1B:

1. Invalid. An Integer constant must have at least one digit.
2. Valid.
3. Invalid. An integer constant must not have a decimal point.
4. Invalid. An integer constant cannot have inverted commas.
5. Invalid. An Integer constant cannot contain a comma.
6. Invalid. There cannot be a space within an Integer constant.
7. Valid.

2A:

1. Invalid. An Integer constant cannot have inverted commas.
2. Invalid. An integer constant cannot have inverted commas.
3. Invalid. An integer constant cannot have a decimal point.
4. Invalid. An integer constant cannot have inverted commas.
5. Invalid. An Integer constant cannot contain a comma.
6. Invalid. There cannot be a space within an Integer constant.
7. Invalid. An integer constant cannot have a decimal point.
8. Valid.

3A:

1. Valid
2. Valid
3. Valid
4. Invalid. The Mantissa part and the exponential part should be separated by letter e or E.
5. Invalid. No blanks are allowed within a real constant.
6. Valid.

3B:

1. Invalid. No quotation marks are allowed within a real constant.
2. Valid.
3. Valid.
4. Valid.
5. Invalid. No quotation marks are allowed within a real constant.
6. Invalid. Real constants must have a decimal point.

4A:

1. Valid.
2. Valid.
3. Valid
4. Invalid. A character constant can contain only 1 character.

4B:

1. Invalid. A character constant can contain only 1 character.
2. Valid.
3. Invalid. Both the inverted commas should point to the left.
4. Invalid. Both the inverted commas should point to the left.
5. Invalid. Both the inverted commas should point to the left.
6. Invalid. A character constant can contain only 1 character.
7. Invalid. A character constant can contain only 1 character.
8. Invalid. A character constant can contain only 1 character.

5B:

1. Invalid. An apostrophe is not allowed in a variable name.
2. Invalid. A variable name must begin with an alphabet or underscore.
3. Valid. OR No special symbol other than an underscore can be used in a variable name.
4. Invalid. No special symbol other than an underscore can be used in a variable name.
5. Invalid. Variable name cannot contain spaces.
6. Invalid. Keyword cannot be used as a variable name.

6A:

1. Invalid. Invalid. A ‘.’ is not allowed in variable name. No special symbol other than an underscore can be used in a variable name.
2. Invalid. No special symbol other than an underscore can be used in a variable name.
3. Invalid. A variable name must start with an alphabet or an underscore.
4. Invalid. A variable name must start with an alphabet or an underscore.
5. Invalid. A variable name must begin with an alphabet or underscore.
6. Valid.

6B:

1. Valid.
2. Invalid. Invalid. A ‘-’ is not allowed in variable name. No special symbol other than an underscore can be used in a variable name.
3. Valid.
4. Valid
5. Valid
6. Invalid. Invalid. A ‘-’ is not allowed in variable name. No special symbol other than an underscore can be used in a variable name.

7A:

1. Invalid. A Variable name must begin with an alphabet or underscore.
2. Invalid. Invalid. A ‘.’ is not allowed in variable name. No special symbol other than an underscore can be used in a variable name.
3. Invalid. A variable name cannot begin with a digit.
4. Invalid. A variable name cannot contain spaces.
5. Invalid. Variable name cannot contain spaces.
6. Valid.

7B:

1. Valid.
2. Valid.
3. Invalid. Invalid. A ‘.’ is not allowed in variable name. No special symbol other than an underscore can be used in a variable name.
4. Invalid. Invalid. An apostrophe is not allowed in variable name. No special symbol other than an underscore can be used in a variable name.
5. Invalid. Invalid. A ‘#’ and spaces not allowed in variable name. No special symbol other than an underscore can be used in a variable name.
6. Invalid. A variable name must start with an alphabet or an underscore.

8A:

1. True
2. True
3. False
4. True
5. True
6. True
7. True
8. True
9. True
10. True
11. False
12. True
13. True
14. True
15. True
16. True

8B:

1. True
2. True
3. True
4. True
5. True
6. true
7. True
8. True
9. True
10. True
11. true
12. True
13. True
14. True
15. True
16. True
17. True
18. True
19. True
20. False

9A:

1. Error. Int is a keyword hence should not be used as a variable.
2. Error. ‘Ajay’ is an invalid character constant.
3. No Error.
4. Error. On the left-hand side of the equal to sign (=), there can only be a variable.

9B:

1. Error. Multiplication operator (\*) missing among the three expressions and also between 2.5 and variable a.
2. Error. ‘Rate of interest’ and ‘amount in rs’ are invalid variable names.
3. No Error.
4. Error. ‘\*\*’ is an invalid operator.

10A:

1. Error. The expression 3.14 \* r will result in a float. The ‘^’ operator cannot be used with floats.
2. Error. Multiplication operator (\*) is missing been the two expressions given in parentheses.
3. Error. A constant 3 as well as 4 cannot be on the right hand side of the assignment operator.
4. No error.
5. Error. Double quotes (“”)should be used instead of single quotes (‘’) to enclose the string 2Mar04.

10B:

1. No Error.
2. No Error.
3. Error. On the left hand side of the equal to sign(=), there can only be a variable.
4. No Error.
5. No Error.

11A:

1. No Error.
2. Error. ‘area of circle’ is an invalid variable name.
3. No Error.
4. No Error.
5. Error. Constant 3 cannot be on the left hand side of the assignment operator.

11B:

1. No Error.
2. Error. Double quotes (“”)should be used instead of single quotes (‘’) to enclose the string 2Mar04.
3. No Error.
4. No Error.
5. Error.

12A:

1. True
2. True
3. True
4. True
5. True

12B

1. True
2. True
3. True or false
4. True
5. False C Programs are converted into machine language with the help of a program called assembler. It is a compiler.

13A:

1. False. It can also be used on MS-DOS, Linux and windows operating system.
2. False
3. False
4. True

13B:

1. True
2. False
3. True
4. True
5. True
6. True

14A:

1. False
2. False

Explanation :   
  
At the end of each C instruction we have to put a semicolon. This will inform the C compiler that the instruction has ended. After the semicolon we can write a new instruction. But it is not compulsory to write the new instruction in a new line. The only thing that is compulsory is that before writing a new instruction we have to end the previous instruction with a semicolon. Below we have written instructions in two different ways  
   
Int x; int y; int z;  
  
Or   
  
Int x;  
Int y;  
Int z;  
  
Both these methods of writing instructions are correct.

1. True **The answer to this question can be both true and false. Please read the explanation to understand the reason.**  
     
   Explanation :   
     
   C statements consists of data types, variables and constants. For example, below is a C statement to create an integer variable  
   Int NUMBER = 100;  
     
   In this statement the data type int is written in small case because it is a reserved word in C language, but the variable name is written in capital letters because it is not a reserved word and it is upto us to decide whether we want to write the variable names in small case or capital case. Therefore saying that usually all C statements are written in small case letters will not be 100% correct. It depends from programmer to programmer.

Explanation :   
  
A C statement consists of data types, variables, constants and other programming statements like conditions, loops etc. When writing a C statement it is mandatory to put a space between data types, variables, constants etc. For example, the statement written below is wrong  
  
intnumber=10;  
  
Here we have not put any space between int and number, so the compiler will not be able to know that we are trying to declare an integer variable called number and give it value 10. The correct way to write this statement will be   
  
Int number = 10;  
  
By putting a space between int and number we are telling the compiler that these are two different words. Therefore putting a space between two words in a C statement is not an option, it is a mandatory thing. At the same time you have to keep in mind that you cannot put a space between the name of a variable, constant or a reserved word. You cannot write int as i n t ( by putting a space between i, n and t)

1. True

14B:

1. True
2. False
3. True
4. False
5. True
6. True

15A:

1. True
2. False
3. True
4. False
5. True

15B

1. B
2. D
3. B

16A:

1. D
2. A
3. None of these

16B

1. D
2. B
3. D

17A

1. D
2. D
3. A

17B

1. C
2. A
3. B

18A

1. D
2. A
3. B

18B

1. B
2. A
3. D

19A

1. A
2. D
3. D

19B

1. B
2. A
3. B

20A

1. 10 \* x
2. 6
3. 5
4. -9
5. 0.285714
6. 21754

20B

1. Z = ((x + 3) \* x\*x\*x)/((y-4) \* (y+5))
2. R = (2 \* v + 6.22 \*(c + d))/(g + v)

21A

1. A = ((7.7 \*b) \* (x \* y + a)/c – 0.8 + 2 \* b)/((x + a) \* (1/y))
2. X = (12 \* x \* x \* x/4 \* x) + (8 \* x \* x \*x/4 \* x) + (x/8 \* x) + (8/8 \* x)

21B

1. No Error
2. No Error
3. No error
4. Error. On the left-hand side of the equal to (=) sign, there can only be a variable.
5. Error. a ^ 3 is not a valid statement. Instead we can use a \* a \* a.
6. No Error.

22A

1. No error.
2. Error. Area of circle is an invalid variable name.
3. No error.
4. Error. There needs to be \* sign before parenthesis.
5. Error. A variable name must be present on the left hand side of ‘=’ operator.
6. No Error
7. Error. Multiple characters cannot be stored in a char value.

22B

ans = **5 \* 2** \*2 \* 5 – 3 \* 3 \* 4 \* 4 – 8 \* 2 \* 2 \* 5 + 10 \* 3 \* 4 operation: \*

ans = **10 \* 2** \* 5 – 3 \* 3 \* 4 \* 4 – 8 \* 2 \* 2 \* 5 + 10 \* 3 \* 4 operation: \*

ans = **20 \* 5** – 3 \* 3 \* 4 \* 4 – 8 \* 2 \* 2 \* 5 + 10 \* 3 \* 4 operation: \*

ans = 100 - **3 \* 3** \* 4 \* 4 – 8 \* 2 \* 2 \* 5 + 10 \* 3 \* 4 operation: \*

ans = 100 – **9 \* 4** \* 4 – 8 \* 2 \* 2 \* 5 + 10 \* 3 \* 4 operation: \*

ans = 100 – **36 \* 4** – 8 \* 2 \* 2 \* 5 + 10 \* 3 \* 4 operation: \*

ans = 100 – 144 – **8 \* 2** \* 2 \* 5 + 10 \* 3 \* 4 operation: \*

ans = 100 – 144 – **16 \* 2** \* 5 + 10 \* 3 \* 4 operation: \*

ans = 100 – 144 – **32 \* 5** + 10 \* 3 \* 4 operation: \*

ans = 100 – 144 – 160 + **10 \* 3** \* 4 operation: \*

ans = 100 – 144 – 160 + **30 \* 4** operation: \*

ans = **100 – 144** – 160 + 120 operation: -

ans = **-44 – 160** + 120 operation: -

ans = **-204 + 120** operation: +

ans = -84

23A

res = **4 \* 4** \* 1/3 – 4 \* 1/3 operation: \*

res = **16 \* 1**/3 – 4 \* 1/3 operation: \*

res = **16/3** – 4 \* 1/3 operation: /

res = 5 – **4 \* 1**/3 operation: \*

res = 5 – **4/3** operation: /

res = **5 – 1** operation: -

res = 4

23B

s = 4.1 + 2.2 \* 3.0 \* 3.0 /0.0 operation: \*

s = 4.1 + 6.6 \* 3.0 / 0.0 operation: \*

s = 4.1 + 19.8/0.0 operation: /

Here we cannot Divide by 0

24A

R = 3.5 \* 3.5 + 2 \* 3.5 +1/2 \* 3.5 \*3.5 + 3.5 +1 operation: \*

R = 12.25 + 2 \* 3.5 + 1 / 2 \* 3.5 \* 3.5 + 3.5 +1 operation: \*

R = 12.25 + 7.0 + 1 / 2 \* 3.5 \* 3.5 + 3.5 +1 operation: \*

R = 12.25 + 7.0 + 1 / 7.0 \* 3.5 + 3.5 +1 operation: \*

R = 12.25 + 7.0 + 1 / 24.5 + 3.5 +1 operation: /

R = 12.25 + 7.0 + 0.04081 + 3.5 + 1 operation: +

R = 19.29081 + 3.5 + 1 operation: +

R = 22.79081 + 1 operation: +

R= 23.75 operation: +

24B

G = 2 / 2 + 2 \* 4 / 2 – 2 + 2.5 / 3 operation : /

G = 1 + 2 \* 4 / 2 – 2 + 2.5 / 3 operation : \*

G = 1 + 8 / 2 – 2 + 2.5 / 3 operation : /

G = 1 + 4 – 2 + 2.5 / 3 operation : /

G = 1 + 4 – 2 + 0.83 operation : /

G = 5 – 2 + 0.83 operation : /

G = 3 + 0.83 operation : /

G = 3.83 operation : +

25A

on = 4 \* 1 / 2 + 3 / 2 \* 1 + 2 + 3.2 operation : \*

on = 4 / 2 + 3 / 2 \* 1 + 2 + 3.2 operation : /

on = 2 + 3 / 2 \* 1 + 2 + 3.2 operation : /

on = 2 + 1 \* 1 + 2 + 3.2 operation : \*

on = 2 + 1 + 2 + 3.2 operation : +

on = 3 + 2 + 3.2 operation : +

on = 5 + 3.2 operation : +

on = 8 operation : +

25B

on = 4 \* 1 / 2 + 3 / 2 \* 1 + 2 + 3.6 operation : \*

on = 4 / 2 + 3 / 2 \* 1 + 2 + 3.6 operation : /

on = 2 + 3 / 2 \* 1 + 2 + 3.6 operation : /

on = 2 + 1 \* 1 + 2 + 3.6 operation : \*

on = 2 + 1 + 2 + 3.6 operation : +

on = 3 + 2 + 3.6 operation : +

on = 5 + 3.6 operation : +

on = 9 operation : +

26A

S = 4 \* 2 / 4 – 6 / 2 + 2 / 3 \* 6 / 2 operation : \*

S = 8 / 4 – 6 / 2 + 2 / 3 \* 6 / 2 operation : /

S = 2 – 6 / 2 + 2 / 3 \* 6 / 2 operation : /

S = 2 – 3 + 2 / 3 \* 6 / 2 operation : /

S = 2 – 3 + 0 \* 6 / 2 operation : \*

S = 2 – 3 + 0 / 2 operation : /

S = 2 – 3 + 0 operation : -

S = -1 + 0 operation : -

S = -1 operation : +

26B

S = **1 / 3** \* 4 / 4 – 6 / 2 + 2 / 3 \* 6 / 3 operation: /

S = **0 \* 4** / 4 – 6 / 2 + 2 / 3 \* 6 / 3 operation: \*

S = **0 / 4** – 6 / 2 + 2 / 3 \* 6 / 3 operation: /

S = 0 – **6 / 2** + 2 / 3 \* 6 / 3 operation: /

S = 0 – 3 + **2 / 3** \* 6 / 3 operation: /

S = 0 – 3 + **0 \* 6** / 3 operation: \*

S = 0 – 3 + **0 / 3** operation: /

S = **0 – 3** + 0 operation: -

S = **-3 + 0** operation: +

S = -3

27A

1. Z = ((8.8 \* (a + b) \* 2 / c) – (0.5 + 2 \* a / (q + r)))/((a + b) \* (1 / m))
2. X = (-b + (b \* b) + 2 \* 4 \* a \* c) / (2 \* a)

27B

1. **Semicolon** is a separator which represents End of statement.
2. **Literal**
3. Identifiers.
4. ‘Address of’ operator
5. Output Function
6. Input Function

28A

1. Escape sequence.
2. Real constant
3. Integer constant
4. Character constant
5. A real constant expressed in exponential form.

30A

%i

%o

%x

%l

%h

%u

%c

%s

%f

%lf

%e

%g

30B

1. 0 to 255

2. 1

3. -128 to 127

4. 1

5. -128 to 127

6. 1

7. -32768 to 32768

8. 2

9. 0 to 65535

10. 2

31A

1. 0 to 65535
2. 2
3. -2147483648 to 2147483647
4. 4
5. 0 to 4294967295
6. 4
7. 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
8. 8
9. 0 to 18,446,744,073,709,551,615
10. 8

31B

1. 1.175E-38   to 3.402E38
2. 4
3. 2.225E-308 to 1.79E + 308
4. 8
5. No value
6. 12

36B

1. No error. Multiple C statements can be written in a single line.
2. No error. The list being printed in printf() may contain variables, constants and expressions.

37A

1. Error: We should use & before each variable used in scanf()
2. Error: Semicolon should be present at the end of type declaration, printf() and scanf() statements.

37B

1. 0 2 0.000000 2.000000
2. a = 2 b = -2 c = 2 d = -2

38A

1. Error Mod(%) operator cannot be used on floats.
2. Since spaces are given after and before double quotes in scanf() we must supply a space, then two numbers and again a space followed by enter. The printf() would then output the two number that you enter.

38B

1. nn

nn /n/n nn/n

1. 20. Whenever there is a conflict between a local variable and a global variable it is the local variable that gets a priority.

39A

1. 20 40. In case of a conflict between two local variables the one that is more local gets the priority. More local means the one that is nearer to the point of usage. In this case the point of usage is the call to printf() function.

39B

D

Because return value of the function scanf() would act as a result of the condition. This value will always be non-zero, hence true.

40A

1. Scope indicates the region over which the variable’s declaration has an effect. The four kinds of scopes are – file, function, block and prototype.
2. First is declaration, second is definition

40B

There are two differences between a declaration and a definition.

In the definition of a variable and some initial value is given to it, whereas a declaration only identifies the type of the variable. Thus, definition is the place where the variable is created or assigned storage, whereas declaration refers to places where the nature of the variable is stated but no storage is allocated.

Secondly, redefinition is an error, whereas, redeclaration is not an error.

41A

extern int x; ------------------------🡪 Declaration

float y;------------------------------🡪 Definition

double pow (double, double); ----------🡪 Declaration

float square (float x) { … } -----------------🡪Definition

41B

D. extern int I is a declaration and not a definition, hence the error.

42A

1. Yes.
2. extern int a is the declaration, whereas int a = 20 is the definition.

42B

A

43A

True

Extern int z

43B

Yes.

We are declaring it. When the function, along with the statements belonging to it is mentioned, we are defining the function.

There is no difference except for the fact that the first one gives hint that the function fun() is probably in another source file.

44A

The error is reported because display() is being called before it is defined. To avoid this error we must declare the prototype of display() at the top.

44B

D. The error occurs because we have mixed the ANSI prototype with Kernighan & Ritchie style of function definition. When we use ANSI prototype for a function and pass a float to the function it is promoted to a double. When the function accepts this double into a float a type mismatch occurs hence the error. The remedy for this error coule be to define the function as:

int fun(float aa)

{

…

}