OBJECTIVE:

Q1.

A. d
B. c
C. b
D. c
E. a,b,c
F. c
G. b

Q2.

1. Single
2. Precedence
3. Itself
4. End-Of-File
5. Unions
6. Complete/finished/over
7. 3, arg1
8. Default
9. Once

Q3:

a) 5 4 3 2 1 (0.5+0.5+0.5+0.25+0.25 marks)

b)
```
From function fine
From main fine          (2+2 marks)
```

c) No (2 marks)

d)
```
2     7
4     9
6     11
8     13
0     15          (0.5+0.5+0.5+0.5+2 marks)
```

e) 5 (2 marks)

Section B: Subjective

Q1:

a) j=highest(record,3);

b)
```
Employee details:
Name: Tanuj
Birthdate: 21.1.75
Salary:  5500.000000
```

c) i) void oldest(emprec record[ ], int n)
   ii) Line 1: if(temp.birthday.month>record[i].birthday.month)
      Line 2: if(temp.birthday.month = = record[i].birthday.month)
      Line 3: if(temp.birthday.day>record[i].birthday.day)
      Line 4: printf("The oldest person is %s", temp.name);

d) Line 1: str=(char*)malloc(strlen(temp));
   Line 2: record[i].name = str;
   Line 3: strcpy(str,temp);

Q2:

a)
```
1    40.500000
2    55.500000
3    70.166664
4    85.000000
```

b)
```
1    40.500000
2    16.833334
3    21.666666
4    26.666666
```

c) Line 1: for(r=0;r<S;r++) **OR** for(r=0;r<3;r++)
   Line 2: for(s=0;s<R;s++) **OR** for(s=0;s<4;s++)
   Line 4: t=mm[s][r];

d) Function header: float fnnew(float mm[R][S])
   Line 3: t=t+mm[s][r];
   Line 5: return(t);

Q3.

A)

1. Printf(" %s \n", a); //There is only one character (l==r) in the remaining string, so it is the only permutation possible and hence print it.

2. for (i=l;i<=r;i++) //to generate all arrangements, one by one every character in the string has to be placed in the first position. So index will begin from first (l) and increment upto the last position (r).

3. permute(a,l+1,r); // Now when the character at the l*th* position is fixed, solve the problem of finding all the permutations for remaining string whose starting index is (l+1)

4. permute(str,0,n-1) //to generate permutations of str whose starting index is and index of with n-1 (excluding null character)

B)

/* Function to swap values at two pointers */

```
void swap(char *x, char *y)
{
    char temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

C) For non-distinct characters, some permutations will repeat and the program will print all of them (and not just the unique permutations). So some of the permutations will be printed multiple times.

Q4.

A.

1. int, int

2. 0, prime-1

3. root, r

4. powerOfRoot, prime

5. root, prime

B.

```
int checkRemArray(int a[], int n)
{
 int i=1;
 for(;i<n;i++)
       if(!a[i])  return 0;
return 1;
}
```

C. // The changes are highlighted by red

```
int main(){
int prime = 19; // finding the primitive root of 19, say
int root = 2;
int flag=0;
      for (; root<prime; root++)
      {
             int r, remainder[19] = {0}; // the index of the array corresponds
//to a remainder value
```

```
            for (r =0; r<prime-1;r++){
                  long powerOfRoot = pow(root,r);
                  int rem= powerOfRoot % prime; //find remainder under mod p
                  remainder[rem] = 1; //put 1 in the corresponding index of
//remainder array
            }
            if (checkRemArray(remainder, prime))
            {
                  printf("The primitive root of %d is %d", prime,root);
                  flag=1;
            }


      }
      if(!flag)
            printf("There is no primitive root of %d ", prime);
      return 0;
}
```

Q5.
(a). 42 42

(b). fourth->parent = first;
first->left_child = fourth;
free(second);
second=NULL;

(c). Space occupied by node second is released. Hence, the total space occupied by the nodes in the heap memory is now 48 bytes.

(d). The program will not even compile because member parent is not known.

(e). Pointer parent gives gives us freedom to move back and forth in the structure. If it is removed, one cannot determine the address of the parent node. For example, node first knows how to reach node second but second will not know how to reach to first.

(f). NODE* fifth;
fifth = malloc(sizeof(NODE));

third->right_child = fifth;
fifth->parent = third;

```
    fifth->data = 21;
    fifth->left_child = NULL;
    fifth->right_child = NULL;
```

Q6.
(a). function2 will return pointer to a local variable which is destroyed once the function is over. The memory allocated to that variable array may be reused. Hence, the behavior of the program is not determined. It may give the correct result, it may not.

(b). Both the array variables happen to have the same name but their scope is completely different. The first array's scope is the block of if and the scope of the other array is the block of else. None of the array have a scope which makes it visible in the whole function. Hence, writing "return array;" just before the last } will result in compiler error stating array undeclared.

(c). The if block allocates memory to the variable array statically to a local variable which will be destroyed as soon as its block is exited. On the other hand, the else block allocates memory dynamically, it will remain allocated until released using the function free().

(d). 5 43

(e). Variables i, x, dm, array1, array2, array3 will be alive because they all have been declared in the scope of main() which is not yet exited. Depending on how we count the variables the count can be 6 or 33 (as each array is a collection of 10 variables.)