```
#include<stdio.h>
void func(int x,int y,int *ps,int *pd,int *pp);
int main(void)
{
        int a,b,sum,diff,prod;
        a=6;
        b=4;
        func(a,b,&sum,&diff,&prod);
        printf("Sum=%d, Difference=%d, Product=%d\n",sum,diff,prod);
        return 0;
}
void func(int x,int y,int *ps,int *pd,int *pp)
{
        *ps=x+y;
        *pd=x-y;
        *pp=x*y;
}
```

Output:
Sum=10, Difference=2, Product=24

In `func()`, variables `a` and `b` are passed by value while variables `sum`, `diff`, `prod` are passed by reference. The function `func()` knows the addresses of variables `sum`, `diff` and `prod`, so it accesses these variables indirectly using pointers and assigns appropriate values to them.

9.16 Function Returning Pointer

We can have a function that returns a pointer. The syntax of declaration of such type of function is-

```
type *func(type1,type 2,...);
```

For example-

```
float *fun(int,char);    /*This function returns a pointer to float*/
int *func(int,int);      /*This function returns a pointer to int*/
```

While returning a pointer, make sure that the memory address returned by the pointer will exist even after the termination of function. For example a function of this form is wrong.

```
int main(void)
{
        int *ptr;
        ptr=func();
        ..............
}
int *func()
{
        int x=5;
        int *p=&x;
        ..........
        return p;
}
```

Here we are returning a pointer which points to a local variable. We know that a local variable exists only inside the function. Suppose the variable `x` is stored at address 2500, the value of `p` will be 2500 and this value will be returned by the function `func()`. As soon as `func()` terminates, the local variables `x` will cease to exist. The address returned by `func()` is assigned to pointer variable `ptr` inside `main()`, so now `ptr` will contain address 2500. When we dereference `ptr`, we are trying to access the value of a variable that no longer exists. So never return a pointer that points to a local variable. Now we will take a program that uses a function returning pointer.

```
/*P9.19 Function returning pointer*/
#include<stdio.h>
int *fun(int *p,int n);
int main(void)
{
        int n=5,arr[10]={1,2,3,4,5,6,7,8,9,10};
```

```
    int *ptr;
    ptr=fun(arr,n);
    printf("Value of arr=%p, Value of ptr=%p, value of *ptr=%d\n",arr,ptr,*ptr);
    return 0;
}
int *fun(int *p,int n)
{
    p=p+n;
    return p;
}
```

Output:
Value of arr=0012FEA4, Value of ptr=0012FEB8, value of *ptr=6

9.17 Passing a 1-D Array to a Function

In the previous chapter we had studied that when an array is passed to a function, the changes made inside the function affect the original array. This is because the function gets access to the original array. Here is a simple program that verifies this fact.

```
/*P9.20 Passing 1-D array to a function*/
#include<stdio.h>
void func(int a[]);
int main(void)
{
    int i,arr[5]={3,6,2,7,1};
    func(arr);
    printf("Inside main(): ");
    for(i=0; i<5; i++)
        printf("%d  ",arr[i]);
    printf("\n");
    return 0;
}
void func(int a[])
{
    int i;
    printf("Inside func(): ");
    for(i=0; i<5; i++)
    {
        a[i]=a[i]+2;
        printf("%d  ",a[i]);
    }
    printf("\n");
}
```

Output :
Inside func() : 5 8 4 9 3
Inside main() : 5 8 4 9 3

Now after studying about pointers we are in a position to understand what actually happens when an array is passed to a function. Actually a whole array is never passed to a function, only a pointer to the first element of the array is passed. There are three ways of declaring a formal parameter which has to receive the array. We can declare it as an unsized or sized array or we can declare it as a pointer.

```
func(int a[])
{
    .....
}
func(int a[5])
{
    .....
}
func(int *a)
{
    .....
}
```