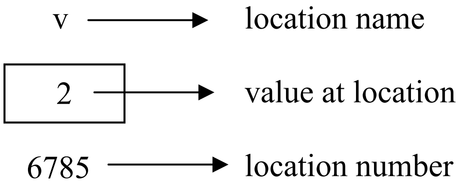
Lecture 13

Pointer: A simple variable in a program is stored in certain number of bytes at a particular memory locations or address in the machine. Pointers are used in program to access memory and manipulate address.

Consider the declaration,

```
int v = 2 ;
```



Example 1:

```
void main(void)
{
int v = 2 ;
printf("Address of v = %u",&v);
printf("\nValue of v = %d",v);
printf("\nValue of v = %d",*(&v));
}
```

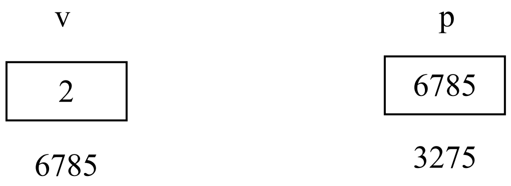
```
Address of v = 6785
Value of v = 2
Value of v = 2
```

We can collect the address of a variable into another variable by saying,

```
p = & v;
```

At first we have to declare p as a variable which will store the address of an integer value.

```
int *p ;
```



Example 2:

```
void main(void)
{
int v = 2, * p ;
p = & v ;
printf("Address of v = %u",&v);
printf("\nAddress of p = %u",&p);
printf("\nValue of p = %u",p);
printf("\nValue of v = %u",v);
printf("\nValue of v = %u", *(&v));
printf("\nValue of v = %u", * p);
}
```

```
Address of v = 6785
Address of p = 3275
Value of p = 6785
Value of v = 2
Value of v = 2
Value of v = 2
```

Example 3:

```
        void main(void)
        {
        int x = 2, y =3, * p, * q ;
        p = & x ;
        q = & y ;
        p = q ;
        printf("%d %d %d %d", x, y, * p, * q );
        * p = 3;
        * q = 4;
        x = y;
        printf("\n%d %d %d %d", x, y, * p, * q );
        }
```

```
2 3 3 3
4 4 4 4
```

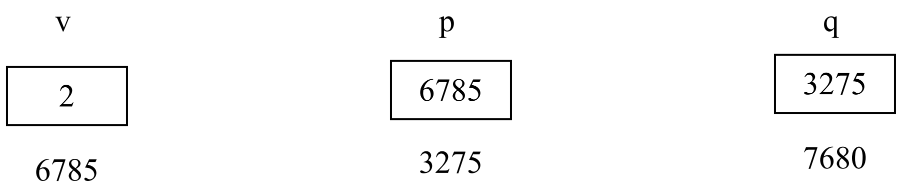
Look at the following declarations,

```
        int  *i ;
        char  *ch ;
        float  *f ;
```

The declaration **float *f** does not mean that f is going to contain a floating point value. What it means is, f is going to contain the address of a floating point value.

Concept of pointers can be further extended. We can declare a pointer which may contains another pointer's address.

```
        int x = 2, * p, ** q ;
        p = & x ;
        q = & p ;
```



Function calls: Arguments of a function can be passed to functions in one of the two ways:

- Sending the values of the arguments (**Call by Value**)
- Sending the address of the arguments (**Call by Reference**)

Call by Value: In this method 'value' of each actual arguments in the calling function is copied into corresponding formal arguments of the called function.

Example 4:

```
void swap(int x, int y);
void main(void)
{
int a = 10, b = 20;
swap(a, b);
printf(“\na = %d b = %d”,a,b);
}

void swap(int x, int y)
{
int temp;
temp = x;
x = y;
y = temp;
printf(“\nx = %d y = %d”,x,y);
}
```

```
x = 20 y = 10
a = 10 b = 20
```

Call by Reference: In this method the addresses of actual arguments in the calling function are copied into corresponding formal arguments of the called function.

Example 5:

```
void swap(int *x, int *y);
void main(void)
{
int a = 10, b = 20;
swap(&a, &b);
printf(“\na = %d b = %d”,a,b);
}

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

```
a = 20 b = 10
```

Note that this program manages to exchange the values of **a** and **b** using their addresses stored in **x** and **y**.

Example 6:

```
void input(int *p, int *q);
int add(int x, int y);
void display(int value);
void main(void)
{
int x,y,sum;
input (&x, &y);
sum = add(x, y);
display(sum);
}

void input(int *p, int *q)
{
scanf("%d%d",p,q);
}

int add(int x, int y)
{
return x+y;
}

void display(int value)
{
printf("The sum = %d", value);
}
```

```
4
5
The sum = 9
```

Using call by reference intelligently we can make a function return more than one value at a time.

Example 7:

```
void areaperi(int r, float *a, float *p);
void main(void)
{
int radius;
float area, perimeter;
printf("Enter radius of a circle");
scanf("%d",&radius);
areaperi(radius, &area, &perimeter);
printf("Area = %f ",area);
printf("\nPerimeter = %f ", perimeter);
}

void areaperi(int r, float *a, float *p)
{
*a = 3.14 * r * r ;
*p = 2 * 3.14 * r ;
}
```

```
Enter radius of a circle 5
Area = 78.500000
Perimeter = 31.400000
```