Determine Output

```
int funct1(char c1, char c2)
{
c1 = 'P'; c2 = 'Q';
return((c1 < c2) ? c1 : c2);
}
int funct2(char *c1, char *c2)
{
*c1 = 'P'; *c2 = 'Q';
return((*c1 == *c2) ? *c1 : *c2);
}

int main() {
   char a = 'X';    char b = 'Y';
   int i, j;
   i = funct1(a, b);
   printf("a=%c b=%c, i=%d\n", a, b,i);

   j = funct2(&a, &b);
   printf("a=%c b=%c j=%d", a, b,j);
   return 0;
}
```

Answer: a=X b=Y, i=80
a=P b=Q j=81

```
void funct(int *p);
main()
{
  int a[5] = {10, 20, 30, 40, 50};
  funct(a+1);
  funct(a+2);
}

void funct(int *p)
{
int i, sum = 0;
for (i = 3; i < 5; ++i)
sum += *(p + i);
printf("sum=%d", sum);
return;
}
```

Answer: Garbage Garbage

What will happen for
static int a[5] = {10, 20, 30, 40, 50}; ?

Answer: 50 0
(since static a[5] onwards will be initialized with 0)

```
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));
   return 0;
}
```

Answer: 19

```
 struct temp
   {
      int a;
   } s;

void func(struct temp s)
   {
      s.a = 10;
      printf("%d\t", s.a);
   }

void main()
   {
      func(s);
      printf("%d\t", s.a);
   }
```

Answer: 10 0
(since in main() global s will be accessed and it has s.a=0 as initialized by the complier)

```
struct newtype
{
    int x;
```

```
 int main()
{
   int a; scanf("%*s %d", &a);  printf("a=%d", a);
```

```
   struct newtype next;
};

int main()
{
   struct newtype temp;
   temp.x = 10;
   temp.next = temp;
   printf("%d", temp.next.x);
   return 0;
}
```

Answer: Compilation Error

(A structure cannot contain a member of its own type because if this is allowed then it becomes impossible for compiler to know size of such struct.)

Why does it work for **struct newtype *next;** (this is known as self-referencing structure. Usage: in linked list)

A pointer of same type can be a member because pointers of all types are of same size and compiler can calculate size of struct. Hence, respective modification will be:

```
   temp.next = &temp;
   printf("%d", temp.next->x);
```

```
   return 0;
}
```

Answer: a=56

(The %*s is used to ignore some input as required. Here, it ignores the input until the next space or newline.)

Now explain the following.

```
   /* Assuming that abc.txt has content in below format
      abc    10   kolkata
      bef    12   delhi
      cce    50   bangalore */
 int main()
{
   FILE* ptr = fopen("abc.txt", "r");
   if (ptr == NULL) {
      printf("no such file.");
      return 0;
   }
   char buf[100];
   while (fscanf(ptr, "%*s %*s %s ", buf) == 1)
      printf("%s\n", buf);
   return 0;
}
```

Answer: kolkata
delhi
bangalore

(due to %*s %*s strings in 1st and 2nd columns will be ignored)

```
int a,b,c;
scanf("%3d %3d %3d", &a, &b, &c);
```

If inputs are follows, what will be the output?

1. 56 4 0
2. 567 890 321
3. 567890321
4. 5678 9032 1

Answer:

1. a=56 b=4 c=0
2. a=567, b=890, c=321
3. a=567, b=890, c=321
4. a=567, b=8, c=903

```
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);
```

```
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);
}

int main()
 {
  printf("%d", fun2(2));  //5
```

```
}

int main()
 {
  printf("%d", SomeFunction(5,15));
  return 0;
 }
```

Answer: 5 (Easy to trace and cross check)

```
 return 0;
 }
```

Answer: 5

(Trace the code Remember static variable will be created once and will be used until the program stops executing. For fun1 and fun2 two static variables will be created.)

```
void fun1(char *s1, char *s2) {
char *temp;
temp = s1; s1 = s2; s2 = temp;
}
void fun2(char **s1, char **s2) {
char *temp;
temp = *s1; *s1 = *s2; *s2 = temp;
}
int main() {
char *str1 = "Hello", *str2 = "World";
fun1(str1, str2);
printf("%s %s", str1, str2);
fun2(&str1, &str2);
printf("%s %s", str1, str2);
return 0;
}
```

Answer: Hello World World Hello

(fun1 follows call by value, fun2 follows call by reference)

```
#include <stdio.h>
struct newtype {
char x, y, z;
};

int main() {
struct newtype p = {'1', '0', 'a' + 2};
struct newtype *q = &p;
printf("%c, %c %c", *q, *(q + 1), *(q + 2));
return 0;
} //
```

Answer: 1, garbage, garbage

(q+1) will point to the next memory location (with some garbage value) where memory allocation for newtype ends. Similar reason for (q+2)

What will happen for

printf("%c, %c %c", *q, *((char *)q + 1), *((char *)q + 2)); ?

Answer: 1, 0, c

('a'+2 gives you c. So, p = {'1', '0', 'c'}. Since typecasting has been done for q+1 and q+2, so now they will point to the next and next to next character in p respectively.)