Question : 1

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

Here c is an integer variable and b is a pointer variable pointing to other integer variable(here it is pointing to c). “a” is another variable pointing to variable b.

Hence “c” stores an integer value, “b” stores address of “c” and “a” stores address of “b”.
These values of the variable are then passed through the function f.
Variable x, py and ppz are containing values of variable c,b and a respectively.
Variables y and z are declared in it.
**ppz is then incremented by 1, hence *ppz is 5 now and then assigned to z(so z = 5).

*py is also 5 now as **ppz is also pointing to same integer location as *py, and then *py incremented by 2, hence *py becomes 7. This value then assigned to y.(so y = 7)
X is incremented by 3 hence x = 7 .
In the end x+y+z is returned which in this case are 7+7+5 = 19.

Question 2 :

```
int a[] = {10,20,30,40,50};
int *p=a;
Int **ptr = &p;
(*ptr)++;
(*(*ptr))++;
printf("%d",*p);
```

In this code p is a pointer variable pointing to the first element of the array. And the variable “ptr” is pointing to the address of the variable p.
The line (*ptr)++ wil increment the value of the location to which ptr is pointing. In this case it is the location of the first element of the array and on incrementing, the variable p will start pointing to the second element of the array.
Then in “ (*(*ptr))++ ” :
First it is dereferencing to the address where ptr is pointing, which is p here. Then it is dereferencing to the location which p contains, which is the second element of the array. The value of this location is then incremented by 1.(so *p become 20 to 21 now)
Hence in the end , on printing *p, we will get 21 on the output screen.

Question 3 :

```
int  a = 4,b=6,c=9;
    int *p = &a;
    int *q= p;
    p = &b;
    a++;
    (*q)++;
    b = *q * 2;
    c = *q + *p;
    printf("%d %d %d",a,b,c);
```

There are three integer variables a,b,c and p is pointing to a and q is also pointing to a initially. Then p is assigned with the address of variable b.
Value of a is incremented by 1,(so a =5 now).
Then q is dereferenced and the value at location of q is incremented by 1.
As q is pointing to ‘a’ so the value of ‘a’ becomes 6 now.
Then in the statement b = *q + 2; b is assigned to the value of *q (which is 6) * 2 ; Hence the value of b becomes 12 now.
Then in the statement c = *q + *p; c is assigned to the value of *q (which is 6) and value of *p(which is 12, as p is pointing to b and b is 12 itself).
So c becomes 6+12=18 now.
Hence the values of a , b ,c are 6 , 12 and 18 respectively.

Question 4 :

```
void ubswap(int **a,int **b){
    int *temp=*a;
    *a=*b;
    *b=temp;
}
int main()
{
    int x = 1,y=9;
    int *u=&x; int *v = &y;
    int **a = &u; int **b = &v;
    printf("%d %d %d %d",*u,*v,**a,**b);
    ubswap(a,b);
    printf("\n%d %d %d %d",*u,*v,**a,**b);
    return 0;
}
```

In this code, there are a total of six variables, x and y are integer variable, u and v are the pointers to integers which are pointing to x and y respectively. And a and b are the pointers to pointers which are pointing to the x and y respectively.
Hence in the function ubswap we are swapping the value of *a and *b. That means we are swapping the values of the location of a and b, which contain addresses of u and v.
So the values of the locations of u and v are swapped but still a is pointing to the same location i.e. u and b is pointing to same location i.e. v.

Hence the values of the variable a and b are not swapped but the values of the u and v are swapped here.

Question 5:

```
    double a[3][3];
    double *p = &a[1][1];
    *(p+2)=7;
```

Here p is a pointer variable pointing to block of memory type double.
Initially it is containing the address of the block a[1][1].
So in the line *(p+2)=7; it is moving forward two blocks so from the location of a[1][1], first it moves to a[1][2] and then to a[2][0].
Then it assigns the value 7 to a[2][0].
Hence the cell of the array ‘a’ which is modified by the assignment in this code is a[2][0].

Question 6 :

```
int a[6] = {1,2,3,4,5},i,*p=a;
    for(i = 1; i <6 ; i++){
        *(a+i) += *(a+i-1);
    }
    printf("value 1 = %d\n",*p++);
    printf("value 2 = %d\n",p[2]);
    printf("value 3 = %d\n",(*p)++);
    printf("value 4 = %d\n",++p[3]);
    printf("value 5 = %d\n",(*(p+3))--);
    printf("value 6 = %d\n",*--p);
    printf("value 7 = %d\n",*++p);
```

The output of this code is :

```
value 1 = 1
value 2 = 10
value 3 = 3
value 4 = 16
value 5 = 16
value 6 = 1
value 7 = 4
```

In this code a is an array of size 6 whose values are initialised in starting.
The pointer p is pointing to the first block of this array.
In the for loop we are adding the value of previous block to current block and then reassigning the obtained value to the current block.
The array initially is : {1,2,3,4,5,0}
After for loop the array is : {1,3,6,10,15,15}
The variable p is pointing to the first element of this array.
In first line we are printing *p++; in this line value at location is dereferenced first and then the variable p is incremented. Hence value at p, which is 1, is printed, after dereferencing the pointer is incremented so it is pointing to next block after incrementing. Hence p is now pointing to the

value at the second block of the array, which is 3.
In the second line we are printing the value at p[2], which is 10. Because p[2] is the fourth element in the original array. As p is now pointing to the second element of the array, the indexing in the p is also done in the same manner.

In the third line we are printing the value at (*p)++, which is 3. Because in (*p)++, dereferencing is done first. So value of *p is printed first and the value is incremented afterward. As post increment is used here.
So now the modified array is {1,4,6,10,15,15}

In the fourth line we are printing the value at ++p[3], which is 16. Because in ++p[3], first the value of p[3] is incremented and then we are accessing the value p[3].
So now the modified array is {1,4,6,10,16,15}

In the fifth line we are printing the value at (*(p+3))--), which is 16. Because in (*(p+3))--), first the value of *(p+3) is assigned and then we are decrementing the value *(p+3).
So now the modified array is {1,4,6,10,15,15}

In the sixth line we are printing the value at *--p, which is 1. Because in *--p, first the value of p is decremented and then we are accessing the value *--p.

In the seventh line we are printing the value at *++p, which is 4. Because in *++p, first the value of p is incremented and then we are accessing the value *++p.

Question 7 :

```
#include <stdio.h>
int main()
{
    char a[]="INTIMETEC";
    char *p=a;
    printf("%c\n",*p++);
    printf("%c\n",*(p+2)=p[3]);
    printf("%c\n",*(p+3)=p[-1]);
    printf("%c\n",*++p);
    printf("%c\n",*p++ = *a +2 );
    printf("%c\n",++*p++);
    printf("%ld\n",++p-a);
    printf("\n%s\n",a);
    return 0;
}
```

The output of this code is :

```
I
M
I
T
K
N
5

INKNIETEC
```

Here ‘a’ is an array of characters and is a pointer pointing to char.

In printf("%c\n",*p++); it prints the character pointed to by p (which is 'I') and then increments p by 1 to point it to next charcter.
Now p is pointing to charcter at index 1.

printf("%c\n",*(p+2)=p[3]); In this line the value at *(p+2) is assigned with value at p[3], which in this case is ‘M’. Hence char ‘M’ is assigned at this index and also printed in this line. Also in this p is pointing to char at index

1, so *(p+2) means 4th character in original string .
Now modified string is : “INTMMETEC”

printf("%c\n",*(p+3)=p[-1]); In this line the value at *(p+3) is assigned with value at p[-1], which in this case is ‘I’. Hence char ‘I’ is assigned at this index and also printed in this line. Also in this p is pointing to char at index 1, so *(p+3) means 5th character in the original string .
Now modified string is : “INTMIETEC”

Currently pointer p is pointing at 1 index of original string.
printf("%c\n",*++p); In this line the pointed is incremented to next index and then value *p is printed. In this case after incrementing the p will point to index 2, and hence print the character ‘T’.
Now p is pointing to index 2.

printf("%c\n",*p++ = *a +2); In this line the value at *p++ is assigned with value *a +2, which in this case is ‘K’ . Hence char ‘K’ is assigned to this index and also printed in this line. Also in this p is incremented by 1, hence it is pointing now to character at index 3 in the original string. Now modified string is : “INKMIETEC”
Now p is pointing at index 3.

printf("%c\n",++*p++); In this line the pointer is dereferenced first, hence the character to which p is pointing is ‘M’. Then it is pre-increment by1 which makes it ‘N’. So this line print the char ‘N’ on the screen. After this, p is incremented by 1 and as a result p is now pointing to index 4;
Now modified string is “INKNIETEC”

printf("%ld\n",++p-a); this line pritnt the difference between p and adress of a . If first increment p by 1 and then evaluate difference between them and then print it. As p is at index 4 alredy. On incrementing it will point to index 5 now. The difference between char location of index 0 and 5 will be 5 .
Hence this line will print 5 on the ouput screen.

printf("\n%s\n",a); this line will simply print the string on the ouput screen.

Which is “INKNIETEC”.

Question 8 :

```
int main()
{
    int a,*ptr,arr[12]={1,2,1,2};
    ptr = arr;
    a = ++*ptr + 3 ;
    //line 1
    printf("%d ",a);
    //line 1
    printf("%d ",a);
    return 0;
}
```

A. line 1 : a = *ptr++ +3; and line 2: a = *ptr++ +3;
B. line 1 : a = *ptr++ +3; and line 2: a = ++*ptr +3;
C. line 1 : a = ++*ptr +3; and line 2: a = ++*ptr +3;
D. line 1 : a = ++*ptr +3; and line 2: a = *ptr++ +3;

In this code the appropriate code that we have to use is in option d .
In code a = ++*ptr +3; ptr is dereferenced first and then the value at address ptr is incremented by 1. The value at *ptr is 1 which is incremented to 2 + 3 =5 , hence 5 is assigned to a and 5 is printed in next line.
Here the value at location ptr is also incremented to 2 for further processing.
In the code a = *ptr++ +3; the value at ptr is dereferenced and then the pointer ptr is incremented. The value at ptr is 2 +3 which is 5, hence 5 will be assigned to a again and 5 will be printed again.
Although ptr will now point to next integer in the array.

Question 9 :

```
If x is an one dimensional array, then
```

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

***(x+i) is same as *(&x[i]) :** This statement is correct because *(x+i) will be the value present at ith index of array x , as x is the base address and x+i will be the address of the location of value of index i , and then the value at that index is dereferenced, hence we'll get the value at index i . At the same time in *(&x[i]), &x[i] is the address of ith block in array x, and then the value at that index is dereferenced. So we get the same result in both way.

&x[i] is same as x+i-1 : These both values are not same because first one is the address of the ith index in the array where second one is the address of the (i-1)th block in the array.

***(x+i) is same as *x[i] :** These both values are not same because first one is the value present at the ith block in the array, and the second is wrong because we are trying to dereference to a int value present at ith index of the array.

***(x+i) is same as *x+i :** These two are not same because first one is the value present at the i index of the array, whereas the second one is the value present at 0th index + i.

Question 10 :

```
#include<stdio.h>
int main(){
    int num = 5;
    int arr[3] = {31,32,33};
    int *ptr = NULL;
    ptr = &num;
    *ptr = 15;
    ptr = &arr[2];
    *(&arr[1]+1) += 3 ;
    printf("%d, %d, %d\n",num,arr[0],arr[1]);
    return 0;
}
```

In this code, arr is an one dimenstion array of size 3. And ptr is an pointer to integer.
At first ptr is NULL, then it is pointing to the location of integer num,
In the line *ptr = 15, location of num variable is stored with value 15. Hence value of num is now 15.
Then ptr is pointing to location of integer at index 2.
In the line *(&arr[1]+1)+=3 first the address of the block of index 1 is accessed and incremented by 1. Hence inside the parentheses the address is of block of index 2. This block is then dereferenced and thats value is increased by 3. Hence now the value of the third block or the value of 2nd index of the array is increased by 3 from 33 to 36.
Then we are printing num, arr[0],arr[1] which are 15, 31,32 respectively.

Question 11 :

```
#include <stdio.h>
void fun(int (*p)[3]);
int main()
{
    int arr[][3]={20,25,30,35,40,45};
    fun(arr);
    return 0;
}
void fun(int (*p)[3]){
    ++p;
    printf("%d ",*(*p+2)); // 45
    p-=1;
    printf("%d",*(*(p+1)-1)); // 30
}
```

2-dimensional array arr is defined without specifying the no. or rows and only specifying the no. of columns, Compiler allocate 6 integers to this 2-d array of 3 columns and 2 rows. It, by itself, decides the no. of rows by the no. of values assigned. If there would have been 9 integers instead of 6 then it will create 2-d array of 3 rows and 3 columns.
So arr is now 2-d array of 2 rows and 3 columns.
Then function fun is called by passing base address of array arr i.e “arr”.
Function fun has one argument of pointers p which is pointer to one - dimensinoal array of size 3.
When the arr is passed to functions fun, its value is assigned to pointer p . Hence pointer p contain now the addres of first row of the 2-d array.
In function fun, first p is increment by the line ++p. So now p is pointing to second row of the 2-d array.
In the very next line it prints 45 in output as *(*p+2)), Because *p means the addres of first element of 2nd row and +2 in it means to point to 3rd element of the same row i.e. 45 . Then it is dereferenced using * and hence it print 45 in the ouput.
In next line p is decremented by 1.(p-=1) Now p is pointing again to the 1st row (1st 1 dimensional array of 2d array).
Then it is printitng *(*(p+1)-1)). In this *(p+1) first it point to the next row. And this *(p+1) mean the addres of the first element of the second row. It is address of the first integer block of 2nd 1-dimensional array. It is decremented by 1 then, and we get the addres of previous block. Which is last element of the previous 1-dimensional array. Then this location is derefrenced and we get the last element of the first array which is 30.
So output is
45 30

Question 12 :

```
#include <stdio.h>
void fun(int (*p)[3]);
int main()
{
    char * name[]={"ravi","ravindra","ravindra saini"};
    printf("%s\n",*(name+1));
    printf("%s\n",*name+1);
    printf("%c\n",*(*(name+2)+7));
    printf("%c\n",name[2][7]);
    return 0;
}
```

Output for this code is :

```
ravindra
avi
a
a
```

In this name is an array of pointers. It stores character pointers.
Hence *name will be first element of the array name, which is address of first character of the string to which it points.

printf("%s\n",*(name+1));
In this line, first name is incremented by 1 which makes it pointing to element of index 1. This is then dereferenced. On dereferencing compiler get the address of the first character of the string at the index 1.
So it print the string “ravindra”

printf("%s\n",*name+1);
In this line, first name is dereferenced and the it is incremented by 1 which makes it pointing to element of index 1 of the first string.
Then this is passed to the function to print it. As it contain the addres of the second character of the first string. It will print the string from that character.
The string is “ravi”. So it will print from the second character i.e. “avi”
So it print the string “avi”

printf("%c\n",*(*(name+2)+7));
In this line, first name is incremented by 2 which makes it pointing to

element of index 2. This is then dereferenced. On dereferencing compiler get the address of the first character of the string at the index 1. Then again it is incremented by 7 which makes it pointing to the address of the character at 7th index of the string at index 2. Then again it is dereferenced, so it print the character at 7th index which is ‘a’
So it print the character ‘a’

printf("%c\n",name[2][7]);
This line print the character at 7th index of the element at index 2. Element at index 2 is “ravindra” and the character at 7th index of it is ‘a’.

Question 13:

```
#include <stdio.h>
int main()
{
char *s[] = {"apple", "carat", "date", "banana"};
char **ptr[] = {s,s+3,+1,+2};
char ***p = ptr;
printf("%s\n",*--*++p + 2);
printf("%ld\n", p+3-(ptr+1));
return 0;
}
```

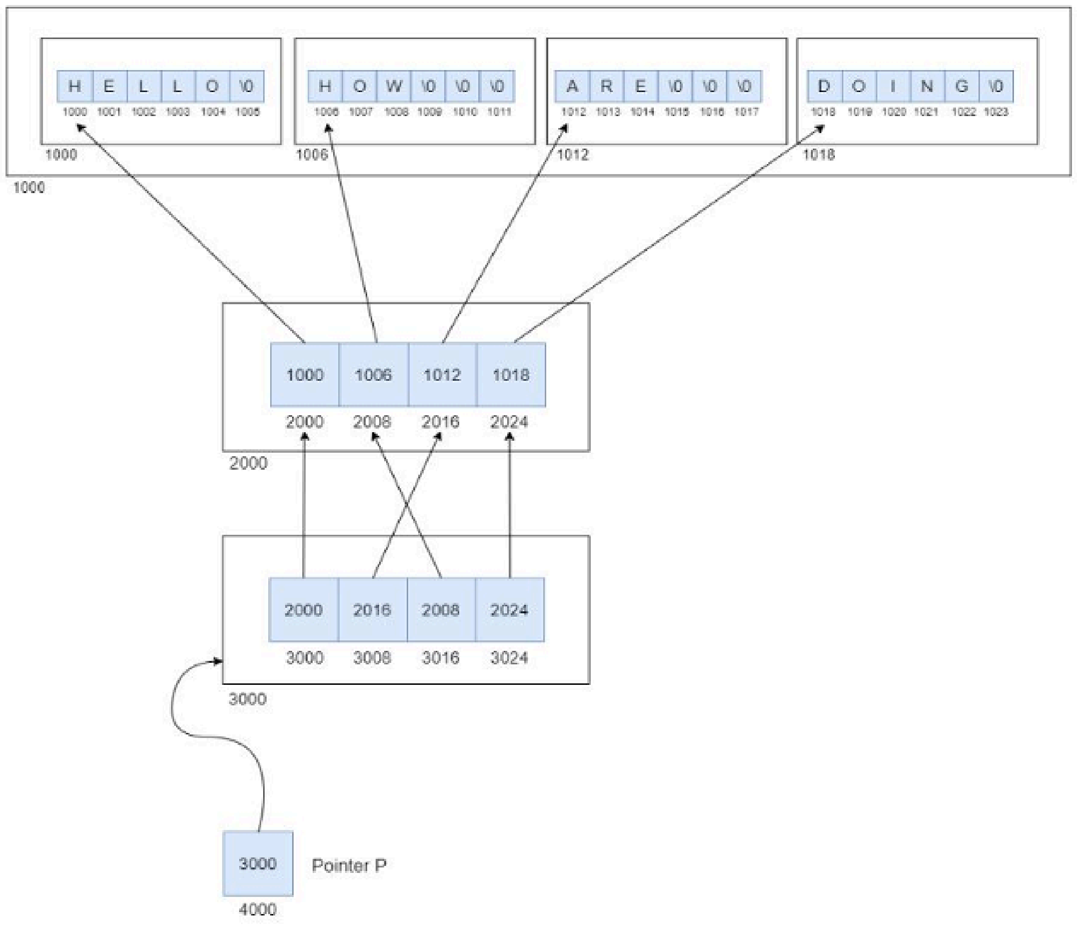
In this code :
→ s is array of character pointers.
→ ptr is array of array of character pointers
→ p is pointint to the first element of the ptr.

Then int the line printf("%s\n", *--*++p + 2); p is first incremented by 1 so it will be pointing to the element at index 1 of the ptr . It is then dereferenced , so we get the element at the index 1 , which is s+3, so we get the address of the character array at index 3 of ‘s’, Then it is decremented which get us the address of the character array at index 2 of ‘s’. Then it is dereferenced, which get us the element present at the 2nd index of the array ‘s’ which is

the address of the first character of the array “date”.
Then we add 2 in it so we now get the address of the index 2.
Then it print the characters onwards from this index. I.e. “te”.

As p is already incremented by 1, now p is containing the address of the second block of ptr array.
Hence the p+3 will mean the block next to 3rd block in ptr and ptr +1 mean the second block in the ptr. And the difference of these two will generate the value 2 as output result.

Question 14 :



Code :

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

```
char statement[4][6] =
{"HELLO\0","HOW\0\0\0","ARE\0\0\0","DOING\0"};
char *ptr[] =
{statement[0],statement[1],statement[2],statement[3]};
char **pptr[] = {ptr,ptr+2,ptr+1,ptr+3};
char ***p = pptr;
printf("%s ",**p);
printf("%s ",**(p+2));
printf("%s ",**(p+1));
printf("%s ",**(p+3));

return 0;
}
```

statement is an 2-d array of characters .
Then there is an array ptr which is array of character pointers.
ptr[0] has the address of first character of first 1-dimensional array of statement array.
ptr[1] has the address of first character of second 1-dimensional array of statement array.
ptr[2] has the address of of first character of third 1-dimensional array of statement array.
ptr[3] has the address of of first character of fourth 1-dimensional array of statement array.

Then again we create an array pptr which contain the addresses of different block of ptr array.
pptr[0] has address ptr.
pptr[1] has the address of the third element of ptr.
pptr[2] has the address of the second element of ptr.
pptr[3] has the address of the fourth element of ptr.

Then we declare a pointer p which is pointer to a pointer to a pointer to a character.

printf("%s ", **p);
In this code p is dereferenced to point to the first character of the string “HELLO”.
So this line prints “HELLO”.

printf("%s ", ** (p + 2));
In this code p incremented by 2 and then is dereferenced to point to the first character of the string “HOW”.
So this line print the string “HOW”

printf("%s ", ** (p + 1));
In this code p incremented by 1 and then is dereferenced to point to the first character of the string “ARE”
So this line print the string “ARE”

printf("%s ", ** (p + 3));
In this code p incremented by 3 and then is dereferenced to point to the first character of the string “DOING” this line print the string “DOING”

Hence the final output on the console will be :
HELLO HOW ARE DOING

Question 15 :
MATRIX MULTIPLICATION :

```c
#include <stdio.h>
#include<stdlib.h>
int main()
{
    //matrix multiplication
    int row1,col1,row2,col2;
    printf("No. of rows and cols for matrix 1 : \n");
    printf("Rows : ");
    scanf("%d",&row1);
    printf("\nCols : ");
    scanf("%d",&col1);
    if(row1<=0||col1<=0){
        printf("Invalid Dimensions\n");
        return 0;
    }
    int matrix1[row1][col1];
    printf("Give Elements of the matrix 1 : \n");
    for(int a = 0;a<row1;a++){
        for(int b = 0 ; b < col1 ; b ++){
            scanf("%d",&matrix1[a][b]);
        }
    }
    printf("No. of rows and cols for matrix 2 : \n");
    printf("Rows : ");
    scanf("%d",&row2);
    printf("\nCols : ");
    scanf("%d",&col2);
    if(row2<=0||col2<=0){
        printf("Invalid Dimensions\n");
        return 0;
    }
    int matrix2[row2][col2];
    printf("Give Elements of the matrix 2 : \n");
    for(int a = 0;a<row2;a++){
        for(int b = 0 ; b < col2 ; b ++){
            scanf("%d",&matrix2[a][b]);
        }
    }
    int result[row1][col2];
    if(col1==row2){
        int total= 0 ;
        for(int a = 0 ;a < row1 ; a++){
            for(int b  = 0 ; b < col2 ; b ++){
                total= 0 ;
                for(int c = 0 ; c < row2  ; c++){
                    total += matrix1[a][c]*matrix2[c][b];
                }
                result[a][b] = total;
            }
        }
        printf("Multiplication result : \n");
        for(int a = 0 ; a < row1 ; a++){
            for(int b = 0 ; b < col2  ;b++){
                printf("%d ",result[a][b]);
            }
            printf("\n");
        }
    }
    else{
        printf("Invalid input for the matrix 
multiplication\n");
    }
    return 0;
}
```

Firstly four variables row1,col1,row2,col2 are declared.
Then row1 and col1 are taken as input from user. Here we check and ensure that the dimensions are not zero or negative and dimensions are valid.
Then we create a 2-d array with dimension row1 and col1 and take the values of this array as input from user.
Then row2 and col2 are taken as input from user. Here we check and ensure that the dimensions are not zero or negative and dimensions are valid.
Then we create a 2-d array with dimension row2 and col2 and take the values of this array as input from user.

Now we have to check and make sure that given two matrices have valid dimensions to perform multiplication on them.
So we check the condition that whether column of first matrix are equal to row of second matrix.
We declare a result matrix of dimensions row1 and col2, to store the multiplication result .
Then we apply three nested for loops to find the multiplication result of the matrix.
After finding the result using for loop, it simply print the data of the result matrix.