1 Programming

1.1 The correct matching for the following pairs is

List-I

A. Activation record
B. Location counter
C. Reference counts
D. Address relocation

List-II

1. Linking loader
2. Garbage collection
3. Subroutine call
4. Assembler

Codes:

(a) A - 3, B - 4, C - 1, D - 2
(b) A - 4, B - 3, C - 1, D - 2
(c) A - 4, B - 3, C - 2, D - 1
(d) A - 3, B - 4, C - 2, D - 1

[1996 : 2 M]

1.2 Heap allocation is required for languages

(a) that support recursion
(b) that support dynamic data structures
(c) that use dynamic scope rules
(d) None of the above

[1997 : 1 M]

1.3 Faster access to non-local variables is achieved using an array of pointers to activation records called a

(a) stack
(b) heap
(c) display
(d) activation tree

[1998 : 2 M]

1.4 Consider the following program in a language that has dynamic scoping:

```
var x: real;
procedure show:
    begin print(x);end;
procedure small;
    var x: real;
    begin x: = 0.125; show; end;
    begin
        x: = 0.25;
        show; small
    end.
```

Then the output of the program is:

(a) 0.125 0.125
(b) 0.25 0.25
(c) 0.25 0.125
(d) 0.125 0.25

[1999 : 2 M]

1.5 A certain processor supports only the immediate and the direct addressing modes. Which of the following programming language features cannot be implemented on this processor?

(a) Pointers
(b) Arrays
(c) Records
(d) Recursive procedures with local variable

[1999 : 2 M]

1.6 Consider the following C function definition:

```
int Trial (int a, int b, int c)
{
    if ((a > = b) && (c < b) return b;
      else if (a > = b) return Trial (a,c,b);
      else return Trial (b,a,c);
}
```

The function Trial:

(a) Finds the maximum of a, b and c
(b) Finds the minimum of a, b and c
(c) Finds the middle number of a, b and c
(d) None of the above

[1999 : 2 M]

1.7 The following C declarations:

```
struct node{
    int i:
  float j;
};
struct node *s[10];
```

define s to be

(a) An array, each element of which is a pointer to a structure of type node

(b) A structure of 2 fields, each field being a pointer to an array of 10 elements
(c) A structure of 3 fields: an integer, a float, and an array of 10 elements
(d) An array, each element of which is a structure of type node

[2000 : 1 M]

1.8 The most appropriate matching for the following pairs

List-I

X. m=malloc(5); m = NULL;
Y. free(n); n->value=5;
Z. char *p; *p='a';

List-II

1. using dangling pointers
2. using uninitialized pointers
3. lost memory

Codes:

(a) X – 1 Y – 3 Z – 2
(b) X – 2 Y – 1 Z – 3
(c) X – 3 Y – 2 Z – 1
(d) X – 3 Y – 1 Z – 2

[2000 : 1 M]

1.9 Aliasing in the context of programming languages refers to

(a) multiple variables having the same memory location
(b) multiple variables having the same value
(c) multiple variables having the same identifier
(d) multiple uses of the same variable

[2000 : 1 M]

1.10 Consider the following C declaration:

```
struct {
    short s[5];
    union {
    float y;
    long z;
    } u;
} t;
```

Assume that objects of the type short, float and long occupy 2 bytes, 4 bytes and 8 bytes, respectively. The memory requirement for variable t, ignoring alignment considerations, is

(a) 22 bytes
(b) 14 bytes
(c) 18 bytes
(d) 10 bytes

[2000 : 1 M]

1.11 The value of j at the end of the execution of the following C program is _____.

```
int incr (int i) {
    static int count = 0;
    count = count + i;
    return (count);
}
main ( ) {
    int i, j;
    for (i = 0; i <=4; i++)
    j = incr(i);
}
```

(a) 10
(b) 4
(c) 6
(d) 7

[2000 : 2 M]

1.12 What is printed by the print statements in the program P1 assuming call by reference parameter passing?

```
Program P1()
{
    x = 10;
    y = 3;
    func1(y, x, x);
    print x;
    print y;
}
func1(x, y, z)
{
    y = y + 4;
    z = x + y + z;
}
```

(a) 10, 3
(b) 31, 3
(c) 27, 7
(d) None of these

[2001 : 2 M]

1.13 Consider the following three C functions:

[P1]
```
int *g (void)
{
    int x = 10;
    return (&x);
}
```

[P2]
```
int *g (void)
{
    int *px;
    *px = 10;
    return px;
}
```

[P3]
```
int *g(void)
{
    int *px;
    px = (int *)malloc (size of (int));
    *px = 10;
    return px;
```

Which of the above three functions are likely to cause problems with pointers?

(a) Only P3 (b) Only P1 and P3
(c) Only P1 and P2 (d) P1, P2 and P3

[2001 : 2 M]

1.14 Consider the following program:

Program P2

```
var n:int;
procedure W(var x:int)
begin
    x = x + 1;
    print x;
end
procedure D
begin
    var n:int;
    n = 3;
    W(n);
end
begin // begin P2
    n = 10;
    D;
end
```

If the language has dynamic scoping and parameters are passed by reference, what will be printed by the program?

(a) 10 (b) 11
(c) 3 (d) None of these

[2001 : 2 M]

1.15 The results returned by function under value-result and reference parameter passing conventions

(a) Do not differ
(b) Differ in the presence of loops
(c) Differ in all cases
(d) May differ in the presence of exception

[2002 : 1 M]

1.16 In the C language

(a) at most one activation record exists between the current activation record and the activation record for the main
(b) the number of activation records between the current activation record and the activation record for the main depends on the actual function calling sequence
(c) The visibility of global variables depends on the actual function calling sequence
(d) Recursion requires the activation record for the recursive function to be saved on a different stack before the recursive function can be called. **[2002 : 1 M]**

1.17 Consider the following declaration of a two-dimensional array in C

```
char a[100][100];
```

Assuming that the main memory is byte-addressable and that the array is stored starting from memory address 0, the address of a [40] [50] is

(a) 4040 (b) 4050
(c) 5040 (d) 5050 **[2002 : 2 M]**

1.18 Assume the following C variable declaration

```
int*A[10], B[10][10];
```

Of the following expressions

1. A[2]
2. A[2] [3]
3. B[1]
4. B[2] [3]

Which will not give compile-time errors if used as left hand sides of assignment statements in a C program?

(a) 1, 2 and 4 only (b) 2, 3 and 4 only
(c) 2 and 4 only (d) 4 only **[2003 : 1 M]**

1.19 In the following C program fragment j, k, n and TwoLog_n are integer variables, and A is an array of integers. The variable n is intialized to an integer ≥ 3, and TwoLog_n is initialized to the value of $2 * \lfloor \log_2(n) \rfloor$

```
for (k = 3; k < = n; k++)
    A [k] = 0;
for (k = 2; k < = TwoLog_n; k++)
    for (j = k + 1; j < = n; j++)
    A [j] = A [j] || (j%k);
for (j = 3; j < = n; j++)
    if (!A[j]) print f ("%d", j);
```

The set of numbers printed by this program fragment is

(a) {m | m ≤ n, (∃i) [m = i!]}
(b) {m | m ≤ n, (∃i) [m = i^2]}
(c) {m | m ≤ n, m is prime}
(d) {} **[2003 : 2 M]**

1.20 Consider the C program shown below:

```
#include <stdio.h>
# define print (x) printf ("%d", x)
int x;
void Q (int z) {
        z + = x; print (z);
}
void p (int *y) {
        int x = *y + 2;
        Q (x); *y = x – 1;
        print (x);
```

```
}
main (void) {
        x = 5;
        p (&x);
        print (x);
}
```

The output of this program is

(a) 12 7 6 (b) 22 12 11
(c) 14 6 6 (d) 7 6 6 **[2003 : 2 M]**

1.21 The goal of structured programming is to

(a) have well indented programs
(b) be able to infer the flow of control from the compiled code
(c) be able to infer the flow of control from the program text
(d) avoid the use of GOTO statements

[2004 : 1 M]

1.22 Consider the following C function:

```
void swap (int a, int b)
    { int temp ;
    temp = a ;
    a = b ;
    b = temp ;
    }
```

In order to exchange the values of two variables x and y.

(a) call swap (x, y)
(b) call swap (&x, &y)
(c) swap (x, y) cannot be used as it does not return any value
(d) swap (x, y) cannot be used as the parameters are passed by value **[2004 : 1 M]**

1.23 Let x be an integer which can take a value of 0 or 1. The statement

```
if (x = = 0) x = 1;
else x = 0;
```

is equivalent to which one of the following?

(a) x = 1 + x; (b) x = 1 − x;
(c) x = x − 1; (d) x = 1% x;

[2004 : 1 M]

1.24 Consider the following C program which is supposed to compute the transpose of a given 4 × 4 matrix M. Note that, there is an X in the program which indicates some missing statements. Choose the correct option to replace X in the program.

```
#include <stdio.h>
#define ROW 4
#define COL 4
int M[ROW][COL] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15, 16};
main( )
{
    int i, j, t;
    for (i = 0; i < 4; ++i)
    {
        X
    }
    for (i = 0; i < 4; ++i)
        for (j = 0; j < 4; ++j)
            printf ("%d", M[i][j]);
}
```

(a)
```
for(j = 0; j < 4; ++j)
{
    t = M[i][j];
    M[i][j] = M[j][i];
    M[j][i] = t;
}
```

(b)
```
for(j = 0; j < 4; ++j)
{
    M[i][j] = t;
    t = M[j][i];
    M[j][i] = M[i][j];
}
```

(c)
```
for(j = i; j < 4; ++j)
{
    t = M[j][i];
    M[i][j] = M[j][i];
    M[j][i] = t;
}
```

(d)
```
for(j = i; j < 4; ++j)
{
    M[i][j] = t;
    t = M[j][i];
    M[j][i] = M[i][j];
}
```

[2004 : 2 M]

1.25 What is the output of the following program?

```
#include <stdio.h>
int funcf (int x);
int funcg (int y);
main( )
{
    int x = 5, y = 10, count;
    for(count = 1; count < = 2; ++count)
    {
        y + = funcf(x) + funcg(x);
        printf ("%d", y);
    }
}
funcf(int x)
{
```

```
    int y;
    y = funcg(x);
    return (y);
}
funcg(int x)
{
    static int y = 10;
    y + = 1;
    return (y + x);
}
```

(a) 43 80 (b) 42 74
(c) 33 37 (d) 32 32 **[2004 : 2 M]**

1.26 Consider the following C program:

```
#include <stdio.h>
typedef struct
{
    char *a;
    char *b;
} t;
void f1(t s);
void f2(t *p);
main( )
{
    static t s = {"A", "B"};
    printf("%s %s\n", s.a, s.b);
    f1(s);
    printf("%s %s\n", s.a, s.b);
    f2(&s);
}
void f1(t s)
{
    s.a = "U";
    s.b = "V"
    printf("%s %s\n", s.a, s.b);
    return;
}
    void f2(t *p)
{
    p → a = "V";
    p → b = "W"
    printf("%s %s\n", p → a, p → b);
    return;
}
```

What is the output generated by the program?

(a) A B
U V
V W
V W

(b) A B
U V
A B
V W

(c) A B
U V
U V
V W

(d) A B
U V
V W
U V **[2004 : 2 M**

1.27 Consider the following C function:

```
int f (int n)
{       static int i = 1 ;
        if (n > = 5) return n ;
        n = n + i ;
        i ++;
        return f (n) ;
}
```

The value returned by f(1) is

(a) 5 (b) 6
(c) 7 (d) 8 **[2004 : 2 M]**

1.28 Consider the following C program:

```
main( )
{
      int  x, y, m, n;
      scanf ("%d %d", &x , &y);
      / * Assume x > 0 and y > 0* /
      m = x; n = y;
      while(m! = n)
      {
        if (m > n)
          m = m – n;
        else
          n = n – m;
      }
      printf ("% d", n);
}
```

The program computes

(a) x + y, using repeated subtraction
(b) x mod y using repeated subtraction
(c) the greatest common divisor of x and y
(d) the least common multiple of x and y

[2004 : 2 M]

1.29 Consider the following program fragment for reversing the digits in a given integer to obtain a new integer. Let $n = d_1 d_2 \ldots d_m$.

```
int n, rev ;
rev = 0 ;
while (n > 0) {
        rev = rev * 10 + n % 10 ;
        n = n / 10 ;
}
```

The loop invariant condition at the end of the i^{th} iteration is

(a) $n = d_1 d_2 \ldots d_{m-i}$ and $rev = d_m d_{m-1} \ldots d_{m-i+1}$
(b) $n = d_{m-i+1} \ldots d_{m-1} d_m$ or $rev = d_{m-i} \ldots d_2 d_1$
(c) $n \neq rev$
(d) $n = d_1 d_2 \ldots d_m$ or $rev = d_m \ldots d_2 d_1$

[2004 : 2 M]

1.30 Consider the following C program segment:

```
char p [20] ;
char * s = "string" ;
int length = strlen (s) ;
for (i = 0 ; i < length; i++)
    p[i] = s[length – i];
printf ("%s", p);
```

The output of the program is

(a) gnirts
(b) string
(c) gnirt
(d) no output is printed

[2004 : 2 M]

1.31 What does the following C-statement declare?

```
int (*f) (int *);
```

(a) A function that takes an integer pointer as argument and returns an integer
(b) A function that takes an integer pointer as argument and returns an integer pointer
(c) A pointer to a function that takes an integer pointer as argument and returns an integer
(d) A function that takes an integer pointer as argument returns a function pointer

[2005 : 1 M]

1.32 An Abstract Data Type (ADT) is

(a) same as an abstract class
(b) a data type that cannot be instantiated
(c) a data type for which only the operations defined on it can be used, but none else
(d) all of the above

[2005 : 1 M]

1.33 Consider the following C-program:

```
double foo (double); /* Line 1*/
int main () {
double da, db;
// input da
db = foo (da);
}
double foo (double a){
return a;
}
```

The above code complied without any error or warning. If Line 1 is deleted, the above code will show

(a) no compile warning or error
(b) some complier-warning not leading to unitended results
(c) Some complier-warning due to type-mismatch eventually leading to unitended results
(d) Complier errors

[2005 : 2 M]

1.34 Consider the following C-program:

```
void foo (int n, int sum ) {
        int k = 0, j = 0;
        if (n = = 0) return;
        k = n % 10;
        j = n / 10;
        sum = sum + k;
        foo ( j, sum);
        printf ("% d", k);
        }
int main( ) {
        int a = 2048, sum = 0;
        foo (a, sum);
        printf ("%d\n", sum);
        }
```

What does the above program print?

(a) 8, 4, 0, 2, 14
(b) 8, 4, 0, 2, 0
(c) 2, 0, 4, 8, 14
(d) 2, 0, 4, 8, 0

[2005 : 2 M]

Linked Answer Questions 1.35 & 1.36:

A sink in a directed graph is a vertex i such that there is an edge from every vertex $j \neq i$ to i and there is no edge from i to any other vertex. A directed graph G with n vertices is represented by its adjacency matrix A, where A[i][j] = 1 if there is an edge directed from vertex i to j and 0 otherwise. The following algorithm determines whether there is a sink in the graph G.

```
i = 0;
do
{
    j = i + 1;
    while ((j < n) && E1) j++;
    if (j < n) E2;
} while (j < n);
flag = 1;
for (j = 0; j < n; j++)
if ((j! = i) && E3) flag = 0;
if (flag) printf("Sink exists");
else printf ("Sink does not exist");
```

1.35 Choose the correct expressions for E_1 and E_2

(a) E_1 : A[i][j] and E_2 : i = j;
(b) E_1 : !A[i][j] and E_2 : i = j + 1;
(c) E_1: !A[i][j] and E_2 : i = j;
(d) E_1 : A[i][j] and E_2 : i = j + 1;

[2005 : 2 M]

1.36 Choose the correct expression for E_3

(a) (A[i][j] && !A[j][i])
(b) (!A[i][j] && !A[j][i])
(c) (!A[i][j] || A[j][i])
(d) (A[i][j] || !A[j][i])

[2005 : 2 M]

1.37 The following C function takes two ASCII strings and determines whether one is an anagram of the other. An anagram of a string s is a string obtained by permuting the letters in s.

```
int anagram (char *a, char *b)
{
    int count [128], j;
    for (j = 0; j < 128; j++)
        count[j] = 0;
    j = 0;
    while (a[j] && b[j])
    {
        A ;
        B ;
    }
    for (j = 0; j < 128; j++)
        if (count [j]) return 0;
    return 1;
}
```

Choose the correct alternative for statements A and B.

(a) A: count [a[j]]++ and
B: count[b[j]]– –

(b) A: count [a[j]]++ and
B: count[b[j]]++

(c) A: count [a[j++]]++ and
B: count[b[j]]– –

(d) A: count [a[j]]++ and
B: count[b[j++]]– – **[2005 : 2 M]**

1.38 What is the output printed by the following program?

```
# include <stdio.h>
int f(int n, int k)
{
    if (n = = 0) return 0;
    else if (n% 2) return f(n/2, 2*k)+k;
    else return f(n/2, 2*k) – k;
}
int main( )
{
    printf("%d", f(20, 1));
    return 0;
}
```

(a) 5 (b) 8
(c) 9 (d) 20 **[2005 : 2 M]**

1.39 Which one of the choices given below would be printed when the following program is executed?

```
#include <stdio.h>
struct test
{
    int i;
    char *c;
} st[ ] = {5, "become", 4, "better", 6, "jungle", 8,
"ancestor", 7, "brother"};
main( )
{
    struct test *p = st;
    p+ = 1;
        ++p → c;
    printf("%s", p++ → c);
    printf("%c", *++p → c);
    printf("%d", p[0].i);
    printf("%s \n", p → c);
}
```

(a) jungle, n, 8, ncestor
(b) etter, u, 6, ungle
(c) cetter, k, 6, jungle
(d) etter, u, 8, ncestor **[2006 : 2 M]**

1.40 Which one of the choices given below would be printed when the following program is executed?

```
#include <stdio.h>
void swap (int *x, int *y)
{
    static int *temp;
    temp = x;
    x = y;
    y = temp;
}
void printab( )
{
    static int i, a = –3, b = –6;
    i = 0;
    while (i < = 4)
{
    if ((i++)%2 = = 1) continue;
    a = a + i;
    b = b + i;
}
    swap (&a, &b);
    printf("a = %d, b = %d\n", a, b);
}
main( )
{
    printab( );
    printab( );
}
```

(a) a = 0, b = 3
a = 0, b = 3

(b) a = 3, b = 0
a = 12, b = 9

(c) a = 3, b = 6
a = 3, b = 6

(d) a = 6, b = 3
a = 15, b = 12 **[2006 : 2 M]**

1.41 Which one of the choices given below would be printed when the following program is executed?

```
#include <stdio.h>
int a1[ ] = {6, 7, 8, 18, 34, 67};
int a2[ ]= {23, 56, 28, 29};
int a3[ ] = {–12, 27, –31};
int *x[ ] = {a1, a2, a3};
void print(int *a[ ])
{
    printf("%d,", a[0][2]);
    printf("%d,", *a[2]);
    printf("%d,", *++a[0]);
    printf("%d,", *(++a)[0]);
    printf("%d\n", a[–1][+1]);
}
main()
{
    print(x);
}
```

(a) 8, –12, 7, 23, 8
(b) 8, 8, 7, 23, 7
(c) –12, –12, 27, –31, 23
(d) –12, –12, 27, –31, 56

[2006 : 2 M]

1.42 The following function computes the value of $\binom{m}{n}$ correctly for all legal values m and n ($m \geq 1$, $n \geq 0$ and $m > n$)

```
int func(int m, int n)
{
    if (E) return 1;
    else return(func(m–1, n) + func(m – 1, n – 1));
}
```

In the above function, which of the following is the correct expression for E?

(a) (n = = 0) | | (m = = 1)
(b) (n = = 0) && (m = = 1)
(c) (n = = 0) | | (m = = n)
(d) (n = = 0) && (m = = n)

[2006 : 2 M]

1.43 Consider the following C-function in which a [n] and b [m] are two sorted integer arrays and c [n + m] be another integer array.

```
void xyz (int a[ ], int b[ ], int c[ ]) {
    int i, j, k;
    i = j = k = 0;
    while ((i < n) && (j < m))
        if (a[i] < b[j]) c[k++] = a[i ++];
        else c[k ++] = b[j ++];
    }
```

Which of the following condition (s) hold (s) after the termination of the while loop?

(i) j < m, k = n + j – 1, & a [n –1] < b [j] if i = n
(ii) i < n, k = m + i – 1, & b [m – 1] ≤ a [i] if j = m

(a) only (i)
(b) only (ii)
(c) either (i) or (ii) but not both
(d) neither (i) nor (ii)

[2006 : 2 M]

1.44 Consider these two functions and two statements S1 and S2 about them.

```
int work1(int* a, int i, int j)
{
    int x=a[i+2];
    a[j]=x+1;
    return a[i+2]–3;
}
```

```
int work2(int* a, int i, int j)
{
    int t1=i+2;
    int t2=a[t1];
    a[j] = t2 + 1;
    return t2–3;
}
```

S1 : The transformation from work1 to work2 is valid, i.e., for any program state and input arguments, work2 will compute the same output and have the same effect on program state as work1

S2: All the transformations applied to work1 to get work2 will always improve the performance (i.e. reduce CPU time) of work2 compared to work1

(a) S1 is false and S2 is false
(b) S1 is false and S2 is true
(c) S1 is true and S2 is false
(d) S1 is true and S2 is true

[2006 : 2 M]

1.45 Consider this C code to swap two integers and these five statements : the code

```
void swap (int*px, int* py) {
        *px = *px – *py;
        *py = *px + *py;
        *px =*py –*px;
}
```

S1: will generate a compilation error
S2: may generate a segmentation fault at runtime depending on the arguments passed

S3: Correctly implements the swap procedure for all input pointers referring to integers stored in memory locations accessible to the process
S4: implements the swap procedure correctly for some but not all valid input pointers
S5: may add or subtract integers and pointers
(a) S1
(b) S2 and S3
(c) S2 and S4
(d) S2 and S5

[2006 : 2 M]

1.46 Consider the following C function:

```
int f(int n)
{   static int r = 0;
    if(n <= 0) return 1;
    if(n > 3)
    {
      r = n;
      return f(n – 2) + 2;
    }
    return f(n –1) + r;
}
```

What is the value of f(5)?

(a) 5 (b) 7
(c) 9 (d) 18 **[2007 : 2 M]**

1.47 The function f is defined as follows:

```
int f(int n)
{
    if (n <= 1)
    return 1;
    else if (n % 2 == 0) return f(n/2);
    else return f(3n – 1);
```

Assuming that arbitrarily large integers can be passed as a parameter to the function, consider the following statements.

(i) The function f terminates for finitely many different values of $n \geq 1$.
(ii) The function f terminates for infinitely many different values of $n \geq 1$.
(iii) The function f does not terminate for finitely many different values of $n \geq 1$.
(iv) The function f does not terminate for infinitely many different values of $n \geq 1$.

Which one of the following options is true of the above?

(a) (i) and (iii)
(b) (i) and (iv)
(c) (ii) and (iii)
(d) (ii) and (iv)

[2007 : 2 M]

1.48 Consider the C program given below:

```
# include <stdio.h>
int main( )
{
    int sum = 0, maxsum = 0, i,  n = 6;
    int a( ) = {2, –2, –1, 3, 4, 2};
    for(i = 0; i < n; i++)
    {
        if(i = = 0 | | a[i] < 0 | | a[i] < a[i – 1])
        {
            if(sum > maxsum)
                maxsum = sum;
            sum = (a[i] > 0) ? a[i] : 0;
        }
        else sum + = a[i];
        if(sum > maxsum) maxsum = sum;
        printf ("%d\n", maxsum);
}
```

What is the value printed out when this program is executed?

(a) 9 (b) 8
(c) 7 (d) 6 **[2007 : 2 M]**

1.49 Consider the following C program:

```
# include <stdio.h>
# define EOF –1
void push (int); /* push the argument on the stack
*/
int pop  (void); /* pop the top of the stack */
void flagError ( );
int main( )
{
    int c, m, n, r;
    while ((c = getchar ( )) ! = EOF)
    {   if (isdigit (c)) push (c);
        else if ((c = = '+') | | (c = = '*'))
        {
            m = pop ( );
            n = pop ( );
            r = (c = = '+') ? n + m : n*m;
            push (r);
        }
        else if (c ! = ' ')
        flagError ( );
    }
    printf("% c", pop ( ));
}
```

What is the output of the program for the following input?

5 2 * 3 3 2 + * +

(a) 15 (b) 25
(c) 30 (d) 150 **[2007 : 2 M]**

1.50 Consider the program below in a hypothetical language which allows global variables and a choice of call by reference or call by value methods of parameter passing.

```
int i ;
program main( )
{   int j = 60;
    i = 50;
    call f(i, j);
    print i, j;
}
procedure f(x, y)
{
    i = 100;
    x = 10;
    y = y + i ;
}
```

Which one of the following options represents the correct output of the program for the two parameter passing mechanisms?

(a) Call by value : i = 70, j = 10; Call by reference : i = 60, j = 70
(b) Call by value : i = 50, j = 60; Call by reference : i = 50, j = 70
(c) Call by value : i = 10, j = 70; Call by reference : i = 100, j = 60
(d) Call by value : i = 100, j = 60; Call by reference : i = 10, j = 70

[2007 : 2 M]

1.51 Consider the program below in a hypothetical programming language which allows global variables and a choice of static or dynamic scoping.

```
int i;
program main( )
{
    i = 10;
    call f( );
}
procedure f( )
{
    int i = 20;
    call g( );
}
procedure g( )
{
    print i;
}
```

Let x be the value printed under static scoping and y be the value printed under dynamic scoping. Then, x and y are

(a) x = 10, y = 10 (b) x = 20, y = 10
(c) x = 10, y = 20 (d) x = 20, y = 20

[2007 : 2 M]

1.52 Which combination of the integer variables x, y and z makes the variable a get the value 4 in the following expression?

```
a = ( x > y ) ? (( x > z ) ? x : z) : ((y>z) ? y : z)
```

(a) x = 3, y = 4, z = 2
(b) x = 6, y = 5, z = 3
(c) x = 6, y = 3, z = 5
(d) x = 5, y = 4, z = 5

[2008 : 1 M]

1.53 What is the output printed by the following C code?

```
# include <stdio.h>
int main( )
{
    char a[6] = "world";
    int i, j;
    for (i = 0, j = 5; i < j; a[i++] = a [j – –]);
    printf ("%s\n", a);
}
```

(a) dlrow (b) Null string
(c) dlrld (d) worow

[2008 : 2 M]

1.54 Consider the C program below. What does it print?

```
# include <stdio.h>
# define swap1 (a,b) tmp = a; a = b; b =tmp;
void swap2 ( int a, int b)
{
    int tmp;
    tmp = a; a = b; b = tmp;
}
void swap3 (int*a, int*b)
{
    int tmp;
    tmp = *a; *a = *b; *b = tmp;
}
int main( )
{
    int num1 = 5, num2 = 4, tmp;
    if(num1 > num2)
    { swap1(num1, num2); }
    if (num1 < num2)
    { swap2(num1 + 1, num2); }
    if(num1 > = num2)
    { swap3 (&num1, &num2); }
    printf ("%d, %d", num1, num2);
}
```

(a) 5, 5 (b) 5, 4
(c) 4, 5 (d) 4, 4

[2008 : 2 M]

1.55 Consider the C program given below. What does it print?

```
# include <stdio.h>
int main( )
{
    int i, j;
    int a[8] = {1, 2, 3, 4, 5, 6, 7, 8};
    for(i = 0; i < 3; i++)
    {
        a[i] = a[i]+1;
        i++;
    }
    i – –;
    for (j = 7; j > 4, j– –)
    {
        int i = j/2;
        a[i] = a[i] –1;
    }
    printf ("%d, %d", i, a [i]);
}
```

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

[2008 : 2 M]

1.56 A C program is given below:

```
# include <stdio.h>
int main( )
{
    int i, j;
    char a[2][3] = {{'a', 'b', 'c'}, {'d', 'e', 'f'}};
    char b[3][2];
    char *p = *b;
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
        {
            *(p + 2*j + i) = a [i] [j];
        }
    }
}
```

What should be the contents of the array b at the end of the program?

(a) a b
c d
e f
(b) a d
b e
c f
(c) a c
e b
d f
(d) a e
d c
b f

[2008 : 2 M]

Directions for Question 1.57 to 1.58:

Consider the code fragment written in C below:

```
void f(int n)
{
    if(n ≤ 1 )
    {
        printf("%d", n);
    }
    else
    {
        f(n/2);
        printf("%d", n%2);
    }
}
```

1.57 What does f(173) print?

(a) 010110101 (b) 010101101
(c) 10110101 (d) 10101101

[2008 : 2 M]

1.58 Which of the following implementations will produce the same output for f(173) as the one from above Question?

P1:

```
void f(int n)
{
  if(n/2)
  {
    f(n/2);
  }
  printf("%d", n%2);
}
```

P2:

```
void f(int n)
{
  if (n < = 1)
  {
    printf("%d", n); }
  else
  { printf("%d", n%2);
    f(n/2); }
}
```

(a) Both P1 and P2 (b) P2 only
(c) P1 only (d) Neither P1 nor P2

[2008 : 2 M]

1.59 What is printed by the following C program?

```
int f(int x, int *py, int ** ppz)
{ int y, z;
  **ppz += 1; z = **ppz;
  *py += 2; y = *py;
  x += 3;
  return x + y + z;
}
void main( )
{ int c, *b, **a;
  c = 4; b = &c; a = & b;
  printf ("% d", f(c, b, a));
}
```

(a) 18 (b) 19
(c) 21 (d) 22

[2008 : 2 M]

1.60 Choose the correct option to fill ?1 and ?2 so that the program below prints an input string in reverse order. Assume that the input string is terminated by a newline character.

```
void reverse (void) {
    int c;
    if (?1) reverse ();
    ?2
}
main () {
    printf ("Enter Text") ; printf ("\n");
    reverse (); printf ("\ n");
}
```

(a) ? 1 is (getchar ( ) ! = '\n')
? 2 is getchar (c);
(b) ? 1 is (c = getchar ( )) ! = '\n')
? 2 is getchar (c)
(c) ? 1 is (c ! = '\n')
? 2 is putchar (c);
(d) ? 1 is ((c = getchar ( )) ! = '\n')
? 2 is putchar (c); **[2008 : 2 M]**

1.61 Consider the program below:

```
# include <stdio.h>
int fun (int n, int *f_p) {
  int t, f;
  if (n < = 1) {
    *f_p = 1 ;
    return 1 ;
  }
  t = fun(n–1, f_p) ;
  f = t + * f_p ;
  *f_p = t ;
  return f ;
}
int main( )
{
  int x = 15 ;
  printf ("% d\n", fun (5, &x)) ;
  return 0;
}
```

The value printed is
(a) 6 (b) 8
(c) 14 (d) 15 **[2009 : 1 M]**

1.62 What does the following program print?

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

(a) 2 2 (b) 2 1
(c) 0 1 (d) 0 2 **[2010 : 1 M]**

1.63 What is the value printed by the following C program?

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

(a) –9 (b) 5
(c) 15 (d) 19 **[2010 : 2 M]**

1.64 The following program is to be tested for statement coverage:

```
begin
  if (a = = b) {S1; exit;}
  else if (c = = d) {S2;}
  else {S3; exit;}
  S4;
end
```

The test cases T1, T2, T3 and T4 given below are expressed in terms of the properties satisfied by the values of variables a, b, c and d. The exact values are not given.

T1 : a, b, c and d are all equal
T2 : a, b, c and d are all distinct
T3 : a = b and c != d
T4 : a !=b and c = d

Which of the test suites given below ensures coverage of statements S1, S2, S3 and S4?
(a) T1, T2, T3 (b) T2, T4
(c) T3, T4 (d) T1, T2, T4 **[2010 : 2 M]**

1.65 What does the following fragment of C program print?

```
char c[ ] = "GATE2011";
char *p = c;
printf("%s", p + p[3] – p[1]);
```

(a) GATE 2011 (b) E2011
(c) 2011 (d) 011 **[2011 : 1 M]**

Common Data Question 1.66 and 1.67:

Consider the following recursive C function that takes two arguments.

```
unsigned int foo(unsigned int n, unsigned int r)
{
  if(n>0) return ((n%r)+foo(n/r,r));
  else return 0;
}
```

1.66 What is the return value of the function foo when it is called as foo(345, 10)?
(a) 345 (b) 12
(c) 5 (d) 3 **[2011 : 2 M]**

1.67 What is the return value of the function foo when it is called as foo(513, 2)?
(a) 9 (b) 8
(c) 5 (d) 2 **[2011 : 2 M]**

1.68 What will be the output of the following C program segment?

```
char inChar = 'A';
switch(inChar)
{
  case  'A'  :  printf ("Choice A\ n") ;
  case  'B'  :
  case  'C'  :  printf ("Choice B") ;
  case  'D'  :
  case  'E'  :
  default    :  printf ("NO Choice") ; }
```

(a) No Choice
(b) Choice A
(c) Choice A
Choice B No Choice
(d) Program gives no output as it is erroneous **[2012 : 1 M]**

Common Data for Questions 1.69 and 1.70:

Consider the following C code segment.

```
int a, b, c = 0;
void prtFun (void);
main()
{static int a = 1;/* Line 1*/
prtFun();
a+=1;
prtFun();
printf("\n %d %d ",a,b);}
void prtFun(void)
{static int a = 2;/*Line 2*/
int b = 1;
a += ++b;
printf("\n %d %d ",a,b);}
```

1.69 What output will be generated by the given code segment?

(a) 3 1 (b) 4 2
4 1 6 1
4 2 6 1
(c) 4 2 (d) 3 1
6 2 5 2
2 0 5 2 **[2012 : 2 M]**

1.70 What output will be generated by the given code segment if:
Line 1 is replaced by auto int a=1;
Line 2 is replaced by register int a = 2;

(a) 3 1 (b) 4 2
4 1 6 1
4 2 6 1
(c) 4 2 (d) 4 2
6 2 4 2
2 0 2 0 **[2012 : 2 M]**

1.71 What is the return value of f(p, p), if the value of p is initialized to 5 before the call? Note that the first parameter is passed by reference, whereas the second parameter is passed by value.

```
int f (int &x, int c)
{  c = c – 1;
   if (c = = 0) return 1;
   x = x + 1;
   return f(x, c) * x; }
```

(a) 3024 (b) 6561
(c) 55440 (d) 161051 **[2013 : 2 M]**

1.72 Consider the following program in C language:

```
#include <stdio.h>
main()
{
  int i;
  int *pi = &i;
  scanf("%d",pi);
  printf("%d\n", i+5);
}
```

Which one of the following statements is TRUE?
(a) Compilation fails.
(b) Execution results in a run-time error.
(c) On execution, the value printed is 5 more than the address of variable *i*.
(d) On execution, the value printed is 5 more than the integer value entered.

[2014 (Set-1) : 1 M]

1.73 Consider the following pseudo code. What is the total number of multiplications to be performed?

```
D = 2
for i = 1 to n do
    for j = i to n do
        for k = j + 1 to n do
            D = D * 3
```

$\sum_{i=0}^{n}\sum_{j=0}^{i}\sum_{k=j+1}^{n} 1 = \frac{n(n+1)(n+2)}{6}$

(a) Half of the product of the 3 consecutive integers.
(b) One-third of the product of the 3 consecutive integers.
(c) One-sixth of the product of the 3 consecutive integers.
(d) None of the above

[2014 (Set-1) : 2 M]

1.74 Consider the function func shown below:

```
int func(int num)
{   int count = 0;
    while (num)
    {  count++; num>>= 1;
    }
    return (count);
}
```

The value returned by function (435) is 9.

[2014 (Set-2) : 1 M]

1.75 Suppose n and p are unsigned int variables in a C program. We wish to set p to ${}^{n}C_{3}$. If n is large, which one of the following statements is most likely to set p correctly?
(a) $p = n*(n-1)*(n-2)/6;$
(b) $p = n*(n-1)/2*(n-2)/3;$
(c) $p = n*(n-1)/3*(n-2)/2;$
(d) $p = n*(n-1)*(n-2)/6.0;$

[2014 (Set-2) : 1 M]

1.76 Consider the following function:

```
double f(double x) {
  if(abs(x*x – 3) < 0.01) return x;
  else return f(x/2 + 1.5/x);
}
```

$|x^2-3| < 0.01$, $x = \sqrt{3} = 1.73$

Give a value *q* (to 2 decimals) such that $f(q)$ will return *q* : 1.73.

[2014 (Set-2) : 2 M]

1.77 Consider the C function given below:

```
int f(int j) {
    static int i = 50;
    int k;
    if (i == j) {
        printf("something");
        k = f(i);
        return 0;
    }
    else return 0;
}
```

Which one of the following is TRUE?
(a) The function returns **0** for all values of *j*.
(b) The function prints the string something for all values of *j*.
(c) The function returns **0** when $j = 50$.
(d) The function will exhaust the runtime stack or run into an infinite loop when $j = 50$.

[2014 (Set-2) : 2 M]

1.78 The output of the following C program is -5

```
void f1 (int a, int b) {
    int c;
    c=a; a=b; b=c;
}
void f2 (int *a, int *b) {
    int c;
    c=*a; *a=*b; *b=c;
}
int main() {
    int a=4, b=5, c=6;
    f1 (a, b);
    f2 (&b, &c);
    printf ("%d", c – a – b);
}
```

[2015 (Set-1) : 1 M]

1.79 Consider the following pseudo code, where x and y are positive integers.

```
begin
    q := 0;
    r := x;
    while r ≥ y do
      begin r := r – y ; q := q + 1 ; end
end
```

The post condition that needs to be satisfied after the program terminates is
(a) $\{r = qx + y \wedge r < y\}$
(b) $\{x = qy + r \wedge r < y\}$
(c) $\{y = qx + r \wedge 0 < r < y\}$
(d) $\{q + 1 < r - y \wedge y > 0\}$

[2015 (Set-1) : 2 M]

1.80 Consider the following C function:

```
int fun (int n) {
    int x=1, k;
    if (n==1) return x;
    for (k=1; k<n; ++k)
        x=x + fun (k) * fun (n – k);
    return x;
}
```

The return value of fun (5) is 51.

[2015 (Set-2) : 1 M]

1.81 Consider the following function written in the C programming language.

```
void foo (char *a) {
    if (*a && *a ! = ' ') {
        foo (a+1);
        putchar (*a);
    }
}
```

The output of the above function on input "ABCD EFGH" is
(a) ABCD EFGH (b) ABCD
(c) HGFE DCBA (d) DCBA

[2015 (Set-2) : 1 M]

1.82 A Young tableau is a 2D array of integers increasing from left to right and from top to bottom. Any unfilled entries are marked with ∞, and hence there cannot be any entry to the right of, or below a ∞. The following Young tableau consists of unique entries.

1	2	5	14
3	4	6	23
10	12	18	25
31	∞	∞	∞

When an element is removed from a Young tableau, other elements should be moved into its place so that the resulting table is still a Young tableau (unfilled entries may be filled in with a ∞). The minimum number of entries (other than 1) to be shifted, to remove 1 from the given Young tableau is 5.

[2015 (Set-2) : 2 M]

1.83 Consider the following C program segment:

```
# include <stdio.h>
int main( ) {
    char s1[7] = "1234", *p;
    p = s1 + 2;
    *p = '0' ;
    printf ("%s", s1);
}
```

What will be printed by the program?
(a) 12 (b) 120400
(c) 1204 (d) 1034

[2015 (Set-3) : 1 M]

1.84 Consider the following C program:

```
# include < stdio.h>
int main( ) {
  static int a[ ] = {10, 20, 30, 40, 50};
  static int *p[ ] = {a, a+3, a+4, a+1, a+2};
  int **ptr = p;  ptr++;
  printf ("%d%d", prt-p, **ptr);
}
```

The output of the program is 140.

[2015 (Set-3) : 2 M]

1.85 Consider the following recursive C function:

```
void get (int n) {
    if (n < 1) return;
    get (n – 1);
    get (n – 3);
    printf ("%d", n);
}
```

If get(6) function is being called in main( ) then how many times will the get( ) function be invoked before returning to the main( )?
(a) 15 (b) 25
(c) 35 (d) 45

[2015 (Set-3) : 2 M]

1.86 Consider the following C program:

```
# include <stdio.h>
int main( ) {
   int i, j, k = 0;
   j = 2 * 3 / 4 + 2.0 / 5 + 8 / 5;
   k – = – –j;
   for(i = 0; i < 5; i++) {
      switch(i + k)
      {
          case 1:
          case 2: printf("\n%d", i + k);
          case 3: printf("\n%d", i + k);
          default: printf("\n%d", i + k);
      }
   }
   return 0;
}
```

The number of times printf statement is executed is 10.

[2015 (Set-3) : 2 M]

1.87 Consider the following C program:

```
# include <stdio.h>
int f1(void);
int f2(void);
int f3(void);
```

```
int x = 10;
int main( )
{
    int x = 1;
    x +=f1( ) + f2( ) + f3( ) + f2( );
    pirntf("%d", x);
    return 0;
}
int f1( ) { int x = 25; x++; return x; }
int f2( ) { static int x = 50; x++; return x; }
int f3( ) { x *= 10; return x; }
```

The output of the program is 230.

[2015 (Set-3) : 2 M]

1.88 Consider the following C program:

```
void f (int, short);
void main()
{
    int i = 100;
    short s = 12;
    short *p = &s;
    _____;      // call to f ( )
}
```

Which one of the following expressions, when placed in the blank above, will NOT result in a type checking error?

(a) f(s, *s) (b) i = f(i, s)
(c) f(i, *s) (d) f(i, *p)

[2016 (Set-1) : 1 M]

1.89 Consider the following C program:

```
#include<stdio.h>
void mystery (int *ptra, int *ptrb)
{   int *temp;
    temp = ptrb;
    ptrb = ptra;
    ptra = temp;
}
int main()
{   int a = 2016, b = 0, c = 4, d = 42;
    mystery(&a, &b);
    if (a < c)
        mystery(&c, &a);
    mystery(&a, &d);
    printf("%d \ n", a);
}
```

The output of the program is 2016

[2016 (Set-1) : 1 M]

1.90 The following function computes the maximum value contained in an integer array p[ ] of size n(n >= 1).

```
int max(int *p, int n)
{   int a = 0, b = n – 1;
    while ( _____ )
    {
        if (p[a] <= p[b])
        {
            a = a + 1;
        }
        else
        {
            b = b – 1;
        }
    }
    return p[a];
}
```

The missing loop condition is

(a) a ! = n (b) b ! = 0
(c) b > (a + 1) (d) b ! = a

[2016 (Set-1) : 2 M]

1.91 What will be the output of the following C program?

```
void count(int n)
{   static int d = 1;
    printf("%d", n);
    printf("%d", d);
    d++;
    if (n > 1) count (n – 1);
    printf("%d", d);
}
void main()
{
    count (3);
}
```

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

[2016 (Set-1) : 2 M]

1.92 What will be the output of the following pseudo-code when parameters are passed by reference and dynamic scoping is assumed?

```
a = 3;
void n(x)
{
    x = x * a; print(x);
}
void m(y)
{   a = 1; a = y – a; n(a);
    print(a);
}
void main()
{
    m(a);
}
```


(a) 6, 2 (b) 6, 6
(c) 4, 2 (d) 4, 4

[2016 (Set-1) : 2 M]

1.93 The value printed by the following program is 30.

```
void f(int* p, int m)
{   m = m + 5;
    *p = *p + m;
    return;
}
void main ( )
{   int i = 5, j = 10;
    f(&i, j);
    printf ("%d", i + j) ;
}
```

[2016 (Set-2) : 1 M]

1.94 The following function computes X^Y for positive integers X and Y.

```
int exp (int X, int Y )
{   int res = 1, a = X, b = Y;
    while (b ! = 0)
    {   if (b% 2 == 0)
        {   a = a*a; b = b/2;   }
            else {res = res*a; b = b – 1;
        }
    }
    return res;
}
```

Which one of the following conditions is TRUE befor every iteration of the loop?

(a) $X^Y = a^b$
(b) $(res * a)^Y = (res * X)^b$
(c) $X^Y = res * a^b$
(d) $X^Y = (res * a)^b$

[2016 (Set-2) : 2 M]

1.95 Consider the following program:

```
int f(int *p, int n)
{   if (n <= 1) return 0;
    else return max (f(p+1, n–1), p[0] –p[1]);
}
int main ( )
{   int a[ ] = {3, 5, 2, 6, 4};
    print f("%d", f(a, 5));
}
```

Note: max (x, y) returns the maximum of x and y.

The value printed by this program is 3.

[2016 (Set-2) : 2 M]

1.96 The output of executing the following C program is 23.

```
#include <stdio.h>
int total (int v)
{   static int count = 0;
    while (v)
    {   count+ = v & 1;
        v >> = 1;
    }
    return count;
}
void main ( )
{   static int x = 0;
    int i = 5;
    for (; i > 0; i --) {
        x = x + total (i);
    }
    printf ("%d\n", x);
}
```

[2017 (Set-1) : 2 M]

1.97 Consider the C functions foo and bar given below:

```
int foo (int val)
{
    int x = 0;
    while (val > 0)
    {
        x = x + foo (val —);
    }
    return val;
}
int bar (int val)
{
    int x = 0;
    {
        while (val > 0)
        x = x + bar (val – 1);
    }
return val;
}
```

Invocations of foo (3) and bar (3) will result in :

(a) Return of 6 and 6 respectively.
(b) Infinite loop and abnormal termination respectively.
(c) Abnormal termination and infinite loop respectively.
(d) Both terminating abnormally.

[2017 (Set-1) : 2 M]

1.98 Consider the following C program:

```
#include <stdio.h>
#include <string.h>
void printlength (char *s, char *t)
{
    unsigned int c = 0;
    int len = ((strlen(s) – strlen(t)) > c) ? strlen(s):
    strlen(t);
    printf("%d\n", len);
}
void main( )
{
    char *x = "abc";
    char *y = "defgh";
    printlength (x, y);
```

Recall that strlen is defined in string.h as returning a value of type size_t, which is an unsigned int. The output of the program is ___3___.

[2017 (Set-1) : 2 M]

1.99 Consider the following two functions:

```
void fun1 (int n)
{   if (n = = 0) return;
    printf ("%d", n);
    fun2 (n – 2);
    printf ("%d", n);
}
void fun2 (int n)
{   if (n = = 0) return;
    printf ("%d", n);
    fun1(++n);
    printf ("%d", n);
}
```

The output printed when fun1 (5) is called is

(a) 53423122233445
(b) 53423120112233
(c) 53423122132435
(d) 53423120213243

[2017 (Set-1) : 2 M]

1.100 Match the following:

List-I

P. static char var;
Q. m = malloc(10); m = NULL;
R. char *ptr[10];
S. register int var1;

List-II

(i) Sequence of memory locations to store addresses.
(ii) A variable located in data section of memory.
(iii) Request to allocate a CPU register to store data.
(iv) A lost memory which cannot be freed.

(a) P → (ii), Q → (iv), R → (i), S → (iii)
(b) P → (ii), Q → (i), R → (iv), S → (iii)
(c) P → (ii), Q → (iv), R → (iii), S → (i)
(d) P → (iii), Q → (iv), R → (i), S → (ii)

[2017 (Set-2) : 1 M]

1.101 Consider the following function implemented in C:

```
void printxy (int x, int y)
{
    int *ptr;
    x = 0;
    ptr = &x;
    y = *ptr;
    *ptr = 1;
    printf ("%d, %d", x, y);
}
```

The output of invoking printxy (1, 1) is

(a) 0, 0 (b) 0, 1
(c) 1, 0 (d) 1, 1

[2017 (Set-2) : 1 M]

1.102 Consider the following C program:

```
# include <stdio.h>
# include <string.h>
int main( )
{   char* c = "GATECSIT2017";
    char* p = c;
    printf("%d", (int) strlen (c+2[p] – 6[p]–1));
    return 0;
}
```

The output of the program is ___2___.

[2017 (Set-2) : 2 M]

1.103 Consider the following C program:

```
# include <stdio.h>
int main( )
{
    int m = 10;
    int n, n1;
    n = ++m;
    n1 = m++;
    n --;
    -- n1;
    n – = n1;
    printf("%d", n);
    return 0;
}
```

The output of the program is ___0___.

[2017 (Set-2) : 2 M]

1.104 Consider the following snippet of a C program. Assume that swap (&x, &y) exchanges the contents of x and y.

```
int main( )
{
    int array[ ] = {3, 5, 1, 4, 6, 2};
    int done = 0;
    int i;
    while (done == 0)
    {
        done = 1;
        for (i = 0; i <= 4; i++)
        {
            if (array[i] < array[i + 1])
            {
                swap(&array[i], &array[i + 1]);
                done = 0;
            }
        }
    }
    for (i = 5; i >= 1; i --)
    {
        if (array[i] > array[i – 1])
        {
            swap(&array[i], &array[i – 1]);
            done = 0;
        }
    }
    printf("%d", array[3]);
}
```

The output of the program is ___3___.

[2017 (Set-2) : 2 M]

1.105 Consider the following C code:

```
#include <stdio.h>
int *assignval (int *x, int val)
{   *x = val;
    return x;
}
void main( )
{   int *x = malloc (size of (int));
    if (NULL == x) return;
    x = assignval(x, 0);
    if(x)
    {   x = (int*) malloc (size of (int));
        if (NULL == x) return;
        x = assignval (x, 10);
    }
    printf ("%d"\n", *x);
    free (x);
}
```

The code suffers from which one of the following problems:

(a) compiler error as the return of malloc is not typecast appropriately
(b) compiler error because the comparison should be made as x == NULL and not as shown
(c) compiles successfully but execution may result in dangling pointer
(d) compiles successfully but execution may result in memory leak

[2017 (Set-1) : 1 M]

1.106 Consider the C program fragment below which is meant to divide x by y using repeated subtractions. The variables x, y, q and r are all unsigned int.

```
while
(r >= y)
{   r = r – y;
    q = q + 1;
}
```

Which of the following conditions on the variables x, y, q and r before the execution of the fragment will ensure that the loop terminates in a state satisfying the condition $x == (y \times q + r)$?

(a) (q == r) && (r == 0)
(b) (x > 0) && (r == x) && (y > 0)
(c) (q == 0) && (r == x) && (y > 0)
(d) (q == 0) && (y > 0)

[2017 (Set-2) : 2 M]

1.107 Consider the following C program:

```
#include <stdio.h>
struct ournode {
    char x, y, z;
};
int main( ) {
    struct ournode p = {'1', '0', 'a' + 2};
    struct ournode *q = &p;
    printf("%c, %c", *((char*) q + 1),
    *((char*) q + 2));
    return 0;
}
```

The output of this program is:

(a) 0, c (b) 0, a + 2
(c) '0', 'a + 2' (d) '0', 'c'

[2018 : 1 M]

1.108 Consider the following C program:

```
#include <stdio.h>
int counter = 0;
int calc (int a, int b) {
    int c;
    counter ++;
    if (b = = 3) return (a × a × a);
    else {
        c = calc (a, b/3);
        return (c × c × c);
    }
}
int main( ) {
    calc(4, 81);
    printf("%d", counter);
}
```

The output of this program is 4.

[2018 : 1 M]

1.109 Consider the following program written in pseudo-code. Assume that x and y are integers:

```
Count (x, y) {
    if (y ! = 1) {
        if (x ! = 1) {
            print ("*");
            Count (x/2, y);
        }
        else {
            y = y – 1;
            Count (1024, y);
        }
    }
}
```

The number of times that the print statement is executed by the call Count (1024, 1024) is 10230.

[2018 : 1 M]

1.110 Consider the following C program:

```
#include <stdio.h>
void fun1 (char *s1, char *s2) {
    char *tmp;
    tmp = s1;
    s1 = s2;
    s2 = tmp;
}
void fun2 (char **s1, char **s2) {
    char *tmp;
    tmp = *s1;
    *s1 = *s2;
    *s2 = tmp;
}
int main( ) {
    char *str1 = "Hi", *str2 = "Bye";
    fun1 (str1, str2); printf("%s %s", str1, str2);
    fun2 (&str1, &str2); printf("%s %s", str1, str2);
    return 0;
}
```

The output of the program above is

(a) Hi Bye Bye Hi
(b) Hi Bye Hi Bye
(c) Bye Hi Hi Bye
(d) Bye Hi Bye Hi

[2018 : 2 M]

1.111 Consider the following C code. Assume that unsigned long int type length is 64 bits.

```
unsigned long int fun (unsigned long int n) {
    unsigned long int i, j = 0, sum = 0;
    for (i = n; i > 1; i = i/2) j++;
    for (; j > 1; j = j/2) sum++;
    return (sum);
}
```

The value returned when we call fun with the input 2^{40} is

(a) 4 (b) 5
(c) 6 (d) 40

[2018 : 2 M]

1.112 Consider the following C program:

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

The number that will be displayed on execution of the program is 6.

[2019 : 1 M]

1.113 Consider the following C program:

```
#include <stdio.h>
int jumble(int x, int y) {
    x = 2 * x + y;
    return x;
}
int main( ) {
    int x = 2, y = 5;
    y = jumble(y, x);
    x = jumble(y, x);
    printf("%d\n", x);
    return 0;
}
```

The value printed by the program is 26.

[2019 : 1 M]

1.114 Consider the following C function:

```
void convert (int n) {
    if (n < 0)
        printf("%d", n);
    else {
        convert (n/2);
        printf("%d", n % 2);
    }
}
```

Which one of the following will happen when the function convert is called with any positive integer n as argument?

(a) It will print the binary representation of n in the reverse order and terminate.
(b) It will print the binary representation of n but will not terminate.
(c) It will not print anything and will not terminate.
(d) It will print the binary representation of n and terminate.

[2019 : 2 M]

1.115 Consider the following C programs:

```
#include <stdio.h>
int main( ) {
    float sum = 0.0, j = 1.0, i = 2.0;
    while (i/j > 0.0625) {
        j = j + j;
        sum = sum + i/j;
        printf("%f\n", sum);
    }
    return 0;
}
```

The number of times the variable sum will be printed, when the above program is executed, is 5.

[2019 : 2 M]

1.116 Consider the following C program:

```
#include<stdio.h>
int r( )
{
    Static int num = 7;
    return num --;
}
int main( )
{
    for ( r( ); r( ); r( ) )
        printf("%d",r( ));
    return 0;
}
```

Which one of the following values will be displayed on execution of the programs?

(a) 41 (b) 630
(c) 63 (d) 52

[2019 : 2 M]

1.117 Consider the following C program:

```
#include <stdio.h>
int main( )
{
    int a[ ] = {2, 4, 6, 8, 10};
    int i, sum = 0, *b = a + 4;
    for (i = 0; i < 5; i++)
        sum = sum + (*b – i) – *(b – i);
    printf ("%d\n", sum);
    return 0;
}
```

The output of the above C program is 10.

[2019 : 2 M]

1.118 Consider the following C functions:

```
int tob(int b, int* arr) {
    int i;
    for(i = 0, b > 0; i++) {
        if(b%2) arr[i] = 1;
        else    arr[i] = 0;
        b = b/2;
    }
    return(i);
}
```

```
int pp(int a, int b) {
    int arr[20];
        int i, tot = 1, ex, len;
    ex = a;
    len = tob(b,arr);
    for(i = 0; i < len; i++) {
            if(arr[i] == 1)
            tot = tot * ex;
            ex = ex * ex;
    }
    return(tot);
}
```

The value returned by pp(3, 4) is 81.

[2020 : 2 M]

1.119 Consider the following C functions:

```
int fun1(int n) {
    static int i = 0;
    if (n > 0) {
        ++i;
        fun1(n – 1);
    }
    return(i);
}
```

```
int fun2(int n) {
    static int i = 0;
    if (n > 0) {
        i = i + fun1(n);
        fun2(n – 1);
    }
    return(i);
}
```

The return value of fun2 (5) is 55. **[2020 : 2 M]**

1.120 Consider the following ANSI C function:

```
int SimpleFunction (int Y[ ], int n, int x)
{
    int total = Y[0], loopIndex;
    for (loopIndex = 1; loopIndex <= n – 1; loopIndex++)
    total = x × total + Y[loopIndex];
    return total;
}
```

Let Z be an array of 10 elements with $Z[i] = 1$, for all i such that $0 \le i \le 9$. The value returned by SimpleFunction (Z, 10, 2) is 1023. **[2021 (Set-1) : 2 M]**

1.121 Consider the following ANSI C program:

```
#include <stdio.h>
int main( )
{
    int i, j, count;
    count = 0;
    i = 0;
    for (j = –3; j <= 3; j++)
    {
        if ((j >= 0) && (i++))
            count = count + j;
    }
    count = count + i;
    printf("%d", count);
    return 0;
}
```

Which one of the following options is correct?

(a) The program will compile successfully and output 8 when executed.
(b) The program will compile successfully and output 13 when executed.
(c) The program will not compile successfully.
(d) The program will compile successfully and output 10 when executed **[2021 (Set-1) : 2 M]**

1.122 Consider the following ANSI C function:

```
int SomeFunction(int x, int y)
{
    if ((x == 1) || (y == 1)) return 1;
    if (x == y) return x;
    if (x > y) return SomeFunction(x – y, y);
    if (y > x) return SomeFunction(x, y – x);
}
```

The value returned by SomeFunction(15, 255) is 15. **[2021 (Set-2) : 1 M]**

1.123 Consider the following ANSI C program:

```
#include <stdio.h>
int foo(int x, int y, int q)
{
    if ((x <= 0) || (y <= 0))
        return q;
    if (x <= 0)
        return foo(x, y – q, q)
    if (y <= 0)
        return foo(x – q, y, q);
    return foo(x, y – q, q) + foo(x – q, y, q);
}
int main( )
{
    int r = foo(15, 15, 10);
    printf("%d", r);
    return 0;
}
```

The output of the program upon execution is 24. **[2021 (Set-2) : 2 M]**

1.124 Consider the following ANSI C program:

```
#include <stdio.h>
int main( ) {
    int arr[4] [5] ;
    int i, j;
    for (i = 0; i < 4; i++) {
        for (j = 0; j < 5; j++) {
            arr[i] [j] = 10 * i + j;
        }
    }
    printf("%d", * (arr[i] + 9));
    return 0;
}
```

What is the output of the above program?

(a) 24 (b) 20
(c) 14 (d) 30 **[2021 (Set-2) : 1 M]**

1.125 What is printed by the following ANSI C program?

```
#include<stdio.h>
int main(int argc, char *argv[ ])
{
    int x = 1, z[2] = {10, 11};
    int *p = NULL;
    p = &x;
    *p = 10;
    p = &z[1];
    *(&z[0] + 1) += 3;
    printf("%d, %d, %d\n", x, z[0], z[1]);
    return 0;
}
```

(a) 1, 10, 11 (b) 1, 10, 14
(c) 10, 14, 11 (d) 10, 10, 14 **[2022 : 1 M]**

1.126 What is printed by the following ANSI C program?

```
#include<stdio.h>
int main(int argc, char *argv[ ])
{
    int a[3][3][3] = {{1, 2, 3, 4, 5, 6, 7, 8, 9},
    {10, 11, 12, 13, 14, 15, 16, 17, 18},
    {19, 20, 21, 22, 23, 24, 25, 26, 27}};
    int i = 0, j = 0, k = 0;
    for(i = 0; i < 3; i++) {
        for(k = 0; k < 3; k++)
            printf("%d", a[i][j][k]);
        printf("\n");
    }
    return 0;
}
```

(a) 1 2 3 / 10 11 12 / 19 20 21
(b) 1 4 7 / 10 13 16 / 19 22 25
(c) 1 2 3 / 4 5 6 / 7 8 9
(d) 1 2 3 / 13 14 15 / 25 26 27 **[2022 : 2 M]**

1.127 What is printed by the following ANSI C program?

```
#include<stdio.h>
int main(int argc, char *argv[ ]) {
    char a = 'P';
    char b = 'x';
    char c = (a & b) + '*';
    char d = (a | b) – '–';
    char e = (a ^ b) + '+';
    printf("%c %c %c\n", c, d, e);
    return 0;
}
```

ASCII encoding for relevant characters is given below:

A	B	C	...	Z
65	66	67	...	90

a	b	c	...	z
97	98	99	...	122

*	+	–
42	43	46

(a) z K S (b) 122 75 83
(c) * – + (d) P x + **[2022 : 2 M]**

1.128 The integer value printed by the ANSI-C program given below is 7.

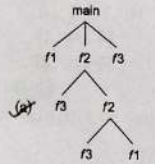
```
#include<stdio.h>
int funcp( ) {
        static int x = 1;
        x++;
        return x;
}
int main(){
        int x, y;
        x = funcp( );
        y = funcp( ) + x;
        printf("%d\n", (x + y));
    return 0;
}
```

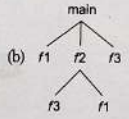
[2023 : 1 M]

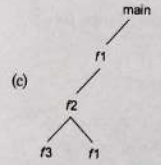
1.129 Consider the following program:

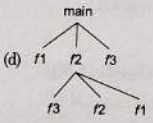
```
int main( )
{
    f1( );
    f1(2);
    f3( );
    return (0);
}

int f1( )
{
    return (1);
}

int f2(int X)
{
    f3( );
    if (X == 1)
        return f1( );
    else
        return (X * f2(X – 1));
}

int f3( )
{
    return(5);
}
```

Which one of the following options represents the activation tree corresponding to the main function?

(a) 

(b) 

(c) 

(d) 

[2023 : 2 M]

1.130 Consider the following C program:

```
# include <stdio.h>
int main( ) {
    int a = 6;
    int b = 0;
    while (a < 10) {
        a = a / 12 + 1;
        a += b; }
    printf("%d", a);
    return 0; }
```

Which one of the following statements is CORRECT?

(a) The program prints 6 as output
(b) The program prints 10 as output
(c) The program prints 9 as output
(d) The program gets stuck in an infinite loop

[2024 (Set-1) : 1 M]

1.131 Consider the following program:

```
# include <stdio.h>
void fX( );
int main( ) {
    fX( );
    return 0; }

void fX( ) {
    char a;
    if ((a=getchar ( )) != '\n' )
        fX( );
    if (a != '\n' )
        putchar (a); }
```

Assume that the input to the program from the command line is 1234 followed by a newline character. Which one of the following statements is CORRECT?

(a) The program will terminate with 1234 as output
(b) The program will not terminate
(c) The program will terminate with no output
(d) The program will terminate with 4321 as output

[2024 (Set-1) : 1 M]

1.132 Consider the operator precedence and associativity rules for the integer arithmetic operators given in the table below.

Operator	Precedence	Associativity
+	Highest	Left
–	High	Right
*	Medium	Right
/	Low	Right

The value of the expression 3 + 1 + 5 * 2 / 7 + 2 – 4 – 7 – 6 / 2 as per the above rules is 6.

[2024 (Set-1) : 1 M]

1.133 Consider the following C function definition.

```
int f (int x, int y) {
    for (int i = 0; i < y; i++) {
        x = x + x + y;
    }
    return x;
}
```

Which of the following statements is/are TRUE about the above function?

(a) If the inputs are $x = 20$, $y = 10$, then the return value is greater than 2^{20}
(b) If the inputs are $x = 20$, $y = 20$, then the return value is greater than 2^{20}
(c) If the inputs are $x = 20$, $y = 10$, then the return value is less than 2^{10}
(d) If the inputs are $x = 10$, $y = 20$, then the return value is greater than 2^{20}

[2024 (Set-1) : 2 M]

1.134 Consider the following C function definition:

```
int fX(char *a) {
    char *b = a;
    while (*b)
        b++;
    return b – a; }
```

Which of the following statements is/are TRUE?

(a) Assuming a character array c is declared as char c[ ] = "abcd" in main( ), the function call fX(c) will always return a value.
(b) The function call fX("abcd") will always return a value.
(c) The code of the function will not compile.
(d) Assuming a character pointer c is declared as char *c = "abcd" in main( ), the function call fX(c) will always return a value.

[2024 (Set-2) : 1 M]

1.135 Consider the following C program. Assume parameters to a function are evaluated from right to left.

```
# include <stdio.h>
int g(int p) { printf ("%d", p); return p; }
int h(int q) { printf ("%d", q); return q; }
void f(int x, int y) {
    g(x);
    h(y);
}
int main( ) {
    f(g(10), h(20));
}
```

Which one of the following options is the CORRECT output of the above C program?

(a) 20102010 (b) 10202010
(c) 10201020 (d) 20101020

[2024 (Set-2) : 1 M]

1.136 What is the output of the following C program?

```
#include <stdio.h>
int main( ) {
    double a[2]={20.0,25.0},*p, *q;
    p = a;
    q = p + 1;
    printf ("%d, %d",(int)(q - p),(int)(*q - *p));
    return 0; }
```

(a) 4, 8 (b) 1, 5
(c) 1, 8 (d) 8, 5

[2024 (Set-2) : 2 M]

1.137 Consider an array X that contains n positive integers. A subarray of X is defined to be a sequence of array locations with consecutive indices.
The C code snippet given below has been written to compute the length of the longest subarray of X that contains at most two distinct integers. The code has two missing expressions labelled (P) and (Q).

```
int first = 0, second = 0, len1 = 0, len2 = 0, maxlen = 0;
for (int i = 0; i < n; i++) {
    if (X[i] == first) {
        len2++; len1++;
    } else if (X[i] == second) {
        len2++;
        len1 = (P);
        second = first;
    } else {
        len2 = (Q);
        len1 = 1; second = first;
    }
    if (len2 > maxlen) {
        maxlen = len2;
    }
    first = x[i];
}
```

Which one of the following options gives the CORRECT missing expressions?
(**Hint:** At the end of the i^{th} iteration, the value of len1 is the length of the longest subarray ending with X[i] that contains all equal values, and len2 is the length of the longest subarray ending with X[i] that contains at most two distinct values.)

(a) (P) len2 + 1 (Q) len1 + 1
(b) (P) 1 (Q) len1 + 1
(c) (P) len1 + 1 (Q) len2 + 1
(d) (P)1 (Q) len2 + 1

[2024 (Set-2) : 2 M]

1.138

```
#include <stdio.h>
void foo(int *p, int x) {
    *p = x;
}
int main( ) {
    int *z;
    int a = 20, b = 25;
    z = &a;
    foo(z, b);
    printf("%d", a);
    return 0;
}
```

The output of the given C program is 25. (Answer in integer)

[2025 (Set-1) : 1 M]

1.139

```
#include <stdio.h>
int foo(int S[ ], int size) {
    if(size == 0) return 0;
    if(size == 1) return 1;
    if(S[0] != S[1]) return 1 + foo(S + 1, size - 1);
    return foo(S + 1, size - 1);
}
int main( ) {
    int A[ ] = {0, 1, 2, 2, 2, 0, 0, 1, 1};
    printf("%d", foo(A, 9));
    return 0;
}
```

The value printed by the given C program is 5. (Answer in integer)

[2025 (Set-1) : 2 M]

1.140 Consider the following C program:

```
#include <stdio.h>
int gate (int n) {
    int d, t, newnum, turn;
    newnum = turn = 0; t = 1;
    while (n >= t) t *= 10;
    t /=10;
    while (t > 0) {
        d = n/t;
        n = n%t;
        t /= 10;
        if (turn) newnum = 10*newnum + d;
        turn = (turn + 1) % 2;
    }
    return newnum;
}
int main( ) {
    printf ("%d", gate(14362));
    return 0;
}
```

The value printed by the given C program is 46. (Answer in integer)

[2025 (Set-1) : 2 M]

1.141

```
int x = 126, y = 105;
do {
    if(x > y) x = x - y;
    else y = y - x;
} while(x!=y);
printf("%d", x);
```

The output of the given C code segment is 21. (Answer in integer)

[2025 (Set-2) : 1 M]

1.142 Consider the following C program:

```
#include <stdio.h>
int main( ) {
    int a;
    int arr[5] = {30, 50, 10};
    int *ptr;
    ptr = &arr[0] + 1;
    a = *ptr;
    (*ptr)++;
    ptr++;
    printf("%d", a + (*ptr) + arr[1]);
    return 0;
}
```

The output of the above program is 111. (Answer in integer)

[2025 (Set-2) : 2 M]

1.143 Consider the following C program:

```
#include <stdio.h>
int g(int n) {
    return (n+10);
}
    int f(int n) {
    return g(n*2);
}
int main( ) {
    int sum, n;
    sum=0;
    for (n=1; n<3; n++)
    sum += g(f(n));
    printf("%d", sum);
    return 0;
}
```

The output of the given C program is 46. (Answer in integer)

[2025 (Set-2) : 2 M]

■■■■

Answers Programming

1.1	(d)	1.2	(b)	1.3	(d)	1.4	(c)	1.5	(a)	1.6	(d)	1.7	(a)	1.8	(d)	1.9	(a)		
1.10	(c)	1.11	(a)	1.12	(b)	1.13	(d)	1.14	(d)	1.15	(b)	1.16	(b)	1.17	(b)	1.18	(a)		
1.19	(d)	1.20	(a)	1.21	(c)	1.22	(d)	1.23	(b)	1.24	(c)	1.25	(a)	1.26	(b)	1.27	(c)		
1.28	(c)	1.29	(a)	1.30	(d)	1.31	(c)	1.32	(c)	1.33	(d)	1.34	(d)	1.35	(c)	1.36	(d)		
1.37	(d)	1.38	(c)	1.39	(b)	1.40	(d)	1.41	(a)	1.42	(c)	1.43	(d)	1.44	(b)	1.45	(c)		
1.46	(d)	1.47	(d)	1.48	(c)	1.49	(b)	1.50	(d)	1.51	(c)	1.52	(a)	1.53	(b)	1.54	(c)		
1.55	(c)	1.56	(b)	1.57	(d)	1.58	(c)	1.59	(b)	1.60	(d)	1.61	(b)	1.62	(d)	1.63	(c)		
1.64	(d)	1.65	(c)	1.66	(b)	1.67	(d)	1.68	(c)	1.69	(c)	1.70	(d)	1.71	(b)	1.72	(d)		
1.73	(c)	1.74	(9)	1.75	(b)	1.76	(1.73)	1.77	(d)	1.78	(–5)	1.79	(b)	1.80	(51)	1.81	(d)		
1.82	(5)	1.83	(c)	1.84	(140)	1.85	(b)	1.86	(10)	1.87	(230)	1.88	(d)	1.89	(2016)	1.90	(d)		
1.91	(a)	1.92	(d)	1.93	(30)	1.94	(c)	1.95	(3)	1.96	(23)	1.97	(c)	1.98	(3)	1.99	(a)		
1.100	(a)	1.101	(c)	1.102	(2)	1.103	(0)	1.104	(3)	1.105	(d)	1.106	(c)	1.107	(a)	1.108	(4)		
1.109	(10230)			1.110	(a)	1.111	(b)	1.112	(6)	1.113	(26)	1.114	(c)	1.115	(5)	1.116	(d)		
1.117	(10)	1.118	(81)	1.119	(55)	1.120	(1023)			1.121	(d)	1.122	(15)	1.123	(24)	1.124	(a)		
1.125	(d)	1.126	(a)	1.127	(a)	1.128	(7)	1.129	(a)	1.130	(d)	1.131	(d)	1.132	(6)	1.133	(b, d)		
1.134	(a, b, d)			1.135	(d)	1.136	(b)	1.137	(b)	1.138	(25)	1.139	(5)	1.140	(46)	1.141	(21)		
1.142	(111)	1.143	(46)																

Explanations Programming

1.1 (d)

Activation record → Subroutine call
Location counter → Assembler
Reference counts → Garbage collection
Address Relocation → Linking loader

1.2 (b)

Heap allocation is required to support dynamic data structure where memory can be allocated and deallocated at run time.

1.3 (d)

Activation tree helps to access the non-local variables in faster using an array of pointers to activation records.

1.4 (c)

Starting value of 'x' stores in global variable.
Var x = 0.25; real, after that put value according to question.

1.5 (a)

Pointers can not be implemented if the processor supports only immediate and direct addressing modes. Pointer requires indirect addressing modes.

1.6 (d)

- Take some values and verify the options as 10, 6, 7.

1.7 (a)

It defines an array each element of which is a pointer to a structure of type node.

1.8 (d)

```
m = malloc (S);
m = NULL;
```
→ Lost memory: (memory is lost for m)

```
free (n);
n → values = 5;
```
→ Dangling pointer: (No memory for n)

```
char * p;
*p = 'a';
```
→ Uninitialized pointer: (p is not initilized yet. So trying for *p will fail)

1.9 (a)

Definition of aliasing.

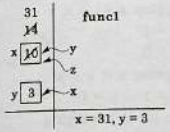
1.10 (c)

Here structure creates the minimum of array and union, but union only creates the minimum for only 'long z' which is max.
So total minimum required = 18.

1.11 (a)

Count is static variable so every time when value is changed it becomes the new value of count. So, first count value is 0 then 1, then 3, then 6 at last 10.

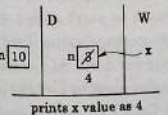
1.12 (b)



1.13 (d)

- P[1] will give segmentation fault, since when function is over, space allocated to local variable will be deallocated and accessing that location will give segmentation fault because address of local variable is return.
- P[2] also give segmentation fault, since px is returned which contain address of memory 10, when function is over accessing that memory will give segmentation fault because may be memory 10 will be allocated to some other pointer variable.
- P[3] may give segmentation error, when system run out of memory and dynamic memory will not allocated to pointer px.

1.14 (d)



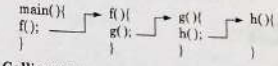
1.15 (b)

Call by value result and call by reference may differ in the presence of aliasing (sending the same parameter more than once).

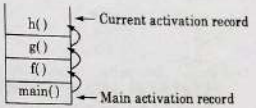
1.16 (b)

In C language number of activation records between the current activation and the activation record for the main depends on the actual function calling sequence.

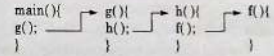
Example 1:



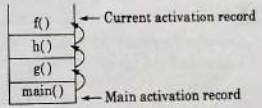
Calling sequence:



Example 2:



Calling sequence:



1.17 (b)

Address of a [40][50] = Base address + 40 × 100 × element size + 50 * element size
= 0 + 4000 × 1 + 50 × 1
= 4050
Therefore, option (b) is correct.

1.18 (a)

Here int *A[10] is an array of 10 pointers.
Which can be A[2][3] as its left hand side.
It can also be A[2] as its left hand side and also B[2][3] can obviously be used as left hand side as it is given in declaration of B[10][10] only, but B[1] can not be used because it is declared as 2-dimensional array.

1.19 (d)

1. First for loop fill array with '0' from index 3 to n.
2. Second for loop location where index j is not multiple of k i.e. non multiple index of k and update index of non multiple index of 4 as 1.

Since we are updating one array A as 1 because of we are doing step 2 for every value of 2 to $2 \times \log_2 n$. So if (! A[j] will return false for each index of array 4 because we set each index in step 2. So it will print nothing.

1.20 (a)

```
x = 5;
p (& x);
x = *y + 2 = 5 + 2 = 7;
Q(x);
z = 7;
z = z + x = 12  ⇒   print(z) = 12
*y = x – 1 = 6  ⇒   print(x) = 7
print(x) = 6
```

1.21 (c)

The goal of structured programming is able to infer the flow of control from the program text.

1.22 (d)

Consider the given C function:

```
void swap (int a, int b)
{   int temp;
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

The main program for given code is

```
#include <stdio.h>
void main()
{   int x, y;
    x = 1;
    y = 2;
    swap(x, y);
    printf("%d\n", a);
    printf("%d\n", y);
}
```

If we call swap (x, y) then there is no interchange in the value of x and y because the parameters are passed by value. There is interchange in formal parameters a and b but no interchange in actual parameter x and y because the scope of a and b lies with in the function but not in the main program so the value of x and y remain unchanged. If we will call it by reference then the value of x and y will be interchanged as below code.

```
void swap (int * a, int * b)
{   int temp ;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

So option (d) is the current.

1.23 (b)

```
Put     x = 0
then    x = 1
so use option x = 1 – x
Put     x = 1
then    x = 0
so use option x = 1 – x
          = 1 – 1 = 0
```

So option (b) is matching.

1.24 (c)

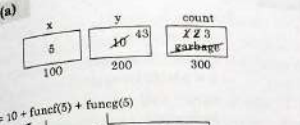
Code for Transpose of matrix:

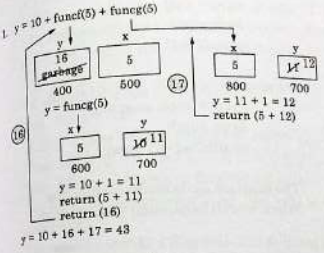
```
for(i = 0; i < 4; ++i)
   {
   for(j = i; j < 4; ++j)
      {
         t = M[j][i];
         M[i][j] = M[j][i];
         M[j][i] = t;
      }
   }
```

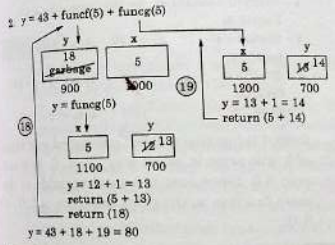
Key part here is both index i.e. i and j should not start with same index. Otherwise transpose will be done 2 time which results original matrix. So here first for loop start with i = 0. So second for loop must be start with j = i. Since t initially contain garbage value if we do M[i] [j] = t then we loose our data. So, inside second for loop 1st statement must be t = M[j] [i].

Note: Take 2 × 2 matrix and run the code on it. Will save time in exam.

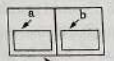
1.25 (a)







1.26 (b)



1. main()
   Struct t s = {"A", "B"}; so structure s will be created of same structure as t and value assign A to S → a and B is assign to S → b.
   printf (s.a, s.b) will print A.B.
2. f1(s)
   f1(t s)
   a
   b
   A U
   B V
   s.a = 'U'
   s.b = 'V'
   Printf (s.a, s.b) will print U and V.
   Call by value
3. printf (s.a, s.b) will print A.B.
4. f2(&s)
   f2(t * p)
   1006
   *p
   P → s = 'V'
   P → b = 'W'
   Printf (P → a, P → b) will print V and W.
   Call by reference

1.27 (c)

```
1. int f(int n)
2. {    static int i = 1;
3.      if (n >= 5) return n;
4.      n = n + i;
5.      i ++;
6.      return f(n)
7. }
```

Iteration	n	i
0	1	1
Initilize 1	1 + 1 = 2	2
2	2 + 2 = 4	3
3	4 + 3 = 7	4

So when we call f(1), the value i = 4 and n = 7, if condition is 7 ≥ 5 so returns the value 7.

1.28 (c)

The given C program is non-recursive. So we will get the value of n after some iteration. The main thing about the program is that.

1. If m = n then n is the GCD
2. If m > n then m = m – n
3. If m < n then n = n – m

For example let x = 40 and y = 32

```
m = x = 40 and n = y = 32
m > n so apply the 2nd condition
m = m – n
m = 40 – 32 = 8
m = 8 and n = 8
```

apply the first condition
GCD (40, 32) = 8

1.29 (a)

Loop invariant is the part of code motion. Loop invariant for while loop is a condition, if we assign this condition before the while loop then there is no effect in the code and produces same output.

Consider the given code.

```
int n, rev;
rev = 0;
while (n > 0){
    rev = rev * 10 + n%10;
    n = n/10;
}
```

Let input n = 12345
where $d_1 = 1$, $d_2 = 2$, $d_3 = 3$,
$d_4 = 4$ and $d_5 = 5$

Iteration	n = n/10 $d_2d_3d_4d_5$	rev = rev * 10 + n%10
0	1 2 3 4 5 $d_1d_2d_3d_4$	rev = 0 d_5
1	1 2 3 4 $d_1d_2d_3$	rev = 0 * 10 + 5 = 5 d_5d_4
2	1 2 3 d_1d_2	rev = 5 * 10 + 5 = 5 4 $d_5d_4d_3$
3	1 2 d_1	rev = 54 * 10 + 2 = 5 4 3 $d_5d_4d_3d_2$
4	1	rev = 543 * 10 + 2 = 5 4 3 2 $d_5d_4d_3d_2d_1$
5	0	rev = 5432 * 10 + 1 = 5 4 3 2 1

Lent n = $d_1d_2 \ldots d_m$ in the i^{th} iteration
rev = $d_m.d_{m-1} - i + 1$
For example if m = 5 and i = 3 the above example produces $n = d_1d_{5-3} = 12$ and
$rev = d_5d_4d_{5-3+1} = 543$

1.30 **(d)**

Because of first character is '\0' nothing will be printed.

1.31 **(c)**

int (*f) (int*);
return type int, (*f) is a pointer to a function the argument is (int*) an integer pointer So, int (*f) (int*) means a pointer to a function that takes an integer pointer as an argument and returns an integer.

1.32 **(c)**

The abstract data type (ADT) refers to a programmer defined data type together with a set of operations that can be performed on that data so the choice (c) is correct.

1.33 **(d)**

Without declaration we can't call any function. If you call it will assume by default return data type as integer. But here return data type is double. So it will give compile time error type mis-match.

1.34 **(d)**

```
void foo(int n, int sum) {
        int k = 0, j = 0;
        if (n == 0) return;
        k = n%10; j = n/10;
        sum = sum + k;
        foo(j, sum);
        printf("%d", k);
}
int main(){
        int a = 2048, sum = 0;
        foo(a, sum);
        printf("%d\n", sum);
}
```

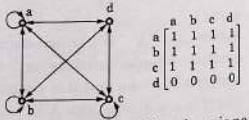
The function foo is recursive function
When we call foo(a, sum) = foo (2048, 0)

k	j	sum
k = 2048%10 = foo(204, 8) foo(204, 8)	j = 204	sum = 0 + 8 = 8
k = 204%10 = 4 foo(20, 12)	j = 20	sum = 8 + 4 = 12
k = 20%10 = 0 foo(2, 12)	j = 2	sum = 12 + 0 = 12
k = 2%10 = 2	j = 0	sum = 12 + 2 = 14

foo(0, 14) function will be terminated and value of k will print in stack way i.e. 2, 0, 4, 8 and sum = 0. Since sum is a local variable in the main function so the print sequence is 2, 0, 4, 8, 0.

1.35 **(c)**

A sink in a directed graph is a vertex i such that there is an edge from every other vertex $j \neq i$ to i and there is no edge from i to any other vertex.



	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	1	1	1	1
d	0	0	0	0

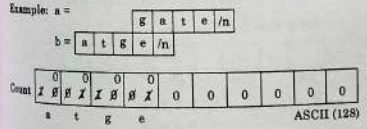
In above image 'd' is sink i.e. there is no outgoing edge by checking row of all vertex if we find any row with all zero, then we find a potential vertex for sink. While (j < n) &&! A[i] [j]) will check every entry row is 0 or not and until it find 1 in row, increment j. If it find 1 in row skip a row by which it does not has any incoming edge by doing i = j.

1.36 **(d)**

In this program initially flag is set to 1 i.e. assume sink is present in the graph. And by using For loop if flag is reset to 0 then sink is not present otherwise sink is present.
For loop will set flag = 0 if the coloum entries are all 1 for sink (vertex) and the entries for row is all 0.
If we look at above diagram for vertex 'd', i = 3 and j = 0.
So while ((j! = i) && (A[3] [0] || !A[0] [3])
True && (0 || !1))
True && false = false. so flag not set to 0.
Then while ((j! = j) && (A[3] [1] || !A[1] [3])
True && false = false, so flag not set to 0.
Try for other vertex we will set flag = 0 i.e. not exist sink.
Like this for vertex d flag will not set to set to 0 i.e. sink exist.

1.37 **(d)**

Definition of anagram: a string s is a string obtained by permuting the letter in s.
1st for loop just create an array count and initialize to 0 for each element i.e. for 128 element.
While loop run till string reaches to end of strings.
In while loop, for every element a[i] count is increased which is decreased by array b[i] when same element arrive in array b[i] and increment loop with j++. If both strings are anagram of each other the count value will be zero for each elements.
So, count[a[j]] ++; and count[b[j++]] − −;



Inside while loop

(1) count[g]++;
count[a]− −;
j++;

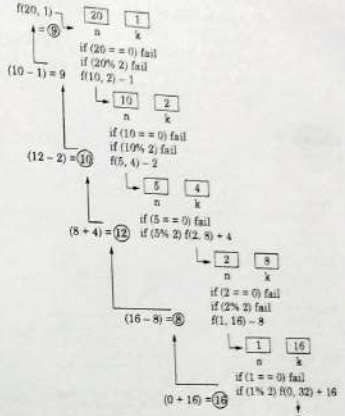
(2) count[a]++;
count[t]− −;
j++;

(3) count[t]++;
count[g]− −;
j++;

(4) count[e]++;
count[e]− −;
j++;

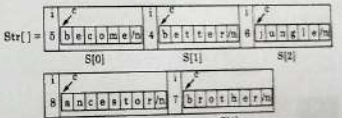
While loop fail for (5).
Since atlast count will contain all zero. So both are anagram of each other.

1.38 **(c)**



So, 9 will be printed.

1.39 **(b)**

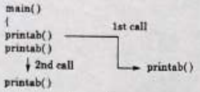


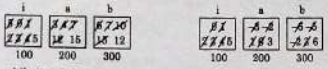
test *p = str; here p is pointing to str[ ] now.
p+ = 1; p = p + 1
Now p is pointing to S[1].
++p → c evaluate as ++(p → c) i.e. now p is pointing to b e t t e r /n.

So,
1. Printf ("%s", p++ → c); prints "etter".
   p++; p points to S[2]
2. Printf ("%c", * ++ p → c); evaluated as *(++ (p → c)) i.e. p is now points to j u n g l e /n now p is increment to 1 character i.e j u n g l e /n. Which is printed i.e. 'u'.

3. Printf ("%d", p[0].i); since after printf p is pointing to S[2]. So p[0].i print integer part of S[2] i.e. 6.
4. Printf (""%s", p→c); since in printf '2' p point to char string 2nd element so at will print "ungle".

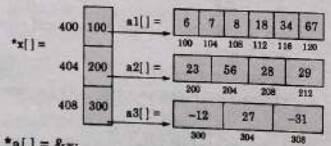
1.40 (d)





while (0 ≤ 4) (True)	while (0 ≤ 4)
if (0% 2 = = 1) i++; False	if (0% 2 = = 1) i++; False
a = a + i = 3 + 1 = 4	a = a + i = –3 + 1 = –2
a = b + i = 6 + 1 = 7	a = b + i = –6 + 1 = –5
while (1 ≤ 4) (True)	while (1 ≤ 4)
if (1% 2 = = 1) i++; True	if (1% 2 == 1) i++; True
while (2 ≤ 4) (True)	while (2 ≤ 4)
if (2% 2 = = 1) i++; False	if (2% 2 == 1) i++; False
a = a + i = 4 + 3 = 7	a = a + i = –2 + 3 = 1
b = b + i = 7 + 3 = 10	b = b + i = –5 + 3 = –2
while (3 ≤ 4) (True)	while (3 ≤ 4)
if (3% 2 = = 1) i++; True	if (3% 2 == 1) i++; True
while (4 ≤ 4) (True)	while (4 ≤ 4)
if (4% 2 = = 1) i++; False	if (4% 2 = = 1) i++; False
a = a + i = 7 + 5 = 12	a = a + i = 1 + 5 = 6
b = b + i = 10 + 5 = 15	b = b + i = –2 + 5 = 3
while (5 ≤ 4) (False)	while (5 ≤ 4) (False)
Now swap function swap value of a and b variable	Now swap function swap value of a and b variable
i.e. a = 15, b = 12	i.e. a = 3, b = 6
Print (a, b) ⇒ a = 15 and b = 12	Print (a, b) ⇒ a = 3 and b = 6

1.41 (a)



*a[ ] = &x;

*a[ ] contain base address of array x;

1. Printf (a[0] [2]) = *(*(a + 0) + 2) = *(100 + 2 × 4) = *(108) = 8
2. Printf (*a[2]) = *(*(a + 2) + 0) = *(408 + 0) = *(408) = –12
3. Printf (*++ a[0]) = *(++(a + 0)) = *(++(100)) = *(104) = 7
4. Printf (*(++ a)[0]) = *(*(a + 1) + 0) = *(*(404) + 0) = *(200 + 0) = *(200) = 23
5. Printf (a[–1] [+1]) = *(*(a – 1) + 1) = *(*(404 – 4) +1) = *(104+1) = *(104 + 1 × 4) = *(108) = 8

Note: In step 3 'a1' point to 'a1' = 104 and in step 4 'a' point to 404 as starting address used in step 5.

1.42 (c)

When n = 0, $^mC_n = {^mC_0} = 1$

m = n, $^mC_n = {^mC_m} = 1$

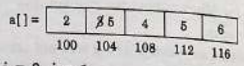
1.43 (d)

If array a and b contain identical elements then neither (i) nor (ii) holds.

1.44 (b)

Let a[ ] be an array:

Working of work 1:



let i = 2, j = 1

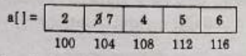
1. x = a[100 + 2 × 4]
   = a[108]
2. a[1] = 4 + 1 = 5
3. return a[i + 2] – 3

⇒ a[100 + 2 × 4] – 3

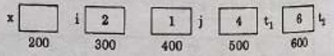
⇒ a[108] – 3

⇒ 4 – 3 = 1

Working of work 2:



let i = 2, j = 1



1. t_1 = i + 2
   = 2 + 2
2. t_2 = a[4]
   = a[116] = 6
3. a[1] = t_2 + 1
   = 6 + 1
   = 7
4. return t_2 – 3

⇒ 6 – 3 = 3

Both work 1 and work 2 will produce different output some time and work 2 is compute i + 2 one time, hence take less time to compute where as work 1 will compute 2 times.

1.45 (c)

- Given code will not give any compilation error.
- When an address passed to pointer variable px and py will be deallocated by some other function, then access those memory location will generate segmentation fault.

Eg:
```
fun () {
    int b = 10;
    int * p = (int *) malloc (size of (int));
        *p = &b;
    int * a = (int *) malloc (size of (int));
        *a = p;
    free (p);
    call (int &a, &a);
}
```

will generate segmentation fault because memory allocated to p will be deallocated which is pointed by pointer variable a, which is passed as parameter.

- Swap procedure work correctly when px and py will have different memory location but when both px and py will contain same memory address then it will not work correctly or lost 1 variable value.

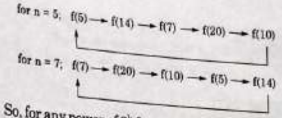
1.46 (d)

To compute f(5), initially r = 5.

f(3) + 2 = 18 ↑	[(n > 3) : f(n – 2) + 2]
f(2) + 5 = 16 ↑	[f(n – 2) + r]
f(1) + 5 = 11 ↑	[f(n – 2) + r]
f(0) + 5 = 6 ↑	[f(n – 2) + r]
1	[n ≤ 0]

1.47 (d)

- By analysing code we see when $n = 2^k$ i.e. power of 2 then f(n/2) will definitely terminate since n% 2 always equal to 0. Hence after $\log_2 n$ operation n = i and function will terminate.
- When we input number like 5 and 7 it will enter into infinite loop i.e.



So, for any power of 2^k function terminates which is infinite so option (b) is true and for infinite values of n for which function will not terminate i.e. number 5×2^k, 14×2^k, 7×2^k, 20×2^k and 10×2^k for any value of k.

So option (d) is true.

1.48 (c)

Syntax error in above code segment. Right code given below:

```
# include <stdio.h>
int main( )
{
    int sum = 0, maxsum = 0, i, n = 6;
    int a[ ] = {2, –2, –1, 3, 4, 2};
    for (i = 0; i < n; i++) {
        if (i == 0 | | a[i] < 0 | | a[i] < a[i – 1])
        {
            if (sum > maxsum) maxsum = sum;
            sum = (a[i] > 0) ? a[i] : 0;
        }
        else
        sum += a[i];
    }
    if (sum > maxsum) maxsum = sum;
    printf ("%d\n", maxsum);
}
```

Simply execute and trace value of sum and maxsum for i = 0 to 5

i → initially	0	1	2	3	4	5
sum	0	0	2	0	3	7	2
maxsum	0	0	2	2	2	2	7

so maxsum = 7

1.49 (b)

push 5

push 2

push 5*2 = 10. (pops 5 and 2)

push 3

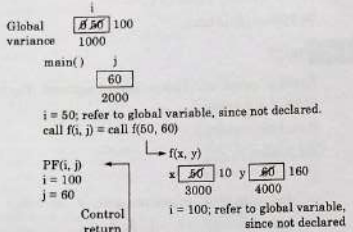
push 3

push 2

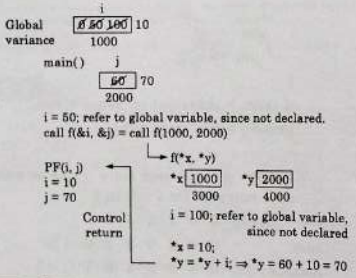
push 3 + 2 = 5 (pops 2 and 3)
push 5 * 3 = 15 (pops (5 and 3)
push 15 + 10 = 25 (pops (15 and 10)

1.50 (d)

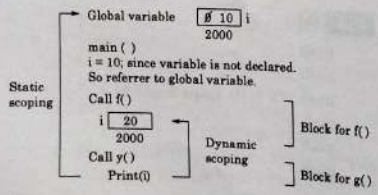
Using call by value: In call by value actual value are passed to formal parameter and operation applied on formal parameter so no change made to actual parameter.



Using call by reference: In call by reference address of actual parameter are passed to call function and any operation performed on it reflect to actual parameter.



1.51 (c)



In dynamic scoping undeclared variable referred to declared variable in block from which call is made i.e. here call f( ). But in static scooping undeclared variable always take value from global variable.
So static scooping x = 10
Dynamic scooping x = 20
Note: Global variable by default initialize to 0.

1.52 (a)

a = (x ≤ y) ? ((x ≤ z) ? x : z) : ((y > z) ? y : z)
In C "? : " is a ternary operator the syntax is (exp1 ? exp2 : exp3). It means if exp1 is true print exp2 else exp3.

Let x = 3, y = 4, z = 2.
⇒ a = (3 > 4) ? ((3 > 2) ? 3 : 2) :
((4 > 2) ? 4 : 2)
a = (3 > 4) ? 3 : 4 = 4

1.53 (b)

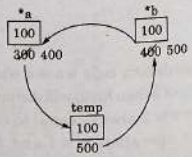
char a[6] =

W	o	r	l	d	\0
a[0]	1	2	3	4	5

When Loop will execute first time,
a[0] = a[5]
a[0] = '\0'
printf("%s", a), the string starting at address a prints the string starting with '/0' and it indicates the end of string, so it will print null string.

1.54 (c)

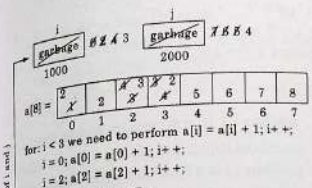
Num 1 = 5 num 1 = 4
1st condition fail
2nd condition fail
3rd condition execute because
(num1> = num2)



So value of a = 4
value of b = 5
i.e., swapping done via call by reference.

1.55 (c)

Inside main function:



Note: When any variable not declared inside any block then it can referred to outer block for same variable.

1.56 (b)

char [2] [3] = {{'a', 'b', 'c', } {'d', 'e', 'f' }}

a	0	1	2
0	1000 a	1001 b	1002 c
1	1003 d	1004 e	1005 f

char*p = *b make new char pointer p which point to array b.

p	0	1
0	3000 garbage	3001 garbage
1	3002 garbage	3003 garbage
2	3004 garbage	3005 garbage

*(p + 2 * j + i) = a[i] [j]; will assign element of array a to array b, by scanning array 'd' row wise and sorting character in array 'b' column wise.

When i = 0 : i < 2 (true)
j = 0; j < 3 (condition true)
*(3000 + 2*0 + 0) = a[0][0];
*(3000) = a;
j++ ;
j = 1; j < 3 (condition true)
*(3000 + 2*1 + 0) = a[0][1];
*(3002) = b;
j++;
j = 2; j < 3 (condition true)
*(3000 + 2*2 + 0) = a[0][2];
*(3004) = c;
j++;
j = 3, j < 3 (condition false)
i++;

When i = 1 : i < 2 (true)
j = 0; j < 3 (condition true)
*(3000 + 2*0 + 1) = a[1][0];
*(3001) = d;
j++;
j = 1; j < 3 (condition true)
*(3000 + 2*1 + 1) = a[1][1];
*(3003) = e;
j++;
j = 2; j < 3 (condition true)
*(3000 + 2*2 + 1) = a[1][2];
*(3005) = f;
j++;
j = 3; j < 3 (condition false)
i++;

Final array b[3] [2] will be:

b	0	1
0	3000 a	3001 d
1	3002 b	3003 e
2	3004 c	3005 f

1.57 (d)

10101101 i.e Option (d)
since recusive funtion calls will be

1st function call $\frac{173}{2}$ = 86(int part only)

2nd function call $\frac{86}{2}$ = 43

3rd function call $\frac{43}{2}$ = 21(int part only)

4th function call $\frac{21}{2}$ = 10(int part only),

5th function call $\frac{10}{2}$ = 5,

6^{th} function call $\frac{5}{2} = 2$ (integer part only),

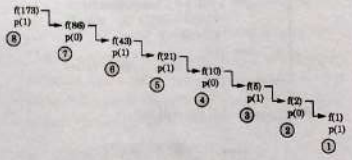
7^{th} function call $\frac{2}{2} = 1$,

In 7^{th} function call condition if(n <=1) will become true so n will be printed (i.e 1 will be printed) now while returning every function will execute its remaining part of code i.e. printf ("%d", n%2);

So 6th function will print 2 mod 2 = 0,
5th function will print 5 mod 2 = 1,
4th function will print 10 mod 2 = 0,
3rd function will print 21 mod 2 = 1,
2nd function will print 43 mod 2 = 1,
1st function will print 86 mod 2 = 0,
Function call f(173) will print
173 mod 2 = 1

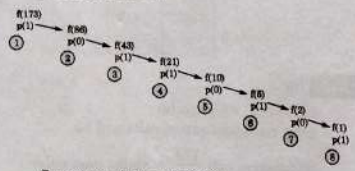
1.58 (c)

Execution of P1:



So output will be 10101101.

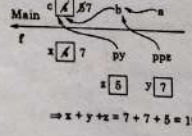
Execution of P2:



So output will be 10110101.

1.59 (b)

f(c, b, a) is called by the main( ) function the graphical execution of the program is given below.



⇒ x + y + z = 7 + 7 + 5 = 19

```
int y, z;
**ppz = **ppz + 1;
z = **ppz = 5;
*py = *py + 2;
y = *py = 7
x = 4 + 3 = 7
return x + y + z = 19
```

1.60 (d)

- getchar ( ) is a function which read from standard input and return integer value.
- putchar ( ) is a function which used to write a character on standard output/screen.

Here if condition need to check input is finished or not by doing ((c = getchar ( ))! = '\n') i.e. when character = '\n', '\n' represent end of input.
After reverse ( ) function we need to print the output by using putchar (c).

1.61 (b)

```
x = 15
fun (5, &x)   let &x = 100
↓
fun (5, 100)
↓              t = 5,    f = 5 + 3, *f_p = 5
               return 8
fun (4, 100)
↓              t = 3,    f = 3 + 2, *f_p = 3
               return 5
fun (3, 100)
↓              t = 2,    f = 2 + 1, *f_p = 2
               return 3
fun (2, 100)
↓              t = 1,    f = 1 + –1, *f_p = 1
               return 2
fun (1, 100)
               *f_p = 1
               return 1
```

So fun(5, & x) will **return 8** and it will be printed.

1.62 (d)

In the given program it begin from main function. i and j are globally initialized by 0 and 1. So when we call function f(& i, & j) the address of i and j are passed.
when p = q and *p = 2 means
*q = 2, so value of *q is passed.
and value of *q return to j.
Value of i and j are 0 and 2 respected.
So printf("% d %d", i, j) gives output (0, 2)

1.63 (c)

a[ ] =

12	7	13	4	11	6

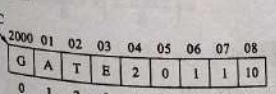
So f(a, 6) → first this call
Thus is the number stored in the array and we can access these by using

```
*a – f(a, 6)
↓12
So12 + f(a + 1, n – 1)          {even}
       ↓
     7 – f(a + 1, n – 1)        {odd}
         ↓
       13 – f(a + 1, n – 1)     {odd}
           ↓
         4 + f(a + 1, n – 1)    {even}
             ↓
           11 – f(a + 1, n – 1) {odd}
               ↓
             6 + f(a + 1, 0)    {even}
                 ↓
                 0
= 12 + [7 – [13 – [4 + [11 –(6 + 0)]]]]
= 12 + [7 – [13 – [4 + [11 – 6]]]
= 12 + [7 – [13 – [4 + 5]]] = 12 + (7 – (13 – 9))
= 12 + (7 – 4) = 12 + 3 = 15
```

1.64 (d)

In given program we take the test cases and apply
From T_1 if all values are equal means a = b = c = d, a = b condition satisfied. S_1 and S_4 executed.
From T_2 when all a, b, c, d distinct:
S_1, S_2 not execute, S_3 executes.
From T_3 when a = b, S_1 execute but c = d.
S_2 will not execute but S_3 and S_4 executes.
Here no need of T_3 because we get all result from above two.
From T_4 if a! = b and c = d, S_1 not executes and S_2 and S_4 executes.
All of S_1, S_2, S_3, S_4 will execute and covered by T_1, T_2 and T_4.

1.65 (c)



P + P[3] – P[1] = 2000 + 69 – 65 = 2004

⇒ 2011 string will be printed.
(From 2004 Address)

1.66 (b)

Consider the following recursive C function foo (n, r)

```
{
    if (n > 0) return (n% r) + foo (n/r, r);
    else return 0;
}
```

Value return by the function foo (345, 10):

```
foo (345, 10)
      ⇓
5 + foo (34, 10)
      ⇓
4 + foo (3, 10)
      ⇓
3 + foo (0, 10)
      ⇓
      0
```

The return value is 5 + 4 + 3 = 12

1.67 (d)

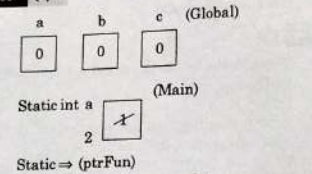
```
foo (513, 2)                          = 1 + 1 = 2
  ⇓
1 + foo (256, 2)                      = 0 + 1 = 1
    ⇓
  0 + foo (128, 2)                    = 0 + 1 = 1
      ⇓
    0 + foo (64, 2)                   = 0 + 1 = 1
        ⇓
      0 + foo (32, 2)                 = 0 + 1 = 1
          ⇓
        0 + foo (16, 2)               = 0 + 1 = 1
            ⇓
          0 + foo (8, 2)              = 0 + 1 = 1
              ⇓
            0 + foo (4, 2)            = 0 + 1 = 1
                ⇓
              0 + foo (2, 2)          = 0 + 1 = 1
                  ⇓
                0 + foo (1, 2)        = 0 + 1 = 1
                    ⇓
                  1 + foo (0, 2)      = 1 + 0 = 1
                      ⇓
                      0
```

foo(513, 2) = 2

1.68 (c)

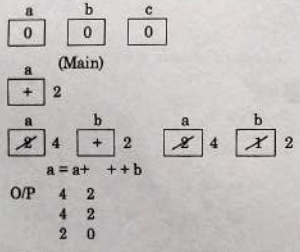
There is no break statement present in the code. Hence the switch case will jump to case : A and execute every statement from there.
∴ Choice A
Choice B No choice.

1.69 (c)



∴ Option 'C' is correct.

1.70 (d)



1.71 (b)

Value of x is passed by reference so every time changes are reflected to original value

f(5, 5)
↓
x * f(6 * 4) ⇒ 9^4
↓
x * f(7 * 3) ⇒ 9^3
↓
x * f(8, 2) ⇒ 9^2
↓
x * f(9, 1) ⇒ 9
↓
1

when c = 1, x = 9
So x * x * x * x = $x^4 = 9^4 = 6561$
Option (b) is correct.

1.72 (d)

pi = &i; pi → [ ] i

scanf("%d", pi); pi → [10] i

printf("%d\n", i + 5); pi → [15] i

The value entered is 10. It prints 15.

1.73 (c)

$$\sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=j+1}^{n} 1 = \frac{1}{6}n\times(n+1)(n+2)$$

1.74 (9)

In fun( ), num >> = 1 will be evaluated as num = num >> 1 i.e. left shift 1 bit of bit representation of num or num divide by 2.

Binary representation of 435 = 110110011 here while loop will terminate when all bit became 0 i.e. left shift 9 time, so bit now becomes 000000000. Hence count = 9.

1.75 (b)

While calculating $^nC_3 = \frac{n(n-1)(n-2)}{6}$ i.e. multiplying n × (n − 1) × (n − 2) then divide by 6. Since n value can be large such that n × (n − 1) × (n − 2) will generate value which cannot be representable by bits assigned to unsigned int.

So, idea is to calculate $^nC_3$ as $\left(\frac{n(n-1)}{2}\right)\frac{(n-2)}{3}$

Since n(n − 1) always gives an even number greater than 0, so always divisible by 2 but if n(n − 1) divide by 3 first then it may be possible that answer comes in floating point, since we use integer variable so part after decimal point need to be discarded i.e. will not get exact answer as $^nC_3$. So most appropriate choice is

$$= \left(\frac{n(n-1)}{2}\right)\times\frac{(n-2)}{3}$$

1.76 (1.73)

$\left|x^2 - 3\right| < 0.01$

if $x = \sqrt{3}$, the given program returns x.

Therefore, $x = \sqrt{3} = 1.732$

For $q = 1.73 \Rightarrow f(q)$ returns q.

1.77 (d)

$f(50)$ goes into infinite loop by calling $f(i)$ recursively for $i = j = 50$.

1.78 (–5)

Here both f_1 and f_2 swaps the two arguments. f_1 is call by value but f_2 is called by reference hence changes made by only f_2 are reflected in main function.

After f_2 is called, the values of a, b and c are 4, 6 and 5 respectively.

Hence the output is 5 − 4 − 6 = –5

1.79 (b)

```
q = 0
r = x
while (r ≥ y) do
    begin
        r = r − y
        q = q + 1
    end
```

Post condition: [after program termination]

r < y and r = r − qy

⇒ r = x − qy

⇒ x = r + qy

∴ Post condition is: (r < y) ∧ (x = qy + r)

1.80 (51)

Initially compute fun(1), fun(2), fun(3) and fun(4)

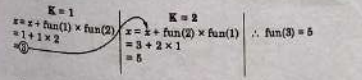
(i) fun(1) = 1

(ii) fun(2)

k = 1: x = x + fun(1) × fun(1) = 1 + 1 × 1 = 2

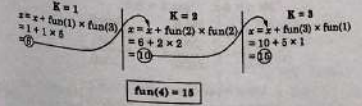
∴ fun(2) = 2

(iii) fun(3):

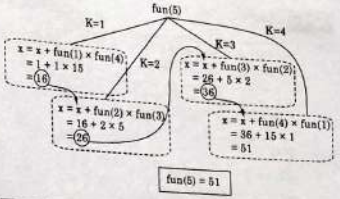


∴ fun(3) = 5

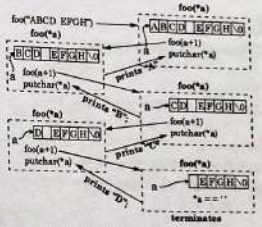
(iv) fun(4):



Compute now fun(5)



1.81 (d)



∴ Output is DCBA

1.82 (5)

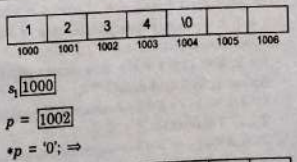
The minimum number of movements are 5. The following sequences will give minimum movements, after 1 moving.

Seq 1: Move 2 left, move 5 left, move 6 up, move 18 up, move 25 left.

Seq 2: Move 2 left, move 4 up, move 6 left, move 18 up, move 25 left.

2	4	5	14
3	6	18	23
10	12	25	∞
31	∞	∞	∞

1.83 (c)



Therefore the output is 1204

1.84 (140)

ptr ++ ; ⇒ *ptr* is $p + 1$.
$ptr - p = (p+1) - p = 1$
$**ptr = *(*(p + 1)) = *(a + 3) = 40$
∴ Output is 1, 40 which is **140**.

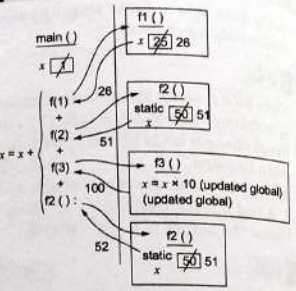
1.85 (b)

Number of times invoked
get (–1) = 1 (itself)
get (0) = 1
get (1) = get (0) + get (–2) + 1
= 1 + 1 + 1 = 3
Ingeneral,
get (*n*) = get (*n*–1)+get (*n*–3) + 1
get (2) = get (1) + get (–1) + 1
= 3 + 1 + 1
= 5
get (3) = get (2) + get (0) + 1
= 5 + 1 + 1
= 7
get (4) = get (3) + get (2) + 1
= 7 + 3 + 1
= 11
get (5) = get (4) + get (2) + 1
= 11 + 5 + 1
= 17
get (6) = get (5) + get (3) + 1
= 17 + 7 + 1 = 25
∴ Number of times get function invoked on get (6) = 25

1.86 (10)

$i = 0$, $k = -1$ only default executes $-i + k = -1$ ⇒ executes 1 time.
$i = 1$, $k = -1$ only default executes $-i + k = 0$ ⇒ executes 1 time.
$i = 2$, $k = -1 - i + k = 1$ ⇒ executes 3 times (cases 1, 2, 3 and default)
$i = 3$, $k = -1 - i + k = 2$ ⇒ executes 3 times (cases 2,3 and default)
$i = 4$, $k = -1 - i + k = 3$ ⇒ executes 2 times (cases 3 and default)
∴ Total 10 times printf is executed.

1.87 (230)

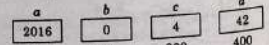


$x = 1 + 26 + 51 + 100 + 52 = 230$

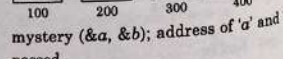
1.88 (d)

f(*s*, **s*) since there is no pointer variable define with **s*. So this function will give type checking error.
i = *f*(*i*, s) since function prototype is void in void *f* (int, short) i.e., *f* is accepting argument int type and short type and its return type should be integer because we store the output of function in variable *i* which is integer type. So there must be type casting into integer type but type casting is not present. So this function will give type checking error.
f(*i*, **s*) since there is no pointer variable define with **s*. So this function will give type checking error.
f(*i*, **p*) in this function to argument are pass one is int type and another is short type defined in the main function. So this function will not give type checking error.

1.89 (2016)



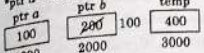
1. mystery (&*a*, &*b*); address of '*a*' and '*b*' is passed.



1. temp = ptr *b*; ⇒ temp = 200
2. ptr *b* = ptr *a*; ⇒ ptr *b* = 100
3. ptr *a* = temp; ⇒ ptr *a* = 200

} No effect on variable '*a*' and '*b*'

2. if (2016 < 4) false
So, mystery (&*a*, &*b*);



1. temp = ptr *b*; ⇒ temp = 400
2. ptr *b* = ptr *a*; ⇒ ptr *b* = 100
3. ptr *a* = temp; ⇒ ptr *a* = 400

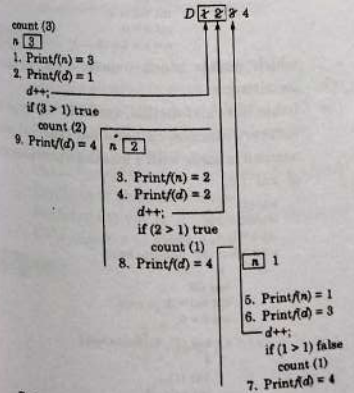
} No effect on variable '*a*' & '*d*'

Printf("*a*"); ⇒ 2016
So, output of above program is 2016.

1.90 (d)

Option (a) is fail for descending array. Since *n* value is not decreament in program.
Option (b) is fail for ascending array. Since value of '*a*' is increment every time and in while condition we check for value of '*b*'. So it is incorrect.
Option (c) is fail for input 5 6 7 1 2 3. Since pointer a is pointing to 6 and '*b*' will pointing to 7 i.e., '*a*' and '*b*' are adjacent to each other. Then while condition *b* > (*a* + 1) is fail and it return 6 is maximum value which is incorrect.
Option (d) at the end of the program, pointer '*a*' and '*b*' pointed to same element then only while condition is false and return the maximum value of array.

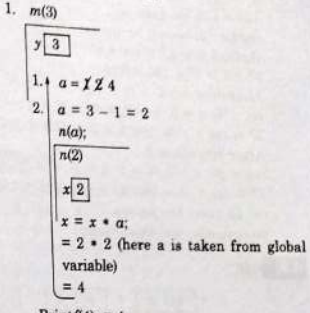
1.91 (a)



So output will be printed in serial order given in solution i.e., 312213444.

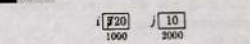
1.92 (d)

a 1 globaly initialize.
1. *m*(3)



Printf(4); = 4
3. Printf(*a*) = 4 since dyanmic scoping is used.
So, take value of inner variable 'a'.
So answer will be 4, 4.

1.93 (30)



1. $m = m + 5$; ∵ $m = 15$
2. $*P = *P + m$; ∵ $*P = 20$
3. Printf(*i* + *j*); ∵ 20 + 10 = 30
So value printed by above code is 30.

1.94 (c)

Take random value of X and Y i.e., X = 5 and Y = 3.
Initially X = 5, Y = 3, res = 1, a =X and b =Y.
Option (a): $X^Y = a^b$
$X^Y = a^b = 5^3 = 5^3 = 125 = 125$
After iteration 1
res = 5; a = 5; b = 2; X = 5; Y = 3
$X^Y = a^b = 5^3 \neq 5^2 = 125 \neq 25$
So, case fail. Option (a) cannot be answer.
Option (b): $(res * a)^Y = (res * X)^b$
$(1 \times 5)^3 = (1 \times 5)^3 = 125 = 125$
After iteration 1
res = 5; a = 5; b = 2; X = 5; Y = 3
$(res * a)^Y = (res * X)^b = (5 \times 5)^3 = (5 \times 5)^2$
$15625 \neq 625$ So, case fail. Option (b) cannot be answer.
Option (d): $X^Y = (res * a)^b$
$5^3 = (1 \times 5)^3 = 125 = 125$

After iteration 1
res = 5; a = 5; b = 2; X = 5; Y = 3
$X^Y = (res * a)^b \equiv 5^3 = (5 \times 5)^2$
$125 \neq 625$ So, case fail.
Option (d) cannot be answer.
Option (c): $X^Y = res * a^b$
$5^3 = 1 \times 5^3 \equiv 125 = 125$
After iteration 1
res = 5; a = 5; b = 2; X = 5; Y = 3
$X^Y = res * a^b \equiv 5^3 = 5 \times 5^2 \equiv 125 = 125$
After iteration 2
res = 25; a = 5; b = 1; X = 5; Y = 3
$X^Y = res * a^b \equiv 5^3 = 25 \times 5^1 \equiv 125 = 125$
So, all cases are passes.
So option (c) will the answer.

1.95 **(3)**

a	3	5	2	6	4

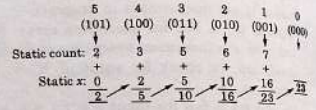
1. *f*(*a*, 5) is a function contain 2 parameter one contain starting address of array and second parameter tell number of element in the array.
2. Every time 'n' value compare with 1 when it is less than equal to 1 return 0 and stop the program otherwise continue with recursive function call.

1. *f*(*a*, 5)
   *P = *a*; *P* pointed to same address pointed by '*a*'.
   n = 5; *n* value greater than 1.
   So, max (*f*(*P* + 1, 5 – 1), 3 – 5); or
   max (*f*(*P* + 1, 4) – 2);
2. *f*(*P* + 1, 4)
   *P = *P* + 1; P is pointed to next element of array i.e., 5.
   n = 4; *n* value greater than 1.
   So, max (max(*f*(*P* + 1, 4 – 1), 5 – 2), –2) or max (max(*f*(*P* + 1) 3), 3), –2)
3. *f*(*P* + 1, 3)
   *P = *P* + 1; P is pointed to next element of array i.e., 2.
   n = 3; *n* value greater than 1.
   So, max(max(max(*f*(*P* + 1, 3 – 1), 2 – 6), 3) –2) or max(max(max(*f*(*P*+1, 2), – 4), 3) –2);
4. *f*(*P* + 1, 2);
   *P = *P*+1; *P* is pointed to next element of array i.e., 6.
   n = 2; *n* value greater than 1.
   So, max(max(max(max(*f*(*P*+ 1, 2 – 1), 2),–4), 3), –2) or
   max(max(max(max(*f*(*P*+1, 1), 2), –4), 3),–2)
5. *f*(*P* + 1, 1);
   *P = *P*+1; *P* is pointed to next element of array i.e., 4.
   n = 1; *n* value equal to 1 so, return 0.
   So max(max(max(max(0, 2), –4), 3), –2)
   max(max(max(2, –4), 3), –2)
   max(max(2, 3), –2)
   max(3, –2) = 3.

So the value printed by given code is 3.

1.96 **(23)**

Here total ( ) function will return number of bits are set i.e., 1 binary representation of value of variable 'v'.
The output of above program is:



1.97 **(c)**

In foo (int var) function, variable 'val' is created for every function call and foo (val –) means first call is made after that 'val' values is decremented by 1 i.e.,

```
foo (3)
int val = 3-2;
int x = 0;
x = x + foo (3 –);
        ↓
        foo (3)
        int val = 3;
        int x = 0;
        x = x + foo (3 –);
```

which makes stack overflow, so abnormal termination.
In bar (int val) function, variable 'val' is created for every function call and bar (val – 1) means new call is made with 1 less than previous value of 'val' i.e.,

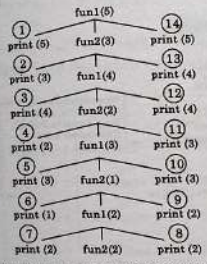
```
bar (3)
int val = 3-2;
int x = 0;
x = x + bar (2); {infinite loop}
        ↓
        bar (2)
        int val = 3;
        int x = 0;
x = x + bar (1); {infinite loop}
        ↓
        bar (1)
        int val = 3;
        int x = 0;
        x = x + bar (0); terminate
```

If we see while (var > 0) is make infinite loop since 'val' value is not decremented in bar( ) function for value of variable val > 0, it created infinite loop.

1.98 **(3)**

Strlen (s) = 3 and Strlen (t) = 5
Value of c is 0
[Strlen (s) – Strlen (t) > 0]: Strlen (s) : Strlen (t);
[Strlen (s) – Strlen (t)] = –2
Since, we are comparing with a unsigned integer.
So, [Strlen (s) – Strlen (t)] = [–2] = 2.
2 > 0 : Strlen (s) : Strlen (t); since 2 > 0 will evaluate True so it will give output as Strlen (s) = 3.
Hence '3' will be printed.

1.99 **(a)**

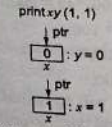


Hence the output is : 53423122233445

1.100 **(a)**

- **static char var;** : Initialization of a variable located in data section of memory.
- ***m* = malloc(10); *m* = NULL;** : A lost memory which can't be freed because free (*m*) is missed in code.
- **char *ptr[10];** : Sequence of memory locations to store addresses.
- **register int var1;** : Request to allocate a CPU register to store data.

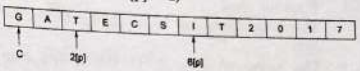
1.101 **(c)**



Hence the output will be (1, 0).

1.102 **(2)**

(C + 2[p] – 6[p] – 1)



C + 'T' – 'T' – 1
= C + 11 – 1
= C + 10
Hence printf will print '2'.

1.103 **(0)**

```
1. int m = 10;   // m = 10
2. int n, n1;
3. n = ++m;      // n = 11
4. n1 = m++;     // n1 = 11, m = 12
5. n – –;        // n = 10
6. – – n1;       // n1 = 10
7. n – = n1;     // n = 0
8. printf("%d", n);
```

The output will be 0.

1.104 **(3)**

3	5	1	4	6	2
0	1	2	3	4	5

First for loop:
(i = 0) 5 3 1 4 6 2
(i = 1) 5 3 1 4 6 2
(i = 2) 5 3 4 1 6 2
(i = 3) 5 3 4 6 1 2
(i = 4) 5 3 4 6 2 1
Second for loop:
(i = 5) 5 3 4 6 2 1
(i = 4) 5 3 4 6 2 1
(i = 3) 5 3 6 4 2 1
(i = 2) 5 6 3 4 2 1
(i = 1) 6 5 3 4 2 1
Now since done is '0', hence the for loops will execute again.
First for loop:
(i = 0) 6 5 3 4 2 1
(i = 1) 6 5 3 4 2 1
(i = 2) 6 5 4 3 2 1
(i = 3) 6 5 4 3 2 1
(i = 4) 6 5 4 3 2 1
Second for loop:
(i = 5) 6 5 4 3 2 1
(i = 4) 6 5 4 3 2 1
(i = 3) 6 5 4 3 2 1
(i = 2) 6 5 4 3 2 1
(i = 1) 6 5 4 3 2 1

Value of done is still '0', hence the for loop will execute again.
First for loop:
This time there will be no change by the for loop.
The value of done is '1'. Hence the loop terminates as

6	5	4	3	2	1
0	1	2	3	4	5

The output of the program will be '3'.

1.105 (d)

In **main( )** function, **malloc( )** function return a void type address. So no need of type casting i.e., **(int*)** malloc (size of (int));, inplace of type casting malloc (size of (int)); will not give compilation error. **Assignval( )** function assign 0 in memory location pointed by 'x' and return address of memory location pointed by 'x'. Since next **malloc( )** function can take memory location assign by previous **malloc( )** function because there is no **free(m)** is given to freed previously assign memory location.
So this can give memory leak problem if variable 'x' non-zero value.

1.106 (c)

Here $x == (y \times q + r)$
q = Quotient and r = Remainder
In given code indirectly we are dividing 'r' by 'y' in form of substraction and 'q' count the number of time number gets divide successfully before reaching $r \geq y$.
So, initial condition will be $q = 0$ i.e., $r = x$ and final condition $(q == 0)$ && $(r == x)$ && $(y > 0)$.

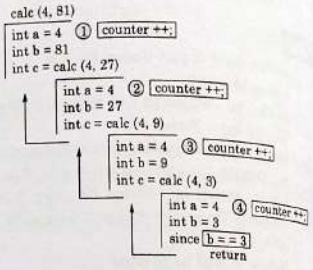
1.107 (a)

	100	101	102
Struct p =	1	0	c

Since we know 'a' + 2 = skip 'a', 'b', i.e. 'c'
Now *q = &p
i.e. now q point to struct p.
So, *((char*) q + 1)) will typecast structure into char and skip one char and points to '0' i.e. addresses '101' and *(101) = 0.
And *((char*) q + 2) will typecast structure into char and skip two char and points to 'c' i.e. addresses '102' and *(102) = c.
Final output by given code is 0, c.

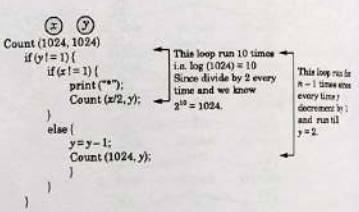
1.108 (4)

Counter = [ ] 4
1000



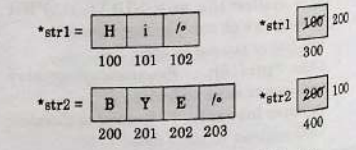
So, Counter = 4
Hence 4 will be printed.

1.109 (10230)



So, inner if loop will run 10 times and y run for 1023 times, which make $x = 1024$ for every y.
So count (1024, 1024) = 10 × 1023 = 10230.

1.110 (a)



1. fun1(100, 200) **"Call By Value"**
   *s1 = 100; *s2 = 200;
   *temp = 100;
   *s1 = 200;
   *s2 = 100;
   Since, fun1 is call by value. So no change in original *str1 and *str2, so printf will print Hi Bye

2. fun2(&str1, &str2) **"Call By Refference"**
   **s1 = 300; **s2 = 400;
   *temp = 300;
   **s1 = 400;
   **s2 = 300;
   Since, fun2 is call by refference. So it will swap the value of str1 and str2. So, str1 points to "Bye" and str2 points to "Hi". So printf will point Bye Hi.
   Final output: Hi Bye Bye Hi

1.111 (b)

[ø40] j [ø5] sum
100 200

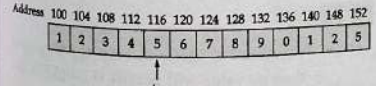
First for loop deal with value of 'j':
1. for ($i = n$; $i > 1$, $i = i/2$)
   j++;
   When $n = 2^{40}$ will make $j = \log(2^{40}) = 40$.
Second for loop deal with value of 'sum':

1	j = 40	Since j > 1	So sum++ = 1	Then j = 40/2 = 20
2	j = 20	Since j > 1	So sum++ = 2	Then j = 20/2 = 10
3	j = 10	Since j > 1	So sum++ = 3	Then j = 10/2 = 5
4	j = 5	Since j > 1	So sum++ = 4	Then j = 5/2 = 2
5	j = 2	Since j > 1	So sum++ = 5	Then j = 2/2 = 1

Now for $j = 1$ loop will terminate and return sum = 5.

1.112 (6)

Let size of integer is 4 byte.
arr[ ] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 5}



ip = arr + 4
ip[1] = *(ip + 1)
= *(starting address of ip + 1 * size of integer)
= *(116 + 1 × 4)
= *(120) = 6
6 will be displayed on execution.

1.113 (26)

int $x = 2$, $y = 5$
Function jumble is call by value
y = jumble (y, x)
In function jumble
$x = 2 \times x + y$
$x = 2 \times 5 + 2$
$x = 12$
It will return 12 and y will hold the return value.
Now value of $x = 2$, $y = 12$
x = jumble (y, x)
= jumble (12, 2)
Function return 26 and x hold the return value x will be printed so 26 is printed by the program.

1.115 (5)

Initially $i = 2.0$, $j = 1.0$

while $\left(\frac{i}{j} > 0.0625\right)$

1. $\frac{i}{j} = \frac{2}{1} > 0.0625$
   $j = j + j = 1 + 1 = 2$
   First time sum will be printed.
2. $\frac{i}{j} = \frac{2}{2} > 0.0625$
   $j = j + j = 2 + 2 = 4$
   Second time sum will be printed.
3. $\frac{i}{j} = \frac{2}{4} > 0.0625$
   $j = j + j = 4 + 4 = 8$
   Third time sum will be printed.
4. $\frac{i}{j} = \frac{2}{8} > 0.0625$
   $j = j + j = 8 + 8 = 16$
   Fourth time sum will be printed.
5. $\frac{i}{j} = \frac{2}{16} > 0.0625$
   $j = j + j = 16 + 16 = 32$
   Fifth time sum will be printed.
6. $\frac{i}{j} = \frac{2}{32} = 0.0625$
   So while condition become false and break.

Total 5 time sum will be printed.

1.116 (d)

```
for (statement 1; statement 2; statement 3)
{
    // code block to be executed
}
```

- **Statement 1** is executed (one time) before the execution of the code block for initialization.
- **Statement 2** defines the condition and executed (every time) before executing the code block.
- **Statement 3** is executed (every time) after the code block has been executed.

Static initialization num

num = 7 6 5 4 3 2 1 0 –1

1st iteration of for loop

- **Statement 1:** r( ) ⇒ return 7 and post decrement num to 6
- **Statement 2:** r( ) ⇒ return 6 ⇒ (condition true) and post decrement num to 5

  printf("%d", r( )) ⇒ return 5 and post decrement num to 4 and **print 5**

2nd iteration

- **Statement 3:** r( ) ⇒ return 4 and post decrement num to 3
- **Statement 2:** r( ) ⇒ return 3 ⇒ (condition true) and post decrement num to 2

  printf("%d", r( )) ⇒ return 2 and post decrement num to 1 and **print 2**

3rd iteration

- **Statement 3:** r( ) ⇒ return 1 and post decrement num to 0
- **Statement 2:** r( ) ⇒ return 0 ⇒ (Condition false) ⇒ Exit loop and post decrement num to –1

Output on the screen : 52

1.117 (10)

a[ ] = {2, 4, 6, 8, 10}

sum = 0, *b = a + 4

	100	104	108	112	116
a	2	4	6	8	10
					↑ b

- **1st iteration**

  $i = 0$

  sum = sum + (*b – i) – * (b – i)

  = 0 + (10 – 0) – 10 = 0
- **2nd iteration**

  $i = 1$

  sum = 0 + (10 – 1) – (8) = 1
- **3rd iteration**

  $i = 2$

  sum = 1 + (10 – 2) – 6 = 3
- **4th iteration**

  $i = 3$

  sum = 3 + (10 – 3) – 4 = 6
- **5th iteration**

  $i = 4$

  sum = 6 + (10 – 4) – 2 = 10

Output of the program is 10.

1.118 (81)

pp(3, 4) a = 3, b = 4, tot = 1

len = length (4, array) ⇒ i will return 3 with array set as [001] (array updated 1 only when b%2! = 0)

Now pp will run loop 3 times:

1. Since arr[0] = 0.

   So ex = 3 × 3 = 9
2. Since arr[1] = 0.

   So ex = 9 × 9 = 81
3. Since arr[1] = 1.

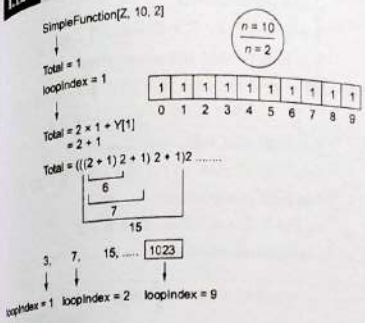
   So tot = 1 × 81 = 81

After executing the given C program integer 81 is returned.

1.119 (55)

- Variable i is static in both the functions thus its value will persist throughout the program.
- When fun2 (5) gets called, it will first call fun1 with value of n as 5 and then fun2 will gets called with $n = 4$.
- Similarly, same steps are taken for all the upcoming values of n.

1.120 (1023)



1.121 (d)

$j = -3$, $j = -2$, $j = -1$

⇒ Short circuiting in if((j >= 0) && (i++))

$j = 0$; $i = 0$ used but i is 1

$j = 1$; $i = 1$ used but i is 2

count = 0 + 1 = 1

$j = 2$; $i = 2$ used but i is 3

count = 1 + 2

$j = 3$; $i = 3$ used but i is 4

count = 3 + 3 = 6

count = 6 + 4 = 10]

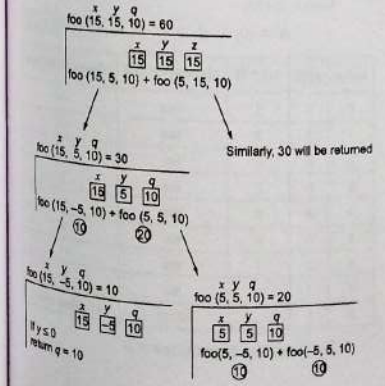
corresponding to count = count + i

Option (d) is correct.

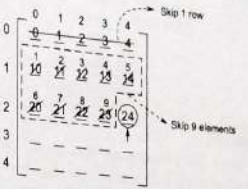
1.122 (15)

This function will keep on subtracting till both x and y becomes equal that is 15.

1.123 (24)



1.124 (a)



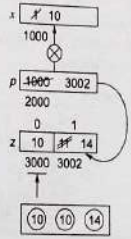
int a[4][5]

[*(a + 1) + 9]

It means skip one row and then skip 9 elements.

So, the resultant value is 24.

1.125 (d)



1.126 (a)

int a[3] [3] [3] = {{1, 2, 3, 4, 5, 6, 7, 8, 9},

{10, 11, 12, 13, 14, 15, 16, 17, 18},

2D 1D 3 {19, 20, 21, 22, 23, 24, 25, 26, 27}}

```
int i = 0, j = 0; k = 0;
for (i = 0; i < 3; i ++)
for (k = 0; k < 3, k ++)
printf(a[i][j][k]);
```

0 0 0	0 0 1	0 0 2	1 0 0	1 0 1
i j k	i j k	i j k	i j k	i j k

and so on.

= 1	2	3
10	11	12
19	20	21

This is the answer.

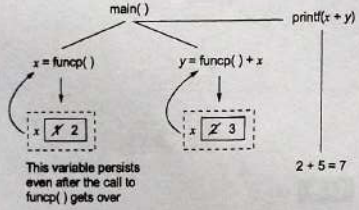
1.127 (a)

$a \rightarrow 80 = 1010000$
$b \rightarrow 120 = 1111000$
$\& \ 1010000 = 80$
$or \ 1111000 = 120$
$\wedge \ 0101000 = 83$

1.128 (7)

While the programme is running, memory is still occupied by a static int variable. When a function call where the variable was declared is finished, a normal or auto variable is destroyed. Memory for static variables is allotted in the data segment rather than the stack segment.

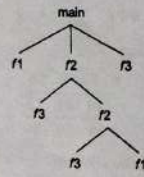
Note: While the scope of the variables x and y declared in main( ) only applies to the function, the scope of static x applies to the entire programme unless a local variable is defined explicitly. Additionally, once a static variable is defined, it cannot be removed before the programme is finished.



1.129 (a)

The way control enters and exits activations is displayed using an activation tree. Aspects of activation trees include:

(a) Each node shows how a procedure has been activated.
(b) The root displays the main function as being active.
(c) Only when control flows from process x to procedure y is the node for procedure "x" the parent of node for procedure "y".



1.132 (6)

Given expression is

$3 + 1 + 5 * 2 / 7 + 2 - 4 - 7 - 6 / 2$

+ is highest and left associative

$((3 + 1) + 5) * 2 / (7 + 2) - 4 - 7 - 6 / 2$

$= 9 * 2 / 9 - 4 - 7 - 6 / 2$

– is high and right associative

$\Rightarrow 9 * 2 / (9 - (4 - (7 - 6))) / 2 = 9 * 2 / 6 / 2$

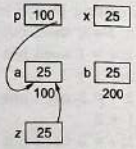
* is high precedence

$\Rightarrow (9 * 2) / 6 / 2 = 18 / 6 / 2$

/ is right associative

$\Rightarrow (18 / (6 / 2)) = \frac{18}{3} = 6$

1.138 (25)



O/P = 25

1.139 (5)

The function foo(s, size) counts distinct consecutive elements in the array by checking if s[0] ≠ s[1]. If true, it increments the count and moves to the next element otherwise, it skips duplicates.

Given array:

$A = \{0, 1, 2, 2, 2, 0, 0, 1, 1\}$

Index	s[1]	s[1 + 1]	s[1] ≠ s[1 + 1] ?	Count Increment
0	0	1	Yes	+1
1	1	2	Yes	+1
2	2	2	No	0
3	2	2	No	0
4	2	0	Yes	+1
5	0	0	No	0
6	0	1	Yes	+1
7	1	1	No	0
8	1	–	(End)	+1 (Base case)

Total unique transitions = 5

1.140 (46)

newnum 0 turn 0 t ~~1~~ 10000 n 14362

We process each digit from left to right, updating all variables in each iteration.

Iteration	d = n / t	n = n % t	t /= 10	turn (before)	newnum (before)	newnum (updated if turn == 1)	turn (after)
1	1 (14362 / 10000)	4362	1000	0	0	–	1
2	4 (4362 / 1000)	362	100	1	0	4 (10 × 0 + 4)	0
3	3 (362 / 100)	62	10	0	4	–	1
4	6 (62 / 10)	2	1	1	4	46 (10 × 4 + 6)	0
5	2 (2 / 1)	0	0	0	46	–	–

1.141 (21)

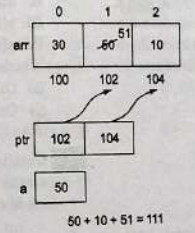
x ~~126~~ 21 y ~~105~~ ~~84 63 42 21~~

When (X = = Y)

(21 = = 21) loop will be stops.

So, GCD of 126 and 105 is 21.

1.142 (111)



50 + 10 + 51 = 111

1.143 (46)