		printf("%d\t", *c);
		++q;
	}
	for (j = 0; j < 5; j++){
		printf("%d\t", *p);
		p++;
	}
	return 0;
}

~
```

ANSWER: 2 2 2 2 2 2 3 4 6 5
Explanation: Initially C is assigned to both p and q. In I loop, since only q is incremented not c, 2 prints 5 times. In II loop p itself is incremented.

8.12 What is the output of the below code

CODE:
```
#include<stdio.h>
int main(){
	int i = 1,
	    j= 1,
	    k= 0, l = 2, m;
	m = i++ && j++ && k++ || l++;
	printf("%d\t%d\t%d\t%d\t%d\n", i, j, k, l, m);
}
```
ANSWER: 0 0 1 3 1

Explanation: Logical && operator has higher priority than logical ||
operator.

8.13 What is the output of the below code

CODE:
```
#include<stdio.h>
int main(){
int i = 10;
i = !i>14;
printf("i = %d\n", i);
return 0;
}
```
ANSWER: i = 0;

Explanation:

(NOT)! has more precedence than > symbol. !10 = 0. 0> 14 false hence i =0.

8.14 Find the bug in the below program? Fix the bug.

CODE:
```
void main(){
fun1(1,100);
}
void fun1(int x, int y){
if(x>=y)
return ;
printf("Hello %
d\n", x);
fun1(x++,y);
}
```
ANSWER:

Output is in infinite loop, printing “Hello”.

Fix:
```
void fun1(int x, int y){
if (x >= y)
return;
```

```
fun1(x+1, y);
printf("Hello world\n");
}
```

8.15 How to print "Hello" 100 times without using loop and goto statement?

ANSWER: By using Recursion.

```
void main(){
fun1(1,100);
}
fun1(int x, int y){
if(x>=y)
return ;
printf("Hello\n");
show(x++,y);
}
```

8.16 Does C supports overloading or not?

ANSWER: NO

Overloading: Same name with different signature.

"Variable no of arguments" like printf() function seems overloading but not real. There is only one function and the compiler uses a special calling convention to call it. Whatever arguments you provide are put in sequence on stack[*]. printf examines the format string and uses that to work out how to read those arguments b ack. That’s why printf isn't typesafe.

8.17 What is the output of the below code?

CODE:

```
int sum(int x,int y,int z){
return x+y+z;
}
void main() {
int x=5,y=10,z=15,val1,val2,val3,val4,val5,val6;
val1 = sum(((z*x),y),z=0,(x1));
val2 = sum((x=0,y),(z=0),(x,y=0));
val3 = sum(((z*x),y,z*y),(x,y=0),25);
val4 = sum((x=0,y=0),(z=0),(x,y1));
val5 = sum((x=0,y==0),(z=0),(x,y=0));
val6 = sum((x=0,y==0),(z=0),(x,y1));
printf("%d\t %d \t %d\t %d \t %d \t%d\n",val1,val2,val3,val4,val5,val6);
}
```

ANSWER:

14 0 25 1

1 0

8.18 If y=10, what is the output of

· printf("%d,%d"+1,y,y);

· printf("%d\t%d\n"+3,y,y);
· printf("%d,%d"+4,y,y);
ANSWER:
a. d,10
b. 10
c. d

8.19 What is the output of the below code?

CODE:

```
int a= 0;int b = 20;char x = 1;
char y =10;
if(a,b,x,y)
printf("a,b,x,y hello\n"); /* as Y is non zero value this executes
if(10) as y = 10*/
if(a,b,x,a)
printf("a,b,x,a hello\n"); /* no output because while execution
condition takes if(0) as a = 0 */
if(a,b,x,x)
printf("a,b,x,x hello\n"); /* as x is non zero value this executes
if(1)
as x = 1*/
```

ANSWER:
from lhs to rhs, right most value will be considered by the if statement
because from stack out parameter is so.., the output is as follows
a,b,x,y hello
a,b,x,x hello

8.20 What is the output of the below code?

CODE:

```
#define AB 50
main(){
printf("%d\n",AB);
#define AB 20
func();
printf("%d\n",AB);
}
void func(){
printf("%d\n",AB);
}
```

ANSWER:
Macro redefining is allowed and the scope of the macro is always global to
that particular file hence the output is
50
20
20

8.21 How can we use STATIC function outside a file scope?

ANSWER: Using functions pointer it is possible. You can declare the pointer to a static function in the same file, in which static function is declared and use the pointer in another file.

8.22 WAP to find if there is an overflow or underflow occured in addition?

ANSWER:

```
#include <stdio.h>
int main() {
	unsigned long a,b,sum;
	printf("enter two unsigned long numbers\n");
	scanf("%lu %lu",&a,&b);
	sum = a+b;
	printf("a = %lu\n",a);
	printf("b = %lu\n",b);
	printf("sum = %lu\n ",sum);{
		if(sum < a || sum < b){
			printf("Overflow detected\n");
		}
		else{
			printf("No overflow detected\n");
		}
	}
}
```

8.23 What is the output of the below program?

CODE:

```
#include<stdio.h>
void main(){
int i ;
char ch;
scanf("%d",&i); //if 1 = 2
scanf("%c",&ch); //if c = b
printf("%d %c",i,ch);
}
```

ANSWER:
It prints only 2, it doesn't print 2b
The reason behind this is , after scanning the integer(we press enter) it also takes the enter keyword as a character and stores it in c.
To avoid this we can modify the scanf statement as:

```
scanf("%d\n",&i); //if 1 = 2
scanf("%c",&ch); //if c = b
```

Now it prints the output 2b.

8.24 What is the output?

CODE:

```
char a = 'A';
sizeof(a) = ?
sizeof('A') = ?
```

ANSWER: sizeof(a) = 1
sizeof('A') = 4
size of a is 1 byte because it is a char, where as 'A' has 4 bytes because by default
other than decimal numbers, 4 bytes will be allocated if it is not type casted.

8.25 What is the output of the below code?

CODE:

```
fun(int,int);
main(){
int a=1,b=2,c=0;
c = fun(a,b);
printf("%d",c);
}
fun(int a, int b){
int d;
d=a+b;
}
```

ANSWER: c = 3
this program will not flag an error, because by default the return type of a function is int, and all the functions will return the value to the calling function through accumulator, so when we are doing a arithmatic operations, accumulator will be used, and the last value stored in the accumulator will be send to the calling function. in our case its a+b = 3, so 3 will be stored in acc and will be received by the calling function.

8.26 Can we assign an auto variable value to static variable?

ANSWER: Erroneous as assignment to static variable from auto variable at runtime is not possible. Static variables are to be initialized with constants."error: initializer element is not constant"

8.27 .What will be the output of the program?

```
void main(){
printf("%d');
}
```

Ans. When we use %d the compiler internally uses it to access the argument in the stack (argument stack). Ideally compiler determines the offset of the data variable depending on the format specification string. Now when only %d will be present in the printf then compiler will calculate the correct offset (which will be the offset to access the integer variable) but as the actual data object is to be printed is not present at that memory location so it will print what ever will be the contents of that memory location.

8.28 Write a one line C function to round floating point numbers?

Ans.

```
# include<stdio.h>
int roundNo(float num){
return num < 0 ? num 0.5
: num + 0.5;
}
int main(){
printf("%d\n", roundNo(1.000));
return 0;
}
```

8.29 How will you print numbers from 1 to 100 without using loop?

Ans. Using recursion.

```
#include<stdio.h>
void printNo(unsigned int n){
if(n > 0){
printNo(n1);
printf("%d ", n);
}
return;
}
void main(){
        printNo(100);
}
```

8.30 How do you determine if a point is
1) inside a triangle
2) outside

Ans. we can check by computing the area of $\triangle ABC$, $\triangle ABP$, $\triangle BC P$, and $\triangle AC P$.
a. Point is inside the triagle If $k \triangle ABC = k \triangle ABP + k \triangle BC P + k \triangle AC P$, the
b. Point is outside the triagle If $k\triangle ABC < k\triangle ABP + k\triangle BCP + k\triangle ACP$, the
c. Point lies on the edge if (k ABP||k △ △BCP||k△ACP) = 0), t)
dthdhgtdhgftjghfgfg
△ACP)

8.31 Will this program compile?. If yes, what is the o/p?

```
static int i;
static int i =5;
static in i;
int main() {
printf("%d\n", i);
}
```

Ans: It will compile.

Static variables can be declared many times, but should be initialized only once.

8.32 Two files File1 and File2 1) if both has static int i what will happen? 2) If both contains int i what will happen?

ans. If we declare static int i in both the files. It will not show any error because static is a file scope.
If we declare int i globally in both the files it will give an compilation
error because both declarations will try to access the same memory location.

8.33 How can we find whether the value is 2 power n or not?

ans:

```
int main(){
int n;
printf("Enter the value\n");
scanf("%d",&n);
if((n&(n1))==0)
printf("Given value is 2 power value\n");
else
printf("Given value is not 2 power value\n");
return 0;
}
```

8.34 How to avoid multiple inclusions of same header file in C.

Ans : using ifndef macro ,

```
example.h
#ifndef EXAMPLE_H
#define EXAMPLE_H
'
'
#endif
```

Q138.Difference between printf vs sprintf vs fprintf
printf : Writes to standerd input output
sprintf : writes to buffer specified by the user.
fprintf : writes to file

8.35 Why do we need #pragma pack(x) ? use-case?

structures that hold network packets and headers are usually packed so that
no memory is wasted because of structure padding. This inturn
saves network
bandwidth while transmitting such packets over network

8.36 To add 2 positive integers without using arithmetic operators

```
#include<stdio.h>
void main(){
```

```
int a=13,b=5;
while(a)
b++;
printf(" sum=%d\n",b);
}
```

8.37 when int i=5; What is the value of 'i' in below cases

a. i++ + ++i 12
b. i + ++i 12
c. i+++++i error:
lvalue required as increment operand
d. ++i + ++i 14
e. i + ++i 10
f. i++ + I 8
g. ++i + i10

8.38 What is the o/p of below program.

```
void main(){
printf("\rab");
printf("\nhat");
printf("\bi");
}
```

Output : ab
Hai

8.39 What is the o/p of below program?

```
main(){
int i;
char ch;
scanf("%d",&i);
scanf("%c",&ch);
printf("%d%c",i,ch);
```

Answer:

8.40 What is the o/p of below program?

```
void main(){
printf("%s","%c" "good" " boy");
}
```

Answer: %cgood boy

8.41 What is the o/p of below program?

```
void calculate(){
int x=3, y=3, z=1;
printf("%d\n",z+=x<y ? 10:20 );
}
```

Ans: 21

8.42 .What is the o/p of below program?

Which is valid

```
int main() {
int a=2,b;
1) a>=5 ? b=100 : b=200;
2) a>=5 ? b=100 :(b=200);
3) b=a>=5 ? 100:200;
printf ("%d\n",b);
return 0;
}
```

Answer : 2) & 3) are right

8.43 What is the o/p of below program?

```
void start();
void end();
#pragma startup start
#pragma exit end
int static i;
void main ( ) {
printf("\n Main Function : %d",++i);
}
void start( ) {
printf("\n Start Function : %d",++i);
}
void end( ) {
printf("\n End Function : %d",++i);
}
```

Answer : 1 2 3

8.44 .Write your own program to convert lowercase letter to uppercase letter?

```
#include <stdio.h>
#define to(a) (((a)>= 'a' && (a)<='z')? (a(' a'' A')): a)
main(){
char ch,ch2;
printf("Enter a character:");
scanf("%c",&ch);
ch2=to(ch);
printf("\nOuput character:%c\n",ch2);
}
```

Output:
Enter a character:a
Ouput character:A

8.45 .What is the o/p of the below program?

```
#include <stdio.h>
main(){
int c = 0;
printf("%d\n", main || c); //line 5
printf("%d",main);
}
```

Output:1
Explanation: In this program main returns address i.e., some +ve value. Eventhought
c has zero,main has value. So it will print 1
In place of line 5 if you give printf("%d\n", main && c); then output will be 0

8.46 .what is the o/p of the below program?

```
#include <stdio.h>
void main (){
int x = 10;
printf ("x = %d, y = %d", x,x++);
}
```

Output: Compiler Error
Explanation: First it will decrement the 'x' value then it will become value. postincrement
of value is not possible. postincrement
of variable
is possible.

8.47 What is the output of the following?

```
int main(void){
char i = 3;
int j;
k=i+i;
j = sizeof(++i + ++i);
printf("i=%d j=%d\n", i, j);
return 0;
}
```

Ans:i=3 j=4

8.48 What is the output of the following?

```
#include<stdio.h>
void main(){
static int i=5;
if(i){
main();
printf("%d\t",i);
```

```
}
}
```

Answer: 0 0 0 0

8.49 What is the output of the following?

```
void fun(int);
int main(int argc){
printf("%d ", argc);
fun(argc);
return 0;
}
void fun(int i){
if(i!=4)
main(++i);
}
```

Ans : 1 2 3 4

8.50 What is the output of the following?

```
#include<stdio.h>
int main()
{ int i=10;
static int x=i;
printf("%d\n",x);
}
```

Compile error: initializer element is not constant
Reason : static variables can not be initialized dynamically, using another Variable

---

8.51 Why n++ executes faster than n+1 ?

Ans:The expression n++ requires a single machine instruction such as INR to carry out the increment operation whereas, n+1 requires more instructions to carry out this operation.

8.52 Explain various GCC compiler options for compiling the source code:

Ans : Suppose if we have main.c file,below are few gcc options.
a)to produce executable of desired name other that a.out :gcc main.c o main
b)Enable all warnings set through Wall
option **:gcc Wall main.c o main**
c)Produce only the preprocessor output with E
option :gcc E main.c > main.i (this will redirect preprocessed output to main.i file)
d) Produce only the assembly code using S
option :gcc S main.c > main.s (this will redirect Assembly output to main.s file)
e)To produce only the compiled code (without any linking), use the C
option :gcc C main.c

f)Produce all the intermediate files using savetemps
function :gcc savetemps main.c
g)Link with shared libraries using l
option **:gcc Wall main.c o main lhello**
The gcc command mentioned above links the code main.c with the shared library libhello.so to produce the final executable ‘main’.
h)For example, the following commands create a shared library libhello.so from source file hello.c:
gcc c Wall Werror fPIC hello.c
gcc shared o libhello.so hello.o
While creating the shared libraries, position independent code should be produced. This helps the shared library to get loaded as any address instead of some fixed address. For this fPIC
option is used.
i)Interpret char as unsigned char using funsignedchar
option
Here is an example :

```
#include<stdio.h>
int main(void){
 char c=10;
printf("%d\n", c);
return 0;
}
```

gcc Wall funsignedchar
main.c o main
$ ./main
Ans : 246 (2's compilment of 10)
similarly to interpret unsigned char as signed char we can use gcc Wall
fsignedchar
main.c o
main
j)Use compile time macros using D
option
The compiler option D can be used to define compile time macros in code.
Here is an example :

```
#include<stdio.h>
int main(void){
#ifdef MY_MACRO
printf("\n Macro defined \n");
#endif
}
```

The compiler option D
can be used to define the macro MY_MACRO from command
line.
$gcc Wall DMY_MACRO main.c o main
k)convert warnings into errors with Werror

option.
Through this option, any warning that gcc could report gets converted into error.

gcc Wall Werror main.c o main

l)Provide gcc options through a file using @ option
The options to gcc can also be provided through a file. This can be done using the @ option followed by the file name containing the options. More than one options are separated by a white space.
Here is an example :
$ vim opt_file
Wall o main
The opt_file contains above the options.
Now compile the code by providing opt_file along with option @.
gcc main.c @opt_file

8.53 What is the output of the following?

```
#include<stdio.h>
main(){
int i=0;
int j=1;
if(1)
i=j,j+=1,printf("%d %d\n",i,j);// multiple statements ,all are executed
else
printf("Else\n");
}
```

Ans: 1 2

8.54 What is the output of the following?

```
# define scanf "%s Geeks For Geeks "
main(){
printf(scanf, scanf);
getchar();
return 0;
}
```

Output: %s Geeks For Geeks Geeks For Geeks

8.55 Can you write two functions in which one executes before main function and other executes after the main function?

```
#include<stdio.h>
void india();
void usa() ;
#pragma startup india 105
#pragma startup usa
#pragma exit usa
#pragma exit india 105
int main(){
```

```
printf("\nI am in main");
return 0;
}
void india(){
printf("\nI am in india");
}
void usa(){
printf("\nI am in USA");
}
```

8.56 What is the o/p of the below program if the machine is little endian?

```
int main(){
union abc{
int i;
char c;
};
union abc ABC;
ABC.i = 256;
ABC.c = '2';
printf("%d",ABC.i);
return 0;
} Ans: 306
```

8.57 What is the o/p of the below program?

```
int main(){
int i=10;
static int x=i;
if(x==i)
printf("Both are equal");
else
printf("Both are not equal");
return 0;
}
```

Ans: compiler error (initializer element is not constant)

8.58 What is the o/p of the below program?

```
int main(){
int num, a=10;
num = aa;
printf("%d, %d\n",num,a);
return 0;
}
```

Ans: 20,8

8.59 What is the o/p of the below program?

```
int main(){
int x=5, y=10, z=15;
printf("%d %d %d");
return 0;
}
```

Ans: garbage

8.60 What is the o/p of the below program?

```
int main(){
int a=4;
a= a++ * a++ * a++ / a++;
printf("%d", a);
return 0;
}
```

Ans: 20

8.61 What is the o/p of the below program?

```
int main(){
int a=20,
b=3;
printf("%d", a % b);
return 0;
}
```

Ans: 2

8.62 What is the o/p of the below program?

```
int main(){
char c='a';
printf("%d %d", sizeof(c), sizeof('a');
return 0;
}
```

Ans: 1 4

8.63 What do lvalue and rvalue mean?

An lvalue is an expression that could appear on the lefthand sign of an assignment (An object that has a location). An rvalue is any expression that has a value (and that can appear on the righthand Sign of an assignment).

The lvalue refers to the lefthand side of an assignment expression. It must always evaluate to a memory location. The rvalue represents the righthand side of an assignment expression; it may have any meaningful combination of variables and constants.

Usecase: Understanding and fixing warnings like “lvalue required as left operand of assignment” warning on below code: if (strcmp("hello", "hello") = 0)

8.64 What are Trigraph characters?

These are used when you keyboard does not support some special characters.

Some are:
· ??= is #
· ??/ is \
· ??(is [
· ??) is]
Eg1: /??/
* A comment *??/
/
is equivalent to
/\
* A comment *\
/
Eg2:
printk("imm: bad interrupt (???)\n");
A strict ISO C compiler must print
imm: bad interrupt (?] Since ??) is the trigraph for]. Many C compilers other than gcc will just print that and never tell you there's a problem.
By default, gcc disables trigraphs, but if you give the ansi flag they are enabled. But it warns about inputs that give different behavior with and without trigraphs.
Gcc4.4 compiler: “warning: trigraph ??/ ignored, use trigraphs to enable” Digraph Equivalent
<: [
:>]
%: #

8.65 Whats the prototype of main()? Can main() return a structure?

The right declaration of main() is
1. int main(void)
2. int main(int argc, char *argv[])
3. int main(int argc, char *argv[], char *env[]) //Compiler dependent, nonstandard C.
In C, main() cannot return anything other than an int.Something like void main() is illegal. There are only three valid return values from main() · 0
· EXIT_SUCCESS
· EXIT_FAILURE
The latter two are defined in <stdlib.h>.

8.66 How to get the list of all environment variables in C?

· Usage of char *env[] in argument of main:

```
int main(int argc, char **argv, char** envp){
char** env;
for (env = envp; *env != 0; env++){
        char* thisEnv = *env;
        printf("%s\n", thisEnv);
        }
return 0;
}
```

· **Using global variable ' environ'**

```
#include<stdio.h>
extern char **environ;
int main() {
int i = 0;
char *s = *environ;
for (; s; i++) {
        printf("%s\n", s);
        s = *(environ+i);
        }
        return 0;
}
```

8.67 Variable Arguments list in C-programming.

Refer: http://www.tutorialspoint.com/cprogramming/c_variable_arguments.htm

8.68 What is the o/p of below program?

```
int main(){
int a,b,c,d;
a=3;
b=5;
c=a,b;
d=(a,b);
printf("c=%d",c);
printf("d=%d",d);
return 0;
}
```

o/p:35

8.69 What is the o/p of below program?

```
enum tag={left=10,right,front=100,back};
printf("%d %d %d %d",left,right,front,back);
```

o/p:10 11 100 101

-

8.70 Can main() be called recursively?

Ans:

```
main() {
main();
}
```

But this will go on till a point where you get a Runtime error : Stack overflow.

8.71 What is the difference between a deep copy and a shallow copy?

Deep copy involves using the contents of one object to create another instance of the same structure. In a deep copy, the two objects may contain ht same information but the target object will have its own buffers and resources. The destruction of either object will not affect the remaining object.
Shallow copy involves copying the contents of one object into another instance of the same structure thus creating a mirror image. Owing to straight copying of references and pointers, the two objects will share the same externally contained contents of the other object to be unpredictable. This method of copying is called shallow copy. If the object is a simple structure, comprised of built in types and no pointers this would be acceptable. This function would use the values and the objects and its behavior would not be altered with a shallow copy, only the addresses of pointers that are members are copied and not the value the address is pointing to. The data values of the object would then be inadvertently altered by the function. When the function goes out of scope, the copy of the object with all its data is popped off the stack.
If the object has any pointers a deep copy needs to be executed. With the deep copy of an object, memory is allocated for the object in free store and the elements pointed to are copied. A deep copy is used for objects that are returned from a function.
Example:

```
struct sample {
char * ptr;
}
void shallowcpy(sample & dest, sample & src) {
dest.ptr=src.ptr;
}
void deepcpy(sample & dest, sample & src) {
dest.ptr=malloc(strlen(src.ptr)+1);
memcpy(dest.ptr,src.ptr);
}
```

8.72 How to declare an array of N pointers to functions returning pointers to functions returning pointers to characters?

Ans:
char *(*(*a[N])())();
Note that the below is incorrect!
(char*(*fun_ptr)())(*func[n])();
Refer 1. 5.12 Complicated Declarations of "The C programming Language",By Brian W. Kernighan and Dennis M. Ritchie.
2. http://www.geeksforgeeks.org/complicateddeclarationsinc /

8.73 Integer overflow?

Ans: Integer overflow is the canonical example of "undefined behaviour" in C (noting that operations on unsigned integers never overflow, they are defined to wraparound instead). This means that once you've executed x + y, if it overflowed, you're already hosed. It's too late to do any checking your program could have crashed already. Think of it like checking for division by zero if you wait until after the division has been executed to check, it's already too late. So this implies that method (1) is the only correct way to do it. For max, you can use INT_MAX from <limits.h>.
If x and/or y can be negative, then things are harder you need to do the test in such a way that the test itself can't cause overflow.
if ((y > 0 && x > INT_MAX y)

```
||
(y < 0 && x < INT_MIN y)) {
/* Oh no, overflow */
}
Else {
sum = x + y;
}
```

8.74 What are inline functions?

Ans: inline functions are nonstandard C. They are provided as compiler extensions. But, nevertheless, one should know what they are used for. The inline comment is a request to the compiler to copy the code into the object at every place the function is called. That is, the function is expanded at each point of call. Most of the advantage of inline functions comes from avoiding the overhead of calling an actual function. Such overhead includes saving registers, setting up stack frames, and so on. But with large functions the overhead becomes less important. Inlining tends to blow up the size of code, because the function is expanded at each point of call.

```
int myfunc(int a) {
...
}
inline int myfunc(int a) {
...
}
```

Inlined functions are not the fastest, but they are the kind of better than macros (which people use normally to write small functions).

```
#define myfunc(a) \
{ \
... \
}
```

The problem with macros is that the code is literally copied into
the location it was called from. So if the user passes a "double" instead of
an "int" then problems could occur. However, if this senerio happens with an
inline function the compiler will complain about incompatible types.
This will save you debugging time stage.
Good time to use inline functions is
1. There is a time criticle function.
2. That is called often.
3. Its small. Remember that inline functions take more space
than normal functions.
Some typical reasons why inlining is sometimes not done are:
1. The function calls itself, that is, is recursive.
2. The function contains loops such as for(;;) or while().
3. The function size is too large.

8.75 converting from one Endian to another?

Ans:
Approach 1:

```
int myreversefunc(int num){
```

```
int byte0, byte1, byte2, byte3;
byte0 = (num & x000000FF) >> 0 ;
byte1 = (num & x0000FF00) >> 8 ;
byte2 = (num & x00FF0000) >> 16 ;
byte3 = (num & xFF000000) >> 24 ;
return((byte0 << 24) | (byte1 << 16) | (byte2 << 8) | (byte3 << 0));
}
```

Approach 2: Another approach is swap the bytes. In a 4 byte scenario, byte1 with byte4 and byte3 with byte2

8.76 Write your own atoi() function in a optimized way

```
#include<stdio.h>
int myatoi(const char *string);
int main(){
printf("\n%d\n", myatoi("1998"));
getch();
return(0);
}
int myatoi(const char *string){
int i;
i=0;
while(*string){
i=(i<<3) + (i<<1) + (*string - '0');
string++;
}
return(i);
}
```

8.77 What is the o/p of below program?

```
int call(int);
int main(){
static main;
int x;
x = call(main);
printf("%d", x);
return 0;
}
int call(int address){
address++;
return address;
}
```

o/p = 1

8.78 What is the o/p of below program?

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

```
int main(){
int a=-20;
int b=-3;
printf("%d",a%b);
return 0;
}
```

Result -2
Explanation:
Sign of resultant of modular division depends upon only the sign of first operand.

8.79 What is the o/p of below program?

```
#include<stdio.h>
int main(){
float f=3.4e39;
printf("%f",f);
return 0;
}
```

Result INF
Explanation:
If you will assign value beyond the range of float data type to the float variable it will not show any compiler error. It will store infinity.

8.80 What is the o/p of below program?

```
main(){
printf ("%d", 10 ? 0 ? 5 : 1 : 12); will print?
}
```

Result 1
Explanation:
Trinary operator is right associative.

8.81 What is the o/p of below program?

```
#include<stdio.h>
int main(){
char *url="c:\tc\bin\rw.c";
printf("%s",url);
return 0;
}
```

Result w.c in
Explanation:
\t is tab character which moves the cursor right after spaces.
\b is back space character which moves the cursor one space back.
\r is carriage return character which moves the cursor beginning of the line

8.82 WAP to print itself?

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

```
int main(){
FILE *fp;
char c;
fp = fopen(__FILE__,"r");
printf("__FILE__ %s\n",__FILE__);
do{
c= getc(fp);
// putchar(c);
printf("%c",c);
}
while(c!=EOF);
fclose(fp);
return 0;
}
```

8.83 Difference between Declaration and Definition :

Declaration: A variable is declared when the compiler is informed that a variable exists (and this is its type); it does not allocate the storage for the variable at that point.
Definition: A variable is defined when the compiler allocates the storage for the variable

- Multiple declarations is possible but definition can happen only once
- Declaration will notify compiler that variable is present where as definition allocates memory for it
- All Definitions are declarations but vice versa is not true

Declaration will tell compiler that a variable exists and it is of some type(int, float so on) but definition will allocate memory for it. There can be multiple declarations. Say

```
extern int var; //declaration , here space is not allocated for 'var'
int var; //definition
```

Declaration tells compiler "This variable will exist, at some point, under this name, so you can use it." The definition tells the compiler to actually arrange for the variable to be created. So Definition can only happen once.

```
int i; // definition
class student; //declaration
class student{
public : int marks;
}; //definition
```

8.84 Volatile keyword in C :

- volatile tells the compiler "the value here might be changed by something external to this program". Volatile is useful mainly when working on hardware registers, that often change "on their own", or when passing data to/from interrupts.
- If you have a variable that could have its contents changed at any time, usually due to another thread acting on it while you are possibly referencing this variable in a main thread, then you may wish to mark it as volatile
- By using volatile we can force it to read always from memory. so that we can get updated result.

8.85 Static and extern storage classes :

• If, within a translation unit, the same identifier appears with both internal and external linkage, the behavior is undefined. So its better not to declare the same variable with both static and extern in the same translation unit. This leads to undefined behavior.

• static at a global level restricts its visibility only to the current translation unit, whereas extern dictates that it is visible across different translation units.

8.86 Simple Puzzle :

There are 3 incandescent bulbs with 3 switches to operate them. But switches and bulbs are placed in different rooms. So you cannot see bulbs when you turn ON switch. Can you find out which switch turns ON which bulb ?? with a condition that you can go only 1 time to the room where bulbs are present , one and only one time ??

Ans :: If we turn on incandescent bulb for some time , it will get hotter. This can be applied to solve above puzzle. ON 1 Switch, corresponding bulb will get ON in diff room, turn it OFF after 10 mins. Now ON another switch, go to room where bulbs are present. Well, the light bulb corresponding to the first switch will still be warm (even though it’s off), the bulb corresponding to the second switch will be ON, and the bulb corresponding to the last switch will be OFF.

8.87 Commandline Arguements:

Command line arguments came into existence to implement commands such as ls,cat,cp,find,grep and many more....

orgaanization of command line arguments in memory:

$echo hi how r u...?

argv:

“echo\0”

“hi\0”

“how\0”

“r\0”

“u...?\0”

(void*0)

All are stored in stack segment(argv,array of pointers to string and null terminated string).

8.87.1 echo command implementation.

```
#include<stdio.h>
int main(int argc,char*argv[]){
int i;
for(i=1;i<argc;i++)
printf(“%s%s“,argv[i],(i<argc1)?” “:””);
printf(“\n”);
return 0;
}
```

8.87.2 implementation of cat command:

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

```
void filecopy(FILE*,FILE*);
int main(int argc,char* argv[]){
if(argc==1)
filecopy(stdin,stdout);
else {
argv++;
while(*argv!=0) {
filecopy(fopen(*argv,"r"),stdout);
argv++;
}
}
return 0;
}
void filecopy(FILE* fin,FILE* fop) {
int c;
while((c=fgetc(fin))!=EOF) {
fputc(c,fop);
}
}
```

8.88 complicated declarations: >

char **argv ; argv: pointer pointer to char.

- int (*daytab)[13]; daytab: pointer to array[13] of int.
- int *daytab[13] ; daytab: array[13] of pointer to int.
- void *comp() ; comp: function returning pointer to void.
- void (*comp)(); comp: pointer to function returning void.
- char (*(*x())[])(); x: function returning pointer to array[] of pointer to function

returning char.

->char (*(*x[3])())[5] x: array[3] of pointer to function returning pointer to array[5] of char.

8.88.1 Example 1:implementation of char (*(*x())[2])();

x: is function receiving nothing but returning pointer to an array of 2 pointers to function where each function recieves nothing and returns character.

```
#include<stdio.h>
char (*(*x())[2])();
char f1();
char f2();
char f3();
char f4();
char f1(){
return 'a';
}
char f2(){
return 'b';
}
char f3(){
```

```
return 'c';
}
char f4(){
return 'd';
}
int main(){
int i;
char (*(*p)[2])();
p=x();
for(i=0;i<4;i++)
printf("%c\n",(*p)[i]());
return 0;
}
char (*(*x())[2])(){
static char(*arr[2][2])();
arr[0][0]=f1;
arr[0][1]=f2;
arr[1][0]=f3;
arr[1][1]=f4;
return arr; }
```

8.88.2 Example 2:implementation of char (*(*x[3])())[2];

x: is an arry of three pointers to function where each function recieving nothing but returning pointer to array of 2 characters.

```
#include<stdio.h>
char (*f1())[2];
char (*f2())[2];
char (*f3())[2];
char (*f1())[2]
{
static char a[2][2]={
{'a','b'},
{'c','d'}
};
return a;
}
char (*f2())[2]
{
static char b[2][2]={
{'e','f'},
{'g','h'}
};
return b;
}
char (*f3())[2]{
static char c[2][2]={
```

```
{'i','j'},
{'k','l'}
};
return c;
}
int main() {
char (*(*x[3])())[2];
char (*p)[2];
int i,j,k;
x[0]=f1;
x[1]=f2;
x[2]=f3;
for(i=0;i<3;i++) {
p=x[i]();
for(j=0;j<2;j++){
	for(k=0;k<2;k++)
	printf("%c\t",p[j][k]);
	printf("\n");
	}
 }
return 0;
}
```

8.88.3 Example 3:implementation of int(*fun())[5];

fun: is function recieving nothing but returning pointer to 1D array of 5 integers.

```
#include<stdio.h>
int(*fun())[5];
int main() {
int (*p)[5];
int i,j;
p=fun();
for(i=0;i<2;i++) {
for(j=0;j<5;j++)
printf("%d\t",p[i][j]);
printf("\n");
}
return 0;
}
int (*fun())[5]
{
static int a[2][5]={
				{1,2,3,4,5},
				{6,7,8,9,10}
			};
return a;
}
```

9 Appendix

9.1 General Topics

9.1.1 Utilities of static keyword

· A static variable inside a function keeps its value between invocations.
· A static global variable or a function is "seen" only in the file it's declared in
· In function argument to specify minimum number of array elements accepted
void fun(int some_array[static 7]);
The standard says in 6.7.6.3:
A declaration of a parameter as ‘‘array of type’’ shall be adjusted to ‘‘qualified pointer to type’’, where the type qualifiers (if any) are those specified within the [and] of the array type derivation. If **the keyword static also appears within the [and] of the array type derivation**, then for each call to the function, the value of the corresponding actual argument shall provide access to the first element of an array with at least as many elements as specified by the size expression.
It's a feature introduced in C99. So : some_array must be at least 7 elements long.

9.1.2 typedef an array

typedef int array [][3];
This means that array is an int array (of as-yet unspecified length) of length 3 arrays.
To use it, you need to specify the length. You can do this by using an initialiser like so:
array A = {{1,2,3,},{4,5,6}}; // A now has the dimensions [2][3]
But you CAN'T say:
array A;
// these are all the same
void foo(array A);
void foo(int A[][3]);
void foo(int (*A)[3]); // this is the one the compiler will see

---

9.1.3 Does the compiler throw an error if the code contains #pragma not supported by it?

Eg: Compiling code with ARM Compiler based #pragma in gcc
Ans: No, Compilers are designed to ignore #pragma not supported by them

---

9.1.4 volatile keyword with static, const, register

register volatile int T=10;
volatile qualifier means that the compiler cannot apply optimizations or reorder access to T, While register is a hint to the compiler that T will be heavily used. If address of T is taken, the hint is simply ignored by the compiler. Note that register is deprecated but still used.
Practical Usage:
Never felt the need for it and can't really think of any right now.
const volatile int T=10;
const qualifier means that the T cannot be modified through code. If you attempt to do so the compiler will provide a diagnostic. volatile still means the same as in case 1. The

compiler cannot optimize or reorder access to T.
unsigned char const volatile *hd_addr
Practical Usage:
Accessing shared memory in read-only mode.
Accessing hardware registers in read-only mode.
static volatile int T=10;
static storage qualifier gives T static storage duration (C++11 §3.7) and internal linkage, while volatile still governs the optimization and reordering.
Practical Usage:
Same as volatile except that you need the object to have static storage duration and to be inaccessible from other translation units.
5. Bubblesort
http://en.wikipedia.org/wiki/Bubblesort
Example:
Let us take the array of numbers "5 1 4 2 8", and sort the array from lowest number to greatest number using bubble sort. In each step, elements written in **bold** are being compared. Three passes will be required.
(5 1 4 2 8) \to (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.
(1 5 4 2 8) \to (1 4 5 2 8), Swap since 5 > 4
(1 4 5 2 8) \to (1 4 2 5 8), Swap since 5 > 2
(1 4 2 5 8) \to (1 4 2 5 8), Now, since these elements are already in order (8 > 5), algorithm does not swap them.
Second Pass:
(1 4 2 5 8) \to (1 4 2 5 8)
(1 4 2 5 8) \to (1 2 4 5 8), Swap since 4 > 2
(1 2 4 5 8) \to (1 2 4 5 8)
(1 2 4 5 8) \to (1 2 4 5 8)
Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one whole pass without any swap to know it is sorted.
Third Pass:
(1 2 4 5 8) \to (1 2 4 5 8)
(1 2 4 5 8) \to (1 2 4 5 8)
(1 2 4 5 8) \to (1 2 4 5 8)
(1 2 4 5 8) \to (1 2 4 5 8)

```
/***********************************************
* bubble.c
***********************************************/
#include <stdio.h>
#include <stdlib.h>
#define T 1
#define F 0
int n;
int * get_input() ;
int bubblesort_ascend(int * input) ;
int bubblesort_descend(int * input) ;
```

```
void display(int * input) ;
int main() {
	int *sort;
	printf("main BEFORE=== --- %d <- %p-- num %d\n", *sort, sort, n);
	sort = get_input();
	printf("main === --- %d <- %p-- num %d\n", *sort, sort, n);
	printf("\n Integers before sorting: \n");
	display(sort);
	bubblesort_ascend(sort );
	bubblesort_descend(sort );
	printf("\n Integers after sorting: \n");
	display(sort);
	return 0;
}

int bubblesort_ascend(int * input) {
	int i, j, temp;
	char swap;
	printf("********************** In ascending order *************************\n");
	for(i=0;i< n-1;i++){
		swap = F;
		for(j=0;j<n-i-1;j++){
			printf("Pass%d Iter%d -> ",i,j);
			display(input);
			if(*(input+j)> *(input+j+1)){
				temp = *(input+j);
				*(input+j) = *(input+j+1);
				*(input+j+1) = temp;
				swap = T;
			}
		}
		if ( swap == F)
			break;
	}
	printf("********************************************************************\n");
	return 0;
}

int bubblesort_descend(int * input) {
	int i, j, temp;
	char swap;
	printf("********************** In descending order *************************\n");
	/* This loop is for number of passes required for complete sorting.
	 * Max passes = Number of input numbers
	 */
	for(i=0;i< n-1;i++){
```

```
        /* swap: This is to check if a target number is swapped during the current pass.
         * If it is not swapped, it means the list is sorted and the function
         returns.However even after the list is sorted ,
         * an extra pass is required to confirm the same.
         */
        swap = F;
        /* This loop is for shifts/swaps each number undergoes during its comparison with neighbours.
         * Max shifts depends on current position of the number in its turn  After each iteration, a target
number will reach its target postion * So, in the next pass the number of iterations depends on number of
input numbers still un-sorted
         */
        for(j=0;j<n-i-1;j++){
            printf("Pass%d Iter%d -> ",i,j);
            display(input);
            if(*(input+j)< *(input+j+1)){
                temp = *(input+j);
                *(input+j) = *(input+j+1);
                *(input+j+1) = temp;
                swap = T;
            }
        }
        if ( swap == F)
            break;
    }
    printf("**************************************************************\n");
    return 0;
}

/**/
int * get_input() {
    int i;
    int *input;
    printf("\n\n Enter integer value for total no.s of elements to be sorted: ");
    scanf("%d",&n);
    if ((input = (int *) calloc(n, sizeof(int))) == NULL)
        return 0;
    for( i=0;i<n;i++) {
        printf("\n\n Enter integer value for element no.%d : ",i+1);
        scanf("%d",input+i);
        printf("--- %d <- %p--\n", *(input+i), input+i);
    }
    return input;
}

void display(int * sort) {
    int i;
```

```
    for( i=0;i<n;i++) {
        printf("%5d ",*(sort+i));
    }
    printf("\n");
}

/*
Sample Output:
main BEFORE=== --- 1416572 <- 0xb7776ff4-- num 0
Enter integer value for total no.s of elements to be sorted: 5
Enter integer value for element no.1 : 23
--- 23 <- 0x8cab008--
Enter integer value for element no.2 : 3
--- 3 <- 0x8cab00c--
Enter integer value for element no.3 : 45
--- 45 <- 0x8cab010--
Enter integer value for element no.4 : 75
--- 75 <- 0x8cab014--
Enter integer value for element no.5 : 54
--- 54 <- 0x8cab018--
main === --- 23 <- 0x8cab008-- num 5
Integers before sorting:
23 3 45 75 54
********************** In ascending order *************************
Pass0 Iter0 -> 23 3 45 75 54
Pass0 Iter1 -> 3 23 45 75 54
Pass0 Iter2 -> 3 23 45 75 54
Pass0 Iter3 -> 3 23 45 75 54
Pass1 Iter0 -> 3 23 45 54 75
Pass1 Iter1 -> 3 23 45 54 75
Pass1 Iter2 -> 3 23 45 54 75
*****************************************************************
********************** In descending order *************************
Pass0 Iter0 -> 3 23 45 54 75
Pass0 Iter1 -> 23 3 45 54 75
Pass0 Iter2 -> 23 45 3 54 75
Pass0 Iter3 -> 23 45 54 3 75
Pass1 Iter0 -> 23 45 54 75 3
Pass1 Iter1 -> 45 23 54 75 3
Pass1 Iter2 -> 45 54 23 75 3
Pass2 Iter0 -> 45 54 75 23 3
Pass2 Iter1 -> 54 45 75 23 3
Pass3 Iter0 -> 54 75 45 23 3
*****************************************************************
Integers after sorting:
75 54 45 23 3
```

*/
****

9.1.5 MemMove implementation in glibc reg:

I checked the glibc implementation and as I mentioned ,there is no need of
temporary copy of the src buffer is required.
corresponding code snippet from glibc source follows here:

```
/* This test makes the forward copying code be used whenever possible.
Reduces the working set. */
if (dstp - srcp >= len) /* *Unsigned* compare! */{
/* Copy from the beginning to the end. */
}
else{
/* Copy from the end to the beginning */
}
```

9.1.6 write a program to extract 10 bit temperature reading (negative or positive value) from sensor registers of size byte each located at adress 0x30 and 0x31 rectively.

out of 10 bits 0x30 hold two Lsbs 1-0 at bit positions 7 and 8 respectively.
remaining 8 bits are from 0x31 reg that holds 9-2 at bit positions 7 and 0 respectively.
0x30 ----> b1 b0 X X X XXX
0X31 -----> b9 b8 b7 b6 b5 b4 b3 b2
Ans:

```
short val_temp = * ( char *)(0X31);
short val = * ( char *)(0X30);
val = (val << 2) |(val_temp);
// do sign extension logic for negative sensor readings
val = val <<6;
val = val >>6;
```

9.1.7 why structure padding is required?

Faster memory access as all members starting addresses are aligned to a boundary
corresponding to their sizes.
but again it is compiler dependent where some can do structure padding and some wont if
hardware support unaligned memory access.

9.1.8 Why do we need #pragma pack(x) ? use-case ?

structures that hold network packets and headers are usually packed so that no memory is
wasted because of structure padding. this in-turn saves network bandwidth while transmitting
such packets over network

9.1.9 Define sizeof(array) or sizeof(int) pointer?

sizeof(x) ((&x +1) - (&x))
it works for any data type

9.1.10 how do you pass any two dimensional array using pass by value method?

```
void func(int arrayx[][6],int rows){
}
int main(){
int array[3][6];
func(array);
```

}

9.1.11 column size must be specified in arg list of func like int arrayx[][6] as in prev question .why?

if we want to access any value say arrayx[1][2] , compiler interpret it as
arrayx + col_size *1 + 2 ... so for compiler column size is required to evaluate any addresses within the received func.

9.2 Linux memory management session short summary:

1)General architecture of memory management.
2)Pages and Zones ,they way getting allocated in linux.
3)API's for getting a page, getting a zero initialized page and freeing a page.
4)Usage of kmalloc for allocating memory.
5)gfp_mask flags.
6)Slab architecture, design and api's in detail.
7)kernel process stacks
8)Usage of vmalloc and mmmap.

9.3 Qualcomm Interview Questions

9.3.1 What is the difference between char s[] and char *s in C?

The difference here is that
char *s = "Hello world";
will place Hello world in the read-only parts of the memory and making s a pointer to that, making any writing operation on this memory illegal. While doing:
char s[] = "Hello world";
puts the literal string in read-only memory and copies the string to newly allocated memory on the stack. Making
s[0] = 'J';
legal

9.3.2 Easy way to multiply a number by 7 ? and also for 3

Ans :
a. n*7 == (n<<3) – n
b. n*8=(n<<3)

9.3.3 what is memory leak

Ans : A memory leak is caused when you allocated memory, haven't yet deallocated it, and you will never be able to deallocate it because you can't access it anymore.
eg: If an exception is raised between allocation and deallocation, memory leak will occur.

9.3.4 what is dangling pointer

```
{
char *dp = NULL;
/* ... */
{
char c;
dp = &c;
} /* c falls out of scope */
```

```
/* dp is now a dangling pointer */
}
```

9.3.5 Given the input as ABCDEF **Expected output as EFCDAB**

Method1:
One way is using Masking Technique.
A=(((num&0x0000ff)<<16) | (num&0x00FF00) | ((num & 0xFF0000) >>16))

9.3.6 Difference Process Vs Thread

· An executing instance of a program is called a process.
· Threads exist as subsets of a process
· Thread is termed as a 'lightweight process', since it is similar to a real process but executes within the context of a process and shares the same resources allotted to the process by the kernel
· threads (of the same process) run in a shared memory space, while processes run in separate memory spaces
· processes interact only through system-provided inter-process communication mechanisms.
· Context switching between threads in the same process is typically faster than context switching between processes.

9.3.7 Copy on write :

· Copy-on-write (sometimes referred to as "COW") is an optimization strategy used in computer programming .
· The fundamental idea is that if multiple callers ask for resources which are initially indistinguishable, you can give them pointers to the same resource.
· This function can be maintained until a caller tries to modify its "copy" of the resource, at which point a true private copy is created to prevent the changes becoming visible to everyone else.

9.3.8 what is deadlock? causes? ways to avoid

Ans : · Is it a state where two or more operations are waiting for each other, say a computing action 'A' is waiting for action 'B' to complete, while action 'B' can only execute when 'A' is completed. Such a situation would be called a deadlock.
In order for deadlock to occur, four conditions must be true.

- **Mutual exclusion** - At least one resource must be non-shareable.[1] Only one process can use the resource at any given instant of time
- **Hold and Wait** - processes currently holding resources can request new resources
- **No preemption** - Once a process holds a resource, it cannot be taken away by another process or the kernel.
- **Circular wait** - Each process is waiting to obtain a resource which is held by another process.
- **Deadlock** can be avoided if certain information about processes are available to the operating system before allocation of resources, such as which resources a process will

consume in its lifetime

9.3.9 Interprocess Communication

- Interprocess Communication (IPC) is a mechanism by which two or more processes communicate with each other to exchange data by synchronizing their activities.
- PC uses read() and write() and some Input/Output (I/O) system functions communication.
- IPC provides a programming interface that allows you to create and manage processes, which can run concurrently in an Operating System (OS).

9.3.10 IPC Types

In Unix, IPC enables communication between processes either on the same computer or on different computers. The different types of IPC available are:

·**Local IPC**: Allows one or more processes to communicate with each other on a single system as follows:

o **Pipe :** Passes information from one process to another process. There are two types of pipes, named pipes and unnamed pipes.An unnamed pipe is a oneway communication IPC whereas a named pipe is a two-way communication IPC.

o **Signals**: Are software-generated interrupts sent to a process when an event occurs.

o **Message queuing :** Allows processes to exchange data by using a message queue.

o **Semaphores** : Synchronizes the processes activities concurrently competing for the same system resource.

o **Shared memory :** Allows memory segments to be shared among processes communicating with each other to exchange data between them. This method of IPC is used for faster communication between processes than the reading and writing of data by other operating system services.

o **Sockets :** Are end-points for communication between processes. They allow communication between a client program and a server program.

9.3.11 Priority Inversion (what, where and why)

Ans: priority inversion is a problematic scenario in scheduling in which a high priority task is indirectly preempted by a medium priority task effectively "inverting" the relative priorities of the two tasks.

Consider a task L, with low priority, that requires a resource R. Now, consider another task H, with high priority. This task also requires resource R. If H starts after L has acquired resource R, then H has to wait to run until L relinquishes resource R.

Everything works as expected up to this point, but problems arise when a new task M (which does not use R) starts with medium priority during this time. Since R is still in use (by L), H cannot run. Since M is the highest priority unblocked task, it will be scheduled before L. Since L has been preempted by M, L cannot relinquish R. So M will run until it is finished, then L will run - at least up to a point where it can relinquish R - and then H will run. Thus, in the scenario above, a task with medium priority ran before a task with high priority, effectively giving us a priority inversion.

Priority inheritance: priority inheritance is a method for eliminating priority inversion problems. Using this programming method, a process scheduling algorithm will increase the priority of a process to the maximum priority of any process waiting for any resource on which the process has a resource lock .

The basic idea of the priority inheritance protocol is that when a job blocks one or more high priority jobs, it ignores its original priority assignment and executes its critica l section at the highest priority level of all

the jobs it blocks. After executing its critical section, the job returns to its original priority level. Consider three jobs:
Job Name Priority
H High
M Medium
L Low
Suppose H is blocked by L for some shared resource. The priority inheritance protocol requires that L executes its critical section at the (high) priority of H. As a result, M will be unable to preempt L and will be blocked. That is, the higher priority job M must wait for the critical section of the lower priority job L to be executed, because L now inherits the priority of H. When L exits its critical section, it regains its original (low) priority and awakens H (which was blocked by L). H, having high priority, immediately preempts L and runs to completion. This enables M and L to resume in succession and run to completion.

9.3.12 Fragmentation.(The process or state of breaking or being broken into small or separate parts.)

· Fragmentation is a phenomenon in which storage space is used inefficiently .
· When a program is started, the free memory areas are long and contiguous. Over time and with use, the long contiguous regions become fragmented into smaller and smaller contiguous areas. Eventually, it may become impossible for the program to request large chunks of memory.

Types of Fragmentation **:**

1. Internal fragmentation

· Due to the rules governing memory allocation , more computer memory is sometimes allocated than is needed. For example, memory can only be provided to programs in chunks divisible by 4, 8 or 16, and as a result if a program requests perhaps 23 bytes, it will actually get a chunk of 24. When this happens, the excess memory goes to waste. In this scenario, the unusable memory is contained within an allocated region, and is thus termed internal fragmentation

2. External fragmentation

· External fragmentation arises when free memory is separated into small blocks and is interspersed by allocated memory.
· The result is that, although free storage is available, it is effectively unusable because it is divided into pieces that are too small individually to satisfy the demands of the application.

3. Data fragmentation

· Data fragmentation occurs when a collection of data in memory is broken up into many pieces that are not close together. It is typically the result of attempting to insert a large object into storage that has already suffered external fragmentation.
· system puts the new data in new non-contiguous data blocks to fit into the available holes. The new data blocks are necessarily scattered, slowing access due to seek time and rotationa l latency of the read/write head.

There are a variety of algorithms for selecting which of those potential holes :

- The **"best fit"** algorithm chooses the smallest hole that is big enough.
- The **"worst fit"** algorithm chooses the largest hole.
- The " first-fit algorithm " chooses the first hole that is big enough.

- The **"next fit"** algorithm keeps track of where each file was written.

The "next fit" algorithm is faster than "first fit", which is in turn faster than "best fit", which is the same speed as "worst fit"

9.4 IMC interview Questions :

Set 1:

9.4.1 What is a Mutex.

9.4.2 Mutex vs Semaphore.

9.4.3 Semaphore lock and unlock implementation ?

9.4.4 Atomic enter and exit implementation in C.

9.4.5 why and what is critical section? what are the ways to implement to critical section?

9.4.6 how do you implement memory manager for a OS?

9.4.7 whats are the parameters passed to create thread?

9.4.8 when a thread will run during thread creation or explicit run api is required?

9.4.9 what is Message queue?

9.4.10 what is interrupt?

9.4.11 how interrupt is different from message Queue event?

9.4.12 what is context switch ? Implement context switch in C?

9.4.13 malloc(0) significance?

9.4.14 what is memory pool ?

9.4.15 what are the different types of schedulers you know? and explain?

9.4.16 how to disable interrupts in C (should not use any OS calls)?

9.4.17 TASK A acquires Semaphore S. TASK B and TASK C tries to acquire Semaphore S.

9.4.18 What happens ? when TASK A releases semaphore what will happen?

9.4.19 How good are you in C? :)-

Set 2.

9.4.20 Three objects,Triangle, Square and circle were given to you of different lengths, You have to form a rectagle which fits all these 3 objects and allocate memory for that.

Ans : Since we have the coardinates of all objects, We can calculate the minimum and maximux coardinates. Once we know the min and max coardinates we can draw a diagonal joining min and max coardinates.

Once we have the diagonal, we can form a rectangle and allocate memory for that rectangle.

9.4.21 we will send an AT commnad in this format (AT+ 5, green). You have to recieve the AT command and revert back with something like this

"5 is green", So that sender of AT command will just print that string. How will you implement this.

Ans : We have to implement a AT handler which will parse and process the given AT command. After parsing the AT command store the arguements in two different variables.
For example var1 = 8
var2 = green
Append "is" to first var1 using stringcat librarry function.
After that append var2 to var1.
For example if command is AT 5 green.
will will get "5 is green". send this string to the transmitter.

9.4.22 Say there is appication in mobile, user keeps on pressing some keys repetetdly, You have to implement a software which displays the charecter and count of that charecter.

see belo example
aaaa bbbb cccc dddd
in this case you have to print
a 4
b 4
c 4
d 4
abcd aabbbcdd
In this case you have to print.
a 1
b 1
c 1
d 1
a 2
b 3
c 1
d 2
Ans : Implement an API which does the following.
1. Once we recieve the interrupt(alphabet), process the input and store it in a table with the count
2. Once we recieve next interrupt compare the current input with previous input which you have already stored in a table.
3. If both are same increment the only count of that input in the table.
4. Once the current input changes from previous input, call a display API which dispalys the Alphabet with the count.

9.4.23 Students and names

9.4.23.1 100 students with names (All names are of different sizes) were given. How will you implement a fastest software which has (O)1 complexity to search a given name.

Ans :
This can be implemented using the hash tables. since all the names are of different length. We can store these names with key as length using the hash function. Since all the names has unique key(which is lenght of the name). We can search the given name using hash function

---

9.4.23.2 100 students with names (names are of same and different sizes) were given. How will you implement a fastest software which has (O)n complexity to search a given name (You should implement less complex software).

Ans :

This can be implemented using the hash tables. since all the names are of same or different length. We can store the names which are of same lengths with key as length using the hash function. That means tag the names with same length with unique key using the linked list.

For example:

Names with legth as 7 charecters can be stored with a key 7, similarly names with 9 charecters can be stored with a key 9.
Once we have the hash table , we can search the given name as fast as possible.

9.4.24 Difference between process and threads and Explain when you prefer process over thread and vice versa.

Ans: Difference between process and threads:

1.Threads share the address space of the process that created it but processes have their own address space.
2.Threads have direct access to the data segment of its process but processes have their own copy of the data segment of the parent process.
3.Threads can directly communicate with other threads of its process but processes must use interprocess communication to communicate with sibling processes.
4.Threads have almost no overhead but processes have considerable overhead.
5.New threads are easily created; new processes require duplication of the parent process.
6.Changes to the main thread (cancellation, priority change, etc.) may affect the behavior of the other threads of the process but changes to the parent process does not affect child processes.
when you prefer process over thread and vice versa:
Whole application can be made as a process, Small componets that application can be made as threads.
Example : Consider printer , Entire printer application can be made as a process, where we can have threads inside that printer application which can perform, printing job, accepting user input etc.

9.4.25 Difference between semaphore and mutex When u prefer semaphore over mutex and vice versa, give example?

Difference between semaphore and mutex

1.Mutex is locking mechanism used to synchronize access to a resource. Semaphore is signaling mechanism.
2.No one owns semaphores, whereas Mutex are owned and the owner is held responsible for them, that means In case the of Mutex, the thread that owns the Mutex is responsible for freeing it. However, in the case of semaphores, this condition is not required. Any other thread can signal to free the semaphore.
3.A semaphore can be a Mutex but a Mutex can never be semaphore. This simply means that a binary semaphore can be used as Mutex, but a Mutex can never exhibit the functionality of semaphore.
When u prefer semaphore over mutex and vice versa
Semaphore: Use a semaphore when thread want to sleep till some other thread tells you to wake up. Semaphore 'down' happens in one thread (producer) and semaphore 'up' (for same semaphore) happens in

another thread (consumer) e.g.: In producerconsumer problem, producer wants to sleep till at least one buffer slot is empty – only the consumer thread can tell when a buffer slot is empty.

Mutex: Use a mutex when a thread want to execute code that should not be executedby any other thread at the same time. Mutex 'down' happens in one thread and mutex 'up' must happen in the same thread later on. e.g. If you are deleting a node from a global linked list, you do not want another thread to muck around with pointers while you are deleting the node. When you acquire a mutex and are busy deleting a node, if another thread tries to acquire the same mutex, it will be put to sleep till you release
the mutex.

9.4.26 Write a program to reverse string.

```
#include<stdio.h>

int main(){
	char str[50];
	char rev[50];
	int i=-1,j=0;
	printf("Enter any string : ");
	scanf("%s",str);
	while(str[++i]!='\0');
	while(i>=0)
		rev[j++] = str[--i];
	rev[j]='\0';
	printf("Reverse of string is : %s",rev);
	return 0;
}
```

9.4.27 Write a program to implemnt stack and queue using arrays.

Stack using arrays:

```
#define MAX 4 //you can take any number to limit your stack size
#include<stdio.h>
#include<conio.h>
int stack[MAX];
int top;
void push(int token){
char a;
if(top==MAX-1){
printf("Stack full");
return;
}
do{
printf("\nEnter the token to be inserted:");
scanf("%d",&token);
top=top+1;
stack[top]=token;
printf("do you want to continue insertion Y/N");
```

```
a=getch();
}
while(a=='y');
}

int pop(){
	int t;
	if(top==-1){
		printf("Stack empty");
		return -1;
	}
	t=stack[top];
	top=top-1;
	return t;
}

void show(){
	int i;
	printf("\nThe Stack elements are:");
	for(i=0;i<=top;i++) {
		printf("%d",stack[i]);
	}
}

int main() {
	char ch , a='y';
	int choice, token;
	top=-1;
	printf("1.Insert");
	printf("\n2.Delete");
	printf("\n3.show or display");
	do{
		printf("\nEnter your choice for the operation: ");
		scanf("%d",&choice);
		switch(choice){
			case 1:{ push(token); show(); break;}
			case 2:{ token=pop(); printf("\nThe token deleted is %d",token); show(); break; }
			case 3: { show(); break; }
			default:printf("Wrong choice"); break;
		}
		printf("\nDo you want to continue(y/n):");
		ch=getch();
	}
	while(ch=='y'||ch=='Y');
	getch();
}
```

Queue using arrays.

```
#include<stdio.h>
#define MAX 50
int queue_arr[MAX];
int rear = - 1;
int front = - 1;
main(){
	int choice;
	while (1) {
		printf("1.Insert\n");
		printf("2.Delete\n");
		printf("3.Display\n");
		printf("4.Quit\n");
		printf("Enter your choice : ");
		scanf("%d", &choice);
		switch (choice) {
			case 1: insert();
				break;
			case 2: del();
				break;
			case 3: display();
				break;
			case 4: exit(1);
			default: printf("Wrong choice\n");
		} /*End of switch*/
	} /*End of while*/
} /*End of main()*/

insert() {
	int added_item;
	if (rear == MAX - 1)
		printf("Queue Overflow\n");
	else {
		if (front == - 1)
			/*If queue is initially empty */
			front = 0;
		printf("Input the element for adding in queue : ");
		scanf("%d", &added_item);
		rear = rear + 1;
		queue_arr[rear] = added_item;
	}
} /*End of insert()*/


del() {
	if (front == - 1 || front > rear){
```

```
            printf("Queue Underflow\n");
            return ;
        }
        Else {
            printf("Element deleted from queue is : %d\n", queue_arr[front]);
            front = front + 1;
        }
} /*End of del() */

display() {
        int i;
        if (front == - 1)
            printf("Queue is empty\n");
        else {
            printf("Queue is :\n");
            for (i = front; i <= rear; i++)
                printf("%d ", queue_arr[i]);
            printf("\n");
        }
} /*End of display() */
```

9.4.28 Write a program to delete a node from linked list?

```
void deleteNode(struct node *head, struct node *n){
// When node to be deleted is head node
if(head == n){
if(head->next == NULL){
        printf("There is only one node. The list can't be made empty ");
        return;
}
/* Copy the data of next node to head */
head->data = head->next->data;
// store address of next node
n = head->next;
// Remove the link of next node
head->next = head->next->next;
// free memory
free(n);
return;
}
// When not first node, follow the normal deletion process
// find the previous node
struct node *prev = head;
while(prev->next != NULL && prev->next != n)
prev = prev->next;
// Check if node really exists in Linked List
```

```
if(prev->next == NULL) {
printf("\n Given node is not present in Linked List");
return;
}
// Remove node from Linked List
prev->next = prev->next->next;
// Free memory
free(n);
return;
}
```

9.4.29 What are staic and dynamic libraries, whats the differece between them. How you specify a path of dynamic library inside a c file(hint: use dlopen)

Static library:

A static library consists of routines that are compiled and linked directly into your program. When you compile a program that uses a static library, all the functionality of the static library becomes part of your executable. On Windows, static libraries typically have a .lib extension, whereas on linux, static libraries typically have an .a (archive) extension. One advantage of static libraries is that you only have to distribute the executable in order for users to run your program. Because the library becomes part of your program, this ensures that the right version of the library is always used with your program. Also, because static libraries become part of your program, you can use them just like functionality you've written for your own program. On the downside, because a copy of the library becomes part of every executable that uses it, this can cause a lot of wasted space. Static libraries also can not be upgraded easy — to update the library, the entire executable needs to be replaced.

Dynamic library:

A dynamic library consists of routines that are loaded into your application at run time. When you compile a program that uses a dynamic library, the library does not become part of your executable — it remains as a separate unit. On Windows, dynamic libraries typically have a .dll (dynamic link library) extension, whereas on Linux, dynamic libraries typically have a .so (shared object) extension. One advantage of dynamic libraries is that many programs can share one copy, which saves space.

Perhaps a bigger advantage is that the dynamic library can be upgraded to a newer version without replacing all of the executables that use it. Because dynamic libraries are not linked into your program, programs using dynamic libraries must explicitly load and interface with the dynamic library. This mechanisms can be confusing, and makes interfacing with a dynamic library awkward. To make dynamic libraries easier to use, an import library can be used.

9.4.30 How you specify a path of dynamic library inside a c file:

We can use dlopen(), and dlsym() subroutines. The dlopen() function gives you direct access to the dynamic linking facilities by making the executable object file specified in pathname available to the calling process. It returns a handle that you can use in subsequent calls to dlsym() and dlclose().

9.4.31 What are hash tables explain.

Definition of a Hash Table

a hash table is a data structure used to implement an associative array, a structure that can map keys to values. A hash table uses a hash function to compute an index

into an array of buckets or slots, from which the correct value can be found.
Hash tables are good for doing a quick search on things.
For instance if we have an array full of data (say 100 items). If we knew the position that a specific item is stored in an array, then we could quickly access it. For instance, we just happen to know that the item we want it is at position 3; I can apply:
myitem=myarray[3];
With this, we don't have to search through each element in the array, we just access position 3.
The question is, how do we know that position 3 stores the data that we are interested in?
This is where hashing comes in handy. Given some key, we can apply a hash function to it to find an index or position that we want to access. Hash function
Key -------------> Index to array.

9.4.32 Say you have a program hello.c, You will compile using gcc compiler like below.
gcc hello.c
From this point untill a.out is produced, Explain what happens internally.
Ans :
compilation model:
c source code (hello.c)
|
Preprocessor
|
Compiler
|
Assembly code
|
Assembler
|
Object code(hello.o) + libraries
|
Linker->Executable(a.out)

9.4.33 Explain reference variables in c++.
A reference variable provides an alias (alternative name) for a previously defined variable. For example:
float total=100;
float &sum = total;
It means both total and sum are the same variables.
cout<< total;
cout << sum;
Both are going to give the same value, 100. Here the & operator is not the address operator; float & means a reference to float.