✔Pointers and Arrays

✔Arrays of Pointers

✔Multiple Indirection (Pointer to Pointer)

✔Initializing Pointers

Pointers

Part-2

For Detailed Reading, Consult

The Complete Reference C++, Chapter-5

Pointers and Arrays

There is a close relationship between pointers and arrays

For Example

```
char str[80], *p1;
p1 = str;
```

Here, **p1** has been set to the address of the first array element in **str**

To access the fifth element in **str**, you could write

```
str[4]
```

or

```
*(p1+4)
```

Both statements will return the fifth element.

Arrays of Pointers

Pointers may be arrayed like any other data type.

The declaration for an **int** pointer array of size 10 is

```
int *x[10];
```

To assign the address of an integer variable called **var** to the third element of the pointer array, write

```
x[2] = &var;
```

To find the value of **var**, write `*x[2]`

Arrays of Pointers
Passing to functions

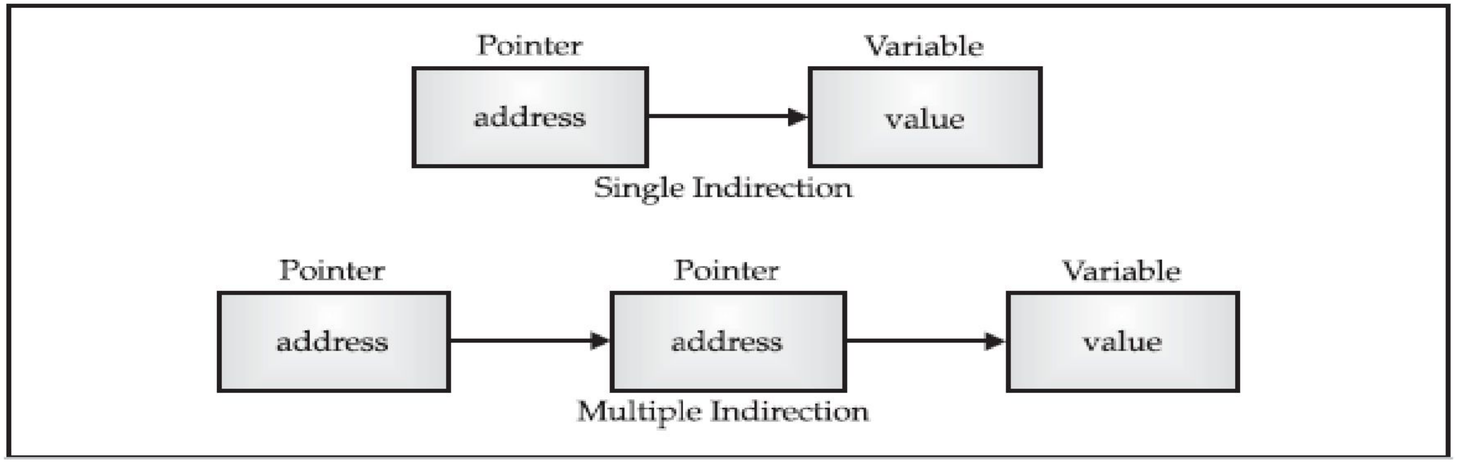
- If you want to pass an array of pointers into a function, you can use the same method that you use to pass other arrays

For example, a function that can receive array **x** looks like this:

```
void display_array(int *q[])
{
  int t;
  for(t=0; t<10; t++)
    cout<< *q[t];
}
```

Multiple indirection (Pointer to Pointer)

- You can have a pointer point to another pointer that points to the target value. This situation is called *multiple indirection*, or *pointers to pointers*



- In the case of a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the object that contains the value desired.

Multiple indirection (Pointer to Pointer) Cont.

- To access the target value indirectly pointed to by a pointer to a pointer, you must apply the asterisk operator twice, as in this example:

```
int x, *p, **q;
x = 10;
p = &x;
q = &p;
cout<<**q;  /* print the value of x */
```

Here, **p** is declared as a pointer to an integer and **q** as a pointer to a pointer to an integer.

Initializing Pointers

A pointer that does not currently point to a valid memory location is given the value null (which is zero)

For example `int *ptr = NULL;`

Null Pointers cannot be de-referenced.

For example if ptr is pointed to null then *p would cause error.