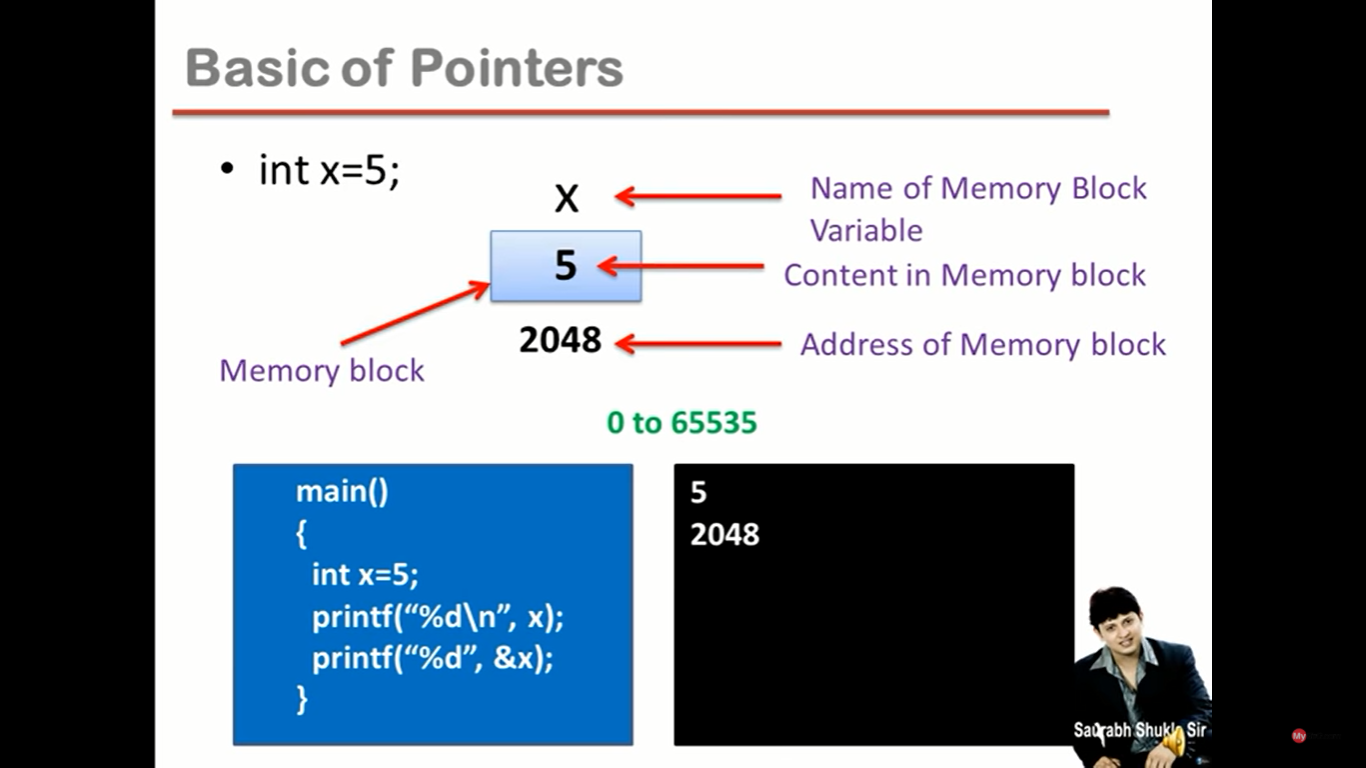
**Pointers**

**Basic of Pointers:-**

int x=5; **//declaration statement**

It means, It says to the compiler that, a variable will create with name ‘x’ which takes some memory (int variable take 2bytes of memory) and assign some value (here, value is 5).



**Note:**

Here, Variable can be consider as memory block or memory location having name ‘X’. So this name of Memory Block i.e. ‘X’ is variable, containing a value ‘5’ i.e. content/value in Memory Block/location or simply say, value of X.

**Address of a variable:** It is a number given to the memory block/location or variable.

According to the Dosed Based Architecture, 64KilloBytes (Kb) i.e. 65536 Bytes of memory given, to run a C Program. So, Every C Program must have 65536 Bytes. We give 0 sequence number to the first byte out of this 65536 bytes of memory. Hence the sequence number of last byte out of 65536 would be 65535.

Sequence Number of 65536 bytes goes in linear way, like this:  
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,..........................,65535

So, the above mentioned every numbers given to each byte of memory is called the sequence number or the reference /reference number or the **Address/ Address number**. Or in Simple word, this Sequence number is called the Address.

So the memory consume by the variable or memory given to each variable in the C program will be in between the range from 0 to 64 Kb or 65535 bytes.

**Note:** In Dosed Based Architecture, Int Variable takes 2 bytes of Memory, whereas In 32Bit Architecture, Int variable takes 4bytes of memory.

Here, We are considering, the above int ‘X’ variable taking 2 bytes of Memory. So, it is obvious that the first & the second byte will take some sequence number or Address number in linear way (like if the address of first byte is 2048 then the second byte will be 2049). But we can’t specify the right address of the memory consuming by the variable by yourself.

When a program run, then the address number is the only thing which specify what will be the exact location of the memory will give to the variable.

**Range of Address of Memory block will be in between 0 to 65535.**

**Example:**

Main(){

Int x=5;

printf(”%d\n”, x); **//print the value of x**

printf(”%d”, &x); **//print the address of the variable ‘x’**

}

Output:  
5  
2048

**Address of Operator:-**

1. **&** is known as address of operator
2. It is an unary operator
3. Operand must be the name of variable
4. & operator gives address number of variable
5. & is also known as referencing operator. Reference and Address both are same.

**Indirection Operator**This operator is the opposite of Address of Operator.

1. **\*** is indirection operator.
2. It is also knows as dereferencing operator.
3. It is an unary operator.**//Uniary Operator always work Right to Left.**
4. It takes address as an argument. **//like printf(“%d”,\*&x);**
5. \* returns the content/container or say value of variable whose address is its argument.

**Example:**

Main()  
{  
int x=5;  
printf(“%d\n”, x);  
printf(“%d\n”, &x);  
printf(“%d”, \*&x);

}

Output:  
5  
2048  
5

**Note:**

* %d print number from -32768 to 32767. So, it’s better to use %u to print number from 0 to 65535.7
* int x =5;  
  &x = 7; **//**It will return an error.  
  We cannot store anything is &x, as &x is not a variable, it is the way to represent address of block x.  
  It is because, here &x is equal to the 2048 (i.e. the address of x). So 2048 is the constant and we cannot put any constant to the left side of the assignment operator because we cannot assign any value in the constant but in the variable.
* But we can put &x right of the assignment operator and that will store the address of x variable and should store their address in another variable.  
  It is necessary to declare that variable before use or assigning address of another variable.  
  We have to use **\*asterics symbol** (indirection operator) with the required predefined data type (depend upon address of which data type variable assigning) while declaring that variable where address of another variable is assigning. **For Ex**: int \*j;  
  If we don’t use \* Asterics symbol while declaring that variable then compiler will unable to differentiate that the value is assigning or the address of that variable is assigning to this variable and return an *([Error] invalid conversion from 'int\*' to 'int' [-fpermissive])*. Because in the case int data type, both value value and address will be integer. **For Ex**: int j;  
    
  **Example:**int x =5; **//storing ordinary integer**  
  int \*j;  
  j = &x; **//storing address and it is also an integer**

**Pointer:-**

* ***Pointer is a variable that contains address of another variable.***
* Pointer always consumes 2 bytes in memory as per Dosed Based Architecture.  
  Maximum Address i.e. 65535 bytes = In Binary, 1^16 (16 times 1)bits = In decimal, 65535. It means, to cocntain maxium address, we need 16 bits of memory i.e. 2bytes which is sufficient to store address. That’s why pointer are always of 2 bytes.

**Note:**

Whenever, we use \*asterics symbol before the pointer variable, then it will representing the value of that variable whose address it’s containing.

**Example:**main(){

int x = 5, \*j;

j = &x;  
printf("%d %u\n", x, j); **//Return 5 Address of x**

printf("%d %u\n", \*j, &x); **//Return 5 Address of x**

printf("%u", \*&j); **// Address of x**  
}

Output:-  
5 2048  
5 2048  
2048

**Base Address:**

Int a, \*j;  
float b, \*k;  
char c, \*r;

A/c to doshed based architecture, ‘a’ variable will consume, 2bytes of memory, ‘b’ variable will consume 4bytes of memory and ‘c’ variable will consume 1byte of memory.

And all pointer ‘j’, ‘k’ and ‘r’ will always consume 2 bytes of memory.

The address to the first byte of the variable is called the **Base Address**.  
We always get the base address of the variable. So if we store, the address of any variable to another variable then it means we are storing the base address of the variable.

If we are storing the address of variable to the pointer then that pointer pointing to that variable.  
For Ex: \*j=&a; \*k=&b; \*r=&c;  
Here, j, k and r pointing to variable a, b and c respectively.

**Note:**Each pointer contains only the address of the variable of same data type. For Ex: ‘k’ pointer is made from float data type so it can only contain/store the address of float datatype variable. It is because pointer only point to the variable of its same data type.

**Extended Concept:-**

If a pointer variable ‘q’ storing the address of another pointer variable ‘p’, then we have to first declare this pointer variable ‘q’ by using \*\* double asterisk symbol.   
For Ex:-   
voind main(){  
 int x=5, \*p, \*\*q, \*\*\*r;  
p = &x;  
q = &p;  
r = &q;  
}  
**Note:**

* Numbers of \* asterisk symbol use by the variable, denotes its level of indirection or say level of pointer or pointer level. As per the above given example, pointer level of ‘q’ variable is 2, pointer level of ‘p’ variable is 1 and pointer level x variable is 0.
* We can only assign or store address of a variable to another variable having level of pointer exactly less than only 1 to another variable.  
  As per the above example, We can only store address of ‘p’ variable to ‘q’ because the pointer level of ‘p’ is less exactly less than by 1 but we can’t assign address of ‘x’ to ‘q’ because the pointer level of ‘x’ is less than by 2.
* int x =5, \*p, \*\*q, \*\*\*r;  
  The above declaration statement can be read as,   
  x – Read as *x is an int* means, x is an int variable.  
  \*p – Read as *p is a pointer to an int*, means p is a pointer which pointing to an int variable (x) containing int address.  
  \*\*q – Read as *q is a pointer to a pointer to an int,* means q is a pointer pointing to a pointer (\*p) which pointing to an int variable (x).  
  \*\*\*r – Read as r is a pointer to a pointer to a pointer to an int, means r is a pointer pointing to a pointer (\*\*q) which pointing to another pointer (\*p) which pointing to an int variable (x).
* If we use \* Asterisk before the variable, it means it is pointing towards its lower variable or say pointing to that variable whose address it’s storing.

**Pointer’s Arithmetic:-**

1. We cannot add, multiply or divide two addresses (**Subtraction is possible**). Ex: &a +\*/ &b;
2. We cannot multiply an integer to an address and similarly we cannot divide an address with an integer value.   
   Ex: &a \*2; or &b/5;
3. We can add or subtract an integer to/from the address.  
   **Example:**voidmain(){  
   int a, b;  
   int \*p, \*q;  
   p = &a;  
   q = &b;  
   p+1;  
   }  
   Let say, the &a is 1000, the p+1 would be 1002 not 1001. Because As we know int variable consume 2byte of variable as per dosed based architecture and By default , Base address always assign to any other variable i.e. the address of first byte of the variable. But whenever we add an integer with the pointer then it returns the address of the next variable/block instead the address of the next byte.  
     
     
   Formula:-  
   Pointer + n = pointer + sizeof(typeof pointer) \*n  
   **Ex:**   
   p+4 = 1000 + 2\*4 = 1008  
   Where, 2 is the size of Int and p is the pointer storing address 1000.  
   p-1 = 1000 – 2\*1 = 998
4. We can subtract two addresses of same type.  
   Formula:-  
   Pointer1 – Pointer2 = Literal Subtraction / sizeof(type of pointer)  
   **Ex:**q –p = 1020 -1000 = 20 / 2  
   Where, q = &b = 1020(assumed) and p = &a = 1000(assumed), 20 is the literal Subtraction and 2 is the int size of pointer.