500+ Questions

Cracking C Programming Interview

500+ interview questions and explanations to sharpen your coding skills for a rewarding career

TANUJ KUMAR JHAMB

bpb

500+ Questions

Cracking C Programming Interview

500+ interview questions and explanations to sharpen your coding skills for a rewarding career

TANUJ KUMAR JHAMB

bpb

Cracking C Programming Interview

500+ Interview Questions and Explanations to Sharpen Your C Concepts for a Lucrative Programming Career

---

Tanuj Kumar Jhamb

www.bpbonline.com

Group Product Manager: Marianne Conor

Publishing Product Manager: Eva Brawn

Senior Editor: Connell

Content Development Editor: Melissa Monroe

Technical Editor: Anne Stokes

Copy Editor: Joe Austin

Language Support Editor: Justin Baldwin

Project Coordinator: Tyler Horan

Proofreader: Khloe Styles

Indexer: V. Krishnamurthy

Production Designer: Malcolm D'Souza

Marketing Coordinator: Kristen Kramer

First published: July 2022

Published by BPB Online

WeWork, 119 Marylebone Road

London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-89845-587

www.bpbonline.com

Dedicated to

To All Computer Engineers

About the Author

Tanuj Kumar Jhamb is holding experience of working with world top IT companies. He is currently working as a Software Engineer with Arista Networks. Prior to that he has worked as a Software Developer Engineer-II at Amazon, System Architect at Huawei, and Senior Software Developer at Oracle Incorporation. He is well versed in solving challenging real world data structure problems, and system design patterns. He completed Master's in Computer Science in 2015 from NIT Calicut. He got AIR 416, Percentile 99.81 in Gate-2013 exam in first attempt. Last but not least he has been teaching data structure and algorithms to gate aspirants in prestigious coaching institutes since 2014.

About the Reviewer

With more than 21 years of experience in academia and industry, **Dr. R Rajkumar** is currently working at RNS Institute of Technology, Bengaluru as Associate Professor. He has a vast experience in teaching and mentoring both undergraduate and postgraduate students with subjects like Computer Organization, UNIX and C Programming, Database Management Systems, Digital Communication, Internet of Things, Java Programming, etc. He has several publications in refereed journals on Deep Learning techniques. His current area of interest is Optimization of Algorithms for Data and Intelligent Systems.

Acknowledgement

I would like to express my special thanks of gratitude and deep regards to Mr. Mahipal Singh Parmar for sharing his software technical expertise, and vast conceptual knowledge of coding. I could have never completed this book without Mahipal’s valuable and constructive suggestions.

I would like to thank a few more people Mr. Lalit Kumar, Mrs. Madhu, Mr. Kuldeep Kumar, Mrs. Ridhi, Mrs Jyotsna, Mr Pratap Singh, Mrs Basanti, Mr Harsh Kumar, Mrs. Smriti, and Mr Chandan Kumar for their continued and ongoing support they have given me during the writing of this book.

My gratitude also goes to the team at BPB Publications for being supportive enough to provide me enough time to finish the book and allow me to publish the book. I want to thank Dr. R Rajkumar, Mr. Ravi Kumar Pedappu for reviewing the book.

Preface

The best way to learn new things is to do things. There are lots of concepts in C programming that are not easily available either on the internet or in textbooks. We tried our best to deliver concepts through explanations of C questions. Each and every question has a concept behind it. These concepts are damn important for students who are planning to appear for any multinational IT company job. Every company needs programmers/coders who are good with logic and sound in fundamentals. This book is proved a diamond for genuine C programmers and for serious job seekers. The book contains more than 500 C programming questions. You only need a maximum of 25 days to complete the book. The guidelines for reading this book are given below. We recommend you to follow these guidelines and try to finish the book before the deadline. At the end of each group, some other important interview questions are present. Try to write programs and execute . If you are having a lack of time and good programming skills then try to write at least an algorithm or pseudocode for them.

All C codes are compiled and executed with gcc 4.8.2 compiler driver running on a Linux system with kernel 3.13.0-32-generic under ubuntu-14.04 32-bit Operating system. We also recommend you to execute the given codes in your system for better understanding and let us know if there is any correction.

Organization of the Book

The chapter-1 "A touch to C" describes conceptual understanding about C programs. It is highly recommendable to the readers to go through this chapter before proceeding further. After this chapter, the book is divided into 7 groups, each group contains around 60 C programming questions with explanations and other important interview questions. After completing all the groups, there are more than 100 questions given in 5 sample papers with solutions for your practice. The book overall contains more than 600 interview questions.

Prerequisite

A very basic understanding of theoretical C concepts is required for reading this book.

Guidelines For Reading the Book

- Try to finish chapter-1 "**A Touch to C**" in 2 hours.
- The book contains 7 groups of questions, each group has around 60 questions. You can solve 20 questions per day. So, each group takes 3 days to solve. You can solve all the conceptual C programming questions in 21 days only.
- Give 4 days to sample papers. Overall you need only 25 days to solve this book.
- A book solving chart is given on next page.

A **book solving chart** is given below. You can write date and hours that you spent for solving each group.

Book Solving chart

S. No	Topic Name	Date	Time (you studied)
1	**A Touch to C**		
2	**Group-1 Questions**	1)	
		2)	
		3)	
3	**Group-2 Questions**	1)	
		2)	
		3)	
4	**Group-3 Questions**	1)	
		2)	
		3)	
5	**Group-4 Questions**	1)	

		2)	
		3)	
6	**Group-5 Questions**	1)	
		2)	
		3)	
7	**Group-6 Questions**	1)	
		2)	
		2)	
8	**Group-7 Questions**	1)	
		2)	
		3)	
9	**Sample Paper-1**		
10	**Sample Paper-2**		
11	**Sample Paper-3**		
12	**Sample Paper-4**		
13	**Sample Paper-5**		

Coloured Images

Please follow the link to download the
Coloured Images of the book:

https://rebrand.ly/zlis9i8

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

exclusive discounts and offers on BPB books and eBooks.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

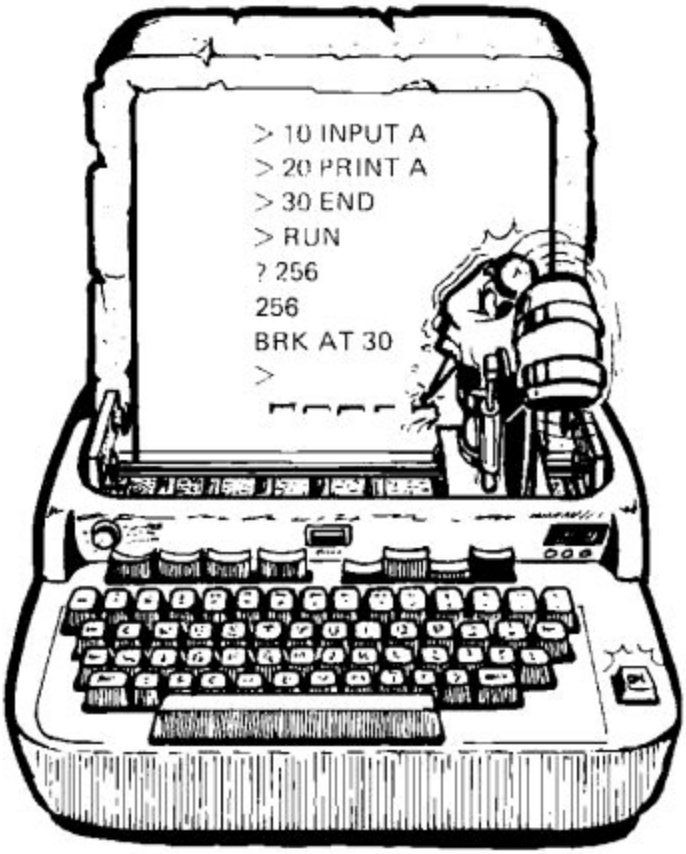
Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Table of Contents

A Touch To C

Imagination is more powerful than knowledge.



Historical Perspective

C was developed from 1969 to 1973 by Dennis Ritchie at Bell Laboratories. The American National Standards Institute (ANSI) signed the ANSI C standard in 1989. Kernighan and Ritchie describe ANSI C in their book, which is known affectionately as "K&R".

C and Unix: C was developed from the beginning as the system programming language for the operating system Unix. Almost all the components and libraries of the Unix kernel were implemented in C. It could be easily portable to new machines (having different architecture), just a compilation of a few C files require to run the Unix in a new machine.

C is a simple language: C is a high-level programming language that allows a programmer to write programs that are independent of a particular type of machine. C totally follows *write once, compile anywhere.* Such languages are considered as high level because they are closer to human languages and a far from machine languages. Whereas the assembly languages are considered low-level because they are very close to machine languages. The foremost advantage of high-level languages is that they are easily understandable to programmers. But for running a program in a computer system, the program must be translated into machine language (low-level language). The compiler or interpreter performs this translation.

Structure

In this chapter, we will cover the following topics:

- Objective
- Introduction to the C Program
- Execution of the C Program
- Memory management and Organization of the C Program
- Types of Error in C
- Assumptions
- Other C Programming Important Aspects
- Conclusion
- Key terms
- Try Yourself

Objective

The intention of this chapter is to get an understanding about the anatomy of the C program which will help in unhiding the abstraction (hidden details) about the C program. It will help the readers visualize the running of C program, which is called Process in terms of operating system. This chapter also explains the types of errors which are mostly unclear for beginners/intermediate programmers.

Introduction to the C Program

It is a code written in the English like language following syntax introduced by Brain Kernighan and Dennis Ritchie & later by ANSI.

A simple C code

`#include<stdio.h>`	Header / Preprocessor directive
`int main()`	Starting Point
`{`	Body of the C program block starts
`printf("hello India");`	Body of the C program
`return 0;`	Return statement
`}`	Body of the C program block end

Table 1.1: *Outlook of the C Program*

Header/PreProcessor directive: These should be inserted in very beginning of the C Program, and help us to include the extra common C code declarations which we want to use in our C program, and don't want to write their functionality again and again e.g. printf() statement.

Starting Point: The main() is considered as starting point of the every C program.

Curly Braces: The curly braces (start- '{', end- '}') signifies the start and end of particular block (if-else, while, function etc.)

Body: These are set of sequence of the logical instructions present under the curly braces, they give the actual meaning to the C program.

Return statement: This should be included in the last line of any C function which is having non-void prototype. The non-void functions have the prototype as "<non-void data type> <func_name>(arg1, arg2, …)", and there are two types of non-void data types (i) primitive (int, float, char etc.) data types, and (ii) user-defined (struct, enum etc.) data types. Here the "return 0" statement in the main function() returns the 0 to the shell/compiler (who is executing this C program).

We will trace the lifetime of the simple hello program, from the time it is being created by a programmer, until it executes on a system, prints "hello India", and terminates.

i. Programmer writes a program in editor (e.g. gedit) and saves as a text file called hello.c, also called source program.

ii. The source program is a sequence of bits in some meaningful (semantic) order and organized in 8-bit chunks called bytes (octets or cells).

iii. Modern computers represent each text character uniquely using ASCII notation. Each character has a unique 8-bit sized integer value. The ASCII representation of text characters is given in tabular form at the end of this chapter (*Table 1.4*).

The hello.c program is stored in a file as a sequence of bytes. The ASCII representation of the hello.c program is shown in the following *figure 1.1*.

#	i	n	c	l	u	d	e	<sp>	<	s	t	d	i	o
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111

.	h	>	\n	i	n	t	<sp>	m	a	i	n	(	)	\n
46	104	62	10	105	110	116	32	109	97	105	110	40	41	10

{	<sp>	<sp>	<sp>	<sp>	p	r	i	n	t	f	(	"
123	32	32	32	32	112	114	105	110	116	102	40	34

h	e	l	l	o	I	n	d	i	a	"	)	;	\n	}
104	101	108	108	111	73	110	100	105	97	34	41	59	10	125

Figure 1.1: *The ASCII text representation of hello.c.*

Each and every character has its corresponding integer value. For example, the first character '#' has a value of 35, the second character 'i' has a value of 105, and so on.

Execution of the C program

Execution refers to using system resources like CPU, memory, I/O devices in order to complete a particular task. For the execution of *hello.c* program, it needs to use all these resources like CPU for processing, I/O device-monitor for displaying the result "hello India" and so on.

The issue is that the source program we have written, *hello.c* is not understandable to the machine. It needs some more effort. The given program is written in human-readable form (high level) and the machine could not understand this. So, *hello.c* needs translation from high-level form

to low-level form (machine understandable). This translation can be done using a compiler or interpreter.

The process of compilation depends on the compiler we are using, some of the compilers are Turbo C, Dev-C, GCC, etc. Their translation process is entirely different. The compiled file is referred to as executable object files. On a Unix system, the execution of a C program can be done using GCC compiler as follows:

unix> gcc -o hello hello.c

The source file *hello.c* is read by GCC compiler driver which translates it into an executable object file.

unix> ./hello.

The loader loads this executable file into the main memory, and the CPU will start processing.

The translation using GCC is not a single phase thing. The GCC passes the source program to its four components: (i) preprocessor, (ii) compiler, (iii) assembler, and (iv) linker. They are altogether referred to as a compilation system. *Figure 1.2* will give a better idea of the compilation system as follows:

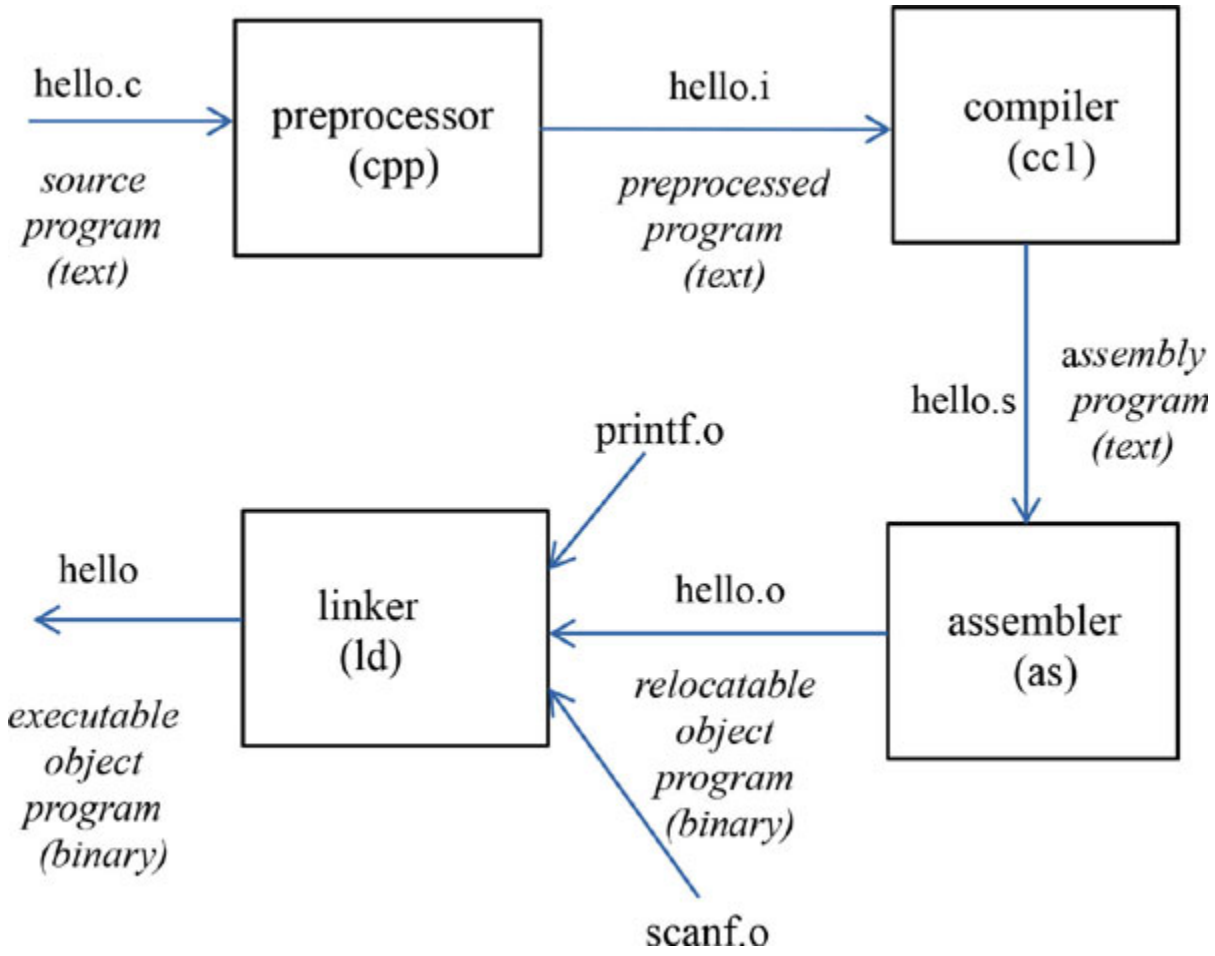


Figure 1.2: *The compilation system*

Preprocessor: The preprocessor (cpp) substitute the macros and the directives that start from # character directly in the program text of the original C program. It obtains another C program suffixed with ".i".

Compiler: It involves lots of sub-phases like lexical analysis, syntactic analysis, and so on. It obtains a program in assembly language that describes the machine instruction in the form of text (also called mnemonics). The output file of the compilation phase is suffixed with ".s".

Assembler: It translates the text file "hello.s" into machine-level instructions and packed it in the form of a relocatable object program and stores the output in file hello.o.

Linker: As we can look into the program that we are calling printf() function, there is no printf() function in our source program. The printf() function is the part of the standard C library provided by every C compiler. The printf function is already present in the form of compiled file and

resides in the C library as printf.o file. The linker manages the merging of these newly compiled (source program) and pre-compiled (printf.o) files. After merging, the linker obtains the executable file, is loaded by a loader into memory, and executed by the CPU.

It was just an overview of all the phases. We can look into the content of all these intermediate files (hello.i, hello.s, hello.o) bypassing different parameters to the gcc command. More detail on this will take us beyond the topic discussion.

The most important issue with any programming language is managing memory, We will see the memory management and memory organization of the C program in the next section.

Memory management and organization of the C program

The memory organization of the C program depends on how variables are organized within a relocatable object file (hello.o). This memory representation of the object file can be categorized into more than 10 areas. But for our better understanding we will focus on four areas (*figure 1.3*) which are as follows:

- Text Area
- Data Area
    - Initialized data fragment
    - Uninitialized data fragment (bss)
- Heap Area
- Stack Area

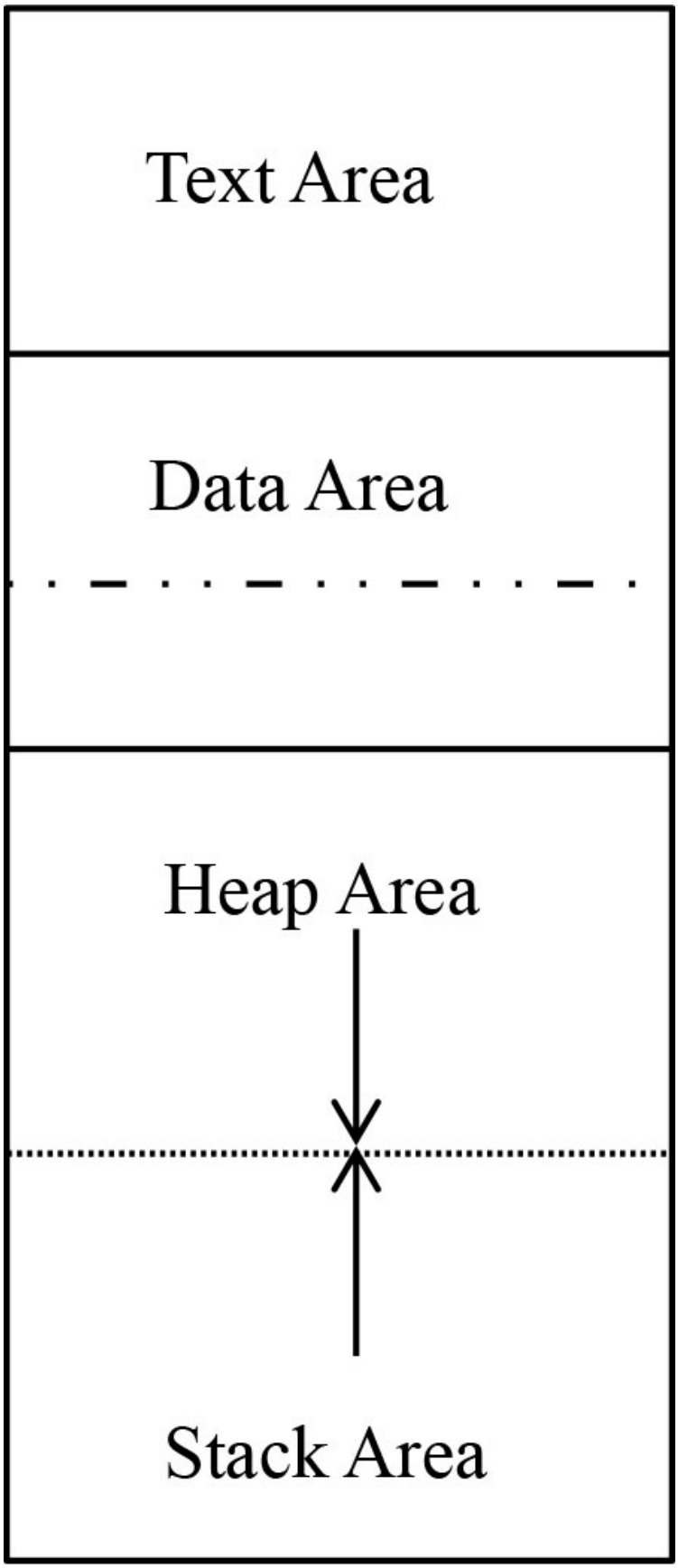

Text Area
Data Area
Heap Area
Stack Area

Figure 1.3: *Memory organization of a C program*

1. **Text Area:** It contains the machine code of the compiled program. It is also called *code area*. It is only readable, preventing machine instructions from getting modified accidentally. The text area is sharable among all the functions in a program. The existence of this area is till the exit of main().
2. **Data Area:** The memory for global C variables and static variables are maintained in this area. This area is readable and writable both. It is also sharable among all the functions in a program. Its lifetime is also till the termination of the program. It is further classified into two fragments:
    i. **Initialized data fragment:** All the global and static variables that are initialized with some value get allocated memory in this area.
    ii. **Uninitialized data fragment:** All the global and static variables that are not initialized lie in this area get allocated memory in this area. This area is also called block start by symbol (bss).
3. **Heap Area:** The dynamic memory allocation takes place in this area. The size for this area is not fixed, it grows till the largest available address is present. In C, it is managed by malloc, calloc, realloc and free, library function which uses the brk and sbrk system calls to regulate the size. This area is also shared by all the functions in a program. Its existence is till the termination of the program.
4. **Stack Area:** It is next to the heap area, the memory addresses for stack area and heap area are not fixed while data and text area are fixed. The heap and stack area grow towards each other. All the local variables (automatic) get stored in the stack area. When a program start execution, first the main() get pushed into the stack and then other functions get pushed when they get called. There are lots of changes that occur in the machine state when any function is called. The program counter gets updated with a new address; CPU executes instruction of a new function. The return address of the caller's, machine register, etc. gets saved onto the stack. The concept stack area is the foundation of recursion in C. The space allocated to any

function call gets free after the execution of a function and the space is reusable for other function calls.

An example of the memory organization of a real C program is shown in *figure 1.4*.

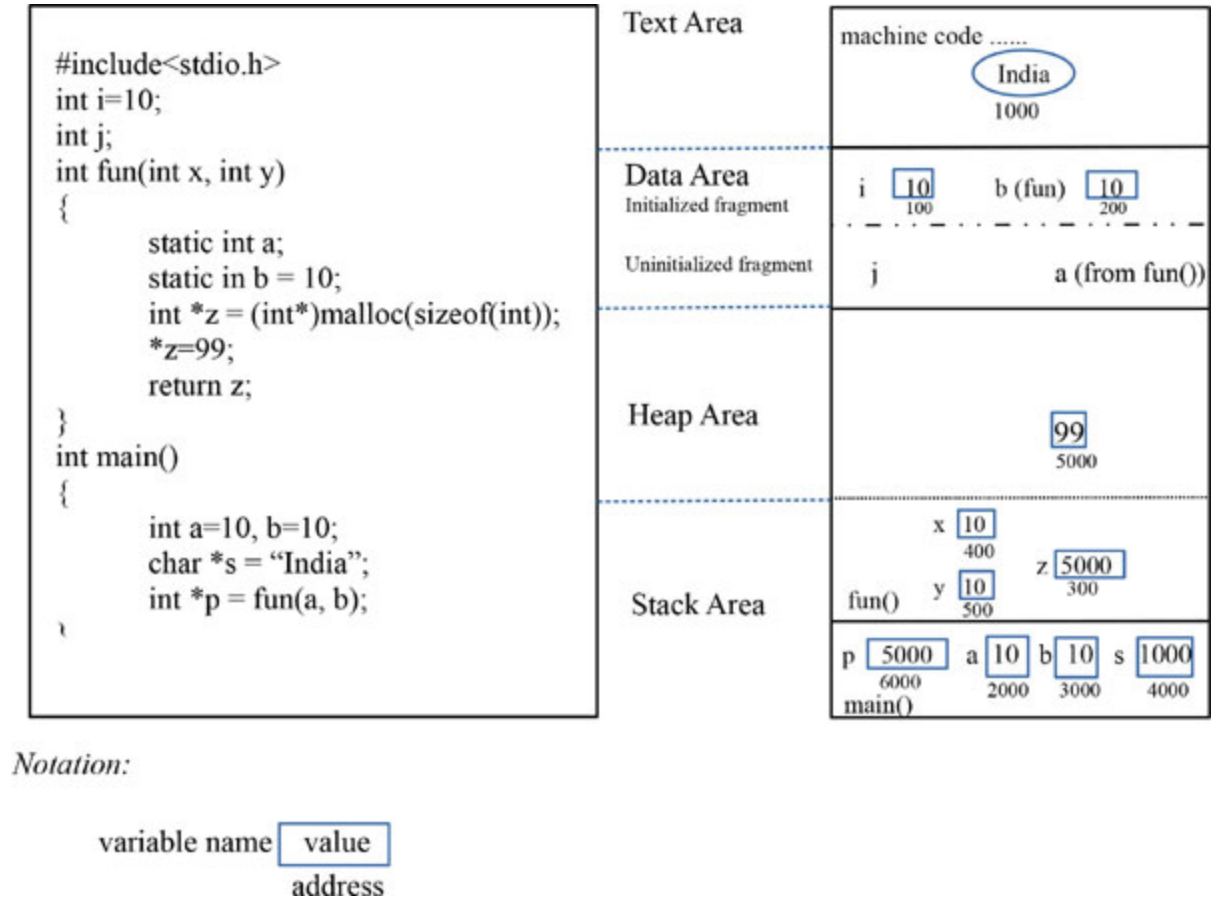


Figure 1.4: *An example- Memory organization of a C program*

Types of Error in C

Generally, four types of error occur in a program which is as follows:

Logical error: While writing a program, there are some logical mistakes in a program. It causes improper output. It could not be checked by the compiler, thus, programmers need to check by scanning the code manually.

Compile Time error: When any programmer does not follow the rules of a particular programming language. Hence, the compiler throws an error. There can be a variety of compile-time errors like syntactic error, semantic error. The syntactic error occurs when a programmer is not following the

syntax of writing a code of a particular language. The semantic error occurs when the compiler is not able to understand the meaning of some statement.

Link Time error: It occurs at the linking stage, which means there is no problem with our code, it compiles fine but some function or library module that is to be linked is missing. Linker does not allow to generate a final executable file. Many compilers do compilation and linking together. Like in the GCC compiler, the link-time error occurs during the compilation phase only.

Run Time error: These errors occur during the execution of a program. Suppose in a program, we are dividing by zero, it's a run time error. Suppose our program is trying to access a memory address, that is not allowed to access or that does not exist causing segmentation fault.

Assumptions

Machine Configuration:

The size of a pointer depends on the number of address lines available in a system. For a 32-bit machine, the size of the pointer is 4 bytes. The type of pointer does not affect its size. An integer pointer, character pointer, float pointer, struct pointer, void pointer, all have the same size that is 4B in our consideration.

Processor: Intel

Word length: 32-bit

Mode: Little Endian (we will discuss in explanation of G-2, Question-25)

Size of Data Types:

char - 1B

short - 2B

int - 4B

long - 4B

long int - 4B

float - 4B

double - 8B

Other C Programming Important Aspects

Keywords: C89 has 32 keywords, keywords are reserved words with special meanings. *Table 1.2* is representing the keywords available in C.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Table 1.2: *Keywords in C*

Character set: C includes the following characters:

- Lowercase and uppercase letters: a–z A–Z
- Decimal digits: 0–9
- Graphic characters: ! " #% & ‘ ( ) * + , - . /:; < = >? [\ ] ^ _ { | } ~
- Whitespace characters: *space, horizontal tab, vertical tab, form feed, newline*

Operator in C:

C supports large set of operators which are as follows:

- arithmetic: +, -, *, /, %
- assignment: =
- augmented assignment: +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
- bitwise logic: ~, &, |, ^
- bitwise shifts: <<, >>
- boolean logic: !, &&, ||
- conditional evaluation: ? :
- equality testing: ==, !=
- calling functions: ( )
- increment and decrement: ++, --
- member selection: ., ->

- object size: sizeof
- order relations: <, <=, >, >=
- reference and dereference: &, *, [ ]
- sequencing: ,
- subexpression grouping: ( )
- type conversion: (*typename*)

Operator precedence Table: *Table 1.3* is comparing the precedence and associativity among various types of operators in C. The upper rows in a table have higher precedence than the lower one and so on. The operators in the same row represent the same precedence. If two operators have the same precedence then the expression gets evaluated using the associativity of the operators.

Operator	Description	Associativity
() **[]** **.** ->** ++ --**	Parentheses (function call) Brackets (array subscript) Member selection via object name Member selection via the pointer Postfix increment/decrement	Left to Right
++ -- + - **! ~** **(*type*)** ***** **&** **sizeof**	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (change *type*) Dereference Address Determine size in bytes	Right to Left
*** / %**	Multiplication/division/modulus	Left to Right
+ -	Addition/subtraction	Left to Right
<< >>	Bitwise shift left, Bitwise shift right	Left to Right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	Left to Right
== **!=**	Relational is equal to/is not equal to	Left to Right
&	Bitwise AND	Left to Right
^	Bitwise exclusive OR	Left to Right
|	Bitwise inclusive OR	Left to Right

&&	Logical AND	Left to Right
||	Logical OR	Left to Right
?:	Ternary conditional	Right to Left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	Right to Left
,	Comma (separate expressions)	Left to Right

Table 1.3: *Precedence and associativity of operators*

Table of ASCII Character Codes: *Table 1.4* is as follows:

ASCII	Character Value	ASCII	Character Value	ASCII	Character Value
0	Null char	**43**	Plus	**86**	Uppercase V
1	Start of Heading	**44**	Comma	**87**	Uppercase W
2	Start of Text	**45**	Hyphen	**88**	Uppercase X
3	End of Text	**46**	Dot or full stop	**89**	Uppercase Y
4	End of Transmission	**47**	Slash or divide	**90**	Uppercase Z
5	Enquiry	**48**	Zero	**91**	Opening bracket
6	Acknowledgment	**49**	One	**92**	Backslash
7	Bell	**50**	Two	**93**	Closing bracket
8	Back Space	**51**	Three	**94**	Caret - circumflex
9	Horizontal Tab	**52**	Four	**95**	Underscore
10	Line Feed	**53**	Five	**96**	Grave accent
11	Vertical Tab	**54**	Six	**97**	Lowercase a
12	Form Feed	**55**	Seven	**98**	Lowercase b
13	Carriage Return	**56**	Eight	**99**	Lowercase c

14	Shift Out / X-On	**57**	Nine	**100**	Lowercase d
15	Shift In / X-Off	**58**	Colon	**101**	Lowercase e
16	Data Line Escape	**59**	Semicolon	**102**	Lowercase f
17	Device Control 1 (oft. XON)	**60**	Less than	**103**	Lowercase g
18	Device Control 2	**61**	Equals	**104**	Lowercase h
19	Device Control 3 (oft. XOFF)	**62**	Greater than	**105**	Lowercase i
20	Device Control 4	**63**	Question mark	**106**	Lowercase j
21	Negative Acknowledgement	**64**	At symbol	**107**	Lowercase k
22	Synchronous Idle	**65**	Uppercase A	**108**	Lowercase l
23	End of Transmit Block	**66**	Uppercase B	**109**	Lowercase m
24	Cancel	**67**	Uppercase C	**110**	Lowercase n
25	End of Medium	**68**	Uppercase D	**111**	Lowercase o
26	Substitute	**69**	Uppercase E	**112**	Lowercase p
27	Escape	**70**	Uppercase F	**113**	Lowercase q
28	File Separator	**71**	Uppercase G	**114**	Lowercase r
29	Group Separator	**72**	Uppercase H	**115**	Lowercase s
30	Record Separator	**73**	Uppercase I	**116**	Lowercase t
31	Unit Separator	**74**	Uppercase J	**117**	Lowercase u
32	Space	**75**	Uppercase K	**118**	Lowercase v
33	Exclamation mark	**76**	Uppercase L	**119**	Lowercase w
34	Double quotes	**77**	Uppercase M	**120**	Lowercase x
35	Number	**78**	Uppercase N	**121**	Lowercase y
36	Dollar	**79**	Uppercase O	**122**	Lowercase z
37	Procenttecken	**80**	Uppercase P	**123**	Opening brace
38	Ampersand	**81**	Uppercase Q	**124**	Vertical bar
39	Single quote	**82**	Uppercase R	**125**	Closing brace
40	Open parenthesis	**83**	Uppercase S	**126**	Equivalency

					sign - tilde
41	Close parenthesis	**84**	Uppercase T	**127**	Delete
42	Asterisk	**85**	Uppercase U		

Table 1.4: *ASCII Character Codes mapping to the Character Value*

Conclusion

In this chapter, we have learned about the structural and executional behavior of the C program. We have covered the several phases of the C program encountered during creation of executional binary file from a source file. We have understood the memory management of the C program and acquired the knowledge about how the C program looks in a machine. We have put some light on the types of errors which can be generated during execution of the C program which is a very important concept during technical papers/interviews.

Key terms

- Pre processor directive
- Compiler
- Assembler
- Linker
- Loader
- Text Area, Data Area, Heap Area, Stack Area

Try yourself

1. What is the difference between Declaration and Definition of a function and variable?
2. What are C Storage Classes (extern, auto, typedef, static) and Type Qualifiers?
3. What is the scope of variable in C program, Illustrate it?
4. Write a program to create a new header file and use it in your own C program?

Task 1

Teach the concepts you learned here to your friends/colleagues/classmates.

Task 2

Try to focus on implementing concepts rather than reading or making notes.

The best way to predict the future is to create it.

Group-1 Questions

Number of 'C' Question: 60

Other Important Interview Questions: 15

1. **What is the output of the following program?**

```
#include<stdio.h>
int *fun()
{
 int b = 10;
 return &b;
}
int main()
{
 int *p;
 p = fun();
 printf("%d\n", *p);
 return 0;
}
```

(a) 10

(b) Address of variable 'b'

(c) Unexpected Behavior

(d) none of these

2. **What modification should we do in the above program such that it returns the address of variable ‘b’ defined in called function?**
3. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int b = 65;
 void *p = &b;
 int *j = (int*)p;
 char *ch = (char*)p;
 printf("%d, %c\n", *j, *ch);
 return 0;
}
```

(a) 65, 65

(b) 65, A

(c) error- compile time

(d) error- run time

4. **What is the output of following program?**

```
#include<stdio.h>
int main()
{
 int b=65;
 void p = b;
 int c = (int)p;
 printf("%d\n", p);
 return 0;
}
```

(a) 65

(b) A

(c) error- compile time

(d) error- run time

5. **What is the output of the following program?**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 int *p = NULL;
 p = (int*)malloc(sizeof(int));
 *p = 10;
 free(p);
 int * q = (int*)malloc(sizeof(int));
 *q = 15;
 printf("%d, %d\n", *p, *q);
 return 0;
}
```

(a) 10, 15

(b) 15, 15

(c) error- run time

(d) 0, 15

6. **What is the output of the following program? Assumption: address of p is 1000 and q is 2000.**

```
#include<stdio.h>
#include<stdlib.h>
void *fun(int **q)
{
 int r = 20;
 **q=r;
 printf("%u ", *q);
}
int main()
{
 int *p = (int *)malloc(sizeof(int));
 *p = 55;
 fun(&p);
 printf("%d %u\n", *p, p);
 return 0;
}
```

(a) 1000 55 2000

(b) 1000 20 1000

(c) 1000 garbage 1000

(d) 1000 55 2000

7. **Which of the given statements about the given program is/are true?**

```
#include<stdio.h>
#include<stdlib.h>
void fun()
{
 int *q = (int *)malloc(sizeof(int));
 *q = 20;
}
int main()
{
 int *p;
 int *r=NULL;
 fun();
 return 0;
}
```

(i) p is wild pointer

(ii) q is dangling pointer

(iii) r is NULL pointer

(iv) p is dangling pointer,

(v) fun() is making memory leak

Options:

(a) i, ii, iii, v

(b) ii, iii, iv

(c) ii, iii, v

(d) i, iii, v

8. **What is the output of the following program?**

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

```
 char *p = "hello";
 char q[5] = "hello" ;
 p[2] = ‘r’; //line3
 q[2] = ‘r’; //line4
 printf("%s %s", p, q);
 return 0;
}
```

(a) hello hello

(b) herlo herlo

(c) run time error due to line 3

(d) run time error due to line 4

9. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *p = "hello";
 char *q = "hello";
 if(p == q)
  printf("hello");
 else
  printf("hi");
 return 0;
}
```

(a) hello

(b) hi

(c) hellohi

(d) error- compile time

10. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
int main()
{
 char *p = "hello";
 char q[] = "hello";
 printf("%d %d %d ", sizeof(*p), sizeof(p), sizeof(q));
```

```
  printf("%d %d", strlen(p), strlen(q));
  return 0;
 }
```

(a) 1 4 5 5 5

(b) 1 4 6 5 5

(c) 1 5 5 5 5

(d) 1 4 infinity 5 5

11. **Which of the following given statements are true?**

(i) `(void*)0 is void pointer`

(ii) `(void*)0 is NULL pointer`

(iii) `int *p = (int *)0; p is null pointer`

(iv) `a[i] == i[a]`

(v) `a[i][j] == *(*(a+i)+j);`

(a) i, ii, iii, iv

(b) ii, iii, iv

(c) ii, iii, iv, v

(d) All are true

12. **In each option, two declarations are present. State true if both the declarations in an option are equivalent?**

(a) i. `char *p=0;`

ii. `char *t = NULL;`

(b) i. `int i=0;`

ii. `char *q = (char*)i;`

(c) i. `char *q = 0;`

ii. `char *q = (char *)0;;`

(d) i. `char **a;`

ii. `char *a[];`

13. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 float a = 3.14;
```

```
 char *j;
 j = (char*)&a;
 printf("%d\n", *j);
 return 0;
}
```

(a) 3

(b) prints ascii value equivalent to 'a' float variable

(c) print ascii value in the first byte of float representation in binary

(d) Compile Time Error

14. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int const *p = 5; //line 1
 ++(*p); //line 2
 printf("%d", *p);
 *(++p) = 2; //line 4
 reutrn 0;
}
```

(a) 5

(b) Garbage

(c) Error at Line2

(d) Error at Line4

15. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = -1, j=-1, k=0, l=2, m;
 m = ++i || k++ && ++j || l++;
 printf("%d %d %d %d %d", i, j, k, l, m);
 return 0;
}
```

(a) 0, -1, 0, 2, 1

(b) 0, -1, 0, 3, 1

(c) 0, -1, 1, 3, 1

(d) None of these

16. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int p=9;
 int *q = &p;
 printf("%ld %ld %ld %ld ", sizeof(p++), sizeof(q),
 sizeof(++p), sizeof(++(*q)));
 printf("%d", p);
 return 0;
}
```

(a) 2 4 2 2 11

(b) 2 2 4 4 12

(c) 4 2 4 4 12

(d) 4 4 4 4 9

17. **What will be the output of the following program.?**

```
#include<stdio.h>
#define A 2 //line 2
#if B && A //line 3
#define C 3 //line 4
#else //line 5
#define C 4 //line 6
#endif
int main()
{
 printf("%d",C);//line 9
 return 0;
}
```

(a) 3

(b) 4

(c) error- compile eime

(d) Garbage Value

18. **What is the output of the following program?**

```
#include<stdio.h>
void fun(int *p)
{
 printf("%d\t", *p);
 if(*p > 0)
 {
  *p = *p-2;
  fun(p);
 }
 else
  *p=0;
}
int main()
{
 int i=10, j=9;
 fun(&i);
 printf("%d\n", i);
 fun(&j);
 printf("%d\n", j);
 return 0;
}
```

(a) 10, 8, 6, 4, 2, 0, 0
9, 7, 5, 3, 1, -1, -1

(b) 10, 8, 6, 4, 2, 0, 0
9, 7, 5, 3, 1, 0, 0

(c) 10, 8, 6, 4, 2, 0, 0
9, 7, 5, 3, 1, -1, 0

(d) None of these

19. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 #define int char
 int i=65;
```

```
 printf("%d", sizeof(i)); //line 3
 printf(" %d",sizeof('A')); //line 4
 return 0;
}
```

(a) 2 1

(b) 1 1

(c) 0 0

(d) 1 4

20. **What is the output of the following program?**

```
#include<stdio.h>int main()
{
 int i = 7;
 switch(i)
 {
  default:
   printf("zero ");
  case 1:
   printf("one ");
  case 2:
   printf("two ");
  case 3:
   printf("three ");
  case 4:
   printf("four ");
 }
 return 0;
}
```

(a) zero

(b) zero one two

(c) zero one two three four

(d) error- misplace default statement

21. **What is the output of the following program?**

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

```
 int i=500, j=700;
 printf("%d %d\n");
 return 0;
}
```

(a) 500, 700

(b) 700, 500

(c) compiler dependent

(d) garbage garbage

22. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int p=1, q=5;
 switch(p)
 {
  case 1:
   printf("hi\n");
  case q:
   printf("Bye\n");
 }
 return 0;
}
```

(a) hi

(b) Bye

(c) hi Bye

(d) error- compile time

23. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int p=1;
 switch(p)
 {
  case 1:
```

```
    printf("Hi\n");
   case 5-4:
    printf("Bye\n");
  }
  return 0;
}
```

(a) hi

(b) bye

(c) error- compile time

(d) Hi Bye

24. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int *j;
 {
  int i=10;
  j=&i;
 }
 printf("%d", *j); //line 6
 return 0;
}
```

(a) error- at line 6

(b) garbage

(c) 10

(d) compiler dependent

25. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 const int p = 4;
 int q = ++p; //line 2
 printf("%d, %d\n", p, q);
 return 0;
```

```
}
```

(a) 5, 5

(b) 4, 5

(c) error- compile time

(d) error- run time

26. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=10, j=6;
 printf("%d\n", i+++j);
 return 0;
}
```

(a) 17

(b) 16

(c) error- compile time

(d) None of these

27. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 void *p;
 int i=20;
 p = &i;
 void *q = p; //line 4
 printf("%d, %d, %d\n", i, *p, *q); //line 5
 return 0;
}
```

(a) 20, 20, 20

(b) 20, g, g

(c) compiler error at line 4

(d) compiler error at line 5

28. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[] = {10, 20, 30, 40, 50};
 printf("%u %u %u", sizeof(a[1]), sizeof(*a), sizeof(a));
 return 0;
}
```

(a) 4 8 20

(b) 4 4 20

(c) 4 4 4

(d) 8 8 20

29. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 void *p;
 int *q;
 int a[1] = {10, 11};
 q = a;
 q++;
 printf("%d ", *q);
 p = a;
 p++;
 printf("%d\n", *p);
 return 0;
}
```

(a) 10 11

(b) 11 11

(c) error- compile time

(d) error- run time

30. **What is the output of the following program?**

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

```
  int p, r;
  int q = scanf("%d %d", &p, &r);
  printf("%d", q+printf("India"));
  return 0;
 }
```

(a) India5

(b) India7

(c) India<garbage>

(d) 7India

31. **How many time the while loop execute?**

```
#include<stdio.h>
int main()
{
  int x, y;
  while(y = scanf("%d", &x)) //line 2
  {
    printf("hello\n");
    y=-2;
  }
  return 0;
}
```

(a) one time

(b) two time

(c) three time

(d) it execute till non integer word is entered

32. **Which of the following statement is/are true about the declaration and definition?**

(a) Declaration tells about the type of variable and does not allocate any storage (address space) to variable in memory.

(b) Definition tells the space is reserved for the variable or initialized to some value.

(c) Declaration specifies both storage and type of a variable.

(d) Definition tells about the type of variable and not concerned about the space.

33. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 extern int p;
 p = 20;
 printf("%d", p);
 return 0;
}
```

(a) 20

(b) error - link time

(c) error - compile time

(d) 40

34. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 printf("%d\n", p);
 return 0;
}
int p =20;
```

(a) 20

(b) error- undeclared 'p'

(c) 0

(d) compiler dependent

35. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 extern int p;
 printf("%d", p);
 return 0;
}
int p=20;
```

(a) 20

(b) error- compile time

(c) 0

(d) compiler dependent

36. What is the output of the following program?

```
#include<stdio.h>
int main()
{
 char *a, b;
 void *p, q;
 a = "hello";
 p = a;
 q = a[2];
 printf("%s, %c", p, q);
 return 0;
}
```

(a) hello, l

(b) hello, garbage

(c) compiler dependent

(d) error- size of q is not known

37. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i;
 for(i=0; i<25; i++)
 {
  switch(i)
  {
   case 0: i+=5;
   case 1: i+=10;
   case 2: i+=6;
   default: i+=1;
    break;
  }
  printf("%d, \n", i);
```

```
 }
 return 0;
}
```

(a) 22, 23, 24, 25

(b) 22, 24, 25

(c) 22, 25, 27

(d) 22, 24

38. **What is the output of the following program?**

```
#include<stdio.h>
int fun(int i, int j)
{
 if(i/2 == 0) return 0;
 else if(i%2 == 1)
  return fun(i/2, 3*j) + i+j;
 else
  return fun(i/2, 2*j) + i+j;
}
int main()
{
 printf("%d", fun(20, 1));
 return 0;
}
```

(a) 48

(b) 49

(c) 54

(d) 56

39. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int p = printf("");
 switch(p)
 {
  case 0: printf("India ");
```

```
   default: printf("is great");
  } return 0;
 }
```

(a) India

(b) is great

(c) India is great

(d) error- compile time

40. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int p = 10;
 int q = 0;
 while(q++ < p) printf("%d, ", q);
 printf("\n");
 q = 0;
 while(++q < p) printf("%d, ", q);
 printf("\n");
 q=0;
 while(p-- > q) printf("%d, ", p);
 printf("\n");
 p=10;
 while(--p > q) printf("%d, ", p);
 return 0;
}
```

(a) 0 to 9

1 to 9

10 to 1

9 to 1

(b) infinite loop

(c) 1 to 10

1 to 9

9 to 0

9 to 1

(d) 0 to 9
0 to 9
10 to 1
10 to 1

41. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 float f = 0.7;
 if(0.7 > f)
  printf("India");
 else
  printf("Bharat\n");
 return 0;
}
```

(a) India
(b) Bharat
(c) Misplaced Else
(d) India Bharat

42. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int p = 4;
 switch(p)
 {
  case 4:
  printf("hello, ");
  case 3:
   printf("bye");
  case 6: continue;
  default: break;
 }
 return 0;
}
```

(a) hello, bye

(b) hello

(c) error- compile time

(d) undefined behavior

43. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
   int i = 5;
   int j;
   i = j = 4;
   printf("%d, ", i);
   if(j=(i=0))
    printf("India");
   else
    printf("Bharat");
   return 0;
}
```

(a) 1, Bharat

(b) 0, India

(c) 4, Bharat

(d) 4, India

44. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i;
 for(i=0; i<10;i++)
 {
  if(i<=5)
  {
   printf("%d\n", i);
   continue;
  }
  else break;
```

```
   printf("hello, ");
  }
  return 0;
 }
```

(a) 0, 1, 2, 3, 4, 5, hello,

(b) 0, hello, 1, hello, 2, hello, 3, hello, 4, hello, 5, hello,

(c) 0, 1, 2, 3, 4, 5

(d) 0, 1, 2, 3, 4, 5, hello, hello, hello, hello,

45. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=3;
 int x = ++i;
 int y = i++;
 printf("%d, %d, %d, ", x, y, i);
 printf("%d, ", i++);
 printf("%d", ++i);
 return 0;
}
```

(a) 4, 4, 5, 6, 7

(b) 3, 3, 5, 6, 7

(c) 4, 4, 5, 5, 7

(d) error- l-value required

46. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = 3>4;
 int j = 4>3;
 int k = (i=j);
 int l = (k==j);
 printf("%d %d %d %d", i, j, k, l);
 return 0;
```

```
}
```

(a) 0, 1, 0, 1

(b) 0, 0, 1, 1

(c) 0, 1, 1, 1

(d) 1, 1, 1, 1

47. **Determine the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = 1, j=-1;
 while(j)
 {
  printf("%d, %d,", i, j);
  j=-5 < i--;
 }
 return 0;
}
```

48. **What is the output of the following program?**

```
#include<stdio.h>
#define fn(v , a , r) v##a##r
int main()
{
 int var = 10;
 printf("%d" , fn(v,a,r));
 return 0;
}
```

(a) v##a##r

(b) error : v, a and r are not declared

(c) 10

(d) var

49. **What is the output of the following program?**

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

```
{
 int p=2;
 int j = p+(3, 7, 9, 10);
 printf("%d\n", j);
 return 0;
}
```

(a) 5

(b) 31

(c) 12

(d) error- syntax error

50. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x = 150;
 printf("%d, %d\n", x=40, x>=50);
 return 0;
}
```

(a) 1, 0

(b) 40, 1

(c) 40, 0

(d) compiler dependent

51. What is the output of the following program?

```
#include<stdio.h>
int main()
{
 char ch[] = "Hello India";
 printf("%s ", &ch[8]-&ch[2] + ch); //line 3
 return 0;
}
```

(a) lo India

(b) India

(c) llo India

(d) syntax error at line 3

52. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 float f = 0.4;
 if(0.4f > f) //line 2
  printf("India");
 else
  printf("Bharat\n");
 return 0;
}
```

(a) India

(b) Bharat

(c) error at line 2

(d) India Bharat

53. **What is the output of the following program?**

```
#include<stdio.h>
#define FOO(x) foo(x)
int foo(char x)
{
 return (int)x;
}
int main()
{
 char x = ‘a’;
 printf("%d",FOO(x));
 return 0;
}
```

(a) 97

(b) undefined behavior

(c) error- compile time

(d) garbage

54. **What is the output of the following program?**

```
#include<stdio.h>
int x, y;
void Q();
void P(int);
int main()
{
   x = 7, y=8;
 Q();
 printf("%d\n", x);
 return 0;
}
void Q()
{
 int x, y;
 x=3; y=4;
 P(y);
 printf("%d, ", x);
}
void P(int n)
{
 x = n+2 / n-3;
}
```

(a) 6, 3

(b) 3, 6

(c) 3, 4

(d) 3, 1

55. **What is the output of the following program?**

```
#include<stdio.h>
int r;
void two();
void one();
int main()
{
 two();
 one();
 two();
```

```
 return 0;
}
void two()
{
 printf("%d, ", r++);
}
void one()
{
 int r=5;
 two();
 printf("%d, ",r++);
}
```

(a) 1, 2, 5, 3,

(b) 0, 1, 5, 2,

(c) 0, 0, 5, 0.

(d) 1, 2, 6, 3,

56. **What is the output of the following program?**

```
#include<stdio.h>
void test(int* , int*);
int main()
{
 int a=5;
 test(&a, &a);
 printf("%d ", a);
 return 0;
}
void test(int *c, int *d)
{
 *c += *d * 2;
 *d *= *c - 3;
}
```

(a) 180

(b) 12

(c) 60

(d) 144

57. **What is the output of the following program?**

```
#include<stdio.h>
void fun(int x, int *y, int *z)
{
 *y += 4;
 *z = *z + x + *y;
}
int main()
{
 int x=10, y=3;
 fun(y, &x, &x);
 printf("%d, %d\n", x, y);
 return 0;
}
```

(a) 31, 3

(b) 32, 2

(c) 31, 2

(d) 32, 3

58. **What is the output of the following program?**

```
#include<stdio.h>
i=0, j=1;
void f(int *p, int *q)
{
 p = q;
 *p = 2;
}
int main()
{
 f(&j, &i);
 printf("%d, %d", i, j);
 return 0;
}
```

(a) 0, 2

(b) 0, 0

(c) 2, 2

(d) 2, 1

59. **What is the output of the following program?**

```
#include<stdio.h>
int f(int a, int *b, int **c)
{
 int y, z;
 **c += 4;
 z = **c;
 *b += 6;
 y = *b;
 a += 6;
 return a+y+z;
}
int main()
{
 int z, *y, **x;
 z = 4;
 y = &z;
 x = &y;
 printf("%d",f(z, y, x));
 return 0;
}
```

(a) 30

(b) 32

(c) 31

(d) 29

60. **What is the output of the following program?**

```
#include<stdio.h>
int a;
void q(int c)
{
 c-=a;
 printf("%d, ", c);
}
void p(int *b)
{
```

```
  int a = *b-2;
  q(a);
  *b = a + 1;
  printf("%d, ", a);
}
int main()
{
  a = 5;
  p(&a);
  printf("%d\n", a);
  return 0;
}
```

(a) 4, 2, 0

(b) 7, 6, 12

(c) -2, 3, 4

(d) 2, 3, 4

Other important questions for the Interview are as follows:

1. Write a C program to check whether the given number is a prime number or not?
2. Write a C program to check whether the given number is a palindrome number or not?
3. Write a C program to check whether the given string is palindrome or not?
4. Write a C program to print Pascal triangle.
5. Write a C program to print the ASCII value of all characters.
6. Write a C program to reverse any number.
7. Write a C program to add two numbers without using the addition operator.
8. Write a C program to subtract two numbers without using the subtraction operator.
9. Write a C program to find out the prime factor of a given number.
10. Why does C treat array parameters as pointers?
11. How to convert string to an integer without using library functions in C.

12. Write a C program to print 1 to n without using a loop.
13. Write a C program for swapping two numbers using function swap().
14. How will you print "Conceptual C questions" without using a semicolon?
15. Write a C program that splits numbers into digits.

Group-1 Explanations

1. **c**

   Explanation: When main gets called by the Operating System, the stack area layout of the C program is as follows:

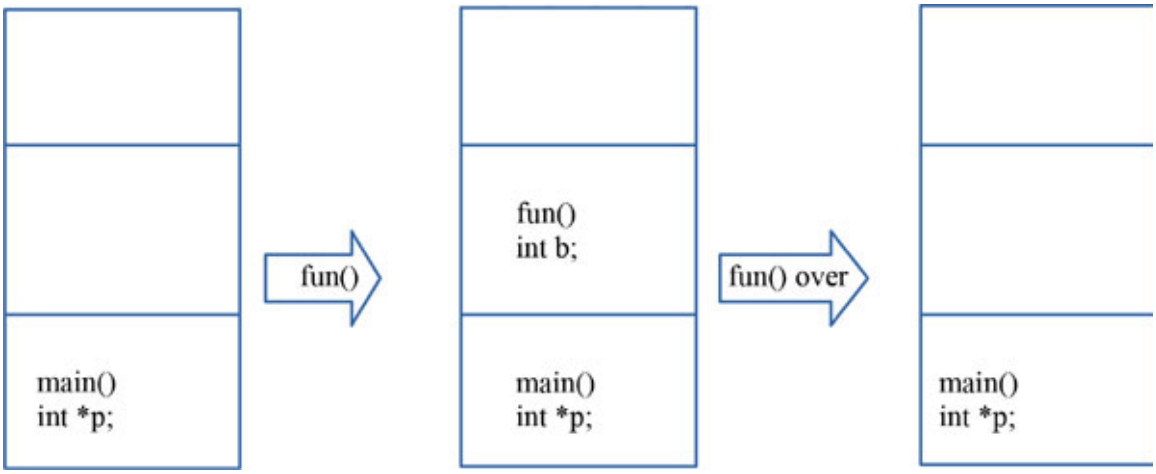


Figure 2.1: *Demonstrating stack area of the given C program*

   - When the execution of fun() gets completed, the addresses assigned to the fun() also get destroyed and the space allocated to fun() in the stack area is available for the next function call.
   - Hence, the address of variable ‘b’ is no more liable after the completion of fun(). If we try to access ‘b’, there is no guarantee that the value assigned to ‘b’ is still present or not. Sometimes it may be there or some time it is overwritten by other function calls. This depends on the implementation of the compiler.
   - Like in Turbo C, it gives an error while in the gcc compiler it gives just a warning.

2.

Explanation: Using static memory allocation The memory for 'b' is getting allocated in data area and now the fun() is returning the address, that will be available even after the completion of fun(). It will not throw any error.

As we discussed in "A touch to C" chapter the data area is sharable among all the functions.

```
int *fun()
{
  static int b = 10;
  return &b;
}
```

Using dynamic memory allocation Here, the memory for 'b' is getting allocated dynamically that resides in heap area. Now the variable 'b' holds the address of memory from heap area.

As we already discussed, the heap area is also shared among all the functions.

```
int *fun()
{
  int *b = (int *)malloc(sizeof(int));
  return b;
}
```

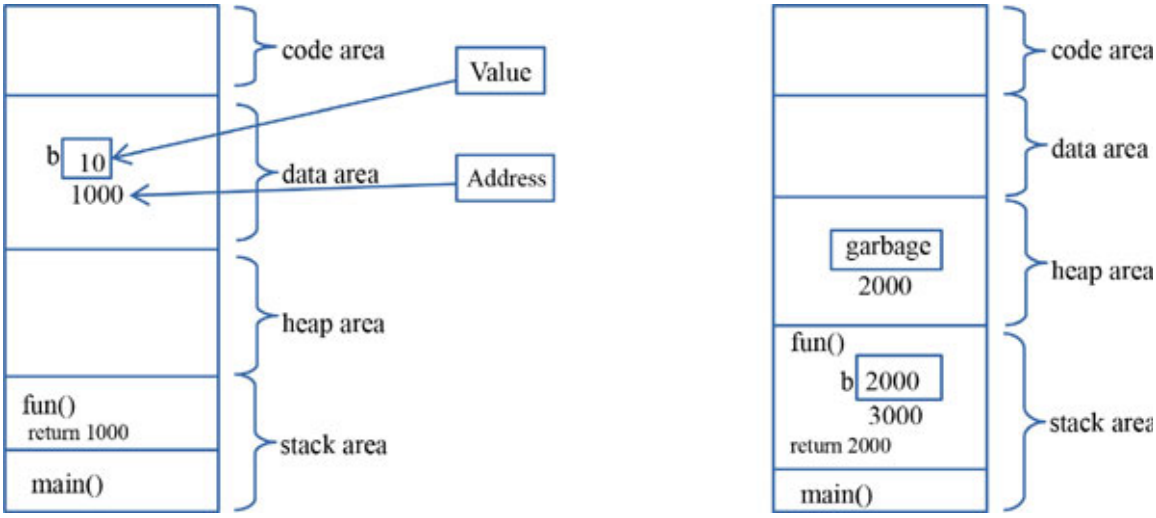


Figure 2.2: *Memory layout of C program*

3. **b**

Explanation: The *void* pointer is a generic pointer type. It can hold addresses of all data types. It is used when the actual data type of a

pointer is not known for the time, and later it may be known. In the given code we are using a void pointer that is holding addresses of an integer and character type.

4. **c**

   Explanation: The *void* data type is not allowed because the size of the *void* data type is not known. It cannot be used as a generic data type.

5. **b**

   Explanation: The memory for malloc is allocated dynamically in heap area using ‘brk’ (keyword) system call. As we can see in figure (i) brk is pointing to the free (unallocated) pool of memory. When we call the malloc() function, the brk pointer allocates memory of 4B to ‘p’ and itself get shifted right, and (ii) also assigns a value 10 to the allocated space.

   Now we are doing free(p), (iii) the memory allocated by malloc get free and its content becomes zero or NULL, and brk return to its previous location. We can notice that ‘p’ is still pointing to the same address (figure iii). We again did malloc() to allocate memory for ‘q’ (figure iv). If we try to access the content of pointer ‘q’, it returns 15 as expected. Suppose if we access the content of pointer ‘p’, it also returns the same content as ‘q’. This was not expected from ‘p’ as we have already freed the ‘p’, but what actually happened is that we freed the memory from the heap area, but p still points to a memory location/address, as p is just a pointer variable, this ‘p’ is called dangling pointer.

   The dangling pointer in the program can be avoided by making p = NULL just after the free(p) statement.

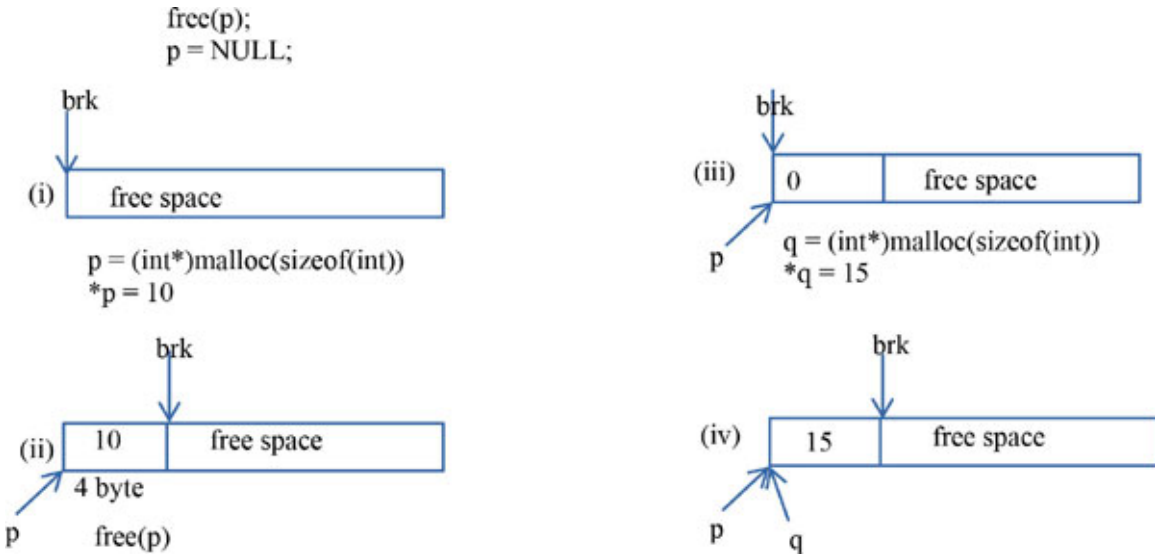


Figure 2.3: *Demonstrating brk system call during allocation of dynamic memory in heap area.*

6. **b**

Explanation: The memory to 'p' gets allocated using malloc() in main(), the memory is accessible to other functions also, here in fun(), we are modifying the shared memory value pointed by pointer 'p', which get updated with value 20. (look at the below figure)

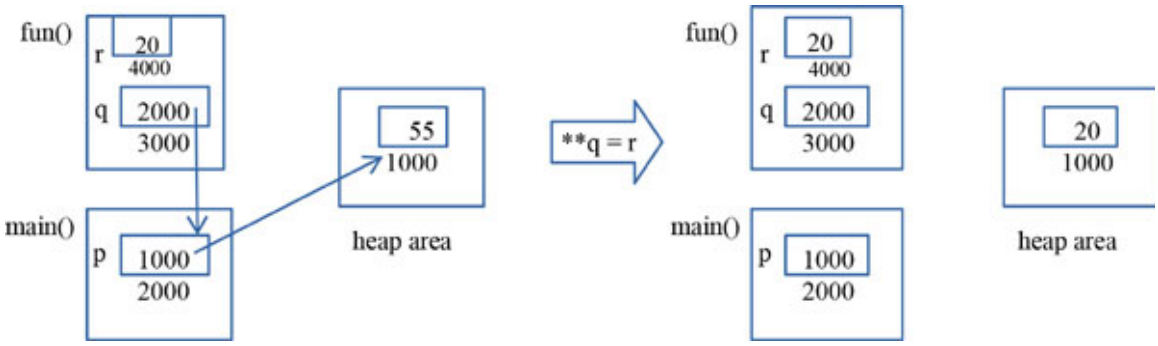


Figure 2.4: *Demonstrating pointer and variable allocation in heap and stack area*

7. **d**

Explanation: Wild Pointer: These pointers are not initialized and point to some random location, e.g. 'p'.

Dangling Pointer: Check the explanation of question 5, there is no dangling pointer in the given program.

NULL Pointer: These pointers are initialized to a NULL value that is not pointing to any location, e.g. 'r'.

Memory Leak: When the memory is allocated dynamically using *malloc()* in some function and the memory is not used by any other function in a program. And also, the memory is not getting deallocated using free() in that function. After the execution of that function, it is not possible to access that memory because we have lost the pointer variable (‘q’) where we stored the address of allocated memory, this memory is remains unused. This unused memory is called a memory leak, fun() is doing memory leak.

8. **c**

   Explanation: When a character pointer is assigned to a string. The string gets stored in the text area or code area, this area is only readable. At line 3 in a given program, we are trying to update the readable area. Hence, the run time error occurs. Have a look at a figure.

Figure 2.5: *Representation of memory allocation of string in text area and stack area*

   The "hello" pointed by ‘p’ is stored in the code/text area (read-only). It cannot be updated. The variable ‘p’ itself is present in the stack area. The "hello" pointed by ‘q’ is stored in the stack area (local block), which can be modified.

   printf("%s", p);// %s takes address (that is 1000 in our case), and start printing all the characters until it encounters a null character ‘\0’. In a program, we are trying to update the read-only area. Run time error will be thrown at line 3.

9. **a**

   Explanation: If a string gets stored in the code area (read-only), the compiler searches whether the same string is already available or not. If it is available, the compiler does not allocate new memory for the string but it returns the address of the string that is available. If the string is not available, the compiler allocates new memory in the code area. In a given program, "hello" already exists in the code area and it is pointed by the ‘p’ pointer. When again the same string comes the compiler returns the same address from code/text area. This is called memory reusability.

10. **b**

Explanation: The *sizeof()* operator returns the number of bytes consumed by its operand. Here, ‘p’ is a character

pointer, which needs only 1 byte for storing the value. Hence, *sizeof(*p)* returns value 1.

Since ‘p’ is a pointer whose size depends on the number of address lines in a machine, as we discussed in the introduction, the size of the pointer variable is 4 bytes. Hence *sizeof(p)* returns 4. The *sizeof(q)* returns 6, 5 characters and one termination (null) character (‘\0’). Each string gets suffixed with a termination character (‘\0’).

The *strlen()* in a library function, present in the *string.h* header file counts the number of characters in a given string and it doesn’t count the ‘\0’ (null character).

11. **c**

    Explanation: *"(void*)0"* is not a void pointer, void pointer is *"void *p;"*. For (ii), (iii), (iv), (v) explanation not required.

12. **a, c- true; b, d- false;**

    Explanation: Not needed.

13. **c**

    Explanation: Character pointer is meant to point only one byte at a time and here character pointer ‘j’ is pointing to 4-byte float variable ‘a’. When the character pointer is accessed, it returns the first byte of float variable ‘a’.

14. **c**

    Explanation: Constant value cannot be modified. In line 2, we are doing so, hence, the compiler throws an error.

15. **c**

    Explanation: *‘||’ operator:* It first checks the value of the left-hand side of the expression if it is true then it does not evaluate the right-hand side expression and returns 1. If the left-hand side expression is false, then only it evaluates the right-hand side expression.

    ‘&&’ operator: It first checks the value of left-hand side expression if it is true then only it evaluates right-hand side expression, if it is false then it does not evaluate right-hand side expression and return 0.

16. **d**

Explanation: *sizeof()* operator does not evaluate its operand, it only checks the data type of its operand. Hence, *sizeof(p++)* or *sizeof(++p)* does not make any change in value of variable ‘p’.

17. **b**

    Explanation: In a given program a macro A is referred to as 2. In line 2, macro B and A are compared by the ‘&&’ operator but macro B is not defined yet. If the macro is not defined before its use then it takes value 0 by default. The if condition at line 3 gets false and hence macro C is referred to as value 4 at line 6.

18. **c**

    Explanation: It’s a simple recursive program, explanation is not required.

19. **d**

    Explanation: First of all macro *int* gets replaced by *char.* The preprocessed program has *char i = 65*, instead of *int i = 65 sizeof()* operator at line 3 returns the size of character data type that is 1.

    In C, all character literals are treated as integers. That’s why the *sizeof the ()* operator in line 4 will return the size of integer data type. Hence, 4 will get printed.

20. **c**

    Explanation: The *default case* can be placed anywhere inside the *switch*. It is executed when all other cases don’t match. If any of the cases match with the switch condition, all subsequent cases and default (if it’s there) will execute until the break is encountered or the end of the switch.

21. **c**

    gcc: garbage, garbage

    turboC: 700, 500

    Explanation: It is a compiler-dependent program.

22. **d**

    Explanation: The *case* label must be a constant or constant expression. The variable/non-constant expression is not allowed in the *case* label. The enumerated type can also be used as a *case* label statement.

23. **c**

Explanation: As we discussed in the above explanation, the constant expressions are allowed as a *case* label. Here, the error is due to duplication in the *case* label.

24. **c**

    Explanation: The variable ‘i’ is visible to the inside block only. But we can access its value through address out of the block because the lifetime of the address of a block in a main exists till the exit of the main function.

25. **c**

    Explanation: The variable ‘p’ is a constant integer and stored in the read-only memory area. Its value cannot be updated. In our program, we are trying to update the value of ‘p’ at line 2. Hence, the compiler will throw an error.

26. **b**

    Explanation: In the gcc compiler, the lexical (phase of compilation) scanning of symbols starts from left to right. The operator (++, +, -, etc) which comes first will be treated as the first symbol. Here, i+++j can be treated in two ways, either *(i++)+j* or *i+(++j)*. The first operator is ‘+’ which comes in lexical scanning but just after ‘+’, the ‘+’ symbol again encounters and now the first operator becomes ‘++’.

    It associates the ‘++’ operator with i, the second operator is ‘+’, which is a binary operator and applied to both operands. The second way of expressing the given expression *i+(++j)* is discarded.

27. **d**

    Explanation: The compiler will throw an error at line 5, the *sizeof()* ‘*p’ and ‘*q’ are not known due to their void nature. If we do typecasting of ‘p’ and ‘q’ to any data type pointer, like **(int *)p* and **(int*) q*, it will work fine.

28. **b**

    Explanation: The same version of this program is already discussed in the explanation of question 10.

29. **c**

    Explanation: The *void* pointer is a generic pointer type, arithmetic operations are not allowed on void pointers. The void pointers are generally used for passing void pointers to functions, after passing to function, type conversion can be done. The void pointers are also referred

to as intermediate pointer types. (check explanation of question 3 and question 4).

30. **c**

    Explanation: The *scanf* returns the total number of valid inputs accepted by it, and *printf* returns the number of characters printed by it on the display. In a given program, *scanf is taking* two valid integer inputs, so ‘q’ is 2 and inside *printf* will print "India" and return 5. Outside printf will print 2+5 i.e. 7. Hence, the output will be "India7".

31. **d**

    Explanation: The *scanf* will return 0 if the non-formatted value is entered as input. In a given program we are formatting *scanf* with ‘%d’ at line 2. So, scanf is expecting integer type value, with integer value, "while" will become infinite loop "while(1)", but if character or any other data type value is given, scanf will return 0, and while loop will get terminated.

32. **a- true, b- true, c- false, d- false**

    Explanation: Not required.

33. **b**

    Explanation: The *extern* is just as a declaration and expect the definition of the variable later in the program (linking phase). Its resolution takes place during the scanning of symbols, and symbol table creation. The *extern* specifies that the definition for the *extern* type variable is present in some other programs/library or in the same program as a global variable. The given program compiles correctly but when the resolution of ‘p’ comes, it checks in other associated files, library files for the definition of variable ‘p’. But it would not find any definition of ‘p’ because it is just a variable name, not any keyword. Hence link-time error gets thrown by the compiler. *Linker Error : Undefined symbol ‘p’*.

34. **b**

    Explanation: The **‘**p’ is not an *external* variable, so the compiler will check from the symbol table and throws an error that ‘p’ is not declared. If we use *extern* it means that compiler skips and waits till the linking phase for resolving p.

35. **a**

Explanation: It works fine, the resolution of ‘p’ will not take place at compile-time, it takes place during link time. At link time, the definition of ‘p’ is present in the symbol table. Hence, the value of ‘p’ can be directly resolved during link time.

36. **d**

    Explanation: It’s a concept of the void pointer, we already discussed it in explanation of questions 3, 4, 29.

37. **d**

    Explanation: One more flavor of switch statement we have discussed in the explanation of questions 20 and 22.

38. **d**

    Explanation: A simple recursive program, explanation not required.

39. **c**

    Explanation: The *printf* returns the number of characters printed. Here, *printf* is accepting a zero-length string "", hence it returns 0. The output will be "India is great".

40. **c**

    Explanation: A simple program with few while loops, explanation not required.

41. **a**

    Explanation: Default decimal number is *double* in C. We know *double* is more precise, so *float* ‘f’ may hold 0.69999999 which is less than 0.7. Hence "India" gets printed. For *real numbers* like *float*, *double*, and *long double* the exact value cannot be predicted.

    Note: Be cautious while comparing *floating point* numbers with relational operators such as ‘== ‘,’ >’, ‘<’, ‘<=’, ‘>=’, etc.

42. **c**

    Explanation: The compiler throws an error due to the *continue* statement. The *continue* can only be used in loops.

43. **c**

    Explanation: The assignment operator is right to left associative. It returns the right-hand side evaluated expression.

    Given Expression *i = (j = 4)* can be evaluated as; *i = 4*; Since, *(j=4)* returns 4.

44. **c**

    Explanation: The *continue* statement skips all the subsequent statements inside the loop and start execution from the beginning of the loop. The *break* statement skips all the subsequent statements inside the loop and control goes out of the loop.

45. **c**

    Explanation:

    pre increment:

    x = ++i; can be decompose into two statements:

    i = i+1;

    x = i;

    post increment:

    y = i++; can be decompose into two statements:

    y = i;

    i = i+1;

46. **d**

    Explanation: The precedence of the relational operator is more than the assignment '=' operator. Hence, the resolution of '>' happens first, x > y returns 1; if 'x' is greater than 'y', else 0. See precedence table in "A touch to C" chapter.

47. **1, -1, 0, 1, -1, 1, -2, 1, -3, 1, -4, 1, -5, 1,**

    Explanation: The expression *j = -5<i--;* can be written in two step instruction: (i) *j = -5 < i;* (ii) *i = i - 1;*

    Now, it is easy to get answer for the given program.

48. **c**

    Explanation: The '##' symbol in macro is used to concatenate the string on which it is applied. In the given program,

    '##' symbol is applying on 'v', 'a', and 'r'. Preprocessor substitutes *fn(v,a,r)* with 'var' in main() function.

49. **c**

    Explanation: The comma operator has left to right associativity. In L-R associativity, left is first evaluated and then right. Here, the evaluation

takes place in order 3, 7, 9, 10. So 3, 7, 9 gets discarded and the parenthesis operator returns 10.

50. **d**

Explanation: In *gcc* and T*urbo C* the function call passes parameters from right to left, the given program outputs option b. And for the online compiler, it can give option c also. Hence, we can say it is compiler-dependent.

51. **b**

```
printf("%s ", &ch[8]-&ch[2] + ch);
→ 1009-1003+1000 = 1006
printf("%s",1006);
```

Output: India

```
The behavior of %s has been already discussed in explanation
of question 8.
```

Explanation: ch[] = "Hello India";

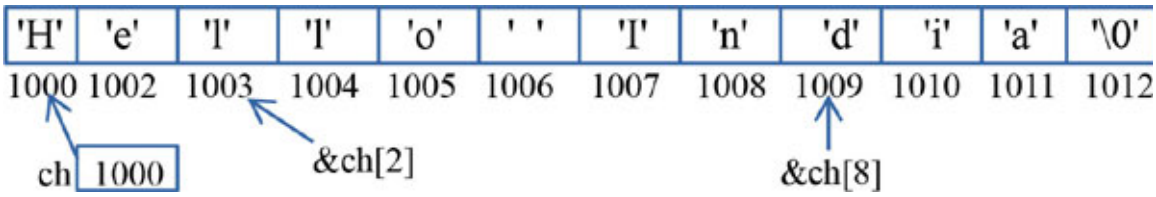


Figure 2.6: *Representation of memory allocation of string in stack area*

52. **b**

Explanation: As we know that in the standard C compiler the real constant value has double data type by default but if we suffixed any constant real number with ‘f’ like o.4f, it becomes equivalent to 0.399999 in float form.

53. **a**

Explanation: The *macros* can also be used with the function name.

54. **d**

Explanation: Take care of *global, local* variables and *precedence* of operators in function P().

55. **b**

Explanation: If *global* variables are not initialized, they automatically get initialized with 0.

56. **a**

Explanation:

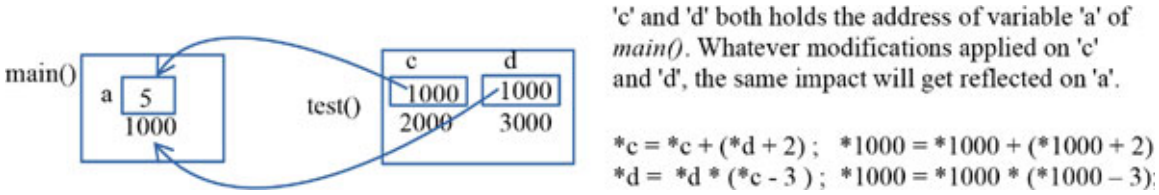


Figure 2.7: *Pointer representation in stack area*

57. **a**

Explanation: Refer to explanation of *question 56.*

58. **d**

Explanation:

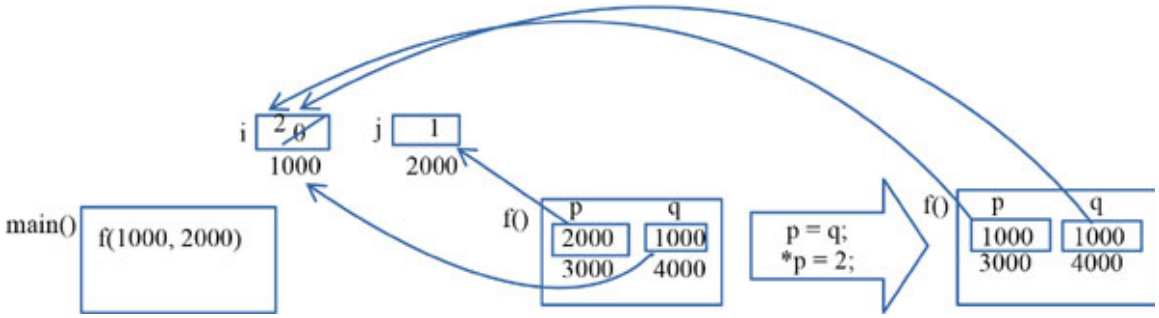


Figure 2.8: *Pointer representation in stack area*

'i' and 'j' are *global* variables (available to all functions). In function f(), initially 'p' was pointing 'j' and 'q' was pointing 'i' but after *p = q* statements both pointers starts to point 'i', 'i' get updated by 2 after **p = 2* statement.

59. **b**

Explanation: It is following the concept of call by reference. Kindly refer explanation of *question 58.*

60. **c**

Explanation: It is following the concept of call by reference. Kindly refer explanation of *question 58.*

Don't forget why you started it.

Group-2 Questions

Number of 'C' Question: 60

Other Important Interview Questions: 15

1. **What is the output of the following program? Assume the base address of a given array ‘a’ is 1000.**

```
#include<stdio.h>
int main()
{
 int a[3][3] = {4, 5, 6, 7, 8, 9, 1, 2, 3};
 printf("%u %u %u", a[1]+2, *(a+1)+2, *(a[1]+2));
 return 0;
}
```

(a) 1000 1008 6

(b) 1008 1012 7

(c) 1012 1016 8

(d) 1016 1016 9

2. **What is the output of the following program?**

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

```
int f(int n)
{
 while(--n>=0)
 {
  printf("%d ", n-2);
 }
 return 1;
}
int main()
{
 int (*p)(int) = f; //line 1
 (*p)(6); //line 2
 return 0;
}
```

(a) error in line 1

(b) error in line 2

(c) 3 2 1 0 -1 -2

(d) 3 2 1

3. **What is the output of the following program?**

```
#include<stdio.h>
int f(int *a, int n)
{
 if(n<=0) return 0;
 else if(*a%2 == 0)
  return *a + f(a+1, n-1);
 else
  return *a - f(a+1, n-1);
}
int main()
{
 int a[] = {12, 7, 13, 4, 11, 6};
 printf("%d", f(a, 6));
 return 0;
}
```

(a) 17

(b) 21

(c) 19

(d) 15

4. **What is the output of the following program?**

```
#include<stdio.h>
void demo()
{
 printf("xx");
}
int main()
{
 void demo();
 void (*fun)();
 fun = demo;
 (*fun)(); //line 4
 fun(); //line 5
 return 0;
}
```

(a) xxxx

(b) yyyy

(c) xx

(d) error- compile time

5. **What is the significance of prefixing extern before the function declaration?**

e.g.

extern int sum(int x, int y, int z)

```
{
 return x+y+z;
}
```

6. **What is the significance of prefixing static before the function declaration?**

```
#include<stdio.h>
static int sum(int x, int y, int z)
{
 return x+y+z;
}
```

7. **What is the output of the following program?**

```
#include<stdio.h>
void fun(int, int*);
void fun(int x, int *y)
{
 int i=0, j=0;
 if(x==0) return ;
 i = x %10;
 j = x/10;
 *y += i;
 fun(j, y);
}
int main()
{
 int p=0 ;
 fun(1024, &p);
 printf("%d ", p);
 return 0;
}
```

(a) 3

(b) 5

(c) 6

(d) 7

8. **What is the output of the following program?**

```
#include<stdio.h>
int fun(int p)
{
 static int count = 0;
 count += p;
 return count;
}
int main()
{
 int i, j;
 for(i=0; i<=4; i++){
  j = fun(i);
```

```
  }
   printf("%d",j);
  return 0;
```

(a) 4

(b) 9

(c) 10

(d) 11

9. **Which of the following statements is/are true or false about the given function?**

```
int fun(int p)
{
 int i = 50;
 if(i == p)
 {
  printf("hi\n");
  int k = fun(p);
  return 5;
 }
 else
  return 0;
}
```

Statements:

(i) The function returns zero for all values of p except 50.

(ii) The function prints the "hi" infinite time for all value of p except 50 till the stack gets full.

(iii) The function return 5 when p = 50.

(iv) The function prints the "hi" infinite when p = 50 till the stack gets full.

Which of the above are true?

(a) (i), (ii)

(b) (iii), (iv)

(c) (i), (iv)

(d) (iv)

10. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int n=0, m=0;
 if(n>0) //line 2
  if(m>0) //line 3
   printf("true");
 else //line 5
  printf("false");
 return 0;
}
```

(a) false

(b) true

(c) nothing will get printed

(d) error- compile time

11. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 if(sizeof(int) > -5) //line 1
  printf("True\n");
 else
  printf("False");
 return 0;
}
```

(a) True

(b) False

(c) error- compile time

(d) compiler dependent.

12. **What is the output of the following program? Assume the base address of a given array ‘a’ is 1000.**

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

```
{
 int a[][3] = {4, 5, 6, 7, 8, 9, 1, 2, 3};
 printf("%u, %u, %u", *a[2], a[2][0], **(a+1+('b'-'a')));
 return 0;
}
```

(a) 1024 1 1

(b) 1024, 1024, 1024;

(c) 1 1 1

(d) 1024, 2, 1024

13. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=3, j=0, l; //line 1
 int k;
 k = j = l; //line 3
 l = i && j;
 printf("%d\n", l);
 return 0;
}
```

(a) 0

(b) 1

(c) undefined behavior

(d) error- compile time

14. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = 3;
 int j = i+(i=10); //line 2
 printf("%d\n", j);
 return 0;
}
```

(a) 13

(b) 20

(c) 30

(d) compiler dependent

15. **What is the output of the following program?**

```
#include<stdio.h>
void print(int i)
{
 if(i<5) return;
 printf("%d ", i);
 print((i--, i)); //line 3
}
int main()
{
 print(10);
 return 0;
}
```

(a) 10 9 8 7 6

(b) 10 9 8 7 6 5

(c) 10 9 8 7

(d) syntax error in line 3

16. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = 1;
 int j = 2;
 if(i & j == 1)
 {
  printf("true");
 }
 else
  printf("false");
 return 0;
}
```

(a) true

(b) false

(c) nothing get printed

(d) error- compile time

17. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=2, j=3;
 int k = i&j == 3;
 printf("%d\n", k);
 return 0;
}
```

(a) 0

(b) 1

(c) 2

(d) 3

18. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=3, j=2;
 int k = i << 1 > 5;// line 2
 printf("%d\n", k);
 return 0;
}
```

(a) 0

(b) 1

(c) 5

(d) 3

19. **What is the output of the following program?**

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

```
 int x=3;
 const int *p = &x;
 *p++;
 printf("%d", *p);
 return 0;
}
```

(a) 4

(b) 3

(c) garbage value

(d) error- compile time

20. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=2, j=2;
 int k = i^j&1;//line 2
 printf("%d", k);
 return 0;
}
```

(a) 0

(b) 1

(c) 2

(d) garbage value

21. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=2, j=0;
 int k = i&&j = 1; // line 2
 printf("%d\n", k);
 return 0;
}
```

(a) 0

(b) 1

(c) 2

(d) error- l-value required.

22. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=0, j=2;
 if(!i && j)
  printf("true");
 else
  printf("false");
 return 0;
}
```

(a) true

(b) false

(c) error- compile time

(d) undefined behavior

23. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = 0, j = 2;
 int k = ~i & j;
 printf("%d\n", k);
 return 0;
}
```

(a) 2

(b) 1

(c) 3

(d) 4

24. **What is the output of the following program?**

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

```
{
 int y = 6;
 int z = 7;
 int x = ++y + z--;
 printf("%d\n", x);
 return 0;
}
```

(a) 13

(b) 15

(c) 14

(d) syntax error

25. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[3] = {5, 6, 7};
 char *c = (char *)a;
 printf("%d, %d, %d, %d ", *c, c[0], c[1], c[2]);
 return 0;
}
```

(a) 0, 5, 0, 7

(b) 5, 5, 7, garbage

(c) 5, 5, 0, 0

(d) 5, 5, 0, garbage

26. **What is the output of the following program?**

```
#include<stdio.h>
#define FOO(x) foo(#x) //line 2
int foo(char *x)
{
 printf("%c %c", *x,x[1]); //line 4
}
int main()
{
 FOO(65); //line 8
```

```
 return 0;
}
```

(a) 5 6

(b) 7 8

(c) 6 5

(d) error- compile time

27. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int main = 3;
 printf("%d\n", main);
 return 0;
}
```

(a) garbage

(b) address of main

(c) 3

(d) error- multiple declaration of main

28. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = 1;
 for(i=1 ; i <= 4 ; i++)
  switch(i){
   case 1: printf("%d",i);break;
   {
    case 2:printf("%d",i);break;
    case 3:printf("%d",i);break;
   }
   default: printf("%d",i);
  }
 return 0;
}
```

(a) 123

(b) 1234

(c) 1

(d) error- syntax error.

29. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 double ch = 0.45;
 switch(ch)
 {
  case 1:
   printf("hello ");
  case 0:
   printf("hi");
 }
 return 0;
}
```

(a) hello

(b) hi

(c) hello hi

(d) error

30. **Figure out invalid declaration of the given "if" statements.**

(a) `if(if(p==1)){}`

(b) `if(fun(p)){}`

(c) `if(p){}`

(d) `if((char)p){}`

31. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a=1;
 if(a)
  printf("All is well, ");//line 3
```

```
  printf("I am well\n");//line 4
 else
  printf("I am not well");
  return 0;
}
```

(a) All is well, I am well

(b) I am not well

(c) I am well

(d) error- compile time

32. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
#define strcpy(x , y) x = y
int main()
{
 int i;
 char s[10];
 strcpy(i , 10); //line 4
 strcpy(s , "hello"); //line 5
 printf("%d %s",i,s);
 return 0;
}
```

(a) 10 hello

(b) error in line 4 : Cannot use strcpy() function with integer

(c) error in line 5

(d) error in macro definition : cannot define macro for inbuilt function.

33. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[3][3] = {4, 5, 6, 7, 8, 9, 1, 2, 3};
 int *p = &a[2];
 int *q = a[2];
 printf("%d %d", *p, *q);
 return 0;
```

```
}
```

(a) 1 2

(b) 8 9

(c) 8 8

(d) 1 1

34. **Evaluate the variables b, c, d, e, f, g, i, j?**

```
#include<stdio.h>
int main()
{
 int b = 5-4+2*5;
 int c = 5&4&6;
 int d = 5&4 | 6;
 int e = 5+7*4-9*(3, 2);
 int h=8;
 int f = (h++, h++);
 int g = h++ + h++ + h++;
 int i = 7*8+6*7<6 ? 7:6;
 int j = 2+3-4+8-5%4;
 return 0;
}
```

35. **What is the output of the following program?**

```
#include<stdio.h>
#define type_char char*
int main()
{
 type_char a , b; // line 1
 char c[10] , d[10];
 strcpy(c , "hello");
 strcpy(d , "world");
 a = c;
 b = d; //line 6
 printf("%c %c" , a[2],b[2]); //line 7
 return 0;
}
```

(a) l r

(b) llo rld

(c) error- compile time

(d) error – run time

36. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int n=3, i=0;
 do
 {
  n = n++;
  i++;
 }while(i!=3);
 printf("%d\n",n);
 return 0;
}
```

(a) 3

(b) 5

(c) 6

(d) 7

37. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x = 65;
 switch(x)
 {
  case ‘A’:
   printf("yes");
   break;
  case 65:
   printf("no");
   break;
 }
 return 0;
```

```
}
```

(a) yes

(b) no

(c) yes no

(d) error- compile time

38. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[3][3] = {5, 6, 7, 8, 9, 10, 1, 2,3};
 int *p = &a[2];
 int *q = a[2];
 printf("%d %d", p[1], q[2]);
 return 0;
}
```

(a) 6 7

(b) 8 9

(c) 2 3

(d) 1 2

39. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=0;
 for(i++; i==1; i=2)
  printf("Inside loop ");
 printf("Outside loop");
 return 0;
}
```

(a) Inside loop Outsuide loop

(b) Outside loop

(c) Infinite time Inside loop

(d) Syntax error in for loop statement

40. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 do
  printf("Inside loop ");
 while(0);
  printf("Outside loop");
 return 0;
}
```

(a) Inside loop

(b) Outside loop

(c) Inside loop Outside loop

(d) syntax error in do-while loop.

41. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *s = "";
 do
 {
  printf("hello");
 }while(s);
 return 0;
}
```

(a) hello

(b) hello is printed infinite time

(c) hello hello

(d) syntax error in do-while loop

42. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=0, p = 0;
```

```
 while(i<10)
 {
  i++;
  printf("hi ");
  while(i<8)
  {
   i++;
   printf("hello ");
  }
 }
 return 0;
}
```

(a) hi-10 times, hello-7 times

(b) hi-10 times

(c) hi hello-10 times

(d) hi-1 time, hello-7 times, hi-2 times

43. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x=0, y=0;
 while(x < 4, y < 7)
 {
  x++;
  y++;
 }
 printf("%d, %d\n", x, y);
 return 0;
}
```

(a) 4, 7

(b) 4, 4

(c) 7, 7

(d) syntax error in while loop

44. **How many times "hi" will get printed in the following program?**

```
#include<stdio.h>
int main()
{
 int i=0, j=0;
 for(i=0; i<3; i++)
 {
  for(j=0; j<3; j++)
  {
   if(i>1)
    continue;
   printf("hi\n");
  }
 }
 return 0;
}
```

(a) 7 times

(b) 3 times

(c) 6 times

(d) 14 times

45. **How many "Inside loop" will get printed in the following program?**

```
#include<stdio.h>
int main()
{
 int x = 0;
 do
 {
  x++;
  if(x == 3)
   continue;
  printf("Inside loop");
 }while(x<4);
 return 0;
}
```

(a) 2 times

(b) 4 times

(c) 3 times

(d) 12 times

46. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=0, j=0;
 for(; i<3; i++)
 {
  for(; j<4; j++)
  {
   printf("hi");
   break;
  }
  printf("bye, ");
 }
 return 0;
}
```

(a) hibye, hibye,

(b) hibye, hibye, hibye,

(c) hihihihibye,

(d) error: break cannot be used without if or switch.

47. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x = 0;
 while(x < 2)
 {
  if(x == 1)
   break;
  x++;
  if(x == 1)
   continue;
  printf("Inside loop ");
 }
```

```
 printf("Outside loop\n");
 return 0;
}
```

(a) Inside loop Inside loop Outside loop

(b) Inside loop Outside loop

(c) Outside loop

(d) Inside loop

48. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=3, j=5;
 if(i==3)
  goto label;
 while(i<j)
 {
  printf("hi ");
  label: printf("%d ", i);
  i++;
 }
 return 0;
}
```

(a) 3 hi 4 hi

(b) 3 hi 4

(c) 3

(d) 3 4 5

49. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=0, j;
 label: printf("hi ");
 if(i == 0)
  goto label;
```

```
 return 0;
}
```

(a) hi

(b) infinite time hi get printed

(c) hi hi

(d) error- label cannot be placed before goto

50. **What is the output of the following program?**

```
#include<stdio.h>
void fun()
{
 label1: printf("3");
}
int main()
{
 printf("1 ");
 goto label1;
 printf("2");
 return 0;
}
```

(a) 1 3 2

(b) 1 2 3

(c) 3 2 1

(d) error- compile time

51. **What is the output of the following program?**

```
#include<stdio.h>
void fun1()
{
 printf("hello");
}
int main()
{
 void fun1();
 void fun2()
 {
```

```
   fun1();
  }
  fun2();
  return 0;
 }
```

(a) hello hello

(b) hello

(c) hello hello hello

(d) compiler dependent

52. **What is the output of the following program? Assume base address of an array ‘a’ is 1000, Integer size is 4B.**

```
#include<stdio.h>
int main()
{
   int a[2][2][3] = {
    { 5, 6, 7,
     8, 9, 10
    },
    {
     1, 2, 3,
     11, 12, 13
    }
     };
   printf("%u, %u, %u, %u", a[0], a[0][1], **a[1], *
   (*a[1]+2));
  return 0;
}
```

(a) 1000, 1004, 1, 3

(b) 1000, 1012, 1 1

(c) 1000, 1004, 3, 1

(d) 1000, 1012, 1, 3

53. **What is the output of the following program?**

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

```
{
 int a[2][2][3] = {5, 6, 7, 8, 9, 10, 1, 2, 3, 11, 12, 13};
 int *x = &a[1][1][2];
 int *y = (int *)a;
 printf("%d %d", *x, *y);
 return 0;
}
```

(a) garbage garbage

(b) 5 garbage

(c) 13 5

(d) 12 13

54. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[2][2][3] = {5, 6, 7, 8, 9, 10, 1, 2, 3, 11, 12, 13};
 int *x = (int *)a[1][2];
 int y = a[2][1][0];
 printf("%d %d", *x, y); return 0;
}
```

(a) garbage garbage

(b) 1 garbage

(c) 11 5

(d) 12 13

55. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[5] = {5, 6, 7, 8, 9};
 char *p;
 p = (char *)a;
 printf("%u, %u, %u", *p, *(p+12), *((int *)p+2)); //line 4
 return 0;
}
```

(a) 5, 5, 7

(b) 5, 8, 7

(c) 5, garbage, 7

(d) 5, address of 8, 8

56. **Determine the size of a given array ‘a’ in a given program in constant time?**

```
#include<stdio.h>
int main()
{
 int a[] = {4, 5, 6, 7, 8, 9, 10, 2, 3, 1, 11, 12, 13, 19,
 18, 17};
 return 0;
}
```

57. **Compare str1, str2, and str3 and figure out the differences among all in a given program.**

```
#include<stdio.h>
int main()
{
 char *str1[] = {"hello", "India", "love", "great"};
 char str2[5][5] = {"hello", "India", "love", "great"};
 char **str3;
 return 0;
}
```

58.

```
#include<stdio.h>
int main()
{
 char *str[] = {"hello", "India", "love", "great"};
 char **ptr[] = {str, str+2, str+1, str+3};
 char ***p;
 p = &ptr;
 p += 2;
 printf("%s %s", *ptr[1], **p);
 return 0;
}
```

(a) hello India

(b) India India

(c) love India

(d) India love

59. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *str[] = {"hello", "India", "love", "great"};
 char **ptr[] = {str, str+2, str+1, str+3};
 char ***p;
 p = ptr;
 p += 3;
 printf("%s %s", (**p)+2, *ptr[1]+’b’-’a’ );
 return 0;
}
```

(a) eat ello

(b) ve ndia

(c) llo ndia

(d) eat ove

60.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 while(1)
 {
  int *p = (int *)malloc(sizeof(int));
  *p=10;
  printf("hello%d", *p);
 }
 return 0;
}
```

(a) "hello10" gets printed infinite time

(b) stack overflow

(c) heap overflow

(d) error- compile time

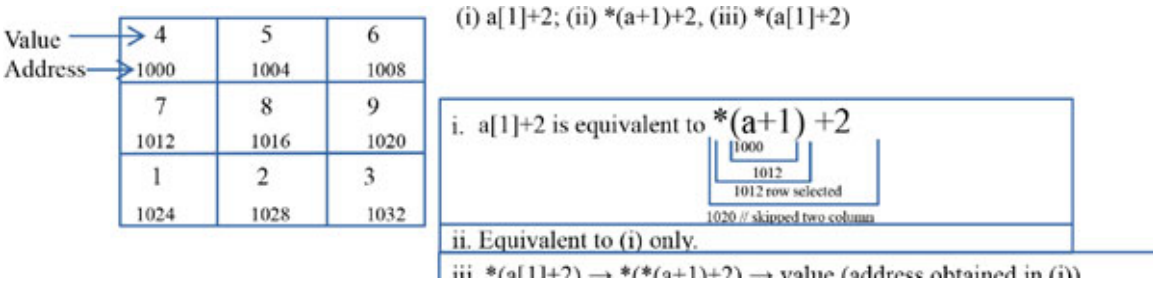
Other important questions for the Interview are as follows:

1. Write a C program to count the number of digits in a given number.
2. Write a C program to find L.C.M. of two numbers.
3. Write a C program to find out H.C.F. of two numbers.
4. Write a C program that opens a file and writes some text and closes it.
5. Write a C program to delete a file.
6. Write a C program to copy a file from one location to another location.
7. Write a C program to copy data from one file to another file.
8. Write a C program that displays source code as an output.
9. Write a C program that writes a string in the file.
10. Write a C program that reads a string from a file.
11. Write a C program that writes an array in the file.
12. Write a C program that concatenates two files and writes it to the third file.
13. Write a C program that finds out the size of any file.
14. Write a C program to know the type of file.
15. Write a C program to know the permission of any file.

Group-2 Explanations

1. **d**

   Explanation:



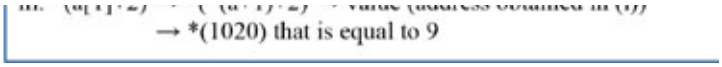


Figure 3.1: *Representation of 2-D array in stack area*

2D array: 2D array is a collection of 1D arrays. The elements of an array get stored at a continuous location in memory. In the standard C compiler, the 2D array gets stored in row-major order (1st row, 2nd row, and so on). The a[1][2] can also be written as *(*(a+1)+2). Here, 1 signifies skipping of one row, and 2 signifies skipping of two columns. First * (with a+1) means selecting a particular row and second * means selecting a particular column. The address of a+1 is 1012, (skipping one row), the address of a+2 is 1024 (skipping two rows), and so on.

e.g. Retrieve a[1][2] or *(*(a+1)+2)?

*(a+1) means that we are skipping the first row and selecting the second row with address 1012. *(a+1)+2 means that we are skipping 2 elements in the selected row. The address of (*(a+1)+2) is given as 1020. *(*(a+1)+2) is selecting an element after skipping the one row and two elements from a column. It gives a particular element of a[1][2] that is 9.

2. **c**

   Explanation: It's a concept of function pointer. C allows making such pointers that are capable of holding the address of a function. In the given program 'p' is a pointer that is pointing to function *f()*. One thing should be noted is that the type of p must be the same as the type of function *f()*. Once a pointer starts to point a function we can call the function using that pointer and can pass parameters as demonstrated in line 2. Hence there is no error in the program.

3. **d**

   Explanation: A simple recursive program, the explanation is not required.

4. **a**

Explanation: The function pointer can be applied in two ways (line 4 and line 5). See the explanation of question 2.

5. **Nil**

   Explanation: The extern is also used for making the variable/function global. A function is always global by default. Prefixing extern before the function doesn't make any difference.

6. **Nil**

   Explanation: The *static* means that a function cannot be accessed from other C files. It is not sharable with other C programs. Access to static functions/variables is limited to the file where they are declared. In C, static provides file-level encapsulation. *The scope of static variable/function is* within a file. If a variable is declared as static inside some function, the name scope of that variable is inside the function only but with the help of the address of that static variable, we can access it from other functions also but within the same program. The lifetime of *static* variables/functions is up to the exit of the main().

7. **d**

   Explanation: This program is evaluating the sum of digits of a number. The *fun(number, &var);* stores the sum of digits of the number in variable 'var' declared in main().

8. **c**

   Explanation: In function *fun(),* the variable 'count' is static, the memory for 'count' is allocated once in the data area. The *main()* is calling the *fun()* several times and the *fun()* is updating 'count' in each call. The change in the value of 'count' takes place at the same address. This program returns the summation of 4.

9. **c**

   Explanation: A simple recursive program, just trace it. An explanation is not required.

   Hint: In given function *fun()*, "return 5;" statement will never execute.

10. **c**

    Explanation: In the given program *else* is associated with the *if* statement of line 3 not with the *if* statement of line 2. Hence, nothing will get printed on the screen.

11. **b**

Explanation: In C, *sizeof()* operator returns unsigned integer value. At line 1, the unsigned integer and default integer (signed) value is getting compared. The relational operators only allow comparison between the same data type on both the left and right-hand sides. So, the standard C compiler promotes the signed integer to an unsigned integer. The value of -5 in an unsigned integer is a very large positive number. This makes the *if* condition false, *else* part will execute, and 'False' will get printed.

12. **c**

Explanation: Be thorough with explanation of question 1. Solve (i) *a[2], (ii) a[2][0] and (iii) **(a+1+('b'-'a'))

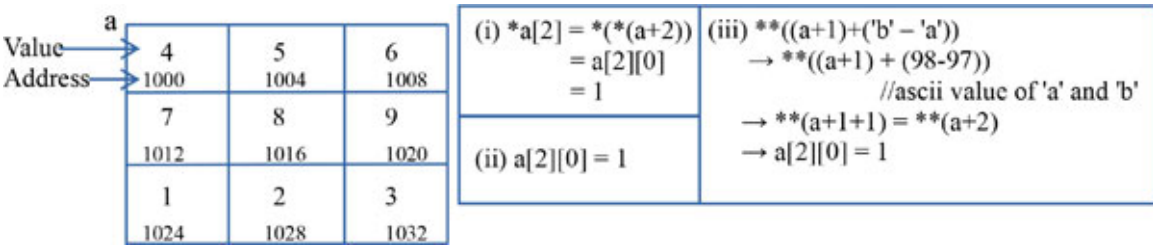


Figure 3.2: *Demonstration of 2-D array in stack area*

13. **c**

Explanation: At line 1, 'l' is having a garbage value and at line 3 'k', 'j' also gets assigned with a garbage value. At line 4 '&&' returns the value 1, when both sides are non-zero otherwise zero. Here, 'i' is having value 3 and 'j' is having garbage (that can be zero or not). If 'j' is non-zero then '&&' operator returns 1 but it can be zero then '&&' operator returns 0. So, the behavior of the given program cannot be expected.

14. **b**

Explanation: At line 2 in a given program, parenthesis has higher precedence than the '+' operator, it first evaluates parenthesis and then '+'. If we remove the parenthesis it throws an l-value required error. "Lvalue required" means that it is not possible to assign a value to something that has no place in memory. A variable on the left-hand side is needed to assign a value. Few l-value error occurring operations are 3 = a; 2 = 2;

15. **b**

Explanation: It's all about the parenthesis at line 3. Refer to explanation of question 49 of group 1.

16. **b**

Explanation: Precedence of the relational operator '==' is higher than logical operator '&'. Hence, 'false' will get printed.

17. **a**

    Explanation: '&' operator does bitwise AND. *i&j → 2&1* → 0000 0010 & 0000 0001 = 0000 0000, 0 is not equal to 3, hence 'k' gets value zero.

18. **b**

    Explanation: '<<' is left shift operator, and its precedence is higher than relational operator '>'.

    k = i << 1 > 5 ; → k = (3 << 1) >5 ; → k = (0000 0011 << 1) > 5 ; → k = (0000 0110) > 5 ; → k = 6 > 5 ; → k=1;

19. **c**

    Explanation: The '++' and '*' have the same precedence. Now, we'll go for associativity, '++' is having R-L associativity. So it evaluate as * (p++).

20. **c**

    Explanation: The '^' (XOR) and '&' (Logical And) operators have the same precedence, associativity will be checked. The logical operators are R-L associative. The expression at line 2 can be written as:

    k = i^(j&1) ;

    k = 0000 0010 ^ (0000 0010 & 0000 0001);

    k = 0000 0010 ^ (0000 0000) ;

    k = 0000 0010 ;

    k = 2 ;

21. **d**

    Explanation: For an assignment operation, the left-hand side of the assignment operator must be a variable. Constants or any kind of expressions are not allowed at the left-hand side of the assignment operator. In line 2, there exists an ambiguity at the left-hand side of the assignment operator. Hence, the compiler will throw the L-value required error. If we made the expression: k = i&&(j=1), then it is solvable.

22. **a**

    Explanation: The '!' (not) is a logical operator and its precedence is higher than relational operator '&&'. '!' makes zero to non-zero and vice

versa. The give expression can be expressed as:

→ !i && j; → (!0) && 2; // left hand side is an expression and right hand side is non-zero

→ 1&& 2;→ 1; // left hand side non-zero and right hand side also non zero, it returns 1.

23. **a**

Explanation: The '~' operator determines 1's complement of a given number.

i = 00 00 00 00; (representation in hexadecimal form)

~i = ff ff ff ff ; equivalent to -1;

k = ~i & j; // '~' , is a unary logical operator, its precedence is higher than binary logical operator.

= (~i) & j ; → -1 & 2 ; → all 1's & 0010 ; → 2 ;

24. **c**

Explanation: y = 6, z = 7 ; → x = ++y + z-- ; → y = y+1; // y = 7 .

→ x = y+z; // x = 7+7 = 14. → z = z-1; // z = 6. → x = 14 ;

25. **c**

Explanation: Little-Endian: The common machine like Intel and AMD use little-endian. It means that the lower significant byte gets stored at the higher memory addresses and the most significant byte gets stored at the lower addresses. The integer value 258 in binary form can be represented (MSB)00000000 00000000 00000001 00000010(LSB) and in a machine following little-endian, it gets stored as:

MSB get stored at lower address

00000000	00000000	00000001	00000010
1000	1001	1002	1003

LSB get stored at higher address

Figure 3.3: *Memory Representation of an integer into Little Endian mode*

Big-Endian: Just an opposite of Little-Endian, here LSB gets stored at lower addresses and MSB gets stored at higher addresses. The 258 can be represented in a machine with big-endian as follows:

LSB get stored at lower address

00000010	00000001	00000000	00000000
1000	1001	1002	1003

MSB get stored at higher address

Figure 3.4: *Memory Representation of an integer into Big Endian mode*

Almost all the machines we used for programming purposes like Intel, AMD, are following little-endian. Here our concern is to build your concepts without misconception. Practically, these machines are following little-endian but if we want to trace addresses of arrays, structures, etc. from the C program, it would be difficult. But if we think about big-endian, it is always easy to trace the memory addresses through the C program.

int a[3] = {5, 6, 7}; get stored in machine following little endian architecture as follows:

00	00	00	05	00	00	00	06	00	00	00	07
1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	1011

Figure 3.5: *Representation of an array element in hexadecimal form in Little Endian mode*

In our discussion for this question, we'll explain the concept using big-endian for better understanding. The representation of an array in the machine using big-endian are as follows:

```
int a[3]={5,6,7};
```

05	00	00	00	06	00	00	00	07	00	00	00
1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	1011

Hence, c[0] = 5; c[1] = 0; c[2] = 0; c[8] = 7;

Figure 3.6: *Representation of an array element in hexadecimal form in Big Endian mode*

The expression, *char *c = (char *)a;* char pointer can only access one byte at a time. Now the step size of accessing a byte becomes 1, but while storing the step size was 4B integer value.

26. **c**

Explanation: The execution of a given program is done in two steps: (i) The macro *FOO(x)* will get substituted as foo(#x), where '#' converts 'x'

to string. The number 65 will be converted into the string "65". (ii) The function call will take place as foo("65"). In function *foo()*, string "65" is pointed by character pointer ‘x’. Hence, x[0] will give ‘6’ and x[1] will give ‘5’.

27. **c**

    Explanation: A program can have the same function name and variable name in different scopes, the given question will print 3. Suppose if we make "int main = 3;" as global variable, then the program would have thrown an error as "int main = 3;" and "int main()" both become global, and are in same scope.

28. **b**

    Explanation: The *curly braces* after the *case 1* statement will never execute, as it is out of the scope of the case.

29. **d**

    Explanation: The *switch* statement can only support type "integer", "character", "short", "long" and "enum". Floating-point data types are not allowed with the switch statement.

30. ***Invalid*- a; *valid*- b, c, d;**

    Explanation: Not required.

31. **d**

    Explanation: ‘else’ without ‘if’ is not allowed. Hence, the compiler throws an error.

32. **c**

    Explanation: In the given program all instances of *strcpy(x, y)* will be substituted by *x = y*, the preprocessed program will not have any instance of library function *strcpy()*, at line 5, we are trying to assign a string to character array using assignment operator that is invalid in C. Hence, the compiler will throw an error at line 5.

33. **d**

    Explanation: Refer to the explanation of questions 1 and question 12.

    We can directly access the address using the ‘*’ operator without skipping any column or row in the 2D array-like *1024 return 1.

    p = 1024 q = 1024

    p = & a[2] = 1024

q = a [2] = 1024

printf("%d %d, *1024 , *1024) ;

Output 1 1

We can directly access the address using the '*' operator without skipping any column or row in the 2D array-like *1024 return 1.

34. **b=11; c=4; d=6; e=15; f=9; g=33; i=6; j=8;**

    Explanation: Not required

35. **c**

    Explanation: After the substitution of macro at line 1, it becomes *char *a, b;* At line 6, ‘b’ is assigned to an address whereas ‘b’ is a character variable. In line 7, we are doing *(b+2), which is not a relevant operation with a character variable. Hence, compile throws an error.

36. **a**

    Explanation: Not required

37. **d**

    Explanation: Duplicate *case* labels are not allowed. The ASCII value of ‘A’ is 65 and case ‘A’ is treated as *case 65*.

38. **c**

    Explanation: Refer to question 33.

    p = 1024 q = 1024

    p = &a[2] = 1024

    q = a[2] = 1024;

    p[1] = *(p+1) = *1028; → 2

    q[2] = *(q+2) = *1032; → 3

    //’p’ and ‘q’ are integer pointer, there step size is 4B, they are independent of an array.

39. **a**

    Explanation: Any expression is allowed in the initialization, decision, and updating part of the *for* loop.

40. **c**

    Explanation: The compiler executes the body of the *do* block and then it checks the condition in the *while* statement. Hence, at least once the *do-while* loop always executes whatever the condition is.

41. **b**

    Explanation: In a given program, ‘s’ is a character pointer that is pointing to an empty string, s contains the address of the empty string. In the while statement, the condition is applied on ‘s’ which contains the address that is always non-zero. The loop will execute infinite time.

42. **d**

    Explanation: The behavior of various while loops can be evaluated easily. The explanation is not required.

43. **c**

    Explanation: The *comma* operator has left to right associativity. In L-R associativity, left is first evaluated and then right. Here the evaluation takes place in order "x < 4" takes place and then "y < 7". The "x < 4" will get discarded and the while loop only checks the condition "y < 7".

44. **c**

    Explanation: Not required.

45. **c**

    Explanation: It’s an example of continuing with the do-while loop.

46. **b**

    Explanation: The *break* statement can also be used in loops.

47. **c**

    Explanation: It is a practice question. The explanation is not required.

48. **b**

    Explanation: The *goto* statement transfers the control of execution from the current instruction to the *labelled* instruction. Its step size is not fixed. The g*oto* statements are not frequently used in C. It makes the control flow of a program logically complex which is difficult to understand. It is not a good programming practice using goto in a program.

49. **b**

    Explanation: Not required.

50. **d**

    Explanation: In a given program, from *main()*, it is tried to use the instruction of a different function without calling the function. It is not possible to access the function body without calling it. The function gets pushed into the memory area (stack) then only it can get used. Here, the

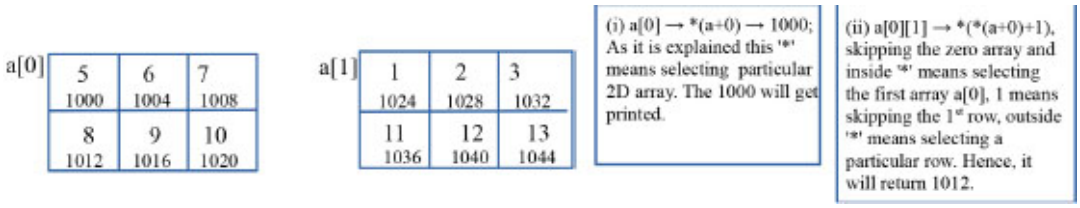
compiler throws an error- *label1 is undefined*. If we want to use a label with goto, it must be local to a function.

51. **b**

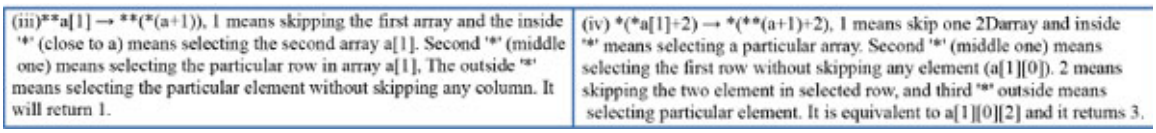
    Explanation: It works fine in gcc but in Turbo C it throws an error. GNU C supports the *nesting* of functions.

52. **d**

    Explanation: Given a[2][2][3], two 2D arrays of size 2x3.



    Suppose we want to access an element a[1][1][2]. In 3D array, a[1] means skipping one 2Darray and a[1][1] (or *(*a+1)+1))), highlighted '*' means selecting second array and highlighted 1 means skipping the row in selected 2D array, a[1][1][2] means skipping 2 column.



53. **c**

    Explanation: Refer explanation of question 52, we can directly access the address using the '*' operator without skipping any column or row. Like x = &a[1][1][2] directly get the address 1044 and we can access element at 1044 by doing '*x' (*1040). The pointer variable 'y' holds address 1000, we can directly access the value at 1000 using *1000. Hence, the program will return *1044 and *1000 that is value 13 and 5.

    x : 1044 y : 1000

54. **a**

    Explanation: Given a[2][2][3] means we are having two 2D arrays of size 2x3. As we discussed in the explanation of question 52, a[1][2] means skipping one 2D array and then skipping 2 rows in the selected array. Here, each 2D array contains only two rows if we skip two rows, it means we are going out of the array space. The garbage value will get printed. a[2][1][0], 2 means skipping two 2D arrays. We are having only

two 2D arrays, we are trying to access the address out of the array space. Hence, the garbage will get printed.

Note: It will not throw an error, because the addresses outside the array space are also readable and garbage will get returned.

55. **b**

Explanation: Refer to explanation of question 25.

At line 4, **(int*)p+2;* we are again converting the character pointer to an integer type. Now the step size becomes 4 which is the same as the integer pointer step size. It is similar to a[2].

56. **The number of elements in a given array is 16.**

Explanation: int number_of_elements = sizeof(a)/sizeof(a[0]);

57. **Nil**

Explanation:

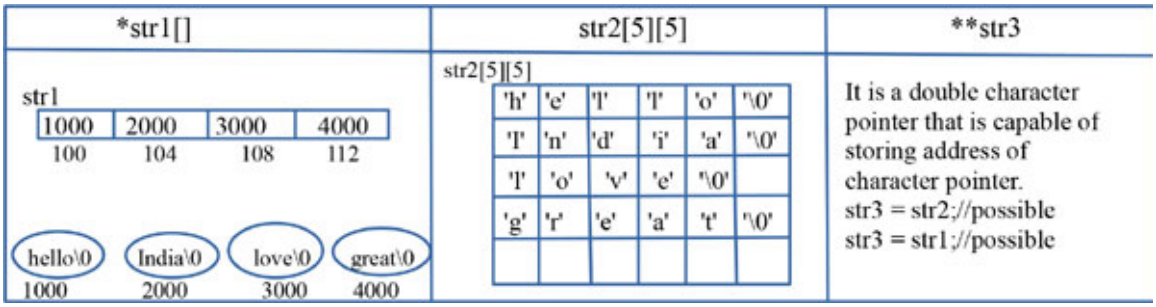


Figure 3.7: *Representation of char pointer in text and stack area*

All three types of declarations are interchangeably used by keeping a few conditions in mind, (i) Base address of an array cannot be updated. The below table briefly describes the possible assignment operations with these three types of declarations.

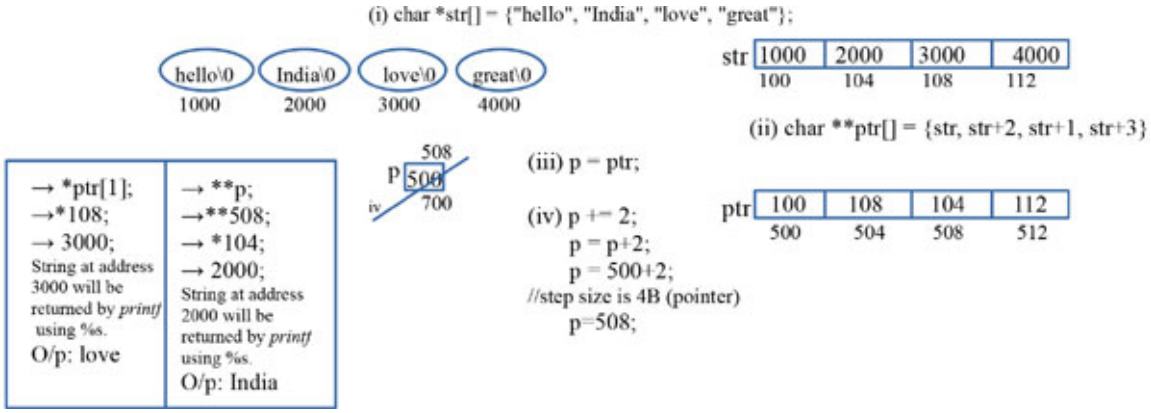
		str declarations (column)		
		*str[];	str[5][5];	**str;
Possible assignment operations (row)	str=&var;	×	×	√
	*str=&var;	√	×	√
	**str=var;	√	√	√

× = invalid assignment
√ = Valid assignment

Table 3.1: *Assignment for str with given declarations*

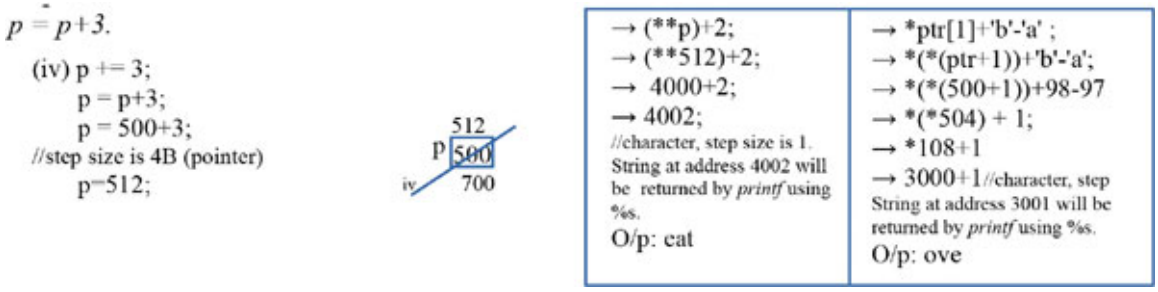
58. **c**

Explanation:



59. **d**

Explanation: The variable assignment is the same as the previous question, there is only a change in step (iv), here, it's *p = p+3.*



60. **c**

Explanation: When a user requests a dynamic memory allocation, the C program requests the operating system to allocate memory. If the memory is available, the memory manager component of the operating system allocates the memory to the program. If the system is too busy, the memory manager can deny the request to allocate memory for the program. Here, in a given program the memory is allocating dynamically in an infinite while loop without deallocating (free()). When the main memory (RAM) gets completely allocated to our program. There is no memory left for other processes, the heap gets overflow and the system (having Unix type operating system) hangs.

If we deallocate the memory in each iteration, the heap will not overflow, the system will not hang, just the hello 10 will get printed infinite time. See the following program:

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

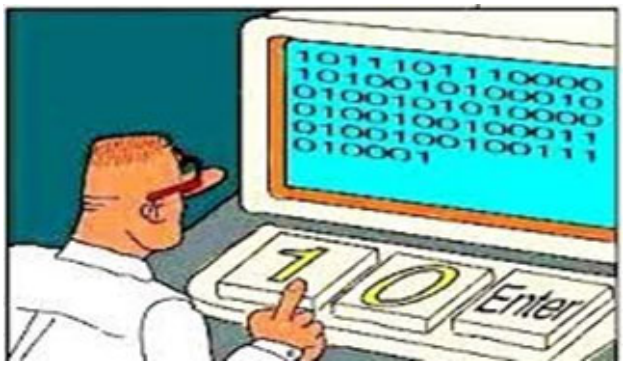
```
int main()
{
 while(1)
 {
  int *p = (int *)malloc(sizeof(int));
  *p=10;
  printf("hello %d", *p);
  free(p);
  p=NULL; // to avoid dangling pointer (Group1, Question 7
  explanation)
 }
return 0;
 }
```

If it is easy then it is not for you.

Group-3 Questions

Number of 'C' Question: 60

Other Important Interview Questions: 15



1. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *str = "India";
 while(*str)
 {
  printf("%s ", str++);
 }
 return 0;
}
```

(a) India

(b) India ndia dia ia a

(c) dia ia a

(d) loop will run infinite time

2. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *str1 = "Hello In\0dia";//'\0' takes one character
 space
 char *str2 = "x%s ";
 printf(str2, str1);
 return 0;
}
```

(a) xHello In dia

(b) xHello In

(c) xHello In\0dia

(d) none of these

3. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
int main()
{
 char str1[] = "Hello In\0dia";
 printf("%u %u", sizeof(str1), strlen(str1));
 return 0;
}
```

(a) 9 8

(b) 13 12

(c) 12 8

(d) 13 8

4. **Which of the following can be the external variable/s in the given program?**

```
#include<stdio.h>
int fun(int x)
{
 int y=90;
 return y;
```

```
}
int main()
{
 extern int i;
 fun(i);
 return 0;
}
int e;
```

(a) x, y

(b) i, x

(c) e, i

(d) e

5. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *str1 = "%d %d %d\n";
 char str2[] = "%d %d %d\n";
 printf(str1, sizeof(str1), strlen(str1), sizeof(str2));
 return 0;
}
```

(a) 10 9 10

(b) 4 9 10

(c) 4 10 11

(d) 4 10 9

6. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
int main()
{
 char *str = "India";
 char a[10];
 int i;
 int len = strlen(str);
```

```
  for(i=len; i>=0; i--)
   a[i] = str[len-i];
 printf("%s", a);
 return 0;
}
```

(a) aidnI

(b) aidn

(c) India

(d) nothing will get printed

7. **What is the output of the following program?**

```
#include<stdio.h>
void fun(int *p)
{
 int i=0;
 for(; i<5; i++)
 {
  *(p+i) = *(p+i) *3 - *(p+i);// line 5
 }
}
int main()
{
 int a[] = {5, 7, 1, -2, 8};
 fun(a);
 int i=0;
 for(; i<5; i++)
 {
  printf("%d ", a[i]);
 }
 return 0;
}
```

(a) 5 7 1 -2 8

(b) 10 14 2 -4 16

(c) 15, 21, 3, -6, 24

(d) none of these

8. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x=5;
 int *p = &x;
 printf("%d", x**p+((p+2)-(p+1))**p);
 return 0;
}
```

(a) 5

(b) 25

(c) 30

(d) 8

9. **What is the output of the following program?**

```
#include<stdio.h>
int *fun(int *a, int *b)
{
 *a = *a-*b+1;
 *b = *a*4 + 2;
 return b;
}
int main()
{
 int x=5, y=6;
 int *z = fun(&x, &x);
 printf("%d", *z+y);
 return 0;
}
```

(i) z contain the address of?

(a) x (b) y (c) none

(ii) output of the program is

(a) 11 (b) 20

(c) 10 (d) 12

10. **What is the output of the following program?**

```
#include<stdio.h>
void fun(static int a, auto int b, register int c)
{
 if(a>2)
 {
  printf("%d %d %d", a--, b, ++c);
  fun(a, b, c);
 }
}
int main()
{
 fun(5, 6, 7);
 return 0;
}
```

(a) 5 6 4 6 3 6

(b) 5 6 5 6 5 6

(c) infinite time

(d) error- compile time

11. **What is the output of the following program? Suppose the address of ‘a’ is 1000 and ‘x’ is 2000.**

```
#include<stdio.h>
int main()
{
 int a = 10, *b = &a;
 int x = 20, *y = &x;
 *b-- = *y++;
 printf("%u %u %u %u", &a, b, &x, y );
 return 0;
}
```

(a) 1000 1000 2000 2000

(b) 20 1000 20 2000

(c) 1000 996 2000 2004

(d) error- l value required at line 3

12. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 register int i=10;
 int *p = &i;
 *p = 11;
 printf("%d %d", i, *p);
}
```

(a) 10, 11

(b) 11, 11

(c) 11, 10

(d) error- compile time

13.

```
register int i=10;
```

Is it mandatory that variable ‘i’ gets stored in the register only?

(a) true

(b) false

(c) compiler dependent

(d) none of these

14. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 register static int i=10;
 i=11;
 printf("%d", i);
 return 0;
}
```

(a) 10

(b) 11

(c) 21

(d) error- compile time

15. **When does the compiler accept the request to use the variable as a register? Which is/are true about the variables?**

    (a) It gets stored in cache memory

    (b) It gets stored in the CPU

    (c) It gets stored in secondary memory.

    (d) It gets stored in the main memory.

16. **Which of the given operation cannot be done with the register?**

    (a) Reading from and updating (writing) into the register variable.

    (b) Global declaration of register variable.

    (c) Copy a value from the memory variable to the register variable.

    (d) All of the above

17. **What is the output of the following program? Suppose the address of ‘a’ is 1000 and ‘x’ is 2000.**

```
#include<stdio.h>
int main()
{
   int a = 10, *b = &a;
   int x = 20, *y = &x;
   ++*y = 3; //line 3
   x = ++*y; //line 4
   printf("%u %u %u %u", &a, b, &x, y );
  return 0;
}
```

    (a) 10 1000 20 2000

    (b) 21 1000 21 2000

    (c) error- l value required at line 3

    (d) error- l value required at line 4

18. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
  char *str = "India";
  printf("%s", str+++2+’z’-’x’);
```

```
 return 0;
}
```

(a) India

(b) dia

(c) Nothing will get printed

(d) a

19. **What is the output of the following program?**

```
#include<stdio.h>
int i;
int main()
{
 char *str1 = "Hello";
 char str2[] = "India is great";
 while(*str1){
  str2[i] = *str1++;
  i++;
 }
 printf("%s", str2);
 return 0;
}
```

(a) Hello is great

(b) India is great

(c) garbage

(d) Hello

20. **In the previous question, can we copy from str2 to str1, yes/no? If yes, justify, or If no, suggest modifications required in the given code?**

21. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=10;
 void *p = &i;
 printf("%d", (int)*p);
```

```
 return 0;
}
```

(a) 10

(b) address of variable i

(c) address of variable p

(d) error- run time

22. **What is the output of the following program?**

```
#include<stdio.h>
void fun(int *arr)
{
 printf("%u, ", *arr);
 printf("%u, ", *(arr+4));
}
int main()
{
 int a[2][4] = {5, 6, 7, 8, 9, 1, 2 ,3};
 fun(&a);
 printf("%u, ", *a);
 printf("%u, %u", *(a+1), **(a+1));
 return 0;
}
```

(a) 5, 9, addr(5), addr(6), 6

(b) 5, 9, addr(5), addr(9), 9

(c) addr(5), addr(9), addr(5), addr(9), addr(9)

d) 5, 9, 5, 6, 9

23. **Write a code in C to allocate a 2D array in memory dynamically using malloc()?**

24. **Given: const int *ptr;**

Which of the following statement is/are true about the given declaration?

(a) We cannot change the pointer itself that is ptr++ not allowed;

(b) We cannot change the value pointed by ptr that is (*ptr)++ not allowed.

(c) Both a and b;

(d) We can change both pointer and value pointed by the pointer

25. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
int main()
{
 printf("%u, %u, %d, %u", sizeof(NULL), sizeof(""),
 strlen(""), sizeof('\0'));
 return 0;
}
```

(a) 1, 1, 1, 1

(b) 1, 1, 2, 1

(c) 4 , 2, 0, 1

(d) 4, 1, 0, 4

26. **What is the output of the following program?**

```
#include<stdio.h>
int x;
int main()
{
 int * y = &5;
 int *const ptr = &x;
 ptr++;
 (*ptr)++;
 printf("%p %d", ptr, *y);
}
```

(a) addr 5

(b) addr addr

(c) 5 5

(d) error- compile time

27. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=10;
```

```
 fun((&i)++);
 return 0;
}
void fun(int *p)
{
 printf("%d", *p);
}
```

(a) 10

(b) garbage

(c) 11

(d) error- compile time

28. **What is the output of the following program?**

```
#include<stdio.h>
void fun2(int *y)
{
 printf("%d ", *y);
}
void fun1(int *x)
{
 printf("%d ", *x);
 fun2(x);
}
int main()
{
 int i=10;
 int *p = &i;
 fun1(p++);
 printf("%d", *p);
 return 0;
}
```

(a) 10 10 10

(b) 10 garbage 10

(c) 11 11 11

(d) 10 10 garbage

29. **What is the output of the following program?**

```
#include<stdio.h>
void fun(float *p)
{
 printf("%d", *(int *)p);
}
int main()
{
 int i=10;
 int *p = &i;
 fun(p);
 return 0;
}
```

(a) 0.00000

(b) 10

(c) 10.00000

(d) error-type conversion

30. **What is the output of the following program?**

```
#include<stdio.h>
void fun(int *p)
{
 int j=4;
 p = &j;
 printf("%d ", *p);
}
int main()
{
 int i = 50;
 fun(&i);
 printf("%d", i);
 return 0;
}
```

(a) 4 50

(b) 50 50

(c) 4 4

(d) 50 4

31. **What is the output of the following program?**

```
#include<stdio.h>
void fun(int **p)
{
 static int j=4;
 *p = &j;
 printf("%d ", **p);
}
int main()
{
 int i=50;
 int *a = &i;
 fun(&a);
 printf("%d", *a);
 return 0;
}
```

(a) 4 50

(b) 50 50

(c) 4 4

(d) 50 4

32. **What is the output of the following program?**

```
#include<stdio.h>
void fun(int *const *p)
{
 int j=4; // line 1
 *p = &j; // line 2
 printf("%d ", **p);
}
int main()
{
 int i=50;
 int *a = &i;
 fun(&a);
 printf("%d", *a);
 return 0;
}
```

(a) 4 50

(b) 50 50

(c) 4 4

(d) error- compile time

33. **What is the output of the following program?**

```
#include<stdio.h>
void fun(int **const p)
{
 int j=4;
 *p = &j;
 printf("%d ", **p);
}
int main()
{
 int i=50;
 int *a = &i;
 fun(&a);
 printf("%d", *a);
 return 0;
}
```

(a) 4 50

(b) 50 50

(c) 4 4

(d) error- compile time

34. **What is the output of the following program?**

```
#include<stdio.h>
void fun(int **p)
{
 int j=4;
 *p = &j;
 printf("%d ", **p);
}
int main()
{
 int i = 50;
```

```
  int * const a = &i;
  fun(&a);
  printf("%d", *a);
  return 0;
 }
```

(a) 4 50

(b) 50 50

(c) 4 4

(d) error- compile time

35. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
  const int a[5] = {3, 1, 5, 4, 2};
  int *p;
  p = a+3;
  *p = 99;
  //a[3] = 78;
  printf("%d, %d, %d", a[2], a[3], a[4]);
  return 0;
}
```

(a) 5, 4, 2

(b) 5, 99, 2

(c) 5, 78, 2

(d) compile time error

36. **What will be the output of the above program if we uncomment the commented line *a[3] = 78;*?**

(a) 5, 4, 2

(b) 5, 99, 2

(c) 5, 78, 2

(d) error- compile time

37. **What is the output of the following program?**

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

```
void fun(int *p)
{
 printf("%d, %d, %d ", p[2], p[3], p[4]);
}
int main()
{
 int a[5] = {3, 1, 5, 4, 2};
 int p[4];
 //p = a;
 int *ptr = a;
 fun(a);
 fun(&a);
}
```

(a) 5, 4, 2

(b) 5, 4, 2 5, 4, 2

(c) 2, 5, 4 2, 5, 4

(d) error- compile time

38. **What will be the output of the above program if we uncomment the commented line *p = a;*?**

(a) same output

(b) error- base address of an array cannot be updated

39. **What will be the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[4] = {5, 6, 7, 8};
 int *p = a+3;
 printf("%d ", p[-2]);
 return 0;
}
```

(a) 8

(b) 6

(c) 5

(d) error-compile time due to p[-2]

40. **What will be the output of the following program?**

```
#include<stdio.h>
int main()
{
 double *ptr = (double *)100;
 ptr += 2;
 printf("%u ", ptr);
 return 0;
}
```

(a) 102

(b) 100

(c) 108

(d) 116

41. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int *p = (int *)200;
 int *q = (int *)300;
 printf("%u", p+q);
 return 0;
}
```

(a) 500

(b) 5

(c) -100

(d) error- compile time

42. **Which of the following operations are possible on two pointer?**

Given: int *p = (int *)200;

int *q = (int *)300;

Arithmetic: (a) p+q (b) p-q (c) p*q (d) p/q

Logical: (a) a | b (b) a&b (c) a^b (d) none of these (e) all of these

Relational: (a) p||q (b) p&&q (c) p<q (d) p>q (e) p==q (f) none of these (g) all of these

43. **What is the value of ‘j’ and ‘i’ in the following program?**

```
#include<stdio.h>
int main()
{
   int i=5; //line 1
   int j = 3*i++ + 2*i++; //line 2
   //i++ = 3*i++; //line 3
   printf("%d %d", j, i);
   return 0;
}
```

(a) 37 7

(b) error- L-value required

(c) 25 7

(d) 27 7

44. **What will be the output of the above program if we uncomment the commented line *3?***

(a) 37 8

(b) error- L-value required

(c) 25 8

(d) compiler dependent.

45. **What is be the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *s = "hello";
 char *p = s; //line 2
 printf("%c %c", *(p+1), s[1]);
 return 0;
}
```

(a) h e

(b) e e

(c) e h

(d) error- compile time at line 2

46. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[4][] = {4, 5, 6, 7, 8, 9, 10, 2}; //line 1
 int i=0, j=0;
 for(i=0; i<4; i++)
 {
  for(j=0; j<4; j++)
  {
   printf("%d\t", a[i][j]);
  }
  printf("\n");
 }
 return 0;
}
```

(a) printing all the elements of 2D array

(b) syntax error at line 1.

47. **Which of the following is/are the correct way of declaring an array?**

(i) int a[]; int a[10];

(ii) int a[3][3]; int a[][3];

(iii) int a[3][4][5]; int a[][4][5]; int [][2][4][5];

(iv) int a[][][5]; int a[][]; int [][][4][5];

(a) i, ii, iv

(b) i, ii, iii

(c) ii, iii, iv

(d) all are correct

48. **Which of the following is/are the correct way of declaring the function whose parameter is array (1D, 2D, or 3D)?**

```
int main()
{
 int a1[10]; a2[4][5]; a3[2][4][5];
 fun1(a1);
 fun2(a2);
```

```
  fun3(&a2)
}
```

The declaration of function should be:

(i) fun1(int *a), fun1(int a[])

(ii) fun2(int **a), fun2(int a[][5]), fun2(int *a[])

(iii) fun3(int ***a), fun3(int a[][4][5]), fun(int *a[][]), fun(**a[])

(iv) fun2(int a[][]), fun3(int a[][][5])

Which of the above declarations of fun1, fun2, and fun3 is correct?

(a) i, ii, iv

(b) ii, iii, iv

(c) i, ii

(d) i, ii, iii

49. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 void *p;
 int a[5] = {5, 6, 4, 7, 8};
 p = &a[3];
 int *ptr = &a[2];
 int n = p-ptr;
 printf("%d %d", n, a[n]);
 return 0;
}
```

(a) 1 6

(b) 4 8

(c) 3, 7

(d) error- compile time

50. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[0];
```

```
 printf("%u %u", sizeof(a), sizeof(a[0]));
 return 0;
}
```

(a) 0, 4

(b) compiler dependent

(c) 0, 0

(d) 1, 4

51. **What will be the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[5] = {5, 6, 7, 8, 9};
 int *ptr = &a[2];
 float n=1; //line 3
 ptr += n;
 pritnf("%d", *ptr);
 return 0;
}
```

(a) 7

(b) 8

(c) 9

(d) error- compile time

52. **What is the output of the following program? Suppose the base address of an array ‘a’ is 1000?**

```
#include<stdio.h>
int main()
{
 int a[] = {5, 6, 7, 8, 9, 10};
 int *p = &a;
 int *q = &a+1;
 printf("%u %u ", a, p);
 printf("%u %u ", a+1, q);
 return 0;
}
```

(a) 1000 1000 1001 1001

(b) 1000 1000 1004 1004

(c) 1000 1000 1004 1020

(d) 1000 1000 1004 1024

53. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *str1 = "hello India";
 char *str2 = "how are you";
 strcpy(str1, str2);
 printf("%s", str1);
 return 0;
}
```

(a) how are you

(b) hello India

(c) error- compile time

(d) error- run time

54. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
int main()
{
 char str1[] = "hello India";
 char *str2 = "how are you";
 strcpy(str1, str2);
 printf("%s", str1);
 return 0;
}
```

(a) how are you

(b) hello India

(c) error- compile time

(d) error- run time

55. **Determine the output of the following program? Suppose the base address of an array is 1000.**

```
#include<stdio.h>
int main()
{
 int a[2][2][3] = {7, 6, 5, 4, 3, 2, 1, 8, 9, 10, 11, 12};
 printf("%u %u %u %u %u", a, a+1, &a+1, a[1]+2, a[1][2]);
 return 0;
}
```

56. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int *s = "hello"; // line 1
 printf("%c %s", s[1], s);
 return 0;
}
```

(a) h hello

(b) e herlo

(c) error- at line 1

(d) o hello

57. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *str1 = "hello world";
 char str2[] = "hello world";
 printf("%ld %ld ", sizeof(str1), sizeof(str2));
 printf("%ld %ld", strlen(str1), strlen(str2));
 return 0;
}
```

(a) 11 11 11 11

(b) 12 12 12 12

(c) 4 11 11 11

(d) 4 12 11 11

58. **Determine the output of the following program? Suppose the base address of an array is 1000.**

```
#include<stdio.h>
int main()
{
 int a[3][4] = {7, 6, 5, 4, 3, 2, 1, 8, 9, 10, 11, 12};
 printf("%u %u %u %u %u ", a, a+1, *(a+2), *a+3, a[2]);
 printf("%u %u %u %u %u %u", *a[1], a[1][0], *a, &a, &a+1,
 &a+2);
 return 0;
}
```

59. **What is the output of the following program?**

```
#include<stdio.h>
void f(char *k)
{
 k++;
 k[2] = ‘m’;
 printf("%c", *k);
}
int main()
{
 char str[] = "hello";
 f(str);
 return 0;
}
```

(a) e

(b) m

(c) error- run time

(d) error- compile time

60. **Which of the following statements is/are true for the given program?**

```
#include<stdio.h>
int main()
{
 char *str1 = "hello";
```

```
  char *str2 = "India";
  str1 = str2; //line 3
  printf("%s %s", str1, str2);
 }
```

(a) memory holding "hello" gets cleared at line 3;

(b) memory holding "hello" loss its reference at line 3;

(c) we cannot assign pointer as we did in line 3, it throws an error

(d) the output will be "hello India"

(e) the output will be "India India"

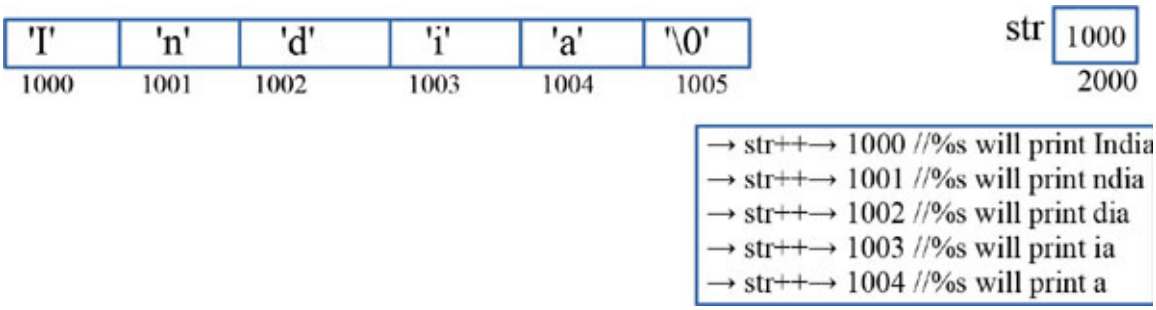
Other important questions for the Interview are as follows:

1. Write a C program that returns the last modified date of a file.
2. Write a C program to find the size and storage location of any file.
3. Write a C program that finds the largest element from an array.
4. Write a C program that finds the second largest element from an unsorted array.
5. Write a C program that finds the second smallest element from an unsorted array.
6. Write a C program that deletes the duplicate element in an array.
7. Write a C program that deletes an element of a given position in an array.
8. Write a C program that inserts an element at a given position in an array.
9. Write a C program that finds the largest and smallest number from an array.
10. Write a C program for bubble sort.
11. Write a C program for insertion sort.
12. Write a C program for selection sort.
13. Write a C program for quick sort.
14. Write a C program for heap sort.
15. Write a C program for merge sort.

Group-3 Explanations

1. **b**

   Explanation:



2. **b**

   Explanation: The statement *printf(str2, str1);* become *printf("x%s", str2);* %s will print the string till the *null character* encounters. *So, "xHello In"* will get printed.

3. **d**

   Explanation: The *sizeof()* is an operator, it returns the total number of bytes taken by the string. So, it returns 13. The *strlen()* is a function, declared in the "string.h" header file. It parses the string till the *null character* encounters and returns the length of the string without including the *null character*. So, *strlen()* will return 8.

4. **c**

   Explanation: The external variable are those which can be used from a different program, global variables are always external variables. Refer explanation 33, 35 of group 1. If you run this program, it will throw link-time error as `i` is undefined, to fix that just add `int i;` at the end of the program.

5. **b**

   Explanation: The '\n' takes 1B of space and %d is stored in two different bytes. One version of this program is already discussed in the explanation of question 3.

6. **d**

   Explanation: The *strlen()* will return 5, we are doing a[0] = str[5]; in for loop's first iteration. So, at the first index of an array 'a' there is a null character. Hence, nothing will get printed.

7. **b**

Explanation: Here we are simply passing an array and modifying all the entries. The expression at line 5 can be written as p[i] = 3*p[i] – p[i]; → p[i] = 2*p[i]; just doubling each element of an array.

8. **c**

   Explanation: The unary operators have higher precedence than the binary operators. Given

   expression can be written as: → x**p+((p+2)-(p+1))**p) ; //given x=5; p = &x;

   → x*(*p)+((p+2-p+1))*(*p);

   → 5*5 + (1)*5; → 30.

9. **a, d**

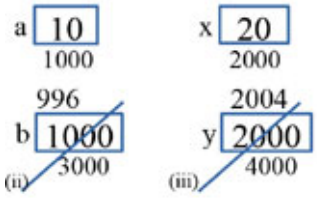
   Explanation: Not required, we discussed various similar questions in previous groups.

10. **d**

    Explanation: Any storage class (*static*, *auto*, *extern*) except register storage class is not allowed to be used as a functional argument. There is no need of making the function argument variable static because arguments are just a source of communication between functions. There is only one instance of a static variable associated with a function. If we make function argument static, we need to create a separate instance of a static variable in each function call, so we get multiple instances of a static variable, that is not allowed.

11. **c**

    Explanation:




```
→ *b-- = *y++;
//Get split into 3 parts:
(i) *b = *y;
(ii) b = b-1;
        //step size is 4
(iii) y = y+1;
→ b = 996; y = 2004; //will get printed
```

12. **d**

    Explanation: The register variable can have only register name (not address) in the processor, we cannot access register variable through

address, ‘*’ is dereferencing operator to memory addresses only. Hence, the compiler throws an error.

13. **b**

    Explanation: It is false, registers are very busy components of CPU and also they are limited in number. If there is no available register, the variable automatically gets an "auto" storage class.

14. **d**

    Explanation: The static variables get memory allocated in the data area (bss segment) that we already discussed. Registers are not stored in the program area, they are part of the machine, not our program. So, the given program will throw an error.

15. **b**

    Explanation: Not required.

16. **b**

    Explanation: Not required.

17. **c**

    Explanation: "Lvalue required" means that it is not possible to assign a value to something that has no place in memory. A variable on the left-hand side is needed to assign a value. Few l-value error occurring operations are 3 = a; 2 = 2; In a given program, at line 3, the operation is ++*y = 3; can be expanded into : *y=*y+1 = 3;. The assignment operator is right to left-associative and we are trying to store 3 in (*y+1) that is not a variable. Hence, the compiler throws an l-value required error.

18. **d**

    Explanation: Not required.

19. **a**

    Explanation: Not required, A practice question, we already discussed similar kinds of questions.

20. **No**

    Explanation: In the given program, str1 is stored in a read-only area. It is not allowed to copy into str1. We have already discussed these kinds of questions.

21. **d**

Explanation: The *void* pointer cannot be accepted without typecasting. The correct statement is *printf("%d", *(int *)p);*

22. **b**

    Explanation: Not required, we already discussed similar kinds of questions in group 1 and group 2.

23. **Nil**

    Explanation:

```
#include<stdlib.h>
#include<stdio.h>
int main()//allocating memory for a[10][10]
{
    int **a = (int**)malloc(sizeof(int*)*10);
    int j, i ;
    for( i = 0 ; i < 10 ; i++)
    {
        a[i] = malloc(sizeof(int)*10);
       *a[i] = i;
        for(j=1; j<10; j++)
          a[i][j] = i;
    }// for end
    for(i = 0 ; i < 10 ; i++)
    {
        for(j=0; j<10; j++)
            printf("%d\t",a[i][j]);
        printf("\n");
    }// for end
}// main end
```

24. **b**

    Explanation: Not required.

25. **d**

    Explanation:

(i) sizeof(NULL)→ 4;
//NULL is a macro defined as type *int* in *stdio.h* header file. Hence, it become *sizeof(int)* that is 4.

(ii) sizeof("")→ 1; It is zero length string, 1 get returned that is for null character.

(iii) strlen("")→ 0; It is zero length string, *strlen()* don't include the null character. Hence, 0 get returned.

(iv) sizeof('\0') -> 4; see explanation of question 19, group 1.

26. **d**

    Explanation: The statement *ptr++;* throws an error and *int *y = &5;* is also invalid..

27. **d**

Explanation: *"l-value required"* error because of the statement *foo((&i)++);*

28. **d**

    Explanation: A simple practice question on call by reference. Explanation is not required.

29. **b**

    Explanation: Implicit type conversion is supported by *gcc* compiler. The integer pointer first get converted into a float pointer implicitly and then in *fun()*, the float pointer gets typecast into an integer pointer explicitly. The value at the address of ‘i’ that is 10 will get printed.

30. **a**

    Explanation: Not required.

31. **c**

    Explanation: A simple concept of double-pointer. We already discussed these questions.

32. **d**

    Explanation: The functional argument in *fun() int *const*p;* //*p is constant (that is read-only) but we tried to update it at line 2. Hence, the compiler throws an error.

33. **c**

    Explanation: The program will run fine, here, ‘p’ is constant in fun(). We are updating *p, that is possible.

34. **c**

    Explanation: Here, we cannot update ‘a’ in function *main()*. But with the help of pointers, it is possible to update *const* variables also, like here, we are updating ‘a’ in *fun()*. This is also called *"Variable Hacking"*. If we want to prevent the variable hacking, just change the prototype of function *fun()* to *void fun(int *const *p)*.

35. **b**

    Explanation: One more example of variable hacking.

36. **d**

    Explanation: A given array ‘a’ is constant, *a[3] = 78;* is an invalid operation.

37. **b**

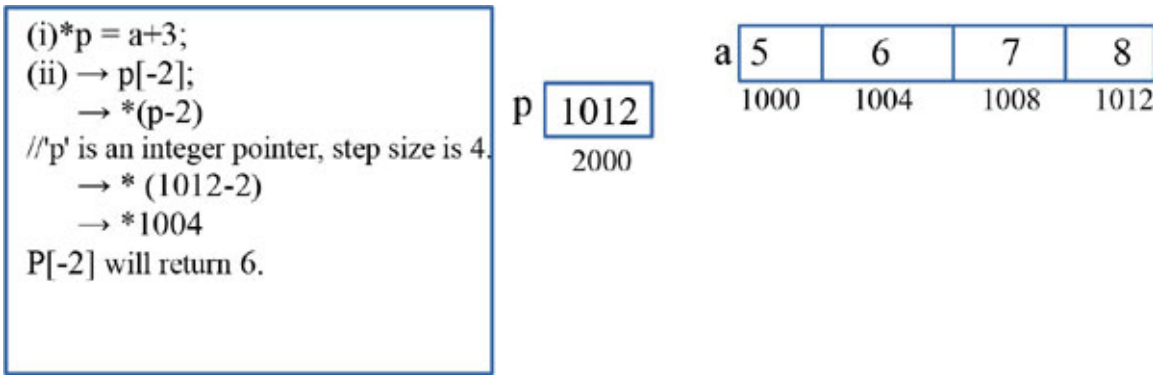
Explanation: Not required.

38. **b**

    Explanation: The base address of an array cannot be updated. Compiler throws an error because of the statement *p=a;*

39. **b**

    Explanation:



    Note: accessing p[-2] is possible, whereas a[-2] will throw an error because the array index cannot be a negative value.

40. **d**

    Explanation: The double type variable has a size of 8 bytes, so, the double-pointer step size is 8. We are adding 2 steps in a double-pointer. Hence it will add 16 to the address 100, the output will be 116.

41. **d**

    Explanation: See next question.

42. **Arithmetic: b, Logical: d and Relational: g**

    Explanation: Not required.

43. **d**

    Explanation: The change of value in the post-increment operator gets updated either a semicolon (;) encounter or any other operators (like + is coming in the given expression) encounter. Like in a given expression *j = 3*i++ + 2*i++; →3*5 + 2*6 → 27*. 'i' has been incremented two times. The program will print 27, 7.

44. **b**

    Explanation: The similar question has been discussed in the explanation of question 17.

45. **b**

    Explanation: Similar questions have been already discussed.

46. **b**

    Explanation: Not required

47. **b**

    Explanation: Read the question carefully, explanation is not required.

48. **d**

    Explanation: Read the question carefully, explanation is not required.

49. **d**

    Explanation: The step size of the void pointer cannot be determined, hence the compiler will throw an error. But if we do *int n = (int*)p-ptr;* → n = 1, output will be option a. For character also it is possible, *char c = (char*)p-(char*)ptr;* → c = 4, output will be option b.

50. **b**

    Explanation: In turbo C compiler will throw an error, In gcc it will give output 0, 4.

51. **d**

    Explanation: The pointer does not allow arithmetic operations with decimal numbers. If we use *int* instead of *float* at line 3, the answer would be 8.

52. **d**

    Explanation: The expression *p = &a;* means that the step size of *&a* is equal to the size of an array (that is, 24). The expression *q = &a+1;* means *&1000+1 → 1024*, where 24 is the step size of *&a*.

53. **d**

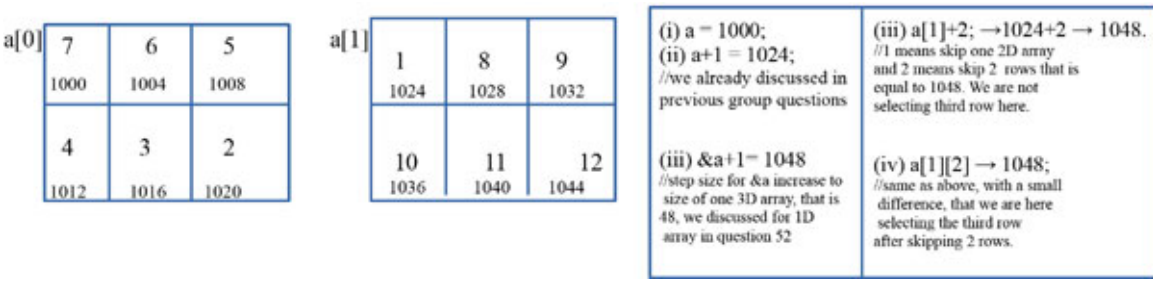
    Explanation: We already discussed several questions similar to this, the memory area for *str1* and *str2* get allocated in *the code area* (that is only readable). Now, we are trying to copy *str2* to *str1*; both are in the read-only area, this is not possible. *Run time error* will occur.

54. **a**

    Explanation: Not required.

55. **1000, 1024, 1048, 1048, 1048**

    Explanation:



56. **d**

    Explanation: Statement: int *s = "hello"; is fine, no error will be reported. The only difference

    between *int *s = "hello";* and *char *s = "hello";* is that *step size*. We discussed several times the step size of an integer pointer and character pointer are 4 and 1 respectively. When we access "s[1]" in the character pointer, it will return ‘e’. When we access "s[1]" using an integer pointer it’ll return ‘o’.

57. **d**

    Explanation: A simple practice question, we discussed several similar questions, explanation not required.

58. **1000, 1016, 1032, 1012, 1032, 3, 3, 1000, 1000, 1048, 1096.**

    Explanation: A practice question.

59. **a**

    Explanation: Explanation is not required.

60. **b, e**

    Explanation: "hello" is still there in read-only memory. We just lose the reference of "hello" at line 3.

Your intention is superior than hard work.

Group-4 Questions

Number of 'C' Question: 60

Other Important Interview Questions: 15

1. **What is the output of the following program?**

```
//basic function pointer example
#include<stdio.h>
int fun(int x, int y)
{
 int z = x*y+x+y*x-y;
 return z;
}
int main()
{
 int (*fun_ptr)(int, int); //line 1
 fun_ptr = fun;
 int x = (int)fun_ptr(1, 2);
 printf("%d", x);
 return 0;
```

```
}
```

(a) 3

(b) 4

(c) 1

(d) syntax error at line 1

2. **What is the output of the following program?**

```
#include<stdio.h>
int fun1(int x, int y)
{
 return x+y;
}
int fun2(int x, int y)
{
 return x*y;
}
int main()
{
 int (*fun_ptr[2])(int, int);
 fun_ptr[0] = fun1;
 int x = fun_ptr[0](4, 5);
 fun_ptr[1] = fun2;
 int y = fun_ptr[1](4, 5);
 printf("%d %d", x, y);
 return 0;
}
```

(a) 4 5

(b) 20 9

(c) 9 20

(d) none of these

3. **Which of the following syntax is /are representing fixed pointer address??**

(a) const <type> *ptr

(b) <type> *const ptr

(c) <type> const *ptr

(d) <type> const *const ptr

4. **The right way of declaring and initializing the function pointer is:**

(a) int (*f_pointer)(int, int) = fName;

(b) int *f_pointer(int, int) = fName;

(c) int *f_pointer(int, int) = &fName;

(d) (int *)f_pointer(int, int) = fName;

5. **What is the output of the following program?**

```
#include<stdio.h>
void fun(char *k)
{
 k++;
 k[2] = ‘m’;
 printf("%c ", *k);
}
int main()
{
 char str[] = "hello";
 fun(str);
 printf("%c ", *str);
 printf("%s ", str);
 return 0;
}
```

(a) e h helmo

(b) e h hello

(c) h e hemlo

(d) e h hemlo

6. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a = 7;
 int *b = &a;
 int **c = &a;
 printf("%d, %u, %u", a, *b, *c);
```

```
 return 0;
}
```

(a) 7, 7, 7

(b) 7, 7, garbage

(c) 7, garbage, garbage

(d) error- compile time

7. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[4] = {10, 20, 30, 40};
 int *p = a;
 int **r = &p;
 p++;
 (**r)++;
 printf("%d", **r);
 return 0;
}
```

(a) 11

(b) 21

(c) 31

(d) 20

8. **Which of the following statement is/are true for the given program?**

```
#include<stdio.h>
int main()
{
 int x=4, y=6, z=7;
 int *p1 = &x, *p2 = &y, *p3 = &z;
 int **ptr = &p2;
 p2 = p1;
 p3 = p2;
 return 0;
}
```

(a) ptr points to x;

(b) ptr points to y;

(c) ptr points to z;

(d) compile time error

9. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[5] = {50, 60, 70, 80, 90};
 int *p = a;
 int *q = &a[2];
 p++;
 int **ptr = p+1;
 ++*ptr;
 printf("%d %d %d", p[2], *q, ptr[0]);
 return 0;
}
```

(a) 80 70 70

(b) 80 71 71

(c) 80 74 74

(d) 80 5 6

10. **Which of the following declaration throws run time error?**

(a) int **ptr = &ptr;

(b) int **ptr = &*ptr;

(c) int **ptr = **ptr;

(d) int **ptr = ptr;

11. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char str[] = "India";
 const char *ptr;
 ptr = str;
 ptr++;
 //++*ptr;
```

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

(a) India

(b) ndia

(c) dia

(d) error- updating constant expression is not allowed

12. **What will be the output if we uncomment the commented line in the above program?**

(a) Jndia

(b) ndia

(c) dia

(d) error- updating of the constant expression not allowed

13. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char str[] = "India";
 char *const ptr;
 ptr = str;
 ptr++;
 printf("%s", ptr);
 return 0;
}
```

(a) India

(b) ndia

(c) dia

(d) error- updating of the constant expression not allowed

14. **What will be the output of the following program?**

```
#include<stdio.h>
int main()
{
 char str[] = "Hello";
```

```
  const char * const ptr;
  ptr = str; //line 3
  //ptr++; //line 4
  ++*ptr; //line 5
  printf("%s", ptr);
  return 0;
}
```

(a) Iello

(b) error- at line 3

(c) error- at line 5

(d) (b) and (c) both

15. **What will be the output if we uncomment the commented line in the above program?**

(a) Jello

(b) no error at line 4

(c) ello

(d) error at line 3, line 4 and line 5.

16. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
   char str[] = "India";
   char *ptr;
   ptr = str;
   ptr++;
   ++*ptr;
   printf("%s", ptr);
}
```

(a) Imdia

(b) Jndia

(c) mdia

(d) odia

17. **Determine the output of the following program if the base address of an array is 1000.**

```
#include<stdio.h>
void fun(int arr[][3])
{
 printf("%u %u %u", *(*(arr+2)+1), (*arr)+2, *
 (((*arr)+1)+1));
}
int main()
{
 int a[][3] = {5, 6, 7, 8, 9, 4, 3, 2, 1};
 fun(&a);
 return 0;
}
```

18. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[][2][4] = {5, 6, 7, 8, 9, 11, 12, 1};
 printf("%d ", *(*(*(a+0))+1)+2);
 return 0;
}
```

(a) 11

(b) 8

(c) 1

(d) error- invalid use of dereferencing operator(*)

19. What is the output of the following program?

```
#include<stdio.h>
#include<string.h>
int main()
{
 char str[] = "India";
 char s[5];
 int i=0, len = strlen(str);
 while(len > 0)
 {
  s[i++] = str[--len];
 }
```

```
 s[i] = ‘\0’;
 printf("%s ", s);
 return 0;
}
```

(a) India

(b) aidnI

(c) blank display

(d) infinite loop

20. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char str1[] = "India";
 char str2[] = "Bharat";
 char *s1 =str1;
 char *s2 =str2;
 while(*s1)
 {
  *s2++ = *s1++;
 }
 printf("%s %s", str1, str2);
 return 0;
}
```

(a) India India

(b) India Indiat

(c) India Bharat

(d) India BIndia

21. **Which of the following function will give the sum of ‘x’ and ‘y’ ?**

```
(i) int add(int x , int y)
{
 return printf("%*c%*c","",x,"",y);
}
(ii) int add(int x , int y)
{
while(y != 0 )
```

```
{
 int c = x&y;
 x = x^y;
 y = c << 1;
 }
 return x;
}
(iii) int add(int x , int y)
{
 if(y == 0)
 return x;
 else
 return add(x^y , (x&y)<<1 );
}
```

(a) i only

(b) ii only

(c) iii only

(d) i and iii only

(e) i, ii ,and iii only

(f) ii and iii only

(g) i and iii only

(h) None of these.

22. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = -3<<4;
 printf("%x", i);
 return 0;
}
```

(a) abcdef33

(b) ffffffff

(c) ffffffd1

(d) ffffffd0

23. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=- -2;
 printf("%d",i);
 return 0;
}
```

(a) 0

(b) 1

(c) 2

(d) error- l-value required

24. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x = 7;
 x = !7>7;
 printf("%d",x);
 return 0;
}
```

(a) 14

(b) 10

(c) 0

(d) 1

25. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char a[]={‘a’,’s’,’c’,’\n’,’i’,’\0’};
 char *b,*q1;
 b=a+3; //line 3
 q1=a;
 printf("%c",++*b + *q1++-30);
```

```
 return 0;
}
```

(a) M

(b) N

(c) O

(d) P

26. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x[2][2][2] = { {10,2,3,4}, {5,6,7,8} };
 int *i,*j;
 i=&x[2][2][2];
 j=**x;
 printf("%d %d",*i,*j);
 return 0;
}
```

(a) 8 10

(b) 7 4

(c) garbage 10

(d) 10 10

27. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 struct student
 {
  int roll=3;
  char name[]="mahi";
 };
 struct student *s1;
 printf("%d ",s1->roll);
 printf("%s",s1->name);
 return 0;
```

```
}
```

(a) 3 mahi

(b) garbage mahi

(c) 3 garbage

(d) error- compile time

28. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 struct student
 {
  int roll;
  struct clg
  {
   char grade;
   struct student *p;
  };
  //line 8
  struct clg *q;
 };
 struct student s;
 s.roll = 10;
 s.clg.grade = ‘A’;
 s.clg.p = NULL;
 printf("%d %c ",s.roll, s.clg.grade);
 return 0;
}
```

(a) 10 A

(b) garbage A

(c) 10 garbege

(d) error- compile time.

29. **What is the output of the following program?**

```
#include<stdio.h>
#define sqr(x) x*x
```

```
int main()
{
 int p;
 p = 32/sqr(8);
 printf("%d",p);
 return 0;
}
```

(a) 32

(b) 0

(c) 64

(d) 0.5

30. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char q[] = "Hello India",*ptr;
 ptr=q;
 while(*ptr!='\0')
 {
  ++*ptr++; // line 4
 }
 printf("%s %s",ptr,q);
 return 0;
}
```

(a) Ifmmp!Joejb

(b) Ifmmp Joejb

(c) Hello India

(d) error- l-value required

31. **What is the output of the following program?**

```
#include <stdio.h>
#define p 10
int main()
{
 #define p 50
```

```
 printf("%d",p);
 return 0;
}
```

(a) 10

(b) 50

(c) error- redefine of macro 'p' is not allowed

(d) 5010

32. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char str[4][20]={"hello", "India", "love", "great"};
 int i;
 char *temp;
 temp=str[2];
 str[2]=str[3];
 str[3]=temp;
 for (i=0;i<4;i++)
  printf("%s",str[i]);
 return 0;
}
```

(a) hello India love great

(b) hello India great love

(c) all garbage values

(d) error- compile time

33. **What is the output of the following program?**

```
#include<stdio.h>
int main( )
{
 int x[2][3][2] = {{{2,4},{7,8},{3,4}},{{2,2},{2,3},{3,4}}};
 printf("%u %u %u %d ",x,*x,**x,***x);
 printf("%u %u %u %d \n",x+1,*x+1,**x+1,***x+1);
 return 0;
}
```

(a) 1000 1000 1000 2 1004 1008 10024 3

(b) 1000 1000 2 2 1024 1008 1004 3

(c) 1000 1000 1000 2 1004 1008 1004 3

(d) 1000 1000 1000 2 1024 1008 1004 3

34. **Determine the output of the following program?**

```
#include<stdio.h>
#include<string.h>
int main()
{
 int i, len;
 char x[] = "India";
 len = strlen(x);
 *x = x[len];
 char *ptr = x;
 for(i=0; i<len; ++i)
 {
  printf("%s ",ptr);
  ptr++;
 }
 return 0;
}
```

35. **What is the output of the following program?**

```
#include<stdio.h>
int main ( )
{
 char *s[] = {"hello", "India", "love", "great"};
 char **ptr[] = {s+3, s+2, s+1, s}, ***p;
 p = ptr;
 **++p;
 printf("%s",*--*++p + 3);
 return 0;
}
```

(a) lo

(b) ia

(c) ve

(d) at

36. **What is the output of the following program?**

```
#include<stdio.h>
int main( )
{
 int x[] = {5,6,7,8,9};
 int *p[] = {x,x+1,x+2,x+3,x+4};
 int **ptr = p;
 ptr++;
 printf("\n %d %d %d", ptr-p, *ptr-x, **ptr);
 return 0;
}
```

(a) 4 4 6

(b) 4 4 8

(c) 1 1 6

(d) error- compile time

37. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x=-5;
 +x; //line 2
 printf("%d %d \n",x,+x);
 return 0;
}
```

(a) -5 -5

(b) 5 5

(c) -5 5

(d) error at line 2

38. **What is the output of the following program?**

```
#include<stdio.h>
#define cal(a , b ,c, d) {\
 int d = 10;\
 int c = 20;\
```

```
 a = b + c + d;\
 }
int main()
{
 int a = 1 , b = 2 , c = 3 , d = 4;
 cal(a ,b ,c ,d);
 printf("%d, %d, %d, %d",a,b,c,d);
 return 0;
}
```

(a) 32, 2, 20, 10

(b) 32, 2, 3, 4

(c) 9, 2, 3, 4

(d) error- compile time

39. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int k=1;
 printf("%d==1 is ""%s",k,k==1?"TRUE":"FALSE");
 return 0;
}
```

(a) error in printf() statement

(b) 0==1 is FALSE

(c) 1==1 is TRUE

(d) 1==1 is garbage.

40. **What is the output of the following program?**

```
#include<stdio.h>
#define max 5
#define int arr1[max]
int main()
{
 typedef char arr2[max];
 arr1 list={0,1,2,3,4}; //line 2
 arr2 name="name"; //line 3
```

```
 printf("%d %s",list[0],name);
 return 0;
}
```

(a) 0 name

(b) 1 name<garbage>

(c) error at line 2

(d) error at line 3

41. **What is the output of the following program?**

```
#include<stdio.h>
char *fun()
{
 char *p = "Hello India";
 return p;
}
int main()
{
 printf(fun());
 return 0;
}
```

(a) garbage value

(b) error- compile time

(c) "Hello India"

(d) "Hello"

42. **What is the output of the following program?**

```
#include<stdio.h>
char *foo1()
{
 char a[ ] = "India";
 return a;
}
char *foo2()
{
 char a[ ] = {‘I’, ‘n’,’d’,’i’,’a’};
 return a;
```

```
}
int main()
{
 char a[] = "India";
 char b[] = {‘I’, ‘n’,’d’,’i’,’a’};
 printf("%u %u",sizeof(a),sizeof(b));
 printf(" %s",foo1()); //line 4
 printf(" %s",foo2()); //line 5
 return 0;
}
```

(a) 6 6 India India

(b) 6 6 garbage garbage

(c) 6 5 India India

(d) 6 5 garbage garbage

43. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char a[4]="kinl";
 printf("%s",a);
 return 0;
}
```

(a) kinl

(b) kinl<some garbage values>

(c) error

(d) kin

44. **Is there any difference between the two given declarations?**

1. int fun(int *a[]) and
2. int fun(int *a[2])

(a) No

(b) Yes

45. **What is the output of the following program?**

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

```
int main()
{
 int x=300;
 char *p = &x;
 *++p=2;
 printf("%d",x);
 return 0;
}
```

(a) 300

(b) 554

(c) 301

(d) 556

46. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = 258;
 int *ptr = &i;
 printf("%d %d", *((char*)ptr), *((char*)ptr+1) );
  return 0;
}
```

(a) 255 0

(b) 255 3

(c) 0 0

(d) 2 1

47. **What is the output of the following program?**

```
#include<stdio.h>
#include<stdlib.h>
#define pi 3.14
void fun();
int main()
{
 printf("%f \n",pi);
 #define pi 3.141516
 fun();
```

```
 return 0;
}
void fun()
{
 printf("%f \n",pi);
}
```

(a) 3.14 3.14

(b) 3.141516 3.141516

(c) 3.14 3.141516

(d) Compile Time Error

48. **What is the output of the following program?**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 int *ptr1, *ptr2;
 ptr1 = malloc(sizeof(int));
 printf("%d ",*ptr1);
 int i;
 ptr2 = (int*)calloc(sizeof(int),1);
 printf("%d",*ptr2);
 return 0;
}
```

(a) 0 0

(b) 0 garbage

(c) garbage 0

(d) garbage garbage

49. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i;
 char a[]="\0";
 char b[] = "\n";
```

```
 if(printf("%s",a )) //line 4
  if(printf("%s",b ))
   printf("1\n");
  else
   printf("2\n");
 else // line 9
  if(printf("%s",b )) // line 10
   printf("3 \n");
  else
   printf("4 \n");
 return 0;
}
```

(a) 1

(b) 2

(c) 3

(d) 4

50. **What is the output of the following program?**

```
#include<stdio.h>
int i = 10;
int main()
{
 int i=i++; //line 1
  printf("%d ",i);
 return 0;
}
```

(a) 10

(b) 11

(c) garbage

(d) error- invalid assignment

51. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 while(1)
```

```
 {
  if(printf("%d",printf("%d", 10)))
   break;
  else
   continue;
 }
 return 0;
}
```

(a) 102

(b) infinite

(c) 12

(d) 101

52. **Explain the below functional pointer declaration.**

```
int ( * fun( float, void ( *pqr) () ) ) ();
```

53. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x = 7;
 int k = x=++x ==7;
 printf("%d %d", k, x);
 return 0;
}
```

(a) 1 8

(b) 0 8

(c) 7 0

(d) 0 0

54. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char p[] = "Hello India";
 char c;
 char *ptr = p;
```

```
 c = ++*ptr++;
 printf("%c",++*ptr++);
 return 0;
}
```

(a) I

(b) f

(c) H

(d) e

55. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char p[] = "hello India";
 char *ptr = p;
 printf("%c",++*(ptr++));
 return 0;
}
```

(a) i

(b) h

(c) l

(d) a

56. **How many times "hello India" will get printed?**

```
#include<stdio.h>
int main()
{
 int i=0;
 while(+(+i--)!=0) //line 2
 {
  printf("hello India\n");
  i-=i++;
 }
 return 0;
}
```

(a) 2

(b) 1

(c) 0

(d) syntax error at line 2

57. **What is the output of the following program?**

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

(a) -127

(b) -128

(c) infinite loop

(d) 0

58. **What is the output of the following program?**

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

(a) -127

(b) -128

(c) infinite loop

(d) 0

59. **What is the output of the following program?**

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

```
 char *ptr;
 char str[] = "India is great";
 ptr = str;
 ptr += 4; //line 4
 printf("%s",ptr);
 return 0;
}
```

(a) India is great

(b) is great

(c) a is great

(d) No output

60. **Which of the following printf prints the value 2 for the given program?**

```
#include<stdio.h>
int main()
{
int a[6][10][15] = {0};
a[4][2][0] = 2;
return 0;
}
```

(a) printf("%d",*(((a+4)+2)+0));

(b) printf("%d",***((a+4)+2)+0);

(c) printf("%d",*(*(*(a+4)+2)+0));

(d) None of these

Other important questions for the Interview are as follows:

1. Write a C program for the addition of two matrices.
2. Write a C program to find the factorial of a given number using recursion.
3. Write a C program for subtraction of two matrices.
4. Write a C program that finds the sum of digits of a given number using recursion.
5. Write a C program that reverses any number using recursion.

6. Write a C program that finds the size of int without using the sizeof operator.
7. Write a C program that finds the size of double without using the size of operator.
8. Write a C program that finds the size of structure without using the size of operator.
9. Write a C program that finds the size of the union without using the size of operator.
10. Write a C program for linear search.
11. Write a C program for binary search.
12. Write a C program for binary search using recursion.
13. Write a C program that passes one dimension array to a function.
14. Write a C program that passes two dimension array to a function.
15. Write a C program for the multiplication of two matrices.

Group-4 Explanations

1. **a**

   Explanation: A demonstration of function pointer.

2. **c**

   Explanation: A demonstration of the array of function pointer where each function is taking two *int* and also returning *int*.

3. **b, d**

   Explanation: Option (a) is a fixed value, pointed by *ptr,* *ptr cannot be updated. Option (b) is fixed pointer address, we cannot update *ptr*; Option (c) is equivalent to option (a). Option(d) is a fixed value and fixed pointer address, the updating of **ptr* and *ptr* is not allowed.

4. **a**

   Explanation: Not required.

5. **a**

   Explanation: A practice question.

6. **a**

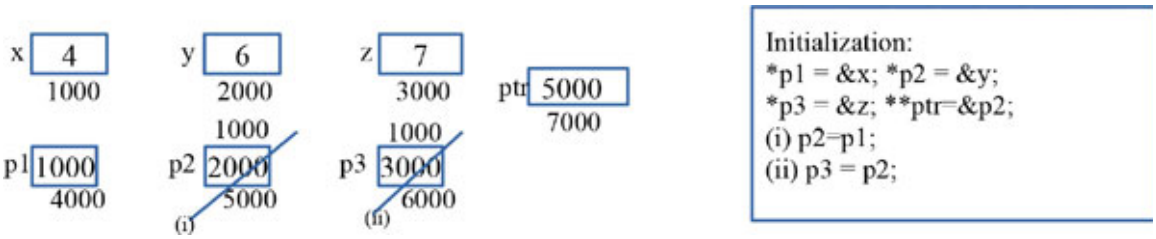
Explanation: Not required.

7. **b**

   Explanation: A practice question.

8. **a**

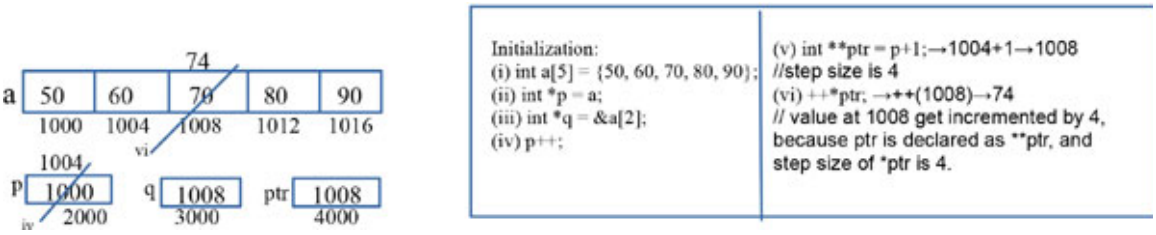
   Explanation:



   At step (i), p2=p1; p2 now holds the ‘p1’, which itself holds the address of x, ptr is still pointing to p2 only. Hence, ptr holds the link to the address of x indirectly.

9. **c**

   Explanation:



   ► p[2] → *(p+2) →*(1004+2);//step size of p is 4 →*1012 → 80 will get printed.

   ► *q →*1008 →74 will get printed.

   ►ptr[0] →*(ptr+0) →*1008 →74 will get printed.

10. **c**

    Explanation: Not needed.

11. **b**

    Explanation: Not required.

12. **d**

    Explanation: Not required.

13. **d**

    Explanation: Not required.

14. **d**

    Explanation: Not required.

15. **d**

    Explanation: Not required. It’s a practice question.

16. **d**

    Explanation: Not required. It’s a simple practice question.

17. **2 1008 7**

    Explanation: A demonstration of a 2D array passing to a function and accessing its elements.

18. **b**

    Explanation: A simple program accessing an element from a 3D array. Given expression is equivalent to a[0][1][2].

19. **b**

    Explanation: Not required. It’s a practice question.

20. **b**

    Explanation: A simple practice program, explanation not required.

21. **f**

    Explanation: Option (i) works fine for the positive number whereas for negative numbers its behavior is unexpected. Option (ii) and (iii) work fine for both positive and negative numbers.

22. **d**

    Explanation: *-3>>4:* It means shifting do 4 left shift of -3. -3 in binary form can be written as 11111111 11111111 11111111 11111101 (-3 is 2’s complement of 3). After 4 left shifting, the binary number becomes 11111111 11111111 11111111 11010000 (-48). In hexadecimal, it can be represented as ff ff ff d0. The %x format specifier specifies that the integer value is printed as a hexadecimal value.

23. **c**

    Explanation: The statement c =- -2; is not a pre-decrement operation. 2 is a constant number, not a variable. The pre-decrement operation is not possible on constants. It is a unary minus operator. This is applied two times on the constant ‘c’. The mathematics rules can be applied here, minus * minus = plus. Hence, the ‘c’ value is 2.

24. **c**

    Explanation: We have already discussed these kind of questions, the precedence of *logical Not operator* (!) is higher than *relational operator* (>). The expression will get evaluated to ® *x = (! 7) > 7* ® *x = 0 > 7* ® *x = 0*. Hence 0 will get printed.

25. **b**

    Explanation: The integer pointer ‘b’ is pointing third element of an array a[3] and ‘q1’ is pointing to first element of an array ‘a’, the first element is ‘a’ whose ascii value is 97. The expression in *printf()* will be evaluated as:

    *++(*b) + (*(q1++)-30)* → *++10 + ((97)++ - 30)* → *11 + (97 - 30)* → *11 + 67* → 78; Because of ‘%c’ format specifier, the output will be ascii value of 78 is ‘N’.

26. **c**

    Explanation: A practice question. Explanation is not required. *Hint:* Variable ‘i’ is pointing out of array memory space.

27. **d**

    Explanation: Initialization of variables is not allowed inside a struct.

28. **d**

    Explanation: In the given question, structure *clg* is nested within the *student*, members of *struct clg* cannot directly be accessed through the instance of *student*, for accessing a member of *clg*, an instance of *clg* must be required. Hence, the compiler will throw an error.

    We can overcome this problem by making an instance of *clg* ("struct clg c;") at line 8 of a given program.

29. **a**

    Explanation: The macro call *sqr(8)* will get substituted as 8*8, so the given expression becomes p = 32/8*8. Since / and ‘*’ have the same precedence, we will go for associativity now. Both are L-R associative, the given expression is evaluated as (32/8)*8 → 4*8 = 32

30. **a**

    Explanation: We already discussed these kinds of questions, the expression ++*ptr++ at line 4 gets expanded as ++*ptr; and ptr++; After the execution of the while loop, each character of a given string will get

incremented by 1. The character ‘H’ becomes ‘I’, ‘e’ becomes ‘f’, blank space gets converted into ‘!’ and so on as per the ASCII convention.

31. **b**

    Explanation: The redefinition of preprocessor directives is allowed, the value will get substituted from the latest preprocessor directive.

32. **d**

    Explanation: We discussed a similar concept in the explanation of question 57, group 2. 2D array is a collection of 1D arrays, Each string get stored in the form of a 1D array. The base address of each 1D array cannot be modified. If we declare given 2D array as *str[4], → *char *str[4]={"hello", "India", "love", "great"};*, the program will give output *"hello India great love"*.

33. **d**

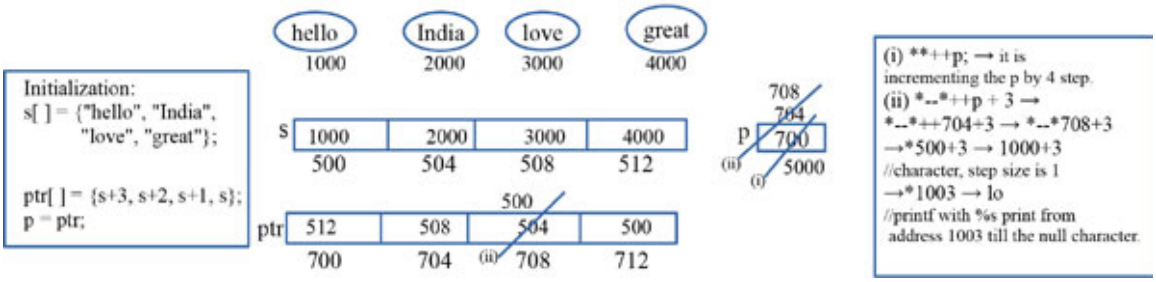
    Explanation: We have already discussed these kinds of questions. Explanation is not required.

34. **<blank space> ndia dia ia a**

    Explanation: A practice question, explanation not required.

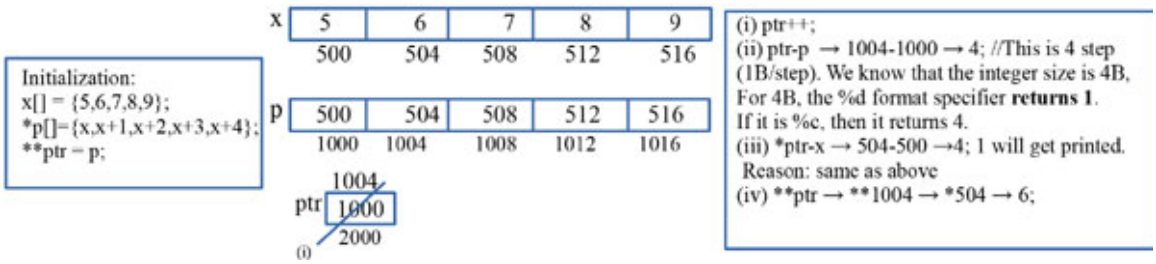
35. **a**

    Explanation:



36. **c**

    Explanation:



37. **a**

Explanation: The unary + operator is called a dummy operator because it doesn't make any change in the expression. We can simply ignore this operator.

38. **b**

Explanation:

The compiler will not throw an error, this is how we can declare multiline macros, each line (except last one) suffixed with '\' represents multiple line macros. The macro cal(a, b, c, d) get substituted as shown on the right-hand side over here:

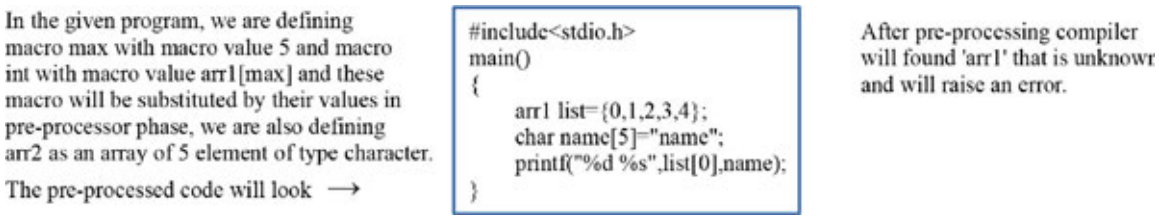
```
#include<stdio.h>
int main(){
int a = 1 , b = 2 , c = 3 , d = 4;
{
        int d = 10;
        int c = 20;
        a = b + c + d;
}
printf("%d %d %d %d",a,b,c,d);
return 0;
}
```

39. **c**

Explanation: In the given program, the two strings, those are separated by white-space, in *printf()* function will concatenate and *printf()* function become : *printf("%d==1 is %s",k,k==1?"TRUE":"FALSE");*. The condition operator "(? :)" evaluates to "TRUE".

40.

Explanation:



41. **c**

Explanation: In a given program, in function *fun()*, the string "Hello India" that is pointed by ‘p’ gets stored in the code area which is shared and available to all the functions in a program. Hence, we can access this string through its address from the main function also.

42. **d**

Explanation: In function *foo1()* and *foo2()*, the memory for array ‘a’ is getting allocated in the local area (stack area) only. After the execution of function *foo1()* and *foo2()*, both return the base address of array ‘a’ and the memory pointed by ‘a’ will not be available in the main function. At line 4 and line 5 in main(), we are trying to access the unavailable memory, it returns garbage or sometimes the same string can also get returned but never be an error. On the other note, array in *foo2()* is not null-terminated, if we print ‘a’ in foo2() with *%s*, it prints "India<garbage>" till the null character encounters.

43. **b**

Explanation: The character array ‘a’ has 4 bytes to store its string and we are also giving exactly 4 characters to it, hence, no place for *termination character(‘\0’)*. When we print the array ‘a’ by *printf()* using "%s" format specifier, it will print characters until it gets *termination character*. So, "kinl<some garbage value>" gets printed.

44. **a**

Explanation: In function argument, only pointer gets passed for an array, if we pass array then also it is treated as a pointer. So, both of the declarations are the same. The second one is good in readability than the first one.

45. **d**

Explanation: The ‘x’ will get stored in memory like 00101100 00000001 00000000 00000000 (little-endian format).

The ‘p’ is a character pointer whose step size is one and it’s pointing to integer variable ‘x’.

At line 3, the operation *(++p) = 2 will break into two steps: (i) p = p + 1 (ii) *p = 2.

In the first step, ‘p’ starts to point to the second byte of ‘x’, and in step (ii) ‘p’ make the second byte of ‘x’ is 2.

Finally, we get ‘x’ in binary representation: 00101100 00000010 00000000 00000000 its decimal equivalent value is 556.

46. **d**

Explanation: We already discussed these kinds of questions. 258 can be represented in binary form as (msb)00000000 00000000 00000001 00000010(lsb). We can access these bytes separately using character pointer, **(char*)iptr* returns 2 because at first location 2 is stored and **(char*)iptr+1*, returns the second byte of integer, 1 will get returned.

47. **c**

Explanation: The C preprocessor parse the program file from top to bottom and treats #define statements like a copy-and-paste operation. Once it encounters line #define pi 3.14, it starts replacing every instance of the word pi with 3.14. The pre-processor does not process C-language scoping mechanisms like parenthesis and curly braces. Once it sees a #define, that definition is in effect until the end of the file is reached, the macro is undefined with #undef, or (as in this case) the macro is re-defined with another #define statement.

48. **c**

Explanation: The memory space allocated by *malloc()* is not initialized, it returns garbage, whereas *calloc()* returns the allocated memory space initialized to zero. The *calloc()* function call takes two parameters, (i) size of a block (variable type size), (ii) number of blocks (for integer it’s 4B) we are allocating, here 1 we are giving, it means we are allocating memory for 1 integer only i.e. 4B.

49. **c**

Explanation: As we discussed earlier, the *printf* returns the total number of characters that it prints on the display. Here, printf is reading one null

character and printing nothing on the display. It means that *printf* is returning 0. The *if* statement at line 4 gets false. The else statement of line 9 gets executed. The *printf* statement at line 10 is printing one character '\n' and returning 1. The if statement of line 10 gets true. Hence, 3 will get printed.

50. **c**

    Explanation: In the given program, two definitions of 'i' are available one is global and one is local to main(). In the main() function, line 1 evaluates in the following way: (i) 'i' declared and initialized with garbage value, and from this point further in the program, if we use 'i', the local 'i' will be considered. (ii) local to main 'i' get assigned with local 'i' only that is garbage, so i contains some garbage value.

51. **a**

    Explanation: In a given program, the inside printf is printing 2 characters ('1' and '0'), the outside printf is taking 2 as message (or input parameter) it will return 1. Hence if the condition gets true, the loop will break in the first iteration only.

52. **Nil**

    Explanation: The *fun* is a pointer function that is taking two parameters. (i) floating-point number (*float*), (ii) a pointer function 'pqr' which can point to a function which is not taking any parameter and returning void type. Finally fun is returning of type int.

53. **d**

    Explanation: The unary operator has more priority than assignment (=) and equal to(==) operator. The expression is evaluated as k = x=++x == 7 → k = x = ((++x) == 7) → k = x = (8==7) → k = x = 0;

54. **b**

    Explanation: A practice question. *Hint:* ++*ptr++ → ++*ptr; ptr = ptr+1; executing twice.

55. **a**

    Explanation: A practice question.

56. **c**

    Explanation: Not required. Hint: '+' becomes dummy operator, i-- is a post decrement.

57. **b**

Explanation: It is to be noticed that the *for* loop is ending with a semicolon. The initial value of ‘i’ is 0. The range of *signed char* is from -128 to +127, when the +127 value is reached by any variable of this type, the signed char rotate to -128 after increment one more time. Hence, the condition of for loop get failed. The value -128 gets printed.

58. **c**

    Explanation: Here, *char* is declared *unsigned*, the range for *unsigned* is from 0 to 255, when it reaches 255 and if any increment occurs its value becomes 0 again. Hence, the i++ never rotates to a value less than 0, then *for* loop condition never gets false. The loop will go infinite.

59. **c**

    Explanation: A practice question. Explanation is not required.

60. **c**

    Explanation: A practice question. Explanation is not required.

The starting point of all achievement is desire.

Group-5 Questions

Number of 'C' Question: 60

Other Important Interview Questions: 15

1. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x = 2,y = 5;
 x = x^y;
 y = y^x;
 printf("%d %d",x,y);
 return 0;
}
```

(a) 5 2

(b) 2 5

(c) 7 7

(d) 7 2

2. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x = 7;
 switch(x)
 {
  default:
   x = 6;
  case 8:
   x--;
  case 7:
   x = x+1;
  case 3:
   x = x-1;
 }
printf("%d \n",x);
return 0;
}
```

(a) 7

(b) 6

(c) 5

(d) 8

3. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x = 2,y = 5;
 x = x^y;
 y = x^y;
 x = x^y;
 printf("%d %d",x,y);
 return 0;
}
```

(a) 2 5

(b) 2 7

(c) 5 2

(d) 7 2

4. **What is the output of the following program?**

```
#include <stdio.h>
int main()
{
 int a[][4] = {3, 4, 5, 6, 7, 8, 9, 10};
 int (*ptr)[4] = a;
 printf("%d %d ", (*ptr)[1], (*ptr)[2]);
 ++ptr;
 printf("%d %d\n", (*ptr)[1], (*ptr)[2]);
 return 0;
}
```

(a) 4 5 8 9

(b) 4 5 7 8

(c) 6 7 4 5

(d) 3 4 5 6

5. **What is the output of the following program?**

```
#include <stdio.h>
int main()
{
 int a[][4] = {3, 4, 5, 6, 7, 8, 9, 10};
 int (*ptr)[4] = a;
 printf("%d %d ", sizeof(ptr), sizeof(*ptr));
 return 0;
}
```

(a) 4, 4

(b) 16 16

(c) 4 32

(d) 4 16

6. **What is the output of the following program?**

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

```
int main()
{
 char c=124;
 c=c+9; //line 2
 printf("%d",c);
 return 0;
}
```

(a) 135

(b) infinite

(c) -8

(d) -123

7. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x, a=5, b=10; //line 1
 x=a+b;
 static int y = x; //line 3
 if(y==x)
  printf("Hello");
 else
  if(y>x)
   printf("Hi");
  else
   printf("Bye");
 return 0;
}
```

(a) Hello

(b) Hi

(c) Bye

(d) error- compile time

8. **What is the output of the following program?**

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

```
{
 int x = 3;
 int y;
 y = sizeof(++x);
 printf("%d %d\n", x, y);
 return 0;
}
```

(a) 4 4

(b) 3 4

(c) 5 4

(d) error- compile time

9. **What is the output of the following program?**

```
#include <stdio.h>
void fun1(int*, int);
void fun2(int*, int);
void (*fun_ptr[2])(int*, int);
int main()
{
 int x = 4;
 int y = 7;
 fun_ptr[0] = fun1;
 fun_ptr[1] = fun2;
 fun_ptr[0](&x, y);
 printf("%d %d ", x, y);
 fun_ptr[1](&x, y);
 printf("%d %d\n", x, y);
 return 0;
}
void fun1(int *a, int b)
{
 int temp = *a;
 *a = b;
 b = temp;
}
void fun2(int *a, int b)
{
```

```
  int temp = *a;
  *a = b;
  b = temp;
 }
```

(a) 5 5 5 5

(b) 3 5 3 5

(c) 4 4 4 4

(d) 7 7 7 7

10. **Write the condition in the if block such that the given program will give the output "Hello India".**

```
#include<stdio.h>
int main()
{
 if(<condition>)
 {
  printf("Hello ");
 }
 else
 {
  printf("India\n");
 }
 return 0;
}
```

11. **What is the output of the given program?**

```
#include<stdio.h>
int main()
{
 char str[] = "India";
 str[0] += 32; //line 2
 printf("%s",str);
 return 0;
}
```

(a) India

(b) garbage

(c) india

(d) error- compile time

12. **If the binary equivalent of 5.375 in normalized form is 0100 0000 1010 1100 0000 0000 0000 0000, What will be the output of the program?**

```
#include<stdio.h>
#include<math.h>
int main()
{
 float a=5.375;
 char *p= (char*)&a; //line 2
 int x = p[2]^p[3]; //line 3
 printf("%d ", x);
 return 0;
}
```

(a) 5

(b) -20

(c) -23

(d) 20

13. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
void foo(char *);
int main()
{
 char a[100] = {0};
 printf("%u %u",sizeof(a),strlen(a));
 return 0;
}
```

(a) 100 100

(b) 100 99

(c) 101 1

(d) 100 0

14. **What is the output of the following program?**

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

```
int fun()
{
 int a = 10;
 return a;
}
int main()
{
 int *a = NULL;
 *a = fun()
 printf("%d ",*a );
 return 0;
}
```

(a) 10

(b) garbage

(c) error- compile time

(d) error- run time

15. **What is the output of the following program? Suppose the base address of an array ‘a’ is 1000.**

```
#include<stdio.h>
int main()
{
 int a[3][4][5]={0};
 printf("%u %u",&a , &a+1);
 return 0;
}
```

(a) 1000 1012

(b) 1000 1020

(c) 1000 1004

(d) 1000 1240

16. **What is the output of the following program?**

```
#include<stdio.h>
int fun()
{
 int a = 10;
```

```
 return a;
}
int main()
{
 int *a = fun();
 printf("%d %u ",a, a+1 );
 return 0;
}
```

(a) 10 14

(b) garbage garbage

(c) error- compile time

(d) 10 11

17. **What is the output of the following program?**

```
#include <stdio.h>
int main()
{
 struct student
 {
  int a;
  int b;
  int c;
 };
 struct student s1 = { 3, 5, 6 }; //line 7
 struct student *p = &s1; //line 8
 printf("%d\n", *((int*)p+1)); //line 9
 return 0;
}
```

(a) 3

(b) 5

(c) 6

(d) 7

18. **What is the output of the following program?**

```
#include <stdio.h>
void fun(int);
```

```
int main()
{
 int x = 3;
 fun(x);
 return 0;
}
void fun(int i)
{
 if (i > 0)
 {
  fun(--i);
  printf("%d ", i);
  fun(--i);
 }
}
```

(a) 0 1 2 0

(b) 0 1 2 1

(c) 1 2 0 1

(d) 0 2 1 1

19. **What is the output of the following program?**

```
#include <stdio.h>
int main(void)
{
 char a[5] = { 6, 7, 8, 9, 10 };
 char *ptr = (char*)(&a + 1);
 printf("%d %d\n", *(a + 1), *(ptr - 1));
 return 0;
}
```

(a) 7 9

(b) 7 6

(c) 7 10

(d) 7 garbage

20. **What is the output of the following program?**

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

```
void fun(int[][3]);
int main(void)
{
 int a[3][3] = { {10, 20, 30}, {40, 50, 60}, {70, 80, 90} };
 fun(a);
 printf("%d\n", a[2][1]);
 return 0;
}
void fun(int b[][3])
{
 ++b;
 b[1][1] = 777;
}
```

(a) 777

(b) 80

(c) 70

(d) error- base address of an array cannot be updated

21. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = 0;
 switch(i)
 {
  default:
   i = 4;
  case 6:
   i--;
  case 5:
   i = i+1;
   break;
  case 1:
   i = i-1;
   break;
 }
 printf("%d \n",i);
```

```
 return 0;
}
```

(a) 3

(b) 4

(c) 5

(d) 0

22. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a = 2;
 if(a == (1,2))
  printf("Hello ");
 if(a == 1,2)
  printf("India");
 return 0;
}
```

(a) Hello

(b) World

(c) Hello India

(d) error- compile time

23. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a = 3,4; //line 1;
 int b = (3,4);
 if(a == b)
  printf("Equal");
 else
  printf("Not Equal");
 return 0;
}
```

(a) Equal

(b) Not Equal

(c) error- run time

(d) error- compile time

24. **What is the output of the following program?**

```
#include<stdio.h>
void fun(char *);
int main()
{
 char *str = "Hello India";
 fun(str);
 printf("%s",str);
 return 0;
}
void fun(char *ptr)
{
 while(*ptr)
 {
  *ptr += 1;
  ptr++;
 }
}
```

(a) Hello India

(b) Ifmmp Jmejb

(c) error- compile time

(d) error- run time (segmentation fault)

25. **What is the output of the following program?**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 char a[] = "Hello India";
 char *ptr = a;
 int len = 0;
 while(*(ptr))
 {
```

```
  len++;
  ptr++;
 }
 printf("%d",len);
 return 0;
}
```

(a) error- run time (Segmentation Fault)

(b) error- compile time

(c) 11

(d) 0

26. **What is the output of the following program?**

```
#include<stdio.h>
int a = 10;
void fun2()
{
 int *p;
 {
  int a=5;
  p = &a;
 }
 a++;
 ++*p;
 printf("%d ", *p);
}
void fun1()
{
 a = 1;
 printf("%d ",a);
 a++;
 fun2();
 int a=4;
 printf("%d ", --a);
}
int main()
{
 fun1();
```

```
 fun2();
 pintf("%d ", a);
 return 0;
}
```

(a) 1 3 5 3 4

(b) 1 6 3 6 4

(c) 1 6 3 5 4

(d) 1 3 3 5 4

27. **Differentiate the given function declarations?**

(i) int fun(int a[]);

(ii) int fun(int a[2]);

(iii) int fun(int *a);

28. **What is the output of the following program?**

```
#include<stdio.h>
void fun(int arr[])
{
 int a, b, c, d;
 a = ++arr[1];
 a ++;
 b = arr[1]++;
 c = a++ - arr[1];
 d = a – arr[1];
 printf("%d %d %d %d", a, b, c, d);
}
int main()
{
 int a[] = {5, 6, 7, 8, 9};
 fun(a);
 return 0;
}
```

(a) 7 7 0 1

(b) 9 7 0 1

(c) 8 7 0 1

(d) 9 7 0 8

29. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 printf(7+"Conceptual C q\0uestions");
 return 0;
}
```

(a) Conceptual C question

(b) Conceptual C q

(c) u al C q

(d) error- compile time

30. **What is the output of the following program?**

```
#include<stdio.h>
void change(char (*ptr)[6])
{
 (*ptr)[2] = ‘x’; // line 1
 (*ptr+1)[2] = ‘x’; // line 2
 ptr++; // line 3
 (*ptr)[2] = ‘x’; // line 4
 (*ptr+1)[2] = ‘x’; // line5
}
int main()
{
 char str[][6] = {"hello", "India"};
 change(str);
 printf("%s %s", str[0], str[1]);
 return 0;
}
```

(a) hello India

(b) hexlo Ixdia

(b) xello Ixdia

(d) hexxo Inxxa

31. **What is the output of the following program?**

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

```
void change(char (*ptr)[6])
{
 char temp;
 temp = (*ptr)[1];
 (*ptr)[1] = *ptr[0];
 (*ptr)[0] = temp;
 ptr++;
 temp = (*ptr)[1];
 (*ptr)[1] = *ptr[0];
 (*ptr)[0] = temp;
}
int main()
{
 char str[][6] = {"Hello", "India"};
 change(str);
 printf("%s %s", str[0], str[1]);
 return 0;
}
```

(a) Hello Indi

(b) India Hello

(b) eHllo nIdia

(d) Iello Hndia

32. **What is the output of the following program?**

```
#include<stdio.h>
void change(char *ptr)
{
 ptr[2] = ‘x’;
 ptr[6] = ‘x’;
}
int main()
{
 char str[][6] = {"hello", "India"};
 change(str);
 printf("%s %s", str[0], str[1]);
 return 0;
}
```

(a) hello India

(b) hexlo xndia

(c) xello Ixdia

(d) hexlo India

33. **What is the output of the following program?**

```
#include<stdio.h>
void change(char *ptr[6])
{
 char *temp=ptr[0];
 ptr[0] = ptr[1];
 ptr[1] = temp;
}
int main()
{
 char *str[6] = {"hello", "India"};
 change(str);
 printf("%s %s", str[0], str[1]);
 return 0;
}
```

(a) hello India

(b) India hello

(c) Iello hindia

(d) Iello Hndia

34. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *str1 = "hello";
 char *str2 = "hello";
 char str3[] = "hello";
 char str4[] = "hello";
 if(str1 == str2)
  printf("one ");
 else
  printf("zero ");
```

```
 if(str3 == str4)
  printf("yes ");
 else
  printf("no ");
 return 0;
}
```

(a) one yes

(b) zero no

(c) zero yes

(d) one no

35. **What is the output of the following program? Assume the First string "hello" is getting address 1000.**

```
#include<stdio.h>
int main()
{
 printf("%u %u ", &"hello", &"hello");
 printf("%s", &"hello");
 return 0;
}
```

(a) error- compile time

(b) 1000 2000 3000

(c) 1000 2000 1000

(d) 1000 1000 hello

36. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char str[8] = "India";
 str[6] = ‘x’; //line 2
 printf("%s", str); //line 3
 return 0;
}
```

(a) India x

(b) Indiax

(c) India

(d) India\0x

37. **Which of the following options return the value 9 from an array ‘a’.**

```
#include<stdio.h>
int main()
{
 int a[][2][3] = {3, 4, 5, 6, 7, 8, 9, 10, 1, 2, 11, 12};
 return 0;
}
```

(a) a[1][0][0]

(b) ***(a+1)

(c) *a[1][0]

(d) **a[1]

(e) all of these

38. **Which of the following statement is/are true about the given declaration?**

char a[]= "India", *b = "India";

(i) a is a constant pointer to a non constant string.

(ii) b is a non constant pointer to a constant string.

(iii) a is a non constant pointer to a non constant string.

(iv) In both the string the ‘\0’ get suffixed automatically.

(a) i, ii

(b) ii, iii, iv

(c) i, iv

(d) i, ii, iv

39. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *str1 = "India is great";
 int i=0 , j =0;
 char str2[10] = "hello";
 str2[j++] = str1[i++];
```

```
 str2[j++] = i++[str1];
 str2[j++] = (++i)[str1];
 str2[i++] = '\0';
 printf("%s %d", str2,j);
 return 0;
}
```

(a) Ini 3

(b) hnd 3

(c) Ini 4

(d) Ind 4

40. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x = 0xabcdef;
 char p = x;
 printf("%d\n",p);
 return 0;
}
```

(a) ab

(b) ef

(c) -17

(d) 0

41. **What is the output of the following program?**

```
#include<stdio.h>
#define abc() 30
#define xyz 20)
#define num abc()-(xyz)
int main()
{
 int x = 10*(num; //line 1
 printf("%d", x);
 return 0;
}
```

(a) 100

(b) 280

(c) 320

(d) error- compile time

42. **What is the output of the following program?**

```
#include<stdio.h>
struct xx
{
 int a;
 char b;
 void *p;
};
int main()
{
 struct xx x ;
 printf("%d", sizeof(x.b));
 return 0;
}
```

(a) 12

(b) 4

(c) 1

(d) 8

43. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[5];
 int *q = &a[1];
 int *p = &a[4];
 printf("%d %d", p-q, &p-&q);
 return 0;
}
```

(a) 3 unknown

(b) 4 unknown

(c) 12 unknown

(d) 1 unknown

44. **What is the output of the following program?**

```
#include<stdio.h>
void fun(char str[3][20])
{
 char str2 = *(str[2]+8);
 *(str[2]+8) = *(str[0]+9);
 *(str[0]+9) = str2;
}
int main()
{
 char table[3][20] = {"India is great", "Hello masters",
 "Love to work"};
 fun(table);
 printf("%s %s", table[0], table[2]);
 return 0;
}
```

(a) wndia is great Love to work

(b) India is wreat Love to gork

(c) India is great Love to work

(d) India iswgreat Love to gork

45. **Determine the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[] = {10, 20, 30, 40, 50};
 int*p[] = {a, a+1, a+2, a+3, a+4};
 int **ptr = p;
 ptr++;
 printf("%d %d %d\n", ptr-p, *ptr-a, **ptr);
 *(++ptr);
 printf("%d %d %d\n", ptr-p, *ptr-a, **ptr);
 ++(*ptr);
 printf("%d %d %d\n", ptr-p, *ptr-a, **ptr);
 *ptr++;
```

```
 printf("%d %d %d\n", ptr-p, *ptr-a, **ptr);
 return 0;
}
```

46. **Determine the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[] = {0, 1, 2, 3, 4};
 int*p[] = {a, a+1, a+2, a+3, a+4};
 int **ptr = p;
 **ptr++;
 printf("%d %d %d\n", ptr-p, *ptr-a, **ptr);
 *(++(*ptr));
 printf("%d %d %d\n", ptr-p, *ptr-a, **ptr);
 ++**ptr;
 printf("%d %d %d\n", ptr-p, *ptr-a, **ptr);
 return 0;
}
```

47. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char p[] = "Conceptual C questions";//line 1
 printf("%s", p+p[4]-p[3]);
 return 0;
}
```

(a) Conceptual C questions

(b) C questions

(c) nceptual C questions

(d) ptual C questions

48. **What is the output of the following program?**

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

```
 char p[20];
 char *s = "string";
 int length = strlen(s);
 int i;
 for(i=0; i<length; i++)
 {
  p[i] = s[length-i];
 }
 printf("%s", p);
 return 0;
}
```

(a) string

(b) nothing gets printed

(c) gnirts

(d) none of these

49. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *str = "India";
 printf("%c", str);
 return 0;
}
```

(a) India

(b) I

(c) ndia

(d) error- compile time

50. **What is the output of the following program?**

```
#include<stdio.h>
int fun(int n)
{
 static int r=0;
 if(n<=0) return 1;
 if(n>3)
```

```
 {
  r = n;
  return fun(n-2)+2;
 }
 return fun(n-1)+2;
}
int main()
{
 printf("%d", fun(5));
 return 0;
}
```

(a) 8

(b) 9

(c) 10

(d) 11

51. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 struct a
 {
  char ch[7];
  char *str;
 };
 struct b
 {
  char *c;
  struct a ssl;
 };
 struct b s2 = {"Raipur", "Kanpur", "Jaipur"};
 printf("%s %s ", s2.c, s2.ssl.str);
 printf("%s %s ", ++s2.c, ++s2.ssl.str);
 return 0;
}
```

(a) Raipur Jaipur aipur aipur

(b) Jaipur Raipur aipur aipur

(c) Raipur Jaipur Raipur aipur

(d) Raipur Jaipur aipur Raipur

52. **What is the output of the following program?**

```
#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},
 {"Kanpur", 3, a}};
 struct s1 *ptr = a;
 printf("%s %s %s %s", a[0].z, (*ptr).z, ptr->z, a[2].p->z);
 return 0;
}
```

(a) Nagpur Raipur Nagpur Raipur

(b) Nagpur Nagpur Nagpur Nagpur;

(c) Nagpur Raipur Kanpur Nagpur

(d) Nagpur Raipur Kanpur Kanpur

53. **What is the output of the following program?**

```
#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},
 {"Kanpur", 3, a}};
 struct s1 *ptr = a;
```

```
 printf("%s ", ++(ptr->z));
 printf("%s ", a[(++ptr)->i].z);
 printf("%s ", a[--(ptr->p->i)].z);
 printf("%d ",--a[2].i);
 return 0;
}
```

(a) agpur Kanpur Kanpur 1

(b) aipur Kanpur aipur 2

(c) Raipur Kanpur Kanpur 3

(d) aipur Kanpur aipur 3

54. **Determine the output of the following program?**

```
#include<stdio.h>
struct Test
{
 int i;
 char *c;
};
int main()
{
 struct Test st[] = {5, "become", 4, "better", 6, "jungle",
 8, "ancestor", 7, "brolter"};
 struct Test *p = st;
 p+=1;
 printf("%s ", ++(p++->c));
 printf("%c ", *p++->c);
 printf("%d ", ++p->i);
 printf("%s ", p[0].c);
 printf("%s ", p->c);
 return 0;
}
```

55. **What is the output of the following program?**

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

```
 char ***p = ptr;
 printf("%s ", **++p);
 printf("%s ", *--*++p+1);
 return 0;
}
```

(a) ice ce

(b) green one

(c) cone ce

(d) cone ice

56. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a1[] = {6, 7, 8, 34, 67};
 int a2[] = {25, 56, 28, 29};
 int a3[] = {-12, 27, -34};
 int *x[] = {a1, a2, a3};
 int **a = x;
 printf("%d ", a[0][2]);
 printf("%d ", *a[2]);
 printf("%d ", *++a[0]);
 printf("%d", *(++a)[0]);
 return 0;
}
```

(a) 6 25 -12 7

(b) 8 -12 7 25

(c) 8 -12 27 25

(d) 6 25 -12 7

57. **Determine the output of the following program?**

```
#include<stdio.h>
int main()
{
 char c[100];
 int i,*a = (int *)c;
```

```
 float *b = (float *)c;
 char *d = (char *)c;
 int (*e)[5] = (int *)c;
 for(i=1; i<=5; i++)
 {
  printf("%u, %u, %u, %u, %u\n", c, a, b, d, e);
  a++; b++; d++; e++;
 }
 return 0;
}
```

58. **What is the output of the following program?**

```
#include<stdio.h>
#include<stdlib.h>
void fun(int **i)
{
 (*i)[4]=99;
 (*i)++; // line 2
}
int main()
{
 int *p =(int*)malloc(sizeof(int)*10);//line 4
 fun(&p);
 printf("%d\n", p[3]);
 free(p); //line 7
 return 0;
}
```

(a) 99;

(b) 0

(c) garbage

(d) 99, error- run time due to line 7

59. **Given a function pointer ‘f’ pointing to a function that is taking two integers as its input parameter and returning integer type pointer.**

int *(*f)(int, int);

Demonstrate it with some example?

60. **Given a function pointer ‘funptr’ pointing to a function which is taking an integer and function pointer ‘f’ as its input parameter and returning void type.**

    ‘f’ is a pointer pointing to a function that is taking two integers as its parameter and returning integer pointers (int *).

    void (*funptr)(int, int*(*f)(int, int));

    Demonstrate it with an example?

Other important questions for the Interview are as follows:

1. Write a C program that finds the sum of a diagonal element of a matrix.
2. Write a C program that finds the transport of a matrix.
3. Write a C program that converts the string from upper case to lower case.
4. Write a C program that deletes all consonants from a given string.
5. Write a C program to sort the characters of a string.
6. Write a C program for concatenation of two strings without using string.h header file.
7. Write a C program that finds the length of a string using a pointer.
8. Write a C program that prints the string from a given character.
9. Write a C program that reverses a string
10. Write a C program that reverses a given string using recursion.
11. Write a C program that copies the given string to another without using strcpy().
12. How to compare two strings in c without using strcmp
13. Write an algorithm for scanf() using the concept of system call.
14. Write an algorithm for printf() using the concept of system call.
15. Review the volatile keyword in C?

Group-5 Explanations

1. **d**

    Explanation: ‘^’ is a bit wise XOR operator. In a given program ‘x’ can be written in binary form as x = 0010 and ‘y’ can be written as 0101. The

expression *x = x^y* is evaluated as → *x = 0010 ^ 0101 → 0111 = 7* and expression *y = x^y* is evaluated as: *y = 0101 ^ 0111 → 0010 = 2*.

2. **a**

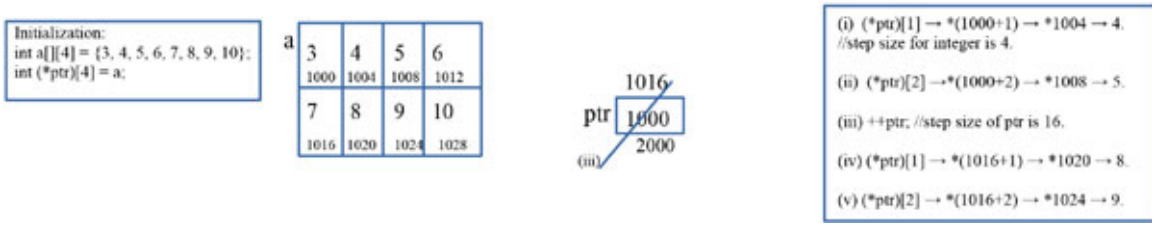
   Explanation: C*ase 7* will execute first, it makes *x = 8*; there is no break statement after *case 7*. As we discussed earlier, all the cases get executed after the match if there is no *break* statement. Then, *case 3* makes *x = 7* again.

3. **c**

   Explanation: It is also one of the application of XOR operation to swap two numbers. The variable ‘x’ and ‘y’ can be written as in binary form 0010 and 0101 respectively. The first expression *x = x^y* is evaluated as *→ x = 0010 ^ 0101 → x = 0111*. The second expression *y = x^y* (updated ‘x’ will be considered) is evaluated as *y = 0111^0101 → y = 0010= 2*. The third expression *x = x^y* is evaluated as *x = 0111^0010 → x = 0101 = 5*. Hence, the ‘x’ and ‘y’ get updated with swapped initial values.

4. **a**

   Explanation: It is a new concept, the ‘ptr’ is pointing to the first 4 elements of an array, If we do (*ptr)[0], 3 will get returned, and so on. The difference between (*ptr)[4] and **ptr is just the step size. Here, the step size for ptr is 4*sizeof(int). Here, if we do ptr++, it points to the next row of an array. It is recommended to execute this program in your system and play around with it. The ‘ptr’ is a pointer to an array.



5. **d**

   Explanation: The pointer ‘ptr’ is a normal pointer, it is depending on the number of address lines in a machine. As we assumed that we are having 32 address lines. The operator sizeof(ptr) returns 4. The *ptr contains a row of an array. The sizeof(*ptr) returns the size of the row, which is 16 here.

6. **d**

Explanation: We already discussed a similar version of this question in group 4. The range of *char* is from -128 to +127, when the +127 value is reached by any variable of this type, the char rotate to -128. In a given program at line 2, the expression becomes c = 124+9 → 127+6 → it gets rotated and make value -128+5, it becomes -123.

7. **d**

   Explanation: The initialization of static variable with some other variable cannot be done, the reason is that all static variables must be initialized before the execution of *main()*. The expression at line 1 *int x = 10;* will get executed only after the execution of *main()*. The expression at line 3, *static int y = x;* the value x is not known, so it throws an error. The expression *static int y = 10;* is allowed, because the value 10 is constant that is known before execution of *main()*.

8. **b**

   Explanation: We already discussed these kinds of questions, *sizeof()* operator doesn't evaluate any expression.

9. **d**

   Explanation: One more demonstration of the function pointer array. We already discussed, the explanation is not required.

10. **Nil**

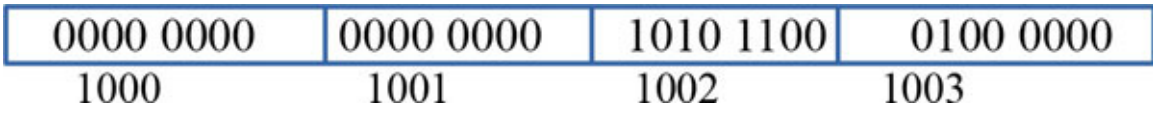
    Explanation: The condition can be: *0 == printf("Hello ");*

11. **c**

    Explanation: The expression at line 2 can be written as *str[0] = str[0] + 32*; *str[0] = 'I' + 32* → str[0] = 73+32 = 105 which is the ascii value of 'i'.

12. **b**

    Explanation: The normalized form value of 5.375 is 0100 0000 1010 1100 0000 0000 0000 0000, as we discussed in earlier groups, it gets stored in memory in the group of 1B in little-endian mode. But for our convenience, we considered as the storage mechanism is big-endian. Refer to explanation of question 25, group 2.



After initializing p at line 2, it become : p[0] = *1000; p[1] = *1001; p[2] = *1002; p[3] = *1003. The expression at line 3 can be evaluated as p[2] ^ p[3] = 1010 1100 ^ 0100 0000 → 1110 1100 (which it the value of -20 in binary form).

13. **d**

    Explanation: In the given program, character array ‘a’ is initialized with {0}. In C, if we initialize some starting elements of an array then all remaining elements get initialized with 0 (by default). The *sizeof* operator returns 100, the amount of memory taken by array ‘a’. The *strlen()* library function returns 0 because the first element of an array ‘a’ is 0. Since 0 is the ASCII value of ‘\0’ (*null character*) and *strlen()* does not consider null character. Hence, strlen() will not count and return 0.

14. **d**

    Explanation: In the given program, it compiles fine but we are trying to access memory location 10 after the function call, which could not be a valid memory address. Hence, a run time error gets thrown.

15. **d**

    Explanation: We already discussed these kinds of questions, step size of &a is equal to the size of the 3D array (60 elements x 4B = 240B). Hence, &a+1 will return address 1240.

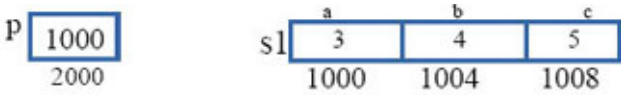
16. **a**

    Explanation: In the main function, ‘x’ contains address 10. In the printf statement, a+1, return 14 because as per our assumptions the step size of an integer pointer is 4.

17. **b**

    Explanation:

(i) *((int*)p+1));
//First we are type casting structure type p to int * type. The step size for integer pointer is 4.
→ *((int*)p+1) → *(1000+1) → *1004 → 5.



The structure instance can be initialize as shown at line 7. At line 8, a structure pointer is pointing to 's1', an instance of structure student. The

memory layout for structure variable 's1' is like:

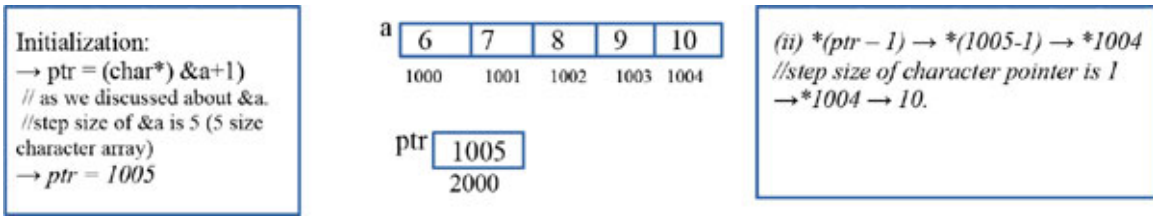
At the first 4B block an integer variable 'a' is stored with value 3. At second 4B block another integer variable 'b' is stored with value 5 and at the last block of 4B 'c' is stored with value 6.

18. **a**

    Explanation: A practice question. We have already discussed these kinds of questions.

19. **c**

    Explanation:



20. **a**

    Explanation: In a given function *fun()*, 'b' is not the base address of an array, so we can update it. As we discussed, there is only a pointer that gets passed for an array during the function call. After the statement, *++b;* the variable 'b' started pointing to the 2nd row of an array 'a'. The remaining array for 'b' is left with *b[2][2]* which includes the 2nd and 3rd row of the given array. The indexing in b[2][2] starts from as usual 0..1 rows and 0..1 columns. With respect to this b[1][1] is equivalent to array a[2][1]. Hence, a[2][1] gets updated with 777.

21. **b**

    Explanation: We already discussed these kinds of questions, *default* will execute when none of the *case* labels matches with the *switch* condition. After the execution of the *default* statement, all the subsequent cases following *default* also get executed without matching the switch condition till the end of switch block or break encounters.

22. **c**

    Explanation: A practice question, we already discussed the question on parenthesis operator.

23. **c**

Explanation: At line 1, there is no such syntax, the compiler throws an error. It is similar to *int a=3, 4;*

24. **d**

    Explanation: A practice question, we already discussed these kinds of questions. It is not allowed to update in the code area (read-only area).

25. **c**

    Explanation: Not required, a practice question.

26. **b**

    Explanation: A good practice question, explanation not required.

27. **Nil**

    Explanation: All the given declarations are the same because when we pass an array to a function it immediately split into pointers. We already discussed for 2D and 3D arrays.

28. **b**

    Explanation: A good practice question, explanation not required.

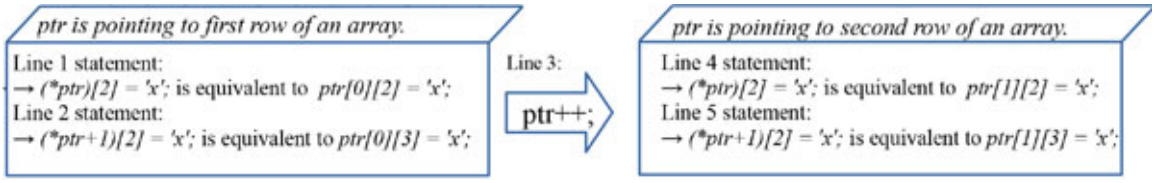
29. **c**

    Explanation: Not required. The '%s' format specifier only returns the string just before the null character encounters.

30. **d**

    Explanation: This is one of the good questions. For basic about "pointer to an array", refer to question 4. Be thorough with that.

    ► (*ptr)[6] can point to a 2D array which has 6 columns, here ptr is pointing to the first row of an array ◄



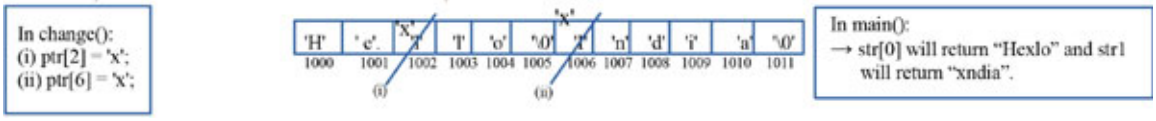
    ► The given array get updated as {"hexxo", "Inxxa"};

31. **c**

    Explanation: A practice question following the concept of the above question. Explanation is not required.

32. **b**

Explanation: We are passing a 2D array to a single pointer. as we know the 2D array in C is stored in row-major order. The 2D array has been converted into a 1D array as follows:



33. **b**

    Explanation: One more good question for practice. Explanation is not required.

34. **d**

    Explanation: Refer to explanation of question 9, group 1. The read-only area is limited, so in order to utilize the memory allocation, the compiler first checks whether the same string gets allocated in the memory or not, if yes the memory manager simply returns the previous address otherwise it allocates the new memory and returns new address. Here, str1 and str2 contain the same string, the string gets stored in the read-only area of memory. Hence the address is the same. str3[] and str4[] get memory allocated in stack area, each of them get separate address space. Hence, they get stored at different locations.

35. **d**

    Explanation: Not required. *Hint*- In the *printf()* function only one instance of "hello" is getting created in read-only memory.

36. **c**

    Explanation: The character array is initialized with the string "India". In array *str,* "India" is stored from array index 0 to 4 and the remaining elements from array index 5 to 7 get a null character (‘\0’) as initialization value (Not garbage). At line 2, ‘x’ is stored at array index 6, but at str[5] we have a null character. At line 3, printf() prints string from str[0] to the index until it get a null character that is at index 5 here. Hence, "India" gets printed.

37. **e**

    Explanation: Not required.

38. **d**

    Explanation: Not required.

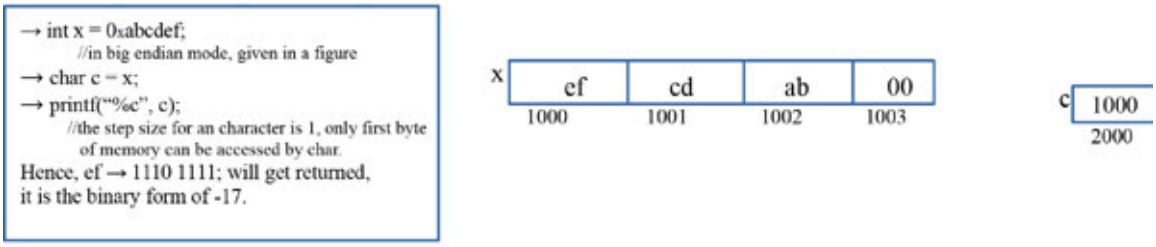
39. **a**

Explanation: Not required.

40. **c**

Explanation:

In gcc, the integer is implicitly typecast into character. We can see this in statement *char p = x;* This question is one of the versions of the little-endian storage mechanism. The *x = 0xabcdef;* is in hexadecimal form, it gets stored in memory in little-endian mode. We have already discussed the answers to these kinds of questions assuming storage in big-endian mode.



41. **a**

Explanation: In the given program all macros will be substituted with their definition at preprocessing phase. The preprocessed code after substituting *num* macro is as follows:

```
#include<stdio.h>
int main()
{
     int x = 10*(30-20); //line 1
     printf("%d", x);
     reture 0; At line 1 ‘x’ is evaluated to 100.
}
```

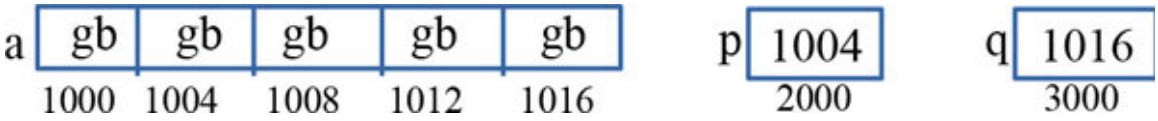
42. **c**

Explanation: The *sizeof()* operator returns the number of bytes that are allocated to its operand (data type of expression). In the given program, the operand is ‘x.b’, ‘b’ is character member of structure ‘’xx’. Hence, the *sizeof()* return number of bytes required to store character ‘b’ is 1.

43. **a**

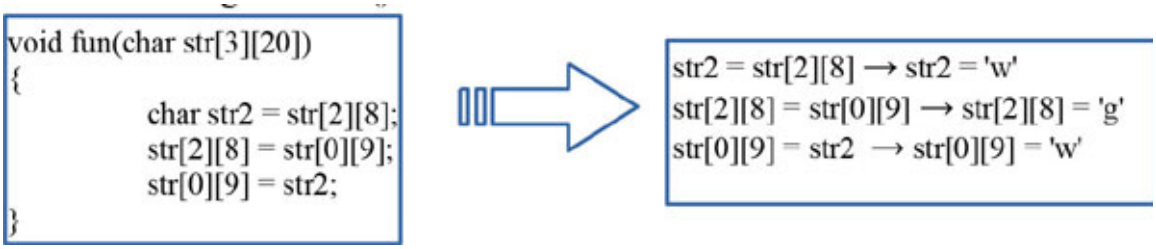
Explanation: Let the array ‘a’ get stored in memory address from 1000 and initialized with garbage values (GB). The pointers ‘p’ and ‘q’ are pointing array elements a[1] and a[4]. If we subtract two pointers pointing to the same array, we get the difference of blocks they are

pointing to, e.g. the pointer ‘p’ is pointing to 2nd block of array ‘a’ and pointer ‘q’ is pointing 5th block of array ‘a’ then the difference of p and q gives 3. The pointer ‘p’ and ‘q’ are stored at a random location, it is not necessary that they get stored on a contiguous location that’s why &p - &q gives an unknown value.



44. **b**

Explanation: The given fun() function can be written as :

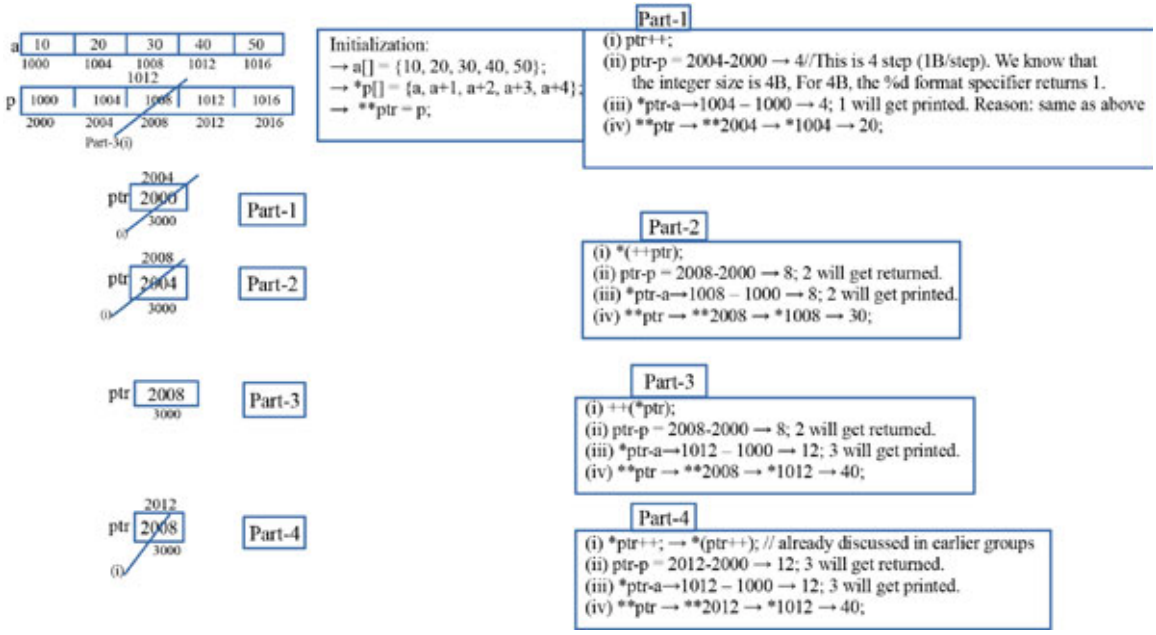


Hence, the character array table become = {"India is wreat", "Hello masters", "Love to gork"}

the printf statement print "India is wreat Love to gork".

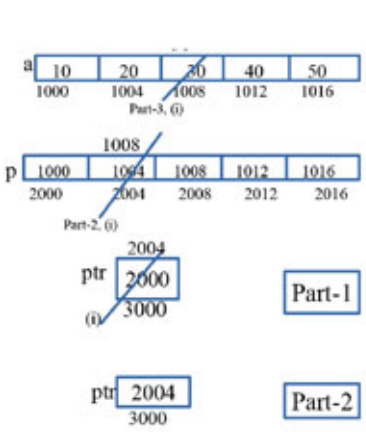
45. **1 1 20; 2 2 30 ; 2 3 40 ; 3 3 40**

Explanation: Refer to explanation of question 36, group 4.



46. **1 1 20 ; 1 2 30 ; 1 2 31**

Explanation: Extension of the above question.



Part-1

(i) **ptr++; →**(ptr++);
(ii) ptr-p = 2004-2000 → 4; 1 will get returned.
(iii) *ptr-a→1004 – 1000 → 4; 1 will get printed.
(iv) **ptr → **2004 → *1004 → 20;

Part-2

(i) *(++(*ptr)) →**(ptr++);
(ii) ptr-p = 2004-2000 → 4; 1 will get returned.
(iii) *ptr-a→1008 – 1000 → 8; 2 will get printed.
(iv) **ptr → **2004 → *1008 → 30;

Part-3

(i) ++**ptr; ++**2004 →++*1008 →value at *1008 get incremented.
(ii) ptr-p = 2004-2000 → 4; 1 will get returned.
(iii) *ptr-a→1008 – 1000 → 8; 2 will get printed.
(iv) **ptr → **2004 → *1008 → 31;

47. **c**

Explanation: Suppose the "Conceptual C questions" string is located at 1000 address onwards.

► The expression at line 1 get expanded as: p+p[4]-p[3] →1000 + ‘e’-’c’ → 1000+101-99 → 1002;

► From 1002 address onwards, %s format specifier prints the remaining string "nceptual C questions".

48. **b**

Explanation:

As ‘\0’ is stored at p[0], nothing gets taken by %s format specifier and nothing gets printed by printf().

49. **d**

Explanation: The *%c* format specifier is used only for characters and %s is used only for addresses. Here, in *printf()* we are giving the address to *%c*. This is invalid, Hence, the compiler throws an error.

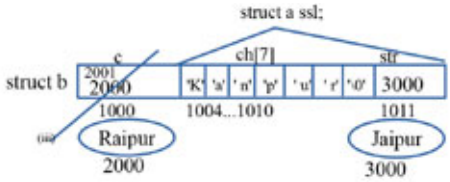
50. **b**

Explanation: A simple recursive program, just be careful with *static*. Explanation is not required.

51. **a**

Explanation:

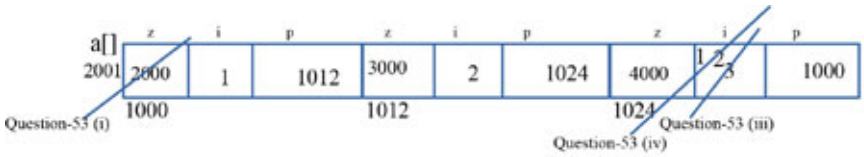
```
Initialization:
 struct a          struct b
 {                 {
char ch[7];        char *c;
char *str;         struct a ssl;
};                 };

struct b s2 = {"Raipur", "Kanpur", "Jaipur"};
```



(i) s2.c → 2000; //%s modifier will print from 2000 onwards; o/p: *Raipur*
(ii) s2.ssl.str → 3000; //%s modifier will print from 2000 onwards; o/p: *Jaipur*
(iii) ++s2.c → ++(s2.c) → 2001 //step size is 1, as it is character// o/p: *aipur*
(iv) ++s2.ssl.str → ++(s2.ssl.str) → 3001; o/p: *aipur*

52. **b**

Explanation: Structure 's1' is of 12B, 'a' is structure array. The indirection operator (->) *x->y* is equivalent to *(*x).y*.




```
Initialization:
struct s1          struct s1 a[] = {{"Nagpur", 1, a+1}, {"Raipur", 2, a+2}, {"Kanpur", 3, a}};
{                  struct s1 *ptr = a;
char *z;
int i;
struct s1 *p;
};
```

(i) *a[0].z → 1000.z →* 2000; //the content at this address get printed by %s. *O/p: Nagpur*
(ii) (*ptr).z → *1000.z →* 2000; *//O/p: Nagpur*
(iii) *ptr->z* is equivalent to *(*ptr).z;* same as above.
(iv) *a[2].p->z → (1024.p)->z → 1000.z →* 2000; *//O/p: Nagpur*

53. **a**

Explanation: Extension of the above question.

(i) ++(ptr->z) → ++(*ptr.z) → ++((*1000).z) → ++(2000) → 2001; //at 2000 location character is there, step size is 1; *O/p: aipur*

(ii) a[(++ptr)->i].z) → a[1012.i].z →a[2].z → 4000; *//O/p: Kanpur*
//++ptr get increased by 12, step size of struct is 12.

(iii) a[--(ptr->p->i)].z → a[--((*ptr.p)->i)].z → a[--(*)] → a[--(1024->i)].z → a[--(3)].z → a[2].z → 4000; *//O/p: Kanpur*

(iv) --a[2].i → --(*(a+2).i) → --(*1024.i) → --(2) → 1; *//O/p: 1*

54. **"etter" ‘j’ 9 "ancestor" "ancestor"**

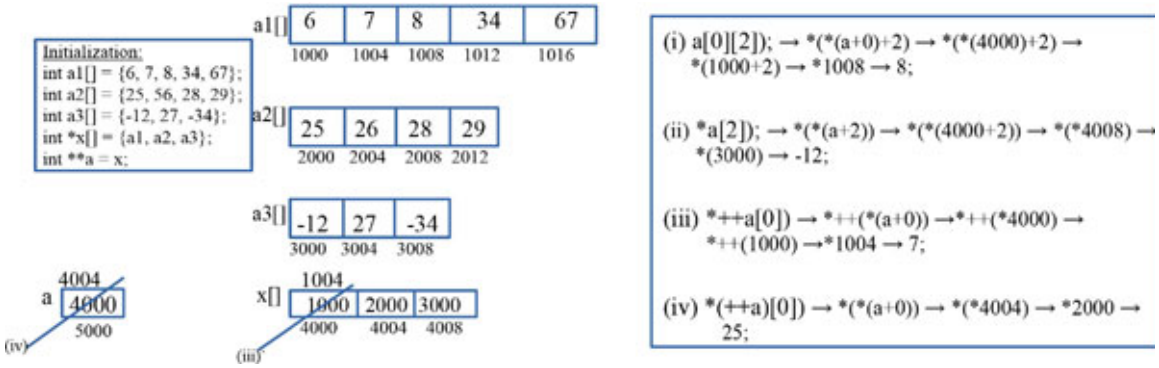
Explanation: A practice question.

55. **c**

Explanation: We already discussed these kinds of questions, explanation is not required.

56. **b**

Explanation: This program demonstrates a 1D array into a 2D array.



57. **See below**

Explanation: Refer of explanation of question 4, 5, and 30. The sizeof(char) is 1 (variable c, d), sizeof(a) is 4, sizeof(b) is 4, sizeof(*e) is 20. Suppose the base address of an array ‘c’ is 1000.

i	c	a	b	d	e
1	1000	1000	1000	1000	1000
2	1000	1004	1004	1001	1020
3	1000	1008	1008	1002	1040
4	1000	1012	1012	1003	1060
5	1000	1016	1016	1004	1080

58. **d**

Explanation: The dynamic memory allocator function *malloc()* and *free()*, we discussed in the explanation of question 5, group 1.

► The function *malloc()* allocates memory in the form of blocks, the block size is the number of bytes required to store a single element of that type. For example, an integer, its block size is 4. Similarly, for character and float, their block sizes are 1 and 4 respectively.

► If we are allocating memory dynamically for 10 integers at line 4 in a given program, the memory that gets allocated is in a contiguous form inside the heap area. The first block contains the header, which has information about the remaining blocks and the memory allocation scheme. The ‘p’ contains the address of 1st block only. It can be assumed as a base address, we can access other blocks by adding an index. It works like an array. The difference between this array and the actual array is that its base address is not constant while with the actual array, whose base address is always constant.

► If we do *free(p)*, p contains the address of the first block. The first block contains the header, which has information about the other blocks also. *free(p)* will make free memory for all the 10 integers.

► Suppose if we do *p++;* , now ‘p’ is pointing to the 2nd block. Suppose we are freeing the memory, free(p), it throws run time error because the second block doesn’t have any information about the remaining blocks. We tried to free blocks of memory without knowing the number of blocks. Hence, it is not possible.

Here, in a given program, At line 2, the ‘i’ is updated, indirectly ‘p’ is updated. Now, ‘p’ started pointing to the 2nd block (element) of an array. In line 7, we are trying to free the dynamic memory. But the pointer ‘p’ is not knowing the memory allocation scheme because it is not pointing to the first block which has information about the memory allocation scheme. Hence, a run time error gets thrown.

59. **Nil**

Explanation:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int *display(int, int);
        int *(*f)(int, int);
        f = display;
        int *x = (*f)(3, 4);

        printf("%d", *x);

}
int *display(int x, int y)
{
        int *i = (int*)malloc(sizeof(int));
        *i = 100+x+y;
        return i;
}
O/p: 107
```

60. **Nil**

    Explanation:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int *display(int, int);
    int *(*f)(int, int);
    f = display;


    void (*funptr)(int, int*(*f)(int, int));
    void show(int, int*);
    funptr = show;
    (*funptr)(3, (*f)(3, 4));
    return 0;
}
void show(int a, int *b)
{
     printf("%d", a+*b);
}
int *display(int x, int y)
{
    int *i = (int*)malloc(sizeof(int));
    *i = 100+x+y;
    return i;
}
```

O/p: 110

Never let your schooling interfere with your education.

Group-6 Questions

Number of 'C' Question: 60

Other Important Interview Questions: 20

1. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a = 10 , b = 20;
 b = printf("%d ",sizeof(sizeof(a *= a+b))); //line 2
 printf("%d %d ", a, b);
 return 0;
}
```

(a) 4 300 20

(b) 4 300 1

(c) 4 10 2

(d) error- compile time

2. **What is the output of the following program.?**

```
#include<stdio.h>
int main()
{
   int x = 10 , y = 10 , z = 10;
   x = y == z && 5;
   printf("%d",x);
   return 0;
}
```

(a) 0

(b) 1

(c) error- compile time

(d) 10

3. **What is the output of the following program?**

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

(a) 10

(b) 5

(c) 100

(d) error- compile time

4. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
int main()
{
 int i = 0;
 char c[] = "hello";
 char d[] = "India";
```

```
 char b[] = i ? c : d ; //line 3
 printf("%s",b);
 return 0;
}
```

(a) India

(b) hello

(c) error- compile time

(d) garbage

5. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a = 0 , b = 20 , 10; //line 1
 b = (110 , 120);
 printf("%d",b);
 return 0;
}
```

(a) 20

(b) 10

(c) 110

(d) 120

(e) error- compile time.

6. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a = 10;
 a++;
 (a++)++; //line 3
 printf("%d",a);
 return 0;
}
```

(a) 13

(b) 12

(c) error- lvalue required

(d) error- run time

7. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
int main()
{
 int i = 0;
 char c[] = "hello";
 char d[] = "India";
 char *b = i ? c : d ; //line 3
 printf("%s",b);
 return 0;
}
```

(a) India

(b) hello

(c) error- compile time

(d) garbage

8. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int b = 20;
 int c = sizeof(scanf("%d",&b));
 printf("%d %d",b,c);
 return 0;
}
```

(a) errror- compile time

(b) 20 4

(c) 30 4

(d) 20 1

9. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
int main()
{
 int i = 1;
 char c[] = "hello";
 char d[] = "India";
 char *b;
 b = i ? c : d ; //line 3
 printf("%s",b);
 return 0;
}
```

(a) India

(b) hello

(c) error- compile time

(d) garbage

10. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a = 1 , b = 0;
 int c = ++(++b)||a++;
 printf("%d %d %d",a,b,c);
 return 0;
}
```

(a) 1 2 1

(b) 1 1 1

(c) 2 2 1

(d) error- lvalue required

11. **Consider the following declarations:**

(i) short i=10;

(ii) static i=10;

(iii) unsigned i=10;

(iv) const i=10;

(a) i, ii, iv, v

(b) ii, iii, v

(c) i, iii, iv, v

(d) i, ii, iii, iv, v

12. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x = 010 , y = 0x10 , z = 10;
 printf("%d %d %d",x,y,z);
 return 0;
}
```

(a) 8 16 10

(b) 10 10 10

(c) error : Invalid use of x

(d) 010 0x10 10

13. **Which of the following does give a compile-time error?**

```
(a) #include<stdio.h>
 #define display(A,B) {\
  printf(A , B);\
 }\
int main(){
 display("%d",10);
 return 0;
}
(b) #include<stdio.h>
 #define display(A,B) do{\
  printf(A , B);\
}while(1 == 2)
int main(){
 display("%d",10);
 return 0;
}
(c) #include<stdio.h>
 #define display(A,B) {\
```

```
   printf(A , B);\
}
int main(){
 display("%d",10)
 return 0;
}
```

(d) None of the above.

14. **What is the output of the following program?**

```
#include<stdio.h>
#define cal(a,b,c,d) a+b*c+d
int main()
{
 int a = 1 , b = 2 , c = 3 , d = 4 , e = 5;
 int f = e * cal(a,b,c,d); //line 2
 printf("%d",f);
 return 0;
}
```

(a) 55

(b) 15

(c) 16

(d) 54

15. **What is the output of the following program?**

```
#include<stdio.h>
#define sqrt(x) x+++x++*++x
int main()
{
 int x = 10;
 int y = sqrt(x); //line 2
 printf("%d %d",x,y);
 return 0 ;
}
```

(a) 13 120

(b) 13 182

(c) 13 153

(d) error- compile time

16. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
#define type_char char*
int main()
{
 type_char a ;
 type_char b;
 char c[10] , d[10];
 strcpy(c , "hello");
 strcpy(d , "India");
 a = c;
 b = d;
 printf("%s %s" , &a[2], &b[2]);
 return 0;
}
```

(a) garbage

(b) llo dia

(c) error- compile time

(d) error- run time

17. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
#define type_char char*
int main()
{
 type_char a ;
 type_char b;
 char c[10] , d[10];
 strcpy(c , "hello");
 strcpy(d , "India");
 a = c;
 b = d;
 printf("%c %c" , a[2],b[2]);
 return 0;
}
```

(a) l d

(b) llo dia

(c) error- compile time

(d) error- run time

18. **What is the output of the following program?**

```
#include<stdio.h>
typedef char *str ;
int main()
{
 str a , b;
 char A[10] , B[10];
 strcpy(A,"hello");
 strcpy(B,"world");
 a = A; b = B;
 printf("%c %c" , a[2] , b[2]);
 return 0 ;
}
```

(a) l r

(b) garbage

(c) error- compile time

(d) error- run time

19. **What is the output of the following program?**

```
#include<stdio.h>
#define cal(a,b) a = b++
int main()
{
 int a = 1 , b = 5;
 cal(a , 2); // line 2
 printf("%d",a);
 printf("%d",b);
 return 0;
}
```

(a) 2 5

(b) 2 3

(c) 3 5

(d) error- compile time

20. **What is the output of the following program ?**

```
#include<stdio.h>
int x = 2;
#if Y == 2 //line 3
 x = 5; //line 4
#else
 x = 6; //line 6
#endif
int main()
{
   printf("%d",x);
   return 0;
}
```

(a) 2

(b) 5

(c) 6

(d) error- compiler time

21. **What is the output of the following program?**

```
#include<stdio.h>
#define a 100
int foo()
{
 printf("%d ",a); //line 5
 return 0;
}
int main()
{
 int b = a; //line 10
 #define a 200 //line 11
 printf("%d ",b);
 printf("%d ",a); //line 13
 foo();
 return 0;
```

```
}
```

(a) 100 200 200

(b) 100 200 100

(c) 200 200 200

(d) error- multiple definitions of macros are not allowed

22. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
void mystrcpy(char p[] , const char q[])
{
   while(*p++ = *q++);
}
int main()
{
 char a[10];
 char b[] = "hello masters";
 mystrcpy(a , b);
 printf("%s",a);
 return 0;
}
```

(a) hello

(b) masters

(c) hello masters

(d) nothing will get printed

23. **What is the output of the following program?**

```
#include<stdio.h>
#define abc main
int main()
{
 int abc = 10;
 static int i = 1;
 while(abc > 5)
 {
  i++;
```

```
 abc--;
}
i++;
if(abc == 5)
{
 abc--;
 i++;
 printf("%d",i);
}
```

(a) 9

(b) 8

(c) 5

(d) error- compile time

24. **What is the output of the following program?**

```
#include<stdio.h>
#define fun1 fun2 //line 2
#if x == 1 //line 3
 #define fun2 fun3 //line 4
#else //line 5
 #define fun2 fun4 //line 6
#endif
int main()
{
 int fun1 = 1 , fun2 = 2 , fun3 = 3 , fun4 = 4; //line 10
 fun1 = fun2 + fun1;
 printf("%d",fun1);
 return 0;
}
```

(a) 8

(b) 6

(c) error- compile time

(d) error- run time

25. **What is the output of the following program?**

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

```
#define fun1 fun2
#define x 1
#if x == 1
 #define fun2 fun3
#else
 #define fun2 fun4
#endif
int main()
{
 int fun1 = 1 , fun2 = 2 , fun3 = 3 , fun4 = 4;
 fun1 = fun2 + fun1;
 printf("%d",fun1);
 return 0;
}
```

(a) 8

(b) 6

(c) error- compile time

(d) error- run time

26. **What is the output of the following program ?**

```
#include<stdio.h>
#define fun1 fun2
#if x == 1
 #define fun2 fun3
#else
 #define fun2 fun4
#endif
int main()
{
 int fun3 = 3 , fun4 = 4;
 fun1 = fun2 + fun1;
 printf("%d",fun1);
 return 0;
}
```

(a) 8

(b) 6

(c) error- compile time

(d) error- run time

27. **What is the output of the following program? Let the address of main() function is 0x1000.**

```
#include<stdio.h>
#define abc main
int main()
{
 int abc = 10;
 printf("the value of abc : %d",abc);
 return 0;
}
```

(a) the value of main : 10

(b) the value of abc : 10

(c) the value of abc : 0x1000

(d) error- compile time

28. **What is the output of the following program?**

```
#include<stdio.h>
#define fun(a , b) (a b ; b = 10)
int main()
{
 fun(int , a);
 printf("%d",a);
 return 0;
}
```

(a) 10

(b) error- compile time

(c) error- run time

(d) Garbage Value

29. **What is the output of the following program?**

```
#include<stdio.h>
#define fun(var) printf("var is %s",#var);
int main()
```

```
{
 int i = 10;
 fun(i);
 return 0;
}
```

(a) i is 10

(b) var is i

(c) var is 10

(d) 10 is 10

30. **What is the output of the following program?**

```
#include<stdio.h>
#define fun max
void max()
{
 printf("Hello");
 return 0;
}
void fun()
{
 printf("World");
 return 0;
}
int main()
{
 fun();
 max();
 return 0;
}
```

(a) World Hello

(b) error in macro definition

(c) error- multiple definition of function

(d) compiler dependent

31. **What is the output of the following program?**

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

```
#define fun(x) x*x
int main()
{
 int x = 2;
 int y = fun(x+2); //line 2
 printf("%d ", y);
 return 0;
}
```

(a) 20

(b) 9

(c) 8

(d) 2

32. **Figure out the differences between two given inclusive directives.**

(i) #include<stdio.h>

(ii) #include"stdio.h"

33. **What is the output of the following program?**

```
#include<stdio.h>
#include<stdlib.h>
void fun();
int main()
{
 typedef int a; //line 1
 a p; //line 2
 p = 10;
 printf("%d",p);
 return 0;
}
void fun()
{
 a q = 1; //line 8
 printf("%d",q);
}
```

(a) 10 1

(b) error: typedef cannot be local to a function

(c) error at line 2

(d) error at line 8

34. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 enum block{one , two = 5 , three , four = 10, five};
 printf("%d %d %d ",one , three, five);
 return 0;
}
```

(a) Compile Time Error

(b) 1 2 3

(c) 0 6 7

(d) 0 6 11

35. **What is the output of the following program?**

```
#include<stdio.h>
enum block{one = 1 , two , three = 1};
int main()
{
 enum block{one = 2, two , three = 2};
 printf("%d %d ",one , two);
 return 0;
}
```

(a) error- compile time

(b) 1 0

(c) 2 0

(d) 2 3

36. **What is the output of the following program?**

```
#include<stdio.h>
enum block{first = 2.5, second , third = 0.5};
int main()
{
 printf("%d ",second);
 return 0;
}
```

(a) 0

(b) 1.5

(c) 3.5

(d) error- compile time

37. **How many times "hello" will get printed?**

```
#include<stdio.h>
enum block{one = 1 , two , three = 3};
int main()
{
 while(two != 2)
 {
  printf("Hello");
  two++; //line 3
 }
 return 0;
}
```

(a) 2

(b) 1

(c) 0

(d) error- compile time

38. **What is the output of the following program?**

```
#include<stdio.h>
enum block{one = 1 , two , three };
int abc(enum*);
int main()
{
 enum block2 {one = 2 , two , three};
 int i = abc(&block2);
 printf("%d",i);
 return 0;
}
int abc(enum *temp)
{
 return (temp->two);
}
```

(a) 0

(b) 3

(c) 2

(d) error- compile time

39. **What is the output of the following program?**

```
#include<stdio.h>
enum block{first = 1 , second , third };
int main()
{
 enum block a = second;
 if(third) //line 2
  a = first;
 printf("%d",strcmp(a,"first"));
 return 0;
}
```

(a) 0

(b) 1

(c) 2

(d) error- run time

40. **What is the output of the following program?**

```
#include<stdio.h>
enum block1{one = 1 , two };
enum block2{two = 3 , three};
int main()
{
 enum block2 b = two;
 printf("%d",b);
 return 0;
}
```

(a) 3

(b) 2

(c) 0

(d) error- compile time

41. **What is the output of the following program?**

```
#include<stdio.h>
enum block{one = "a" , two = "b" };
int main()
{
 printf("%s",one);
 return 0;
}
```

(a) a

(b) 97

(c) error- compile time

(d) garbage

42. **What is the output of the following program?**

```
#include<stdio.h>
enum block{one = ‘a’ , two = ‘b’ };
int main()
{
 printf("%c ",one);
 printf("%d",two);
 return 0;
}
```

(a) a 98

(b) garbage

(c) error- char is not allowed in enum

(d) error- run time

43. **What is the output of the following program?**

```
#include<stdio.h>
enum block{one = 1 , two = ‘b’ , three = "c" };
int main()
{
 printf("%d",one);
 printf("%c",two); //line 2
 printf("%s",three); //line 3
 return 0;
```

```
}
```

(a) 1 b c

(b) error: different types of elements are not allowed in enum

(c) error: enum cannot be char at line 2

(d) error: enum cannot be string at line 3

44. **What is the output of the following program?**

```
#include<stdio.h>
#define i 10
int main()
{
 enum block{int = i , two };
 printf("%d",two);
 return 0;
}
```

(a) 0

(b) 1

(c) 11

(d) error- compile time

45. **What is the difference between the given definitions?**

(i) const int i = 10;

(ii) int const i = 10;

(a) Both are legal but not equivalent.

(b) (i) is illegal but not (ii).

(c) (i) is legal but not (ii).

(d) Both are legal and equivalent.

46. **Which of the following option is true about the given declarations?**

(i) const char *p.

(ii) char const *p.

(iii) char *const p.

(a) (i) and (ii) are equivalent and (iii) is invalid

(b) All are valid and equivalent

(c) Only (i) is valid

(d) All are valid and (i) and (ii) are equivalent but (iii) is not equivalent to (i) and (ii).

47. **What is the output of the following program?**

```
#include<stdio.h>
int fun()
{
 return 20;
}
int main()
{
 const int i = fun();
 const int *p = &i;
 printf("%d" , *p);
 return 0;
}
```

(a) 20

(b) error- run time

(c) 10

(d) error- compile time

48. **What is the output of the following program?**

```
#include<stdio.h>
int fun(){return 1;}
int main()
{
 enum block{int = fun() , two };
 printf("%d",two);
 return 0;
}
```

(a) 1

(b) garbage

(c) 2

(d) error- compile time

49. **What is the output of the following program?**

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

```
enum block {one = 1 , two = 2};
int main()
{
 enum block = two; //line 1
 printf("%d",block);
 return 0;
}
```

(a) 2

(b) 1

(c) error- compile time

(d) garbage

50. **What is the output of the following program?**

```
#include<stdio.h>
enum block1 {sun = 1 , mon };
enum block2 {tue = 1 , thus};
int main()
{
 enum block1 day = thus; //line 1
 printf("%d",day);
 return 0;
}
```

(a) 2

(b) 0

(c) error- compile time

(d) 1

51. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
enum block1 {sun = 1 , mon };
enum block2 {sun = 1 , mon};
int main()
{
 enum block1 a = sun;
 enum block2 b = sun;
```

```
 printf("%d",strcmp(a,b));
 return 0;
}
```

(a) 1

(b) 0

(c) error- compile time

(d) 2

52. **What is the output of the following program?**

```
#include<stdio.h>
enum block {one = 1+2 , two };
int main()
{
 enum block a = two;
 if(two)
  a = two+1;
 printf("%d",a);
 return 0;
}
```

(a) 0

(b) 3

(c) 5

(d) error- compile time

53. **What is the output of the following program?**

```
#include<stdio.h>
int sun = 10;
enum block {sun , mon};
int main()
{
 int i = block.sun + mon;
 printf("%d",i);
 return 0;
}
```

(a) 21

(b) 11

(c) 1

(d) error- compile time

54. **What is the output of the following program?**

```
#include<stdio.h>
int one = 10;
enum block {one , two};
int main()
{
 int i = one + two;
 printf("%d",i);
 return 0;
}
```

(a) 21

(b) 11

(c) 1

(d) error- compile time

55. **What is the output of the following program?**

```
#include<stdio.h>
enum block {first = 1 , second = 2};
int main()
{
 enum block a ;
 printf("%d ",sizeof(a));
 return 0;
}
```

(a) error- compile time

(b) 8

(c) 1

(d) 4

56. **What is the output of the following program?**

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

```
union unit
{
 int marks = 95;
 int roll ;
} std1 , std2;
std2.roll = 512;
std1.marks = 68;
printf("%d %d",std1.marks , std2.marks);
return 0;
}
```

(a) 68 95

(b) 68 Garbage

(c) 68 512

(d) error-compile time

57. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 union unit{
  int marks ;
  int roll ;
 } ;
 union unit a;
 a.marks = 90;
 printf("%d",a.roll);
 return 0;
}
```

(a) Garbage Value

(b) 0

(c) 90

(d) error- compile time

58. **What is the output of the following program?**

```
#include<stdio.h>
union unit{
```

```
 int a ;
 double c;
 float b;
} obj;
int main()
{
 int i = printf("%d",sizeof(obj));
 printf(" %d",i);
 return 0;
}
```

(a) 16 2

(b) 20 2

(c) 8 1

(d) 8 8

59. **What is the output of the following program?**

```
#include<stdio.h>
union unit{
 int a;
 char b[4];
} obj;
int main()
{
 obj.a = 100; //line 1
 printf("%s ",obj.b); //line 2
 strcpy(obj.b , "baby"); //line 3
 printf("%d %s\n",obj.a,obj.b); //line 4
 return 0;
}
```

(a) error- compile time

(b) garbage 100 baby

(c) d 100 baby

(d) d int equivelent of baby baby

60. **What is the output of the following program?**

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

```
union unit{
   int a;
   char b[4];
  } obj;
int main()
{
  obj.a = 100;
  printf("%s ",obj.b);
  obj.b[0] = ‘a’;
  printf("%d ",obj.a);
  obj.b[1] = ‘b’;
  printf("%d %s",obj.a,obj.b);
  return 0;
}
```

(a) garbage 100 100 ab

(b) d 97 9798 ab

(c) d 97 25185 ab

(d) error- compile time

Other important questions for the Interview are as follows:

1. Write a C program to implement your own sizeof().
2. Write an executable program without main().
3. Write a C program to insert a node in a given linked list.
4. Write a C program to delete the complete linked list.
5. Write a C program to delete a node from the linked list. You have given two pointers, the start pointer that is pointing to the first node of the linked list and the ‘ptr’ pointer that is pointing to the node to be deleted. You cannot use the start pointer.
6. Write a C program to print the Nth node from the end of a linked list.
7. Write a C program that counts the number of times a given key is occurring in a linked list.
8. Write a C program to check whether a singly linked list is palindrome or not.

9. Write a C program to insert a key in a sorted linked list, the linked list should be sorted after insertion.
10. Write a C program to remove duplicates from a sorted linked list.
11. Write a C program to remove duplicates from an unsorted linked list.
12. Move the last element to the front of a given linked list.
13. Write a C program in C of merging two sorted linked lists.
14. Write a C program in C of merge sort for linked lists.
15. Write a C program to delete nodes that have a greater value on the right side.
16. Write a C program for Union and Intersection of two linked lists.
17. Write a C program that reverses a linked list without using recursion.
18. Write a C program that reverses a linked list in a group of n nodes. You have a linked list and a number n, you have to reverse first n nodes then second n node, and so on, at the last, if less than n nodes remain then don't reverse them.
19. Write a C program for finding the middle node of a linked list.
20. Write a C program for checking whether a given linked list is NULL terminated or not.

Group-6 Explanations

1. **c**

   Explanation: At line 2, the inner *sizeof()* operator has not evaluated because it is an operand of the outer *sizeof()* operator. The value returned by *sizeof()* operator is of type *unsigned int.* The outer *sizeof()* operator evaluates the size of *unsigned int* that was returned by the inside *sizeof()* which is 4. We already discussed that *sizeof()* does not evaluate any expression, hence the value of 'a' remains 10 and 'b' gets updated by the printf() function and its value becomes 2 because printf() is printing only one character.

2. **b**

   Explanation: The precedence of relational operator is higher than logical operator (than assignment operator). The given expression *x=y==z&&5*,

is evaluated as: *x = ((y == z)&& 5) → x = ((10==10) && 5) → x = (1&&5)) → x = 1*.

3. **b**

   Explanation: Not required. We have already discussed these kinds of questions.

4. **c**

   Explanation: At line 3, the ternary operator is returning a value here. The array cannot be initialized in this way. Array can be initialized using curly braces ({}) element by element or if it is character array it can be initialized as: `char str[] = "hello";` `char str[] = <expression>;` is not allowed. The given statement in line 3 is invalid.

5. **e**

   Explanation: In line 1, we are trying to define a variable whose name is 10. In C, the variable names cannot start with numbers and special symbols except '_'.

6. **c**

   Explanation: Not required. *Hint:* We cannot assign a value to a number, there must be a variable at the left-hand side of the assignment operator for storing a value.

7. **a**

   Explanation: Not required. Refer to question 4.

8. **b**

   Explanation: The return type of *scanf()* library function is *int*. At line 2, the *sizeof()* operator returns the size of the *int* data type, that is 4. The variable 'c' gets assigned the value 4.

9. **b**

   Explanation: Not required. Refer to questions 4 and 7.

10. **d**

    Explanation: Not required. *Hint:* At line 2, we are applying ++ operator on value not on variable, *(++(++b))*.

11. **d**

    Explanation: Default data types of all the storage classes are *int*.

12. **a**

Explanation: At line 1, ‘x’ is assigned to 010, if any number is prefixed with ‘0’ then that number becomes octal, the decimal equivalent to octal 10 is 8, ‘y’ is assigned 0x10, if any number is prefixed with ‘0x’ then that number becomes hexadecimal, the decimal equivalent of hexadecimal 10 is 16, z is assigned 10. At line 2, *printf()* is printing decimal values of ‘x’, ‘y’ and ‘z’. Hence, 8 16 10 get printed.

13. **a**

    Explanation: Option (a) gives an error because the multiple line macro does not have ‘\’ at the last line.

14. **b**

    Explanation: Not required. *Hint:* The statement at line 2, ater macro substitution is: *int f = e*a+b*c+d;*

15. **c**

    Explanation: The line 2, after macro *sqrt()* substitution become: *int y = x+++x++*++x;* this expression get evaluated as: *y = (x++)+(x++)* (++x) → y = (10)+(11)*(13) → y = 153*.

    Hint: The order of precedence of operators for the given expression is: post increment > pre increment > ‘*’ > ‘+’(lowest among these operator). The post increment gets evaluated on encountering any other operator like ‘+’ and ‘*’.

16. **b**

    Explanation: Not required. *Hint:* After macro substitution ‘a’ and ‘b’ become pointer of character type.

17. **a**

    Explanation: Not required.

18. **a**

    Explanation: Not required. Another version of question 17 with *typedef*. *typedef* vs *#define*

    1. *#define* and *typedef* both are used to define aliases for various data types and values.
    2. *typedef* can assign an alias for data types only whereas *#define* can be used to assign aliases for values also e.g. *#define 2 MONDAY*
    3. *#define* is pre-processor directive and substitutes its aliases at pre-processing time only whereas *typedef* is resolved during compilation

time.

4. *#define* is not bounded by scoping rules, its scope is till the *#undef* of the macro or the end of the file or re-define the same macro whereas *typedef* has scoping rules same as variables in C. It is global if it is defined outside all functions and it is local if it is defined within a function.

19. **d**

    Explanation: The line 2, after macro substitution is: *a = 2++* , this expression get evaluated in two steps :(i) a = 2; (ii) 2++, compiler throw an error for the second step.

20. **d**

    Explanation: If a macro is used without defining a value, it automatically takes value 0. In the given question macro Y is not defined before its use at line 3, Y gets defined with value 0, line 3 is not a cause of the error. Error in given code is because of statement at line 4 and line 6, we cannot assign any variable outside the function but initialization is possible outside the function.

21. **b**

    Explanation: The macro substitution is done, from top to bottom, at preprocessing phase of compilation, and at this phase, no scoping rules are considered, the macro ‘a’ will get substituted textually with its definition 100 until it is undefined by *#undef*, redefined by *#define* or the program gets over. The macro ‘a’ at line 5 and line 10 get substituted by its definition 100, at line 11 macro ‘a’ is redefined and its new definition become 200, and ‘a’ get substituted by 200 at line 13. The output of the given program is 100 200 100.

22. **c**

    Explanation: Not required.

23. **b**

    Explanation: In the given program, the macro *abc* will get substituted by its value *main.* It is allowed to have a variable name the same as the function name. Hence, 8 get printed.

24. **c**

    Explanation: At line 3, macro *x* has value 0 (default value of macro because it is not defined), if condition become false and macro *fun2* get

definition *fun4*. At line 2, *fun1* has definition *fun2* and at line 6, *fun2* has definition *fun4*. Finally, *fun1* and *fun2* has definition *fun4*. After macro substitution, the line 10 become: *int fun4 = 1*, *fun4 = 2*, *fun3 = 3*, *fun4 = 4;* Multiple definition of variable *fun4* is available, hence compiler throws an error.

25. **c**

    Explanation: Not required.

26. **a**

    Explanation: Not required.

27. **b**

    Explanation: The given program, at preprocessing phase after the substitution of macro *abc* is written as follows:

```
#include<stdio.h>
int main()
{
    int main = 10;
    printf("the value of abc: %d", main);
    return 0;
}
```

    The local variable can have the name main. Function *printf()* prints value 10.

28. **b**

    Explanation: The given program after macro substitution :

```
#include<stdio.h>
int main()
{
     (int a ; a = 10;)
     printf("%d", a);
     return 0;
}
```

    Here, we are giving the definition of ‘a’ inside parenthesis that is not allowed. Hence, the compiler throws an error.

29. **b**

    Explanation: In the given program, we are using ‘#’ as macro definition, as we discussed, the ‘#’ converts element of macro to string. At line 3,

fun(i) gets substituted with *printf("var is %s","i");*. The *printf()* will print "var is i".

30. **c**

    Explanation: Not required. *Hint:* After macro substitution, two definitions of function max are present.

31. **c**

    Explanation: After macro substitution, line 2 become: *int y = x+2*x+2; → y = 2+2*2+2; → y = 8.*

32. **Nil**

    Explanation: The (i) "include" directive searches the stdio.h header file in the standard *bin* directory of gcc compiler and the (ii) "include" directive searches *stdio.h* header file in the local directory then searches in the standard bin directory.

33. **d**

    Explanation: Discussed in explanation of question 18. *Hint: typedef* is not visible in fun().

34. **d**

    Explanation: In the given program will not give any error. The following facts about *enum* should be known : (i) *enum* can be local or/and global. (ii) If any element of enum is not initialized, it takes value 0 if it is the first element or the value one more than its previous element. (iii) Element of *enum* can have a negative value. The *enum* for a given program is enum block{one = 0, two = 5 , three = 6 , four = 10, five = 11}; The *printf()* function prints 0 6 11.

35. **d**

    Explanation: The *printf()* in *main()* function takes the *enum* elements from local enum block.

36. **d**

    Explanation: The *enum* element must be of type integer only, floating-point numbers are not allowed as the value of *enum* element.

37. **d**

    Explanation: Once enum elements get initialized we cannot change their value. In line 3, we are trying to change the value of element two, compiler throws an error: *l-value required as increment operand.*

38. **d**

    Explanation: We cannot pass an *enum* to the function as its argument, we can only pass its element.

39. **d**

    Explanation: In the given program, till *printf()* all statements are valid. If the condition gets true at line 2, the value of enum variable ‘a’ of type "block" becomes equal to the value of enum "block" element "first" that is 1. Now in *printf()* statement we are comparing an integer value 1 with a string "first" by *strcmp()* function. Hence, the error comes at run time.

40. **d**

    Explanation: Two enums cannot have the same name of the variable in the same scope. In the given program, *enum block1* and *enum block2* have the same scope, and both have element "two" which is not allowed.

41. **c**

    Explanation: Not required. *Hint:* The *enum* element must be of type *int*, in the given code we are initializing elements *one* and *two* with string.

42. **a**

    Explanation: The given program does not report any error, here we are initializing *enum block* elements with character literals. In C, implicitly type casting from character to integer is provided, an element *one* gets value 97 (ASCII value of char ‘a’) and element *two* gets value 98 (ASCII value of ‘b’). Hence, the *printf()* statement prints the character value of element *one*, an integer value of element *two* gets printed.

43. **d**

    Explanation: Not required.

44. **d**

    Answer: d

    Explanation: We cannot have an *enum* element with name as data type (e.g. *int* is here).

45. **d**

    Explanation: Not required.

46. **d**

    Explanation: In (i) and (ii) *const* is present before ‘*p’, the string pointed by p becomes constant, and (iii) *const* is suffixed after ‘*’ and

before ‘p’, here, ‘*p’ is not constant whereas ‘p’ is constant. Hence, (i) and (ii) are equivalent but (iii) is not equivalent to either (i) or (ii).

47. **a**

    Explanation: A constant variable can be initialized by the *return* value of the function.

48. **d**

    Explanation: We can have an enum element with name as data type (e.g. int is here). In the given code, we are initializing the element of an enum with a value returned by a function that is not allowed.

49. **c**

    Explanation: In the given program, at line 1, we are trying to assign an *enum* element to the *enum block* that is not allowed. We can assign an *enum* element to a variable of enum, not with enum itself. This concept is also true for *struct* variables. If we do as: *enum block a = two;* then it is allowed.

50. **a**

    Explanation: The *enum* element represents an integer value. In line 1, we are creating a variable of type *enum block1* and initializing it with the value of element "thus" that is of type *enum block2*. The value of "thus" is 2, so, the element "day" of type *enum bock1* also gets initialized with 2.

51. **c**

    Explanation: We can’t have two *enum* elements, of different *enum*, with the same name, same variables in the same scope.

52. **c**

    Explanation: Not required. *Hint:* The *enum* element *one* initialized with value 3 and *two* with value 4.

53. **d**

    Explanation: The *enum* is another way of defining constant integer variable if we define as *int const a = 10*; it is exactly equivalent to *enum {a = 10};* In the given program, we are providing two definitions of variable *sun*, one definition at line 2 and one definition at line 3. We cannot have multiple definitions of a variable in the same scope. Hence, the compiler throws an error.

54. **d**

Explanation: Not required, another version of question 53.

55. **d**

    Explanation: In a given program, we are creating an *enum block* type variable ‘a’ that can hold a value of type *enum* element, *enum* elements are of type integer that is why variable ‘a’ is also of type integer. The *sizeof()* operator returns the size of an integer data type that is 4.

56. **d**

    Explanation: *Union* is a user-defined data type. In the given program, we are defining a *union unit*, the *unit* has two fields (i) variable marks of type integer (ii) variable roll of type integer but we are providing the value of marks in the definition that is not allowed, we cannot provide value to *union* field without creating an instance of it. The same thing is also true for *struct* user-defined data types. Hence, the compiler throws an error.

57. **c**

    Explanation: This program is showing how union elements can get accessed from other functions. We can access the *union* element by using the dot(.) operator.

58. **c**

    Explanation: The *union* has a property that it does not allocate memory for all elements, it allocates memory equal to the size of the largest element. Here, the largest element is ‘c’ of data type *double*. The memory allocated for the *union unit* is equal to the size of the *double*, that is of 8B. Hence, sizeof(obj) returns 8.

59. **d**

    Explanation: As we discussed earlier, the union does not get allocated memory for all of its elements, it gets memory that can accommodate the biggest element. For the given program, the union *unit* will get a memory of 4 bytes, any modification done by any of the elements of the unit will be reflected on these 4 bytes only. In line 1, these 4 bytes are assigned a number 100. In line 2, these 4 bytes can be printed using ‘%s’ format specifier. It prints string "d", (ASCII value of ‘d’ is 100). In line 3, the string "baby" gets copied into these 4 bytes. In line 5, these 4 bytes are printed using integer format specifier (%d) and string format specifier (%s). The integer equivalent of "baby" gets printed and the string “baby”

will also be printed. If the program is run the “baby” integer value will be 2036490594. This value will be evaluated as follow.

binary of 2036490594::: '01111001 01100010 01100001 01100010' [Little Endian: Lower bytes come at higher address, ‘y’ is coming at MSB, ‘b’ is coming at LSB]

binary(ascii('b'))::: '01100010'

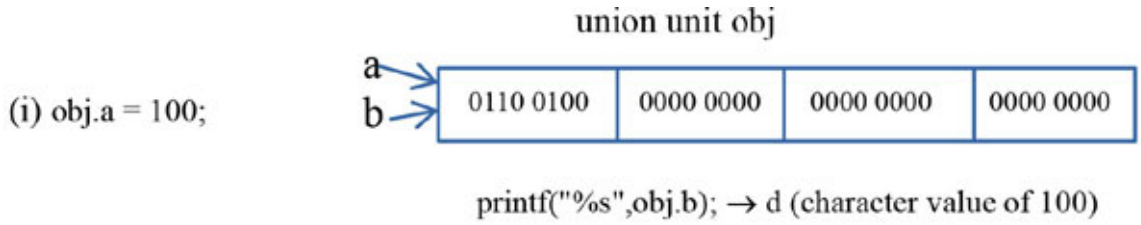
binary(ascii('a'))::: '01100001'

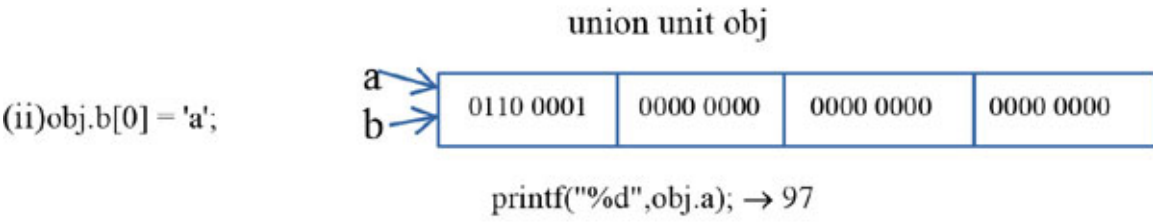
binary(ascii('b'))::: '01100010'

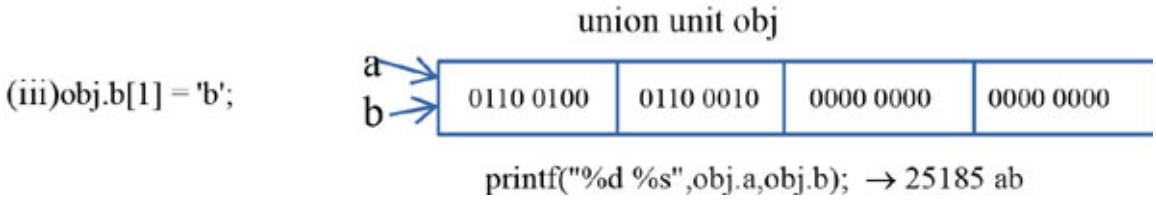
binary(ascii('y'))::: '01111001'

60. **c**

Explanation: As we discussed in previous groups, we are following the *big-endian* mode for storage just for simplifying the concepts and for our convenience.







There is nothing in this world which doesn't surrender to your will power.

Group-7 Questions

Number of 'C' Question: 60

Other Important Interview Questions: 15

1. **What is the output of the following program?**

```
#include<stdio.h>
union unit {
 char a[4];
 int b;
} obj;
int main()
{
 obj.b = 4;
 obj.a[2] = 'A';
 obj.b = obj.b >> 8;
 printf("%s",obj.a);
 return 0;
}
```

(a) 40A

(b) error- compile time

(c) A

(d) Nothing will be printed

2. **What is the output of the following program?**

```
#include<stdio.h>
union unit{
} obj;
struct student{
} s1;
int main()
{
 printf("%d %d",sizeof(obj) , sizeof(s1));
 return 0;
}
```

(a) 0 0

(b) 0 1

(c) 1 0

(d) 1 1

3. **What is the output of the following code?**

```
#include<stdio.h>
int main()
{
 const int i = 10;
 const int j = 20;
 const int *p = &i;
 p = &j;
 printf("%d",*p);
 return 0;
}
```

(a) error- compile time

(b) error- run time

(c) 10

(d) 20

4. **What is the output of the following program?**

```
#include<stdio.h>
union unit {
 int i;
 char str[10]; //line2
} obj;
int main()
{
 obj.i = 0;
 printf("%s",obj.str);
 obj.str = "hello";//line 5
 printf("%s",obj.str);
 return 0;
}
```

(a) 0 hello

(b) hello

(c) error- compile time

(d) error- run time

5. **Suggest modifications in the above code that will allow assignment of string "hello" in enum from main() function?**

6. **What is the output of the following program?**

```
#include<stdio.h>
union {
 int i;
 char j;
} obj;
int main()
{
 obj.j = ‘a’;
 printf("%d",sizeof(obj));
 return 0;
}
```

(a) 1

(b) garbage

(c) 4

(d) compiler dependent

7. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = 64 , j = 5 , res1 , res2;
 res1 = res2 = i;
 for( ; j > 0 ; j--)
 {
  res1 = res1 >> 1;
  res2 = res2 << 1;
 }
 printf("%d %d",res1,res2);
 return 0;
}
```

(a) 2 2048

(b) 4 2048

(c) 1 1024

(d) 1 1048

8. **What is the output of the following program?**

```
#include<stdio.h>
union {
 int i;
 static char j;
} obj;
int main()
{
   obj.i = 65;
   printf("%d",sizeof(obj));
   return 0;
}
```

(a) error- compile time

(b) garbage

(c) 4

(d) 1

9. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 const int i = 10;
 int *p = &i;
 *p = 20;
 printf("%d %d",i,*p);
 return 0;
}
```

(a) 10 20

(b) 10 10

(c) 20 20

(d) error- compile time

10. **What is the output of the following program?**

```
#include<stdio.h>
int i;
void fun()
{
 printf("%d ",i);
 i++;
}
int main()
{
 const int *p = &i;
 fun();
 printf("%d",*p);
 return 0;
}
```

(a) garbage

(b) error- compile time

(c) 0 1

(d) 0 0

11. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = 10 , j = 20;
 int *const p = &i; //line1
 *p = 20; //line2
 printf("%d %d",i,*p);
 return 0;
}
```

(a) 10 20

(b) 20 20

(c) error at line1

(d) error at line2

12. **What is the output of the following program?**

```
#include<stdio.h>
int fun()
{
   static int const i = 10;
   int j = i+1;
   return j;
}
int main()
{
 int i , j = 0 ;
 for(i = 1 ; i <= 5 ; i++)
  j += fun();
 printf("%d", j);
 return 0;
}
```

(a) 55

(b) 0

(c) error- compile time in fun

(d) error- compile time in main

13. **What is the output of the following program?**

```
#include<stdio.h>
#include<stdlib.h>
int* fun(){
 const int *i = calloc(sizeof(int), 1);
 return i;
}
int main(){
 int *p = fun();
 printf("%d ",*p);
 *p = 10;
 printf("%d",*p);
 return 0;
}
```

(a) error in fun()

(b) error in main()

(c) garbage 10

(d) 0 10

14. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int *const i = (int*)malloc(sizeof(int)); //line1
 *i = 10; //line2
 printf("%d",*i);
 return 0;
}
```

(a) error due to line1

(b) error due to line2

(c) 10

(d) garbage

15. **What is the output of the following program?**

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

```
int main()
{
 char const name[20]; //line1
 char *p = name; //line2
 strcpy(p , "john"); //line3
 printf("%s",name);
 return 0;
}
```

(a) error- compile time in line1

(b) error- compile time in line2

(c) error- compile time in line3

(d) john

16. **What is the output of the following program?**

```
#include<stdio.h>
union unit {
 int i;
 char k;
} ;
int main()
{
 union unit obj; obj = {10 , '\0'}; //line 1
 if(obj.i == 0)
  printf("hi");
 else
  printf("bye");
 return 0;
}
```

(a) hi

(b) bye

(c) error- compile time

(d) error- run time

17.

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

```
 char a;
 char arr[10] = {1, 2, 3, 4, 5, 6, 9, 8};
 a = (arr + 1)[5]; //line 3
 printf("%d\n", a);
 return 0;
}
```

(a) 5

(b) 6

(c) 9

(d) none of the above

18. **What is the output of the following program?**

```
#include<stdio.h>
struct student{
 char name[10];
} ;
int main()
{
 struct student s1;
 strcpy(s1.name , "john");
 printf("%s",s1.name);
 return 0;
}
```

(a) error- compile time

(b) error- run time

(c) john

(d) garbage

19. **What is the output of the following program?**

```
#include<stdio.h>
extern int j = 20;
int main()
{
 extern int i; //line 1
 i = 10; //line 2
 printf("%d ",i);
```

```
 printf("%d",j);
 return 0;
}
```

(a) garbage garbage

(b) 10 20

(c) 10 10

(d) error- link time

20. **What is the output of the following program?**

```
#include<stdio.h>
extern int i;
int main()
{
 extern int i; //line 1
 int i = 10; //line 2
 printf("%d",i);
 return 0;
}
```

(a) 10

(b) error due to line 2

(c) garbage

(d) 0

21. **What is the output of the following program?**

```
#include<stdio.h>
int i = 10;
int fun();
int main()
{
 extern int i;
 int p , res = 0;
 for( p = 1 ; p <= 2 ; p++)
  res += fun();
 printf("%d ",res);
 printf("%d",i);
 return 0;
```

```
}
int fun()
{
 extern int i;
 return (++i);
}
```

(a) 23 12

(b) error- in fun()

(c) error- in main()

(d) garbage garbage

22. **What is the output of the following program?**

```
#include<stdio.h>
int i = 10;
int main()
{
 extern int i; //line1
 int i = 1; //line2
 printf("%d",i);
 return 0;
}
```

(a) 1

(b) 10

(c) error due line1

(d) error due line2

23. **What is the output of the following program?**

Two files are given as one.c and two.c.

```
one.c
-----
#include<stdio.h>
extern int i;
int i = 10;
extern int p;
int main()
{
```

```
 extern int i;
 printf("%d %d",i,p);
 return 0;
}
two.c
-----
int p = 20;
```

(a) 10 20

(b) error- compile time in two.c for no header files and/or no main()

(c) error- compile time in one.c for invalid use of extern int i and/or extern int p

(d) error- run time

24. **What is the output of the following program?**

Two files are given as one.c and two.c one.c

```
-----
#include<stdio.h>
extern int i;
int i = 10;
extern int p;
int main()
{
 extern int i;
 printf("%d %d",i,p);
 return 0;
}
two.c
-----
#include<stdio.h>
int p = 20;
int main()
{
 printf("%d",p);
 return 0;
}
```

(a) 10 20 or 20 10 it depends on compiler implementation

(b) error due to multiple instances of header file <stdio.h>

(c) error due to multiple definition of main()

(d) (b) and (c) both are true

25. **Choose the correct option about the given statements?**

(i) extern int i;

(ii) int i = 10;

(iii) extern int i = 10;

(a) All are definitions of i

(b) (i) is a declaration of i and (ii) and (iii) are equivalent

(c) All are declarations of i

(d) (i) and (iii) are the same but not (ii)

26. **Choose the correct option about the given statements?**

(i) int fun();

(ii) extern int fun();

(a) Both are same.

(b) Both are different

(c) (i) is valid, (ii) is invalid

(d) (ii) is valid, (i) is invalid

27. **What is the output of the following program?**

```
#include<stdio.h>
 void fun(char a[])
 {
 int i;
  for (i = 0; i < sizeof(a)/sizeof(a[0]); i++)
   a[i] = (char)i;
 }
 int main()
 {
  int i;
  char a[] = "India";
  fun(a);
  printf("%s",a+1);
  return 0;
 }
```

(a) 0123456

(b) 01234567

(c) India

(d) Nothing will be printed.

28. **What is the output of the following program?**

```
#include<stdio.h>
int j = 1;
int fun()
{
 static int i = 10;
 i += j;
 j++;
 return i;
}
int main()
{
 int i = 10 , p , res = 0;
 for(p = 0 ; p <= 5 ; p++)
  res += fun();
 printf("%d",res);
 return 0;
}
```

(a) 85

(b) 116

(c) 65

(d) error- compile time

29. **What is the output of the following program?**

```
#include<stdio.h>
int i = 20;
int j;
int main()
{
 printf("%d %d",i,j);
 return 0;
}
```

```
i = 10; //line 8
j = 5; //line 9
```

(a) 20 0

(b) 10 5

(c) 20 5

(d) error- compile time

30. **What is the output of the following program?**

Two files are given as one.c and two.c

```
compile it: gcc one.c two.c
execute it: ./a.out
one.c
-----
#include<stdio.h>
static int i = 10;
extern int p;
int main()
{
 printf("%d %d",i,p);
 return 0;
}
two.c
-----
static int p = 100;
```

(a) 10 100

(b) error- compile time due to incomplete two.c

(c) error- compile time because we can't have an global variable as static

(d) error- undeclare symbol p in one.c

31. **What is the output of the following program?**

```
#include<stdio.h>
#include<stdlib.h>
int* abc()
{
 int *i = malloc(sizeof(int));
 *i = 1;
```

```
 return i;
}
int main()
{
 static int *p;
 p = abc();
 printf("%d" ,*p);
 return 0;
}
```

(a) 1

(b) error in abc()

(c) error in main()

(d) garbage Value

32. **What is the output of the following program when n=10?**

```
#include<stdio.h>
int fun(int);
int main()
{
 static auto int n;
 printf("%d",res);
 scanf("%d",&n);
 int res = fun(n);
 return 0;
}
int fun(int n)
{
 if(n == 0)
  return 0;
 else if(n == 1)
  return 1;
 else
  return (fun(n-1) + fun(n-2));
}
```

(a) 34

(b) 21

(c) 55

(d) error- compile time

33. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a = 10;
 register int *p = &a;
 printf("%d",*p); //line 3
 return 0;
}
```

(a) 10

(b) error- register cannot store the memory address

(c) error- invalid ‘*p’ at line 3, we cannot dereference the register contents

(d) compiler dependent

34. **What is the output of the following program?**

Given: sizeof(int) = 4, sizeof(char) = 1, sizeof(double) = 8, sizeof(float) = 4

```
#include<stdio.h>
struct student {
 int roll = 50;
 char name[50] = "john";
 double marks = 90.5;
};
int main()
{
 printf("%u",sizeof(struct student));
 return 0;
}
```

(a) 62

(b) 16

(c) 17

(d) error- compile time

35. **What is the output of the following program?**

```
#include<stdio.h>
struct student {
 int roll ;
 struct student s1;
 }s2;
int main()
{
 s2.roll = 10;
 s2.s1.roll = 20;
 printf("%d",s2.roll);
 return 0;
}
```

(a) 10

(b) 20

(c) error- compile time

(d) garbage

36. **What is the output of the following program?**

```
#include<stdio.h>
struct student {
 int roll ;
 float marks;
 } s1 , s2;
int main()
{
 printf("%d ",sizeof(s1));
 s1.roll = 10;
 s1.marks = 95.5;
 s2 = s1;
 printf("%d %f ",s2.roll , s2.marks);
 return 0;
}
```

(a) error- compile time

(b) 4 garbage garbage

(c) 8 garbage garbage

(d) 8 10 95.5

37. **What is the output of the following program?**

```
#include<stdio.h>
struct student {
 int roll ;
 float marks;
 }s1 , s2;
int main()
{
 s1.roll = 90;
 s1.marks = 95;
 s2.roll = 90;
 s2.marks = 90;
 if(s1 == s2)
  printf("hi");
 else
  printf("bye");
 return 0;
}
```

(a) bye

(b) hi

(c) error- compile time

(d) nothing get printed

38. **What is the output of the following program?**

```
#include<stdio.h>
struct student {
 int roll ;
 char name[0];
 };
int main()
{
 struct student s1 , s2;
 s1.roll = 1;
 strcpy(s1.name , "hist");
 printf("%d %s",s1.roll , s1.name); //line 5
 s2 = s1;
```

```
 printf("%d %s",s2.roll , s2.name); //line 7
 return 0;
}
```

(a) error- compile time

(b) line 5:: Undefined Behavior, line 7:: Undefined Behavior

(c) line 5:: 1 hist, line 7:: Undefined Behavior

(d) Segmentation Fault

39. **What is the output of the following program?**

```
#include<stdio.h>
struct student {
 int roll ;
 char name[0];
};
struct student *s1 = malloc(sizeof(int)); //line 1
int main()
{
 s1->roll = 1;
 printf("%d %s",s1->roll,s1->name); //line 4
 return 0;
}
```

(a) 1 garbage

(b) 1 <nothing get printed>

(c) error at line 4

(d) error at line 1

40. **What is the output of the following program?**

```
#include<stdio.h>
struct student {
 int roll ;
 char name[0];
 };
int main()
{
 struct student *s1 = malloc(sizeof(int)+ 10); // line 1
 s1->roll = 1;
```

```
 strcpy(s1->name , "jack"); // line 2
 printf("%d %s",s1->roll,s1->name);
 return 0;
}
```

(a) error at line 1

(b) error at line 2

(c) 1 garbage

(d) 1 Jack

41. **What is the output of the following program?**

```
#include<stdio.h>
#include<stdlib.h>
struct student {
 int roll ;
 float marks;
 char name[0];
 };
int main()
{
 struct student *s1 = malloc(20);
 struct student *s2 = malloc(100);
 printf("%d %d ",sizeof(s1),sizeof(s2)); //line 3
 printf("%d %d",sizeof(*s1),sizeof(*s2)); //line 4
 return 0;
}
```

(a) 20 100 20 100

(b) 4 4 20 100

(c) 4 4 8 8

(d) error- compile time

42. **What is the output of the following program?**

```
#include<stdio.h>
struct std{
 int roll;
 float marks;
 } s1 , s2;
int main()
```

```
{
 struct std s1 = {1 , 10.5}; //line 1
 s2 = s1; //line 2
 printf("%d %f ",s2.roll , s2.marks);
 if(s1 == s2) //line 4
  s2.marks = s2.marks+1;
 printf("%f",s2.marks);
 return 0;
}
```

(a) 1 10.5 11.5

(b) 1 10.5 10.5

(c) 1 10.5 11.0

(d) error- compile time

43. **What is the output of the following program?**

```
#include<stdio.h>
struct student1 {
   char a ;
   int b;
   char c;
   double d;
 }s1;
struct student2 {
   double a;
   char b ;
 }s2;
struct student3 {
   char a ;
   double b;
   int c;
 }s3[10];
int main()
{
 printf("%d %d %d",sizeof(s1),sizeof(s2),sizeof(s3));
 return 0;
}
```

(a) 14 9 130

(b) 16 10 140

(c) 20 12 160

(d) 18 10 140

44. **What is the output of the following program?**

```
#include<stdio.h>
struct std{
 int sn;
 char name[20];
 } s1;
int main()
{
 struct std s2 = {1 , "john"};
 s1 = s2; //line 2
 printf("%p %p",&s1.name , &s2.name); //line 3
 strcpy(s2.name , "will");
 printf("%s %s",s1.name , s2.name); //line 5
 return 0;
}
```

(a) line 3:: Print both same address, line 5:: john will

(b) line 3:: Print both different address, line 5:: john will

(c) Error at line 2: character array cannot assign using = sign

(d) error- run time

45. **What is the default access specifier of structure in C?**

(a) Private

(b) Public

(c) Protected

(d) Extern

46. **What is the output of the following program?**

```
#include<stdio.h>
struct student{
 unsigned int roll : 4;
 unsigned int marks : 5 ;
 } s1;
int main()
```

```
{
 s1.roll = 14;
 s1.marks = 30;
 printf("%d %d %ld ",s1.roll,s1.marks,sizeof(s1));
 return 0;
}
```

(a) 14 30 4

(b) 14 30 8

(c) error- compile time

(d) error- run time

47. **What is the output of the following program?**

```
#include<stdio.h>
struct student{
 int sn : 10;
 int roll : 10;
 int room : 10 ;
 int id : 10;
 } s1;
int main()
{
 s1.sn = 100;
 s1.roll = 1;
 s1.id = s1.roll; //L1
 printf("%d %ld",s1.id , sizeof(s1));
 return 0;
}
```

(a) error- compile time in structure declaration

(b) error- compile time in assignment in L1

(c) 1 16

(d) 1 8

48. **What is the output of the following program?**

```
#include<stdio.h>
struct student{
 int a : 4;
```

```
 int b : 5 ;
 } s1;
int main()
{
 s1.a = 10;
 s1.b = 17;
 char *p = &s1;
 printf("%d ",*p);
 p++;
 printf("%d",*p);
 return 0;
}
```

(a) 10 17

(b) 41 64

(c) 26 1

(d) 41 17

49. **What is the output of the following program?**

```
#include<stdio.h>
struct student{
 int SN;
 char grade ;
 } s1;
int main()
{
 s1.SN = 1;
 s1.grade = 'A';
 printf("%c %ld ",s1.grade,sizeof(s1));//line 3
 char *p = &s1; //line 4
 *p = 10; //line 5
 p += 4;
 *p = 'B'; //line 7
 printf("%d %c",s1.SN,s1.grade); // line 8
 return 0;
}
```

(a) error- compile time

(b) A 5 10 B

(c) A 8 10 A

(d) A 8 10 B

50. **What is the output of the following program?**

```
struct student {
 int roll;
 struct dob{
  int dd;
  int mm;
  int yyyy;
  };
 }s1;
int main()
{
 s1.roll = 1;
 s1.dob.dd = 6;
 s1.dob.mm = 1;
 s1.dob.yyyy = 2006;
 printf("%d %ld",s1.dob.yyyy,sizeof(s1));
 return 0;
}
```

(a) 2006 16

(b) 2006 12

(c) error in structure declaration

(d) error in main function

51. **What is the output of the following program?**

```
struct student {
 int roll;
 struct dob{
  int dd;
  int mm;
  int yyyy;
  } u ;
 }s1;
int main()
{
```

```
 s1.roll = 1;
 s1.u.dd = 6;
 s1.u.mm = 1;
 s1.u.yyyy = 2006;
 printf("%d %ld",s1.u.yyyy,sizeof(s1));
 return 0;
}
```

(a) 2006 16

(b) 2006 24

(c) error- compile time in structure declaration

(d) error- compile time in main function

52. **What is the output of the following program?**

```
#include<stdio.h>
#include<stdlib.h>
struct addr{
 int h_no[2];
 char streat[40];
 char city[40];
 };
struct one {
 int SN;
 char name[10];
 struct addr loc;
 };
int main()
{
 struct one obj1 = {1 , "Ranbir" , 80 , 81 ,
 "Mandakiny","Mumbai"}; // line 1
 printf("%s %d %s", obj1.name,obj1.loc.h_no[0],
 obj1.loc.city); // line 2
 return 0;
}
```

(a) error in structure one

(b) error in line1

(c) error at line 2

(d) Ranbir 80 Mumbai

53. **What is the output of the following program?**

```
#include<stdio.h>
struct node{
 int key1;
 int key2;
 struct node *next;
 } s1;
struct nodetab{
 int hash;
 struct node* item;
 struct nodetab *next;
 } nt;
int main()
{
 printf("%ld %ld",sizeof(s1),sizeof(nt));
 return 0;
}
```

(a) 12 20

(b) 16 40

(c) error- compile time

(d) 12 12

54. **Consider the above program's structure here also. What is the output of the following program?**

```
#include<stdio.h>
struct node fun(struct node);
int main()
{
 struct node s1 , s2;
 s1.key1 = 10;
 s1.key2 = 20;
 s1.next = NULL;
 s2 = fun(s1);
 printf("%d %d",s1.key1,s1.key2);
 printf("%d %d",s2.key1,s2.key2);
 return 0;
```

```
}
struct node fun(struct node s3)
{
 s3.key1 = s3.key1^s3.key2;
 s3.key2 = s3.key1^s3.key2;
 s3.key1 = s3.key1^s3.key2;
 return s3;
}
```

(a) 20 10 20 10

(b) error- compile time

(c) the fun function produces compiler dependent output

(d) the fun function always swaps key1 and key2 of s3.

55. **What is the output of the following program?**

```
#include<stdio.h>
#include<string.h>
struct student{
 int roll;
 char name[10];
 int getRoll(struct student *s1){
  return s1->roll;
 }
 void show(struct student *s1){
  printf("%d %s",s1->roll,s1->name);
  }
 }s1;
int main()
{
   s1.roll = 10;
   strcpy(s1.name , "john");
   s1.show(&s1);
   int i = s1.getRoll(&s1);
   printf("%d %ld",i,sizeof(s1));
   return 0;
}
```

(a) 10 john 10 16

(b) 10 john 10 14

(c) 10 john 10 24

(d) error- compile time

56. **What is the output of the following code?**

```
#include<stdio.h>
union unit{
 int a;
 int b;
 struct std{
  char name[4];
  } s1;
 } u1;
int main()
{
 u1.a = 67;
 u1.b = 68;
 printf("%ld %s", sizeof(u1), u1.s1.name);
 return 0;
}
```

(a) 8 C

(b) 4 D

(c) error- compile time

(d) 12 D

57. **What is the output of the following program?**

```
#include<stdio.h>
struct student {
 char *name;
 } s1;
int main()
{
 strcpy(s1.name , "john"); //line 1
 printf("%s",s1.name); //line 2
 s1.name[0] = 'T'; //line 3
 printf("%s",s1.name);
 return 0;
}
```

(a) john Tohn

(b) error at line 1

(c) error at line 2

(d) error at line 3

58. **What is the output of the following program?**

```
#include<stdio.h>
#include<stdlib.h>
struct node{
 int key1;
 int key2;
 };
void swap(struct node *start)
{
 struct node *root = malloc(sizeof(struct node));
 root->key1 = 10;
 root->key2 = 20;
 root->key1 = root->key1 ^ root->key2;
 root->key2 = root->key1 ^ root->key2;
 root->key1 = root->key1 ^ root->key2;
 start = root; //line 8
}
int main()
{
   struct node *start = malloc(sizeof(int)*2);
   start = NULL; //line 12
   swap(start);
   printf("%d %d",start->key1,start->key2);
   return 0;
 }
```

(a) 20 10

(b) 10 20

(c) garbageValue

(d) error- run time

59. **What modifications require in the above code such that the swap function correctly swaps the two numbers?**

60. **Determine the output of the following program?**

```
#include<stdio.h>
struct student{
 int roll;
 int class_no;
 } ;
void foo(struct student temp)
{
 static struct student s1 = {44 , 1};
 s1.roll++;
 s1.class_no++;
 temp = s1;
 printf("%d %d ", s1.roll, s1.class_no);
}
int main()
{
 struct student s1 = {50 , 50};
 int i;
 for(i = 1 ; i < 5 ; i++)
  foo(s1);
 printf("%d %d ", s1.roll, s1.class_no);
 return 0;
}
```

Other important questions for the Interview are as follows:

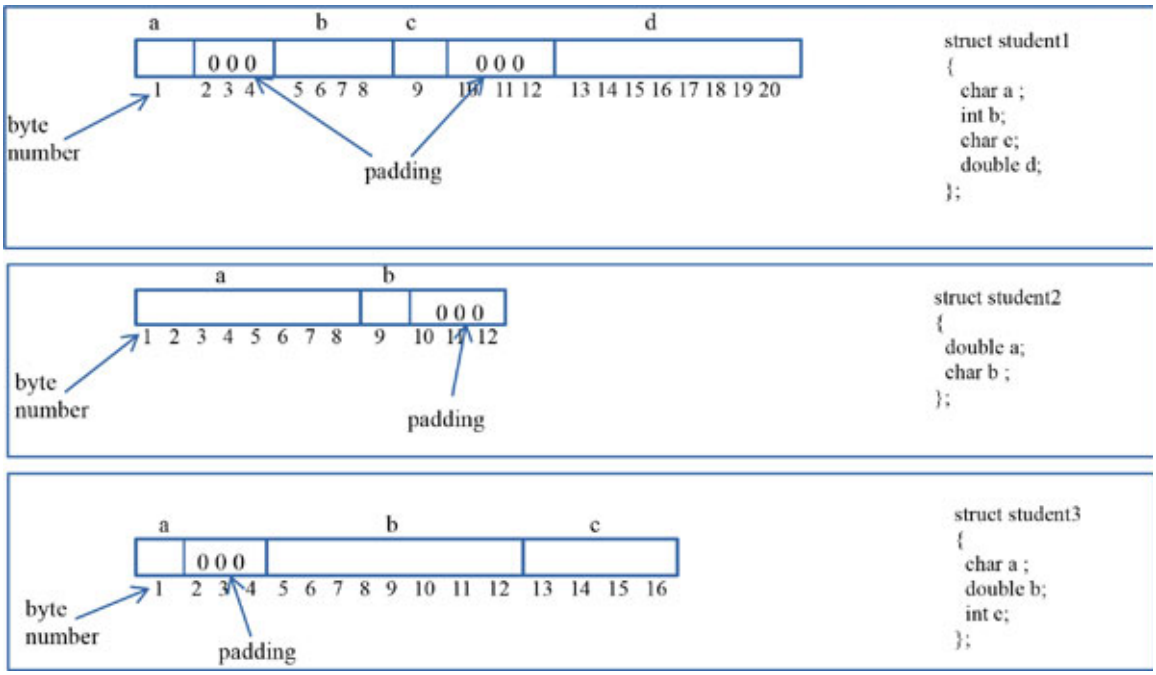
1. Write a C program to copy a tree.
2. Write a C program to create a mirror of a tree.
3. Write a C program for the in-order traversal of a tree without using recursion.
4. Write a C program for pre-order traversal of a tree without using recursion.
5. Write a C program for post-order traversal of a tree without using recursion.
6. Write a C program for finding the diameter of a tree.
7. Write a C program for finding the height of a tree.

8. Write a C program that gives the least common ancestor of two given nodes in a tree.
9. Write a C program for the level order traversal of a tree.
10. Write a C program that finds root to leaf path for each leaf node.
11. Write a C program that finds the sum of all roots to the leaf node.
12. Write a C program for constructing a tree for which in-order and pre-order are given.
13. Write a C program for constructing a tree for which in-order and post-order are given.
14. Write a C program that accepts information of ‘n’ employees and displays it using structure (name, department, age, salary).
15. Write a C program for constructing a full binary tree.

Group-7 Explanations

1. **d**

   Explanation:



   The statement *pritnf("%s",obj.a)* prints the string equivalent of each byte until it gets ‘\0’ (termination character). After step (iii), we have ‘\0’ at

the first byte, *printf()* will not print anything.

2. **a**

   Explanation: The size of an empty *union* and an empty *structure* is zero.

3. **d**

   Explanation: Not required.

4. **c**

   Explanation: Assignment of string to an array (*str[]*) is not allowed using the assignment operator. The given assignment is equivalent to *char str[]; str = "hello";*, this is an invalid assignment of string. Hence, the compiler throws an error.

5. **Nil**

   Explanation: If we use character pointer in union. Correction1: Modify line 1 as: *char *str;*. The given code will work fine. Correction 2: Replace line 5 with few statement as follows: *obj.str[0] = 'h'; obj.str[1] = 'e'; obj.str[2] = 'l'; obj.str[3] = 'l'; obj.str[4] = '0'; obj.str[5] = '\0';* Correction 3: Using standard library function *strcpy();* If we modify line 5 with *strcpy(str, "hello");* It works fine.

6. **c**

   Explanation: The size of the *union* does not depend on the element that gets used in the main() or other function. It is equal to the maximum size of an element present in it.

7. **a**

   Explanation: When 1 left shift is done of a number, in a simple way, the number is getting multiplied by 2 and for 1 right shift the number gets divided by 2.

8. **a**

   Explanation: The static variables are not allowed as *union* elements.

9. **c**

   Explanation: The given program will throw an error in C++, not in C because C++ is more type restrictive than C. We have assigned the address of a constant variable to a non-constant pointer, this concept is called down qualification. We are updating 'i' with the help of its address, this concept is also called "variable hacking".

10. **c**

Explanation: In the given program, variable ‘i’ is a global variable, initialized with 0, and pointer ‘p’ is a constant pointer. Pointer ‘p’ is pointing variable ‘i’, any change in ‘i’ cannot be done by pointer ‘p’ because ‘p’ is constant but ‘i’ is not constant, we can directly change the value of ‘i’.

11. **b**

    Explanation: In the given program, the *const* keyword is used after the ‘*’ operator and before ‘p’, so, ‘p’ is constant, we cannot change ‘p’ but we can change the value of the variable pointed by ‘p’. Hence, the program will not throw any error and 20 20 get printed by *printf()* function.

12. **a**

    Explanation: Not required.

13. **d**

    Explanation: Not required.

14. **c**

    Explanation: Not required.

15. **d**

    Explanation: Not needed. Refer to the explanation of question 9.

16. **b**

    Explanation: At line 1, initialization of *union* object can also be done using *curly braces"{}"* like structures. The union gets assigned only with a first value. The ‘\0’ character is not considered. Now, the *union uni*t contains 10, if the condition gets *false*, "bye" gets printed.

17. **c**

    Explanation: The expression at line 3 is equivalent to a = *(arr + 1 + 5) → *(arr + 6). The printf() will print the value stored at *(arr+6), that is 9

18. **c**

    Explanation: Not required.

19. **d**

    Explanation: In the given program, at line 1, we are declaring the variable ‘i’, not defining. When a variable is declared then the compiler only knows the type of variable but no memory is allocated to a variable and when a variable has been defined the memory for the variable is also

allocated. The compiler delays the work of defining ‘i’ and ‘j’ to link time. The compiler believes, maybe the definition of this variable is present in other programs or any standard library. That is not true for variables ‘i’ and ‘j’, they are just declared, and are not defined anywhere. Hence, the compiler throws a link-time error. In line 2, we are trying to store value 10 in variable ‘i’ and variable ‘i’ that are not bound to memory yet. Hence compiler throws a link-time error.

20. **b**

    Explanation: Not required.

21. **a**

    Explanation: Not required. *Hint:* The *extern* storage class is used to refer to the global variable or variables whose definition is present in other files.

22. **d**

    Explanation: Line 1 will not give any error because the compiler is taking the definition of *extern* variable ‘i’ from the global scope. Compiler throws an error at line 2 due to the redefinition of variable ‘i’.

23. **a**

    Explanation: The given program will not report any error. In *one.c* file the variable ‘p’ is an *extern* variable that can refer to variables of other files also and in *two.c* we are having the definition of ‘p’.

24. **c**

    Explanation: As we discussed all the functions in a program are *extern* by default, after linking the given C files, we will have two instances of the *main()* function and we cannot have two functions with the same name. Hence, the compiler reports an error.

25. **b**

    Explanation: Statement (i) is only the declaration of variable ‘i’, memory for ‘i’ has not been allocated. Statement (ii) is the definition of ‘i’, memory for ‘i’ is allocated and it is initialized with value 10. Statement (iii) is also the definition of ‘i’, we initialize an extern variable then the extern variable gets memory it does not wait till linking time.

26. **a**

    Explanation: Not required.

27. **d**

Explanation: As we can see in function *fun(),* the first index value is coming 0, which is similar to the '\0' null character. Hence, at first index '\0' is there, nothing gets printed.

28. **b**

    Explanation: Not required. It's a practice program.

29. **d**

    Explanation: In the given program, we are performing assignment operation at line 8 and line 9 that is not allowed, any executable statement can execute from a function only.

30. **d**

    Explanation: In file two.c the variable 'p' is defined as static, static variables have scope within a file only, the definition of 'p' that is defined in two.c is not present in file one.c. Hence, the compiler throws an error. If we change the storage class from static to global for variable 'p' in file two.c, given two programs work fine.

31. **a**

    Explanation: Not required.

32. **d**

    Explanation: The given program reports an error. A variable cannot have multiple storage class because each storage class have their own life rules and scoping rules, there is no sense of defining a variable of multiple storage class. All combinations of multiple storage classes are illegal like *static auto n; static extern int i; register static int i;* etc.

33. **a**

    Explanation: The given program will not report any error the reason is that a register pointer can point to the address of a variable. Here, we are defining a variable of type int, this will take memory in the stack and we are storing this stack location in p, this will not create any problem. Value of variable 'a' that is 10 get printed.

34. **d**

    Explanation: The given program report an error. Initialization of structure elements is not allowed.

35. **c**

Explanation: The given program reports an error because we are trying to create an object of type *struct student* before the compilation of *struct* student. At the time of object creation, the compiler must know the size of an object and in the given program we are creating an object inside the definition of its type. It makes it impossible to calculate the size of *struct student.* Defining an object of the self type is not permitted inside the definition of structure.

36. **d**

Explanation: The given program works fine, the assignment operator is compatible with structures. Assignment operator copies all elements of object ‘s1’ to elements of object ‘s2’.

37. **c**

Explanation: In the given program, we are comparing object ‘s1’ with object ‘s2’ with the help of the relational operator "equals to" (==). Relational operators are not compatible with structures. Hence, the compiler throws an error.

38. **c**

Explanation: The given program is an example of *structure hacking*. The structure *student* has a character array ‘name’ of size 0. Any array of size 0 is assumed to be an incomplete type. It is incomplete, but any string assigned to it is considered legal. The first printf() will print: *1 "hist"*. We are trying to assign object ‘s1’ in object ‘s2’ with the help of the assignment operator, as character array ‘name’ is incomplete in ‘s2’, we can’t predict the state of ‘name’ in ‘s2’.

39. **d**

Explanation: Compiler report an error for the given program. In line 1, we are executing a statement outside of *main()*. We cannot write an executable statement in the global scope. All executable statements must be inside a function. Execution of a program starts from a function main() and flows from one function to another, we don’t have any way to execute a statement that is present outside of all functions. Hence, the compiler throws an error.

40. **d**

Explanation: In the given program, the array ‘name’ has size 0B, which means it is an incomplete type. The gcc considers the size of the incomplete array as 0B. This is an example of structure hacking, the array

of 0 lengths must be the last member of a structure. An incomplete array works as a variable-length array, in the given program we are allocating 14 bytes memory for struct student pointer ‘s1’ from these 14 bytes 4 bytes are reserved for *int roll,* and the remaining 10 bytes are used by character array *name*. We are assigning value 1 in *roll* that is the member of ‘s1’ and storing string "jack" in ‘name’ member of ‘s1’. Function *printf()* prints: *1 jack.*

41. **c**

    Explanation: In the given program, at line 3 we are printing the size of struct pointer that is always 4 bytes (Any type of pointer has the same size, in our case, it is 4 because a pointer is depending on the machine configuration). In line 4, we are printing the size of the structure that is pointed by these pointers. In the definition of struct student, we have created a character array of size 0B, a 0B array is considered to be incomplete and for an incomplete variable, gcc compiler gives size 0B. Even though structure pointers ‘s1’ and ‘s2’ are allocated 20B and 100B, the *sizeof()* operator will return the size of struct pointed by pointers that are 8B.

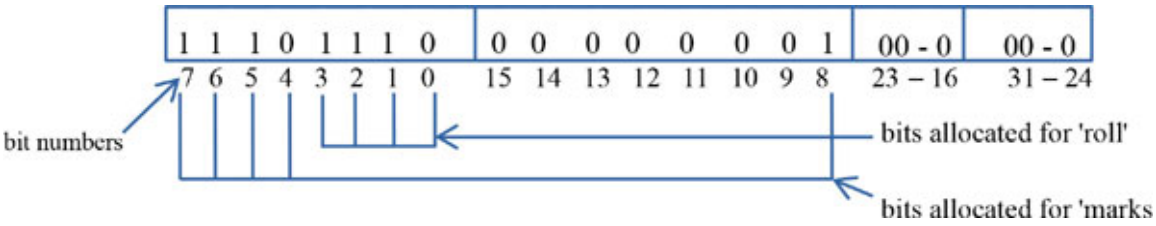
42. **d**

    Explanation: In the given program we are initializing object ‘s1’at line 1, it is allowed. In line 2, object ‘s1’ is copying into object ‘s2’ using the assignment operator, assignment operator is compatible with structures, this will not give an error. In line 4, we are comparing two structure objects with the help of the ‘equal to’ operator, comparing two struct objects by ‘equal to’ operator is not allowed. Hence, the compiler throws an error due to line 4.

43. **c**

    Explanation: This is following the concept of *structure padding*. Considering a machine of 32 bit. All members of a structure are stored on memory locations such that a minimum number of memory access is required. In 32 bit machine, the bus transfers 4 bytes at a time. Considering struct student1, character variable ‘a’ is stored at the first byte of student1, if we place integer variable ‘b’ on bytes 2, 3, 4 and 5 then to read variable ‘b’ machine required two memory access, first access to read bytes from 2 to 4 and second access to read byte 5, it is not feasible that is why memory from byte 2 to 4 will be padded by NULL and variable ‘b’ will be stored from 5 bytes to 8 bytes. Variable ‘c’ will

be stored at the 9th byte and 10, 11,12th bytes will be padded with NULL. Variable ‘d’ will be stored from the 13th to the 20th byte. Hence, the size of object ‘s1’ becomes 20B.



44. **b**

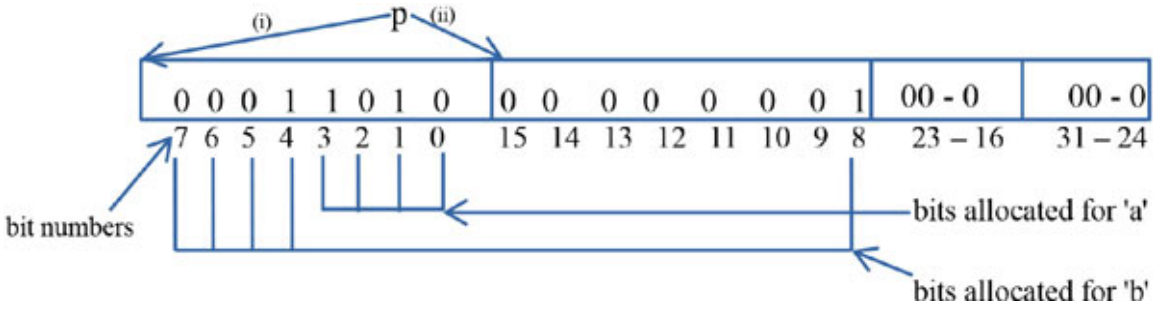
Explanation: In a given program, the assignment operator will copy the string ‘name’ from object ‘s1’ to ‘s2’, it will not throw an error. As we have created two different objects of structure *std*, both get stored at different memory locations, the addresses of ‘name’ in ‘s1’ and ‘s2’ are different.

45. **b**

Explanation: Not required.

46. **a**

Explanation: In the given program, we are using bit fields with structure. Consider a situation, you have to store the roll number and marks of all students in a class, there are 14 students in the class and the maximum marks is 30. Now define a structure student that will take minimum memory. As we know, there are 14 students in the class and if we use integer data type to store their roll number then it will be wasted of memory because integer data type takes 4 bytes and for representing 14 we need only 4 bits likewise we can have 5 bits for structure member ‘marks’.



The member ‘roll’ of structure *student* stored in bit number 0, 1, 2, and 3 and member ‘marks’ stored in bit number 4, 5, 6, 7, and 8. When we print

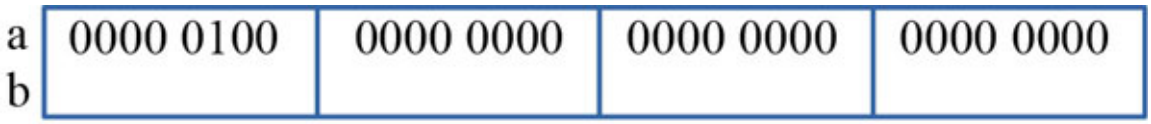
member ‘roll’ in decimal we get the decimal equivalent of binary number 1110 which is 14 and when we print member ‘marks’ in decimal we get the decimal equivalent of binary number 11110 which is 30. The size of the structure student will be 4B, not 2B.

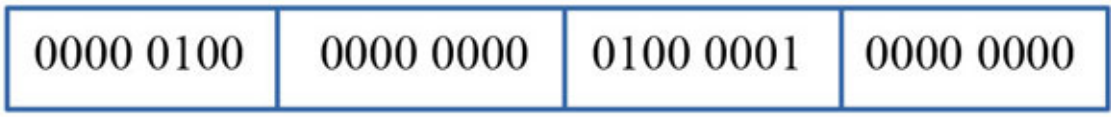
47. **d**

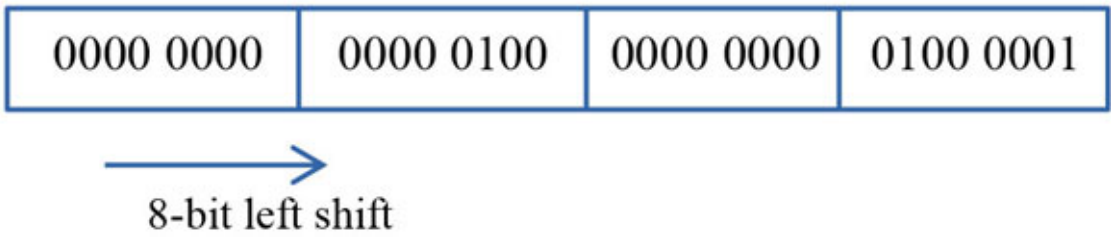
Explanation: In the given program, we are allocating 10 bits for structure member ‘sn’, 10 bits for member ‘roll’, 10 bits for member ‘room’, and 10 bits for member ‘id’, the total number of bits allocated are 40. In structure, if we are using bit fields then the size of the structure will be in multiples of the size of the biggest data type among the members of the structure. Here, the size of the biggest data type of members of structure student is 4B, the size of structure student is in multiples of 4, as we have allocated 40 bits for members of the student so we can have the size of student equals to 4B, so the size of student become 8B.

48. **c**

Explanation: In the given program, we are storing 10 in structure member ‘a’ and 17 in structure member ‘b’. The structure member ‘a’ and ‘b’ get stored like :







(i) char *p = &s1;

printf("%d",*p); → 26; //step size of character pointer is 1B. *p will print decimal equivalent of first byte.

(ii) p++; //step size is 1B, pointer p start to point second byte.

printf("%d",*p); →1.

49. **d**

    Explanation: Given program works fine, as we discussed earlier about structure padding, the size of the structure student is 8B, it is equal to the size of the integer variable, that is 4B, plus the size of a character variable, that is 1B, and three padded bytes. In line 3, ‘A’ and 8 get printed. In line 4, a character pointer ‘p’ is pointing structure object ‘s1’. The step size of the character pointer is 1 byte, at line 5, pointer ‘p’ is storing 10 at the first byte of integer variable ‘SN’ and then get incremented by 4 steps that are four bytes and it starts pointing character variable ‘grade’. In line 7, pointer ‘p’ is storing ‘B’ at the address of ‘grade’. In line 8, 10, and ‘B’ get printed.

50. **d**

    Explanation: In a given program, we are defining the nested structure, we can’t access a member of the inner structure without its instance. We are trying to access the member of the inner structure by structure name that is not allowed. Hence, the compiler reports an error. To avoid this error we should declare an object of inner structure ‘dob’ in the definition of structure ‘student’.

51. **a**

    Explanation: The given program works fine, this is the demonstration of question 50 without error.

52. **d**

    Explanation: The given program works fine, this program is meant to show the way of initialization of structure object.

53. **d**

    Explanation: In the beginning of the book, we have assumed it is 32-bit machine (pointer size will also become 32 bit/4B), size of int is also 4B, output is 12, 12. For 64-bit machine output will differ.

54. **d**

    Explanation: Not required. It’s a simple practice question.

55. **d**

    Explanation: In the given program we are defining functions inside the structure, this is not allowed.

56. **b**

    Explanation: Not required. It's a practice question.

57. **d**

    Explanation: Not required. We have done these types of questions. Hint- the character pointer is pointing to a constant string, we cannot modify the constant string.

58. **d**

    Explanation: The given program is syntactically valid and will not throw any error at compile time, it has a logical problem. In line 12, we are assigning NULL to pointer start, in function calling swap() we are passing NULL (value of start pointer) as an argument. In *swap()* function a struct node *start pointer is defined and initialized with NULL. In line 8, the pointer 'start' is assigned with the value of pointer 'root' but this pointer 'start' is local, any modification done in local pointer will not reflect in pointer 'start' of main(). The pointer 'start' in main() is NULL and in line 14 we are trying to access 'key1' and 'key2' fields of NULL (NULL-key1, NULL->key2), this is invalid. Hence, run time error occurs.

59. **c**

    Explanation: The required modifications are written in bold in the following program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int key1;
 int key2;
};
void swap(struct node **start)
{
 struct node *root = malloc(sizeof(struct node));
 root->key1 = 10;
 root->key2 = 20;
```

```
 root->key1 = root->key1 ^ root->key2;
 root->key2 = root->key1 ^ root->key2;
 root->key1 = root->key1 ^ root->key2;
 *start = root;
}
int main()
{
struct node *start = malloc(sizeof(int)*2);
start = NULL;
swap(&start);
printf("%d %d",start->key1,start->key2);
return 0;
}
```

60. **45 2 46 3 47 4 48 5 50 50**

    Explanation: Not required.

Sample Papers

Sample Paper - 1

No. of Questions: 20 ***Maximum Time: 45 min.***

1. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[2][2][2] = {{10,2,3,4}, {5,6,7,8}};
 int *p,*q;
 p=&a[2][2][2];
 q=**a;
 printf("%d %d",*p,*q);
 return 0;
}
```

(a) garbage 10

(b) garbage garbage

(c) 10 10

(d) error- compile time

2. **What is the output of the following program?**

```
#include<stdio.h>
struct point
{
 int x;
 int y;
};
struct point origin,*pp;
int main()
{
 pp=&origin;
 printf(“origin is (%d, %d)\t”,(*pp).x,(*pp).y);
 printf("origin is (%d, %d)\n",pp->x,pp->y);
 return 0;
```

```
}
```

(a) origin is (0, 0) origin is (0, 0)

(b) origin is (1, 1) origin is (1, 1)

(c) origin is(garbage, garbage)

(d) error- compile time

3. **What is the output of the following program?**

```
#include<stdio.h>
int _l_abc(int i)
{
 return(i++);
}
int main()
{
 int i=_l_abc(10);
 printf("%d\n",-i);
 return 0;
}
```

(a) -10

(b) 9

(c) garbage

(d) error- invalid function name

4. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *p;
 int *q;
 long *r;
 p=q=r=0;
 p++;
 q++;
 r++;
 printf("%u %u %u",p,q,r);
```

```
}
```

(a) 0 0 0

(b) garbage garbage garbage

(c) 1 4 4

(d) error- compile time

5. **What is the output of the following program?**

```
# include <stdio.h>
int main()
{
 int one_d[3]={1,2,3};
 int *ptr;
 ptr=one_d;
 ptr+=3;
 printf("%d",*ptr);
 return 0;
}
```

(a) 3

(b) 2

(c) 1

(d) garbage

6. **What is the output of the following program?**

```
# include<stdio.h>
void aaa()
{
 printf("hi");
}
void bbb()
{
 printf("hello");
}
void ccc()
{
 printf("bye");
```

```
}
int main()
{
 void (*ptr[3])();
 ptr[0]=aaa;
 ptr[1]=bbb;
 ptr[2]=ccc;
 ptr[2]();
 return 0;
}
```

(a) hi

(b) hello

(c) bye

(d) error- compile time

7. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a= 0;int b = 0;char x =’\0’;char y =1;
 if(a,b,x,y)
  printf("hello");
 else
  printf("bye");
 return 0;
}
```

(a) hello

(b) bye

(c) error- compile time

(d) undefined behavior

8. **What is the output of the following program?**

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

```
 printf(" %d ", sizeof( void *));
 printf("%d ", sizeof(int *));
 printf("%d ", sizeof(double *));
 printf("%d \n", sizeof(struct unknown *));
 return 0;
}
```

(a) 0 0 0 0

(b) 0 4 8 unknown

(c) 4 4 4 4

(d) error- compile time

9. **Is the following code legal?**

```
struct a
{
 int x;
 struct a b;
};
```

10. **Determine the output of the following code?**

```
#include<stdio.h>
int main()
{
 int a[3][3]={1,2,3,4,5,6,7,8,9};
 int i,j;
 int *p[]={a,a+1,a+2};
 for(i=0;i<2;i++)
 {
  for(j=0;j<1;j++)
   printf("%d\t%d\t%d\t%d\n",*(*(p+i)+j), *(*(j+p)+i),*(*
   (i+p)+j),*(*(p+j)+i));
 }
 return 0;
}
```

11. **Determine the output of the following program?**

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

```
int main()
{
 char *p="India";
 char a[ ]="India";
 printf("%d %d %d ", sizeof(p), sizeof(*p), strlen(p));
 printf("%d %d", sizeof(a), strlen(a));
 return 0;
}
```

12. **Determine the output of the following code?**

```
#include<stdio.h>
int main()
{
 int i=4,j=7;
 j = j || i++ && printf("YOU CAN");
 printf("%d %d", i, j);
 return 0;
}
```

(a) 4 7

(b) 4 1

(c) 4 8

(d) 5 7

13. **Determine the output of the following code?**

```
#include<stdio.h>
int main()
{
 int i=5,j=10;
 i=i&=j&&10;
 printf("%d %d",i,j);
 return 0;
}
```

(a) 5 10

(b) 1 10

(c) 1 11

(d) 10 10

14. **Differentiate the following declarations.**

(i) const char *a;

(ii) char* const a;

(iii) char const *a;

15. **Determine the output of the following code?**

```
#include<stdio.h>
int main()
{
 static int i=5;
 if(--i)
 {
  main();
  printf("%d ",i);
 }
 return 0;
}
```

16. **Determine the output of the following code.**

```
#include<stdio.h> #include<string.h>
int main()
{
 char a[]="12345\0";
 int i=strlen(a);
 printf("%d\n",++i);
 return 0;
}
```

(a) 5

(b) 4

(c) 7

(d) 6

17. **Determine the output of the following code?**

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

```
{
 int i=i++, j=j++, k=k++;
 printf("%d %d %d", i, j, k);
 return 0;
}
```

18. **Determine the output of the following code?**

```
#include<stdio.h>
int main()
{
 signed char i=125;
 while(i++>=0)
  printf("%d ",i);
 return 0;
}
```

19. **Determine the output of the following program?**

```
#include<stdio.h>
int 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 ", i);
 }
}
```

20. **What is the output for the following program**

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

```
{
 int arr2D[3][3];
 printf("%d\n", ((arr2D==*arr2D)&&(*arr2D == arr2D[0])) );
 return 0;
}
```

(a) 0

(b) 1

(c) error- syntax error

(d) garbage

Solution of Sample Paper-1

1. **a**
2. **a**
3. **a**
4. **c**
5. **d**
6. **c**
7. **a**
8. c
9. **No**
10. **1 1 1 1 4 2 4 2**
11. **4 1 5 6 5**
12. **b**
13. **b**
14. **(i) ‘const’ applies to char * rather than ‘a’ (pointer to a constant char)**

    ***a=’F’ : illegal**

    a="Hi" : legal

    (ii) ‘const’ applies to ‘a’ rather than to the value of a (constant pointer to char)

***a=’F’ : legal**

a="Hi" : illegal

(iii) Same as (i).

15. **0 0 0 0**
16. **d**
17. **garbage garbage garbage**
18. **126 127 -128**
19. **16 21**
20. **b**

Sample Paper - 2

No. of Questions: 20 ***Maximum Time: 45 min.***

1. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x=0,y=2,z;
 if(x=y%2)
  z=2;
 printf("%d %d ",z,x);
 return 0;
}
```

(a) 2 0

(b) garbage 0

(c) 0 0

(d) error- compile time

2. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a[10];
 printf("%d",*a+1-*a+3);
 return 0;
}
```

(a) garbage

(b) 0

(c) 4

(d) error- compile time

3. **What does the following statement mean?**

int (*x)[10];

4. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 const int x=5;
 const int *ptrx;
 ptrx = &x;
 *ptrx = 10;
 printf("%d\n", x);
 return 0;
}
```

(a) 5

(b) 10

(c) error- compile time

(d) garbage

5. **What is the output of the following program?**

```
#include<stdio.h>
#define x 4+1 //line 2
int main()
{
 int i;
 i = x*x*x;
 printf("%d",i);
 return 0;
}
```

(a) 125

(b) 13

(c) 17

(d) error at line 2

6. **Which of these functions are using pointers correctly?**

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

```
int *f1()
{
 int x = 10;
 return &x;
}
int *f2()
{
 int *ptr;
 *ptr = 10;
 return ptr;
}
int *f3()
{
 int *ptr;
 ptr = (int*) malloc(sizeof (*ptr));
 return ptr;
}
```

(a) f3 only

(b) f1 and f3

(c) f1 and f2

(d) f1, f2, and f3

7. What is the output of the following program? Suppose the base address of an array ‘a’ is 1000.

```
#include<stdio.h>
int main()
{
 int a[2][3][4] = {0};
 int *b = a;
 int *c = a+1;
 printf("%u %u", b, c);
 return 0;
}
```

(a) 1000 1048

(b) 1000 1096

(c) 1000 1001

(d) 1000 1004

8. **Which mathematical function is computed by given function foo(x, n)?**

```
int foo(int x, int n)
{
 int val = 1;
 if (n > 0)
 {
  if (n % 2 == 1)
   val *= x;
  val *= foo(x * x, n / 2);
 }
 return val;
}
```

(a) x^n

(b) n^x

(c) n^n

(d) n * x

9. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char a[] = "India";
 char *b = "is great";
 a = "Love";
 b = "yes";
 printf("%s %s", a, b);
 return 0;
}
```

(a) India is great

(b) Love is great

(c) yes is great

(d) error- compile time

10. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *str = "Love p\0eop\0le";
 printf("%ld %ld", sizeof(str), strlen(str));
}
```

(a) 6 6

(b) 7 6

(c) 13 6

(d) 4 6

11. **Determine the output of the following program? Suppose the base address of an array is 1000.**

```
#include<stdio.h>
int main()
{
 int a[2][3][2] = {2, 4, 7, 3, 3, 2, 2, 3, 3, 4};
 printf("%u %u %u %d ", a, *a, **a, ***a);
 printf("%u %u %u %d ", a+1, *a+1, *(*a+1), ***a+1);
 printf("%u %u %u %d\n", *(*(a+2)+3), *(*a[1]+2), *(a+2),
 *(*(a[1]+1)));
 return 0;
}
```

12. **Consider the following declarations**

(i) int a[10][10];

(ii) int *b[10];

Which of the following assignment operation is/are invalid?

(a) a[5][6] = 10;

(b) a[5] = 1000;

(c) b[5][6] = 10;

(d) b[5] = 1000;

13. **What will be the output of the following program?**

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

(a) 10

(b) 5

(c) 100

(d) Compile Time Error.

14. **What will be the output of the following program?**

```
#include<stdio.h>
int main()
{
 int a = 10 , b = sizeof(a++);
 printf("%d %d", a, b);
 return 0;
}
```

(a) error- compile time

(b) 11 4

(c) error- run time

(d) 10 4

15. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 const a = 10; //line1
 const b = 2.5; //line2
 printf("%d %d",a,b);
 return 0;
}
```

(a) error at line 1

(b) error at line 2

(c) 10 2.5

(d) 10 2

16. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 register int i = 10;
 int *p = &i;
 printf("%d",*p);
 return 0;
}
```

(a) 10

(b) error- compile time

(c) error- run time

(d) garbage

17. **What is the output of the following program?**

```
void two()
{
 static int r = 2;
 r++;
 printf("%d ", r);
}
void one()
{
 int r=5;
 two();
 printf("%d ", r);
}
#include<stdio.h>
int r = 10;
int main()
{
```

```
 int r=2;
 two();
 one();
 two();
 printf("%d",r);
 return 0;
}
```

(a) 3 4 5 5 2

(b) 3 3 5 3 2

(c) error- compile time

(d) 3 4 5 3 2

18. **Determine the output of the following program?**

```
#include<stdio.h>
int main()
{
 char a=-10;
 int b=-5 ;
 int c =-3;
 if(c>a)
  {
   printf("hello ");
   if(c<b)
    printf("India ");
   else
    printf("Bharat ");
  }
 else
  printf("hello ");
 if(a<b)
  printf("Bharat ");
 else
  printf("India ");
 return 0;
}
```

19. **Determine the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *str1="abcd";
 char str2[]="abcd";
 printf("%d %d %d",sizeof(str1),
 sizeof(str2),sizeof("abcd"));
 return 0;
}
```

20. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *p;
 printf("%d %d", sizeof(*p), sizeof(p));
 return 0;
}
```

(a) 1 4

(b) 1 1

(c) 4 4

(d) None of these

Solution of Sample Paper-2

1. **b**
2. **c**
3. **It is a definition where ‘x’ is a pointer to an array of (size 10) integers.**
4. **c**
5. **b**
6. **a**
7. **a**

8. **a**
9. **d**
10. **d**
11. **1000 1000 1000 2 1024 1008 1008 3 1072 3 1048 3**
12. **b**
13. **c**
14. **d**
15. **d**
16. **b**
17. **a**
18. **hello Bharat Bharat**
19. **4 5 5**
20. **a**

Sample Paper - 3

No. of Questions: 22 ***Maximum Time: 45 min.***

1. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i=5;
 if(i<1);
  printf("hello");
 return 0;
}
```

(a) nothing get printed

(b) hello

(c) hello infinite time

(d) error- compile time

2. **What is the output of the following program?**

```
#include<stdio.h>
#define num(a) a
int main()
{
 int a=1;
 switch(a)
 {
  case num(2):
   printf("yes ");
  case num(1):
   printf("no ");
   break;
 }
 switch(a)
 {
```

```
  case 1:
   printf("%d ", a);
  case 2:
   printf("%d ", a);
  case 3:
   printf("%d ", a);
  default:
   printf("%d", a);
 }
 return 0;
}
```

(a) yes 1 1 1 1

(b) no 1 1 1 1

(c) yes no 1 1 1 1

(d) error in case label

3. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x=1;
 printf("%d, %d", 74*x, x++);
 return 0;
}
```

(a) 74, 74

(b) 1, 75

(c) 74, 75

(d) compiler dependent

4. **Determine the output of the following program?**

```
#include<stdio.h>
int main()
{
 char str[ ]="India";
 int i=0;
```

```
 while(str[i])
 {
  printf("\n%c %c %c %c ", str[i], *(str+i), *(i+str),
  i[str]);
  i++;
 }
 return 0;
}
```

5. **Determine the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *cptr,c;
 void *vptr,v; // line 2
 c=10; v=0;
 cptr=&c; vptr=&v;
 printf("%c %d",c,v); // line 5
 return 0;
}
```

(a) 10 0

(b) 10 garbage

(c) error at line 2 and line 5, size of 'v' is not known

(d) 10 10

6. **What is the output of the following program?**

```
#include <stdio.h>
int counter(int i)
{
 static int count = 0;
 count = count + i;
 return count;
}
int main()
{
 int i, j;
```

```
 for (i = 0; i <= 5; i++)
 j = counter(i);
 printf("%d\n", j);
 return 0;
}
```

(a) 10

(b) 15

(c) 6

(d) 7

7. **What is the output of the following program?**

```
#include<stdio.h>
#define x 0xabcdef12
int main()
{
 int i=x;
 char *p= &i;
 printf("%x %d \t", p[0], p[0]);
}
```

(a) 12 -17

(b) 12 18

(c) 12 12

(d) 18 18

8. **What is the output of the following program?**

```
#include<stdio.h>
#define MAN(x,y) (x)>(y)?(x):(y)
int main()
{
 int i=10;
 int j=5;
 int k=0;
 k=MAN(i++,++j);
 printf("%d %d %d",i,j,k);
 return 0;
```

```
}
```

(a) 12 6 10

(b) 10 5 0

(c) 12 6 11

(d) 12 7 11

9. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 char *p1="India";
 char *p2;
 p2=(char *)malloc(20);
 while(*p2++=*p1++); //line 4
 printf("%s",p2);
 return 0;
}
```

(a) nothing gets printed

(b) India

(c) India<garbage>

(d) error at line 4

10. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x=5;
 printf("%d %d %d\n",x,x = x<<2,x = x>>2);
 return 0;
}
```

(a) 4 4 4

(b) 5 5 5

(c) compiler dependent

(d) error- compile time

11. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 static int a;
 int x = 5;
 while( x-- != a);
  printf("%d ",x);
 return 0;
}
```

(a) 5 4 3 2 1 0

(b) 5 4 3 2 1 0 -1

(c) -1

(d) 5

12. **What is the output of the following program?**

```
int main()
{
 int p = fun1() + fun2();
 printf("%d",p);
 return 0;
}
#include <stdio.h>
int fun1()
{
 return printf("hello");
}
int fun2() {
 return printf("world");
}
```

(a) helloworld 10

(b) compiler dependent

(c) worldhello 10

(d) error- compile time

13. **What is the output of the following program?**

```
#include <stdio.h>
int main()
{
 printf("%c ", "India"[2]);
 return 0;
}
```

(a) I

(b) n

(c) d

(d) dia

14. **What is the output of the following program?**

```
#include <stdio.h>
int main()
{
     if (sizeof(int) > -1) //line1
    printf("Love");
     else
    printf("India");
     return 0;
}
```

(a) Love

(b) India

(c) nothing gets printed

(d) error at line 1

15. **What is the output of the following program?**

```
#include<stdio.h>
#include<stdlib.h>
int a ;
int main()
{
   int b = 10;
  b = b + a + c;
```

```
 printf("%d",b);
 return 0;
}
int c = 5;
```

(a) garbage

(b) error- undeclared symbol 'c'

(c) 15

(d) 16

16. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 const int a = 5;
 int i = 5;
 int *p = &a;
 while(i)
  switch(*p)//line 5
  {
   case 5
    printf("%d ",a);
    i--;
    *p = i;
   default :
    printf("exit");
    exit(0);
   }
 return 0;
}
```

(a) error at line 5

(b) 54321 exit

(c) 543210 exit

(d) 5 exit

17. **Determine the output of the following program?**

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

```
int main(void)
{
 int a[] = {1, 2} , b[] = {3 , 4} , c[] = { 5 , 6};
 int *p = a , *q = b , *r = c;
 ++*p; *q++ ; *++r;
 printf("%d %d %d ", a[0], a[1], *p);
 printf("%d %d %d ", b[0], b[1], *q);
 printf("%d %d %d", c[0], c[1], *r);
 return 0;
}
```

18. **What is the output of the following program?**

```
#include<stdio.h>
void fun(char * , char *);
int main()
{
 char *s1 = "hello";
 char *s2 = "India";
 fun(s1, s2);
 printf("%s %s", s1, s2);
 return 0;
}
void fun(char *s1, char *s2) {
 char *temp = s1;
 s1 = s2;
 s2 = temp;
}
```

(a) India hello

(b) hello India

(c) hello hello

(d) India India

19. **What is the output of the following program?**

```
#include<stdio.h>
int main(){
 int a=10 , b;
```

```
 b=a++ + ~++a;
 printf("%d %d",a,b);
 return 0;
}
```

(a) error- compile time

(b) 12 -2

(c) 12 -1

(d) 12 -3

20. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 printf("%d",sizeof(1)+sizeof(1.0)+sizeof('1')+sizeof("1.0
 "));
 return 0;
}
```

(a) 12

(b) 15

(c) 16

(d) 20

21. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = 1;
 int j = i+++2*i++;
 printf("%d",j);
 return 0;
}
```

(a) 3

(b) 4

(c) 5

(d) 6

22. **What is the output of the following program?**

```
#include<stdio.h>
enum block {one , two = -10 , three};
int main()
{
 printf("%d %d",one , three);
 return 0;
}
```

(a) error- enum cannot have negative element

(b) 0 1

(c) 1 2

(d) 0 -9

Solution of Sample Paper-3

1. **b**
2. **b**
3. **d**
4. **I I I I**
   n n n n
   d d d d
   i i i i
   a a a a
5. **c**
6. **b**
7. **b**
8. **c**
9. **a**
10. **c**
11. **c**

12. **b**
13. **c**
14. **b**
15. **b**
16. **d**
17. **2 2 2 3 4 4 5 6 6**
18. **b**
19. **d**
20. **d**
21. **c**
22. **d**

Sample Paper - 4

No. of Questions: 23 ***Maximum Time: 45 min.***

1. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int i = 4;
 while(i)
 {
  switch(i)
  {
   case 1: printf("%d ",i--);break;
   {
   case 2:printf("%d ",i--);break;
   i = 4;
   }
   case 3:printf("%d ",i--);break;
   case 4:printf("%d ",i--);
   default : printf("%d",i--);
  }
 }
}
```

(a) while loop will execute infinite times

(b) 4 3 2 1

(c) syntax error

(d) 4 3

2. **What is the output of the following program?**

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

```
{
 char a[];
 char *p= "hello India";
 char *q = &a;
 a = malloc(strlen(p)); //line 1
 strcpy(a,p); //line 2
 printf("%s %s ",p,q);
 return 0;
}
```

(a) error due to line 1

(b) error due to line 2

(c) hello India <some garbage value, because of the absence of '\0' in q>.

(d) hello India

3. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
 int x;
 unsigned int y=-10;
 x = ~0 - 9; //line 1
 if(x == y) //line 2
  printf("hello Bharat");
 else
  printf("hello India");
 return 0;
}
```

(a) hello Bharat

(b) hello India

(c) error due to line 1

(d) error due to line 2

4. **What is the output of the following program?**

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

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

(a) error due to: address of local variable should not be returned

(b) error due to: return type of fun should be static int*

(c) 10

(d) garbage

5. **Which of the following statements are true about the functions?**

(i) Default return type of a function is int.

(ii) Default return type of a function is void.

(iii) Function can return structure.

(iv) Function can return enum and its elements.

(v) Function can not return enum.

(vi) All types of variables, functions, and structures can become static in C.

(a) i, iii, v

(b) i, iv, v

(c) i, iii, iv, vi

(d) ii, iii, iv, vi

6. **What is the output of the following program?**

```
#include<stdio.h>
int main(){
const enum block{a = 1 , b = 5 , Z = 1 } i = 23; //line 1
static q = 12; //line 2
```

```
enum block p;
 p = a;
 p += i + q; //line 3
 printf("%d",i);
 return 0;
}
```

(a) error due to line 1

(b) error due to line 2: invalid data type

(c) error due to line 3: cannot modify const

(d) 23

7. **What is the output of the following program?**

```
#include<stdio.h>
enum block{a = 0 , b = -4 , enum subbloc{i , j = 3} sb,
c}; // line1
int main()
{
 enum block bl.
 static register p ; //line 4
 p = a + bl.sb.i; //line 5
 printf("%d ",p);
 return 0;
}
```

(a) error due to line 1, line 4 and line 5

(b) error due to line 4

(c) 4

(d) 0

8. **What is the output of the following program ?**

```
#include<stdio.h>
union {
 int i;
 char j;
} obj;
int main()
```

```
{
 obj.i = 65;
 printf("%d ",sizeof(obj));
 printf("%c",obj.j); //line 3
 return 0;
}
```

(a) error- compile time

(b) garbage

(c) 4 A

(d) 4 \0

9. **Consider the following two programs:**

Prog 2:

```
#include <stdio.h>
#include <string.h>
#include<stdlib.h>
struct student
{
char name[20];
};
int main()
{
 struct student s1 , s2;
 strcpy(s1.name, "masters");
 s2 = s1;
 s1.name[2] = ‘P’;
 s1.name[3] = ‘Q’;
 printf("%s\n%s",s1.name,s2.name);
 return 0;
}
```

Prog 1:

```
#include <stdio.h>
#include <string.h>
#include<stdlib.h>
 struct student
{
```

```
 char *name;
};
int main()
{
 struct student s1 , s2;
 s1.name = malloc(10);
 strcpy(s1.name, "masters");
 s2 = s1;
 s1.name[2] = ‘P’;
 s1.name[3] = ‘Q’;
 printf("%s\n%s",s1.name,s2.name);
 return 0;
}
```

Compare the output of prog 1 and prog 2.

(a) both are the same

(b) both are different

(c) error: compile time in prog. 1

(d) error: compile time in prog. 2

10. **Consider the following two programs:**

Prog 2:

```
#include<stdio.h>
int main()
{
 char a[] = {‘m’,’a’,’s’,’t’,’e’,’r’,’s’};
 printf("%d",sizeof(a));
 return 0;
}
```

Prog 1:

```
#include<stdio.h>
int main()
{
 char a[] = "masters";
 printf("%d",sizeof(a));
 return 0;
}
```

Compare the output of prog 1 and prog 2.

(a) both give 7

(b) both give 8

(c) Prog 1: 7, Prog 2: 8

(d) Prog 1: 8, Prog 2: 7

11. **What is the output of the following program?**

```
#include "stdio.h"
int main()
{
 char array[100] ;
 int i;
 printf("%d", scanf("%s%d", array , &i));
 return 0;
}
```

Assume value given to scanf are "India" and 10

(a) 9

(b) 2

(c) 10

(d) 1

12. **What will the following program do?**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
 int i;
 char a[]="India";
 char *p="is great";
 char *Temp;
 Temp=a;
 a=malloc(strlen(p) + 1); //line 6
 strcpy(a,p); //line 7
 p = malloc(strlen(Temp) + 1);
 strcpy(p,Temp);
```

```
 printf("%s %s",a,p);
 free(p);
 free(a);
 return 0;
}
```

(a) India is great

(b) is great India

(c) error at line 6

(d) error at line 7

13. **What will be the result of the following program ?**

```
#include<stdio.h>
#include<string.h>
char *gxxx()
{
 static char xxx[1024];
 return xxx;
}
int main()
{
 char *g="India";
 strcpy(gxxx(),g); //line 2
 g = gxxx();
 strcpy(g,"is great");
 printf("%s",gxxx());
 return 0;
}
```

(a) India

(b) India is great

(c) is great

(d) run time error due to line 2

14. **Determine the outout of the following program ?**

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

```
int main()
{
 typedef union {
  int a;
  char b[10];
  float c;
 } Union;
 Union x,y = {100};
 x.a = 50;
 strcpy(x.b,"hello");
 x.c = 21.50;
 printf("%f ", x.c);
 printf("%d %s \n",y.a,y.b);
 return 0;
}
```

15. **What is the output of the following program?**

```
#include<stdio.h>
int main()
{
    printf("%d\t",sizeof(0.5));
     printf("%d\t",sizeof(9000));
     printf("%d",sizeof('X'));
     return 0;
}
```

(a) 4 4 4

(b) 8 4 4

(c) 4 4 1

(d) 8 4 1

16. **Consider the following declarations of enum. Choose the correct one**

(i) enum college {student, teacher, staff}x;

(ii) enum college {student, teacher, staff};

(iii) enum {student, teacher = -5, staff}x;

(iv) enum x {student, teacher, staff};

(a) i, ii, iv

(b) ii, iii

(c) i, iii, iv

(d) i, ii, iii, iv

17. **What is the output of the following program?**

```
#include<stdio.h>
int main(){
     signed x;
    unsigned y;
     x = 10 +- 10u + 10u +- 10;
     y = x;
     if(x==y)
     printf("%d %d",x,y);
     else if(x!=y)
     printf("%u %u",x,y);
     return 0;
}
```

(a) 0 0

(b) 65536 -10

(c) 0 65536

(d) 65536 0

18. **What is the output of the following program?**

```
#include<stdio.h>
int j = 1;
int fun(int n)
{
 static int i = 0;
 i = n;
 i += j;
 j++;
 return i;
}
int main()
{
```

```
 int i = 10 , p , res = 0;
 for(p = 0 ; p <= 5 ; p++)
  res += fun(i);
 printf("%d",res);
 return 0;
}
```

(a) 85

(b) 116

(c) 81

(d) error- compile time

19. **What will be output when you will execute the following c code?**

```
#include<stdio.h>
int main()
{
     int a=-7;
     unsigned int b=-7u; //line 2
     if(a==b)
     printf("India");
     else
     printf("Bharat");
     return 0;
}
```

(a) India

(b) Bharat

(c) error at line 2

(d) error: signed and unsigned int cannot be compared

20. **What is the output of the following program?**

```
#include<stdio.h>
int main(){
     int a=5;
     {
    int b=10;
     ++b;
     ++a;
```

```
    {
     int a=20;
     ++a;
     a=++b;
    }
    ++a;
    ++b;
    printf("%d %d",a,b);
    }
    printf(" %d",a);
    return 0;
}
```

(a) 10 20 10

(b) 6 14 6

(c) 7 13 7

(d) 6 13 6

21. **What is the output of the following program?**

```
#include<stdio.h>
union unit {
 int i;
 char k;
} ;
int main()
{
 union unit obj = {10};
 obj.k = '\0'; //line 1
 if(obj.i == 0)
  printf("hi");
 else
  printf("bye");
 return 0;
}
```

(a) hi

(b) bye

(c) error- compile time

(d) error- run time

22. **What is the output of the following program?**

```
#include<stdio.h>
struct student
{
 unsigned int roll : 6 , batch : 6;
} s1;
int main()
{
 s1.roll = 1;
 s1.batch = 1;
 char *p = &s1;
 printf("%c %u ",*p,sizeof(s1));
 return 0;
}
```

(a) A 4

(b) A 2

(c) garbage 4

(d) garbage 2

23. **What is the output of the following program?**

```
#include<stdio.h>
struct std
{
 int a;
 union unit
 {
  int a, b;
 } u;
};
int main()
{
 struct std s = {1, 2, 3};
 printf("%d %d %d ", s.a, s.u.a, s.u.b);
 return 0;
```

```
}
```

(a) 1 2 3

(b) 1 3 3

(c) 3 2 2

(d) 1 2 2

Solution of Sample Paper-4

1. **b**
2. **a**
3. **a**
4. **c**
5. **c**
6. **d**
7. **a**
8. **c**
9. **a**
10. **d**
11. **b**
12. **c**
13. **c**
14. **21.5 100 ‘d’**
15. **b**
16. **d**
17. **a**
18. **c**
19. **a**
20. **c**
21. **a**
22. **a**

23. **d**

Sample Paper - 5

It contains previous year's GATE questions.

No. of Questions: 29 ***Maximum Time: 1 hr.***

1. **Given a piece of code, define 's' to be** **(GATE CS 2000)**

```
struct node
{
    int i;
    float j;
};
struct node *s[10] ;
```

(a) An array, each element of which is a pointer to a structure of type node

(b) A structure of 2 fields, each field being a pointer to an array of 10 elements

(c) A structure of 3 fields: an integer, a float, and an array of 10 elements

(d) An array, each element of which is a structure of type node.

2. **The number of tokens in the following C statement are**

```
printf("i = %d, &i = %x", i, &i);
```

(GATE 2000)

(a) 3

(b) 26

(c) 10

(d) 21

3. **Consider the following C declaration** **(GATE CS 2000)**

```
struct {
     short s [5]
     union {
     float y;
     long z;
```

```
    }u;
} t;
```

Assume that objects of the type short, float, and long occupy 2 bytes, 4 bytes, and 8 bytes, respectively. The memory requirement for variable t, ignoring alignment considerations are:

(a) 22 bytes

(b) 14 bytes

(c) 18 bytes

(d) 10 bytes

4. **The value of j at the end of the execution of the following C program.** **(GATE CS 2000)**

```
#include<stdio.h>
int incr (int i)
{
    static int count = 0;
    count = count + i;
    return (count);
}
int main ()
{
    int i,j;
    for (i = 0; i <=4; i++)
     j = incr(i);
 return 0;
}
```

(a) 10

(b) 4

(c) 6

(d) 7

5. **What will be the most appropriate matching for the following pairs** **(GATE CS 2000)**

X: m=malloc(5); m= NULL; 1: using dangling pointers

Y: free(n); n->value=5; 2: using uninitialized pointers

Z: char *p; *p = ’a’; 3. lost memory is:

(a) X—1 Y—3 Z-2

(b) X—2 Y—1 Z-3

(c) X —3 Y—2 Z-1

(d) X—3 Y—1 Z-2

6. **Consider the following three C functions :** **(GATE 2001)**

```
[PI] int * g (void)
{
   int x = 10;
   return (&x);
}
[P2] int * g (void)
{
   int * px;
   *px = 10;
   return px;
}
[P3] int *g (void)
{
 int *px;
   px = (int *) malloc (sizeof(int));
   *px = 10;
   return px;
}
```

Which of the above three functions are likely to cause problems with pointers?

(a) Only P3

(b) Only P1 and P3

(c) Only P1 and P2

(d) P1, P2, and P3

7. **Consider the following declaration of a ‘two-dimensional array in C:** **(GATE CS 2002)**

```
   char a[100][100];
```

Assuming that the main memory is byte-addressable and that the array is stored starting from memory address 0, the address of a[40][50] is

a) 4040

b) 4050

c) 5040

d) 5050

8. **The C language is.** **(GATE CS 2002)**

   a) A context-free language

   b) A context-sensitive language

   c) A regular language

   d) Parsable fully only by a Turing machine

9. **Assume the following C variable declaration** **(GATE CS 2003)**

```
int *A [10], B[10][10];
```

   Of the following expressions

   I A[2]

   II A[2][3]

   III B[1]

   IV B[2][3]

   which will not give compile-time errors if used as left-hand sides of assignment statements in a C program?

   a) I, II, and IV only

   b) II, III, and IV only

   c) II and IV only

   d) IV only

10. **Consider the C program shown below.** **(GATE CS 2003)**

```
# include <stdio.h>
# define print(x) printf ("%d ", x)
void P(int *y)
{
    int x = *y+2;
    Q(x);
```

```
   *y = x-1;
    print(x);
}
main(void)
{
   x=5;
   P(&x);
   print(x);
   getchar();
}
int x;
void Q(int z)
{
   z += x;
   print(z);
}
```

The output of this program is

a) 12 7 6

b) 22 12 11

c) 14 6 6

d) 7 6 6

11. **Consider the following C program segment:** **(GATE CS 2004)**

```
char p[20];
char *s = "string";
int length = strlen(s);
int i;
for (i = 0; i < length; i++)
     p[i] = s[length – i];
printf("%s",p);
```

The output of the program is

a) gnirts

b) gnirt

c) string

d) no output is printed

12. **Consider the following C function** **(GATE CS 2004)**

```
void swap (int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

In order to exchange the values of two variables x and y

a) call swap (x, y)

b) call swap (&x, &y)

c) swap (x,y) cannot be used as it does not return any value

d) swap (x,y) cannot be used as the parameters are passed by value

13. **Consider the following C function:** **(GATE CS 2004)**

```
int f(int n)
{
    static int i = 1;
    if (n >= 5)
     return n;
    n = n+i;
    i++;
    return f(n);
}
```

The value returned by f(1) is

a) 5

b) 6

c) 7

d) 8

14. **Consider the following program fragment for reversing the digits in a given integer to obtain a new integer. Let n = D1D2…Dm** **(GATE CS 2004)**

```
int n, rev;
rev = 0;
while (n > 0)
```

```
{
    rev = rev*10 + n%10;
    n = n/10;
}
```

The loop invariant condition at the end of the ith iteration is:

a) n = D1D2….Dm-i and rev = DmDm-1…Dm-i+1

b) n = Dm-i+1…Dm-1Dm and rev = Dm-1….D2D1

c) n \not = rev

d) n = D1D2….Dm and rev = DmDm-1…D2D1

15. **Consider the following C function:** **(GATE CS 2004)**

```
main()
{
    int x, y, m, n;
    scanf ("%d %d", &x, &y);
    /* x > 0 and y > 0 */
    m = x; n = y;
    while (m != n)
    {
     if(m>n)
     m = m - n;
     else
     n = n - m;
    }
    printf("%d", n);
}
```

The program computes

a) x + y using repeated subtraction

b) x mod y using repeated subtraction

c) the greatest common divisor of x and y

d) the least common multiple of x and y

16. **Consider the following C-program:** **(GATE CS 2005)**

```
int main ()
{
```

```
  int a = 2048, sum = 0;
  foo (a, sum);
  printf ("%d\n", sum);
  getchar();
}
void foo(int n, int sum)
{
  int k = 0, j = 0;
  if (n == 0) return;
    k = n % 10;
  j = n / 10;
  sum = sum + k;
  foo (j, sum);
  printf ("%d,", k);
}
```

What does the given program print?

(a) 8, 4, 0, 2, 14

(b) 8, 4, 0, 2, 0

(c) 2, 0, 4, 8, 14

(d) 2, 0, 4, 8, 0

17. **What is printed by the following C program?** **(GATE 2008)**

```
int f(int x, int *py, int **ppz)
{
  int y, z;
  **ppz += 1;
   z = **ppz;
  *py += 2;
   y = *py;
   x += 3;
   return x + y + z;
}
void main()
{
   int c, *b, **a;
   c = 4;
```

```
    b = &c;
    a = &b;
    printf( "%d", f(c,b,a));
    getchar();
}
```

(a) 18

(b) 19

(c) 21

(d) 22

18. **What does the following program print?** **(GATE 2010)**

```
#include<stdio.h>
void f(int *p, int *q)
{
  p = q;
 *p = 2;
}
int i = 0, j = 1;
int main()
{
  f(&i, &j);
  printf("%d %d \n", i, j);
  getchar();
  return 0;
}
```

(a) 2 2

(b) 2 1

(c) 0 1

(d) 0 2

19. **What is the value printed by the following C program?**

(GATE 2010)

```
#include<stdio.h>
int f(int *a, int n)
{
  if(n <= 0) return 0;
```

```
  else if(*a % 2 == 0) return *a + f(a+1, n-1);
  else return *a - f(a+1, n-1);
}
int main()
{
  int a[] = {12, 7, 13, 4, 11, 6};
  printf("%d", f(a, 6));
  getchar();
  return 0;
}
```

(a) -9

(b) 5

(c) 15

(d) 19

20. **What does the following fragment of C-program print?** **(GATE2011)**

```
char c[] = "GATE2011";
char *p =c;
printf("%s", p + p[3] - p[1]) ;
```

(a) GATE2011

(b) E2011

(c) 2011

(d) 011

21. **Consider the following recursive C function that takes two arguments** **(GATE 2011)**

```
unsigned int foo(unsigned int n, unsigned int r) {
  if (n > 0) return (n%r + foo (n/r, r ));
  else return 0;
}
```

What is the return value of the function foo when it is called foo(513, 2)?

(a) 9

(b) 8

(c) 5

(d) 2

22. **What will be the output of the following C program segment?** **(GATE 2012)**

```
char inchar = ‘A’;
switch (inchar)
{
 case ‘A’ :
 printf ("choice A \n") ;
 case ‘B’ :
 printf ("choice B ") ;
 case ‘C’ :
 case ‘D’ :
 case ‘E’ :
 default:
 printf ("No Choice") ;
 }
```

(a) No choice

(b) Choice A

(c) Choice A Choice B No choice

(d) Program gives no output as it is erroneous

23. **Consider the following C program, what is the output generated?** **(GATE 2014)**

```
int a, b, c = 0;
void prtFun (void);
int main ()
{
     static int a = 1; /* line 1 */
     prtFun();
     a += 1;
     prtFun();
     printf ( "\t %d %d " , a, b) ;
}
void prtFun (void)
{
     static int a = 2; /* line 2 */
     int b = 1;
```

```
    a += ++b;
    printf ( "\t %d %d " , a, b) ;
}
```

(a) 3 1 4 1 4 2

(b) 4 2 6 1 6 1

(c) 4 2 6 2 2 0

(d) 3 1 5 2 5 2

24. **What will be the output of the following program?**

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

(a) 10

(b) 5

(c) 100

(d) Compile Time Error.

25. **Consider the following program in C language:** **(GATE 2014)**

```
#include <stdio.h>
main()
{
    int i;
    int *pi = &i;
    scanf("%d", pi);
    printf("%d\n", i+5);
}
```

Which one of the following statements is TRUE?

(a) Compilation fails.

(b) Execution results in a run-time error.

(c) On execution, the value printed is 5 more than the address of variable i.

(d) On execution, the value printed is 5 more than the integer value entered.

26. **Consider the function func shown below:** **(GATE 2014)**

```
int func(int num)
{
     int count = 0;
     while (num)
     {
     count++;
    num >>= 1;
     }
     return (count);
}
```

The value returned by func(435) is _____.

27. **Consider the C function given below.** **(GATE 2014)**

```
int f(int j)
{
  static int i = 50;
  int k;
  if (i == j)
  {
    printf("something");
    k = f(i);
    return 0;
  }
  else return 0;
}
```

Which one of the following is TRUE?

(a) The function returns 0 for all values of j.

(b) The function prints the string something for all values of j.

(c) The function returns 0 when j = 50.

(d) The function will exhaust the runtime stack or run into an infinite loop when j = 50

28. **Consider the C function given below. Assume that the array listA contains n (> 0) elements, sorted in ascending order.** **(GATE 2014)**

```
int ProcessArray(int *listA, int x, int n)
{
     int i, j, k;
     i = 0;
     j = n-1;
     do{
     k = (i+j)/2;
     if (x <= listA[k])
      j = k-1;
     if (listA[k] <= x)
      i = k+1;
     } while (i <= j);
     if (listA[k] == x)
     return(k);
     else
     return -1;
}
```

Which one of the following statements about the function ProcessArray is CORRECT?

(a) It will run into an infinite loop when x is not in listA.

(b) It is an implementation of binary search.

(c) It will always find the maximum element in listA.

(d) It will return -1 even when x is present in listA.

29. **Consider the following function** **(GATE 2014)**

```
double f(double x)
{
    if (abs(x*x - 3) < 0.01) return x;
    else return f(x/2 + 1.5/x);
}
```

Give a value q (to 2 decimals) such that f(q) will return q:_____.

Solution of Sample Paper-5

1. **a**
2. **c**
3. **c**
4. **a**
5. **d**
6. **c**
7. **b**
8. **a**
9. **a**
10. **a**
11. **d**
12. **d**
13. **c**
14. **a**
15. **c**
16. **d**
17. **b**
18. **d**
19. **c**
20. **c**
21. **d**
22. **c**
23. **c**
24. **c**
25. **d**
26. **9**
27. **d**
28. **b**
29. **1.732**

References

[1] Randal E. Bryant, David R. O'Hallaron book "Computer Systems- A programmer's Perspective", (Beta Draft) 2001.

[2] R. Nageswara Rao "The Ultimate C: Concepts, Programs, and Interview Questions", Careermonk Publication, Founder: Narsimha Karumanchi, Published: June 2012.

[3] http://www.geeksforgeeks.org

Index

Symbols

A

B

C

D

E

F

G

H

K

L

M

N

O

R

S

T

U

V

W