Chapter 1

Programming with C

1. Choose the correct statement.
   (a) Use of `goto` enhances the logical clarity of a code.
   (b) Use of `goto` makes the debugging task easier.
   (c) Use `goto` when you want to jump out of a nested loop.
   (d) Never use `goto`.
2. Which is true of conditional compilation?
   (a) It is taken care of by the compiler.
   (b) It is setting the compiler option conditionally.
   (c) It is compiling a program based on a condition.
   (d) It is taken care of by the pre-processor.
3. C was primarily developed as a
   (a) systems programming language
   (b) general purpose language
   (c) data processing language
   (d) none of the above
4. C is a
   (a) high level language
   (b) low level language
   (c) high level language with some low level features.
   (d) low level language with some high level features.
5. Even if a particular implementation doesn't limit the number of characters in an identifier, it is advisable to be concise because
   (a) chances of typographic errors are less
   (b) it may be processed by assembler, loaders, etc., which may have their own rules that may contradict the language rules

(c) by being concise, one can be mnemonic

(d) none of the above

***6.** The minimum number of temporary variables needed to swap the contents of two variables is

(a) 1 (b) 2 (c) 3 (d) 0

7. The purpose of the following program fragment

```
b = s + b;
s = b - s;
b = b - s;
```

where s,b are two integers is to

(a) transfer the contents of s to b

(b) transfer the contents of b to s

(c) exchange (swap) the contents of s and b

(d) negate the contents of s and b

8. Consider the function

```
find(int x, int y)
{return(( x < y) ? 0 : ( x - y));}
```

Let a,b be two non-negative integers. The call find(a, find(a, b)) can be used to find the

(a) maximum of a, b (b) positive difference of a, b

(c) sum of a, b (d) minimum of a, b

9. Let a,b be two non-negative integers. Which of the following calls, finds the positive difference of a and b?

(a) find(a,b) + find(b,a) (b) find(a,find(a,b))

(c) a + find(a,b) (d) b + find(a,b)

10. If integer needs two bytes of storage, then maximum value of an unsigned integer is

(a) $2^{16} - 1$ (b) $2^{15} - 1$ (c) 2^{16} (d) 2^{15}

***11.** If integer needs two bytes of storage, then maximum value of a signed integer is

(a) $2^{16} - 1$ (b) $2^{15} - 1$ (c) 2^{16} (d) 2^{15}

***12.** printf("%d", printf("tim"));

(a) results in a syntax error (b) outputs tim3

(c) outputs garbage (d) prints tim and terminates abruptly

***13.** If abc is the input, then the following program fragment

```
char x, y, z;
printf("%d", scanf ("%c%c%c", &x, &y, &z)); results in
```

(a) a syntax error (b) a fatal error

(c) segmentation violation (d) printing of 3

***14.** Consider the statements

```
putchar(getchar());
putchar(getchar());
```

If

```
a
b
```

is the input, the output will be

(a) an error message (b) this can't be the input

(c) `ab` (d) `a b`

15. Let `a, b` be two positive integers. Which of the following options correctly relates `/` and `%`?

(a) `b = (a/b)*b + a%b` (b) `a = (a/b)*b + a%b`

(c) `b = (a%b)*b + a/b` (d) `a = (a%b)*b + a/b`

16. Literal means

(a) a string (b) a string constant (c) a character (d) an alphabet

17. Length of the string "correct" is

(a) 7 (b) 8

(c) 6 (d) implementation dependent.

18. Which of the following are true regardless of the implementation?

(a) `sizeof(int)` is not less than `sizeof(long)`

(b) `sizeof(short)` equals `sizeof(int)`

(c) `sizeof(int)` equals `sizeof(unsigned)`

(d) `sizeof(double)` is not less than `sizeof(float)`

19. Coercion

(a) takes place across an assignment operator.

(b) takes place if an operator has operands of different data types.

(c) means casting.

(d) none of the above.

20. Choose the correct statements.

(a) Casting refers to implicit type conversion.

(b) Coercion refers to implicit type conversion.

(c) Casting refers to explicit type conversion.

(d) Coercion refers to explicit type conversion.

21. Consider the following program fragment

```
char c = 'a';
while (c++ <= 'z')
 putchar(xxx);
```

If the required output is `abcdefghijklmnopqrstuvwxyz`, then `xxx` should be

(a) `c` (b) `c++` (c) `c-1` (d) `--c`

22. Which of the following comments are true?

(a) C provides no input-output features.

(b) C provides no file access features.

(c) C borrowed most of its ideas from BCPL.

(d) C provides no features to manipulate composite objects.

*23. If `y` is of integer type then the expressions

```
    3 * (y - 8) / 9 and (y - 8) / 9 * 3
```

(a) must yield the same value (b) must yield different values
(c) may or may not yield the same value (d) none of the above

24. If `y` is of integer type then the expressions

```
    3 * (y - 8) / 9 and (y - 8) / 9 * 3
```

yield the same value if
(a) `y` is an even number (b) `y` is an odd number
(c) `y - 8` is an integral multiple of 9 (d) `y - 8` is an integral multiple of 3

25. Integer division results in
(a) truncation (b) rounding (c) overflow (d) none of the above

26. Which of the following comments about EOF are true?
(a) Its value is defined within `stdio.h`.
(b) Its value is implementation dependent.
(c) Its value can be negative.
(d) Its value should not equal the integer equivalent of any character.

*27. The value of an automatic variable that is declared but not initialized will be
(a) 0 (b) –1 (c) unpredictable (d) none of the above

28. Choose the correct statements.
(a) An identifier may start with an underscore.
(b) An identifier may end with an underscore.
(c) `IF` is a valid identifier.
(d) The number of significant characters in an identifier is implementation dependent.

29. Choose the correct statements.
(a) Constant expressions are evaluated at compile time.
(b) String constants can be concatenated at compile time.
(c) Size of array must be known at compile time.
(d) None of the above.

30. The `const` feature can be applied to
(a) an identifier (b) an array
(c) an array argument (d) none of the above

31. Which of the following operators takes only integer operands?
(a) + (b) * (c) / (d) %

32. In an expression involving || operator, evaluation
(a) will be stopped if one of its components evaluates to false
(b) will be stopped if one of its components evaluates to true
(c) takes place from right to left
(d) takes place from left to right

33. The statement

```
    if(myPtr != NULL)
        *myPtr = NULL ;
    else
        *myPtr = NULL ;
```

has the same effect as the statement(s)

(a) if(myPtr)   *myPtr = NULL ;
    else   *myPtr = NULL ;

(b) *myPtr = NULL;

(c) if(!myPtr) *myPtr = NULL ;
    else *myPtr = NULL ;

(d) if(myPtr == NULL)   *myPtr = NULL ;
    else *myPtr = NULL ;

34. Pick the operators that associate from the left.

(a) + (b) , (c) = (d) <

35. Pick the operators that associate from the right.

(a) ?: (b) += (c) = (d) !=

36. The operators ., ||, <, =, if arranged in the ascending order of precedence reads

(a) . , || , < , = (b) = , <, || , .

(c) = , || , < , . (d) < , || , = , .

37. Pick the operators whose meaning is context dependent.

(a) * (b) #

(c) & (d) No such operator exists.

38. Pick the operators that associate from the left.

(a) && (b) || (c) ?: (d) ,

***39.** The following code fragment

```
    int x, y = 2, z, a;
    x = (y *= 2) + (z = a = y);
        printf ("%d", x);
```

(a) prints 8

(b) prints 6

(c) prints 6 or 8 depending on the compiler implementation

(d) is syntactically wrong

***40.** If n has the value 3, then the output of the statement printf("%d %d", n++, ++n);

(a) is 3 5 (b) is 4 5

(c) is 4 4 (d) is implementation dependent

41. x -= y + 1; means

(a) x=x - y + 1 (b) x =-x - y - 1

(c) x=-x + y + 1 (d) x =x - y - 1

42. Which of the following comments about the ++ operator are correct?

(a) It is a unary operator.

(b) The operand can come before or after the operator.

(c) It cannot be applied to an expression.

(d) It associates from the right.

43. In standard C, trigraphs in the source program are translated

(a) before the lexical analysis

(b) after the syntax analysis

(c) before the recognition of escape characters in strings

(d) during the intermediate code generation phase

***44.** The expression `5 - 2 - 3 * 5 - 2` will evaluate to 18, if

(a) – is left associative and * has precedence over –

(b) – is right associative and * has precedence over –

(c) – is right associative and – has precedence over *

(d) – is left associative and – has precedence over *

45. `printf("%c", 100);`

(a) prints 100 (b) prints the ASCII equivalent of 100

(c) prints garbage (d) none of the above

***46.** The program fragment

```
    int i = 263;
    putchar(i);
```

(a) prints `263` (b) prints the ASCII equivalent of `263`

(c) rings the bell (d) prints garbage

47. Which of the following comments regarding the reading of a string, using `scanf` (with `%s` option) and `gets`, is true?

(a) Both can be used interchangeably.

(b) `scanf` is delimited by end of line, while `gets` is not.

(c) `scanf` is delimited by blank space, while `gets` is not.

(d) None of the above.

***48.** The following statement

```
    printf("%f", 9/5);
```

prints

(a) `1.8` (b) `1.0` (c) `2.0` (d) none of the above

49. The statement

```
    printf("%f", (float)9/5);
```

prints

(a) `1.8` (b) `1.0` (c) `2.0` (d) none of the above

***50.** Which of the following are not keywords in C?

(a) `printf` (b) `main` (c) `IF` (d) none of the above

***51.** The following program fragment

```
    unsigned i = 1;
    int j = -4;
    printf("%u", i + j);
```

prints

(a) garbage

(b) –3

(c) an integer that changes from machine to machine

(d) none of the above

***52.** If the following program fragment (assume negative numbers are stored in 2's complement form)

```
    unsigned i = 1;
    int j = -4 ;
    printf("%u", i + j);
```

prints x, then `printf("%d", 8 * sizeof(int));`

outputs an integer that is same as (`log` in the answers are to the base two)

(a) an unpredictable value (b) `8 * log(x+3)`

(c) `log(x+3)` (d) `none of the above`

53. Choose the statements that are syntactically correct.

(a) `/* Is /* this a valid */ comment */`

(b) `for(;;);`

(c) `return;`

(d) `return(5+2);`

***54.** The following program fragment

```
      for(i = 3; i < 15; i += 3);
      printf("%d", i);
```

results in

(a) a syntax error (b) an execution error

(c) printing of `12` (d) printing of `15`

***55.** The following program fragment

```
    for (i = 1; i < 5; ++i)
    if (i == 3) continue;
    else printf("%d ", i);
```

results in the printing of

(a) `1 2 4 5` (b) `1 2 4` (c) `2 4 5` (d) none of the above

56. The following program fragment

```
    if (a = 0)
    printf("a is zero");
    else
    printf("a is not zero");
```

results in the printing of

(a) `a is zero` (b) `a is not zero`

(c) nothing (d) garbage

57. The following program fragment

```
if(a = 7)
printf("a is seven");
else
printf("a is not seven");
```

results in the printing of

(a) a is seven

(b) a is not seven

(c) nothing

(d) garbage

***58.** The following program fragment

```
int k = -7;
printf("%d", 0 < !k);
```

(a) prints 0

(b) prints a non-zero value

(c) is illegal

(d) prints an unpredictable value

59. The following loop

```
for(putchar('c'); putchar('a'); putchar('r'))
putchar('t');
```

outputs

(a) a syntax error

(b) cartrt

(c) catrat

(d) catratratratrat...

60. The following loop

```
for(i = 1, j = 10; i < 6; ++i, --j)
printf("%d %d ", i, j);
```

prints

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

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

(c) 1 1 1 1 1 9 9 9 9 9

(d) none of the above

61. The following program fragment

```
int a = 4, b = 6;
printf("%d", a == b);
```

(a) outputs an error message

(b) prints 0

(c) prints 1

(d) none of the above

62. The following program fragment

```
int a = 4, b = 6;
printf("%d", a != b);
```

(a) outputs an error message

(b) prints 0

(c) prints 1

(d) none of the above

63. The following program fragment

```
int a = 4, b = 6;
printf("%d", a = b);
```

(a) outputs an error message

(b) prints 0

(c) prints 1

(d) none of the above

64. A possible output of the following program fragment

```
for(i = getchar();; i = getchar())
if(i == 'x') break;
else putchar(i);
```

is

(a) `mi` (b) `mix` (c) `mixx` (d) none of the above

65. The following program

```
main()
{
   int i = 5;
  if (i == 5) return;
  else printf("i is not five");
  printf("over");
}
```

results in

(a) a syntax error (b) an execution error

(c) printing of `over` (d) execution termination, without printing anything

***66.** The following program fragment

```
int i = 5;
do {putchar(i + 100); printf("%d", i--);}
while(i);
```

results in the printing of

(a) i5h4g3f2el (b) i4h3g2fle0

(c) an error message (d) none of the above

***67.** The following program fragment

```
int i = 107, x = 5;
printf((x > 7)? "%d": "%c", i);
```

results in

(a) an execution error (b) a syntax error

(c) printing of `k` (d) none of the above

***68.** Replacing > by < in the previous question results in

(a) printing of `107` (b) a syntax error

(c) printing of `k` (d) none of the above.

***69.** The following loop

```
while(printf("%d", printf("az")))
printf("by");
```

(a) prints `azbybybyby...` (b) prints `azbyazbyazbyazby...`

(c) results in a syntax error (d) none of the above

70. The following statements

```
for(i = 3; i < 15; i += 3)
{ printf("%d ", i);
  ++i;
}
```

will result in the printing of

(a) 3 6 9 12 (b) 3 6 9 12 15 (c) 3 7 11 (d) 3 7 11 15

71. If a = 9, b = 5 and c = 3, then the expression `(a - a/b * b%c) > a%b%c` evaluates to

(a) true (b) false (c) invalid (d) 0

72. In a `for` loop, if the condition is missing, then,

(a) it is assumed to be present and taken to be false

(b) it is assumed to be present and taken to be true

(c) it results in a syntax error

(d) execution will be terminated abruptly

73. In a `for` loop, if the condition is missing, then infinite looping can be avoided by a

(a) `continue` statement (b) `goto` statement

(c) `return` statement (d) `break` statement

74. Choose the correct statement.

(a) `0` represents a false condition.

(b) Non-zero value represents a false condition.

(c) `1` represents a false condition.

(d) Anything that is not `1`, represents a false condition.

75. Which of the following comments about `for` loop are correct?

(a) Index value is retained outside the loop.

(b) Index value can be changed from within the loop.

(c) `goto` can be used to jump, out of loop.

(d) Body of the loop can be empty.

76. Which of the following comments about `for` loop are correct?

(a) Using `break` is equivalent to using a `goto` that jumps to the statement immediately following the loop.

(b) Continue is used to by-pass the remainder of the current pass of the loop.

(c) If comma operator is used, then the value returned is the value of the right operand.

(d) It can always be replaced by a `while` loop.

77. Choose the correct answers.

(a) `for` loops can be nested

(b) Nested `for` loop can't use the same index variable

(c) Nested `for` loop can't overlap

(d) None of the above

78. Consider the following program fragment

```
if (a > b)
if (b > c)
s1;
else s2;
```

s2 will be executed if

(a) a <= b (b) b > c (c) b <= c and a <= b (d) a > b and b <= c

79. If switch feature is used, then

(a) default case must be present

(b) default case, if used, should be the last case

(c) default case, if used, can be placed anywhere

(d) none of the above

80. The switch feature

(a) can always be replaced by a nested if-then-else clause

(b) enhances logical clarity

(c) can't always be replaced by a nested if-then-else clause

(d) none of the above

81. break statement can be simulated by using

(a) goto (b) return

(c) exit (d) any of the above features

***82.** The following program fragment

```
if (2 < 1)
;
else
x = (2 < 0)? printf("one") : printf("four");
printf ("%d", x);
```

(a) prints nothing (b) results in a syntax error

(c) prints four0 (d) none of the above

***83.** Consider the following program fragment

```
if (a > b)
   printf("a > b");
else
   printf("else part");
  printf("a <= b");
```

a <= b will be printed if

(a) a > b (b) a < b (c) a = = b (d) none of the above

84. Consider the following flow chart.

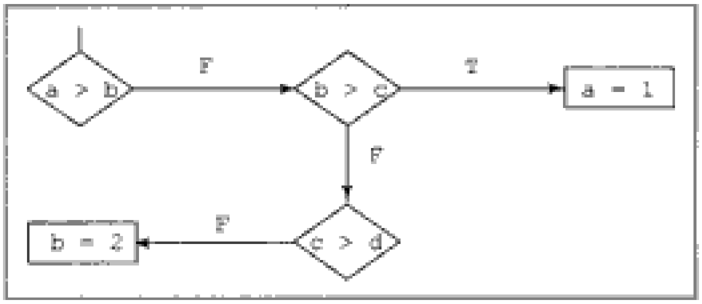

Fig. 1.1

Which of the following, correctly implements the above flow chart?

(a)
```
if (a > b)
if (b > c)
a = 1;
else if (c > d)
b = 2;
```

(b)
```
if (a <= b)
if (b > c)
a = 1;
else if (c <= d)
b = 2;
```

(c)
```
if (a > b)
;
else if (b > c)
a = 1;
else if (c <= d)
b = 2;
```

(d)
```
if (a > b)
;
else if (b > c)
a = 1;
else if (c > d)
;
else b = 2;
```

***85.** The body of the following `for` loop

```
for(putchar('a'); putchar(0); putchar('c'))
   putchar('b');
```

will be executed

(a) 0 times

(b) 1 time

(c) infinitely many times

(d) will not be executed because of syntax error

86. The following statement

```
if (a > b)
if (c > b)
printf("one");
else
if (c == a) printf("two");
else printf("three");
else printf("four");
```

(a) results in a syntax error

(b) prints `four` in c <= b

(c) prints `two` if c <= b

(d) prints `four` in a <= b

87. The above statement can never print

(a) `one` (b) `two` (c) `three` (d) `four`

88. The following program fragment

```
int x = 4, y = x, i;
for (i = 1; i < 4; ++i)
    x += x;
```

outputs an integer that is same as

(a) `8 * y` (b) `y * (1 + 2 + 3 + 4)`

(c) `y * 4` (d) `y * y`

89. Using `goto` inside `for` loop is equivalent to using

(a) `continue` (b) `break` (c) `return` (d) none of the above

***90.** Choose the correct statements.

(a) All the elements of the array should be of the same data type and storage class.

(b) The number of subscripts determines the dimension of the array.

(c) The array elements need not be of the same storage class.

(d) In an array definition, the subscript can be any expression yielding a non-zero integer value.

91. Consider the declaration

```
    static char hello[]="hello";
```

The output of `printf("%s\n", hello);`
will be the same as that of

(a) `puts("hello");` (b) `puts(hello);`

(c) `printf("%s\n", "hello");` (d) `puts("hello\n");`

***92.** If storage class is missing in the array definition, by `default` it will be taken to be

(a) automatic

(b) external

(c) static

(d) either automatic or external depending on the place of occurrence.

93. The following program fragment

```
    int x[5][5], i, j;
    for(i = 0; i < 5; ++i)
       for(j = 0; j < 5 ; j++)
         x[i][j] = x[j][i];
```

(a) transposes the given matrix `x` (b) makes the given matrix `x`, symmetric

(c) doesn't alter the matrix `x` (d) none of the above

94. Which of the following features of C is meant to provide reliable access to special memory locations?

(a) static_const (b) pragma (c) volatile (d) immutable

95. Consider the array definition

```
    int num[10] = {3, 3, 3};
```

Pick the correct answers.

(a) num[9] is the last element of the array num

(b) The value of `num[8]` is 3

(c) The value of `num[3]` is 3

(d) None of the above.

96. Consider the following type definition.

```
typedef  char x[10];
x myArray[5];
```

What will sizeof(myArray) be? (Assume one character occupies 1 byte)

(a) 15 bytes (b) 10 bytes (c) 50 bytes (d) 30 bytes

97. While passing an array as an actual argument, the function call must have the array name

(a) with empty brackets (b) with its size

(c) alone (d) none of the above

***98.** The following program

```
    main( )
    {
        static int a[] = {7,8,9};
        printf(“%d”, 2[a] + a[2]);
    }
```

(a) results in bus error (b) results in segmentation violation error

(c) will not compile successfully (d) none of the above

99. The parameter passing mechanism for an array is

(a) call by value (b) call by value-result

(c) call by reference (d) none of these

100. Consider the statement

```
    int val[2][4] = {1, 2, 3, 4, 5, 6, 7, 8};
```

4 will be the value of

(a) `val[1][4]` (b) `val[0][4]`

(c) `val[1][1]` (d) none of the above

101. The maximum number of dimension an array can have in C is

(a) 3 (b) 4 (c) 5 (d) compiler dependent

102. The following program fragment

```
    int m, n, b = m = n = 8;
    char wer[80];
    sprintf(wer, "%d%d%d", m, n, b);
    puts(wer);
```

(a) prints the string `8 8 8` (b) prints the null string

(c) prints the string `888` (d) none of the above

103. Under which of the following conditions, the size of an one-dimensional array need not be specified?

(a) when initialization is a part of definition

(b) when it is a declaration

(c) when it is a formal parameter

(d) when it is an actual argument

104. If a two dimensional array is used as a formal parameter, then

(a) both the subscripts may be left empty

(b) the first (row) subscript may be left empty

(c) the first subscript must be left empty

(d) both the subscripts must be left empty

***105.** The following program

```
main()
{
    static char a[3][4] = {"abcd", "mnop", "fghi"};
    putchar(**a);
}
```

(a) will not compile successfully (b) results in run-time error

(c) prints garbage (d) none of the above

***106.** C does no automatic array bound checking. This is

(a) true (b) false (c) C's asset (d) C's shortcoming

Answer the next three questions based on the program fragment given below

```
int hh = 16;
static char wer[] = "NO SUBSTITUTE FOR HARD WORK";
```

107. `printf("%10.5s", wer);`

outputs

(a) `     NO SU`

(b) `NO SUBSTIT`

(c) `NO SU`

(d) `UTE F`

108. `printf("%-10.5s", wer);`

outputs

(a) `     NO SU`

(b) `NO SUBSTIT`

(c) `NO SU`

(d) `UTE F`

109. `printf("%-10.*s", hh, wer);`
outputs

(a) `NO  SU` (b) `NO  SUBSTITUTE  FO`

(c) `NO  SU` (d) error message

110. If `n` has the value `3`, then the statement `a[++n] = n++;`

(a) assigns `3` to `a[5]` (b) assigns `4` to `a[5]`

(c) assigns `4` to `a[4]` (d) what is assigned is compiler-dependent

111. Choose the statement that best defines an array.

(a) It is a collection of items that share a common name.

(b) It is a collection of items that share a common name and occupy consecutive memory locations.

(c) It is a collection of items of the same type and storage class that share a common name and occupy consecutive memory locations.

(d) None of the above.

***112.** Choose the correct statements.

(a) Strictly speaking C supports 1-dimensional arrays only.

(b) An array element may be an array by itself.

(c) Array elements need not occupy contiguous memory locations.

(d) None of the above.

***113.** The order in which actual arguments are evaluated in a function call

(a) is from the left (b) is from the right

(c) is compiler-dependent (d) none of the above

114. If a global variable is of storage class static, then

(a) the static declaration is unnecessary if the entire source code is in a single file

(b) the variable is recognized only in the file in which it is defined

(c) it results in a syntax error

(d) none of the above

***115.** Which of the following statements are correct?

(a) It is possible for a function to access a variable that is local to another function.

(b) Two local variables may have the same name.

(c) The scope of a local variable should be a function.

(d) The scope of a local variable may be a single statement.

***116.** The default parameter passing mechanism is

(a) call by value (b) call by reference

(c) call by value result (d) none of the above

117. Choose the correct statements.

(a) During external variable definition, storage is set aside by the compiler.

(b) During external variable declaration, no storage is set aside by the compiler.

(c) The use of external variables may make debugging difficult.

(d) None of the above.

118. The storage class static can be used to

(a) restrict the scope of an external identifier

(b) preserve the exit value of variables

(c) provide privacy to a set of functions

(d) none of the above

***119.** The following program

```
main()
{printf("tim");
  main();}
```

(a) is illegal (b) keeps on printing `tim`

(c) prints `tim` once (d) none of the above

***120.** Consider the following program.

```
main()
{ putchar(‘M’);
   first();
   putchar(‘m’); }
first()
{ ________ }
second()
 { putchar(‘d’); }
```

If `Madam` is the required output, then the body of `first()` must be

(a) empty

(b) `second(); putchar(‘a’);`

(c) `putchar(‘a’); second(); printf("%c", ‘a’);`

(d) none of the above.

121. Use of functions

(a) helps to avoid repeating a set of statements many times

(b) enhances the logical clarity of the program

(c) helps to avoid repeated programming across programs

(d) makes the debugging task easier

122. Which of the following comments about wide characters is/are true?

(a) It is the binary representation of a character in the extended binary set.

(b) It is of integer type wchar_t.

(c) End of file is represented by WEOF.

(d) None of the above.

123. Pick the correct statements.

(a) The body of a function should have only one `return` statement.

(b) The body of a function may have many `return` statements.

(c) A function can `return` only one value to the calling environment.

(d) If `return` statement is omitted, then the function does its job but returns no value to the calling environment.

124. max is a function that returns the larger of the two integers, given as arguments. Which of the following statements finds the largest of three given numbers?

(a) max(max(a, b), max(a, c))

(b) max(a, max(a, c))

(c) max(max(a, b), max(b, c))

(d) max(b, max(a, c))

125. Forward declaration is absolutely necessary

(a) if a function returns a non-integer quantity

(b) if the function call precedes its definition

(c) if the function call precedes its definition and the function returns a non integer quantity

(d) none of the above

126. void can be used

(a) as a data-type of a function that returns nothing to its calling environment

(b) inside the brackets of a function that does not need any argument

(c) in an expression

(d) in a printf statement

127. Any C program

(a) must contain at least one function (b) need not contain any function

(c) needs input data (d) none of the above

***128.** The following program

```
main()
{ int a = 4;
    change(a);
    printf("%d", a);
}
change(a)
int a;
{ printf("%d", ++a);}
```

outputs

(a) 55 (b) 45 (c) 54 (d) 44

129. Choose the best answer.

Storage class defines

(a) the datatype (b) the scope

(c) the scope and permanence (d) the scope, permanence and datatype

130. Which of the following is true of external variables?

(a) They provide a way for two way communication between functions.

(b) Their scope extends from the point of definition through the remainder of the program.

(c) If they are not initialized, they will have garbage value.

(d) None of the above.

***131.** The following program

```
main()
{
  int i = 2;
  { int i = 4, j = 5;
    printf("%d%d", i, j);
  }
  printf("%d%d", i, j);
}
```

(a) will not compile successfully

(b) prints 4525

(c) prints 2525

(d) none of the above

***132.** The following program

```
main()
{
inc(); inc(); inc();
}
inc()
{
static int x;
printf("%d", ++x);
}
```

(a) prints 012

(b) prints 123

(c) prints 3 consecutive, but unpredictable numbers

(d) prints 111

133. printf ("ab", "cd", "ef");
prints

(a) ab

(b) abcdef

(c) abcdef, followed by garbage

(d) none of the above

134. The expression 4 + 6 / 3 * 2 - 2 + 7 % 3 evaluates to

(a) 3 (b) 4 (c) 6 (d) 7

135. Consider the following program segment.

```
i = 6720; j = 4;
while ((i%j)==0)
 { i = i / j;
   j = j + 1;
 }
```

On termination j will have the value

(a) 4 (b) 8 (c) 9 (d) 6720

136. The output of the following program is

```
main()
{ static int x[] = {1, 2, 3, 4, 5, 6, 7, 8};
  int i;
  for(i = 2; i < 6; ++i)
   x[x[i]] = x[i];
  for (i = 0; i < 8; ++i)
   printf ("%d ", x[i]);
}
```

(a) 1 2 3 3 5 5 7 8   (b) 1 2 3 4 5 6 7 8

(c) 8 7 6 5 4 3 2 1   (d) 1 2 3 5 4 6 7 8

***137.** `main ()`

```
{ int a = 5, b = 2;
  printf("%d", a+++b);
}
```

(a) results in syntax error   (b) prints 7

(c) prints 8   (d) none of the above

***138.** The program fragment

```
int a = 5, b = 2;
printf ("%d", a+++++b);
```

(a) prints 7   (b) prints 8   (c) prints 9   (d) none of the above

***139.** Consider the following program

```
main()
{ int x = 2, y = 5;
  if (x < y) return (x = x + y);
  else printf ("z1");
  printf ("z2");
}
```

Choose the correct statements

(a) The output is `z2`   (b) The output is `z1z2`

(c) This will result in compilation error   (d) None of the above

***140.** `puts(argv[0]);`

(a) prints the name of the source code file

(b) prints `argv`

(c) prints the number of command line arguments

(d) prints the name of the executable code file

141. A possible output of the following program fragment

```
static char wer[][5] = {"harmot", "merli", "axari"};
printf(("%d %d %d", wer, wer[0], &wer[0][0]);
```

is

(a) `262164 262164 262164` (b) `262164 262165 262166`
(c) `262164 262165 262165` (d) `262164 262164 262165`

***142.** The following program

```
main()
{printf("%u", main);}
results in
```

(a) printing of a garbage number
(b) an execution error
(c) printing of starting address of the function main
(d) an infinite loop

***143.** The following program

```
main()
{ int abc();
  abc();
  (*abc)();
}
int abc()
{ printf("come"); }
```

(a) results in a compilation error (b) prints `come come`
(c) results in a run time (d) prints `come  come`

The next five questions are based on the following program fragment.

```
static char wer[3][4] = {"bag", "let", "bud"};
char(*ptr)[4] = wer;
```

***144.** The possible output of `printf("%d %d", ptr, ptr+1);` is
(a) `262 262` (b) `262 266` (c) `262 263` (d) `262 265`

145. The possible output of `printf("%d %d", wer[1], wer[1]+1);` is
(a) `162 163` (b) `162 162` (c) `162 166` (d) `162 165`

146. The possible output of `printf("%d %d", wer, wer+1);` is
(a) `262 262` (b) `262 266` (c) `262 263` (d) `262 265`

147. `putchar (*(wer[1]+1));`
(a) prints `e` (b) prints `a` (c) prints `l` (d) prints `b`

148. In which of the following cases will the character `'t'` be printed?
(a) `putchar(*(*(ptr+1) + 2));`
(b) `putchar(*(wer[1] + 2));`
(c) `putchar(*(ptr+1) + 2);`
(d) `none of the above`

149. Choose the correct statements.

(a) Address is the numeric value associated with a memory location.
(b) Two variables can have the same address.
(c) Address is bound to a variable by the compiler.
(d) Value of a variable can be an address.

150. Feature for accessing a variable through its address is desirable because

(a) call by reference can be simulated
(b) call by value can be simulated
(c) a function can return more than one value
(d) excessive use of global variables can be avoided

151. `int i = 5;`

is a statement in a C program. Which of the following are true?

(a) During execution, value of `i` may change but not its address
(b) During execution both the address and value may change
(c) Repeated execution may result in different addresses for `i`
(d) `i` may not have an associated address

152. Choose the correct statements.

(a) Address operator cannot be applied to register variables.
(b) Address operator can be applied to register variables.
(c) Misuse of register declaration will increase the execution time.
(d) None of the above.

***153.** Choose the best answer.

Prior to using a pointer variable

(a) it should be declared
(b) it should be initialized
(c) it should be both declared and initialized
(d) none of the above

154. The operators > and < are meaningful when used with pointers, if

(a) the pointers point to data of similar type
(b) the pointers point to structure of similar data type
(c) the pointers point to elements of the same array
(d) none of these

155. A set of names can be represented as a

(a) two-dimensional array of characters
(b) one-dimensional array of strings
(c) one-dimensional array of pointers to character
(d) none of these

156. If `arr` is a two dimensional array of 10 rows and 12 columns, then `arr[5]` logically points to the

(a) sixth row (b) fifth row (c) fifth column (d) sixth column

157. While sorting a set of names, representing the names as an array of pointers is preferable to representing the names as a two dimensional array of characters, because

(a) storage needed will be proportional to the size of the data

(b) execution will be faster

(c) swapping process becomes easier and faster

(d) none of the above

158. The statement

```
int **a;
```

(a) is illegal

(b) is legal but meaningless

(c) is syntactically and semantically correct

(d) none of the above

***159.** Consider the following declaration.

```
int a, *b = &a, **c = &b;
```

The following program fragment

```
a = 4;
**c = 5;
```

(a) does not change the value of a
(b) assigns address of c to a

(c) assigns the value of b to a
(d) assigns 5 to a

***160.** If the statement

```
b = (int *)**c;
```

is appended to the above program fragment, then

(a) value of b is unaffected
(b) value of b will be the address of c

(c) value of b becomes 5
(d) none of these

161. Consider the two declarations

```
void *voidPtr ;
char *charPtr ;
```

Which of the following assignments are syntactically correct?

(a) `voidPtr = charPtr`
(b) `charPtr = voidPtr`

(c) `*voidPtr = *charPtr`
(d) `*charPtr = voidPtr`

162. Which of the following operators can be applied to pointer variable(s)?

(a) Division (b) Multiplication (c) Casting (d) None of these

***163.** Pointers are of

(a) integer datatype
(b) character datatype

(c) unsigned integer datatype
(d) none of these

164. The address operator &, cannot act on

(a) R-values
(b) arithmetic expressions

(c) members of a structure
(d) local variables

165. Consider the following program fragment.

```
int v = 3, *pv = &v;
printf ("%d %d", v, *pv);
```

The output will be

(a) an error message
(b) 3 address of v
(c) 3 3
(d) none of the above

166. If the two statements

```
*pv = 0;
printf ("%d %d", *pv, v);
```

are appended to the previous program fragment, then the output will be

(a) 0 3
(b) 0 0
(c) unpredictable
(d) none of the above

167. A pointer variable can be

(a) passed to a function as argument
(b) changed within a function
(c) returned by a function
(d) can be assigned an integer value

168. A string that is a formal parameter can be declared

(a) an array with empty bracket
(b) a pointer to character
(c) a pointer to a character
(d) none of the above

169. Choose the correct statements.

(a) An entire array can be passed as argument to a function.
(b) A part of an array can be passed as argument to a function.
(c) Any change done to an array that is passed as an argument to a function will be local to the function.
(d) None of these.

170. Consider the following program.

```
main()
{
    char x[10], *ptr = x;
    scanf("%s", x);
    change(&x[4]);
}
change(char a[])
{puts(a);}
```

If abcdefg is the input, the output will be

(a) abcd
(b) abc
(c) efg
(d) garbage

171. For the previous problem the function calls

`change(x);` and `change(ptr);`

(a) serves the same purpose
(b) the second call is illegal
(c) both the calls are illegal
(d) none of the above

172. If `x` is an array of integer, then the value of `&x[i]` is same as that of

(a) `&x[i-1] + sizeof(int)` (b) `x + sizeof(int)*i`

(c) `x + i` (d) `++(&x[i])`

173. Pick the correct answers.

If `x` is an one dimensional array, then

(a) `&x[i]` is same as `x + i - 1`

(b) `*(x + i)` is same as `*(&x[i])`

(c) `*(x + i)` is same as `x[i]`

(d) `*(x + i)` is same as `*x + i`

174. Let `x` be an array. Which of the following cannot be present in the left hand side of an assignment statement?

(a) `x` (b) `x + i` (c) `*(x + i)` (d) `&x[i]`

175. Let `x` be an array. Which of the following operations are illegal?

(a) `++x` (b) `x + 1` (c) `x++` (d) `x * 2`

176. Consider the declaration

```
    char x[] = "WHATIZIT";
    char *y = "WHATIZIT";
```

Pick the correct answers.

(a) The output of `puts(x)` and `puts(y)` will be the same.

(b) The output of `puts(x)` and `puts(y)` will be different.

(c) The output of `puts(y)` is implementation dependent.

(d) None of the above comments are true.

177. If `func` is a function needing three arguments `a1, a2, a3`, then `func` can be invoked by

(a) `func(a1, a2, a3);` (b) `(*func)(a1, a2, a3);`

(c) `*func(a1, a2, a3);` (d) all of the above

***178.** Consider the declarations

```
    char first(int (*) (char, float));
    int second(char, float);
```

Which of the following function invocation is valid?

(a) `first(*second);` (b) `first(&second);`

(c) `first(second);` (d) none of the above

179. The declaration

```
        int(*p)[5];
```

means

(a) `p` is a one dimensional array of size 5, of pointers to integers

(b) `p` is a pointer to a 5 element integer array

(c) the same as `int *p[5];`

(d) none of the above

180. A function `q` that accepts a pointer to a character as argument and returns a pointer to an array of integer can be declared as

(a) `int(*q(char*))[]`
(b) `int *q(char *)[]`
(c) `int(*q)(char *)[]`
(d) none of the above

***181.** Consider the declaration

```
int a = 5, *b = &a;
```

The statement

```
printf("%d", a * b);
```

prints

(a) 25
(b) garbage
(c) 5 × address of `b`
(d) an error message

***182.** In the previous question, `printf("%d", a**b);` prints

(a) 25
(b) garbage
(c) 0
(d) an error message

183. The following program

```
main()
{
  float a = .5, b = .7;
  if (b < .7)
    if (a < .5)
        printf("TELO");
    else
        printf("LTTE");
  else
    printf("JKLF");
}
```

outputs

(a) LTTE
(b) TELO
(c) JKLF
(d) PLO

184. What is the output of the following program segment?

```
void max(int x, int y, int m)
{ if(x > 5) m = x;
   else m = y;}
int main()
{ int i = 20, j = 5, k = 0;
  max(i, j, k); printf("%d", k); }
```

(a) 5
(b) 20
(c) 0
(d) none of the above

185. Consider the program

```
main( )
{
 int  y = 1 ;
 printf("%d", (*(char *)&x)) ;
}
```

If the machine in which this program is executed is little-endian (meaning, the lower significant digits occupy lower addresses), then the output will be

(a) 0 (b) 99999999 (c) 1 (d) unpredictable

186. Choose the correct statements.

(a) Array stores data of the same type (b) Array can be a part of a structure

(c) Array of structure is allowed (d) Structure stores data of the same type

187. a → b is syntactically correct if

(a) a and b are structures

(b) a is a structure and b is a pointer to a structure

(c) a is a pointer to a structure and b is a structure

(d) a is a pointer to a structure in which b is a field

188. A file is preferable to an array of structures because

(a) file lives even after the program that created it terminates

(b) memory space will not be wasted

(c) there are many system tools to manipulate files

(d) there are language as well as system features to deal with files

189. The program

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

prints

(a) 2 (b) 5 (c) 1 (d) 6

190. If a file is opened in `r+` mode then

(a) reading is possible (b) writing is possible

(c) it will be created if it does not exist (d) all the above comments are true

191. `ftell`

(a) is a function

(b) gives the current file position indicator

(c) can be used to find the size of a file

(d) is meant for checking whether a given file exists or not

192. If a file is opened in `w+` mode then

(a) appending is possible

(b) reading is possible

(c) writing is possible

(d) after write operation reading is possible without closing and re-opening.

193. The fseek function

(a) needs 2 arguments
(b) makes the rewind function unnecessary
(c) is meant for checking whether a given file exists or not
(d) needs 3 arguments

194. The statement fseek(fp, 0L, 0); - if syntactically correct, means

(a) fp is a file pointer
(b) position the read-write-head at the start of the file
(c) position the read-write-head at the end of the file
(d) erase the contents of the file

195. The contents of a file will be lost if it is opened in

(a) a mode (b) w mode (c) w+ mode (d) a+ mode

196. Which of the following comments about union are true?

(a) Union is a structure whose members share the same storage area.
(b) The compiler will keep track of what type of information is currently stored.
(c) Only one of the members of union can be assigned a value at a particular time.
(d) Size allocated for union is the size of its member needing the maximum storage.

197. Which of the following comments about the usage of structure is true?

(a) Storage class can be assigned to an individual member.
(b) Individual members can be initialized within a structure type declaration.
(c) The scope of a member name is confined to the particular structure, within which it is defined.
(d) None of the above.

Answer the next 4 questions, based on the following declaration.

```
struct addr
{
char city[10];
char street[20];
int pincode;
};
struct
{
char name[20];
int sex;
struct addr locate ;
} criminal, *kd = &criminal;
```

198. sex can be accessed by

(a) criminal.sex (b) kd → sex
(c) (*kd).sex (d) either (a) or (c), but not by (b)

199. `pincode` can be accessed by

(a) `criminal.locate.pincode`

(b) `criminal.pincode`

(c) `kd → locate.pincode`

(d) `kd.locate → pincode`

200. The third character in the criminal name can be accessed by

(a) `criminal.name[2]` (b) `kd → name[2]`

(c) `((*kd).name)[2]` (d) either (b) or (c), but not by (a)

201. `*(kd → name + 2)` can be used instead of

(a) `*(criminal.name + 2)` (b) `kd → (name + 2)`

(c) `*((*kd).name + 2)` (d) either (a) or (b), but not (c)

202. How many bits are absolutely necessary to store an ASCII character?

(a) 7 (b) 8 (c) 16 (d) 15

***203.** If 7 bits are used to store a character, the percentage reduction of needed storage will be

(a) 22.5 (b) 2.5 (c) 8 (d) 12.5

204. Bit field

(a) is a field having many sub-fields

(b) is a structure declaring the sizes of the members in terms of bits

(c) is a member of a structure whose size is specified in terms of bits

(d) none of the above

205. Choose the correct comments.

In a bit-field

(a) a field can be un-named

(b) a field can be of width 0

(c) if a field is un-named, its width must not be zero

(d) a field must have a name

206. The declaration

```
    int x : 4;
```

means

(a) `x` is a four digit integer

(b) `x` cannot be greater than a four digit integer

(c) `x` is a four-bit integer

(d) none of the above

207. Bit-fields will be accommodated in a word

(a) from left to right

(b) from right to left

(c) in a way that depends on the implementation

(d) none of the above

Answer the next four questions assuming that bit-fields are accommodated from right to left and word size is 16 bits.

***208.** Consider the declaration

```
    static struct    {
                        unsigned a:5;
                        unsigned b:5;
                        unsigned c:5;
                        unsigned d:5;
                     } v = {1, 2, 3, 4};
    v occupies
```

(a) 4 words (b) 2 words (c) 1 word (d) none of the above

209. In the previous question, information about d will be in the

(a) first word (b) second word (c) in both words (d) none of the above

***210.** If the declaration unsigned c:5; is replaced by unsigned:6; then,

(a) it results in a syntax error

(b) it is meaningless

(c) the compiler will give a new name for the field, which can be used in the program

(d) none of the above

***211.** Consider the declaration

```
    struct wer {unsigned a:5;
                unsigned :0;
                unsigned b:3;
                unsigned :0;
                unsigned c:2;
                unsigned :0;} v;
```

The storage needed for v is

(a) 1 word (b) 2 words (a) 3 words (b) 4 words

***212.** The above declaration is

(a) syntactically correct (b) semantically correct

(c) a misuse of bit-fields (d) none of the above

213. Which of the following is not a low-level feature of C?

(a) Register storage class (b) Bit-fields

(c) Bit-wise operations (d) None of the above

214. C preprocessor

(a) takes care of conditional compilation (b) takes care of macros

(c) takes care of include files (d) acts before compilation

215. A preprocessor command

(a) need not start on a new line (b) need not start on the first column

(c) has # as the first character (d) comes before the first executable statement

216. Choose the correct statement.

(a) The scope of a macro definition need not be the entire program.

(b) The scope of a macro definition extends from the point of definition to the end of the file.

(c) New line is a macro definition delimiter.

(d) A macro definition may go beyond a line.

***217.** The use of macro in the place of functions

(a) reduces execution time (b) reduces code size

(c) increases execution time (d) increases code size

218. The output of the following program

```
main()
{ int a = 1, b = 2, c = 3;
  printf("%d", a += (a += 3, 5, a));
}
```

will be

(a) 8 (b) 12 (c) 9 (d) 6

219. The process of transforming one bit pattern into another by bit-wise operations is called

(a) masking (b) pruning (c) biting (d) chopping

220. Consider the following program segment.

```
char *a, *b, c[10], d[10];
a = b;
b = c;
c = d;
d = a;
```

Choose the statements having errors.

(a) No error (b) `a = b;` and `b = c;`

(c) `c = d;` and `d = a;` (d) `a = b;` and `d = a;`

***221.** The operation of a staircase switch best explains the

(a) `or` operation (b) `and` operation

(c) exclusive `nor` operation (d) exclusive `or` operation

***222.** `a << 1` is equivalent to

(a) multiplying a by 2 (b) dividing a by 2

(c) adding 2 to a (d) none of the above

223. The most significant bit will be lost in which of the following operations?

(a) `>>` (b) complementation (c) `>>` (d) none of the above

***224.** Assume an unsigned integer occupies 1 byte. Let myVar be an unsigned integer. Then myVar << 1 multiplies myVar by 2 if it is not greater than

(a) 127 (b) 255 (c) 256 (d) 128

225. If the bit pattern corresponding to a signed integer is shifted to the right then

(a) vacant bit will be filled by the sign bit

(b) vacant bit will be filled by 0

(c) the outcome is implementation dependent

(d) none of the above

***226.** In a certain machine, the sum of an integer and its 1's complement is $2^{20} - 1$. Then sizeof(int), in bits, will be

(a) 16 (b) 32 (c) unpredictable (d) none of the above

227. If the word size is 16 bit, then ~0xc5 will be

(a) 0x3a (b) 0xff3a (c) 0x5c (d) none of the above

228. Which of the following operations produce an 1, if the input bits are 1 and 1?

(a) or (b) and (c) exclusive or (d) exclusive nor

229. Preprocessing is typically done

(a) either before or at the beginning of the compilation process

(b) after compilation but before execution

(c) after loading

(d) none of the above

230. Which of the following comments about the preprocessor directive # are correct?

(a) It converts the formal argument in the macro definition into a string.

(b) It strips out redundant blanks.

(c) It concatenates adjacent strings, if any.

(d) None of the above.

231. The scope of a macro definition

(a) cannot be beyond the file in which it is defined

(b) may be part of a file

(c) is the entire program

(d) excludes string of characters within double quotes

232. The number of possible values of m, such that m & 0x3f equals 0x23 is

(a) 1 (b) 2 (c) 3 (d) 4

***233.** The for loop

```
for(i = 0; i < 10; ++i)
printf("%d", i & 1);
```

prints

(a) 0101010101 (b) 0111111111 (c) 0000000000 (d) 1111111111

234. As soon as a pointer variable is freed, its value

(a) is set to null (b) becomes unpredictable

(c) is set to 1 (d) remains the same

235. calloc(m, n); is equivalent to

(a) malloc(m*n, 0);

(b) memset(0, m*n);

(c) ptr = malloc(m*n); memset(p. 0, m*n);

(d) ptr = malloc(m*n); strcpy(p, 0);

236. Which of the following comments are correct when a macro definition includes arguments?

(a) The opening parenthesis should immediately follow the macro name.

(b) There should be at least one blank between the macro name and the opening parenthesis.

(c) There should be only one blank between the macro name and the opening parenthesis.

(d) All the above comments are correct.

237. Consider the program fragment

```
j = 2;
while ((i % j) != 0)
j = j + 1;
if (j < i) printf ("%d", j);
```

If `i >= 2`, then the value of `j`, will be printed only if

(a) `i` is prime

(b) `j` does not divide `i`

(c) `j` is odd

(d) `i` is not prime

***238.** Choose the correct statements.

(a) `'x'` is same as `"x"`.

(b) Length of the string `"x"` is two.

(c) Unless otherwise specified, the first name in an `enum` has the value `1`.

(d) None of the above.

239. Choose the correct statements.

(a) `enum` is a data type.

(b) In the same enumeration, values must be distinct.

(c) `enum` feature is an alternative to the `define` feature.

(d) None of the above.

***240.** The declaration

```
enum cities{bethlehem, jericho, nazareth = 1, jerusalem}
```

assigns the value 1 to

(a) bethlehem

(b) nazareth

(c) bethlehem and nazareth

(d) jericho and nazareth

241. Choose the correct statements.

(a) `enum` variables can be assigned new values.

(b) `enum` variables can be compared.

(c) Enumeration feature does not increase the power of C.

(d) Use of enumeration enhances the logical clarity of a program.

***242.** Consider the following statement.

```
    # define hypotenuse(a, b) sqrt(a * a + b * b);
```

The macro-call `hypotenuse(a + 2, b + 3);`

(a) finds the hypotenuse of a triangle with sides `a + 2` and `b + 3`

(b) finds the square root of $(a + 2)^2 + (b + 3)^2$

(c) is invalid

(d) finds the square root of `3*a + 4*b + 5`

243. For the previous question, which of the following macro-calls, will find the hypotenuse of a right angled triangle with sides a + 1 and b + 1?

(a) `hypotenuse (a+1,b+1)`   (b) `hypotenuse (++a, ++b)`

(c) `hypotenuse (a++,b++)`   (d) none of the above

***244.** If a variable can take only integral values from `0` to `n`, where `n` is a constant integer, then the variable can be represented as a bit-field whose width is the integral part of (the `log` in the answers are to the base `2`)

(a) `log(n) + 1`   (b) `log(n - 1) + 1`

(c) `log(n + 1) + 1`   (d) none of the above

245. The statement `printf("%d", 10?0?5:11:12);` prints

(a) `10`   (b) `0`   (c) `12`   (d) `11`

246. The statement `printf("%d", (a++));` prints

(a) the current value of `a`   (b) the value of `a + 1`

(c) an error message   (d) garbage

247. The statement `printf("%d", ++5);` prints

(a) `5`   (b) `6`   (c) an error message   (d) garbage

248. The statement `printf("%d", sizeof(""));` prints

(a) an error message   (b) `0`

(c) garbage   (d) `1`

249. If `p` is a pointer to an integer and `t` is a pointer to a character then `sizeof(p)` will be

(a) same as that of `sizeof(t)`   (b) greater than that of `sizeof(t)`

(c) less than that of `sizeof(t)`   (d) none of the above

250. Which of the following comments about arrays and pointers is/are not true?

(a) Both are exactly same   (b) Array is a constant pointer

(c) Pointer is an one-dimensional array   (d) Pointer is a dynamic array

251. `lint` is

(a) a C compiler   (b) an inter-active debugger

(c) a C interpreter   (d) a tool for analyzing a C program

252. `cb` is a

(a) C code beautifying tool   (b) C interpreter

(c) C compiler   (d) none of the above

253. It is not advisable to use macros instead of functions because
(a) it increases the code size
(b) no type checking will be done
(c) recursion is not possible
(d) pointers cannot be used with macro identifiers

254. In a C program constant is defined
(a) before main (b) after main
(c) anywhere, but starting on a new line (d) none of the above

255. The rule for implicit type conversion is
(a) `int < unsigned < float < double`
(b) `unsigned < int < float < double`
(c) `int < unsigned < double < float`
(d) `unsigned < int < double < float`

256. Which of the following is/are syntactically correct?
(a) `for();` (b) `for(;);` (c) `for(,);` (d) `for(;;);`

257. Use of macro instead of function is recommended
(a) when one wants to reduce the execution time
(b) when there is a loop with a function call inside
(c) when a function is called in many places in a program
(d) in all the above cases

258. The ascending order of precedence of the bit-wise operators `&`, `^`, `|` is
(a) `&, ^, |` (b) `^, &, |` (c) `|, ^, &` (d) `&, |, ^`

***259.** Consider the declaration
char street[10] = "abcdefghi";
Choose the correct remark(s).
(a) &street and street will have different values
(b) &street is meaningless
(c) &street+1 and street+1 will have the same values
(d) None of the above

***260.** Consider the following program fragment.

```
d = 0;
for(i = 1; i < 31; ++i)
  for(j = 1; j < 31; ++j)
     for(k = 1; k < 31; ++k)
         if(((i + j + k) % 3) == 0)
             d = d+1;
 printf("%d",d);
```

The output will be
(a) `9000` (b) `27000` (c) `3000` (d) none of the above

***261.** The number of additions performed oy the above program fragment is

(a) 27000 (b) 27000 × 3

(c) 9000 + 3 × 27000 (d) 9930 + 27000 × 3

Answers

1. c
2. c, d
3. a
4. c
5. a, b
6. d
7. c
8. d
9. a
10. a
11. b
12. b
13. d
14. b
15. b
16. b
17. a
18. c, d
19. a, b
20. b, c
21. c
22. a, b, c, d
23. c
24. c
25. a
26. a, b, c, d
27. c
28. a, b, c, d
29. a, b, c
30. a, b, c
31. d
32. b, d
33. a, b, c, d
34. a, b, d
35. a, b, c
36. c
37. a, b, c
38. a, b, d
39. c
40. d
41. d
42. a, b, c, d
43. a, c
44. c
45. b
46. c
47. c
48. d
49. a
50. a, b, c
51. c
52. c
53. b, c, d
54. d
55. b
56. b
57. a
58. a
59. d
60. a
61. b
62. c
63. d
64. a
65. d
66. a
67. c
68. a
69. d
70. c
71. a
72. b
73. b, c, d
74. a
75. a, b, c, d
76. a, b, c, d
77. a, c
78. d
79. c
80. a, b
81. a
82. d
83. a, b, c
84. b, c, d
85. a
86. d
87. b
88. a
89. d
90. a, b
91. a, b, c
92. d
93. b
94. c
95. a
96. c
97. c
98. d
99. a
100. d
101. d
102. c
103. a, b, c, d
104. b
105. d
106. a, d
107. c
108. c
109. b
110. d
111. c
112. a, b
113. c
114. a, b
115. b, d
116. a
117. a, b, c
118. a, b, c
119. b
120. c
121. a, b, c, d
122. a, b, c
123. b, c
124. a, c, d
125. c
126. a, b
127. a
128. c
129. c
130. a, b
131. a
132. b
133. a
134. d
135. c
136. a
137. b
138. d
139. d
140. d
141. a
142. c
143. b
144. b
145. a
146. b
147. a
148. a, b
149. a, b, d
150. a, c, d
151. b, c
152. a, c
153. c
154. c
155. a, b, c
156. a
157. a, b, c
158. c
159. d
160. c

161. a
162. c
163. d
164. a, b
165. c
166. b
167. a, b, c
168. a, b
169. a, b
170. c
171. a
172. c
173. b, c
174. a, b, d
175. a, c, d
176. a
177. a, b
178. c
179. b
180. a
181. d
182. a
183. a
184. c
185. c
186. a, b, c
187. d
188. a, b, c, d
189. a
190. a, b
191. a, b, c
192. b, c, d
193. b, d
194. a, b
195. b, c
196. a, c, d
197. c
198. a, b, c
199. a, c
200. a, b, c
201. a, c
202. a
203. d
204. c
205. a, b
206. c
207. c
208. b
209. b
210. d
211. c
212. a, b, c
213. d
214. a, b, c, d
215. c
216. a, b, c, d
217. a, d
218. a
219. a
220. c
221. d
222. d
223. a
224. a
225. c
226. d
227. b
228. a, b, d
229. a
230. a, b, c
231. a, b, d
232. d
233. a
234. d
235. c
236. a
237. d
238. d
239. a, c
240. d
241. a, b, c, d
242. d
243. d
244. a
245. d
246. a
247. c
248. d
249. a
250. a, b, c, d
251. d
252. a
253. a, b, c, d
254. c
255. a
256. d
257. a, b
258. c
259. d
260. a
261. d

Explanations

6. Without any temporary variable, one can swap two given variables. Refer Qn. 7.

11. In signed magnitude form, one bit is dedicated to store the `sign.` (e.g., 1 for negative and 0, otherwise). Only the remaining 15 bits are available to store the magnitude. Hence the answer.

12. Any function (including `main()`), returns a value to the calling environment. In the case of `printf,` it is the number of characters it printed. So, the output will be `tim3` (since it printed the three characters `a, b, c`).

13. Refer Qn. 12.

The `scanf` function returns the number of successful matches. i.e., `3` in this case.

14. The input is actually `a\nb.` Since we are reading only two characters, only `a` and `\n` will be read and printed.

23. If `y = 11`, the expression `3 * (y - 8) / 9` becomes `3 * 3 / 9`, which evaluates to `1`. But the expression `(y - 8) / 9 * 3` becomes `3 / 9 * 3`, which evaluates to `0` (since 3/9 is 0).

27. Strictly speaking, it will have a garbage value. Some implementations initialize to `0` on declaration.

39. y *= 2 means y = y*2 i.e., y = 4, in this problem. So, the expression is equivalent to x = 4 + 4, which is 8. So, 8 will be printed. However, the order in which the operands are evaluated is implementation-dependent. If the right operand is evaluated first, the result will be 6. Don't take things for granted.

40. Most of the compilers give 4 4 as the output. This is because most of the compilers use stacks to evaluate the arguments. If so, the first argument n++ will be pushed before the ++n is pushed. This implies that ++n will be evaluated before n++ is evaluated. However, the order of printing will be in accordance with the order the variables are listed in the printf statement.

44. 5 - 2 - 3 * 5 - 2 will yield 18, if it is treated as (5 - (2 - 3)) * (5 - 2). i.e., if – has precedence over * and if it associates from the right.

46. 263 in binary form is 100000111. If one tries to print an integer as a character, only the last 8 bits will be considered—the rest chopped off. So, in this case the ASCII value of 00000111 (i.e., decimal 7) will be printed. Look in the ASCII table. It is ringing a bell!

48. 9/5 yields integer 1. Printing 1 as a floating point number prints garbage.

50. IF is not a keyword, because it is in upper case.

51. In the computer I used to execute this program, the output was 4294967293. That's because in my system,

sizeof(int) is 4 bytes (32 bits), and negative numbers are represented in 2's complement form. This means –4 will be represented as 11111111 11111111 11111111 11111100 (i.e. 30 one's followed by 2 zeroes). Note that this number is $2^{32} - 1 - 3$. Before j gets added to i, it will be converted to an unsigned integer. So, i+j is essentially adding 1 to $2^{32} - 1 - 3$, which gives 4294967293.

52. Let sizeof(int) = 1. So, -4 will be stored as 11111100. Since we are adding unsigned and signed integers, the signed gets converted to unsigned. So, i + j will become 11111101. We are trying to print this as an unsigned integer. So, what is printed will be $2^8 - 1 - 2$. So, log(x + 3) = 8(i.e., 8*sizeof(int)).

54. 'i' is initialized to 3, and incremented by 3. When i is 15, control will go out of the loop. So 15 will be printed. (The empty semi-colon immediately following the 'for' statement, means the body of the for loop is empty.)

55. The use of continue statement forces the execution to skip the remainder of the current pass over the loop and initiates the next. If 'i' is 3, printf statement will be skipped. Hence the answer is b.

58. k = -7. So, if 'k' is used as a Boolean variable, it will be treated as a true condition. So, !k will be false i.e., 0. So, 0 <? !k is actually 0 < 0, which is false. So, 0 will be printed.

66. putchar(105) will print the ASCII equivalent of 105 i.e., 'i'. The printf statement prints the current value of i, i.e. 5 and then decrements it. So, h4 will be printed in the next pass. This continues until 'i' becomes 0, at which point the loop gets terminated.

67. Since x > 7 is false, the ternary operator ?: returns "%c". So, printf("%c", i) will be executed. So, the ASCII character corresponding to 107, i.e., 'k' will be printed.

68. Refer Qn. 67.

69. `printf("az")` prints `az` and returns a value `2` (since it printed two characters). So, the condition results in the printing of `az2`. Since it always returns `2`, it is an infinite loop. The output will be `az2byaz2by. . .`

82. Refer Qn. 69

Here the `else` clause will be executed. Since `2 < 0` is false, `four4` will be printed.

83. The `else` clause has no brackets i.e., { and }. This means the `else` clause is made up of only one statement. So, `printf("a <= b");` will be executed anyway, i.e. if `a>b` or `a<=b`. Hence the answer.

85. The condition is `putchar(0)`. This returns a value 0 which is a false condition. So, the loop will not be executed even once.

90. The expression should be evaluatable at compile time and it should evaluate to a positive integer.

92. If it is coming within a function, the storage class will be taken to be automatic, otherwise external.

98. `a[2]` will be converted to `*(a + 2)`.

`*(a + 2)` can as well be written as `*(2 + a)`.

`*(2 + a)` is nothing but `2[a]`. So, `a[2]` is essentially same as `2[a]`, which is same as `*(2 + a)`. So, it prints 9 + 9 = 18. Some of the modern compilers don't accept 2[a].

105. `*a` points to the string `"abcd"`. `**a` is the first character of `"abcd"`, which is the character `'a'`.

106. C does no array bound checking. Because of this, one can access fifth element of an array that is declared to be of lesser size.

112. C supports 1-dimensional arrays only. But, the array element can be an array by itself. Using this, one can simulate multi-dimensional arrays. Though at the user level, we use 2-dimensional arrays, the compiler interprets this as a 1-dimensional array, each of whose element is a 1-dimensional array. As a matter of fact, a declaration like `char x[3][4]`, will be interpreted as a 1-dimensional array of size 3 (rather than 4)—each element being a character array of length 4.

113. As an implication of this, the output of the following program

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

is unpredictable.

115. Consider the following program

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

Its output clearly shows, local variables can have the same name and their scope may be confined to a single statement.

116. Which means a function will be manipulating a copy of the local variable, passed as argument. So, any change will be local and hence will not be reflected in the calling routine. (Refer Qn. 128)

119. This involves recursion – `main()` calling itself. So, it keeps on printing `tim`

120. Since `Madam` is the required output, the function `first()`, should print `'a'`, call the function `second()` that prints the `'d'` and print `'a'` again. Hence c is the correct answer.

128. Refer Qn. 116. `change(a)`, prints `5` but the value of `'a'` in `main()` is still 4. So, `main()` will print 4.

131. This will not compile successfully. The scope of the variable `'j'` is the single `printf` statement that follows it. So, the last statement that involves `'j'` will complain about the undeclared identifier `'j'`.

132. By default `x` will be initialized to `0`. Since its storage class is static, it preserves its exit value (and forbids reinitialization on re-entry). So, `123` will be printed.

137. The compiler will tokenize `a+++b` as `a, ++, +, b.` So, `a+++b` is equivalent to `a+++  b,` which evaluates to `7.`

138. Refer Qn. 137. `a+++++b` will be tokenized to `a,  ++,  ++,  +,  b`. The compiler (parser), while grouping the tokens to expression, finds the second `++`, applied to `a++`, an integer. Since `++` and `++b` operator needs address (i.e., L-value), it will display the error message – `Invalid lvalue in increment`. So, to add `a++` and `++b`, use parenthesis or blanks to tokenize the way you intended.

139. `return` always terminates the function that executed it. `main()` being a function, will be terminated when it executes the `return` statement. The `return` value will be returned to the calling environment, which is the operating system in this case.

140. `arg[0]` is a pointer to the executable code file name. So, `puts(argv[0]);` prints it.

142. Like array name, name of a function is a pointer to it.

143. The function `abc` can be invoked as `abc()` or `(*abc)()`. Both are two different ways of doing the same thing.

144. The declaration means, `ptr` is a pointer, pointing to a one dimensional character array of size 4. It is assigned the address `wer`. So, `ptr` and `ptr + 1` will differ by 4 bytes.

153. Using a pointer variable, without initializing it, will be disastrous, as it will have a garbage value.

159. `**c = 5,` essentially means `a = 5,` as can be seen with the following pictorial representation of the given declarations.

Address	Value	Name
100	4	a
120	100	b
135	120	c

160. The statement is same as (int *)a. So, the value of 'a' i.e., 5 is converted into a pointer to integer data type, because of the casting assigned to a.

163. Pointers are actually addresses. Though the address will be an integer, it is not of integer data type. Both have different set of operations defined on them, e.g., integer addition is different from pointer addition.

178. The first declaration means, first is a function (returning a character), whose only argument is, a pointer to a function that takes a character and float as arguments and returns an integer. The name of a function can be used as the starting address of the function (i.e., a pointer to it). So, option c is correct.

181. Since 'a' is an integer and 'b' is a pointer, they can't be multiplied.

182. a**b will be semantically interpreted as a *(*b). Since 'a' and *b are integers, they can be multiplied.

203. For each 8 bits one can save 1 bit. So percentage reduction will be 1/8*100 i.e., 12.5%.

208. If there is no space to accommodate the entire bit-field, it will be completely shifted to the next word.

210. A bit-field need not be named.

211. A field of width 0, forces the next bit-field to the next word. So, three words are necessary.

212. Bit-fields are meant to reduce the memory needed. So, this is indeed a misuse of bit-fields. (Refer Qn. 211)

217. Use of functions involves storing the current contents and branching to its starting address. These things add to the execution time. On the other hand, macro substitution increases the code size. This is of serious concern, if the macro is used in many places.

221. First form the truth table of the exclusive OR operation. If both the switches are off i.e., 0, 0 then the light will be off i.e., 0. So, 0, 0 yields 0. If you switch on either of the two switches i.e., 0 1 or 1 0, the light will be on. So, 0 1 yields 1 (so does 1 0). Now, if you switch on the other one, which is currently off, it will be 1 1. This should yield a 0. Compare these results with the truth table of XOR.

222. The left shift operator <<, pushes out the most significant (left-most) bit. If it happens to be a 1, a << 1, will not be same as multiplying a by 2.

224. Refer Qn. 222. If the most significant bit is to be zero, the maximum number that can be stored in 7 bits is 127.

226. The sum (or bit-wise OR) of a number and its 1's complement will be all 1's. How many 1's depends on how many bits are needed to represent the number. If the sum is $2^{20} - 1$, then the sizeof(int) in bits must be 20.

233. The binary representation of odd numbers will have a 1 as the least significant digit. So, an odd number ANDed with 1, produces a 1. Even number end with 0. So, an even number ANDed with 1, produces a 0. This for loop generates even and odd numbers alternatively. So, it prints alternate 0's and 1's.

238. "x" is made up of two characters 'x' and '\0'. Anyway its length is 1.

By default, the first name in an enum will be assigned the value 0.

240. The listed places will be assigned the values 0, 1, 1, 2 respectively.

242. The macro call will be expanded as

```
sqrt(a + 2 * a + 2 + b + 3 * b + 3).
```

i.e., `sqrt(3 * a + 4 * b + 5)`. Hence the answer.

244. Let `n = 7`. It needs actually a 3 bit-field. But `log(n + 1) + 1` will be `log(8)+1`, i.e., 4, which is wrong. If `n = 8, 4` bits are needed. But, `log(n - 1) + 1` will be `log(7) + 1,` which will have an integral part of 3. `log(n) + 1` will yield the correct result in both the cases.

259. &street and street will have the values which is the starting address of the street array. However, street is a pointer to the first character whereas &street is a pointer to the entire array. The incremented values of street and &street reflects this difference.

260. `a+b+c%3` will be `0` if `a+b+c` is a multiple of `3`. This will happen in one of the following ways. All three – `a, b,` and `c` are multiples of `3`. This can only happen if `a, b,` and `c` take one of the `10` values, – `3, 6, 9, ..., 30,` independent of one another. So, there are `10x10x10 = 1000` ways this can happen. Another possibility is that `a, b,` and `c` all leave a remainder `1` so that `a+b+c` is evenly divisible by `3`. Considering all the different possibilities and adding, we get `9000`. That will be the integer that gets printed.

261. Refer Qn. 260.

The result can be analytically reasoned out. It can also be programmatically verified by having an integer variable `countAddition` (initialized to 0) and incrementing this variable each time an addition is performed. With these changes the program fragment looks like,

```
int countAddition = 0;
d = 0;
for(i=1; i<31; ++i, ++countAddition) //To account for the addition in ++i
for(j=1; j<31; ++j, ++countAddition) //To account for the addition in ++j
for(k=1; k<31; ++k, ++countAddition) //To account for the addition in ++k
    if(((i + j + k) %3) == 0)
    {
       d = d+1;
       ++countAddition;    // To account for the addition in d = d+1
       ++countAddition;    // To account for the addition in i + j
       ++countAddition;    // To account for the addition in j + k
    }
    else
    {
       ++countAddition;    // To account for the addition in i + j
       ++countAddition;    // To account for the addition in j + k
    }
printf("%d",d);
printf("\n%d", countAddition);
```

The value of the variable countAddition that is printed by the last statement is the answer.