C Programs with Solutions

C
Programs with Solutions

By

S. ANANDAMURUGAN

M.E., (Ph.D)., MISTE., MACEEE.,

Senior Lecturer, Department of Computer Science & Engineering,

Kongu Engineering College, Perundurai

Tamil Nadu

UNIVERSITY SCIENCE PRESS

(An Imprint of Laxmi Publications Pvt. Ltd.)

BANGALORE • CHENNAI • COCHIN • GUWAHATI • HYDERABAD

JALANDHAR • KOLKATA • LUCKNOW • MUMBAI • PATNA

RANCHI • NEW DELHI

Published by :

UNIVERSITY SCIENCE PRESS

(*An Imprint of Laxmi Publications Pvt. Ltd.*)
113, Golden House, Daryaganj,
New Delhi-110002

Phone : 011-43 53 25 00
Fax : 011-43 53 25 28

www.laxmipublications.com
info@laxmipublications.com

Price: ₹ **180.00** *Only.* *First Edition*: 2011

OFFICES

Bangalore	080-26 75 69 30	**Chennai**	044-24 34 47 26
Cochin	0484-237 70 04, 405 13 03	**Guwahati**	0361-251 36 69, 251 38 81
Hyderabad	040-24 65 23 33	**Jalandhar**	0181-222 12 72
Kolkata	033-22 27 43 84	**Lucknow**	0522-220 99 16
Mumbai	022-24 91 54 15, 24 92 78 69	**Patna**	0612-230 00 97
Ranchi	0651-221 47 64		

UCP–9612–180–C PROGRAM WITH SOLUTION–ANA **C—**

Typeset at : Monu Printographics, Delhi *Printed at* : Ajit Printers, Delhi

Dedicated to
My Son
Master A. Shrikarthick

Contents

Preface

This book gives a rich collection of C programs. These programs that support the theoretical concepts are given in a large number to help students understand the concepts better. This book will be useful for students of BE, MCA, BCA, MSc, and BSc, which have C programming language as a part of the course.

The first chapter deals with the fundamental concepts of C language. The second chapter focuses on introduction C programming. The third chapter provides with detailed program on next level to the basic C program. Fourth chapter focuses on C debugging. The fifth chapter deals with the simple C questions and Answers. Sixth chapter deals with the short questions and answers.

The main aim of this book is to give maximum guidance to the students, faculty and research scholars. Suggestions for improvement will be appreciated and incorporated.

—Author

Chapter 1 C CONCEPTS

1.0 OVERVIEW OF C PROGRAMMING

C language is one of the most popular computer languages today because it is a structured, high level, machine independent language. It allows software developers to develop programs without worrying about the hardware platforms where they will be implemented. C is called a high level, compiler language. The aim of any high level computer language is to provide an easy and natural way of giving a programme of instructions to a computer.

C is one of a large number of high level languages which can be used for general purpose programming, *i.e.*, anything from writing small programs for personal amusement to writing complex applications. It is unusual in several ways. Before C, high level languages were criticized by machine code programmers because they shielded the user from the working details of the computer. The C language has been equipped with features that allow programs to be organized in an easy and logical way. This is vitally important for writing lengthy programs because complex problems are only manageable with a clear organization and program structure.

C allows meaningful variable names and meaningful function names to be used in programs without any loss of efficiency and it gives a complete freedom of style, it has a set of very flexible loop constructions and neat ways of making decisions. These provide an excellent basis for controlling the flow of programs. Another feature of C is the way it can express ideas concisely. The richness of a language shapes what it can talk about. C gives us the apparatus to build neat and compact programs. C tries to make the best of a computer by linking as closely as possible to the local environment.

The increasing popularity of C is probably due to its many desirable qualities. It is a robust language whose rich set of built-in functions and operators can be used to write any complex program. The C compiler combines the capabilities of an assembly language with the features of a high-level language and therefore it is well suited for writing both system software and business packages. Programs written in C are efficient and fast. This is due to its variety of data types and powerful operators. C is highly portable. This means that C programs written for one computer can be run on another with little or no modification. Another feature of C is its ability to extend itself.

1.1 INTRODUCTION

C is a remarkable language. Designed originally by **Dennis Ritchie,** working at **AT&T Bell** Laboratories in New Jersey, it has increased in use until now it may well be one of the most widely-written computer languages in the world. C is a structured language. It allows variety of programs in small modules. It is easy for debugging, testing, and maintenance if a language is a structured one.

1.2 STRUCTURE OF A C PROGRAM

```
Include header file section
Global declaration section
main()
{
Declaration part
Executable part
}
User-defined functions
{
Statements
}
```

Include Header File Section

C program depends upon some header files for function definition that are used in program. Each header file by default is extended with .h. The header file should be included using # include directive as given here.

Global Declaration

This section declares some variables that are used in more than one function. These variables are known as global variables. This section must be declared outside of all the functions.

Function Main

Every program written in C language must contain main () function. The function main() is a starting point of every C program. The execution of the program always begins with the function main ().

Declaration Part

The declaration part declares the entire variables that are used in executable part. The initialisations of variables are also done in this section. Initialisation means providing initial value to the variables.

Executable Part

This part contains the statements following the declaration of the variables. This part conatins a set of statements or a single statement. These statements are enclosed between the braces.

User Defined Function

The functions defined by the user are called user-defined functions. These functions are generally defined after the main () function.

1.3 STEPS FOR EXECUTING THE PROGRAM

1. **Creation of program**

   Programs should be written in C editor. The file name does not necessarily include extension C. The default extension is C.

2. **Compilation of a program**

   The source program statements should be translated into object programs which is suitable for execution by the computer. The translation is done after correcting each statement. If there is no error, compilation proceeds and translated program are stored in another file with the same file name with extension ".obj".

3. **Execution of the program**

   After the compilation the executable object code will be loaded in the computers main memory and the program is executed.

1.4 C CHARACTER SET

Letters	Digits	White Spaces
Capital A to Z	All decimal digits 0 to 9	Blank space
Small a to z		Horizontal tab
		Vertical tab
		New line
		Form feed

Special Characters

,	Comma	&	Ampersand
.	dot	^	Caret
;	Semicolon	*	Asterisk
:	Colon	-	Minus
'	Apostrophe	+	Plus

"	Quotation mark	<	Less than
!	Exclamation mark	>	Greater than
|	Vertical bar	()	Parenthesis left/right
/	Slash	[ ]	Bracket left/right
\	Back slash	{}	Braces left/right
~	Tilde	%	Percent
_	Underscore	#	Number sign or Hash
$	Dollar	=	Equal to
?	Question mark	@	At the rate

1.5 DELIMITERS

Delimiters	Use
: Colon	Useful for label
; Semicolon	Terminates the statement
( ) Parenthesis	Used in expression and function
[ ] Square Bracket	Used for array declaration
{ } Curly Brace	Scope of the statement
# hash	Preprocessor directive
, Comma	Variable separator

1.6 C KEYWORDS

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

1.7 IDENTIFIERS

Identifiers are names of variables, functions, and arrays. They are user-defined names, consisting sequence of letters and digits, with the letter as the first character.

1.8 CONSTANTS

Values do not change during the execution of the program

Types:

1. **Numerical constants:**
   - — Integer constants

     These are the sequence of numbers from 0 to 9 without decimal points or fractional part or any other symbols. It requires minimum two bytes and maximum four bytes.

     Eg: 10,20, + 30, – 14
   - — Real constants

     It is also known as floating point constants.

     Eg: 2.5, 5.342
2. **Character constants:**
   - — Single character constants

     A character constant is a single character. Characters are also represented with a single digit or a single special symbol or white space enclosed within a pair of single quote marks

     Eg: 'a', '8', " ".
   - — String constants

     String constants are sequence of characters enclosed within double quote marks.

     Eg: "Hello", "india", "444"

1.9 VARIABLES

It is a data name used for storing a data value. Its value may be changed during the program execution. The value of variables keeps on changing during the execution of a program.

1.10 DATA TYPES

Data type	Size (Bytes)	Range	Format Specifiers
Char	1	– 128 to 127	%c
Unsigned char	1	0 to 255	%c

Short or int	2	– 32,768 to 32, 767	%i or %d
Unsigned int	2	0 to 655355	%u
Float	4	3.4e – 38 to +3.4e +38	%f or %g
Long	4	= 2147483648 to 2147483647	%ld
Unsigned long	4	0 to 4294967295	%lu
Double	8	1.7e – 308 to 1.7e+308	%lf
Long double	10	3.4e – 4932 to 1.1e+4932	%lf

1.11 OPERATORS

It indicates an operation to be performed on data that yields value.

Types

Type of Operator	Symbolic representation
Arithmetic operators	+, -, *, /, %
Relational operators	>, <, ==, >=, <=, !=
Logical operators	&&, ||, !=
Increment and decrement operator	++ and --
Assignment operator	=
Bitwise operator	&, |, ^, >>, <<, ~
Comma operator	,
Conditional operator	?:

1.12 INPUT AND OUTPUT

Reading data from input devices and displaying the results on the screen are the two main tasks of any program.

Formatted Functions

— The formatted input/output functions read and write all types of values

Input	*Output*
Scanf()	printf()

Unformatted Functions

— The unformatted input/output functions only work with the charcter data type

Input	*Output*
getch()	putch()
getche()	putchar()
getchar()	put()
gets()	

1.13 DECISION STATEMENTS

It checks the given condition and then executes its sub-block. The decision statement decides the statement to be executed after the success or failure of a given condition.

Types:

1. If statement
2. If-else statement
3. Nested if-else statement
4. Break statement
5. Continue statement
6. Goto statement
7. Switch() statement
8. Nested switch ()case
9. Switch() case and Nested if

Statement	**Syntax**
If statement	if(condition) Statement;

If-else statement	If (condition)
	{
	Statement 1;
	Statement 2;
	}
	else
	{
	Statement 3;
	Statement 4;
	}

Nested if-else statement	If (condition) {

	Statement 1;
	Statement 2;
	}
	Else if (condition)
	{
	Statement 3;
	Statement 4;
	}
	Else
	{
	Statement 5;
	Statement 6;
	}

Break statement	Break;
Continue statement	Continue;
Goto statement	goto label;

Switch() statement	Switch (variable or expression)
	{
	Case constant A:
	Statement;
	Break;
	Case constant B:
	Statement;
	Break;
	Default:
	Statement;
	}

1.14 LOOP CONTROL STATEMENTS

Loop is a block of statements which are repeatedly executed for certain number of times.

Types

1. For loop
2. Nested for loops
3. While loop
4. do while loop
5. do-while statement with while loop

Statement	Syntax

For loop	For(initialize counter; test condition; re-evaluation parameter)
	{
	Statement;
	Statement;
	}

Nested for loop	for(initialize counter; test condition; re-evaluation parameter)
	{
	Statement;
	Statement;
	for(initialize counter; test condition; re-evaluation parameter)
	Statement;
	Statement;
	}
	}

While loop	While (test condition)
	{
	Body of the loop
	}

Do while loop	do
	{
	Statement;
	}
	While(condition);

Do-while with while loop	Do while(condition)
	{
	Statement;
	}
	While (condition);

1.15 ARRAYS

It is a collection of similar data types in which each element is located in separate memory locations.

Types

1. One dimensional array
2. Two dimensional arrays
3. Three or multi dimensional arrays

Operations

1. Insertion
2. Deletion
3. Searching
4. Sorting
5. Merging

1.16 STRINGS

Character arrays are called strings. Group of characters, digits, symbols enclosed within quotation marks are called as strings.

String Standard Functions

Functions	Description
Strlen()	Determines the length of a string
Strcpy()	Copies astring from source to destination
Strncpy()	Copies charcters of a string to another string upto the specified length
Stricmp()	Compares characters of two strings
Strcmp()	Compares characters of two strings upto the specified length
Strncmp()	Compares characters of two strings upto the specified length
Strnicmp()	Compares characters of two strings upto the specified length
Strlwr()	Converts uppercase characters of a string to lower case
Strupr()	Converts lowercase characters of a string to upper case
Strdup()	Duplicates a string
Strchr()	Determines the first occurrence of a given character in a string
Strrchr()	Determines the last occurrence of a given character in a string
Strstr()	Determines the first occurrence of a given string in another string
Strcat()	Appends source string to destination string
Strrev()	Reverses all characters of a string
Strset()	Sets all characters of a string with a given argument or symbol
Strspn()	Finds up to what length two strings are identical
Strpbrk()	Searches the first occurrence of the character in a given string and then displays the string starting from that charcter

1.17 FUNCTIONS

It is a self-contained block or a sub program of one or more statements that performs a special task.

Declaration of functions

```
Function_name (argument/parameter)
Argument declaration;
{
Local variable declaration;
Statement1;
Statement 2;
Return (value);
}
```

Call by value

In this type, value of actual arguments is passed to the formal arguments and the operation is done on the formal arguments. Any change made in the formal argument does not effect the actual arguments because formal arguments are photo copies of actual arguments.

Call by reference

In this type, instead of passing values, addresses are passed. Function operates on address rather than values. Here the formal arguments are pointers to the actual argument.

1.18 RECURSION

A function is called repetitively by itself.

Chapter 2 INTRODUCTION—C PROGRAMS

1] Program to find sum of two numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,s;
clrscr();
printf(“Enter two no: ”);
scanf(“%d%d",&a,&b);
s=a+b;
printf(“sum=%d”,s);
getch();
}
```

Output:

```
Enter two no: 5
6
sum=11
```

2] Program to find area and circumference of circle.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int r;
float pi=3.14,area,ci;
clrscr();
printf(“enter radius of circle: ”);
```

```
scanf("%d",&r);
area=pi*r*r;
printf("area of circle=%f ",area);
ci=2*pi*r;
printf("circumference=%f ",ci);
getch();
}
```

Output:

```
enter radius of a circle: 5
area of circle=78.000
circumference=31.4
```

3] Program to find the simple interest.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int p,r,t,si;
clrscr();
printf("enter principle, Rate of interest & time to find simple interest: ");
scanf("%d%d%d",&p,&r,&t);
si=(p*r*t)/100;
printf("simple intrest= %d",si);
getch();
}
```

Output:

```
enter principle, rate of interest & time to find simple interest: 500
5
2
simple interest=50
```

4] Program to convert temperature from degree centigrade to Fahrenheit.

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

```
float c,f;
clrscr();
printf("enter temp in centigrade: ");
scanf("%f",&c);
f=(1.8*c)+32;
printf("temp in Fahrenheit=%f ",f);
getch();
}
```

Output:

```
enter temp in centigrade: 32
temp in Fahrenheit=89.59998
```

5] Program to calculate sum of 5 subjects and find percentage.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int s1,s2,s3,s4,s5,sum,total=500;
float per;
clrscr();
printf("enter marks of 5 subjects: ");
scanf("%d%d%d%d%d",&s1,&s2,&s3,&s4,&s5);
sum=s1+s2+s3+s4+s5;
printf("sum=%d",sum);
per=(sum*100)/total;
printf("percentage=%f",per);
getch();
}
```

Output:

```
enter marks of 5 subjects: 60
65
50
60
60
sum=300
percentage=60.000
```

6] Program to show swap of two no's without using third variable.

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

```
void main()
{
int a,b;
clrscr();
printf("enter value for a & b: ");
scanf("%d%d",&a,&b);
a=a+b;
b=a-b;
a=a-b;
printf("after swapping the value of a & b: %d %d",a,b);
getch();
}
```

Output:

```
enter value for a & b: 4  5
after swapping the value of a & b: 5 4
```

7] Program to reverse a given number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,a,r=0;
clrscr();
printf("enter any no to get its reverse: ");
scanf("%d",&n);
while(n>=1)
{
a=n%10;
r=r*10+a;
n=n/10;
}
printf("reverse=%d",r);
getch();
}
```

Output:

```
enter any no to get its reverse: 456
reverse=654
```

8] Program to find gross salary.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int gs,bs,da,ta;
clrscr();
printf("enter basic salary: ");
scanf("%d",&bs);
da=(10*bs)/100;
ta=(12*bs)/100;
gs=bs+da+ta;
printf("gross salary=%d",gs);
getch();
}
```

Output:

```
enter basic salary: 100
gross salary=122
```

9] Program to print a table of any number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int gs,bs,da,ta;
clrscr();
printf("enter basic salary: ");
scanf("%d",&bs);
da=(10*bs)/100;
ta=(12*bs)/100;
gs=bs+da+ta;
printf("gross salary=%d",gs);
getch();
}
```

Output:

```
enter a no to know table: 2
2*1=2
```

```
2*2=4
2*3=6
2*4=8
2*5=10
2*6=12
2*7=14
2*8=16
2*9=18
2*10=20
```

10] Program to find greatest in 3 numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
clrscr();
printf("enter value of a, b & c: ");
scanf("%d%d%d",&a,&b,&c);
if((a>b)&&(a>c))
printf("a is greatest");
if((b>c)&&(b>a))
printf("b is greatest");
if((c>a)&&(c>b))
printf("c is greatest");
getch();
}
```

Output:

```
enter value for a, b& c: 5
7
4
b is greatest
```

11] Program to show the use of conditional operator.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
printf("enter value for a & b: ");
```

```
scanf("%d%d",&a,&b);
(a>b)?printf("a is greater"):printf("b is greater");
getch();
}
```

Output:

```
enter value for a & b: 5
7
b is greater
```

12] Program to find that entered year is leap year or not.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n;
clrscr();
printf("enter any year: ");
scanf("%d",&n);
if(n%4==0)
printf("year is a leap year");
else
printf("year is not a leap year");
getch();
}
```

Output:

```
enter any year: 1947
year is not a leap year
```

13] Program to find whether given no. is even or odd.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n;
clrscr();
printf("enter any no: ");
scanf("%d",&n);
if(n%2==0)
 printf("no is even");
else
```

```
printf("no is odd");
getch();
}
```

Output:

```
enter any no: 5
no is odd
```

14] Program to shift inputed data by two bits to the left.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x,y;
clrscr();
printf("Read the integer from keyboard :- ");
scanf("%d",&x);
x<<=3;
y=x;
printf("\nThe left shifted data is = %d ",y);
getch();
}
```

Output:

```
Read the integer from keyboard :- 2
The left shifted data is = 16
```

15] Program to use switch statement. Display Monday to Sunday.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char ch;
clrscr();
printf("enter m for Monday\nt for Tuesday\nw for Wednesday\nh for Thursday\nf for
Friday\ns for Saturday\nu for Sunday);
scanf("%c",&ch);
switch(ch)
{
case 'm':
```

```
case ‘M’:
printf(“monday”);
break;
case ‘t’:
case ‘T’:
printf(“tuesday”);
break;
case ‘w’:
case ‘W’:
printf(“wednesday”);
break;
case ‘h’:
case ‘H’:
printf(“thursday”);
break;
case ‘f ’:
case ‘F’:
printf(“friday”);
break;
case ‘s’:
case ‘S’:
printf(“saturday”);
break;
case ‘u’:
case ‘U’:
printf(“sunday”);
break;
default :
printf(“wrong input”);
break;
}
getch();
}
```

Output:

```
enter m for Monday
t for Tuesday
```

w for Wednesday
h for Thursday
f for Friday
s for Saturday
u for Sunday: f
friday

16] Program to display arithmetic operator using switch case.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,n,s,m,su,d;
clrscr();
printf("enter two no's : ");
scanf("%d%d",&a,&b);
printf("enter 1 for sum\n2 for multiply\n3for subtraction\n4 for division: ");
scanf("%d",&n);
switch(n)
{
case 1:
s=a+b;
printf("sum=%d",s);
break;
case 2:
m=a*b;
printf("multiply=%d",m);
break;
case 3:
su=a-b;
printf("subtraction=%d",su);
break;
case 4:
d=a/b;
printf("divission=%d",d);
break;
```

```
default:
printf("wrong input");
break;
}
getch();
}
```

Output:

```
enter two no's: 8
4
enter 1 for sum
2 for multiply
3 for subtraction
4 for division: 1
sum=12
```

17] Program to display first 10 natural no. & their sum.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,sum=0;
clrscr();
for(i=1;i<=10;i++)
{
printf("%d no is= %d\n",i,I);
sum=sum+i;
}
printf("sum =%d",sum);
getch();
}
```

Output:

```
1 no is=1
2 no is=2
3 no is=3
4 no is=4
5 no is=5
```

6 no is=6
7 no is=7
8 no is=8
9 no is=9
10 no is=10
sum=55

18] Program to print stars Sequence1:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j;
clrscr();
for(i=1;i<=5;i++)
{
for(j=1;j<=i;j++)
printf("*");
printf("\n");
}
getch();
}
```

Output:

```
*
**
***
****
*****
```

19] Program to print stars Sequence2.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,k;
clrscr();
for(i=1;i<=5;i++)
```

```
{
for(j=5;j>=i;j--)
printf(“ ”);
for(k=1;k<=i;k++)
printf(“*”);
printf(“\n”);
}
getch();
}
```

Output:

```
    *
   **
  ***
 ****
*****
```

20] Program to print stars Sequence3.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,k;
clrscr();
for(i=1;i<=3;i++)
{
for(j=3;j>=i;j--)
printf(“ ”);
{
for(k=1;k<=i*2-1;k++)
printf(“*”);
}
printf(“\n”);
}
getch();
}
```

Output:

```
    *
   ***
  *****
```

21] Program to print Fibonacci series up to 100.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=1,b=1,c=0,i;
clrscr();
printf("%d\t%d\t",a,b);
for(i=0;i<=10;i++)
{
c=a+b;
if(c<100)
{
printf("%d\t",c);
}
a=b;
b=c;
}
getch();
}
```

Output:

1 1 2 3 5 8 13 21 34 55 89

22] Program to find factorial of a number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,i,fact=1;
clrscr();
printf(“Enter any no: ”);
scanf(“%d”,&n);
```

```
for(i=n;i>=1;i--)
{
fact=fact*i;
}
printf("Factorial=%d",fact);
getch();
}
```

Output:

```
Enter a no: 5
Factorial=120
```

23] Program to find whether given no. is a prime no. or not.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,n,r=0;
clrscr();
printf("Enter any no: ");
scanf("%d",&n);
for(i=2;i<=n-1;i++)
{
if(n%i==0)
r=1;
break;
}
if(r==0)
printf("prime no");
else
printf("Not prime");
getch();
}
```

Output:

```
Enter any no: 16
Not prime
```

24] Program to display sum of series 1+1/2+1/3+…….+1/n.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,i,sum=0;
clrscr();
printf("Enter any no: ");
scanf("%d",&n);
printf("1");
for(i=2;i<=n-1;i++)
printf(" 1/%d +",i);
for(i=1;i<=n;i++)
sum=sum+i;
printf(" 1/%d",n);
printf("\nSum=1/%d",sum+1/n);
getch();
}
```

Output:

```
Enter any no: 7
1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/6 + 1/7
Sum=1/28
```

25] Program to display series and find sum of 1+3+5+……..+n.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,i,sum=0;
clrscr();
printf("Enter any no: ");
scanf("%d",&n);
for(i=1;i<n;i=i+2)
{
printf("%d+",i);
sum=sum+i;
```

```
}
printf("%d",n);
printf("\nsum=%d",sum+n);
getch();
}
```

Output:

```
Enter any no: 7
1+3+5+7
Sum=16
```

26] Program to use bitwise AND operator between the two integers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
clrscr();
printf(“Read the integers from keyboard:- ”);
scanf(“%d %d”,&a,&b);
c=a&b;
printf(“\nThe Answer after ANDing is:  %d ”,c);
getch();
}
```

Output:

```
Read the integers from keyboard:- 8 4
The Answer after ANDing is: 0
```

27] Program to add two number using pointers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int *p1,*p2,sum;
clrscr();
printf(“enter two no’s: ”);
scanf(“%d%d”,&*p1,&*p2);
sum=*p1+*p2;
```

```
printf("sum=%d",sum);
getch();
}
```

Output:

```
enter two no's: 10
20
sum=30
```

28] Program to add two number using pointers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int *p1,*p2,sum;
clrscr();
printf("enter two no's: ");
scanf("%d%d",&*p1,&*p2);
sum=*p1+*p2;
printf("sum=%d",sum);
getch();
}
```

Output:

```
enter two no's: 10
20
sum=30
```

29] Program to show sum of 10 elements of array & show the average.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10],i,sum=0;
float av;
clrscr();
printf("enter elements of an aaray: ");
for(i=0;i<10;i++)
scanf("%d",&a[i]);
```

```
for(i=0;i<10;i++)
sum=sum+a[i];
printf("sum=%d",sum);
av=sum/10;
printf("average=%.2f",av);
getch();
}
```

Output:

```
enter elements of an array: 4
5
6
1
2
3
5
5
4
7
sum=42
average=4.22
```

30] Program to find the maximum no. in an array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5],max,i;
clrscr();
printf("enter element for the array: ");
for(i=0;i<5;i++)
scanf("%d",&a[i]);
max=a[0];
for(i=1;i<5;i++)
{
if(max<a[i])
max=a[i];
```

```
}
printf("maximum no= %d",max);
getch();
}
```

Output:

```
enter elements for array: 5
4
7
1
2
maximum no= 7
```

31] Program to display a matrix.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[3][2],b[3][2],i,j;
clrscr();
printf("enter value for a matrix: ");
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
scanf("%d",&a[i][j]);
}
printf("enter value for b matrix: ");
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
scanf("%d",&b[i][j]);
}
printf("\na matrix is\n\n");
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
```

```
{
printf(" %d ",a[i][j]);
}
printf("\n");
}
printf("\nb matrix is\n\n");
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
{
printf(" %d ",b[i][j]);
}
printf("\n");
}
getch();
}
```

Output:

```
enter value for a matrix: 7
8
9
4
5
6
enter value for b matrix: 3
2
1
4
5
6

a matrix is
7  8
9  4
5  6
```

b matrix is
3 2
1 4
5 6

32] Program to find sum of two matrices.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[3][2],b[3][2],c[3][2],i,j;
clrscr();
printf("Enter value for 1 matrix: ");
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
scanf("%d",&a[i][j]);
}
printf("Enter value for 2 matrix: ");
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
scanf("%d",&b[i][j]);
}
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
c[i][j]=a[i][j]+b[i][j];
}
printf("Sum of matrix is\n");
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
{
printf("%d\t",c[i][j]);
}
```

```
printf("\n");
}
getch();
}
```

Output:

```
Enter value for 1 matrix: 1
2
3
4
5
6
Enter value for 2 matrix: 4
5
6
1
3
2
Sum of matrix is
5 7
9 5
8 8
```

33] Program to find subtraction of two matrices.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5],b[5],c[5],i;
clrscr();
printf("enter value for array a ");
for(i=0;i<5;i++)
scanf("%d",&a[i]);
printf("enter value for array b ");
for(i=0;i<5;i++)
scanf("%d",&b[i]);
for(i=0;i<5;i++)
```

```
c[i]=a[i]-b[i];
printf(“subtraction”);
for(i=0;i<5;i++)
printf(“ %d ”,c[i]);
getch();
}
```

Output:

```
enter value for array a: 7
8
9
4
5
enter value for array b: 4
5
6
1
2
subtraction 3 3 3 3 3
```

34] Program to find multiplication of two matrices.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[3][2],b[3][2],c[3][2],i,j;
clrscr();
printf(“enter value for 1 matrix: ”);
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
scanf(“%d”,&a[i][j]);
}
printf(“enter value for 2 matrix: ”);
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
```

```
scanf("%d",&b[i][j]);
}
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
c[i][j]=a[i][j]*b[i][j];
}
printf("matrix is\n");
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
{
printf(" %d ",c[i][j]);
}
printf("\n");
}
getch();
}
```

Output:

```
enter value for 1 matrix: 7
8
9
4
5
6
enter value for 2 matrix: 3
2
1
2
5
6
matrix is
21  16
9    8
25  36
```

35] Program to find transpose of a matrix.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[3][2],b[2][3],i,j;
clrscr();
printf("Enter value for matrix: ");
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
scanf('%d",&a[i][j]);
}
printf("Matrix:\n");
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
printf(" %d ",a[i][j]);
printf("\n");
}
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
b[j][i]=a[i][j];
}
printf("Transpose matrix:\n");
for(i=0;i<2;i++)
{
for(j=0;j<3;j++)
printf(" %d ",b[i][j]);
printf("\n");
}
getch();
}
```

Output:

```
Enter value for matrix: 4
5
6
1
2
3
Matrix:
4 5
6 1
2 3
Transpose matrix:
4 6 2
5 1 3
```

36] Program to find the maximum number in array using pointer.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int max,i,*a[5];
clrscr();
printf("enter element for the array: ");
for(i=0;i<5;i++)
scanf("%d",&*a[i]);
max=*a[0];
for(i=1;i<5;i++)
{
if(max<*a[i])
max=*a[i];
}
printf("maximum no= %d",max);
getch();
}
```

Output:

enter elements for array: 5
4
7
1
2
maximum no= 7

37] Program to show input and output of a string.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char a[50];
clrscr();
printf("enter any string: ");
gets(a);
puts(a);
getch();
}
```

Output:

enter any string: hi everyone
hi everyone

38] Program to find square of a number using functions.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int rev(int);
int r,a;
clrscr();
printf("enter any no: ");
scanf("%d",&a);
r=rev(a);
printf("square is : %d",r);
```

```
getch();
}
int rev(int x)
{
return(x*x);
}
```

Output:

```
enter any no: 5
square is : 25
```

39] Program to swap two numbers using functions.

```
#include<stdio.h>
#include<conio.h>
void main()
{
void swap(int,int);
int a,b,r;
clrscr();
printf("enter value for a&b: ");
scanf("%d%d",&a,&b);
swap(a,b);
getch();
}
void swap(int a,int b)
{
int temp;
temp=a;
a=b;
b=temp;
printf("after swapping the value for a & b is : %d %d",a,b);
}
```

Output:

```
enter value for a&b: 4
5
after swapping the value for a & b : 5 4
```

40] Program to find factorial of a number using functions.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,f;
int fact(int);
clrscr();
printf("enter a no: ");
scanf("%d",&a);
f=fact(a);
printf("factorial= %d",f);
getch();
}
int fact(int x)
{
int fac=1,i;
for(i=x;i>=1;i--)
fac=fac*i;
return(fac);
}
```

Output:

```
enter a no: 5
factorial=120
```

41] Program to show table of a number using functions.

```
#include<stdio.h>
#include<conio.h>
void main()
{
void table();
clrscr();
table();
getch();
}
```

```
void table()
{
int n,i,r;
printf("enter a no to know table: ");
scanf("%d",&n);
for(i=1;i<=10;i++)
{
r=n*i;
printf("%d*%d=%d\n",n,i,r);
}
}
```

Output:

```
enter a no to know table: 2
2*1=2
2*2=4
2*3=6
2*4=8
2*5=10
2*6=12
2*7=14
2*8=16
2*9=18
2*10=20
```

42] Program to show call by value.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,swap();
clrscr();
a=5;
b=10;
printf("value of a=%d & value of b=%d before swap ",a,b);
swap(a,b);
```

```
printf("\nvalue of a =%d & b=%d after swap",a,b);
getch();
}
int swap(int x,int y)
{
int temp;
temp=x;
x=y;
y=temp;
}
```

Output:

value of a=5 & value of b=10 before swap
value of a=5 & b=10 after swap

43] Program to show call by reference.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,*aa,*bb,swap();
clrscr();
a=5;
b=10;
aa=&a;
bb=&b;
printf("value of a= %d & value of b=%d before swap",a,b);
swap(aa,bb);
printf("\nvalue of a=%d & b=%d after swap",a,b);
getch();
}
int swap(int *x,int *y)
{
int temp;
temp=*x;
```

```
*x=*y;
*y=temp;
}
```

Output:

value of a= 5 & value of b=10 before swap
value of a=10 & b=5 after swap

44] Program to find largest of two numbers using functions.

```
#include<stdio.h>
#include<conio.h>
void main()
{
void max();
clrscr();
max();
getch();
}
void max()
{
int a[5],max,n,i;
printf("How many no's you want to enter: ");
scanf("%d",&n);
printf("Enter element for the array: ");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
max=a[0];
for(i=1;i<5;i++)
{
if(max<a[i])
max=a[i];
}
printf("maximum no= %d",max);
}
```

Output:

How many no's you want to enter: 4
Enter element for array: 4
5
6
1
maximum no: 6

45] Program to find factorial of a number using recursion.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n;
clrscr();
printf("enter number: ");
scanf("%d",&n);
if(n<0)
printf("invalid number");
else
printf("%d!=%d",n,fact(n));
getch();
}
int fact(int x)
{
if(x==0)
return 1;
else
return(x*fact(x-1));
}
```

Output:

enter number: 5
5!=120

46] Program to find whether a string is palindrome or not.

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

```
void main()
{
char s1[20],s2[20];
clrscr();
printf("enter a string: ");
scanf("%s",s1);
strcpy(s2,s1);
strrev(s2);
if(strcmp(s1,s2)==0)
printf("string is a palindrome");
else
printf("not a palindrome string");
getch();
}
```

Output:

```
enter a string: abc
not a palindrome string
```

47. File Operations:

```
#include<stdio.h>
#include<conio.h>
void main()
{
file *fp,*fp1;
char c;
clrscr();
fp=fopen("test.c","w");
printf("\nenter the contents  for file1(#.end)\n");
c=getchar();
while(c!='#')
{
fputc(c,fp);
c=getchar();
}
rewind(fp);
fp=fopen("test.c","r");
```

```
fp1=fopen("tes.c","w");
c=fgetc(fp);
while(c!=eof)
{
fputc(c,fp);
c=fgetc(fp);
}
fclose(fp);
fclose(fp1);
fp1=fopen("tes.c","r");
c=fgetc(fp1);
printf("\nthe contents in file 2\n");
while(c!=eof)
{
putchar(c);
c=fgetc(fp1);
}
fclose(fp1);
getch();
}
```

Output:

enter the contents of file1(#-end)
good morning#
the contents of file2
good morning

48. Merging One Dimensional Array – Excluding The Repeating Element

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[50],b[50],n1,n2,i,x;
clrscr();
printf("enter the number of elements in the first array");
scanf("%d",&n1);
printf("enter the elements\n");
```

```
for(i=0;i<n1;i++)
{
printf("enter a[%d]",i+1);
scanf("%d",&a[i]);
}
printf("enter the number of elements in the second array");
scanf("%d",&n2);
printf("enter the elements\n");
for(i=0;i<n2;i++)
{
printf("enter b[%d]",i+1);
scanf("%d",&b[i]);
}
for(x=0;x<n1;x++)
{
for(i=0;i<n2;i++)
{
if(b[i]==a[x])
{
b[i]=' ';
}
}
}
for(i=o;i<n1;i++)
{
printf("%d",a[i]);
}
for(i=0;i<n2;i++)
{
if(b[i]==' ';)
continue;
else
printf("%d",b[i]);
}
getch();
}
```

Output:

```
Enter the number of elements in the first array
3
Enter the elements
3
5
7
Enter the number of elements in the first array
3
Enter the elements
2
5
9
3 5 7 2 9
```

49. Number Of Occurrences Of Vowels, Consonants, Words, Spaces And Special Characters In The Given Statement.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
void main()
{
char s[100];
int vow=0,cons=0,spc=0,punc=0,l,i;
clrscr();
printf("enter the statement\n");
gets(s);
l=strlen(s);
for(i=0;i<l;i++)
{
if(isalpha(s[i]))
{
if(s[i]=='a'||s[i]=='e'||s[i]=='i'||s[i]=='o'||s[i]=='u')
{
```

```
vow++;
}
else
{
cons++;
}
}
if(isspace(s[i])
{
spc++;
}
if(ispunct(s[i]))
{
punc++;
}
}
printf("\nno.of words=%d",spc+1);
printf("\nno.of vowels=%d",vow);
printf("\nno.of consonants=%d",cons);
printf("\nno.of space=%d",spc);
printf("\nno.on special characters=%d",punc);
getch();
}
```

Output:

```
Enter the statement
*Nothing is impossible in the world.
No.of words=6
No.of vowels=10
No.of consonants=19
No.of space=5
No.of special characters=1
```

50. Write a program to create enumerated data type for 12 months. Display their values in integer constants.

```
#include<stdio.h>
#include<conio.h>
void main()
{
Enum month(Jan, Feb, Mar, Apr, May, June, July, Aug,Sep, Oct, Nov, Dec)
clrscr();
printf("Jan=%d", Jan);
printf("Feb=%d", Feb);
printf("June=%d", June);
printf("Dec=%d", Dec);
}
```

Output:

```
Jan =0
Feb=1
June=5
Dec=11
```

Chapter 3 FUNDAMENTALS—C PROGRAMS

1. A program to evaluate the equation y=xn when n is a non-negative integer.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int count, n;
        float x,y;
        printf("Enter the values of x and n:");
        scanf("%f%d", &x,&n);
        y=1.0;
        count=1;
        while(count<=n)
        {
                y=y*x;
                count++;
        }
        printf("x=%f; n=%d; x to power n=%f", x, n,y);
}
```

Output:

Enter the values of x and n: 2.5 4

X=2.500000; n=4; x to power n= 39.062500

2. A program to print the multiplication table from 1*1 to 12*10.

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

```
#define COLMAX 10
#define ROWMAX 12
void main()
{
    int  row, column, y;
    row=1;
    printf("MULTIPLICATION TABLE ");
    printf("----------------------------------------");
        do
        {
        column=1;
        do
        {
        y=row*column;
        printf("%d", y);
        column=column +1;
        }
        while(column<=COLMAX);
        printf("\n");
        row=row+1;
        }
        while(row<=ROWMAX);
        printf("-----------------------------------------")
}
```

Output:

MULTIPLICATION TABLE

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80

9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100
11	22	33	44	55	66	77	88	99	110
12	24	36	48	60	72	84	96	108	120

3. Program uses a for loop to print the powers of 2 table for the power 0 to 20, both positive and negative.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        long int p;
        int n;
        double q;
        printf("-------------------------------------------------------------");
        printf(" 2 to power n        n        2 to power -n");
        printf("-------------------------------------------------------------");
        p=1;
        for(n=0; n<21; ++n)
        {
                if(n==0)
                        P=1;
                else
                        p=p*2;
                q=1.0/(double)p;
                printf("10ld 10%d  %20.12lf", p,n,q);
                }
                printf("-----------------------------------------");
        }
```

Output:

```
-----------------------------------------------------------------------------------------
2 to power n                    n                    2 to power -n
-----------------------------------------------------------------------------------------
                1                    0              1.000000000000
                2                    1              0.500000000000
```

4	2	0.250000000000
8	3	0.125000000000
16	4	0.062500000000
32	5	0.031250000000
64	6	0.015625000000
128	7	0.007812500000
256	8	0.003906250000
512	9	0.001953125000
1024	10	0.000976562500
2048	11	0.000488281250
4096	12	0.000244140625
8192	13	0.000122070313
16384	14	0.000061035156
32768	15	0.000030517578
65536	16	0.000015258789
131072	17	0.000007629395
262144	18	0.000003814697
524288	19	0.000001907349
1048576	20	0.000000953674

4. A class of n students take an annual examination in m subjects. A program to read the marks obtained by each student in various subjects and to compute and print the total marks obtained by each of them.

```
#include<stdio.h>
#include<conio.h>
#define FIRST 360
# define SECOND 240
void main()
{
int n, m, i, j, roll number, marks, total;
printf("Enter number of students and subjects");
scanf("%d%d", &n,&m);
printf("\n");
for(i=1; i<=n; ++i)
{
printf("Enter roll number:");
```

```
scanf("%d", &roll_number);
total=0;
printf("Enter marks of %d subjects for ROLL NO", m, roll number);
for(j=1; j<=m; j++)
{
scanf("%d", &marks);
total=total+marks;
}
printf("TOTAL MARKS =%d", total);
if(total>=FIRST)
printf("(First division)");
else if (total>=SECOND)
printf("(Second division)");
else
printf("(***FAIL***)");
}
}
```

Output:

```
Enter number of students and subjects
3 6
Enter roll_number: 8701
Enter marks of 6 subjects for ROLL NO 8701
81 75 83 45 61 59
TOTAL MARKS =404 (First division)
Enter roll_number:8702
Enter marks of 6 subjects for ROLL NO 8702
51 49 55 47 65 41
TOTAL MARKS =308(Second division)
Enter roll_number: 8704
40 19 31 47 39 25
TOTAL MARKS=201(*** FAIL ***)
```

5. The program illustrates the use of the break statement in a C program.

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

```
{
int m;
float x, sum, average;
printf("This program computes the average of a set of computers");
printf("Enter values one after another");
printf("Enter a NEGATIVE number at the end");
sum=0;
for(m=1;m<=1000; ++m)
{
scanf("%f",&x);
if(x<0)
break;
sum+=x;
}
average=sum/(float)(m-1);
printf("\n");
printf("Number of values =%d",m-1);
printf("sum=%f", sum);
printf("Average=%f", average);
}
```

Output:

```
This program computes the average of a set of numbers
Enter values one after another
Enter a NEGATIVE number at the end
21 23 24 22 26 22 -1
Number of values =6
Sum= 138.000000
Average=23.000000
```

6. A program to evaluate the series $1/1-x= 1+x+x^2 +x^3 +.....+x^n$.

```
#include<stdio.h>
#include<conio.h>
#define LOOP 100
#define ACCURACY 0.0001
void main()
{
```

```
int n;
float x, term, sum;
printf("input value of x :");
scanf("%f", &x);
sum=0;
for(term=1, n=1; n<=LOOP; ++n)
{
sum+=term;
if(term<=ACCURACY)
goto output;
term*=x;
}
printf("FINAL VALUE OF N IS NOT SUFFICIENT TO ACHIEVE DESIRED ACCURACY");
goto end;
output:
printf("EXIT FROM LOOP");
printf("sum=%f; no. of terms=%d", sum,n);
end:
;
}
```

Output:

```
Input value of x: .21
EXIT FROM LOOP
Sum=1.265800; No. of terms=7
Input value of x: .75
EXIT FROM LOOP
Sum=3.999774; No. of terms=34
Input value of x:.99
FINAL VALUE OF N IS NOT SUFFICIENT TO ACHIEVE DESIRED ACCURACY
```

7. Program illustrates the use of continue statement.

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

```
{
int count, negative;
double number, sqroot;
printf("enter 9999 to STOP");
count=0;
negative=0;
while(count<=100)
{
printf("enter a number:");
scanf("%lf", &number);
if(number==9999)
break;
if(number<0)
{
printf("Number is negative ");
negative++;
continue;
}
sqroot=sqrt(number);
printf("Number=%lf square root=%lf ", number, sqroot);
count++;
}
printf("Number of items done =%d", count);
printf("Negative items=%d", negative);
printf("END OF DATA");
}
```

Output:

```
Enter 9999 to STOP
Enter a number: 25.0
Number=25.000000
Square root =5.000000
Enter a number: 40.5
Number =40.500000
Square root=6.363961
Enter a number:-9
```

```
Number is negative
Enter a number: 16
Number= 16.000000
Square root=4.000000
Enter a number: -14.75
Number is negative
Enter a number: 80
Number=80.000000
Square root=8.944272

Enter a number: 9999
Number of items done=4
Negative items=2
END OF DATA
```

8. Program to print binomial coefficient table.

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
void main()
{
int m, x, binom;
printf("m x");
for (m=0; m<=10; ++m)
printf("------------------------------------------------------------");
m=0;
do
{
printf("%2d", m);
X=0; biom=1;
while(x<=m)
{
if(m= =0 || x= =0)
printf("%4d", binom);
else
```

```
{
binom=binom* (m-x+1)/x;
printf("%4d", binom);
}
x=x+1;
}
printf("\n");
M=m+1'
}
while(m<=MAX);
printf("----------------------------------------------------------");
}
```

Output:

Mx	0	1	2	3	4	5	6	7	8	9	10
0	1										
0	1	1									
2	1	2	1								
3	1	3	3	1							
4	1	4	6	4	1						
5	1	5	10	10	5	1					
6	1	6	15	20	15	6	1				
7	1	7	21	35	35	21	7	1			
8	1	8	28	56	70	56	28	8	1		
9	1	9	36	84	126	126	84	36	9	1	
10	1	10	45	120	210	252	210	120	45	10	1

9. Program to draw a histogram.

```
#include<stdio.h>
#include<conio.h>
#define N 5
void main()
{
int value[N];
int i, j, n, x;
for(n=0; n<N; ++n)
{
```

```
printf("enter employees in group-%d: ", n+1);
scanf("%d", &x);
value[n]=x;
printf("%d", value[n]);
}
printf("\n");
printf("|\n");
for(n=0; n<N; ++n)
{
for(i=1; i<=3; i++)
{
if(i= =2)
printf(" Group-%d | ", n+1);
else
printf("|");
for(j=1; j<=value[n]; ++j)
printf("*");
if(i= =2)
printf("(%d)", value[n]);
else
printf("\n");
}
printf("|\n");
}
}
```

Output:

```
Enter employees in Group -1:12
12
Enter employees in Group -2:23
23
Enter employees in Group -3:35
35
Enter employees in Group -4:20
20
```

```
Enter employees in Group -5:11
11

                |
                |**********
Group-1         |********** (12)
                |**********
                |
                |**********************
Group-2         |**********************(23)
                |**********************
                |
                |************************************
Group-3         |************************************(35)
                |************************************
                |
                |********************
Group-4         |********************(20)
                |********************
                |
                |**********
Group-5         |**********(11)
                |**********
                |
```

10. Program of minimum cost problem.

```
#include<stdio.h>
#include<conio.h>
void main()
{
float p, cost, p1, cost1;
for(p=0; p<=10; p=p+0.1)
{
cost=40-8*p+p*p;
if(p= =0)
```

```
{
        cost1=cost;
        continue;
}
if(cost>=cost1)
break;
cost1=cost;
p1=p;
}
p=(p+p1)/2.0;
cost=40-8&p+p*p;
printf("MINIMUM COST=%.2f AT p=%.1f", cost, p);
}
```

Output:

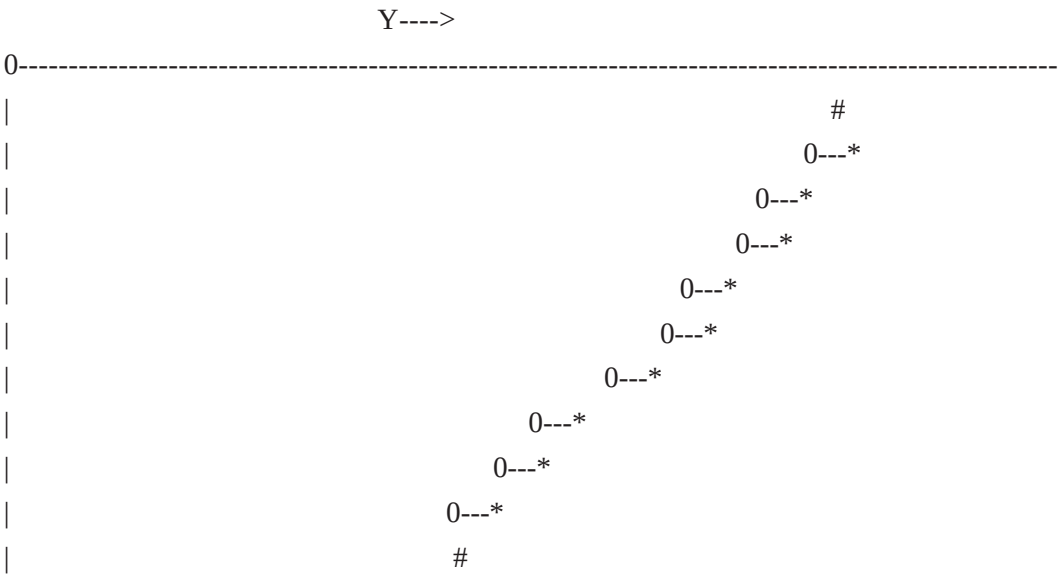
MINIMUM COST = 24.00 AT p=4.0

11. Program for plotting of two functions (y1=exp(-ax); y2=exp(-ax^2 /2)).

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int i;
float a, x, y1, y2;
a=0.4;
printf("                              Y-------->                    " );
printf("0-------------------------------------------------------------------------");
for(x=0; x<5; x=x+0.25)
y1=(int) (50*exp(-a*x)+0.25);
y2=(int) (50*exp(-a*x*x/2)+0.5);
if(y1= =2)
{
if(x= = 2.5)
```

```
printf("X   |");
else
printf("|");
for(i=1; i<=y1-1; ++i)
printf("");
printf("#");
continue;
}
if(y1>y2)
{
if(x= =2.5)
printf("X |");
else
printf(" |");
for(i=1; i<=y2-1; ++i)
printf(" ");
printf("*");
for(i=1; i<=(y1-y2-1); ++i)
printf("_");
printf("0");
continue;
}
if(x==2.5)
printf("X|");
else
printf(" |");
for(i=1; i<=y1-1; ++i)
printf(" ");
printf("0");
for(i=1; i<=(y2-y1-1); ++i)
printf("_");
printf("*");
}
}
```

Output:



12. Write a program using a single –subscribed variable to evaluate the following expressions:

Total = $\sum_{i=1}^{10} X_2^2$ the values of x1, x2, ... are read from the terminal.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i;
float x[10], value, total;
printf("ENTER 10 REAL NUMBERS");
for(i=0; i<10; i++)
{
scanf("%f", &value);
x[i]=value;
}
total=0.0;
for(i=0; i<10; i++)
```

```
total = total + x[i]*x[i];
printf("\n");
for(i=0; i<10; i++)
printf("x[%2d] = %5.2f", i+1, x[i]);
printf("total=%.2f", total);
}
```

Output:

```
ENTER 10 REAL NUMBERS
1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9 10.10
X[1]=1.10
X[2]=2.20
X[3]=3.30
X[4]=4.40
X[5]=5.50
X[6]=6.60
X[7]=7.70
X[8]=8.80
X[9]=9.90
X[10]=10.10

Total = 446.86
```

13. Given below is the list of marks obtained by a class of 50 students in an annual examination.

43 65 51 27 79 11 56 61 82 09 25 36 07 49 55 63 74 81 49 37 40 49 16 75 87 91 33 24 58 78 65 56 76 67 45 54 36 63 12 21 73 49 51 19 39 49 68 93 85 59

Write a program to count the number of students belonging to each of the following groups of marks: 0-9,10-19,20 -29,----,100.

```
#include<stdio.h>
#include<conio.h>
#define MAXVAL 50
#define COUNTER 11
void main()
```

```
{
float value [MAXVAL];
int i, low, high;
int group[COUNTER]={0,0,0,0,0,0,0,0,0,0};
for(i=0; i<MAXVAL; i++)
{
scanf("%f", &value[i]);
++group[(int)(value[i]/10)];
}
printf("\n");
printf("GROUP RANGE FREQUENCEY");
for(i=0; i<COUNTER; i++)
{
low=i*10;
if(i= =10)
high=100;
else
high=low+9;
printf("%2d%3d to %3d%d", i+1, low, high, group[i]);
}
}
```

Output:

43 65 51 27 79 11 56 61 82 09 25 36 07 49 55 63 74 81 49 37 40 49 16 75 87 91 33 24 58 78
65 56 76 67 45 54 36 63 12 21 73 49 51 19 39 49 68 93 85 59 (Input data)

GROUP	RANGE	FREQUENCY
1	0 to 9	2
2	10 to 19	4
3	20 to 29	4
4	30 to 39	5
5	40 to 49	8
6	50 to 59	8
7	60 to 69	7

8	70 to 79	6
9	80 to 89	4
10	90 to 99	2
11	100 to 100	0

14. Write a program for sorting the elements of an array in descending order.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int *arr, temp, i, j, n;
clrscr();
printf("enter the number of elements in the array");
scanf("%d", &n);
arr=(int*)malloc(sizeof(int)*n);
for(i=0;i<n;i++)
{
for(j=i+1; j<n; j++)
{
if(arr[i]<arr[j])
{
temp=arr[i];
arr[i]=arr[j];
arr[j]=temp;
}
}
printf("Elements of array in descending order are");
for(i=0; i<n; i++);
getch();
}
```

Output:

Enter the number of elements in the array: 5
Enter a number: 32
Enter a number: 43
Enter a number: 23

```
Enter a number: 57
Enter a number: 47

Elements of array in descending order are:
57
47
43
32
23
```

15. Write a program for finding the largest number in an array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int *arr, i, j, n, LARGE;
clrscr();
printf("Enter the number of elements in the array");
scanf("%d", &n);
arr=(int*) malloc(sizeof(int)*n);
for(i=0; i<n; i++)
{
printf("Enter a number");
scanf("%d", &arr[i]);
}
LARGE=arr[0];
for(i=1; i<n;i++)
{
if(arr[i]>LARGE)
LARGE=arr[i];
}
printf("The largest number in the array is : %d", LARGE);
getch();
}
```

Output:

Enter the number of elements in the array:5
Enter a number: 32
Enter a number: 43
Enter a number: 23
Enter a number: 57
Enter a number: 47
The largest number in the array is : 57

16. Write a program for removing the duplicate element in an array.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int *arr, i, j, n, x, temp;
clrscr();
printf("Enter the number of elements in the array");
scanf("%d", &n);
arr=(int*) malloc(sizeof(int)*n);
for(i=0; i<n; i++)
{
printf("Enter a number");
scanf("%d", &arr[i]);
}
for(i=0; i<n; i++)
{
for(j=i+1; j<n; j++)
{
if(arr[i]>arr[j])
{
temp=arr[i];
arr[i]=arr[j];
arr[j]=temp;
}
```

```
}
}
printf("Elements of array after sorting");
for(i=0; i<n; i++)
printf("%d", arr[i]);
i=0;
j=1;
while(i<n)
{
if(arr[i]==arr[j])
{
for(x=j; x<n-1; x++)
arr[x]=arr[x+1];
n-;
}
else
{
i++;
j++;
}
}
printf("Elements of array after removing duplicate elements");
for(i=0; i<=n; i++)
printf("%d", arr[i]);
getch();
}
```

Output:

```
Enter the number of elements in the array:5
Enter a number: 3
Enter a number: 3
Enter a number: 4
Enter a number: 6
Enter a number: 4
```

Elements of array after sorting:
3
3
4
4
6
Elements of array after removing duplicate elements:
3
4
6

17. Write a program for finding the desired k^{th} smallest element in an array.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int *arr, i, j, n, temp, k;
clrscr();
printf("enter the number of elements in the array");
scanf("%d", &n);
arr=(int*)malloc(sizeof(int)*n);
for(i=0;i<n;i++)
{
printf("Enter a number");
scanf("%d", &arr[i]);
}
for(j=i+1; j<n; j++)
{
if(arr[i]<arr[j])
{
temp=arr[i];
arr[i]=arr[j];
arr[j]=temp;
}
}
```

```
}
printf("Elements of array after sorting");
for(i=0; i<n; i++);
printf("%d", arr[i]);
printf("Which smallest element do you want to determine");
scanf("%d", k);
if(k<=n)
printf("Desired smallest element is %d", arr[k-1]);
else
printf("Please enter a valid value for finding the particular smallest element");
getch();
}
```

Output:

```
Enter the number of elements in the array:5
Enter a number: 33
Enter a number: 32
Enter a number: 46
Enter a number: 68
Enter a number: 47
Elements of array after sorting:
32
33
46
47
68
Which smallest element do you want to determine? 3
Desired smallest element is 46
```

18. Program to sort a list of numbers and to determine median.

```
#include<stdio.h>
#include<conio.h>
#define N 10
```

```
void main()
{
int i, j, n;
float median, a[N], t;
printf("Enter the number of items");
scanf("%d", &n);
printf("Input %d values ", n);
for(i=1; i<=n; i++)
scanf("%f", &a[i]);
for(i=1; i<=n-1; i++)
{
for(j=1; j<=n-1; j++)
{
if(a[j]<=a[j+1])
t=a[j];
a[j]=a[j+1];
a[j+1]=t;
}
else
continue;
}
}
if(n%2 = =0)
median=(a[n/2]+a[n/2+1])/2.0;
else
median=a[n/2+1];
for(i=1; mi<=n; i++)
printf("%f", a[i]);
printf("median is %f", median);
}
```

Output:

```
Enter the number of items
5
Input 5 values
1.111 2.222 3.333 4.444 5.555
```

```
5.555000 4.444000 3.333000 2.222000 1.111000
Median is 3.333000

Enter the number of items
6
Input 6 values
3 5 8 9 4 6

9.000000  8.000000  6.000000 5.000000 4.000000 3.000000
Median is 5.500000
```

19. Program to calculate standard deviation.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define MAXSIZE 100
void main()
{
int i, n;
float  value [MAXSIZE], deviation;
sum=sumsqr=n=0;
printf("Input values: input -1 to end");
for(i=1; i<MAXSIZE; i++)
{
scanf("%f", &value[i]);
if(value[i]= =-1)
break;
sum+=value[i];
n+=1;
}
mean=sum/(float)n;
for(i=1; i<=n; i++)
{
deviation=value[i]-mean;
sumsqr+=deviation* deviation;
```

```
}
variance=sumsqr/(float)n;
stddeviation=sqrt(variance);
printf("Number of items: %d", n);
printf("Mean: %f",mean);
printf("Standard deviation:%f", stddeviation);
}
```

Output:

Input values: input -1 to end
65 9 27 78 12 20 33 49 -1

Number of items: 8
Mean: 36.625000
Standard deviation: 23.510303

20. Program to evaluate responses to a multiple-choice test.

```
#include<stdio.h>
#include<conio.h>
#define STUDENTS 3
#define ITEMS 25
void main()
{
char key[items+1], response[ITEMS+1];
int count, i, student, n, correct[ITEMS+1];
printf("Input key to the items");
for(i=0; i<ITEMS; i++)
scanf("%c", &key[i]);
scanf("%c", &key[i]);
key[i]='\0';
for(student=1; student<=STUDENTS; student++)
{
count=0;
```

```
printf("\n");
printf("Input responses of student -%d", student);
for(i=0; i<ITEMS; i++)
scanf("%c", &response[i]);
scanf("%c", &response[i]);
response[i]='\0';
for(i=0; i<ITEMS; i++)
correct[i]=0;
for(i=0; i<ITEMS; i++)
if(response[i]= = key[i])
{
count=count+1;
count[i]=1;
}
printf("\n");
printf("student-%d", student);
printf("score is %d out of %d", count, ITEMS);
printf("Response to the items below are wrong");
n=0;
for(i=0; i<ITEMS; i++)
if(correct[i]= =0)
{
printf("%d", i+1);
n=n+1;
}
if(n= =0)
printf("NIL");
printf("\n");
}
}
```

Output:

Input key to the items
abcdabcdabcdabcdabcdabcda

Input responses of student-1
abcdabcdabcdabcdabcdabcda

student-1
score is 25 out of 25
response to the following items are wrong
NIL

Input responses of student-2
abcddcbaabcdabcddddddddddd

student-2
score is 14 out of 25
Response to the following items are wrong
5 6 7 8 17 18 19 21 22 23 25

Input responses of student-3
aaaaaaaaaaaaaaaaaaaaaaaa

student-3
score is 7 out of 25
Response to the following items are wrong
2 3 4 6 b7 8 10 11 12 14 15 16 18 19 20 22 23 24

21. Program for production and sales analysis.

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

int M[5][6], s[5][6], c[6], Mvalue[5][6], Svalue [5][6],Mweek[5], Sweek[5], Mproduct[6],
Sproduct[6], Mtotal, Stotal, i, j, number;
printf("Enter products manufactured week wise");
```

```
printf("M11, M12,-, M21, M22, - etc");
for(i=1; i<=4; i++)
for(j=1; j<=5; j++)
scanf("%d", &M[i][j]);
printf("Enter products sold week wise");
printf("S11, S12,-, S21,S22,- etc");
for(i=1; i<=4; i++)
for(j=1; j<=5; j++)
scanf("%d", &S[i][j]);
printf("enter the cost of each product");
for(j=1;j<=5; j++)
scanf("%d", &C[j]);
for(i=1; i<=4; i++)
for(j=1; j<=5; j++)
{
Mvalue[i][j]=M[i][j]*c[j];
Svalue[i][j]=S[i][j]*c[j];
}
for(i=1; i<=4; i++)
{
Mweek[i]=0;
Sweek[i]=0;
for(j=1; j<5; j++)
{
Mweek[i] +=Mvalue[i][j];
Sweek[i]+= Svalue[i][j];
}
}
for(j=1; j<=5; j++)
{
Mproduct[j]=0;
Sproduct[j]=0;
for(i=1; i<=4; i++)
{
Mproduct[i] +=Mvalue[i][j];
```

```
Sproduct[i]+= Svalue[i][j];
}
Mtotal=stotal=0;
for(i=1; i<=4; i++)
{
Mtotal+=Mweek[i];
Stotal+=Sweek[i];
}

printf("\n\n");
printf("Following is the list of things you can ");
printf("request for  enter appropriate item number and press RETURN key");
printf("1. Value matrices of production & sales");
printf("2. Total value of weekly production & sales");
printf("3. Product-wise monthly value of production& sales");
printf("4. Grand total value of production & sales");
printf("5.Exit");
number=0;
while(1)
{
printf("ENTER YOUR CHOICE:");
scanf("%d", &number);
printf("\n");
if(number= =5)
{
printf("GOOD BYE");
break;
}
switch(number)
{
case 1:
printf("VALUE MATRIX OF PRODUCTION");
for(i=1; i<=4; i++)
{
printf("Week (%d), i");
```

```
for(j=1; j<=5; j++)
printf("%7d", Mvalue);
printf("\n");
}
printf("VALUE MATRIX OF SALES");
for(i=1; i<=4; i++)
{
printf("Week(%d)", i);
for(j=1; j<=5; j++)
printf("%7d", Svalue[i][j]);
printf("\n");
}
break;
case 2:
printf("TOTAL WEEKLY PRODUCTION & SALES");
printf("                    PRODUCTION SALES");
printf("                    ----------      ----    ");
for(i=1; i<=4; i++)
{
printf("week(%d", i);
printf("%7d%7d", Mweek[i], Sweek[i]);
}
break;
case 3:
printf("PRODUCT WISE TOTAL PRODUCTION &  SALES");
printf("                    PRODUCTION SALES");
printf("                    ----------      ----    ");
for(j=1; j<=5; j++)
{
printf("Product(%d",ji);
printf("%7d%7d", Mproduct[j], Sproduct[j]);
}
break;
case 4:
printf("GRAND TOTAL OF PRODUCTION SALES");
```

```
printf("Total production=%d", Mtotal);
break;
default:
printf("Wrong choice, select gain");
break;
}
}
Printf("Exit from the program");
}
```

Output:

```
Enter products manufactured week wise
M11, M12, ----, M21, M22,------etc
11    15        12    14        13
13    13        14    15        12
12    16        10    15        14
14    11        15    13        12

Enter products sold week wise
S11,  S12, ----, S21,  S22,-----etc
10    13    9     12    11
12    10    12    14    10
11    14    10    14    12
12    10    13    11    10

Enter cost of each product
10 20 30 15 25
```

Following is the list of things you can request for enter appropriate item number and press RETURN key.

1. Value matrices of production & sales
2. Total value of weekly production & sales
3. Product-wise monthly value of production & sales
4. Grand total value of production & sales
5. Exit

```
Enter your choice: 1
VALUE MATRIX OF PRODUCTION
Week (1)    110    300    360    210    325
Week (2)    130    260    420    225    300
Week (3)    120    320    300    225    350
Week (4)    140    220    450    210    300

VALUE MATRIX OF SALES
Week (1)    100    260    270    180    275
Week (2)    120    200    360    210    250
Week (3)    110    280    300    210    300
Week (4)    120    200    390    165    250

Enter your choice: 2
TOTAL WEEKLY PRODUCTION & SALES
          PRODUCTION          SALES
        --------------------  ----------
Week(1)     1305                1085
Week(2)     1335                1140
Week(3)     1305                1200
Week(4)     1315                1125

Enter your choice: 3
PRODUCT WISE TOTAL PRODUCTION & SALES
              PRODUCTION          SALES
              -------------------  ---------
Product(1)        500                450
Product(2)       1100                450
Product(3)       1530                450
Product(4)        855                450
Product(5)       1275               1075

Enter your choice: 4
GRAND TOTAL OF PRODUCTION SALES
Total production=5260
```

```
Total sales=4550
ENTER YOUR CHOICE:5
GOOD BYE
Exit from the program
```

22. Write a program to read a series of words from a terminal using scanf function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char word1[40], word2[40], word3[40], word4[40];
printf("enter text:");
scanf("%s%s", word1, word2);
scanf("%s", word3);
scanf("%s", word4);
printf("\n");
printf("word1=%s\nword2=%s\n", word1, word2);
printf("word3=%s\nword4=%s\n", word3, word4");
}
```

Output:

```
Enter text:
Seventh Street, sakthinagar, erode

Word1=seventh
Word2=street
Word3=sakthinagar
Word4=erode
```

23. Program to read a line of text from terminal.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char line[81], character;
```

```
int c;
c=0;
printf("Enter text. Press<return> at end");
do
{
charcter=getchar();
line[c]=character;
c++;
}
while(character!='\n');
c=c-1;
line[c]='\0';
printf("\n%s\n", line);
}
```

Output:

```
Enter text.press<return> at end
Programming in c is interesting
Programming in c is interesting
Enter text. Press <Return> at end
```

24. Write a program to copy one string into another and count the number of characters copied.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char string1[80], string2[80];
int i;
printf("Enter a string\n");
printf("?");
scanf("%s, string2");
for(i=0; string2[i]!='\0'; i++)
string1[i]=string2[i];
string1[i]='\0';
printf("\n");
```

```
printf("%s\n", string1);
printf("number of charcters = %d\n", i);
}
```

Output:

```
Enter a string
? Manchester
Manchester
Number of characters=10

Enter a string
?westminister

Westiminister
Number of characters=11
```

25. Program for printing of the alphabet set in decimal and character form.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char c;
printf("\n\n");
for(c=65; c<=122; c=c+1)
{
if(c>90&&c<97)
continue;
printf("|%4d-%c", c,c);
}
printf("|n");
}
```

Output:

```
|65-A|66-B|67-C|68-D|69-E|70-F
```

```
|71-G|72-H|73-I|74-J|75-K|76-L
|77-M|78-N|79-O|80-P|81-Q|82-R
|83-S|84-T|85-U|86-V|87-W|88-X
|89-Y|90-Z|97-A|98-B|99-C|100-d
|101-e|102-f|103-g|104-h|105-i|106-j
|107-k|108-l|109-m|110-n|111-o|112-p
|113-q|114-r|115-s|116-t|117-u|118-v
|119-w|120-x|121-y|122-z|
```

26. Program to concatenation of strings.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i, j, k;
char first_name[10]={ANANDA};
char second_name[10]={MURUGAN};
char last_name[10] = {SELVARAJ};
char name[30];
for(i=0; first _name[i]!='\0'; i++)
name[i]=first_name[i];
name[i]=' ';
for(j=0; second _name[j]!='\0';j++)
name[i+j+1]=second_name[j];
name[i+j+1]=' ';
for(k=0; last _name[k]!='\0';k++)
name[i+j+k+2]='\0';
printf("\n\n");
printf("%s\n", name);
}
```

Output:

ANANDA MURUGAN SELVARAJ

27. Program to illustration of string handling functions.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
        char s1[20], s2[20], s3[20];
        int x, l1,l2,l3;
        printf("enter two string constants");
        printf("?");
        scanf("%s %s", s1,s2);
        x=strcmp(s1,s2);
        if(x!=0)
        {
printf("strings are not equal");
strcat(s1, s2);
}
else
printf("strings are equal");
strcpy(s3,s1);
l1=strlen(s1);
l2=strlen(s2);
l3=strlen(s3);
printf("s1=%s length =%d characters", s1,l1);
printf("s2=%s length =%d characters", s2,l2);
printf("s3=%s length =%d characters", s3,l3);
}
```

Output:

```
Enter two string constants
? ananda murugan

Strings are not equal
S1=ananda murugan  length=13 characters
```

```
S2=murugan          length=7 characters
S3=ananda murugan length=13 characters

Enter two string constants
? anand anand
Strings are equal
S1=anand length=5  characters
S2=anand length=5  characters
S3=anand length=5  characters
```

28. Write a program that would sort a list of names in alphabetical order.

```
#include<stdio.h>
#include<conio.h>
#define ITEMS 5
#define MAXCHAR 20
 void main()
{
        char string[ITEMS][MAXCHAR], dummy[MAXCHAR];
        int i=0, j=0;
        printf("enter names of %d items", ITEMS);
        while(i<ITEMS)
                scanf("%s", string[i++]);
        for(i=1; i<ITEMS; i++)
        {
        for(j=1; j<=ITEMS-i; j++)
        {
        if(strcmp(string[j-1), string[j]>0)
        {
        strcpy(dummy, string[j-1]);
        strcpy(string[j-1], string[j]);
        strcpy(string[j], dummy);
        }
        }
        }
```

```
printf("alphabetical list");
for(i=0; i<ITEMS; i++)
printf("%s", string[i]);
}
```

Output:

enter names of 5 times

Ananda
murugan renuka devi shri

alphabetical list

Ananda
devi
murugan
renuka
shri

29. Programs for counting of characters, words and lines in a text.

```
#include<stdio.h>
#include<conio.h>
void main()
{
char line[81], ctr;
int i, c, end=0, characters =0, words=0. lines=0;
printf("KEY IN THE TEXT");
printf("GIVE ONE SPACE AFTER EACH WORD WHEN COMPLETED, PRESS RETURN");
while(end= =0)
{
c=0;
while((ctr=getchar()) !='\n')
line[c++]=ctr;
line[c]='\0';
```

```
if(line[0] = ='\0')
break;
else
{
words++;
for(i=0; line[i]!='\0'; i++)
if(line[i] = =' ' || line[i] = = '\t')
words++;
}
lines=lines+1;
characters =characters + strlen(line);
}
printf("\n");
printf("number of lines=%d", lines);
printf("number of words=%d", words);
printf("number of characters=%d", characters);
}
```

Output:

```
KEY IN THE TEXT
GIVE ONE SPACE AFTER EACH WORD WHEN COMPLETED, PRESS RETURN

Admiration is a very short-lived passion.
Admiration involves a glorious obliquity of vision.
Always we like those who admire us but we do not like those whom we admire.
Fools admire, but men of sense approve

Number of lines=4
Number of words=36
Number of characters=205
```

30. Program to alphabetize a customer list.

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

```
#define CUSTOMERS 10
void main()
{
char first_name[20] [10], second_name[20][10], surname[20][10], name[20][20],
telephone[20][10], dummy[20];
int i,j;
printf("input names and telephone numbers");
printf("?");
for(i=0; i<CUSTOMERS; i++)
{
scanf("%s %s %s %s", first_name[i], second_name[i], surname[i], telephone[i]);
strcpy(name[i], surname[i]);
strcat(name[i], " ,");
dummy[0]=first_name[i][0];
dummy[1]='\0';
strcat(name[i], dummy);
strcat(name[i], ".");
dummy[0]=second_name[i][0];
dummy[1]='\0';
strcat(name[i], dummy);
}
for(i=1; i<=CUSTOMERS-1; i++)
for(j=1; j<=CUSTOMERS-i; j++)
if(strcmp(name[j-1], name[j])>0)
{
strcpy(dummy, name[j-1]);
strcpy(name[j-1], name[j]);
strcpy(name[j], dummy);
strcpy (dummy, telephone[j-1]);
strcpy(telephone[j-1], telephone[j]);
strcpy(telephone[j], dummy);
}
printf("CUSTOMERS LIST IN ALPHABETICAL ORDER");
```

```
for(i=0; i<CUSTOMERS; i++)
printf("%-20s\t%-10s", name[i], telephone[i]);
}
```

Output:

```
Input names and telephone numbers
? anandamurugan  9486153102
  Renukadevi  9486644542

CUSTOMERS LIST IN ALPHABETICAL ORDER

anandamurugan 9486153102
Renukadevi 9486644542
```

31. Program for functions with no arguments and no return values

```
#include<stdio.h>
#include<conio.h>
void printline(void);
void value (void);
void main()
{
printline();
value();
printline();
}
void printline(void)
{
int i;
for(i=1; i<=35; i++)
printf("%c",  '-');
printf("\n");
}
void value(void)
{
int year, period;
```

```
float inrate, sum, principal;
printf("principal amount?");
scanf("%f", &principal);
printf("interest rate?");
scanf("%f", &inrate);
printf("period?");
scanf("%d", &period);
sum=principal;
year=1;
while(year<=period)
{
sum=sum*(1+inrate);
year=year+1;
}
printf("\n%8.2f %5.2f %5d %12.2f\n", principal, inrate, period, sum);
}
```

Output:

```
Principal amount? 5000
Interest rate?    0.12
Period?                5

5000.00           0.12        5          8811.71
```

---

32. Program for functions with arguments but no return values.

```
#include<stdio.h>
#include<conio.h>
void printline(char c);
void value (float, float, int);
void main()
{
```

```
float principal, inrate;
int period;
printf("Enter principal amount, interest");
printf("rate and period");
scanf("%f %f %d", & principal, &inrate, &period);
printline('c');
}
void printline(char ch)
{
int i;
for(i=1; i<=52; i++)
printf("%c", ch);
printf("\n");
}
void value(float p, float r, int n)
{
int year;
float sum;
sum=p;
year=1;
while(year<=n)
{
sum=sum*(1+r);
year=year+1;
}
printf("%f\t%f\t%d\t%f\n", p, r,n,sum);
}
```

Output:

```
Enter principal amount, interest rate, and period
5000 0.12  5
ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
5000.000000          0.120000      5        8811.708984
ccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

33. Program for functions with arguments and return values.

```
#include<stdio.h>
#include<conio.h>
void printline(char ch, int len);
value (float, float, int);
void main()
{
float principal, inrate, amount;
int period;
printf("enter principal amount, interest");
printf("rate, and period");
scanf("%f%f%d", &principal, &inrate, &period);
printline('*', 52);
amount=value(principal, inrate, period);
printf("\n%f\t%f\t%f\t%d\t%f\n\n", principal, inrate, period, amount);
printline();
}
void printline(char ch, int len)
{
int i;
for(i=1; i<=len; i++)
printf("%c",ch);
printf("\n");
}
value(float p, float r, int n)
{
int year;
float sum;
sum=p;
year=1;
while(year<=n)
{
Sum=sum*(1+r);
```

```
year=year+1;
}
return(sum);
}
```

Output:

```
Enter principal amount, interest rate, and period
5000  0.12       5
****************************************************************************
5000.000000                    0.1200000    5        8811.00000
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = =
```

34. Write a function power that computes x raised to the power y for integers x and y and returns double-type value.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x,y;
double power (int, int);
printf("enter x,y: ");
scanf("%d%d",  &x,&y);
printf("%d to power %d is %f", x,y, power (x,y));
}
double power (int x, int y);
{
P=1.0;
if(y>=0)
while(y--)
p*=x;
else
while(y++)
p/=x;
return(p);
}
```

Output:

Enter x,y: 16 2
16 to power 2 is 256.000000

Enter x,y : 16 -2
16 to power -2 is 0.003906

35. Write a program to show how user-defined function is called.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x=1, y=2, z;
z=add(x,y);
printf("z=%d", z);
}
add(a,b);
{
return(a+b);
}
```

Output:

z=3

36. Write a program to define user-defined function. Call them at different places.

```
#include<stdio.h>
#include<conio.h>
void y();
voidy()
{
printf("Y");
void main()
{
```

```
void a(), b(), c(), d();
return();
clrscr();
y();
a();
b();
c();
d();
}
void a()
{
printf("A");
y();
}
void b()
{
printf("B");
a();
}
void c()
{
a();
b();
printf("C");
}
void d()
{
printf("D");
c();
b();
a();
}
```

Output:

YAYBAYAYBAYCDAYBAYCBAYAY

37. Write a program to show how similar variable names can be used in different functions.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int b=10, c=5;
return();
printf("In main() B=%d  C=%d", b,c);
fun();
}
fun()
{
int b=20, c=10;
printf("In fun() B=%d c=%d", b,c);
}
```

Output:

```
In main() B=10 c=5
In fun()  B=20 c=10
```

38. Write a program to show the effect of global variables on different functions.

```
#include<stdio.h>
#include<conio.h>
int b=10, c=5;
void main()
{
clrscr();
printf("In main () B=%d c=%d", b,c);
fun();
b++;
c--;
printf("Again In main () B=%d c=%d", b,c);
}
fun()
```

```
{
b++;
c--;
printf(" In fun () B=%d c=%d", b,c);
}
```

Output:

```
In main() B=10 c=5
In fun()  B=11  c=4
Again In main() B=12 c=3
```

39. Write a program to display message using user-defined function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
void message();
message();
}
void message()
{
puts ("Have a nice day");
}
```

Output:

```
Have a nice day
```

40. Write a program to return more than one value from user defined function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x,y, add, sub, change(int*, int*, int*, int*);
clrscr();
```

```
printf("Enter values of X and Y:");
scanf("%d %d", &x, &y);
change(&x &y, &add, &sub);
printf("Addition:%d", add);
printf("Subtraction:%d", sub);
return 0;
}
change(int * a, int*b, int*c, int d)
{
c=*a+*b;
*d=*a-*b;
}
```

Output:

```
Enter values of x & y: 5  4
Addition: 9
Subtraction:1
```

41. Write a program to pass arguments to user-defined function by value and by reference.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int k, m, other(int, int*);
clrscr();
printf("Address of k&m in main(): %u%u", &k&m);
other(k&m);
return 0;
}
other(int k, int*m)
{
printf("Address of k&m in other(): %u%u, &k,m");
}
```

Output:

Address of k&m in main(): 65524 65522

Address of k&m in other(): 65518 65522

42. Write a program to return only absolute value like abs() function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int uabs(int), x;
clrscr();
printf("enter a negative value:");
scanf("%d", x);
x=uabs(x);
printf("X=%d", x);
return 0;
}
uabs(int y)
{
if(y<0)
return(y*-1);
else
return(y);
}
```

Output:

Enter a negative value: -5

X=5

43. Write a program to calculate square and cube of an entered number. Use function as an argument.

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

```
int m;
clrscr();
printf("Cube: %d", cube(sqr(input())));
}
input()
{
int k;
printf("Number:");
scanf("%d", &k);
return k;
}
sqr(m)
{
printf("Square:%d", m*m);
return m;
}
cube(m)
{
return m*m*m;
}
```

Output:

```
Number:2
Square: 4
Cube:8
```

44. Write a program to assign return value of a function to another variable.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int input(int);
int x;
clrscr();
x=input(x);
```

```
printf("x=%d", x);
}
input(int k)
{
printf("Enter value of x=");
scanf("%d", &k);
return(k);
}
```

Output:

```
Enter value of x=5
x=5
```

45. Write a program to perform addition and subtraction of numbers with return value of function.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int input(int);
int sqr(int);
int x;
clrscr();
x=sqr(1-input(x)+1);
printf("Square=%d, x");
}
input(int k)
{
printf("Enter value of x=");
scanf("%d, &k");
return(k);
{
sqr(int m)
{
```

```
return (pow(m,2));
}
```

Output:

```
Enter value of x=5
Square=9
```

46. Write a program to perform multiplication and division of numbers with return value of function.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int input(int);
int sqr(int);
int x;
clrscr();
x=sqr(5*input(x)/2);
printf("Square=%d, x");
}
input(int k)
{
printf("Enter value of x=");
scanf("%d, &k");
return(k);
{
sqr(int m)
{
return (pow(m,2));
}
```

Output:

```
Enter value of x=5
Square=144
```

47. Write a program to use (++) operator with return value of function.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int input(int);
int sqr(int);
irit x,y =0;
clrscr();
x=sqr(++(y-(input(x))));
printf("Square=%d", x);
}
input(int k)
{
printf("Enter value of x=");
scanf("%d", &k);
return(k);
}
sqr(int m)
{
return (pow(m,2));
}
```

Output:

```
Enter value of x=7
Square=64
```

48. Write a program to use mod(%) with function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int j();
if(j()%2= =0)
```

```
printf("Number is even");
else
printf("Number is odd");
return 0;
}
j()
{
int x;
clrscr();
printf("Enter a number");
scanf("%d", &x);
return(x);
}
```

Output:

```
Enter a number:5
Number is odd
```

49. Write a program to evaluate the equation s=sqr(a()+b()) using function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int s=0, a(), b(),sqr(int);
clrscr();
s=sqr(a()+b());
printf("square of sum=%d", s);
return 0;
}
a()
{
int a;
printf("Enter value of a:");
scanf("%d", &a);
return(a);
```

```
}
b()
{
int b;
printf("Enter value of b:");
scanf("%d", &b);
return(b);
}
sqr(int x)
{
return(x*x);
}
```

Output:

```
Enter value of a:5
Enter value of b: 3
Square of sum=64
```

50. Write a program to call user-defined function through if statement.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a();
clrscr();
if(a()%2= =0)
printf("The number is even");
else
printf("The number is odd");
}
a()
{
int a;
printf("Enter value of a:");
```

```
scanf("%d",&a);
return(a);
}
```

Output:

Enter value of a:5
The number is odd

51. Write a program to call user-defined function through switch() statement.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<ctype.h>
void main()
{
int a();
int x=5;
clrscr();
switch(a())
{
case's':
printf("Square of %d is %d", x, pow(x,2));
break;
case'c':
printf("Cube of %d is %d", x, pow(x,3));
break;
case'd':
printf("Double of %d is %d", x, x*2);
break;
default:
printf("Unexpected choice printed as it is : %d", x);
}
}
a()
{
char c='c';
```

```
printf("Enter your choice square(s), cube(c), double(d):");
c=getche();
c=tolower(c);
return(c);
}
```

Output:

```
Enter your choice square(s), cube(c), double (d):D
Double of 5 is 10
```

52. Write a program to call function through the for loop.

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
int plus(int), m=1;
clrscr();
for(; plus(m); m++)
{
printf("%3d", m);
}
}
plus (int k)
{
if(k= =10)
{
exit(1);
return NULL;
}
else
return(k);
}
```

Output:

```
1 2 3 4 5 6 7 8 9
```

Chapter 4 C DEBUGGING

Find the output or error(s) for the following Programs:

1. void main()

```
{
    int  const * p=5;
    printf("%d",++(*p));
}
```

Answer:

Compiler error: Cannot modify a constant value.

Explanation:

p is a pointer to a "constant integer". But we tried to change the value of the "constant integer".

2. main()

```
{
    char s[ ]="man";
    int i;
    for(i=0;s[ i ];i++)
    printf("\n%c%c%c%c",s[ i ],*(s+i),*(i+s),i[s]);
}
```

Answer:

```
mmmm
aaaa
nnnn
```

Explanation:

s[i], *(i+s), *(s+i), i[s] are all different ways of expressing the same idea. Generally array name is the base address for that array. Here **s** is the base address. **i** is the index number/displacement

from the base address. So, indirecting it with * is same as s[i]. i[s] may be surprising. But in the case of C it is same as s[i].

3. main()

```
{
    float me = 1.1;
    double you = 1.1;
    if(me==you)
    printf("I love U");
    else
    printf("I hate U");
}
```

Answer:

I hate U

Explanation:

For floating point numbers (float, double, long double) the values cannot be predicted exactly. Depending on the number of bytes, the precession with of the value represented varies. Float takes 4 bytes and long double takes 10 bytes. So float stores 0.9 with less precision than long double.

Rule of Thumb:

Never compare or at-least be cautious when using floating point numbers with relational operators (== , >, <, <=, >=,!=) .

4. main()

```
    {
    static int var = 5;
    printf("%d ",var--);
    if(var)
        main();
    }
```

Answer:

5 4 3 2 1

Explanation:

When *static* storage class is given, it is initialized once. The change in the value of a static variable is retained even between the function calls. Main is also treated like any other ordinary function, which can be called recursively.

5. main()

```
{
    int c[ ]={2.8,3.4,4,6.7,5};
```

```
    int j,*p=c,*q=c;
    for(j=0;j<5;j++) {
        printf(" %d ",*c);
        ++q;     }
    for(j=0;j<5;j++){
        printf(" %d ",*p);
        ++p;     }
}
```

Answer:

2 2 2 2 2 2 3 4 6 5

Explanation:

Initially pointer c is assigned to both **p** and **q.** In the first loop, since only **q** is incremented and not c , the value 2 will be printed 5 times. In second loop **p** itself is incremented. So the values 2 3 4 6 5 will be printed.

6.
```
main()
{
    extern int i;
    i=20;
    printf("%d",i);
}
```

Answer:

Linker Error : Undefined symbol '_i'

Explanation:

extern storage class in the following declaration,

extern int i;

specifies to the compiler that the memory for **i** is allocated in some other program and that address will be given to the current program at the time of linking. But linker finds that no other variable of name **i** is available in any other program with memory space allocated for it. Hence a linker error has occurred.

7.
```
main()
{
    int i=-1,j=-1,k=0,l=2,m;
    m=i++&&j++&&k++||l++;
    printf("%d %d %d %d",i,j,k,l,m);
```

```
}
```

Answer:

0 0 1 3 1

Explanation:

Logical operations always give a result of **1 or 0.** And also the logical AND (&&) operator has higher priority over the logical OR (||) operator. So the expression **'i++ && j++ && k++'** is executed first. The result of this expression is 0(-1 && -1 && 0 = 0). Now the expression is 0 || 2 which evaluates to 1 (because OR operator always gives 1 except for '0 || 0' combination- for which it gives 0). So the value of m is 1. The values of other variables are also incremented by 1.

8.
```
main()
{
    char *p;
    printf("%d %d ",sizeof(*p),sizeof(p));
}
```

Answer:

1 2

Explanation:

The sizeof() operator gives the number of bytes taken by its operand. P is a character pointer, which needs one byte for storing its value (a character). Hence sizeof(*p) gives a value of 1. Since it needs two bytes to store the address of the character pointer sizeof(p) gives 2.

9.
```
main()
{
    int i=3;
    switch(i)
     {
       default:printf("zero");
        case 1: printf("one");
                break;
       case 2:printf("two");
              break;
       case 3: printf("three");
              break;
      }
}
```

Answer:

three

Explanation:

The default case can be placed anywhere inside the loop. It is executed only when all other cases doesn't match.

10. main()

```
{
    printf("%x",-1<<4);
}
```

Answer:

fff0

Explanation :

-1 is internally represented as all 1's. When left shifted four times the least significant 4 bits are filled with 0's. The %x format specifier specifies that the integer value be printed as a hexadecimal value.

11. main()

```
{
    char string[]="Hello World";
    display(string);
}
void display(char *string)
{
    printf("%s",string);
}
```

Answer:

Compiler Error : Type mismatch in redeclaration of function display

Explanation:

In third line, when the function **display** is encountered, the compiler doesn't know anything about the function display. It assumes the arguments and return types to be integers, (which is the default type). When it sees the actual function **display,** the arguments and type contradicts with what it has assumed previously. Hence a compile time error occurs.

12. main()

```
{
    int c=- -2;
```

```
    printf("c=%d",c);
}
```

Answer:

c=2;

Explanation:

Here unary minus (or negation) operator is used twice. Same maths rules applies, *ie.*, minus * minus= plus.

Note:

However you cannot give like --2. Because -- operator can only be applied to variables as a **decrement** operator (*eg.*, i--). 2 is a constant and not a variable.

13.
```
#define int char
main()
{
    int i=65;
    printf("sizeof(i)=%d",sizeof(i));
}
```

Answer:

sizeof(i)=1

Explanation:

Since the #define replaces the string **int** by the macro **char**

14.
```
main()
{
    int i=10;
    i=!i>14;
    printf ("i=%d",i);
}
```

Answer:

i=0

Explanation:

In the expression **!i>14,** NOT (!) operator has more precedence than ' >' symbol. ! is a unary logical operator. !i (!10) is 0 (not of true is false). 0>14 is false (zero).

15.
```
#include<stdio.h>
main()
{
```

```
char s[]={'a','b','c','\n','c','\0'};
char *p,*str,*str1;
p=&s[3];
str=p;
str1=s;
printf("%d",++*p + ++*str1-32);
}
```

Answer:

77

Explanation:

p is pointing to character '\n'. str1 is pointing to character 'a' ++*p. "p is pointing to '\n' and that is incremented by one." the ASCII value of '\n' is 10, which is then incremented to 11. The value of ++*p is 11. ++*str1, str1 is pointing to 'a' that is incremented by 1 and it becomes 'b'. ASCII value of 'b' is 98.

Now performing (11 + 98 – 32), we get 77("M");

So we get the output 77 :: "M" (ASCII is 77).

16.
```
#include<stdio.h>
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);
}
```

Answer:

SomeGarbageValue---1

Explanation:

p=&a[2][2][2] you declare only two 2D arrays, but you are trying to access the third 2D(which you are not declared) it will print garbage values. *q=***a starting address of a is assigned integer pointer. Now q is pointing to starting address of a. If you print *q, it will print first element of 3D array.

17.
```
#include<stdio.h>
main()
{
```

```
    struct xx
    {
        int x=3;
        char name[]="hello";
     };
    struct xx *s;
    printf("%d",s->x);
    printf("%s",s->name);
}
```

Answer:

Compiler Error

Explanation:

You should not initialize variables in declaration.

18.

```
#include<stdio.h>
main()
{
    struct xx
    {
        int x;
        struct yy
        {
            char s;
            struct xx *p;
        };
        struct yy *q;
    };
}
```

Answer:

Compiler Error

Explanation:

The structure yy is nested within structure xx. Hence, the elements are of yy are to be accessed through the instance of structure xx, which needs an instance of yy to be known. If the instance is created after defining the structure the compiler will not know about the instance relative to xx. Hence for nested structure yy you have to declare member.

19. main()

```
{
    printf("\nab");
    printf("\bsi");
    printf("\rha");
}
```

Answer:

hai

Explanation:

\n - newline

\b - backspace

\r - linefeed

20. main()

```
{
    int i=5;
    printf("%d%d%d%d%d%d",i++,i--,++i,--i,i);
}
```

Answer:

45545

Explanation:

The arguments in a function call are pushed into the stack from left to right. The evaluation is by popping out from the stack and the evaluation is from right to left, hence the result.

21. #define square(x) x*x

```
main()
{
int i;
i = 64/square(4);
printf("%d",i);
}
```

Answer:

64

Explanation:

The macro call square(4) will substituted by 4*4 so the expression becomes i = 64/4*4. Since / and * has equal priority the expression will be evaluated as (64/4)*4 *i.e.,* 16*4 = 64.

22.
```
main()
{
    char *p="hai friends",*p1;
    p1=p;
    while(*p!='\0') ++*p++;
    printf("%s   %s",p,p1);
}
```

Answer:

ibj!gsjfoet

Explanation:

++*p++ will be parse in the given order

➢ *p that is value at the location currently pointed by p will be taken

➢ ++*p the retrieved value will be incremented

➢ when ; is encountered the location will be incremented that is p++ will be executed

Hence, in the while loop initial value pointed by p is 'h', which is changed to 'i' by executing ++*p and pointer moves to point, 'a' which is similarly changed to 'b' and so on. Similarly blank space is converted to '!'. Thus, we obtain value in p becomes "ibj!gsjfoet" and since p reaches '\0' and p1 points to p thus p1does not print anything.

23.
```
#include <stdio.h>
#define a 10
main()
{
    #define a 50
    printf("%d",a);
}
```

Answer:

50

Explanation:

The preprocessor directives can be redefined anywhere in the program. So the most recently assigned value will be taken.

24.
```
#define clrscr() 100
main()
{
```

```
    clrscr();
    printf("%d\n",clrscr());
}
```

Answer:

100

Explanation:

Preprocessor executes as a seperate pass before the execution of the compiler. So textual replacement of clrscr() to 100 occurs. The input program to compiler looks like this :

```
main()
{
    100;
    printf("%d\n",100);
}
```

Note:

100; is an executable statement but with no action. So it doesn't give any problem.

25.

```
main()
{
printf("%p",main);
}
```

Answer:

Some address will be printed.

Explanation:

Function names are just addresses (just like array names are addresses).

main() is also a function. So the address of function main will be printed. %p in printf specifies that the argument is an address. They are printed as hexadecimal numbers.

26.

```
main()
{
clrscr();
}
clrscr();
```

Answer:

No output/error

Explanation:

The first clrscr() occurs inside a function. So it becomes a function call. In the second clrscr(); is a function declaration (because it is not inside any function).

27.
```
enum colors {BLACK,BLUE,GREEN}
 main()
{
 printf("%d..%d..%d",BLACK,BLUE,GREEN);
 return(1);
}
```
Answer:

0..1..2

Explanation:

enum assigns numbers starting from 0, if not explicitly defined.

28.
```
void main()
{
 char far *farther,*farthest;
 printf("%d..%d",sizeof(farther),sizeof(farthest));
 }
```
Answer:

4..2

Explanation:

The second pointer is of char type and not a far pointer.

29.
```
main()
{
 int i=400,j=300;
 printf("%d..%d");
}
```
Answer:

400..300

Explanation:

printf takes the values of the first two assignments of the program. Any number of printf's may be given. All of them take only the first two values. If more number of assignments given in the program, then printf will take garbage values.

30. main()

```
{
 char *p;
p="Hello";
 printf("%c\n",*&*p);
}
```

Answer:

H

Explanation:

* is a dereference operator & is a reference operator. They can be applied any number of times provided it is meaningful. Here p points to the first character in the string "Hello". *p dereferences it and so its value is H. Again & references it to an address and * dereferences it to the value H.

31. main()

```
{
   int i=1;
   while (i<=5)
    {
      printf("%d",i);
      if (i>2)
        goto here;
      i++;
    }
}
fun()
{
  here:
    printf("PP");
}
```

Answer:

Compiler error: Undefined label 'here' in function main.

Explanation:

Labels have functions scope, in other words the scope of the labels is limited to functions. The label 'here' is available in function fun() hence it is not visible in function main.

32. main()

```
{
   static char names[5][20]={“pascal”,“ada”,“cobol”,“fortran”,“perl”};
   int i;
    char *t;
    t=names[3];
    names[3]=names[4];
    names[4]=t;
   for (i=0;i<=4;i++)
      printf(“%s”,names[i]);
}
```

Answer:

Compiler error: Lvalue required in function main

Explanation:

Array names are pointer constants. So it cannot be modified.

33. void main()

```
{
      int i=5;
      printf(“%d”,i++ + ++i);
}
```

Answer:

Output Cannot be predicted exactly.

Explanation:

Side effects are involved in the evaluation of i.

34. void main()

```
{
      int i=5;
      printf(“%d”,i+++++i);
}
```

Answer:

Compiler Error

Explanation:

The expression i+++++i is parsed as i ++ ++ + i which is an illegal combination of operators.

35. #include<stdio.h>
main()
{
int i=1,j=2;
switch(i)
{
case 1: printf("GOOD");
break;
case j: printf("BAD");
break;
}
}

Answer:

Compiler Error: Constant expression required in function main.

Explanation:

The case statement can have only constant expressions (this implies that we cannot use variable names directly so an error).

Note:

Enumerated types can be used in case statements.

36. main()
{
int i;
printf("%d",scanf("%d",&i)); // value 10 is given as input here
}

Answer:

1

Explanation:

Scanf returns number of items successfully read and not 1/0. Here 10 is given as input which should have been scanned successfully. So number of items read is 1.

37. #define f(g,g2) g##g2
main()
{
int var12=100;

```
printf("%d",f(var,12));
}
```

Answer:

100

38.

```
main()
{
int i=0;
for(;i++;printf("%d",i)) ;
    printf("%d",i);
}
```

Answer:

1

Explanation:

Before entering into the for loop the checking condition is "evaluated". Here it evaluates to 0 (false) and comes out of the loop, and i is incremented (note the semicolon after the for loop).

39.

```
#include<stdio.h>
main()
{
  char s[]={'a','b','c','\n','c','\0'};
 char *p,*str,*str1;
  p=&s[3];
  str=p;
  str1=s;
 printf("%d",++*p + ++*str1-32);
}
```

Answer:

M

Explanation:

p is pointing to character '\n'.str1 is pointing to character 'a' ++*p meAnswer: "p is pointing to '\n' and that is incremented by one." the ASCII value of '\n' is 10. Then it is incremented to 11. The value of ++*p is 11. ++*str1 meAnswer:"str1 is pointing to 'a' that is incremented by 1 and it becomes 'b'. ASCII value of 'b' is 98. both 11 and 98 is added and result is subtracted from 32.

i.e., (11+98-32)=77("M");

40.
```
#include<stdio.h>
main()
{
  struct xx
   {
     int x=3;
      char name[]="hello";
   }
struct xx *s=malloc(sizeof(struct xx));
printf("%d",s->x);
printf("%s",s->name);
}
```

Answer:

Compiler Error

Explanation:

Initialization should not be done for structure members inside the structure declaration

41.
```
#include<stdio.h>
main()
{
struct xx
 {
        int x;
        struct yy
         {
           char s;
           struct xx *p;
         }
        struct yy *q;
        }
       }
```

Answer:

Compiler Error

Explanation:

In the end of nested structure yy a member have to be declared.

42. main()

```
{
 extern int i;
 i=20;
 printf("%d",sizeof(i));
}
```

Answer:

Linker error: undefined symbol '_i'.

Explanation:

extern declaration specifies that the variable i is defined somewhere else. The compiler passes the external variable to be resolved by the linker. So compiler doesn't find an error. During linking the linker searches for the definition of i. Since it is not found the linker flags an error.

43. main()

```
{
printf("%d", out);
}
int out=100;
```

Answer:

Compiler error: undefined symbol out in function main.

Explanation:

The rule is that a variable is available for use from the point of declaration. Even though a is a global variable, it is not available for main. Hence an error.

44. main()

```
{
 extern out;
 printf("%d", out);
}
 int out=100;
```

Answer:

100

Explanation:

This is the correct way of writing the previous program.

45. main()

```
{
show();
}
void show()
{
 printf("I'm the greatest");
}
```

Answer:

Compier error: Type mismatch in redeclaration of show.

Explanation:

When the compiler sees the function show it doesn't know anything about it. So the default return type (ie, int) is assumed. But when compiler sees the actual definition of show mismatch occurs since it is declared as void. Hence the error.

The solutions are as follows:

1. declare void show() in main().
2. define show() before main().
3. declare extern void show() before the use of show().

46. main( )

```
{
 int a[2][3][2] = {{{2,4},{7,8},{3,4}},{{2,2},{2,3},{3,4}}};
  printf("%u %u %u %d \n",a,*a,**a,***a);
  printf("%u %u %u %d \n",a+1,*a+1,**a+1,***a+1);
      }
```

Answer:

100, 100, 100, 2

114, 104, 102, 3

Explanation:

The given array is a 3-D one. It can also be viewed as a 1-D array.

2	4	7	8	3	4	2	2	2	3	3	4
100	102	104	106	108	110	112	114	116	118	120	122

thus, for the first printf statement a, *a, **a give address of first element since the indirection ***a gives the value. Hence, the first line of the output.

for the second printf a+1 increases in the third dimension thus points to value at 114, *a+1 increments in second dimension thus points to 104, **a +1 increments the first dimension thus points to 102 and ***a+1 first gets the value at first location and then increments it by 1. Hence, the output.

47.
```
main( )
{
 int a[ ] = {10,20,30,40,50},j,*p;
 for(j=0; j<5; j++)
   {
     printf("%d" ,*a);
     a++;
   }
   p = a;
  for(j=0; j<5; j++)
     {
            printf("%d " ,*p);
            p++;
     }
 }
```

Answer:

Compiler error: lvalue required.

Explanation:

Error is in line with statement a++. The operand must be an lvalue and may be of any of scalar type for the any operator, array name only when subscripted is an lvalue. Simply array name is a non-modifiable lvalue.

48.
```
main( )
{
 static int  a[ ]  = {0,1,2,3,4};
 int  *p[ ] = {a,a+1,a+2,a+3,a+4};
 int  **ptr =  p;
 ptr++;
 printf("\n %d  %d  %d", ptr-p, *ptr-a, **ptr);
 *ptr++;
 printf("\n %d  %d  %d", ptr-p, *ptr-a, **ptr);
```

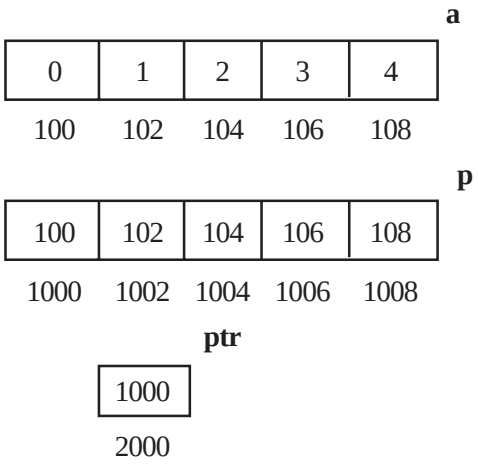
```
*++ptr;
printf("\n %d  %d  %d", ptr-p, *ptr-a, **ptr);
++*ptr;
 printf("\n %d  %d  %d", ptr-p, *ptr-a, **ptr);
}
```

Answer:

111
222
333
444

Explanation:

Let us consider the array and the two pointers with some address



After execution of the instruction ptr++ value in ptr becomes 1002, if scaling factor for integer is 2 bytes. Now ptr – p is value in ptr – starting location of array p, (1002 – 1000) / (scaling factor) = 1, *ptr – a = value at address pointed by ptr – starting value of array a, 1002 has a value 102 so the value is (102 – 100)/(scaling factor) = 1, **ptr is the value stored in the location pointed by the pointer of ptr = value pointed by value pointed by 1002 = value pointed by 102 = 1. Hence the output of the firs printf is 1, 1, 1.

After execution of *ptr++ increments value of the value in ptr by scaling factor, so it becomes1004. Hence, the outputs for the second printf are ptr – p = 2, *ptr – a = 2, **ptr = 2.

After execution of *++ptr increments value of the value in ptr by scaling factor, so it becomes1004. Hence, the outputs for the third printf are ptr – p = 3, *ptr – a = 3, **ptr = 3.

After execution of ++*ptr value in ptr remains the same, the value pointed by the value is incremented by the scaling factor. So the value in array p at location 1006 changes from 106 10 108. Hence, the outputs for the fourth printf are ptr – p = 1006 – 1000 = 3, *ptr – a = 108 – 100 = 4, **ptr = 4.

49.
```
main( )
{
 char  *q;
 int  j;
 for (j=0; j<3; j++) scanf("%s" ,(q+j));
 for (j=0; j<3; j++) printf("%c" ,*(q+j));
 for (j=0; j<3; j++) printf("%s" ,(q+j));
}
```

Explanation:

Here we have only one pointer to type char and since we take input in the same pointer thus we keep writing over in the same location, each time shifting the pointer value by 1. Suppose the inputs are MOUSE, TRACK and VIRTUAL. Then for the first input suppose the pointer starts at location 100 then the input one is stored as

M	O	U	S	E	\0

When the second input is given the pointer is incremented as j value becomes 1, so the input is filled in memory starting from 101.

M	T	R	A	C	K	\0

The third input starts filling from the location 102

M	T	V	I	R	T	U	A	L	\0

This is the final value stored.

The first printf prints the values at the position q, q+1 and q+2 = M T V

The second printf prints three strings starting from locations q, q+1, q+2

i.e., MTVIRTUAL, TVIRTUAL and VIRTUAL.

50.
```
main( )
{
 void *vp;
 char ch = 'g', *cp = "goofy";
 int j = 20;
 vp = &ch;
 printf("%c", *(char *)vp);
 vp = &j;
 printf("%d",*(int *)vp);
 vp = cp;
```

```
 printf("%s",(char *)vp + 3);
}
```

Answer:

g20fy

Explanation:

Since a void pointer is used it can be type casted to any other type pointer. vp = &ch stores address of char ch and the next statement prints the value stored in vp after type casting it to the proper data type pointer. the output is 'g'. Similarly the output from second printf is '20'. The third printf statement type casts it to print the string from the 4^{th} value hence the output is 'fy'.

51.

```
main ( )
{
 static char *s[ ]  = {"black", "white", "yellow", "violet"};
 char **ptr[ ] = {s+3, s+2, s+1, s}, ***p;
 p = ptr;
 **++p;
 printf("%s",*--*++p + 3);
}
```

Answer:

ck

Explanation:

In this problem we have an array of char pointers pointing to start of 4 strings. Then we have ptr which is a pointer to a pointer of type char and a variable p which is a pointer to a pointer of type char p hold the initial value of ptr, *i.e.*, p = s+3. The next statement increment value in p by 1, thus now value of p = s+2. In the printf statement the expression is evaluated *++p causes gets value s+1 then the pre decrement is executed and we get s+1 – 1 = s the indirection operator now gets the value from the array of s and adds 3 to the starting address. The string is printed starting from this position. Thus, the output is 'ck'.

52.

```
main()
{
 int  i, n;
 char *x = "girl";
 n = strlen(x);
 *x = x[n];
 for(i=0; i<n; ++i)
```

```
  {
printf("%s\n",x);
x++;
  }
 }
```

Answer:

(blank space)

irl

rl

l

Explanation:

Here a string (a pointer to char) is initialized with a value "girl". The strlen function returns the length of the string, thus n has a value 4. The next statement assigns value at the nth location ('\0') to the first location. Now the string becomes "\0irl" . Now the printf statement prints the string after each iteration it increments it starting position. Loop starts from 0 to 4. The first time x[0] = '\0' hence it prints nothing and pointer value is incremented. The second time it prints from x[1] *i.e.*, "irl" and the third time it prints "rl" and the last time it prints "l" and the loop terminates.

53.
```
int i,j;
    for(i=0;i<=10;i++)
    {
    j+=5;
    assert(i<5);
    }
```

Answer:

Runtime error: Abnormal program termination.

assert failed (i<5), <file name>,<line number>

Explanation:

Asserts are used during debugging to make sure that certain conditions are satisfied. If assertion fails, the program will terminate reporting the same. After debugging use, #undef NDEBUG and this will disable all the assertions from the source code. Assertion is a good debugging tool to make use of.

54.
```
main()
{
int i=-1;
+i;
```

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

Answer:

i = -1, +i = -1

Explanation:

Unary + is the only dummy operator in C. Where-ever it comes you can just ignore it just because it has no effect in the expressions (hence the name dummy operator).

55. What are the files which are automatically opened when a C file is executed?

Answer:

stdin, stdout, stderr (standard input,standard output,standard error).

56. What will be the position of the file marker?

a: fseek(ptr,0,SEEK_SET);

b: fseek(ptr,0,SEEK_CUR);

Answer:

a: The SEEK_SET sets the file position marker to the starting of the file.

b: The SEEK_CUR sets the file position marker to the current position of the file.

57.

```
main()
{
char name[10],s[12];
scanf(" \"%[^\"]\"",s);
}
```

How scanf will execute?

Answer:

First it checks for the leading white space and discards it.Then it matches with a quotation mark and then it reads all character upto another quotation mark.

58. What is the problem with the following code segment?

```
while ((fgets(receiving array,50,file_ptr)) != EOF);
```

Answer & Explanation:

fgets returns a pointer. So the correct end of file check is checking for != NULL.

59. main()

```
{
main();
}
```

Answer:

Runtime error : Stack overflow.

Explanation:

main function calls itself again and again. Each time the function is called its return address is stored in the call stack. Since there is no condition to terminate the function call, the call stack overflows at runtime. So it terminates the program and results in an error.

60. main()

```
{
char *cptr,c;
void *vptr,v;
c=10;  v=0;
cptr=&c;  vptr=&v;
printf("%c%v",c,v);
}
```

Answer:

Compiler error (at line number 4): size of v is Unknown.

Explanation:

You can create a variable of type void * but not of type void, since void is an empty type. In the second line you are creating variable vptr of type void * and v of type void hence an error.

61. main()

```
{
char *str1="abcd";
char str2[]="abcd";
printf("%d %d %d",sizeof(str1),sizeof(str2),sizeof("abcd"));
}
```

Answer:

2 5 5

Explanation:

In first sizeof, str1 is a character pointer so it gives you the size of the pointer variable. In second sizeof the name str2 indicates the name of the array whose size is 5 (including the '\0' termination character). The third sizeof is similar to the second one.

62.
```
main()
{
char not;
not=!2;
printf("%d",not);
}
```

Answer:

0

Explanation:

! is a logical operator. In C the value 0 is considered to be the boolean value FALSE and any non-zero value is considered to be the boolean value TRUE. Here 2 is a non-zero value so TRUE. !TRUE is FALSE (0) so it prints 0.

63.
```
#define FALSE -1
#define TRUE   1
#define NULL   0
main() {
  if(NULL)
     puts("NULL");
  else if(FALSE)
     puts("TRUE");
  else
     puts("FALSE");
  }
```

Answer:

TRUE

Explanation:

The input program to the compiler after processing by the preprocessor is,

```
main(){
if(0)
        puts("NULL");
```

```
else if(-1)
        puts("TRUE");
else
        puts("FALSE");
}
```

Preprocessor doesn't replace the values given inside the double quotes. The check by if condition is boolean value false so it goes to else. In second if -1 is boolean value true hence "TRUE" is printed.

64.
```
main()
{
int k=1;
printf("%d==1 is ""%s",k,k==1?"TRUE":"FALSE");
}
```

Answer:

1==1 is TRUE

Explanation:

When two strings are placed together (or separated by white-space) they are concatenated (this is called as "stringization" operation). So the string is as if it is given as "%d==1 is %s". The conditional operator(?:) evaluates to "TRUE".

65.
```
main()
{
int y;
scanf("%d",&y); // input given is 2000
if( (y%4==0 && y%100 != 0) || y%100 == 0)
      printf("%d is a leap year");
else
    printf("%d is not a leap year");
}
```

Answer:

2000 is a leap year.

Explanation:

An ordinary program to check if leap year or not.

66.
```
#define max 5
#define int arr1[max]
main()
{
typedef char arr2[max];
arr1 list={0,1,2,3,4};
arr2 name="name";
printf("%d %s",list[0],name);
}
```

Answer:

Compiler error (in the line arr1 list = {0,1,2,3,4})

Explanation:

arr2 is declared of type array of size 5 of characters. So it can be used to declare the variable name of the type arr2. But it is not the case of arr1. Hence an error.

Rule of Thumb:

#defines are used for textual replacement whereas typedefs are used for declaring new types.

67.
```
int i=10;
main()
{
extern int i;
{
    int i=20;
      {
      const volatile unsigned i=30;
      printf("%d",i);
      }
  printf("%d",i);
 }
printf("%d",i);
}
```

Answer:

30,20,10

Explanation:

'{' introduces new block and thus new scope. In the innermost block i is declared as, const volatile unsigned.

which is a valid declaration. i is assumed of type int. So printf prints 30. In the next block, i has value 20 and so printf prints 20. In the outermost block, i is declared as extern, so no storage space is allocated for it. After compilation is over the linker resolves it to global variable i (since it is the only variable visible there). So it prints i's value as 10.

68.
```
main()
{
  int *j;
   {
   int i=10;
   j=&i;
   }
   printf("%d",*j);
}
```

Answer:

10

Explanation:

The variable i is a block level variable and the visibility is inside that block only. But the lifetime of i is lifetime of the function so it lives upto the exit of main function. Since the i is still allocated space, *j prints the value stored in i since j points i.

69.
```
main()
{
int i=-1;
-i;
printf("i = %d, -i = %d \n",i,-i);
}
```

Answer:

i = -1, -i = 1

Explanation:

-i is executed and this execution doesn't affect the value of i. In printf first you just print the value of i. After that the value of the expression -i = -(-1) is printed.

70.
```
#include<stdio.h>
main()
```

```
{
  const int i=4;
  float j;
  j = ++i;
  printf("%d  %f", i,++j);
}
```

Answer:

Compiler error

Explanation:

i is a constant. You cannot change the value of constant.

71.

```
#include<stdio.h>
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);
}
```

Answer:

garbagevalue..1

Explanation:

p=&a[2][2][2] you declare only two 2D arrays but you are trying to access the third 2D(which you are not declared) it will print garbage values. *q=***a starting address of a is assigned integer pointer now q is pointing to starting address of a.if you print *q meAnswer:it will print first element of 3D array.

72.

```
#include<stdio.h>
main()
 {
   register i=5;
   char j[]= "hello";
    printf("%s  %d",j,i);
}
```

Answer:

hello 5

Explanation:

If you declare i as register compiler will treat it as ordinary integer and it will take integer value. i value may be stored either in register or in memory.

73.
```
main()
{
    int i=5,j=6,z;
    printf("%d",i+++j);
    }
```

Answer:

11

Explanation:

The expression i+++j is treated as (i++ + j).

74.
```
struct aaa{
    struct aaa *prev;
    int i;
    struct aaa *next;
    };
main()
{
 struct aaa abc,def,ghi,jkl;
 int x=100;
 abc.i=0;abc.prev=&jkl;
 abc.next=&def;
 def.i=1;def.prev=&abc;def.next=&ghi;
 ghi.i=2;ghi.prev=&def;
 ghi.next=&jkl;
 jkl.i=3;jkl.prev=&ghi;jkl.next=&abc;
 x=abc.next->next->prev->next->i;
 printf("%d",x);
 }
```

Answer:

2

Explanation:

Above all statements form a double circular linked list;

abc.next->next->prev->next->i

this one points to "ghi" node the value of at particular node is 2.

75.
```
struct point
 {
 int x;
 int y;
 };
struct point origin,*pp;
main()
{
pp=&origin;
printf("origin is(%d%d)\n",(*pp).x,(*pp).y);
printf("origin is (%d%d)\n",pp->x,pp->y);
}
```

Answer:

origin is(0,0)

origin is(0,0)

Explanation:

pp is a pointer to structure. We can access the elements of the structure either with arrow mark or with indirection operator.

Note:

Since structure point is globally declared x & y are initialized as zeroes.

76.
```
main()
{
 int i=_l_abc(10);
printf("%d\n",--i);
}
int _l_abc(int i)
{
```

```
return(i++);
}
```

Answer:

9

Explanation:

return(i++) it will first return i and then increments. *i.e.*, 10 will be returned.

77.
```
main()
{
 char *p;
 int *q;
 long *r;
 p=q=r=0;
 p++;
 q++;
 r++;
 printf("%p...%p...%p",p,q,r);
}
```

Answer:

0001...0002...0004

Explanation:

++ operator when applied to pointers increments address according to their corresponding data-types.

78.
```
main()
{
 char c=' ',x,convert(z);
 getc(c);
 if((c>='a') && (c<='z'))
 x=convert(c);
 printf("%c",x);
}
convert(z)
{
  return z-32;
```

```
}
```

Answer:

Compiler error.

Explanation:

Declaration of convert and format of getc() are wrong.

79.

```
main(int argc, char **argv)
{
 printf("enter the character");
 getchar();
 sum(argv[1],argv[2]);
}
sum(num1,num2)
int num1,num2;
{
 return num1+num2;
}
```

Answer:

Compiler error.

Explanation:

argv[1] & argv[2] are strings. They are passed to the function sum without converting it to integer values.

80.

```
# include <stdio.h>
int one_d[]={1,2,3};
main()
{
 int *ptr;
 ptr=one_d;
 ptr+=3;
 printf("%d",*ptr);
}
```

Answer:

garbage value

Explanation:

ptr pointer is pointing to out of the array range of one_d.

81.
```
# include<stdio.h>
aaa() {
 printf("hi");
 }
bbb(){
 printf("hello");
 }
ccc(){
 printf("bye");
 }
main()
{
  int (*ptr[3])();
   ptr[0]=aaa;
  ptr[1]=bbb;
   ptr[2]=ccc;
   ptr[2]();
}
```

Answer:

bye

Explanation:

ptr is array of pointers to functions of return type int.ptr[0] is assigned to address of the function aaa. Similarly ptr[1] and ptr[2] for bbb and ccc respectively. ptr[2]() is in effect of writing ccc(), since ptr[2] points to ccc.

82.
```
#include<stdio.h>
main()
{
FILE *ptr;
char i;
ptr=fopen("zzz.c","r");
```

```
while((i=fgetch(ptr))!=EOF)
printf("%c",i);
}
```

Answer:

contents of zzz.c followed by an infinite loop.

Explanation:

The condition is checked against EOF, it should be checked against NULL.

83.

```
main()
{
 int i =0;j=0;
 if(i && j++)
 printf("%d..%d",i++,j);
printf("%d..%d,i,j);
}
```

Answer:

0..0

Explanation:

The value of i is 0. Since this information is enough to determine the truth value of the boolean expression. So the statement following the if statement is not executed. The values of i and j remain unchanged and get printed.

84.

```
main()
{
 int i;
 i = abc();
 printf("%d",i);
}
abc()
{
 _AX = 1000;
}
```

Answer:

1000

Explanation:

Normally the return value from the function is through the information from the accumulator. Here _AH is the pseudo global variable denoting the accumulator. Hence, the value of the accumulator is set 1000 so the function returns value 1000.

85.
```
int i;
main(){
int t;
for ( t=4;scanf("%d",&i)-t;printf("%d\n",i))
      printf("%d--",t--);
      }
      // If the inputs are 0,1,2,3 find the o/p
```

Answer:

4--0
3--1
2--2

Explanation:

Let us assume some x= scanf("%d",&i)-t the values during execution will be,

t	i	x
4	0	-4
3	1	-2
2	2	0

86.
```
main(){
 int a= 0;int b = 20;char x =1;char y =10;
 if(a,b,x,y)
      printf("hello");
 }
```

Answer:

hello

Explanation:

The comma operator has associativity from left to right. Only the rightmost value is returned and the other values are evaluated and ignored. Thus the value of last variable y is returned to check in if. Since it is a non zero value if becomes true so, "hello" will be printed.

87. main(){
unsigned int i;
for(i=1;i>-2;i--)
printf("c aptitude");
}

Explanation:

i is an unsigned integer. It is compared with a signed value. Since the both types doesn't match, signed is promoted to unsigned value. The unsigned equivalent of -2 is a huge value so condition becomes false and control comes out of the loop.

88. In the following pgm add a stmt in the function fun such that the address of 'a' gets stored in 'j'.

```
main(){
 int * j;
 void fun(int **);
 fun(&j);
}
void fun(int **k) {
 int a =0;
 /* add a stmt here*/
}
```

Answer:

*k = &a

Explanation:

The argument of the function is a pointer to a pointer.

89. What are the following notations of defining functions known as?

i. int abc(int a,float b)
{
/* some code */
}

ii. int abc(a,b)
int a; float b;
{
/* some code*/
}

Answer:

i. ANSI C notation

ii. Kernighan & Ritche notation.

90.
```
main()
{
char *p;
p="%d\n";
p++;
p++;
printf(p-2,300);
}
```

Answer:

300

Explanation:

The pointer points to % since it is incremented twice and again decremented by 2, it points to '%d\n' and 300 is printed.

91.
```
main(){
 char a[100];
 a[0]='a';a[1]='b';a[2]='c';a[4]='d';
 abc(a);
}
abc(char a[]){
a++;
printf("%c",*a);
 a++;
 printf("%c",*a);
}
```

Explanation:

The base address is modified only in function and as a result a points to 'b' then after incrementing to 'c' so bc will be printed.

92.
```
func(a,b)
int a,b;
```

```
{
    return( a= (a==b) );
}
main()
{
int process(),func();
printf("The value of process is %d !\n ",process(func,3,6));
}
process(pf,val1,val2)
int (*pf) ();
int val1,val2;
{
return((*pf) (val1,val2));
 }
```

Answer:

The value of process is 0 !

Explanation:

The function 'process' has 3 parameters - 1, a pointer to another function 2 and 3, integers. When this function is invoked from main, the following substitutions for formal parameters take place: func for pf, 3 for val1 and 6 for val2. This function returns the result of the operation performed by the function 'func'. The function func has two integer parameters. The formal parameters are substituted as 3 for a and 6 for b. since 3 is not equal to 6, a==b returns 0. Therefore the function returns 0 which in turn is returned by the function 'process'.

93.

```
void main()
{
    static int i=5;
    if(--i){
        main();
        printf("%d ",i);
    }
}
```

Answer:

0 0 0 0

Explanation:

The variable "I" is declared as static, hence memory for I will be allocated for only once, as it encounters the statement. The function main() will be called recursively unless I becomes equal to 0, and since main() is recursively called, so the value of static I ie., 0 will be printed every time the control is returned.

94.
```
void main()
{
    int k=ret(sizeof(float));
    printf("\n here value is %d",++k);
}
int ret(int ret)
{
    ret += 2.5;
    return(ret);
}
```

Answer:

Here value is 7

Explanation:

The int ret(int ret), ie., the function name and the argument name can be the same.

Firstly, the function ret() is called in which the sizeof(float) *i.e.*, 4 is passed, after the first expression the value in ret will be 6, as ret is integer hence the value stored in ret will have implicit type conversion from float to int. The ret is returned in main() it is printed after and preincrement.

95.
```
void main()
{
    char a[]="12345\0";
    int i=strlen(a);
    printf("here in 3 %d\n",++i);
}
```

Answer:

here in 3 6

Explanation:

The char array 'a' will hold the initialized string, whose length will be counted from 0 till the null character. Hence the 'I' will hold the value equal to 5, after the pre-increment in the printf statement, the 6 will be printed.

96. void main()

```
{
    unsigned giveit=-1;
    int gotit;
    printf("%u ",++giveit);
    printf("%u \n",gotit=--giveit);
}
```

Answer:

0 65535

Explanation:

give it value is incremented. In printf, the value of gotit is the decrement of give it.

97. void main()

```
{
    int i;
    char a[]="\0";
    if(printf("%s\n",a))
        printf("Ok here \n");
    else
        printf("Forget it\n");
}
```

Answer:

Ok here

Explanation:

printf will return how many characters does it print. Hence printing a null character returns 1 which makes the if statement true, thus "Ok here" is printed.

98. void main()

```
{
    void *v;
    int integer=2;
    int *i=&integer;
    v=i;
    printf("%d",(int*)*v);
}
```

Answer:

Compiler Error. We cannot apply indirection on type void*.

Explanation:

Void pointer is a generic pointer type. No pointer arithmetic can be done on it. Void pointers are normally used for:

1. Passing generic pointers to functions and returning such pointers.
2. As a intermediate pointer type.
3. Used when the exact pointer type will be known at a later point of time.

99.
```
void main()
{
    int i=i++,j=j++,k=k++;
    printf("%d%d%d",i,j,k);
}
```

Answer:

Garbage values.

Explanation:

An identifier is available to use in program code from the point of its declaration.

So expressions such as i = i++ are valid statements. The i, j and k are automatic variables and so they contain some garbage value. *Garbage in is garbage out (GIGO).*

100.
```
void main()
{
    static int i=i++, j=j++, k=k++;
    printf("i = %d j = %d k = %d", i, j, k);
}
```

Answer:

i = 1 j = 1 k = 1

Explanation:

Since static variables are initialized to zero by default.

101.
```
void main()
{
    while(1){
        if(printf("%d",printf("%d")))
            break;
        else
            continue;
    }
}
```

Answer:

Garbage values

Explanation:

The inner printf executes first to print some garbage value. The printf returns No. of characters printed and this value also cannot be predicted. Still the outer printf prints something and so returns a non-zero value. So it encounters the break statement and comes out of the while statement.

102.
```
main()
{
    unsigned int i=10;
    while(i-->=0)
    printf("%u ",i);

}
```

Answer:

10 9 8 7 6 5 4 3 2 1 0 65535 65534…..

Explanation:

Since i is an unsigned integer it can never become negative. So the expression i-- >=0 will always be true, leading to an infinite loop.

103.
```
#include<conio.h>
main()
{
    int x,y=2,z,a;
    if(x=y%2) z=2;
    a=2;
    printf("%d %d",z,x);
}
```

Answer:

Garbage-value 0

Explanation:

The value of y%2 is 0. This value is assigned to x. The condition reduces to if (x) or in other words if(0) and so z goes uninitialized.

Thumb Rule: Check all control paths to write bug free code.

104.
```
main()
{
    int a[10];
    printf("%d",*a+1-*a+3);
}
```
Answer:

4

Explanation:

*a and -*a cancels out. The result is as simple as 1 + 3 = 4 !

105.
```
#define prod(a,b) a*b
main()
{
    int x=3,y=4;
    printf("%d",prod(x+2,y-1));
}
```
Answer:

10

Explanation:

The macro expands and evaluates to as:
x+2*y-1 => x+(2*y)-1 => 10

106.
```
main()
{
    unsigned int i=65000;
    while(i++!=0);
    printf("%d",i);
}
```
Answer:

1

Explanation:

Note the semicolon after the while statement. When the value of i becomes 0 it comes out of while loop. Due to post-increment on i the value of i while printing is 1.

107.
```
main()
{
```

```
    int i=0;
    while(+(+i--)!=0)
    i-=i++;
    printf("%d",i);
}
```

Answer:

-1

Explanation:

Unary + is the only dummy operator in C. So it has no effect on the expression and now the while loop is, while(i--!=0) which is false and so breaks out of while loop. The value –1 is printed due to the post-decrement operator.

108.

```
main()
{
    float f=5,g=10;
    enum{i=10,j=20,k=50};
    printf("%d\n",++k);
    printf("%f\n",f<<2);
    printf("%lf\n",f%g);
    printf("%lf\n",fmod(f,g));
}
```

Answer:

Line no 5: Error: Lvalue required
Line no 6: Cannot apply leftshift to float
Line no 7: Cannot apply mod to float

Explanation:

Enumeration constants cannot be modified, so you cannot apply ++.
Bit-wise operators and % operators cannot be applied on float values.
fmod() is to find the modulus values for floats as % operator is for ints.

109.

```
main()
{
    int i=10;
    void pascal f(int,int,int);
    f(i++,i++,i++);
    printf(" %d",i);
```

```
}
void pascal f(integer :i,integer:j,integer :k)
{
write(i,j,k);
}
```

Answer:

Compiler error: unknown type integer

Compiler error: undeclared function write

Explanation:

Pascal keyword doesn't mean that pascal code can be used. It means that the function follows Pascal argument passing mechanism in calling the functions.

110.

```
void pascal f(int i,int j,int k)
{
printf("%d %d %d",i, j, k);
}
void cdecl f(int i,int j,int k)
{
printf("%d %d %d",i, j, k);
}
main()
{
    int i=10;
    f(i++,i++,i++);
    printf(" %d\n",i);
    i=10;
    f(i++,i++,i++);
    printf(" %d",i);
}
```

Answer:

10 11 12 13

12 11 10 13

Explanation:

Pascal argument passing mechanism forces the arguments to be called from left to right. cdecl is the normal C argument passing mechanism where the arguments are passed from right to left.

111. What is the output of the program given below?

```
main()
  {
    signed char i=0;
    for(;i>=0;i++) ;
    printf("%d\n",i);
  }
```

Answer:

-128

Explanation:

Notice the semicolon at the end of the for loop. The initial value of the i is set to 0. The inner loop executes to increment the value from 0 to 127 (the positive range of char) and then it rotates to the negative value of -128. The condition in the for loop fails and so comes out of the for loop. It prints the current value of i that is -128.

112.

```
main()
  {
    unsigned char i=0;
    for(;i>=0;i++) ;
    printf("%d\n",i);
  }
```

Answer:

infinite loop

Explanation:

The difference between the previous question and this one is that the char is declared to be unsigned. So the i++ can never yield negative value and i>=0 never becomes false so that it can come out of the for loop.

113.

```
main()
    {
    char i=0;
    for(;i>=0;i++) ;
    printf("%d\n",i);

 }
```

Answer:

Behavior is implementation dependent.

Explanation:

The detail if the char is signed/unsigned by default is implementation dependent. If the implementation treats the char to be signed by default the program will print –128 and terminate. On the other hand if it considers char to be unsigned by default, it goes to infinite loop.

Rule:

You can write programs that have implementation dependent behavior. But don't write programs that depend on such behavior.

114. Is the following statement a declaration/definition. Find what does it mean?

int (*x)[10];

Answer:

Definition.

x is a pointer to array of(size 10) integers.

Apply clock-wise rule to find the meaning of this definition.

115. What is the output for the program given below?

```
typedef enum errorType{warning, error, exception,}error;
main()
{
  error g1;
  g1=1;
  printf("%d",g1);
}
```

Answer:

Compiler error: Multiple declaration for error.

Explanation:

The name error is used in the two meanings. One means that it is a enumerator constant with value 1. The another use is that it is a type name (due to typedef) for enum errorType. Given a situation the compiler cannot distinguish the meaning of error to know in what sense the error is used:

```
        error g1;
        g1=error;
// which error it refers in each case?
```

When the compiler can distinguish between usages then it will not issue error (in pure technical terms, names can only be overloaded in different namespaces).

Note: The extra comma in the declaration, enum errorType{warning, error, exception,}

is not an error. An extra comma is valid and is provided just for programmer's convenience.

116. typedef struct error{int warning, error, exception;}error;

```
main()
{
   error g1;
   g1.error =1;
   printf("%d",g1.error);
}
```

Answer:

1

Explanation:

The three usages of name errors can be distinguishable by the compiler at any instance, so valid (they are in different namespaces).

Typedef struct error{int warning, error, exception;}error;

This error can be used only by preceding the error by struct kayword as in:

struct error someError;

typedef struct error{int warning, error, exception;}error;

This can be used only after . (dot) or -> (arrow) operator preceded by the variable name as in :

```
g1.error =1;
printf("%d",g1.error);
typedef struct error{int warning, error, exception;}error;
```

This can be used to define variables without using the preceding struct keyword as in:

```
error g1;
```

Since the compiler can perfectly distinguish between these three usages, it is perfectly legal and valid.

Note:

This code is given here to just explain the concept behind. In real programming don't use such overloading of names. It reduces the readability of the code. Possible doesn't mean that we should use it!

117. #ifdef something

```
int some=0;
#endif
main()
{
    int thing = 0;
    printf("%d %d\n", some ,thing);
}
```

Answer:

Compiler error : undefined symbol some

Explanation:

This is a very simple example for conditional compilation. The name something is not already known to the compiler making the declaration

int some = 0;

effectively removed from the source code.

118. #if something == 0

```
int some=0;
#endif
main()
{
    int thing = 0;
    printf("%d %d\n", some ,thing);
}
```

Answer:

0 0

Explanation:

This code is to show that preprocessor expressions are not the same as the ordinary expressions. If a name is not known the preprocessor treats it to be equal to zero.

119. What is the output for the following program?

```
main()
 {
 int arr2D[3][3];
```

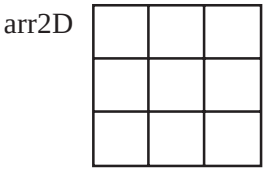
```
    printf("%d\n", ((arr2D==* arr2D)&&(* arr2D == arr2D[0])) );
}
```

Answer:

1

Explanation:

This is due to the close relation between the arrays and pointers. N dimensional arrays are made up of (N-1) dimensional arrays. arr2D is made up of a 3 single arrays that contains 3 integers each.



arr2D[1]

arr2D[2]

arr2D[3]

The name arr2D refers to the beginning of all the 3 arrays. *arr2D refers to the start of the first 1D array (of 3 integers) that is the same address as arr2D. So the expression (arr2D == *arr2D) is true (1).

Similarly, *arr2D is nothing but *(arr2D + 0), adding a zero doesn't change the value/meaning. Again arr2D[0] is the another way of telling *(arr2D + 0). So the expression (*(arr2D + 0) == arr2D[0]) is true (1).

Since both parts of the expression evaluates to true the result is true(1) and the same is printed.

120.
```
void main()
{
    if(~0 == (unsigned int)-1)
    printf("You can answer this if you know how values are represented in memory");
}
```

Answer:

You can answer this if you know how values are represented in memory.

Explanation:

~ (tilde operator or bit-wise negation operator) operates on 0 to produce all ones to fill the space for an integer. –1 is represented in unsigned value as all 1's and so both are equal.

121.
```
int swap(int *a,int *b)
    {
     *a=*a+*b;*b=*a-*b;*a=*a-*b;
    }
    main()
    {
        int x=10,y=20;
        swap(&x,&y);
        printf("x= %d y = %d\n",x,y);
    }
```

Answer:

x = 20 y = 10

Explanation:

This is one way of swapping two values. Simple checking will help understand this.

122.
```
main()
{
char *p = "ayqm";
printf("%c",++*(p++));
}
```

Answer:

b

Explanation:

Address of p is incremented and the value of p is incremented after that.

123.
```
main()
{
    int i=5;
    printf("%d",++i++);
}
```

Answer:

Compiler error: Lvalue required in function main

Explanation:

++i yields an rvalue. For postfix ++ to operate an lvalue is required.

124.
```
main()
{
    char *p = "ayqm";
```

```
    char c;
    c = ++*p++;
    printf("%c",c);
}
```

Answer:

b

Explanation:

There is no difference between the expression ++*(p++) and ++*p++. Parenthesis just works as a visual clue for the reader to see which expression is first evaluated.

125.

```
int aaa() {printf("Hi");}
int bbb(){printf("hello");}
iny ccc(){printf("bye");}

main()
{
int ( * ptr[3]) ();
ptr[0] = aaa;
ptr[1] = bbb;
ptr[2] =ccc;
ptr[2]();
}
```

Answer:

bye

Explanation:

int (* ptr[3])() says that ptr is an array of pointers to functions that takes no arguments and returns the type int. By the assignment ptr[0] = aaa; it means that the first function pointer in the array is initialized with the address of the function aaa. Similarly, the other two array elements also get initialized with the addresses of the functions bbb and ccc. Since ptr[2] contains the address of the function ccc, the call to the function ptr[2]() is same as calling ccc(). So it results in printing "bye".

126.

```
main()
{
```

```
int i=5;
printf("%d",i=++i ==6);
}
```

Answer:

1

Explanation:

The expression can be treated as i = (++i==6), because == is of higher precedence than = operator. In the inner expression, ++i is equal to 6 yielding true(1). Hence the result.

127.

```
main()
{
    char p[ ]="%d\n";
    p[1] = 'c';
    printf(p,65);
}
```

Answer:

A

Explanation:

Due to the assignment p[1] = 'c' the string becomes, "%c\n". Since this string becomes the format string for printf and ASCII value of 65 is 'A', the same gets printed.

128.

```
void ( * abc( int, void ( *def) () ) ) ();
```

Answer:

abc is a ptr to a function which takes 2 parameters .(a). an integer variable.(b). a ptrto a funtion which returns void. the return type of the function is void.

Explanation:

Apply the clock-wise rule to find the result.

129.

```
main()
{
while (strcmp("some","some\0"))
printf("Strings are not equal\n");
}
```

Answer:

No output

Explanation:

Ending the string constant with \0 explicitly makes no difference. So "some" and "some\0" are equivalent. So, strcmp returns 0 (false) hence breaking out of the while loop.

130. main()

```
{
    char str1[] = {'s','o','m','e'};
    char str2[] = {'s','o','m','e','\0'};
    while (strcmp(str1,str2))
    printf("Strings are not equal\n");
}
```

Answer:

"Strings are not equal"

"Strings are not equal"

….

Explanation:

If a string constant is initialized explicitly with characters, '\0' is not appended automatically to the string. Since str1 doesn't have null termination, it treats whatever the values that are in the following positions as part of the string until it randomly reaches a '\0'. So str1 and str2 are not the same, hence the result.

131. main()

```
{
    int i = 3;
    for (;i++=0;) printf("%d",i);
}
```

Answer:

Compiler Error: Lvalue required.

Explanation:

As we know that increment operators return rvalues and hence it cannot appear on the left hand side of an assignment operation.

132.
```
void main()
{
    int *mptr, *cptr;
    mptr = (int*)malloc(sizeof(int));
    printf("%d",*mptr);
    int *cptr = (int*)calloc(sizeof(int),1);
    printf("%d",*cptr);
}
```

Answer:

garbage-value 0

Explanation:

The memory space allocated by malloc is uninitialized, whereas calloc returns the allocated memory space initialized to zeros.

133.
```
void main()
{
    static int i;
    while(i<=10)
    (i>2)?i++:i--;
    printf("%d", i);
}
```

Answer:

32767

Explanation:

Since i is static it is initialized to 0. Inside the while loop the conditional operator evaluates to false, executing i--. This continues till the integer value rotates to positive value (32767). The while condition becomes false and hence, comes out of the while loop, printing the i value.

134.
```
main()
{
    int i=10,j=20;
    j = i, j?(i,j)?i:j:j;
    printf("%d %d",i,j);
}
```

Answer:

10 10

Explanation:

The Ternary operator (? :) is equivalent for if-then-else statement. So the question can be written as:

```
if(i,j)
   {
if(i,j)
   j = i;
else
   j = j;
}
else
   j = j;
```

135. 1. const char *a;

2. char* const a;

3. char const *a;

-Differentiate the above declarations.

Answer:

1. 'const' applies to char * rather than 'a' (pointer to a constant char)

   *a='F' : illegal

   a="Hi" : legal

2. 'const' applies to 'a' rather than to the value of a (constant pointer to char)

   *a='F' : legal

   a="Hi" : illegal

3. Same as 1.

136.
```
main()
{
int i=5,j=10;
i=i&=j&&10;
printf("%d %d",i,j);
}
```

Answer:

1 10

Explanation:

The expression can be written as i=(i&=(j&&10)); The inner expression (j&&10) evaluates to 1 because j==10. i is 5. i = 5&1 is 1. Hence the result.

137.
```
main()
{
    int i=4,j=7;
    j = j || i++ && printf("YOU CAN");
    printf("%d %d", i, j);
}
```

Answer:

4 1

Explanation:

The boolean expression needs to be evaluated only till the truth value of the expression is not known. j is not equal to zero itself means that the expression's truth value is 1. Because it is followed by || and *true* || (*anything*) => *true where* (*anything*) *will not be evaluated.* So the remaining expression is not evaluated and so the value of i remains the same.

Similarly when && operator is involved in an expression, when any of the operands become false, the whole expression's truth value becomes false and hence the remaining expression will not be evaluated. *false &&* (*anything*) => *false where* (*anything*) *will not be evaluated.*

138.
```
main()
{
    register int a=2;
    printf("Address of a = %d",&a);
    printf("Value of a   = %d",a);
}
```

Answer:

Compiler Error: '&' on register variable

Rule to Remember:

& (address of) operator cannot be applied on register variables.

139. main()

```
{
    float i=1.5;
    switch(i)
        {
            case 1: printf("1");
            case 2: printf("2");
            default : printf("0");
    }
}
```

Answer:

Compiler Error: switch expression not integral

Explanation:

Switch statements can be applied only to integral types.

140. main()

```
{
        extern i;
        printf("%d\n",i);
        {
            int i=20;
            printf("%d\n",i);
        }
}
```

Answer:

Linker Error : Unresolved external symbol i

Explanation:

The identifier i is available in the inner block and so using extern has no use in resolving it.

141. main()

```
{
        int a=2,*f1,*f2;
        f1=f2=&a;
        *f2+=*f2+=a+=2.5;
```

```
    printf("\n%d %d %d",a,*f1,*f2);
}
```

Answer:

16 16 16

Explanation:

f1 and f2 both refer to the same memory location a. So changes through f1 and f2 ultimately affects only the value of a.

142.

```
main()
{
        char *p="GOOD";
        char a[ ]="GOOD";
        printf("\n sizeof(p) = %d, sizeof(*p) = %d, strlen(p) = %d", sizeof(p),
        sizeof(*p), strlen(p));
        printf("\n sizeof(a) = %d, strlen(a) = %d", sizeof(a), strlen(a));
}
```

Answer:

sizeof(p) = 2, sizeof(*p) = 1, strlen(p) = 4

sizeof(a) = 5, strlen(a) = 4

Explanation:

sizeof(p) => sizeof(char*) => 2

sizeof(*p) => sizeof(char) => 1

Similarly,

sizeof(a) => size of the character array => 5

When sizeof operator is applied to an array it returns the sizeof the array and it is not the same as the sizeof the pointer variable. Here the sizeof(a) where a is the character array and the size of the array is 5 because the space necessary for the terminating NULL character should also be taken into account.

143.

```
#define DIM( array, type) sizeof(array)/sizeof(type)
main()
{
    int arr[10];
    printf("The dimension of the array is %d", DIM(arr, int));
}
```

Answer:

10

Explanation:

The size of integer array of 10 elements is 10 * sizeof(int). The macro expands to sizeof(arr)/sizeof(int) => 10 * sizeof(int) / sizeof(int) => 10.

144.
```
int DIM(int array[])
{
return sizeof(array)/sizeof(int );
}
main()
{
    int arr[10];
    printf("The dimension of the array is %d", DIM(arr));
}
```

Answer:

1

Explanation:

Arrays cannot be passed to functions as arguments and only the pointers can be passed. So the argument is equivalent to int * array (this is one of the very few places where [] and * usage are equivalent). The return statement becomes, sizeof(int *)/ sizeof(int) that happens to be equal in this case.

145.
```
main()
{
    static int a[3][3]={1,2,3,4,5,6,7,8,9};
    int i,j;
    static *p[]={a,a+1,a+2};
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        printf("%d\t%d\t%d\t%d\n",*(*(p+i)+j),
        *(*(j+p)+i),*(*(i+p)+j),*(*(p+j)+i));
    }
}
```

Answer:

1	1	1	1
2	4	2	4
3	7	3	7
4	2	4	2
5	5	5	5
6	8	6	8
7	3	7	3
8	6	8	6
9	9	9	9

Explanation:

((p+i)+j) is equivalent to p[i][j].

146.
```
main()
{
    void swap();
    int x=10,y=8;
    swap(&x,&y);
    printf("x=%d y=%d",x,y);
}
void swap(int *a, int *b)
{
  *a ^= *b,  *b ^= *a, *a ^= *b;
}
```

Answer:

x=10 y=8

Explanation:

Using ^ like this is a way to swap two variables without using a temporary variable and that too in a single statement.

Inside main(), void swap(); means that swap is a function that may take any number of arguments (not no arguments) and returns nothing. So this doesn't issue a compiler error by the call swap(&x,&y); that has two arguments.

This convention is historically due to pre-ANSI style (referred to as Kernighan and Ritchie style) style of function declaration. In that style, the swap function will be defined as follows,

```
void swap()
int *a, int *b
{
 *a ^= *b,  *b ^= *a, *a ^= *b;
}
```

where the arguments follow the (). So naturally the declaration for swap will look like, void swap() which means the swap can take any number of arguments.

147.
```
main()
{
    int i = 257;
    int *iPtr = &i;
    printf("%d %d", *((char*)iPtr), *((char*)iPtr+1) );
}
```

Answer:

1 1

Explanation:

The integer value 257 is stored in the memory as, 00000001 00000001, so the individual bytes are taken by casting it to char * and get printed.

148.
```
main()
{
    int i = 258;
    int *iPtr = &i;
    printf("%d %d", *((char*)iPtr), *((char*)iPtr+1) );
}
```

Answer:

2 1

Explanation:

The integer value 257 can be represented in binary as, 00000001 00000001. Remember that the INTEL machines are 'small-endian' machines. Small-endian means that the lower order bytes are stored in the higher memory addresses and the higher order bytes are stored in lower addresses. The integer value 258 is stored in memory as: 00000001 00000010.

149. main()

```
{
    int i=300;
    char *ptr = &i;
    *++ptr=2;
    printf("%d",i);
}
```

Answer:

556

Explanation:

The integer value 300 in binary notation is: 00000001 00101100. It is stored in memory (small-endian) as: 00101100 00000001. Result of the expression *++ptr = 2 makes the memory representation as: 00101100 00000010. So the integer corresponding to it is 00000010 00101100 => 556.

150.

```
#include <stdio.h>
main()
{
    char * str = "hello";
    char * ptr = str;
    char least = 127;
    while (*ptr++)
    least = (*ptr<least ) ?*ptr :least;
    printf("%d",least);
}
```

Answer:

0

Explanation:

After 'ptr' reaches the end of the string the value pointed by 'str' is '\0'. So the value of 'str' is less than that of 'least'. So the value of 'least' finally is 0.

151. Declare an array of N pointers to functions returning pointers to functions returning pointers to characters?

Answer:

```
(char*(*)( )) (*ptr[N])( );
```

152. main()

```
{
    struct student
    {
        char name[30];
        struct date dob;
    }stud;
    struct date
        {
        int day,month,year;
        };
        scanf("%s%d%d%d", stud.rollno, &student.dob.day, &student.dob.month,
        &student.dob.year);
}
```

Answer:

Compiler Error: Undefined structure date

Explanation:

Inside the struct definition of 'student' the member of type struct date is given. The compiler doesn't have the definition of date structure (forward reference is not allowed in C in this case) so it issues an error.

153. main()

```
{
    struct date;
    struct student
{
    char name[30];
    struct date dob;
}stud;
    struct date
    {
        int day,month,year;
    };
    scanf("%s%d%d%d", stud.rollno, &student.dob.day, &student.dob.month,
    &student.dob.year);
}
```

Answer:

Compiler Error: Undefined structure date

Explanation:

Only declaration of struct date is available inside the structure definition of 'student' but to have a variable of type struct date the definition of the structure is required.

154. There were 10 records stored in "somefile.dat" but the following program printed 11 names. What went wrong?

```
void main()
{
    struct student
    {
    char name[30], rollno[6];
    }stud;
    FILE *fp = fopen("somefile.dat","r");
    while(!feof(fp))
     {
          fread(&stud, sizeof(stud), 1 , fp);
          puts(stud.name);
    }
}
```

Explanation:

fread reads 10 records and prints the names successfully. It will return EOF only when fread tries to read another record and fails reading EOF (and returning EOF). So it prints the last record again. After this only the condition feof(fp) becomes false, hence comes out of the while loop.

155. Is there any difference between the two declarations?

1. int foo(int *arr[]) and
2. int foo(int *arr[2])

Answer:

No

Explanation:

Functions can only pass pointers and not arrays. The numbers that are allowed inside the [] is just for more readability. So there is no difference between the two declarations.

156. What is the subtle error in the following code segment?

```
void fun(int n, int arr[])
{
    int *p=0;
    int i=0;
    while(i++<n)
        p = &arr[i];
        *p = 0;
}
```

Answer & Explanation:

If the body of the loop never executes p is assigned no address. So p remains NULL where *p =0 may result in problem (may rise to runtime error "NULL pointer assignment" and terminate the program).

157. What is wrong with the following code?

```
int *foo()
{
    int *s = malloc(sizeof(int)100);
    assert(s != NULL);
    return s;
}
```

Answer & Explanation:

assert macro should be used for debugging and finding out bugs. The check s != NULL is for error/exception handling and for that assert shouldn't be used. A plain if and the corresponding remedy statement has to be given.

158. What is the hidden bug with the following statement?

```
assert(val++ != 0);
```

Answer & Explanation:

assert macro is used for debugging and removed in release version. In assert, the experssion involves side-effects. So the behavior of the code becomes different in case of debug version and the release version thus leading to a subtle bug.

Rule to Remember:

Don't use expressions that have side-effects in assert statements.

159.
```
void main()
{
int *i = 0x400;  // i points to the address 400
*i = 0;          // set the value of memory location pointed by i;
}
```

Answer:

Undefined behavior

Explanation:

The second statement results in undefined behavior because it points to some location whose value may not be available for modification. This type of pointer in which the non-availability of the implementation of the referenced location is known as 'incomplete type'.

160.
```
#define assert(cond) if(!(cond)) \
 (fprintf(stderr, "assertion failed: %s, file %s, line %d \n",#cond,\
 __FILE__,__LINE__), abort())

void main()
{
int i = 10;
if(i==0)
   assert(i < 100);
else
   printf("This statement becomes else for if in assert macro");
}
```

Answer:

No output

Explanation:

The else part in which the printf is there becomes the else for if in the assert macro. Hence nothing is printed.

The solution is to use conditional operator instead of if statement,

#define assert(cond) ((cond)?(0): (fprintf (stderr, "assertion failed: \ %s, file %s, line %d \n",#cond, __FILE__,__LINE__), abort()))

Note:

However this problem of "matching with nearest else" cannot be solved by the usual method of placing the if statement inside a block like this,

```
#define assert(cond) { \
if(!(cond)) \
 (fprintf(stderr, "assertion failed: %s, file %s, line %d \n",#cond,\
 __FILE__,__LINE__), abort()) \
}
```

161. Is the following code legal?

```
struct a
  {
    int x;
    struct a b;
  }
```

Answer:

No

Explanation:

It is not legal for a structure to contain a member that is of the same type as in this case. Because this will cause the structure declaration to be recursive without end.

162. Is the following code legal?

```
struct a
 {
    int x;
    struct a *b;
 }
```

Answer:

Yes.

Explanation:

*b is a pointer to type struct a and so is legal. The compiler knows, the size of the pointer to a structure even before the size of the structure is determined(as you know the pointer to any type is of same size). This type of structures is known as 'self-referencing' structure.

163. Is the following code legal?

```
typedef struct a
  {
    int x;
    aType *b;
  }aType
```

Answer:

No

Explanation:

The typename aType is not known at the point of declaring the structure (forward references are not made for typedefs).

164. Is the following code legal?

```
typedef struct a aType;
struct a
{
    int x;
    aType *b;
};
```

Answer:

Yes

Explanation:

The typename aType is known at the point of declaring the structure, because it is already typedefined.

165. Is the following code legal?

```
void main()
{
    typedef struct a aType;
    aType someVariable;
    struct a
{
    int x;
        aType *b;
  };
}
```

Answer:

No

Explanation:

When the declaration,

typedef struct a aType;

is encountered body of struct a is not known. This is known as 'incomplete types'.

166. void main()

{

printf("sizeof (void *) = %d \n", sizeof(void *));

printf("sizeof (int *) = %d \n", sizeof(int *));

printf("sizeof (double *) = %d \n", sizeof(double *));

printf("sizeof(struct unknown *) = %d \n", sizeof(struct unknown *));

}

Answer:

sizeof (void *) = 2

sizeof (int *) = 2

sizeof (double *) = 2

sizeof(struct unknown *) = 2

Explanation:

The pointer to any type is of same size.

167. char inputString[100] = {0};

To get string input from the keyboard which one of the following is better?

1) gets(inputString)

2) fgets(inputString, sizeof(inputString), fp)

Answer & Explanation:

The second one is better because gets(inputString) doesn't know the size of the string passed and so, if a very big input (here, more than 100 chars) the charactes will be written past the input string. When fgets is used with stdin performs the same operation as gets but is safe.

168. Which version do you prefer of the following two?

1) printf("%s",str); // or the more curt one

2) printf(str);

Answer & Explanation:

Prefer the first one. If the str contains any format characters like %d then it will result in a subtle bug.

169.
```
void main()
{
int i=10, j=2;
int *ip= &i, *jp = &j;
int k = *ip/*jp;
printf("%d",k);
}
```

Answer:

Compiler Error: "Unexpected end of file in comment started in line 5".

Explanation:

The programmer intended to divide two integers, but by the "maximum munch" rule, the compiler treats the operator sequence / and * as /* which happens to be the starting of comment. To force what is intended by the programmer,

```
int k = *ip/ *jp;
    // give space explicity separating / and *
    //or
    int k = *ip/(*jp);
    // put braces to force the intention will solve the problem.
```

170.
```
void main()
{
char ch;
for(ch=0;ch<=127;ch++)
printf("%c   %d \n", ch, ch);
}
```

Answer:

Implementaion dependent

Explanation:

The char type may be signed or unsigned by default. If it is signed then ch++ is executed after ch reaches 127 and rotates back to -128. Thus ch is always smaller than 127.

171. Is this code legal?

```
int *ptr;
ptr = (int *) 0x400;
```

Answer:

Yes

Explanation:

The pointer ptr will point at the integer in the memory location 0x400.

172.

```
main()
{
    char a[4]="HELLO";
    printf("%s",a);
}
```

Answer:

Compiler error: Too many initializers

Explanation:

The array a is of size 4 but the string constant requires 6 bytes to get stored.

173.

```
main()
{
    char a[4]="HELL";
    printf("%s",a);
}
```

Answer:

HELL%@!~@!@???@~~!

Explanation:

The character array has the memory just enough to hold the string "HELL" and doesnt have enough space to store the terminating null character. So it prints the HELL correctly and continues to print garbage values till it accidentally comes across a NULL character.

174.

```
main()
{
    int a=10,*j;
    void *k;
    j=k=&a;
    j++;
```

```
    k++;
    printf("\n %u %u ",j,k);
}
```

Answer:

Compiler error: Cannot increment a void pointer

Explanation:

Void pointers are generic pointers and they can be used only when the type is not known and as an intermediate address storage type. No pointer arithmetic can be done on it and you cannot apply indirection operator (*) on void pointers.

175.

```
main()
{
    extern int i;
{   int i=20;
 {
 const volatile unsigned i=30; printf("%d",i);
 }
    printf("%d",i);
 }
    printf("%d",i);
    }
    int i;
```

176. printf can be implemented by using _____ list.

Answer:

Variable length argument lists

177.

```
char *someFun()
    {
    char *temp = "string constant";
    return temp;
    }
    int main()
    {
    puts(someFun());
    }
```

Answer:

string constant

Explanation:

The program suffers no problem and gives the output correctly because the character constants are stored in code/data area and not allocated in stack, so this doesn't lead to dangling pointers.

178. char *someFun1()

```
{
char temp[ ] = “string";
return temp;
}
char *someFun2()
{
char temp[ ] = {‘s’, ‘t’,‘r’,‘i’,‘n’,‘g’};
return temp;
}
int main()
{
puts(someFun1());
puts(someFun2());
}
```

Answer:

Garbage values.

Explanation:

Both the functions suffer from the problem of dangling pointers. In someFun1() temp is a character array and so the space for it is allocated in heap and is initialized with character string “string”. This is created dynamically as the function is called, so is also deleted dynamically on exiting the function so the string data is not available in the calling function main() leading to print some garbage values. The function someFun2() also suffers from the same problem but the problem can be easily identified in this case.

179. What will print it?

```
main()
{
```

```
char *k1="hitechskill.com";
char *k2;
k2=(char*)malloc(20);
memset (k2, 0, 20);
while(*k2++ = *k1++);
printf("%s\n",k2);

}
```

Answer: empty string.

180. What will be printed as the result of the operation below?

```
main()
{
    int a=20,b=35;
    a=b++ + a++;
    b= ++b + ++a;
    printf("%d%d\n",a,b);

}
```

Answer: 5794

181. What will be printed as the result of the operation below?

```
main()
{
  int a=5;
  printf(?%d,%d,%d\n?,a,a< <2,a>>2);
}
```

Answer: 5,20,1

182. What will be printed as the result of the operation below?

```
#define swap(a,b) a=a+b;b=a-b;a=a-b;
void main()
{
  int m=5, n=10;
```

```
    swap (m,n);
    printf("%d %d\n",m,n);
    swap2(m,n);
    printf("%d %d\n",m,n);
}

int swap2(int x, int y)
{
    int temp;
    temp=x;
    y=x;
    x=temp;
    return 0;

}
```

Answer: 10, 5
10, 5

183. What will be printed as the result of the operation below?

```
main()
{
    char *ptr = "SC Systems";
    *pt++; printf("%s\n",pt);
    pt++;
    printf("%s\n",pt);

}
```

Answer: SC Systems
C Systems

184. What will be printed as the result of the operation below?

```
main()
{
    char s1[]="SC";
    char s2[]= "Systems";
```

```
    printf("%s",s1);
}
```

Answer: SC

185. What will be printed as the result of the operation below?

```
main()
{
    char *ptr1;
    char *ptr2;
    ptr1=(char *)malloc(25);
    ptr2=(char *)malloc(25);
    strcpy(ptr1,"SC");
    strcpy(ptr2,"Systems");
    strcat(ptr1,ptr2);
    printf("%s",ptr1);

}
```

Answer: SCSystems

186. The following variable is available in file1.c, who can access it?:

```
static int average;
```

Answer: All the functions in the file1.c can access the variable.

187. What will be the result of the following code?

```
#define TRUE 0 // some code
while(TRUE)
{
    // some code
}
```

Answer: This will not go into the loop as TRUE is defined as 0.

188. What will be printed as the result of the operation below?

```
int x;
int modifyvalue()
```

```
{
   return(x+=10);
}

int changevalue(int x)
{
   return(x+=1);
}

void main()
{
   int x=10;
   x++;
    changevalue(x);
   x++;
   modifyvalue();
   printf("First output:%d\n",x);
   x++;
    changevalue(x);
   printf("Second output:%d\n",x);
   modifyvalue();
   printf("Third output:%d\n",x);

}
```

Answer: 12 , 13 , 13

189. What will be printed as the result of the operation below?

```
main()
{
   int x=10, y=15;
   x = x++;
   y = ++y;
   printf("%d %d\n",x,y);
```

```
}
```

Answer: 11, 16

190. How many times main is get called?

```
main()
{
printf("Jumboree");
main();
}
```

Answer: Till stack overflow

191. What is output for the following program?

```
#include<stdio.h>
main()
{
int *p,*q,i;
p=(int *)100;
q=(int *)200;
i=q-p;
printf("%d",i);
}
```

a) 100 b) 25 c) 0 d) compile error

Answer : b) 25

192. What is output for the following program?

```
#include<stdio.h>
#define swap(a,b)  temp=a,a=b,b=temp;
main()
{
int a=5,b=6;
int temp;
if(a>b)
swap(a,b);
printf("a=%d,b= %d",a,b);
}
```

a) a=5 b=6 b) a=6 b=5 c) a=0 b=6 d) None

Answer: a) a=5 b=6

193. What is output for the following program?

```
#include<stdio.h>
main()
{
unsigned char i;
for( i=0;i<300;i++)
{
printf("*");
}
}
```

a)299 b)300 c)infinite d)none

Answer: c) (infinite)

194. What is output for the following program?

```
#include<stdio.h>
main()
{
int n=2;
int sum=5;
switch(n)
{
case 2:sum=sum-2;
       break;
case 3:sum*=5;
       break;
default :sum=0;
}
printf("%d",sum);
}
```

a)15 b)0 c)6 d)none

Answer: a) 15

195. What is output for the following program?

```
#include<stdio.h>
main()
{
int a=10,b=5;
if(a=a&b)
b=a^b;
printf("a=%d,b=%d",a,b);
}
```

a)a=0,b=5 b)a=10 b=5 c)a=0,b=0 d)none

Answer: a) a=0,b=5

196. What is output for the following program?

```
#include<stdio.h>
main()
{
int a[5],i,*ip;
for(i=0;i<5;i++)
a[i]=i;
ip=a;
printf("%d",*(ip+3*sizeof(int)));
}
```

a)0 b)5 c)1 d)none

Answer: d) none

197. What is the size of the structure?

```
#include<stdio.h>
main()
{
struct
{
char a;
short b;
int c;
}temp;
```

```
}
```

a)7 b)8 c)12 d)120

Answer: b)8

198. Define pointer to function that take argument as character pointer and return void pointer.

Answer: void *(*f)(char *)

199. 5-2-3*5-2 evaluates 18 then

i) - left associative * has precedence over -

ii) - right associative * has precedence over -

iii) * left associative - has precedence over *

iv) * right associative - has precedence over *

a) i b)ii c) iii d)iv

Answer: iv

200. Find the output of the following program.

```
void main()
{
float b=1,h=1,a;
a=1/2*b*h;
printf("%.2f",a);
}
```

a) 0.00

b) 0.0

c) 0.50

d) 0.5

Answer: a) 0.00

201. Find the output of the following program.

```
void main()
{
static int n[3];
printf("%c",*(n+10)+90);
}
```

a) Garbage value
b) Runtime error
c) Compiler error
d) None of these

Answer: d) None of these

202. Find the output of the following program.

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

a) %%%
b) %%%%
c) %%%%%
d) None of these

Answer: a) %%%

203. Find the output of the following program.

```
int v()
{
    float m=0;
    return m++;
}
int main()
{
printf("%.7f ",v());
}
```

a) 0.0000000
b) 0.00
c) Error
d) None

Answer: a) 0.0000000

204. Find the output of the following program.

```
void main()
{
```

```
int i=2,j=1;
j>=i;
printf("%d",j);
}
```

a) 1
b) 0
c) 2
d) None of these

Answer: a) 1

205. Find the output of the following program.

```
void main()
{
int a=10,b=20;
a^=b;
b^=a;
printf("%d : %d",a,b);
}
```

a) 10:30
b) 30:10
c) Garbage value
d) 20:10

Answer: b) 30:10

206. Find the output of the following program.

```
void main()
{
int a=10;
if(!(!a)==!0)
printf("God");
else
printf("Good");
}
```

a) Good
b) God

c) We can't predict
d) Error
Answer: a) Good

207. Find the output of the following program.

```
void main()
{
int a=100,b=200;
printf("%c",a>b?'a':'b');
}
```

a) b
b) a
c) 200
d) 100
Answer: a) b

208. Find the output of the following program.

```
void main()
{
char a=65,b=97;
if('a>'b')
printf("A")
printf("B")
}
```

a) AB
b) A
c) B
d) Error
Answer: a) AB

209. What will be the output?

```
void main()
{
 do
 {
```

```
    if(0)
    main();
    printf("Thaal");
    }while(0);
  }
```

a) Thaal
b) Infinite Loop
c) Syntax error
d) Nothing get printed

Answer: a) Thaal

210. For the following C program

```
#define AREA(x)(3.14*x*x)
main()
{float r1=6.25,r2=2.5,a;
a=AREA(r1);
printf("\n Area of the circle is %f ", a);
a=AREA(r2);
printf("\n Area of the circle is %f ", a);
}
```

What is the output?

Answer: Area of the circle is 122.656250
Area of the circle is 19.625000

211. What will be the output?

```
void main()
{
int d=5;
printf("%f ",d);
}
```

Answer: Undefined

212. What will be the output?

```
void main()
{
```

```
int i;
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;
}
switch(i) case 4:printf("%d",i);
}
```

Answer: 1,2,3,4

213. What will be the output?

```
void main()
{
char *s="\12345s\n";
printf("%d",sizeof(s));
}
```

Answer: 6

214. What will be the output?

```
void main()
{
unsigned i=1; /* unsigned char k= -1 => k=255; */
signed j=-1; /* char k= -1 => k=65535 */
/* unsigned or signed int k= -1 =>k=65535 */
if(i<j)
printf("less");
else
if(i>j)
printf("greater");
else
if(i==j)
printf("equal");
}
```

Answer: less

215. What will be the output :-

```
void main()
{
float j;
j=1000*1000;
printf("%f ",j);
}
```

1. 1000000
2. Overflow
3. Error
4. None

Answer: 4

216. A structure pointer is defined of the type time . With 3 fields min,sec hours having pointers to intergers.

Write the way to initialize the 2nd element to 10.

217. What will be the output?

```
void main()
{
int i=7;
printf("%d",i++*i++);
}
```

Answer: 56

218.

```
int f()
void main()
{
f(1);
f(1,2);
f(1,2,3);
}
f(int i,int j,int k)
{
```

```
printf("%d %d %d",i,j,k);
}
```

What are the number of syntax errors in the above?

Answer: None.

219. What will be the output?

```
void main()
{
int i=7;
printf(“%d”,i++*i++);
}
```

Answer: 56

220. What will be the output?

```
#define one 0
#ifdef one
printf(“one is defined ”);
#ifndef one
printf(“one is not defined ”);
```

Answer: “one is defined”

221. What will be the output?

```
void main()
{
int count=10,*temp,sum=0;
temp=&count;
*temp=20;
temp=&sum;
*temp=count;
printf(“%d %d %d ”,count,*temp,sum);
}
```

Answer: 20 20 20

222. What is alloca()?

Answer: It allocates and frees memory after use/after getting out of scope.

223. What will be the output?

```
main()
{
static i=3;
printf("%d",i--);
return i>0 ? main():0;
}
```

Answer: 321

224. What will be the output?

```
char *foo()
{
char result[100]);
strcpy(result,"anything is good");
return(result);
}
void main()
{
char *j;
j=foo()
printf("%s",j);
}
```

Answer: anything is good.

225. What will be the output?

```
void main()
{
char *s[]={ "dharma","hewlett-packard","siemens","ibm"};
char **p;
p=s;
printf("%s",++*p);
```

```
printf("%s",*p++);
printf("%s",++*p);
}
```

Answer: "harma" (p->add(dharma) && (*p)->harma)
"harma" (after printing, p->add(hewlett-packard) &&(*p)->harma)
"ewlett-packard"

226. What will be the output?

```
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);
}
}
```

a) 0,5,9,13,17
b) 5,9,13,17
c) 12,17,22
d) 16,21
e) Syntax error

Answer: (d)

227. What will be the output?

```
main()
{char c=-64;
int i=-32
unsigned int u =-16;
if(c>i)
{printf("pass1,");
if(c<u)
```

```
printf("pass2");
else
printf("Fail2");
}
else
printf("Fail1);
if(i<u)
printf("pass2");
else
printf("Fail2")
}
```

a) Pass1,Pass2
b) Pass1,Fail2
c) Fail1,Pass2
d) Fail1,Fail2
e) None of these

Answer: (c)

228. What will the following program do?

```
void main()
{
int i;
char a[]="String";
char *p="New Sring";
char *Temp;
Temp=a;
a=malloc(strlen(p) + 1);
strcpy(a,p); //Line number:9//
p = malloc(strlen(Temp) + 1);
strcpy(p,Temp);
printf("(%s, %s)",a,p);
free(p);
free(a);
} //Line number 15//
```

a) Swap contents of p & a and print:(New string, string)

b) Generate compilation error in line number 8
c) Generate compilation error in line number 5
d) Generate compilation error in line number 7
e) Generate compilation error in line number 1
Answer: (b)

229. In the following code segment what will be the result of the function,
value of x , value of y

```
{unsigned int x=-1;
int y;
y = ~0;
if(x == y)
printf("same");
else
printf("not same");
}
```

a) same, MAXINT, -1
b) not same, MAXINT, -MAXINT
c) same , MAXUNIT, -1
d) same, MAXUNIT, MAXUNIT
e) not same, MAXINT, MAXUNIT
Answer: (a)

230. What will be the result of the following program ?

```
char *gxxx()
{static char xxx[1024];
return xxx;
}
main()
{char *g="string";
strcpy(gxxx(),g);
g = gxxx();
strcpy(g,"oldstring");
printf("The string is : %s",gxxx());
}
```

a) The string is : string
b) The string is : Oldstring
c) Run time error/Core dump
d) Syntax error during compilation
e) None of these
Answer: (b)

231. Find the output for the following C program.

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

Answer: An empty string

232. Find the output for the following C program.

```
main()
{
int x=20,y=35;
x = y++ + x++;
y = ++y + ++x;
printf("%d %d\n",x,y);
}
```

Answer: 57 94

233. Find the output for the following C program.

```
main()
{
int x=5;
printf("%d %d %d\n",x,x<<2,x>>2);
}
```

Answer: 5 20 1

234. Find the output for the following C program.

```
#define swap1(a,b) a=a+b;b=a-b;a=a-b;
main()
{
int x=5,y=10;
swap1(x,y);
printf("%d %d\n",x,y);
swap2(x,y);
printf("%d %d\n",x,y);
}
int swap2(int a,int b)
{
int temp;
temp=a;
b=a;
a=temp;
return;
}
```

Answer: 10 5

235. Find the output for the following C program.

```
main()
{
char *ptr = "Ramco Systems";
(*ptr)++;
printf("%s\n",ptr);
ptr++;
printf("%s\n",ptr);
}
```

Answer: Samco Systems

236. Find the output for the following C program.

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

```
{
char s1[]="Ramco";
char s2[]="Systems";
s1=s2;
printf("%s",s1);
}
```

Answer: Compilation error giving it cannot be an modifiable 'lvalue'.

237. Find the output for the following C program.

```
#include<stdio.h>
main()
{
char *p1;
char *p2;
p1=(char *) malloc(25);
p2=(char *) malloc(25);
strcpy(p1,"Ramco");
strcpy(p2,"Systems");
strcat(p1,p2);
printf("%s",p1);
}
```

Answer: RamcoSystems

238. Find the output for the following C program given that

[1]. The following variable is available in file1.c static int average_float;

Answer: All the functions in the file1.c can access the variable.

239. Find the output for the following C program.

```
# define TRUE 0
some code
while(TRUE)
{
some code
}
```

Answer: This won't go into the loop as TRUE is defined as 0.

240. struct list
```
{
 int x;
struct list *next;
 }*head;
 the struct head.x =100
```
Is the above assignment to pointer is correct or wrong ?

Answer: Wrong

241. What is the output of the following ?
```
main( )
{ int i;
i=1;
i=i+2*i++;
 printf(%d,i);
}
```
Answer: 4

242.
```
 main( )
{FILE *fp1,*fp2;
fp1=fopen(“one”,“w”)
fp2=fopen(“one”,“w”)
fputc(‘A’,fp1)
fputc(‘B’,fp2)
fclose(fp1)
fclose(fp2)
}
```
Find the Error, If Any?

Answer: No error. But It will over writes on same file.

243. What will be the output?
```
#define MAN(x,y) (x)>(y)?(x):(y)
 {int i=10;
```

```
j=5;
k=0;
k=MAX(i++,++j);
printf(%d %d %d %d,i,j,k);
}
```

Answer: 10 5 0

244. int z,x=5,y=-10,a=4,b=2;

```
z = x++ - --y * b / a;
```

What number will z in the sample code above contain?

Choice 1 5

Choice 2 6

Choice 3 10 [Ans] Corrected by buddy by running the program

Choice 4 11

Choice 5 12

245. With every use of a memory allocation function, what function should be used to release allocated memory which is no longer needed?

Choice 1 unalloc()

Choice 2 dropmem()

Choice 3 dealloc()

Choice 4 release()

Choice 5 free() [Ans]

246. void *ptr;

myStruct myArray[10];

ptr = myArray;

Which of the following is the correct way to increment the variable "ptr"?

Choice 1 ptr = ptr + sizeof(myStruct); [Ans]

Choice 2 ++(int*)ptr;

Choice 3 ptr = ptr + sizeof(myArray);

Choice 4 increment(ptr);

Choice 5 ptr = ptr + sizeof(ptr);

247.
```
char* myFunc (char *ptr)
{
 ptr += 3;
 return (ptr);
}
int main()

{
 char *x, *y;
 x = “HELLO”;
 y = myFunc (x);
 printf (“y = %s \n”, y);
 return 0;
}
```

What will print when the sample code above is executed?

Choice 1 y = HELLO

Choice 2 y = ELLO

Choice 3 y = LLO

Choice 4 y = LO [Ans]

Choice 5 x = O

248.
```
struct node *nPtr, *sPtr;    /* pointers for a linked list. */
for (nPtr=sPtr; nPtr; nPtr=nPtr->next)
{
 free(nPtr);
}
```

The sample code above releases memory from a linked list. Which of the choices below accurately describes how it will work?

Choice 1 It will work correctly since the for loop covers the entire list.

Choice 2 It may fail since each node "nPtr" is freed before its next address can be accessed. (Ans)

Choice 3 In the for loop, the assignment "nPtr=nPtr->next" should be changed to "nPtr=nPtr.next".

Choice 4 This is invalid syntax for freeing memory.

Choice 5 The loop will never end.

249. What function will read a specified number of elements from a file?

Choice 1 fileread()

Choice 2 getline()

Choice 3 readfile()

Choice 4 fread()(Ans)

Choice 5 gets().

250. "My salary was increased by 15%!"

Select the statement which will EXACTLY reproduce the line of text above.

Choice 1

printf("\"My salary was increased by 15/%\!\"\n");

Choice 2

printf("My salary was increased by 15%!\n"); (Ans)

Choice 3

printf("My salary was increased by 15'%'!\n");

Choice 4

printf("\"My salary was increased by 15%%!\"\n");[Ans]

Choice 5

printf("\"My salary was increased by 15'%'!\"\n");

251. What is a difference between a declaration and a definition of a variable?

Choice 1

Both can occur multiple times, but a declaration must occur first.

Choice 2

There is no difference between them.

Choice 3

A definition occurs once, but a declaration may occur many times.

Choice 4

A declaration occurs once, but a definition may occur many times. [Ans]

Choice 5

Both can occur multiple times, but a definition must occur first.

252. int testarray[3][2][2] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};

What value does testarray[2][1][0] in the sample code above contain?

Choice 1 3

Choice 2 5

Choice 3 7

Choice 4 9

Choice 5 11[Ans].

253. int a=10,b;

b=a++ + ++a;

printf("%d,%d,%d,%d",b,a++,a,++a);

What will be the output when following code is executed?

Choice 1 12,10,11,13

Choice 2 22,10,11,13

Choice 3 22,11,11,11

Choice 4 12,11,11,11

Choice 5 22,13,13,13[Ans].

254. int x[] = { 1, 4, 8, 5, 1, 4 };

int *ptr, y;

ptr = x + 4;

y = ptr - x;

What does y in the sample code above equal?

Choice 1 -3

Choice 2 0

Choice 3 4[Ans]

Choice 4 4 + sizeof(int)

Choice 5 4 * sizeof(int)

255. void myFunc (int x)

```
{
  if (x > 0)
  myFunc(--x);
  printf("%d, ", x);

}
int main()
{
  myFunc(5);
  return 0;
}
```

What will the above sample code produce when executed?

Choice 1 1, 2, 3, 4, 5, 5,

Choice 2 4, 3, 2, 1, 0, 0,

Choice 3 5, 4, 3, 2, 1, 0,

Choice 4 0, 0, 1, 2, 3, 4, [Ans]

Choice 5 0, 1, 2, 3, 4, 5,

256. 11 ^ 5

What does the operation shown above produce?

Choice 1 1

Choice 2 6

Choice 3 8

Choice 4 14 [Ans]

Choice 5 15.

257. #define MAX_NUM 15

Referring to the sample above, what is MAX_NUM?

Choice 1 MAX_NUM is an integer variable.

Choice 2 MAX_NUM is a linker constant.

Choice 3 MAX_NUM is a precompiler constant.
Choice 4 MAX_NUM is a preprocessor macro. [Ans]
Choice 5 MAX_NUM is an integer constant.

258. Which one of the following will turn off buffering for stdout?
Choice 1 setbuf(stdout, FALSE);
Choice 2 setvbuf(stdout, NULL);
Choice 3 setbuf(stdout, NULL); [Ans]
Choice 4 setvbuf(stdout, _IONBF);
Choice 5 setbuf(stdout, _IONBF);

259. What is a proper method of opening a file for writing as binary file?
Choice 1 FILE *f = fwrite(“test.bin”, “b”);
Choice 2 FILE *f = fopenb(“test.bin”, “w”);
Choice 3 FILE *f = fopen(“test.bin”, “wb”);
Choice 4 FILE *f = fwriteb(“test.bin”);
Choice 5 FILE *f = fopen(“test.bin”, “bw”); [Ans]

260. Which one of the following functions is the correct choice for moving blocks of binary data that are of arbitrary size and position in memory?
Choice 1 memcpy()
Choice 2 memset()
Choice 3 strncpy()
Choice 4 strcpy()
Choice 5 memmove()[Ans]

261. int x = 2 * 3 + 4 * 5;
What value will x contain in the sample code above?
Choice 1 22
Choice 2 26[Ans]
Choice 3 46
Choice 4 50
Choice 5 70

262. void * array_dup (a, number, size)
const void * a;
size_t number;

```
 size_t size;
{
 void * clone;
 size_t bytes;
  assert(a != NULL);
 bytes = number * size;
 clone = alloca(bytes);
 if (!clone)
   return clone;
 memcpy(clone, a, bytes);
 return clone;

}
```

The function array_dup(), defined above, contains an error. Which one of the following correctly analyzes it?

Choice 1 If the arguments to memcpy() refer to overlapping regions, the destination buffer will be subject to memory corruption.

Choice 2 array_dup() declares its first parameter to be a pointer, when the actual argument will be an array.

Choice 3 The memory obtained from alloca() is not valid in the context of the caller. Moreover, alloca() is nonstandard. [Ans]

Choice 4 size_t is not a Standard C defined type and may not be known to the compiler.

Choice 5 The definition of array_dup() is unusual. Functions cannot be defined using this syntax.

263. int var1;

If a variable has been declared with file scope, as above, can it safely be accessed globally from another file?

Choice 1 Yes; it can be referenced through the register specifier.

Choice 2 No; it would have to have been initially declared as a static variable.

Choice 3 No; it would need to have been initially declared using the global keyword.[Ans]

Choice 4 Yes; it can be referenced through the publish specifier.

Choice 5 Yes; it can be referenced through the extern specifier.

264. time_t t;

Which one of the following statements will properly initialize the variable t with the current time from the sample above?

Choice 1 t = clock();[Ans]

Choice 2 time(&t);

Choice 3 t = ctime();

Choice 4 t = localtime();

Choice 5 None of the above

265. Which one of the following provides conceptual support for function calls?

Choice 1 The system stack[Ans]

Choice 2 The data segment

Choice 3 The processor's registers

Choice 4 The text segment

Choice 5 The heap

266. C is which kind of language?

Choice 1 Machine

Choice 2 Procedural[Ans]

Choice 3 Assembly

Choice 4 Object-oriented

Choice 5 Strictly-typed

267.
```
int i,j;
int ctr = 0;
int myArray[2][3];
for (i=0; i<3; i++)
  for (j=0; j<2; j++)
  {
    myArray[j][i] = ctr;
    ++ctr;
  }
```

What is the value of myArray[1][2]; in the sample code above?

Choice 1 1

Choice 2 2

Choice 3 3
Choice 4 4
Choice 5 5 [Ans] 00,10,01,11,12

268. int x = 0;
for (x=1; x<4; x++);
printf("x=%d\n", x);

What will be printed when the sample code above is executed?
Choice 1 x=0
Choice 2 x=1
Choice 3 x=3
Choice 4 x=4[Ans]
Choice 5 x=5

269. int x = 3;
if(x == 2);
x = 0;
if(x == 3)
x++;
else x += 2;
What value will x contain when the sample code above is executed?
Choice 1 1
Choice 2 2[Ans]
Choice 3 3
Choice 4 4
Choice 5 5

270. char *ptr;
char myString[] = "abcdefg";
ptr = myString;
ptr += 5;
What string does ptr point to in the sample code above?
Choice 1 fg [Ans]/*because string*/
Choice 2 efg

Choice 3 defg
Choice 4 cdefg
Choice 5 None of the above

271. double x = -3.5, y = 3.5;
printf("%.0f : %.0f\n", ceil(x), ceil(y));
printf("%.0f : %.0f\n", floor(x), floor(y));

What will the code above print when executed?
ceil =>rounds up 3.2=4 floor =>rounds down 3.2=3
Choice 1 -3 : 4 -4 : 3 [Ans]
Choice 2 -4 : 4 -3 : 3
Choice 3 -4 : 3 -4 : 3
Choice 4 -4 : 3 -3 : 4
Choice 5 -3 : 3 -4 : 4

272. Which one of the following will declare a pointer to an integer at address 0x200 in memory?
Choice 1 int *x; *x = 0x200;[Ans]
Choice 2 int *x = &0x200;
Choice 3 int *x = *0x200;
Choice 4 int *x = 0x200;
Choice 5 int *x(&0x200);

273.
```
int x = 5;
int y = 2;
char op = '*';
switch (op)
{
 default : x += 1;
 case '+' : x += y; /*It will go to all the cases*/
 case '-' : x -= y;
}
```

After the sample code above has been executed, what value will the variable x contain?
Choice 1 4
Choice 2 5
Choice 3 6 [Ans]

Choice 4 7

Choice 5 8

274. x = 3, counter = 0;

```
while ((x-1))
{
  ++counter;
  x--;
}
```

Referring to the sample code above, what value will the variable counter have when completed?

Choice 1 0

Choice 2 1

Choice 3 2[Ans]

Choice 4 3

Choice 5 4

275. char ** array [12][12][12];

Consider array, defined above. Which one of the following definitions and initializations of p is valid?

Choice 1 char ** (* p) [12][12] = array; [Ans]

Choice 2 char ***** p = array;

Choice 3 char * (* p) [12][12][12] = array;

Choice 4 const char ** p [12][12][12] = array;

Choice 5 char (** p) [12][12] = array;

276. void (*signal(int sig, void (*handler) (int))) (int);

Which one of the following definitions of sighandler_t allows the above declaration to be rewritten as follows:

sighandler_t signal (int sig, sighandler_t handler);

Choice 1 typedef void (*sighandler_t) (int);[Ans]

Choice 2 typedef sighandler_t void (*) (int);

Choice 3 typedef void *sighandler_t (int);

Choice 4 #define sighandler_t(x) void (*x) (int)

Choice 5 #define sighandler_t void (*) (int)

277. All of the following choices represent syntactically correct function definitions. Which one of the following represents a semantically legal function definition in Standard C?

Choice 1 Code:[Ans]

```
int count_digits (const char * buf) {
 assert(buf != NULL);
int cnt = 0, i;
for (i = 0; buf[i] != '\0'; i++)
 if (isdigit(buf[i]))
    cnt++;

return cnt;
}
```

Choice 2 Code:

```
int count_digits (const char * buf) {
 int cnt = 0;
 assert(buf != NULL);
for (int i = 0; buf[i] != '\0'; i++)
 if (isdigit(buf[i]))
    cnt++;
return cnt;
}
```

Choice 3 Code:

```
int count_digits (const char * buf) {
 int cnt = 0, i;
 assert(buf != NULL);
for (i = 0; buf[i] != '\0'; i++)
 if (isdigit(buf[i]))
    cnt++;
return cnt;
}
```

Choice 4 Code:

```
int count_digits (const char * buf) {
 assert(buf != NULL);
```

```
 for (i = 0; buf[i] != '\0'; i++)
  if (isdigit(buf[i]))
     cnt++;
 return cnt;
}
```

Choice 5 Code:

```
int count_digits (const char * buf) {
 assert(buf != NULL);
 int cnt = 0;
 for (int i = 0; buf[i] != '\0'; i++)
  if (isdigit(buf[i]))
     cnt++;
 return cnt;
}
```

278. struct customer *ptr = malloc(sizeof(struct customer));

Given the sample allocation for the pointer "ptr" found above, which one of the following statements is used to reallocate ptr to be an array of 10 elements?

Choice 1 ptr = realloc(ptr, 10 * sizeof(struct customer)); [Ans]

Choice 2 realloc(ptr, 9 * sizeof(struct customer));

Choice 3 ptr += malloc(9 * sizeof(struct customer));

Choice 4 ptr = realloc(ptr, 9 * sizeof(struct customer));

Choice 5 realloc(ptr, 10 * sizeof(struct customer));

279. Which one of the following is a true statement about pointers?

Choice 1 Pointer arithmetic is permitted on pointers of any type.

Choice 2 A pointer of type void * can be used to directly examine or modify an object of any type.

Choice 3 Standard C mandates a minimum of four levels of indirection accessible through a pointer.

Choice 4 A C program knows the types of its pointers and indirectly referenced data items at runtime. [Ans]

Choice 5 Pointers may be used to simulate call-by-reference.

280. Which one of the following functions returns the string representation from a pointer to a time_t value?

Choice 1 localtime

Choice 2 gmtime

Choice 3 strtime [Ans]

Choice 4 asctime

Choice 5 ctime

281.
```
short testarray[4][3] = { {1}, {2, 3}, {4, 5, 6} };
printf( "%d\n", sizeof( testarray ) );
```

Assuming a short is two bytes long, what will be printed by the above code?

Choice 1 It will not compile because not enough initializers are given.

Choice 2 6

Choice 3 7

Choice 4 12

Choice 5 24 [Ans]

282.
```
char buf [] = "Hello world!";
char * buf = "Hello world!";
```

In terms of code generation, how do the two definitions of buf, both presented above, differ?

Choice 1 The first definition certainly allows the contents of buf to be safely modified at runtime; the second definition does not.

Choice 2 The first definition is not suitable for usage as an argument to a function call; the second definition is.

Choice 3 The first definition is not legal because it does not indicate the size of the array to be allocated; the second definition is legal.

Choice 4 They do not differ -- they are functionally equivalent. [Ans]

Choice 5 The first definition does not allocate enough space for a terminating NULL character, nor does it append one; the second definition does.

283. In a C expression, how is a logical AND represented?

Choice 1 @@

Choice 2 ||

Choice 3 .AND.

Choice 4 && [Ans]

Choice 5 .AND

284. How do printf()'s format specifiers %e and %f differ in their treatment of floating-point numbers?

Choice 1 %e always displays an argument of type double in engineering notation; %f always displays an argument of type double in decimal notation. [Ans]

Choice 2 %e expects a corresponding argument of type double; %f expects a corresponding argument of type float.

Choice 3 %e displays a double in engineering notation if the number is very small or very large. Otherwise, it behaves like %f and displays the number in decimal notation.

Choice 4 %e displays an argument of type double with trailing zeros; %f never displays trailing zeros.

Choice 5 %e and %f both expect a corresponding argument of type double and format it identically. %e is left over from K&R C; Standard C prefers %f for new code.

285. Which one of the following Standard C functions can be used to reset end-of-file and error conditions on an open stream?

Choice 1 clearerr()

Choice 2 fseek()(Ans)

Choice 3 ferror()

Choice 4 feof()

Choice 5 setvbuf()

286. Which one of the following will read a character from the keyboard and will store it in the variable c?

Choice 1 c = getc();

Choice 2 getc(&c);

Choice 3 c = getchar(stdin);

Choice 4 getchar(&c)

Choice 5 c = getchar(); [Ans]

287.
```
#include <stdio.h>
int i;
void increment( int i )
{

  i++;
}
```

```
int main()
{
  for( i = 0; i < 10; increment( i ) )
  {
  }
  printf("i=%d\n", i);
  return 0;
}
```

What will happen when the program above is compiled and executed?

Choice 1 It will not compile.

Choice 2 It will print out: i=9.

Choice 3 It will print out: i=10.

Choice 4 It will print out: i=11.

Choice 5 It will loop indefinitely.[Ans]

288.
```
char ptr1[] = "Hello World";
char *ptr2 = malloc( 5 );
ptr2 = ptr1;
```

What is wrong with the above code (assuming the call to malloc does not fail)?

Choice 1 There will be a memory overwrite.

Choice 2 There will be a memory leak.

Choice 3 There will be a segmentation fault.

Choice 4 Not enough space is allocated by the malloc. (Ans)

Choice 5 It will not compile.

289.
```
int i = 4;
switch (i)
{
  default:
    ;
  case 3:

    i += 5;
    if ( i == 8)
```

```
    {
      i++;
      if (i == 9) break;
      i *= 2;
    }
    i -= 4;
    break;
  case 8:
    i += 5;
    break;
}

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

What will the output of the sample code above be?

Choice 1 i = 5[Ans]
Choice 2 i = 8
Choice 3 i = 9
Choice 4 i = 10
Choice 5 i = 18

290. Which one of the following C operators is right associative?

Choice 1 = [Ans]
Choice 2 ,
Choice 3 []
Choice 4 ^
Choice 5 ->

291. What does the "auto" specifier do?

Choice 1 It automatically initializes a variable to 0;.
Choice 2 It indicates that a variable's memory will automatically be preserved.[Ans]
Choice 3 It automatically increments the variable when used.
Choice 4 It automatically initializes a variable to NULL.
Choice 5 It indicates that a variable's memory space is allocated upon entry into the block.

292. How do you include a system header file called sysheader.h in a C source file?

Choice 1 #include <sysheader.h> [Ans]

Choice 2 #incl "sysheader.h"

Choice 3 #includefile <sysheader>

Choice 4 #include sysheader.h

Choice 5 #incl <sysheader.h>

293. Which one of the following printf() format specifiers indicates to print a double value in decimal notation, left aligned in a 30-character field, to four (4) digits of precision?

Choice 1 %-30.4e

Choice 2 %4.30e

Choice 3 %-4.30f

Choice 4 %-30.4f [Ans] decimal notation

Choice 5 %#30.4f

294.
```
int x = 0;
for ( ; ; )
{
 if (x++ == 4)
  break;
 continue;

}
printf("x=%d\n", x);
```

What will be printed when the sample code above is executed?

Choice 1 x=0

Choice 2 x=1

Choice 3 x=4

Choice 4 x=5[Ans]

Choice 5 x=6

295. According to the Standard C specification, what are the respective minimum sizes (in bytes) of the following three data types: short; int; and long?

Choice 1 1, 2, 2

Choice 2 1, 2, 4

Choice 3 1, 2, 8

Choice 4 2, 2, 4[Ans]
Choice 5 2, 4, 8

296. int y[4] = {6, 7, 8, 9};
int *ptr = y + 2;
printf("%d\n", ptr[1]); /*ptr+1 == ptr[1]*/

What is printed when the sample code above is executed?
Choice 1 6
Choice 2 7
Choice 3 8
Choice 4 9[Ans]
Choice 5 The code will not compile.

297. **penny = one**
nickel = five
dime = ten
quarter = twenty-five
How is enum used to define the values of the American coins listed above?
Choice 1 enum coin {(penny,1), (nickel,5), (dime,10), (quarter,25)};
Choice 2 enum coin ({penny,1}, {nickel,5}, {dime,10}, {quarter,25});
Choice 3 enum coin {penny=1,nickel=5,dime=10,quarter=25};[Ans]
Choice 4 enum coin (penny=1,nickel=5,dime=10,quarter=25);
Choice 5 enum coin {penny, nickel, dime, quarter} (1, 5, 10, 25);

298. char txt [20] = "Hello world!\0";
How many bytes are allocated by the definition above?
Choice 1 11 bytes
Choice 2 12 bytes
Choice 3 13 bytes
Choice 4 20 bytes[Ans]
Choice 5 21 bytes

299. int i = 4;
int x = 6;
double z;

```
z =  x / i;
printf("z=%.2f\n", z);
```

What will print when the sample code above is executed?

Choice 1 z=0.00

Choice 2 z=1.00[Ans]

Choice 3 z=1.50

Choice 4 z=2.00

Choice 5 z=NULL

300. Which one of the following variable names is NOT valid?

Choice 1 go_cart

Choice 2 go4it

Choice 3 4season[Ans]

Choice 4 run4

Choice 5 _what

301. int a [8] = { 0, 1, 2, 3 };

The definition of a above explicitly initializes its first four elements. Which one of the following describes how the compiler treats the remaining four elements?

Choice 1 Standard C defines this particular behavior as implementation-dependent. The compiler writer has the freedom to decide how the remaining elements will be handled.

Choice 2 The remaining elements are initialized to zero(0).[Ans]

Choice 3 It is illegal to initialize only a portion of the array. Either the entire array must be initialized, or no part of it may be initialized.

Choice 4 As with an enum, the compiler assigns values to the remaining elements by counting up from the last explicitly initialized element. The final four elements will acquire the values 4, 5, 6, and 7, respectively.

Choice 5 They are left in an uninitialized state; their values cannot be relied upon.

302. Which one of the following is a true statement about pointers?

Choice 1 They are always 32-bit values.

Choice 2 For efficiency, pointer values are always stored in machine registers. (Ans)

Choice 3 With the exception of generic pointers, similarly typed pointers may be subtracted from each other.

Choice 4 A pointer to one type may not be cast to a pointer to any other type.

Choice 5 With the exception of generic pointers, similarly typed pointers may be added to each other.

303. Which one of the following statements allocates enough space to hold an array of 10 integers that are initialized to 0?

Choice 1 int *ptr = (int *) malloc(10, sizeof(int));

Choice 2 int *ptr = (int *) calloc(10, sizeof(int));

Choice 3 int *ptr = (int *) malloc(10*sizeof(int)); [Ans]

Choice 4 int *ptr = (int *) alloc(10*sizeof(int));

Choice 5 int *ptr = (int *) calloc(10*sizeof(int));

304. What are two predefined FILE pointers in C?

Choice 1 stdout and stderr

Choice 2 console and error

Choice 3 stdout and stdio [Ans]

Choice 4 stdio and stderr

Choice 5 errout and conout

305.
```
u32 X (f32 f)
{
 union {
       f32 f;
       u32 n;

      } u;
 u.f = f;
 return u.n;
}
```

Given the function X(), defined above, assume that u32 is a type-definition indicative of a 32-bit unsigned integer and that f32 is a type-definition indicative of a 32-bit floating-point number. Which one of the following describes the purpose of the function defined above?

Choice 1 X() effectively rounds f to the nearest integer value, which it returns.

Choice 2 X() effectively performs a standard typecast and converts f to a roughly equivalent integer. [Ans]

Choice 3 X() preserves the bit-pattern of f, which it returns as an unsigned integer of equal size.

Choice 4 Since u.n is never initialized, X() returns an undefined value. This function is therefore a primitive pseudorandom number generator.

Choice 5 Since u.n is automatically initialized to zero (0) by the compiler, X() is an obtuse way of always obtaining a zero (0) value.

306.
```
long factorial (long x)
{
 ????
 return x * factorial(x - 1);
}
```

With what do you replace the ???? to make the function shown above return the correct answer?

Choice 1 if (x == 0) return 0;

Choice 2 return 1;

Choice 3 if (x >= 2) return 2;

Choice 4 if (x == 0) return 1;

Choice 5 if (x <= 1) return 1; [Ans]{more probable}

307.
```
/* Increment each integer in the array 'p' of * size 'n'. */

void increment_ints (int p [/*n*/], int n)
{
  assert(p != NULL);  /* Ensure that 'p' isn't a null pointer. */
  assert(n >= 0);  /* Ensure that 'n' is nonnegative. */
  while (n)  /* Loop over 'n' elements of 'p'. */

  {
    *p++;          /* Increment *p. */
    p++, n--;      /* Increment p, decrement n. */
  }
}
```

Consider the function increment_ints(), defined above. Despite its significant inline commentary, it contains an error. Which one of the following correctly assesses it?

Choice 1 *p++ causes p to be incremented before the dereference is performed, because both operators have equal precedence and are right associative.

Choice 2 An array is a nonmodifiable lvalue, so p cannot be incremented directly. A navigation pointer should be used in conjunction with p.

Choice 3 *p++ causes p to be incremented before the dereference is performed, because the autoincrement operator has higher precedence than the indirection operator. (Ans)

Choice 4 The condition of a while loop must be a Boolean expression. The condition should be n != 0.

Choice 5 An array cannot be initialized to a variable size. The subscript n should be removed from the definition of the parameter p.

308. How is a variable accessed from another file?

Choice 1 The global variable is referenced via the extern specifier.[Ans]

Choice 2 The global variable is referenced via the auto specifier.

Choice 3 The global variable is referenced via the global specifier.

Choice 4 The global variable is referenced via the pointer specifier.

Choice 5 The global variable is referenced via the ext specifier.

309. When applied to a variable, what does the unary "&" operator yield?

Choice 1 The variable's value

Choice 2 The variable's binary form

Choice 3 The variable's address [Ans]

Choice 4 The variable's format

Choice 5 The variable's right value

310.
```
/* sys/cdef.h */
#if     defined(__STDC__) || defined(__cplusplus)
#define __P(protos)    protos
#else
#define __P(protos)    ()
#endif
/* stdio.h */
#include <sys/cdefs.h>
div_t div __P((int, int));
```

The code above comes from header files for the FreeBSD implementation of the C library. What is the primary purpose of the __P() macro?

Choice 1 The __P() macro has no function and merely obfuscates library function declarations. It should be removed from further releases of the C library.

Choice 2 The __P() macro provides forward compatibility for C++ compilers, which do not recognize Standard C prototypes.

Choice 3 Identifiers that begin with two underscores are reserved for C library implementations. It is impossible to determine the purpose of the macro from the context given.

Choice 4 The __P() macro provides backward compatibility for K&R C compilers, which do not recognize Standard C prototypes. [Ans]

Choice 5 The __P() macro serves primarily to differentiate library functions from application-specific functions.

311. Which one of the following is NOT a valid identifier?

Choice 1 __ident

Choice 2 auto [Ans]

Choice 3 bigNumber

Choice 4 g42277

Choice 5 peaceful_in_space

312.
```
int read_long_string (const char ** const buf) {
 char * p = NULL;
 const char * fwd = NULL;
 size_t len = 0;
 assert(buf);
 do
 {
  p = realloc(p, len += 256);
  if (!p)
   return 0;
  if (!fwd)
   fwd = p;
  else
   fwd = strchr(p, '\0');
```

```
 } while (fgets(fwd, 256, stdin));
 *buf = p;
 return 1;
}
```

The function read_long_string(), defined above, contains an error that may be particularly visible under heavy stress. Which one of the following describes it?

Choice 1 The write to *buf is blocked by the const qualifications applied to its type.

Choice 2 If the null pointer for char is not zero-valued on the host machine, the implicit comparisons to zero (0) may introduce undesired behavior. Moreover, even if successful, it introduces machine-dependent behavior and harms portability.

Choice 3 The symbol stdin may not be defined on some ANCI C compliant systems.

Choice 4 The else causes fwd to contain an errant address. (Ans)

Choice 5 If the call to realloc() fails during any iteration but the first, all memory previously allocated by the loop is leaked.

313.
```
FILE *f = fopen( fileName, "r" );
readData( f );
if( ???? )

{
 puts( "End of file was reached" );
}
```

Which one of the following can replace the ???? in the code above to determine if the end of a file has been reached?

Choice 1 f == EOF[Ans]

Choice 2 feof(f)

Choice 3 eof(f)

Choice 4 f == NULL

Choice 5 !f

314. Global variables that are declared static are _____.

Which one of the following correctly completes the sentence above?

Choice 1 Deprecated by Standard C

Choice 2 Internal to the current translation unit

Choice 3 Visible to all translation units

Choice 4 Read-only subsequent to initialization

Choice 5 Allocated on the heap[Ans]

```
double read_double (FILE * fp) {
 double d;
  assert(fp != NULL);

  fscanf(fp, " %lf ", d);
  return d;
}
```

The code above contains a common error. Which one of the following describes it?

Choice 1 fscanf() will fail to match floating-point numbers not preceded by whitespace.

Choice 2 The format specifier %lf indicates that the corresponding argument should be long double rather than double. (Ans)

Choice 3 The call to fscanf() requires a pointer as its last argument.

Choice 4 The format specifier %lf is recognized by fprintf() but not by fscanf().

Choice 5 d must be initialized prior to usage.

315. Which one of the following is NOT a valid C identifier?

Choice 1 ___S

Choice 2 1___ [Ans]

Choice 3 ___1

Choice 4 ___

Choice 5 S___

316. According to Standard C, what is the type of an unsuffixed floating-point literal, such as 123.45?

Choice 1 long double

Choice 2 Unspecified

Choice 3 float[Ans]

Choice 4 double

Choice 5 signed float

317. Which one of the following is true for identifiers that begin with an underscore?

Choice 1 They are generally treated differently by preprocessors and compilers from other identifiers.

Choice 2 They are case-insensitive. (Ans)

Choice 3 They are reserved for usage by standards committees, system implementers, and compiler engineers.

Choice 4 Applications programmers are encouraged to employ them in their own code in order to mark certain symbols for internal usage.

Choice 5 They are deprecated by Standard C and are permitted only for backward compatibility with older C libraries.

318. Which one of the following is valid for opening a read-only ASCII file?

Choice 1 fileOpen (filenm, "r");

Choice 2 fileOpen (filenm, "ra");

Choice 3 fileOpen (filenm, "read");

Choice 4 fopen (filenm, "read");

Choice 5 fopen (filenm, "r");[Ans]

319. f = fopen(filename, "r");

Referring to the code above, what is the proper definition for the variable f?

Choice 1 FILE f;

Choice 2 FILE *f;[Ans]

Choice 3 int f;

Choice 4 struct FILE f;

Choice 5 char *f;

320. If there is a need to see output as soon as possible, what function will force the output from the buffer into the output stream?

Choice 1 flush()

Choice 2 output()(Ans)

Choice 3 fflush()

Choice 4 dump()

Choice 5 write()

321. short int x; /* assume x is 16 bits in size */

What is the maximum number that can be printed using printf("%d\n", x), assuming that x is initialized as shown above?

Choice 1 127

Choice 2 128

Choice 3 255

Choice 4 32,767 [Ans]

Choice 5 65,536

322.
```
void crash (void)
{
 printf(“got here”);
 *((char *) 0) = 0;

}
```

The function crash(), defined above, triggers a fault in the memory management hardware for many architectures. Which one of the following explains why "got here" may NOT be printed before the crash?

Choice 1 The C standard says that dereferencing a null pointer causes undefined behavior. This may explain why printf() apparently fails.

Choice 2 If the standard output stream is buffered, the library buffers may not be flushed before the crash occurs. (Ans)

Choice 3 printf() always buffers output until a newline character appears in the buffer. Since no newline was present in the format string, nothing is printed.

Choice 4 There is insufficient information to determine why the output fails to appear. A broader context is required.

Choice 5 printf() expects more than a single argument. Since only one argument is given, the crash may actually occur inside printf(), which explains why the string is not printed. puts() should be used instead.

323.
```
char * dwarves [] = {
 “Sleepy”,
 “Dopey” “Doc”,
 “Happy”,
 “Grumpy” “Sneezy”,
```

```
 "Bashful",
};
```

How many elements does the array dwarves (declared above) contain? Assume the C compiler employed strictly complies with the requirements of Standard C.

Choice 1 4

Choice 2 5[Ans]

Choice 3 6

Choice 4 7

Choice 5 8

324. Which one of the following can be used to test arrays of primitive quantities for strict equality under Standard C?

Choice 1 qsort()

Choice 2 bcmp() [Ans]

Choice 3 memcmp()

Choice 4 strxfrm()

Choice 5 bsearch()

325.
```
int debug (const char * fmt, ...) {
 extern FILE * logfile;
 va_list args;
  assert(fmt);
 args = va_arg(fmt, va_list);
return vfprintf(logfile, fmt, args);
}
```

The function debug(), defined above, contains an error. Which one of the following describes it?

Choice 1 The ellipsis is a throwback from K&R C. In accordance with Standard C, the declaration of args should be moved into the parameter list, and the K&R C macro va_arg() should be deleted from the code.

Choice 2 vfprintf() does not conform to ISO 9899: 1990, and may not be portable.

Choice 3 Library routines that accept argument lists cause a fault on receipt of an empty list. The argument list must be validated with va_null() before invoking vfprintf().

Choice 4 The argument list args has been improperly initialized. [Ans]

Choice 5 Variadic functions are discontinued by Standard C; they are legacy constructs from K&R C, and no longer compile under modern compilers.

326.
```
char *buffer = "0123456789";
char *ptr = buffer;
ptr += 5;
printf( "%s\n", ptr );
printf( "%s\n", buffer );
```

What will be printed when the sample code above is executed?

Choice 1 0123456789 56789
Choice 2 5123456789 5123456789
Choice 3 56789 56789
Choice 4 0123456789 0123456789
Choice 5 56789 0123456789 [Ans]

327.
```
char * get_rightmost (const char * d)
{
 char rightmost [MAXPATHLEN];
 const char * p = d;
 assert(d != NULL);
 while (*d != '\0')
 {
  if (*d == '/')
   p = (*(d + 1) != '\0') ? d + 1 : p;
  d++;
 }
 memset(rightmost, 0, sizeof(rightmost));
 memcpy(rightmost, p, strlen(p) + 1);
 return rightmost;
}
```

The function get_rightmost(), defined above, contains an error. Which one of the following describes it?

Choice 1 The calls to memset() and memcpy() illegally perform a pointer conversion on rightmost without an appropriate cast.

Choice 2 The code does not correctly handle the situation where a directory separator '/' is the final character. [Ans]

Choice 3 The if condition contains an incorrectly terminated character literal.

Choice 4 memcpy() cannot be used safely to copy string data.

Choice 5 The return value of get_rightmost() will be invalid in the caller's context.

328. What number is equivalent to -4e3?

Choice 1 -4000 [Ans]

Choice 2 -400

Choice 3 -40

Choice 4 .004

Choice 5 .0004

329.
```
void freeList( struct node *n )
{
 while( n )
 {
  ????
 }
}
```

Which one of the following can replace the ???? for the function above to release the memory allocated to a linked list?

Choice 1 n = n->next; free(n);

Choice 2 struct node m = n; n = n->next; free(m);

Choice 3 struct node m = n; free(n); n = m->next;

Choice 4 free(n); n = n->next; [Ans]

Choice 5 struct node m = n; free(m); n = n->next; (Ans)

330. How does variable definition differ from variable declaration?

Choice 1 Definition allocates storage for a variable, but declaration only informs the compiler as to the variable's type.

Choice 2 Declaration allocates storage for a variable, but definition only informs the compiler as to the variable's type.

Choice 3 Variables may be defined many times, but may be declared only once.[Ans]

Choice 4 Variable definition must precede variable declaration.

Choice 5 There is no difference in C between variable declaration and variable definition.

331. int x[] = {1, 2, 3, 4, 5};

```
int u;
int *ptr = x;
????
for( u = 0; u < 5; u++ )
{
 printf("%d-", x[u]);
}
printf( "\n" );
```

Which one of the following statements could replace the ???? in the code above to cause the string 1-2-3-10-5- to be printed when the code is executed?

Choice 1 *ptr + 3 = 10;

Choice 2 *ptr[3] = 10;

Choice 3 *(ptr + 3) = 10;[Ans]

Choice 4 (*ptr)[3] = 10;

Choice 5 *(ptr[3]) = 10;

332.
```
/*sum_array() has been ported from FORTRAN. No * logical changes have been made*/
double sum_array(double d[],int n)
{
 register int i;
 double total=0;
  assert(d!=NULL);
 for(i=l;i<=n;i++)
  total+=d[i];
 return total;
}
```

The function sum_array(), defined above, contains an error. Which one of the following accurately describes it?

Choice 1 The range of the loop does not match the bounds of the array d.
Choice 2 The loop processes the incorrect number of elements.
Choice 3 total is initialized with an integer literal. The two are not compatible in an assignment. [Ans]
Choice 4 The code above fails to compile if there are no registers available for i.
Choice 5 The formal parameter d should be declared as double * d to allow dynamically allocated arrays as arguments.

333.
```
#include <stdio.h>
void func()
{
  int x = 0;
  static int y = 0;

  x++; y++;
  printf( "%d -- %d\n", x, y );
}

int main()
{
  func();
  func();
  return 0;
}
```

What will the code above print when it is executed?
Choice 1 1 -- 1 1 -- 1
Choice 2 1 -- 1 2 -- 1
Choice 3 1 -- 1 2 -- 2
Choice 4 1 -- 0 1 -- 0
Choice 5 1 -- 1 1 -- 2 [Ans]

334.
```
int fibonacci (int n)
{
 switch (n)
```

```
{
 default:
    return (fibonacci(n - 1) + fibonacci(n - 2));

 case 1:
 case 2:
}
 return 1;
}
```

The function above has a flaw that may result in a serious error during some invocations. Which one of the following describes the deficiency illustrated above?

Choice 1 For some values of n, the environment will almost certainly exhaust its stack space before the calculation completes.[Ans]

Choice 2 An error in the algorithm causes unbounded recursion for all values of n.

Choice 3 A break statement should be inserted after each case. Fall-through is not desirable here.

Choice 4 The fibonacci() function includes calls to itself. This is not directly supported by Standard C due to its unreliability.

Choice 5 Since the default case is given first, it will be executed before any case matching n.

335. When applied to a variable, what does the unary "&" operator yield?

Choice 1 The variable's address [Ans]

Choice 2 The variable's right value

Choice 3 The variable's binary form

Choice 4 The variable's value

Choice 5 The variable's format

336. short int x; /* assume x is 16 bits in size */

What is the maximum number that can be printed using printf("%d\n", x), assuming that x is initialized as shown above?

Choice 1 127

Choice 2 128

Choice 3 255

Choice 4 32,767[Ans]

Choice 5 65,536

337. int x = 011 | 0x10;

What value will x contain in the sample code above?

Choice 1 3

Choice 2 13 [Ans]

Choice 3 19

Choice 4 25

Choice 5 27

338. Which one of the following calls will open the file test.txt for reading by fgetc?

Choice 1 fopen("test.txt", "r");

Choice 2 read("test.txt")

Choice 3 fileopen("test.txt", "r"); [Ans]

Choice 4 fread("test.txt")

Choice 5 freopen("test.txt")

339. int m = – 14;

int n = 6;

int o;

o = m % ++n;

n += m++ – o;

m <<= (o ^ n) & 3;

Assuming two's-complement arithmetic, which one of the following correctly represents the values of m, n, and o after the execution of the code above?

Choice 1 m = – 26, n = – 7, o = 0

Choice 2 m = – 52, n = – 4, o = – 2 (Ans)

Choice 3 m = – 26, n = – 5, o = – 2

Choice 4 m = – 104, n = – 7, o = 0

Choice 5 m = – 52, n = – 6, o = 0

340. How do printf()'s format specifiers %e and %f differ in their treatment of floating-point numbers?

Choice 1 %e displays an argument of type double with trailing zeros; %f never displays trailing zeros.

Choice 2 %e displays a double in engineering notation if the number is very small or very large. Otherwise, it behaves like %f and displays the number in decimal notation.

Choice 3 %e always displays an argument of type double in engineering notation; %f always displays an argument of type double in decimal notation. [Ans]

Choice 4 %e expects a corresponding argument of type double; %f expects a corresponding argument of type float.

Choice 5 %e and %f both expect a corresponding argument of type double and format it identically. %e is left over from K&R C; Standard C prefers %f for new code.

341. $$Except 1 all choices are O.K.$$

c = getchar();

What is the proper declaration for the variable c in the code above?

Choice 1 char *c;

Choice 2 unsigned int c;

Choice 3 int c;

Choice 4 unsigned char c;

Choice 5 char c;[Ans]

342. Which one of the following will define a function that CANNOT be called from another source file?

Choice 1 void function() { ... }

Choice 2 extern void function() { ... }

Choice 3 const void function() { ... }

Choice 4 private void function() { ... } [Ans]

Choice 5 static void function() { ... }

Chapter 5 SAMPLE QUESTIONS

1. Base class has some virtual method and derived class has a method with the same name. If we initialize the base class pointer with derived object, calling of that virtual method will result in which method being called?

(*a*) Base method (*b*) Derived method.

Ans. (*b*)

2. For the following C program

```
#define AREA(x)(3.14*x*x)
main()
{float r1=6.25,r2=2.5,a;
a=AREA(r1);
printf("\n Area of the circle is %f ", a);
a=AREA(r2);
printf("\n Area of the circle is %f ", a);
}
```

What is the output?

Ans. Area of the circle is 122.656250

Area of the circle is 19.625000

3. Write a C program to find the sum of numbers between 1 and n.

Ans. **Program:**

```
main()
{
    int n, i, s = 0;
    scanf("%d", &n);
for (i = 1; i<=n, i++)
    s = s + i;
Printf ("\n sum = %.d", s);
}
```

4.
```
void main()
{
int d=5;
printf("%f ",d);
}
```
Ans. Undefined

5.
```
void main()
{
int i;
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;
}
switch(i) case 4:printf("%d",i);
}
```
Ans. 1,2,3,4

6.
```
void main()
{
char *s="\12345s\n";
printf("%d",sizeof(s));
}
```
Ans. 6

7.
```
void main()
{
unsigned i=1; /* unsigned char k= -1 => k=255; */
signed j=-1; /* char k= -1 => k=65535 */
/* unsigned or signed int k= -1 =>k=65535 */
if(i<j)
printf("less");
else
if(i>j)
```

```
printf("greater");
else
if(i= =j)
printf("equal");
}
```

Ans. less

8.
```
void main()
{
float j;
j=1000*1000;
printf("%f ",j);
}
```

1. 1000000
2. Overflow
3. Error
4. None

Ans. 4

9. How do you declare an array of N pointers to functions returning pointers to functions returning pointers to characters?

Ans. The first part of this question can be answered in at least three ways:

1. `char *(*(*a[N])())();`

2. Build the declaration up incrementally, using typedefs:

```
typedef char *pc;       /* pointer to char */
typedef pc fpc();       /* function returning pointer to char */
typedef fpc *pfpc;      /* pointer to above */
typedef pfpc fpfpc();   /* function returning... */
typedef fpfpc *pfpfpc;  /* pointer to... */
pfpfpc a[N];            /* array of... */
```

3. Use the cdecl program, which turns English into C and vice-versa:

cdecl> declare a as array of pointer to function returning pointer to function returning pointer to char

```
char *(*(*a[])())()
```

cdecl can also explain complicated declarations, help with casts, and indicate which set of parentheses the arguments go in (for complicated function definitions, like the one above). Any good book on C should explain how to read these complicated C declarations "inside out" to understand them ("declaration mimics use").

The pointer-to-function declarations in the examples above have not included parameter type information. When the parameters have complicated types, declarations can *really* get messy.

(Modern versions of cdecl can help here, too.)

10. A structure pointer is defined of the type time . With 3 fields min,sec hours having pointers to intergers.

Write the way to initialize the 2nd element to 10.

11. In the above question an array of pointers is declared.

Write the statement to initialize the 3rd element of the 2 element to 10;

12.
```
int f()
void main()
{
f(1);
f(1,2);
f(1,2,3);
}
f(int i,int j,int k)
{
printf("%d %d %d",i,j,k);
}
```
What are the number of syntax errors in the above?

Ans. None.

13.
```
void main()
{
int i=7;
printf("%d",i++*i++);
}
```
Ans. 56

14.
```
#define one 0
#ifdef one
printf("one is defined ");
#ifndef one
printf("one is not defined ");
```
Ans. "one is defined"

15.
```
void main()
{
int count=10,*temp,sum=0;
temp=&count;
```

```
*temp=20;
temp=&sum;
*temp=count;
printf("%d %d %d ",count,*temp,sum);
}
```

Ans. 20 20 20

16. What is alloca()?

Ans. It allocates and frees memory after use/after getting out of scope.

17.
```
main()
{
static i=3;
printf("%d",i--);
return i>0 ? main():0;
}
```

Ans. 321

18.
```
char *foo()
{
char result[100]);
strcpy(result,"anything is good");
return(result);
}
void main()
{
char *j;
j=foo()
printf("%s",j);
}
```

Ans. anything is good.

19.
```
void main()
{
char *s[]={ "dharma","hewlett-packard","siemens","ibm"};
char **p;
p=s;
printf("%s",++*p);
```

```
printf("%s",*p++);
printf("%s",++*p);
}
```

Ans. "harma" (p->add(dharma) && (*p)->harma)
"harma" (after printing, p->add(hewlett-packard) &&(*p)->harma)
"ewlett-packard"

20. Output of the following program is

```
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);
}
}
```

(*a*) 0,5,9,13,17 (*b*) 5,9,13,17
(*c*) 12,17,22 (*d*) 16,21
(*e*) Syntax error

Ans. (*d*)

21. What is the ouptut in the following program?

```
main()
{char c=-64;
int i=-32
unsigned int u =-16;
if(c>i)
{printf("pass1,");
if(c<u)
printf("pass2");
else
printf("Fail2");
}
```

```
else
printf("Fail1);
if(i<u)
printf("pass2");
else
printf("Fail2")
}
```

(*a*) Pass1,Pass2 (*b*) Pass1,Fail2

(*c*) Fail1,Pass2 (*d*) Fail1,Fail2

(*e*) None of these

Ans. (*c*)

22. What will the following program do?

```
void main()
{
int i;
char a[]="String";
char *p="New Sring";
char *Temp;
Temp=a;
a=malloc(strlen(p) + 1);
strcpy(a,p); //Line number:9//
p = malloc(strlen(Temp) + 1);
strcpy(p,Temp);
printf("(%s, %s)",a,p);
free(p);
free(a);
} //Line number 15//
```

(*a*) Swap contents of p & a and print:(New string, string)

(*b*) Generate compilation error in line number 8

(*c*) Generate compilation error in line number 5

(*d*) Generate compilation error in line number 7

(*e*) Generate compilation error in line number 1

Ans. (*b*)

23. In the following code segment what will be the result of the function,

value of x , value of y

```
{unsigned int x=-1;
int y;
y = ~0;
if(x == y)
printf("same");
else
printf("not same");
}
```

(*a*) same, MAXINT, -1 (*b*) not same, MAXINT, -MAXINT

(*c*) same , MAXUNIT, -1 (*d*) same, MAXUNIT, MAXUNIT

(*e*) not same, MAXINT, MAXUNIT

Ans. (*a*)

24. What will be the result of the following program ?

```
char *gxxx()
{static char xxx[1024];
return xxx;
}
main()
        {char *g="string";
        strcpy(gxxx(),g);
g = gxxx();
strcpy(g,"oldstring");
printf("The string is : %s",gxxx());
}
```

(*a*) The string is : string (*b*) The string is :Oldstring

(*c*) Run time error/Core dump (*d*) Syntax error during compilation

(*e*) None of these

Ans. (*b*)

25. Find the output for the following C program.

```
main()
{
char *p1="Name";
```

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

Ans. An empty string

26. Find the output for the following C program.

```
main()
{
int x=20,y=35;
x = y++ + x++;
y = ++y + ++x;
printf("%d %d\n",x,y);
}
```

Ans. 57 94

27. Find the output for the following C program.

```
main()
{
int x=5;
printf("%d %d %d\n",x,x<<2,x>>2);
}
```

Ans. 5 20 1

28. Find the output for the following C program.

```
#define swap1(a,b) a=a+b;b=a-b;a=a-b;
main()
{
int x=5,y=10;
swap1(x,y);
printf("%d %d\n",x,y);
swap2(x,y);
printf("%d %d\n",x,y);
}
int swap2(int a,int b)
```

```
{
int temp;
temp=a;
b=a;
a=temp;
return;
}
```

Ans. 10 5

29. Find the output for the following C program.

```
main()
{
char *ptr = “Ramco Systems”;
(*ptr)++;
printf(“%s\n”,ptr);
ptr++;
printf(“%s\n”,ptr);
}
```

Ans. Samco Systems

30. Find the output for the following C program.

```
#include<stdio.h>
main()
{
char s1[]=“Ramco”;
char s2[]=“Systems”;
s1=s2;
printf(“%s”,s1);
}
```

Ans. Compilation error giving it cannot be an modifiable 'lvalue'

31. Find the output for the following C program.

```
#include<stdio.h>
main()
{
char *p1;
char *p2;
```

```
p1=(char *) malloc(25);
p2=(char *) malloc(25);
strcpy(p1,“Ramco”);
strcpy(p2,“Systems”);
strcat(p1,p2);
printf(“%s”,p1);
}
```

Ans. RamcoSystems

32. Write a function in C to calculate n!

Ans.
```
int fact (int k)
{
int P = 1, i;
for (i = 1; i< = k; i++)
P = p*i;
return (P);
}
```

33. Find the output for the following C program.

```
# define TRUE 0
some code
while(TRUE)
{
some code
}
```

Ans. This won’t go into the loop as TRUE is defined as 0

34.
```
struct list{
int x;
struct list *next;
}*head;
```
the struct head.x =100

Is the above assignment to pointer is correct or wrong ?

Ans. Wrong

35. What is the output of the following ?

```
int i;
```

```
i=1;
i=i+2*i++;
printf(%d,i);
```

Ans. 4

36.
```
FILE *fp1,*fp2;
fp1=fopen("one","w")
fp2=fopen("one","w")
fputc('A',fp1)
fputc('B',fp2)
fclose(fp1)
fclose(fp2)
}
```

Find the Error, If Any?

Ans. no error. But It will over writes on same file.

37. What is the output for the following ?

Program:

```
main()
{
int x = 5;
while (x= =1)
x = x – 1;
printf ("%d\n", x);
}
```

Output:

The while loop will not be executed because the condition is false and the value of x will be printed.

x = 5.

38. Write a C program to find the square root of a given number.

```
main()
{
    float x;
    scanf("%f ", &x);
    printf("\n square root of %.6.2f is %6.2f ", x, sqrt(x));
}
```

39. #define MAN(x,y) (x)>(y)?(x):(y)

```
{int i=10;
j=5;
k=0;
k=MAX(i++,++j);
printf(%d %d %d %d,i,j,k);
}
```

Ans. 10 5 0

40.

```
void main()
{
int i=7;
printf("%d",i++*i++);
}
```

Ans. 56

Chapter 6 SHORT QUESTIONS AND ANSWERS

1. What is C language?

Ans. The C programming language is a standardized programming language developed in the early 1970s by Ken Thompson and Dennis Ritchie for use on the UNIX operating system. It has since spread to many other operating systems, and is one of the most widely used programming languages.

2. What is the output of printf("%d")?

Ans. 1. When we write printf("%d",x); this means compiler will print the value of x.

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

3. Some compilers check the format string and will generate an error without the proper number and type of arguments for things like printf(...) and scanf(...).

3. What is the difference between "calloc(...)" and "malloc(...)"?

Ans. **1. calloc(...)** allocates a block of memory for an array of elements of a certain size. By default the block is initialized to 0. The total number of memory allocated will be (number_of_elements * size).

malloc(...) takes in only a single argument which is the memory required in bytes. **malloc(...)** allocated bytes of memory and not blocks of memory like **calloc(...)**.

2. malloc(...) allocates memory blocks and returns a void pointer to the allocated space, or NULL if there is insufficient memory available.

calloc(...) allocates an array in memory with elements initialized to 0 and returns a pointer to the allocated space. **calloc(...)** calls **malloc(...)** in order to use the C++ _set_new_mode function to set the new handler mode.

4. What is the difference between "printf(...)" and "sprintf(...)"?

Ans. **sprintf(...)** writes data to the character array whereas printf(...) writes data to the standard output device.

5. How to reduce a final size of executable?

Ans. Size of the final executable can be reduced using dynamic linking for libraries.

6. Can you tell me how to check whether a linked list is circular?

Ans. Create two pointers, and set both to the start of the list. Update each as follows:

```
while (pointer1) {
pointer1 = pointer1->next;
pointer2 = pointer2->next;
if (pointer2) pointer2=pointer2->next;
if (pointer1 == pointer2) {
print ("circular");
}
}
```

If a list is circular, at some point pointer2 will wrap around and be either at the item just before pointer1, or the item before that. Either way, its either 1 or 2 jumps until they meet.

7. "union" Data Type What is the output of the following program? Why?

```
#include
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("Union x : %d %s %f n",x.a,x.b,x.c);
        printf("Union y : %d %s %f n",y.a,y.b,y.c);
        }
```

8. Write out a function that prints out all the permutations of a string. For example, abc would give you abc, acb, bac, bca, cab, cba.

```
void PrintPermu (char *sBegin, char* sRest) {
int iLoop;
char cTmp;
char cFLetter[1];
char *sNewBegin;
char *sCur;
int iLen;
static int iCount;
iLen = strlen(sRest);
if (iLen == 2) {
iCount++;
printf("%d: %s%s\n",iCount,sBegin,sRest);
iCount++;
printf("%d: %s%c%c\n",iCount,sBegin,sRest[1],sRest[0]);
return;
} else if (iLen == 1) {
iCount++;
printf("%d: %s%s\n", iCount, sBegin, sRest);
return;
} else {
// swap the first character of sRest with each of
// the remaining chars recursively call debug print
sCur = (char*)malloc(iLen);
sNewBegin = (char*)malloc(iLen);
for (iLoop = 0; iLoop < iLen; iLoop ++) {
strcpy(sCur, sRest);
strcpy(sNewBegin, sBegin);
cTmp = sCur[iLoop];
sCur[iLoop] = sCur[0];
sCur[0] = cTmp;
sprintf(cFLetter, "%c", sCur[0]);
strcat(sNewBegin, cFLetter);
debugprint(sNewBegin, sCur+1);
}
}
}
```

```
void main() {
char s[255];
char sIn[255];
printf("\nEnter a string:");
scanf("%s%*c",sIn);
memset(s,0,255);
PrintPermu(s, sIn);
}
```

9. What will print out?

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

The pointer p2 value is also increasing with p1 .

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

e.g. :

initially p1 = 2000 (address) , p2 = 3000

*p1 has value "n" ..after 4 increments , loop exits … at that time p1 value will be 2004 , p2 =3004 … the actual result is stored in 3000 - n , 3001 - a , 3002 - m , 3003 -e … we r trying to print from 3004 …. where no data is present … that's why its printing null .

Ans. empty string.

10. What will happen in these three cases?

```
if(a=0){
//somecode
}
if (a==0){
//do something
}
```

```
if (a===0){
//do something
}
```

What are x, y, y, u

```
#define Atype int*
typedef int *p;
p x, z;
Atype y, u;
```

Ans. x and z are pointers to int. y is a pointer to int but u is just an integer variable.

11. What does static variable mean?

Ans. There are 3 main uses for the static.

1. If you declare within a function:

It retains the value between function calls

2.If it is declared for a function name:

By default function is extern..so it will be visible from other files if the function declaration is as static..it is invisible for the outer files

3. Static for global variables:

By default we can use the global variables from outside files If it is static global..that variable is limited to with in the file.

12. Advantages of a macro over a function.

Ans. Macro gets to see the Compilation environment, so it can expand __ __TIME__ __FILE__ #defines. It is expanded by the preprocessor.

For example, you can't do this without macros

```
#define PRINT(EXPR) printf( #EXPR "=%d\n", EXPR)
PRINT( 5+6*7 ) // expands into printf("5+6*7=%d", 5+6*7 );
```

You can define your mini language with macros:

```
#define strequal(A,B) (!strcmp(A,B))
```

Macros are a necessary evils of life. The purists don't like them, but without it no real work gets done.

13. What are the differences between malloc() and calloc()?

Ans. There are 2 differences.

First, is in the number of arguments. malloc() takes a single argument(memory required in bytes), while calloc() needs 2 arguments(number of variables to allocate memory, size in bytes of a single variable).

Secondly, malloc() does not initialize the memory allocated, while calloc() initializes the allocated memory to ZERO.

14. What are the different storage classes in C?

Ans. C has three types of storage: automatic, static and allocated.

Variable having block scope and without static specifier have automatic storage duration.

Variables with block scope, and with static specifier have static scope. Global variables (*i.e.*, file scope) with or without the static specifier also have static scope.

Memory obtained from calls to malloc(), alloc() or realloc() belongs to allocated storage class.

15. What is the difference between strings and character arrays?

Ans. A major difference is: string will have static storage duration, whereas as a character array will not, unless it is explicity specified by using the static keyword.

Actually, a string is a character array with following properties:

* the multibyte character sequence, to which we generally call string, is used to initialize an array of static storage duration. The size of this array is just sufficient to contain these characters plus the terminating NUL character.

* it not specified what happens if this array, *i.e.,* string, is modified.

* Two strings of same value[1] may share same memory area. For example, in the following declarations:

char *s1 = “Calvin and Hobbes”;

char *s2 = “Calvin and Hobbes”;

the strings pointed by s1 and s2 may reside in the same memory location. But, it is not true for the following:

char ca1[] = “Calvin and Hobbes”;

char ca2[] = “Calvin and Hobbes”;

[1] The value of a string is the sequence of the values of the contained characters, in order.

16. Write down the equivalent pointer expression for referring the same element a[i][j][k][l].

Ans. a[i] == *(a+i)

a[i][j] == *(*(a+i)+j)

a[i][j][k] == *(*(*(a+i)+j)+k)

a[i][j][k][l] == *(*(*(*(a+i)+j)+k)+l)

17. Which bitwise operator is suitable for checking whether a particular bit is on or off?

Ans. The bitwise AND operator. Here is an example:enum {

KBit0 = 1,

KBit1,

…

KBit31,

};

```
if ( some_int & KBit24 )
printf ( "Bit number 24 is ON\n" );
else
printf ( "Bit number 24 is OFF\n" );
```

18. Which bitwise operator is suitable for turning off a particular bit in a number?

Ans. The bitwise AND operator, again. In the following code snippet, the bit number 24 is reset to zero.

```
some_int = some_int & ~KBit24;
```

19. Which bitwise operator is suitable for putting on a particular bit in a number?

Ans. The bitwise OR operator. In the following code snippet, the bit number 24 is turned ON:

```
some_int = some_int | KBit24;
```

20. Does there exist any other function which can be used to convert an integer or a float to a string?

Ans. Some implementations provide a nonstandard function called itoa(), which converts an integer to string.

```
#include
char *itoa(int value, char *string, int radix);
DESCRIPTION
The itoa() function constructs a string representation of an integer.
PARAMETERS
value:
Is the integer to be converted to string representation.
string:
Points to the buffer that is to hold resulting string.
The resulting string may be as long as seventeen bytes.
radix:
Is the base of the number; must be in the range 2 - 36.
```

A portable solution exists. One can use sprintf():

```
char s[SOME_CONST];
int i = 10;
float f = 10.20;
sprintf ( s, "%d %f \n", i, f );
```

21. Why does malloc(0) return valid memory address? What's the use?

Ans. malloc(0) does not return a non-NULL under every implementation.

An implementation is free to behave in a manner it finds suitable, if the allocation size requested is zero. The implmentation may choose any of the following actions:

* A null pointer is returned.

* The behavior is same as if a space of non-zero size was requested. In this case, the usage of return value yields to undefined-behavior.

22. Notice, however, that if the implementation returns a non-NULL value for a request of a zero-length space, a pointer to object of ZERO length is returned! Think, how an object of zero size should be represented?

Ans. For implementations that return non-NULL values, a typical usage is as follows:

```
void
func ( void )
{
int *p; /* p is a one-dimensional array,
whose size will vary during the
the lifetime of the program */
size_t c;
p = malloc(0); /* initial allocation */
if (!p)
{
perror (”FAILURE” );
return;
}
/* … */
while (1)
{
c = (size_t) … ; /* Calculate allocation size */
p = realloc ( p, c * sizeof *p );
/* use p, or break from the loop */
/* … */
}
return;
}
```

Notice that this program is not portable, since an implementation is free to return NULL for a malloc(0) request, as the C Standard does not support zero-sized objects.

23. How do I write code that reads data at memory location specified by segment and offset?

Ans. Use peekb( ) function. This function returns byte(s) read from specific segment and offset locations in memory. The following program illustrates use of this function. In this program from VDU memory we have read characters and its attributes of the first row. The information stored in file is then further read and displayed using peek( ) function.

```
#include <stdio.h>
#include <dos.h>
main( )
{
char far *scr = 0xB8000000 ;
FILE *fp ;
int offset ;
char ch ;
if ( ( fp = fopen ( "scr.dat", "wb" ) ) == NULL )
{
printf ( "\nUnable to open file" ) ;
exit( ) ;
}
// reads and writes to file
for ( offset = 0 ; offset < 160 ; offset++ )
fprintf ( fp, "%c", peekb ( scr, offset ) ) ;
fclose ( fp ) ;
if ( ( fp = fopen ( "scr.dat", "rb" ) ) == NULL )
{
printf ( "\nUnable to open file" ) ;
exit( ) ;
}
// reads and writes to file
for ( offset = 0 ; offset < 160 ; offset++ )
{
fscanf ( fp, "%c", &ch ) ;
printf ( "%c", ch ) ;
}
fclose ( fp ) ;
}
```

24. What is conversion operator?

Ans. class can have a public method for specific data type conversions.

for example:

```
class Boo
{
```

```
double value;
public:
Boo(int i )
operator double()
{
return value;
}
};
Boo BooObject;
```

double i = BooObject; // assigning object to variable i of type double. now conversion operator gets called to assign the value.

25. What is the difference between malloc()/free() and new/delete?

Ans. malloc allocates memory for object in heap but doesn't invoke object's constructor to initiallize the object.

new allocates memory and also invokes constructor to initialize the object.

malloc() and free() do not support object semantics

Does not construct and destruct objects

```
string * ptr = (string *)(malloc (sizeof(string)))
```

Are not safe

Does not calculate the size of the objects that it construct

Returns a pointer to void

```
int *p = (int *) (malloc(sizeof(int)));
int *p = new int;
```

Are not extensible

new and delete can be overloaded in a class

"delete" first calls the object's termination routine (*i.e.* its destructor) and then releases the space the object occupied on the heap memory. If an array of objects was created using new, then delete must be told that it is dealing with an array by preceding the name with an empty []:-

```
Int_t *my_ints = new Int_t[10];
...
delete []my_ints;
```

26. What is the difference between "new" and "operator new" ?

Ans. "operator new" works like malloc.

27. What is difference between template and macro?

Ans. There is no way for the compiler to verify that the macro parameters are of compatible types. The macro is expanded without any special type checking.

If macro parameter has a postincremented variable (like c++), the increment is performed two times.

Because macros are expanded by the preprocessor, compiler error messages will refer to the expanded macro, rather than the macro definition itself. Also, the macro will show up in expanded form during debugging.

for example:

Macro:

```
#define min(i, j) (i < j ? i : j)
```

template:

```
template<class T>
T min (T i, T j)
{
return i < j ? i : j;
}
```

28. What are advantages and disadvantages of external storage class?

Ans. Advantages of external storage class are:

1)Persistent storage of a variable retains the latest value

2)The value is globally available

Disadvantages of external storage class are:

1)The storage for an external variable exists even when the variable is not needed

2)The side effect may produce surprising output

3)Modification of the program is difficult

4)Generality of a program is affected.

29. What is a void pointer?

Ans. A void pointer is a C convention for a raw address. The compiler has no idea what type of object a void Pointer really points to. If you write

```
int *ip;
```

ip points to an int. If you write

```
void *p;
```

p doesn't point to a void!

In C and C++, any time you need a void pointer, you can use another pointer type. For example, if you have a char*, you can pass it to a function that expects a void*. You don't even need to cast it. In C (but not in C++), you can use a void* any time you need any kind of pointer, without casting. (In C++, you need to cast it).

A void pointer is used for working with raw memory or for passing a pointer to an unspecified type.

Some C code operates on raw memory. When C was first invented, character pointers (char *) were used for that. Then people started getting confused about when a character pointer was a string, when it was a character array, and when it was raw memory.

30. How can type-insensitive macros be created?

Ans. A type-insensitive macro is a macro that performs the same basic operation on different data types.

This task can be accomplished by using the concatenation operator to create a call to a type-sensitive function based on the parameter passed to the macro. The following program provides an example:

```
#include
#define SORT(data_type) sort_ ## data_type
void sort_int(int** i);
void sort_long(long** l);
void sort_float(float** f);
void sort_string(char** s);
void main(void);
void main(void)
{
int** ip;
long** lp;
float** fp;
char** cp;
...
sort(int)(ip);
sort(long)(lp);
sort(float)(fp);
sort(char)(cp);
...
}
```

This program contains four functions to sort four different data types: int, long, float, and string (notice that only the function prototypes are included for brevity). A macro named SORT was created to take the data type passed to the macro and combine it with the sort_ string to form a valid function call that is appropriate for the data type being sorted. Thus, the string

sort(int)(ip);

translates into

sort_int(ip);

after being run through the preprocessor.

31. When should a type cast not be used?

Ans. A type cast should not be used to override a const or volatile declaration. Overriding these type modifiers can cause the program to fail to run correctly.

A type cast should not be used to turn a pointer to one type of structure or data type into another. In the rare events in which this action is beneficial, using a union to hold the values makes the programmer's intentions clearer.

32. When is a switch statement better than multiple if statements?

Ans. A switch statement is generally best to use when you have more than two conditional expressions based on a single variable of numeric type.

33. What is storage class and what are storage variable?

Ans. A storage class is an attribute that changes the behavior of a variable. It controls the lifetime, scope and linkage.

There are five types of storage classes

1) auto

2) static

3) extern

4) register

5) typedef.

34. What is a static function?

Ans. A static function is a function whose scope is limited to the current source file. Scope refers to the visibility of a function or variable. If the function or variable is visible outside of the current source file, it is said to have global, or external, scope. If the function or variable is not visible outside of the current source file, it is said to have local, or static, scope.

35. How can I sort things that are too large to bring into memory?

Ans. A sorting program that sorts items that are on secondary storage (disk or tape) rather than primary storage (memory) is called an external sort. Exactly how to sort large data depends on what is meant by too large to fit in memory. If the items to be sorted are themselves too large to fit in memory (such as images), but there aren't many items, you can keep in memory only the sort key and a value indicating the data's location on disk. After the key/value pairs are sorted, the data is rearranged on disk into the correct order. If too large to fit in memory means that there are too many items to fit into memory at one time, the data can be sorted in groups that will fit into memory, and then the resulting files can be merged. A sort such as a radix sort can also be used as an external sort, by making each bucket in the

sort a file. Even the quick sort can be an external sort. The data can be partitioned by writing it to two smaller files. When the partitions are small enough to fit, they are sorted in memory and concatenated to form the sorted file.

36. What is a pointer variable?

Ans. A pointer variable is a variable that may contain the address of another variable or any valid address in the memory.

37. What is a pointer value and address?

Ans. A pointer value is a data object that refers to a memory location. Each memory location is numbered in the memory. The number attached to a memory location is called the address of the location.

38. What is a modulus operator? What are the restrictions of a modulus operator?

Ans. A Modulus operator gives the remainder value. The result of x%y is obtained by (x-(x/y)*y). This operator is applied only to integral operands and cannot be applied to float or double.

A linker converts an object code into an executable code by linking together the necessary build in functions. The form and place of declaration where the variable is declared in a program determine the linkage of variable.

39. What is a function and built-in function?

Ans. A large program is subdivided into a number of smaller programs or subprograms. Each subprogram specifies one or more actions to be performed for a large program. such subprograms are functions.

The function supports only static and extern storage classes. By default, function assumes extern storage class. functions have global scope. Only register or auto storage class is allowed in the function parameters. Built-in functions that predefined and supplied along with the compiler are known as built-in functions. They are also known as library functions.

40. What is a macro, and how do you use it?

Ans. A macro is a preprocessor directive that provides a mechanism for token replacement in your source code. Macros are created by using the #define statement.

Here is an example of a macro: Macros can also utilize special operators such as the stringizing operator (#) and the concatenation operator (##).The stringizing operator can be used to convert macro parameters to quoted strings, as in the following example:

#define DEBUG_VALUE(v) printf(#v is equal to %d.n, v)

In your program, you can check the value of a variable by invoking the DEBUG_VALUE macro:

...

int x = 20;

DEBUG_VALUE(x);

...

The preceding code prints x is equal to 20. on-screen. This example shows that the stringizing operator used with macros can be a very handy debugging tool.

41. What is the difference between goto and longjmp() and setjmp()?

Ans. A goto statement implements a local jump of program execution, and the longjmp() and setjmp() functions implement a nonlocal, or far, jump of program execution.

Generally, a jump in execution of any kind should be avoided because it is not considered good programming practice to use such statements as goto and longjmp in your program.

A goto statement simply bypasses code in your program and jumps to a predefined position. To use the goto statement, you give it a labeled position to jump to. This predefined position must be within the same function. You cannot implement gotos between functions.

When your program calls setjmp(), the current state of your program is saved in a structure of type jmp_buf. Later, your program can call the longjmp() function to restore the program's state as it was when you called setjmp().Unlike the goto statement, the longjmp() and setjmp() functions do not need to be implemented in the same function.

However, there is a major drawback to using these functions: your program, when restored to its previously saved state, will lose its references to any dynamically allocated memory between the longjmp() and the setjmp(). This means you will waste memory for every malloc() or calloc() you have implemented between your longjmp() and setjmp(), and your program will be horribly inefficient. It is highly recommended that you avoid using functions such as longjmp() and setjmp() because they, like the goto statement, are quite often an indication of poor programming practice.

42. Is it acceptable to declare/define a variable in a C header?

Ans. A global variable that must be accessed from more than one file can and should be declared in a header file. In addition, such a variable must be defined in one source file.

Variables should not be defined in header files, because the header file can be included in multiple source files, which would cause multiple definitions of the variable. The ANSI C standard will allow multiple external definitions, provided that there is only one initialization. But because there's really no advantage to using this feature, it's probably best to avoid it and maintain a higher level of portability.

Global variables that do not have to be accessed from more than one file should be declared static and should not appear in a header file.

43. Why should I prototype a function?

Ans. A function prototype tells the compiler what kind of arguments a function is looking to receive and what kind of return value a function is going to give back. This approach helps the compiler ensure that calls to a function are made correctly and that no erroneous type conversions are taking place.

44. What is the quickest searching method to use?

Ans. A binary search, such as bsearch() performs, is much faster than a linear search. A hashing algorithm can provide even faster searching. One particularly interesting and fast method

for searching is to keep the data in a digital trie. A digital trie offers the prospect of being able to search for an item in essentially a constant amount of time, independent of how many items are in the data set.

A digital trie combines aspects of binary searching, radix searching, and hashing. The term digital trie refers to the data structure used to hold the items to be searched. It is a multilevel data structure that branches N ways at each level.

45. What are the advantages of auto variables?

Ans. 1) The same auto variable name can be used in different blocks

2) There is no side effect by changing the values in the blocks

3) The memory is economically used

4) Auto variables have inherent protection because of local scope.

46. What are the characteristics of arrays in C?

Ans. 1) An array holds elements that have the same data type.

2) Array elements are stored in subsequent memory locations.

3) Two-dimensional array elements are stored row by row in subsequent memory locations.

4) Array name represents the address of the starting element.

5) Array size should be mentioned in the declaration. Array size must be a constant expression and not a variable.

47. How do you print only part of a string?

Ans. /* Use printf() to print the first 11 characters of source_str. */

printf(First 11 characters: '%11.11s'n, source_str);

48. In C, what is the difference between a static variable and global variable?

Ans. A static variable declared outside of any function is accessible only to all the functions defined in the same file (as the static variable). However, a global variable can be accessed by any function (including the ones from different files).

49. In C, why is the void pointer useful?

Ans. When would you use it? The void pointer is useful because it is a generic pointer that any pointer can be cast into and back again without loss of information.

50. What is Polymorphism ?

Ans. 'Polymorphism' is an object oriented term. Polymorphism may be defined as the ability of related objects to respond to the same message with different, but appropriate actions. In other words, polymorphism means taking more than one form. Polymorphism leads to two important aspects in Object Oriented terminology - Function Overloading and Function Overriding. Overloading is the practice of supplying more than one definition for a given function name in the same scope. The compiler is left to pick the appropriate version of the function or operator based on the arguments with which it is called. Overriding refers to the

modifications made in the sub class to the inherited methods from the base class to change their behavior.

51. What is Operator overloading ?

Ans. When an operator is overloaded, it takes on an additional meaning relative to a certain class. But it can still retain all of its old meanings.

Examples:

1) The operators >> and << may be used for I/O operations because in the header, they are overloaded.

2) In a stack class it is possible to overload the + operator so that it appends the contents of one stack to the contents of another. But the + operator still retains its original meaning relative to other types of data.

52. What are Templates?

Ans. C++ Templates allow u to generate families of functions or classes that can operate on a variety of different data types, freeing you from the need to create a separate function or class for each type. Using templates, u have the convenience of writing a single generic function or class definition, which the compiler automatically translates into a specific version of the function or class, for each of the different data types that your program actually uses. Many data structures and algorithms can be defined independently of the type of data they work with. You can increase the amount of shared code by separating data-dependent portions from data-independent portions, and templates were introduced to help you do that.

53. What is the difference between dynamic binding and static binding?

Ans. ***Dynamic Binding :***

The address of the functions are determined at runtime rather than @ compile time. This is also known as “Late Binding”.

Static Binding :

The address of the functions are determined at compile time rather than @ run time. This is also known as “Early Binding”.

54. What is difference between C/C++?

Ans. C does not have a class/object concept.

C++ provides data abstraction, data encapsulation, Inheritance and Polymorphism.

C++ supports all C syntax.

In C passing value to a function is “Call by Value” whereas in C++ its “Call by Reference”

File extension is .c in C while .cpp in C++.(C++ compiler compiles the files with .c extension but C compiler can not!)

In C structures can not have contain functions declarations. In C++ structures are like classes, so declaring functions is legal and allowed.

C++ can have inline/virtual functions for the classes.

c++ is C with Classes hence C++ while in c the closest u can get to an User defined data type is struct and union.

Why doesn't the following code give the desired result?

int x = 3000, y = 2000 ;

long int z = x * y ;

Here the multiplication is carried out between two ints x and y, and the result that would overflow would be truncated before being assigned to the variable z of type long int. However, to get the correct output, we should use an explicit cast to force long arithmetic as shown below:

long int z = (long int) x * y ;

Note that (long int)(x * y) would not give the desired effect.

55. Why doesn't the following statement work?

Ans. char str[ ] = "Hello" ;

strcat (str, '!') ;

The string function strcat( ) concatenates strings and not a character. The basic difference between a string and a character is that a string is a collection of characters, represented by an array of characters whereas a character is a single character. To make the above statement work writes the statement as shown below:

strcat (str, "!") ;

56. How do I know how many elements an array can hold?

Ans. The amount of memory an array can consume depends on the data type of an array. In DOS environment, the amount of memory an array can consume depends on the current memory model (*i.e.* Tiny, Small, Large, Huge, etc.). In general an array cannot consume more than 64 kb. Consider following program, which shows the maximum number of elements an array of type int, float and char can have in case of Small memory model.

```
main( )
{
int i[32767] ;
float f[16383] ;
char s[65535] ;
}
```

Chapter 7 QUESTIONS

1. What is a compiler?
2. How is a C program run?
3. How is a C program compiled usually?
4. Are upper and lower case equivalent in C?
5. What the two different kinds of error which can be in a program?
6. Write a command to print out the message “Wow big deal”.
7. Write a command to print out the number 22.
8. Write two commands to print out “The 3 Wise Men” two different ways.
9. Why are there only a few reserved command words in C?
10. What is an operating system for?
11. What is a pseudo-device name?
12. If you had a C source program which you wanted to call ‘accounts’ what name would you save it under?
13. What would be the name of the file produced by the compiler of the program in 3?
14. How would this program be run?
15. How is a library file incorporated into a C program?
16. Name the most common library file in C.
17. Is it possible to define new functions with the same names as standard library functions?
18. What is another name for a library file?
19. What is a block?
20. Name the six basic things which make up a C program.
21. Does a C program start at the beginning? (Where is the beginning?)
22. What happens when a program comes to a } character? What does this character signify?
23. What vital piece of punctuation goes at the end of every simple C statement?
24. What happens if a comment is not ended? That is if the programmer types /* .. to start but forgets the ..*/ to close?

25. Write a function which takes two values a and b and returns the value of (a*b).

26. Is there anything wrong with a function which returns no value?

27. What happens if a function returns a value but it is not assigned to anything?

28. What happens if a function is assigned to an object but that function returns no value?

29. How can a function be made to quit early?

30. What is an identifier?

31. Say which of the following are valid C identifiers:

1. `Ralph23`
2. `80shillings`
3. `mission_control`
4. `A%`
5. `A$`
6. `_off`

32. Write a statement to declare two integers called i and j.

33. What is the difference between the types `float` and `double`?

34. What is the difference between the types `int` and `unsigned int`?

35. Write a statement which assigns the value 67 to the integer variable "I".

36. What type does a C function return by default?

37. If we want to declare a function to return `long float`, it must be done in, at least, two places. Where are these?

38. Write a statement, using the cast operator, to print out the integer part of the number 23.1256.

39. Is it possible to have an automatic global variable?

40. Name two ways that values and results can be handed back from a function.

41. Where are parameters declared?

42. Can a function be used directly as a value parameter?

43. Does it mean anything to use a function directly as a variable parameter?

44. What do the symbols * and & mean, when they are placed in front of an identifier?

45. Do actual and formal parameters need to have the same names?

46. Define a macro called "birthday" which describes the day of the month upon which your birthday falls.

47. Write an instruction to the preprocessor to include to maths library `math.h`.

48. A macro is always a number. True or false?

49. A macro is always a constant. True or false?

50. What is a pointer?

51. How is a variable declared to be a pointer?

52. What data types can pointers "point to"?

53. Write a statement which converts a pointer to a character into a pointer to a `double` type. (This is not as pointless as it seems. It is useful in dealing with unions and memory allocation functions.)

54. Why is it incorrect to declare: `float *number = 2.65;` ?

55. Write a program which simply prints out: `6.23e+00`.

56. Investigate what happens when you type the wrong conversion specifier in a program. *e.g.* try printing an integer with `%f` or a floating point number with `%c`. This is bound to go wrong - but how will it go wrong?

57. What is wrong with the following statements?

1. `printf (x);`
2. `printf ("%d");`
3. `printf ();`
4. `printf ("Number = %d");`

Hint: if you don't know, try them in a program!

58. What is a white space character?

59. Write a program which fetches two integers from the user and multiplies them together. Print out the answer. Try to make the input as safe as possible.

60. Write a program which just echoes all the input to the output.

61. Write a program which strips spaces out of the input and replaces them with a single newline character.

62. `scanf` always takes pointer arguments. True or false?

63. What is an operand?

64. Write a statement which prints out the remainder of 5 divided by 2.

65. Write a short statement which assigns the remainder of 5 divided by 2 to a variable called "rem".

66. Write a statement which subtracts -5 from 10.

67. Write in C: if 1 is not equal to 23, print out "Thank goodness for mathematics!"

68. How many kinds of loop does C offer, and what are they?

69. When is the condition tested in each of the loops?

70. Which of the loops is always executed once?

71. Write a program which copies all input to output line by line.

72. Write a program to get 10 numbers from the user and add them together.

73. Given any array, how would you find a pointer to the start of it?

74. How do you pass an array as a parameter? When the parameter is received by a function does C allocate space for a local variable and copy the whole array to the new location?

75. Write a statement which declares an array of type double which measures 4 by 5. What numbers can be written in the indicies of the array?

76. What are the two main ways of declaring strings in a program?

77. How would you declare a static array of strings?

78. Write a program which gets a number between 0 and 9 and prints out a different message for each number. Use a pre-initialized array to store the strings.

79. What type of data is returned from mathematical functions?

80. All calculations are performed using long variables. True or false?

81. What information is returned by `strlen()`?

82. What action is performed by `strcat()`?

83. Name five kinds of error which can occur in a mathematical function.

84. Which operators can be hidden inside other statements?

85. Give a reason why you would not want to do this in every possible case.

86. Is `FILE` a reserved word? If so why is it in upper case?

87. Write a statement which declares a file pointer called fp.

88. Enumerated data are given values by the compiler so that it can do arithmetic with them. True or false?

89. Does `void` do anything which C cannot already do without this type?

90. What type might a timer device be declared if it were to be called by a variable name?

91. Write a statement which declares a new type "real" to be like the usual type "double".

92. Variables declared `const` can be of any type. True or false?

93. What distinguishes a bit pattern from an ordinary variable? Can any variable be a bit pattern?

94. What is the difference between an inclusive OR operation and an exclusive OR operation?

95. If you saw the following function call in a program, could you guess what its parameter was?

96. OpenWindow (BORDER | GADGETS | MOUSECONTROL | SIZING);

97. What is the difference between call by value and call by reference?

98. Find out what the denary (decimal) values of the following operations are:

1. `7 & 2`
2. `1 & 1`
3. `15 & 3`
4. `15 & 7`
5. `15 & 7 & 3`

Try to explain the results. (Hint: draw out the numbers as binary patterns, using the program listed.)

99. Find out what the denary (decimal) values of the following operations are:

1. `1 | 2`
2. `1 | 2 | 3`

100. Find out the values of:

1. `1 & (~1)`
2. `23 & (~23)`
3. `2012 & (~2012)`

(**Hint:** write a short program to work them out. Use short type variables for all the numbers).

101. What are the following?

1. File name
2. File pointer
3. File handle

102. What is the difference between high and low level filing?

103. Write a statement which opens a high level file for reading.

104. Write a statement which opens a low level file for writing.

105. Write a program which checks for illegal characters in text files. Valid characters are ASCII codes 10,13,and 32..126. Anything else is illegal for programs.

106. What statement performs formatted writing to text files?

107. Print out all the header files on your system so that you can see what is defined where!

108. What is the difference between a structure and a union?

109. What is a member?

110. If x is a variable, how would you find out the value of a member called `mem`.

111. If `ptr` is a pointer to a structure, how would you find out the value of a member called `mem`.

112. A union is a group of variables in a single package. True or false?

113. What is a structure diagram?

114. How are data linked together to make a data structure?

115. Every separate struct type in a data structure has its own variable name. True or false?

116. How are the members of structures accessed in a data structure?

117. Write a statement which creates a new structure of type "struct BinaryTree" and finds its address. Store that address in a variable which is declared as follows:

```
struct BinaryTree *ptr;
```

118. What happens if the condition in a while loop is initially false?

119. What is a null string?

120. Write a small program which makes a linked list, three structures long and assigns all their data to be zero. Can you automate this program with a loop? Can you make it work for any number of structures?

121. What is the output of **printf("%d")?**

122. What will happen if I say **delete this?**

123. Differentiate between "C structure" and "C++ structure".

124. Diffrentiate between a “assignment operator” and a “copy constructor”.
125. What is the difference between “overloading” and “overridding”?
126. Explain the need for “Virtual Destructor”.
127. Can we have “Virtual Constructors”?
128. What are the different types of polymorphism?
129. What are Virtual Functions? How to implement virtual functions in “C”?
130. What are the different types of Storage classes?
131. What is Namespace?
132. What are the types of STL containers?.
133. Differentiate between “vector” and “array”.
134. How to write a program such that it will delete itself after exectution?
135. Can we generate a C++ source code from the binary file?
136. What are inline functions?
137. Talk sometiming about profiling.
138. How many lines of code you have written for a single program?
139. What is “strstream” ?
140. How to write Multithreaded applications using C++?
141. Explain “passing by value”, “passing by pointer” and “passing by reference”.
142. Write any small program that will compile in “C” but not in “C++”.
143. Have you heard of “mutable” keyword?
144. What is a “RTTI”?
145. Is there something that I can do in C and not in C++?
146. Why preincrement operator is faster than postincrement?
147. What is the difference between “calloc” and “malloc”?
148. What will happen if I allocate memory using “new” and free it using “free” or allocate sing “calloc” and free it using “delete”?
149. What is Memory Alignment?
150. Explain working of printf.
151. Differentiate between “printf ” and “sprintf”.
152. What is “map” in STL?
153. When shall I use Multiple Inheritance?
154. What are the techniques you use for debugging?
155. How to reduce a final size of executable?
156. Give 2 examples of a code optimization.
157. Is it possible to use multiple else with if statement?

158. Is it possible to use multiple default statements in switch () statement?
159. Why goto statement is avoided?
160. What are the limitations of switch() case statement?
161. Can we put default statement anywhere in the switch () case structure?
162. What happens if you create a loop that never ends?
163. Is it possible to create a for loop that is never executed?
164. What is a loop? Why it is necessary in the program?
165. Is it possible to nest while loop within for loops?
166. Is it possible to use multiple while statement with do statement?
167. What are the values of NULL and !NULL?
168. What is the difference between (!0) and (!0)?
169. Write the definition of a function.
170. How do functions help to reduce the program size?
171. Differentiate between library and the user defined functions.
172. How does a function work?
173. Explain How arguments are passes and results are returned?
174. List any five library functions and illustrate them with suitable examples.
175. What are actual and formal argumnets?
176. What are the uses of return statements?
177. What does it mean if there is no return statement in the function?
178. What are void functions?
179. What is global pointer?
180. What is recursion? Explain its advantages.
181. Explain types of recursions.
182. Is it possible to call library functions recursively?