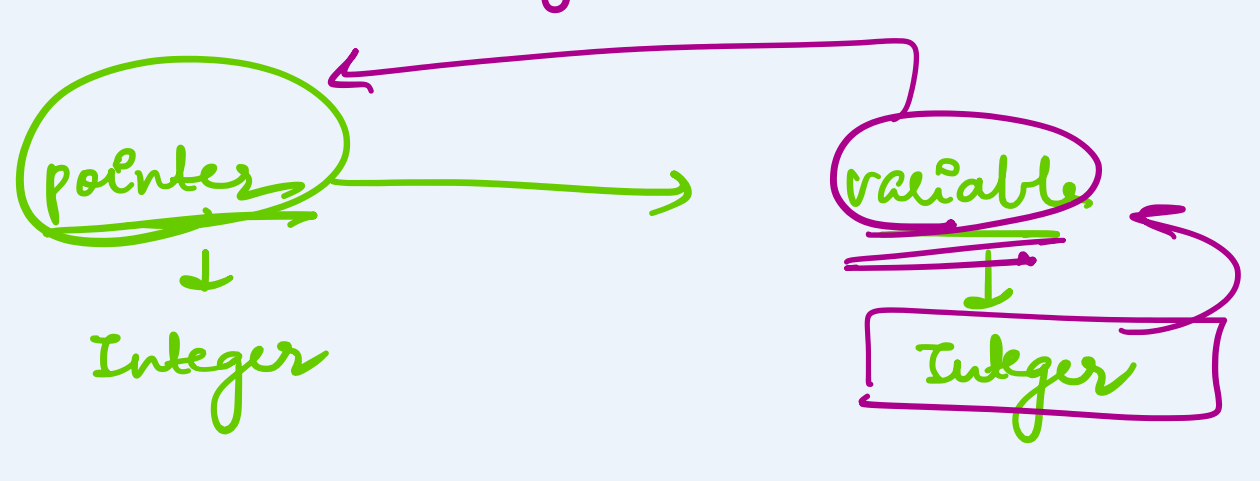
Friday, 30 April 2021 8:21 PM

Pointers

→ A pointer refers to a variable that holds the address of another variable.

→ have a data type



Syntax

datatype *variableName ;

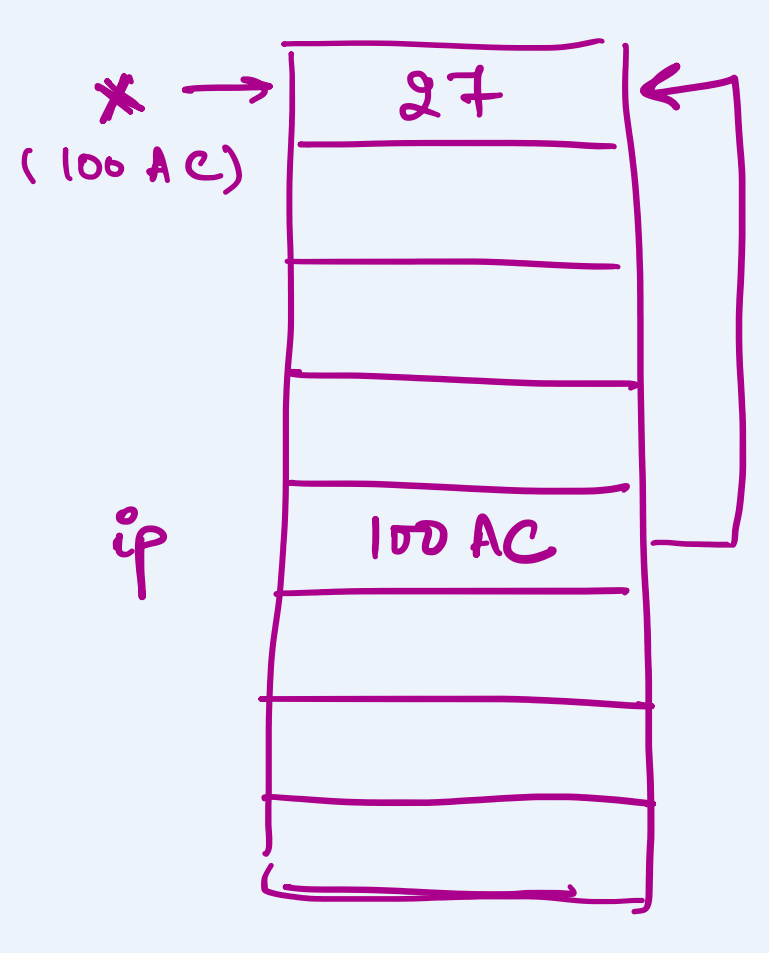
datatype variableName;

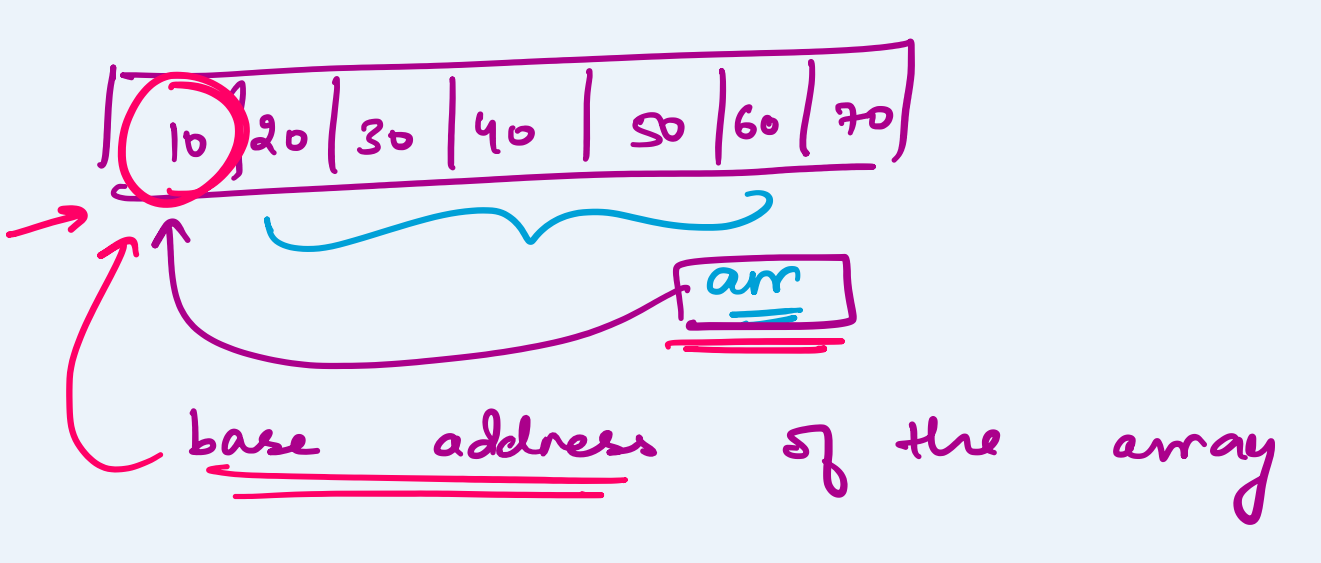
```
int *val ;
float * val ;
char * chs
```

& → Reference operator
→ returns the variable address

* → Deference operator
→ retus the value that has been stored in a memory address.

```
int x = 27.
int * ip;
ip = & x;
```





→ p = arr;

```
int arr [20];
int * ip;
ip = arr; x
```

NULL pointers

If there is no exact address to be assigned, then the pointer variable can be assigned to NULL.

Pointers of variables

→ manipulate data directly from computer memory.

→ Pointer variables point to a specific address in the computer's memory pointed to by another variable.

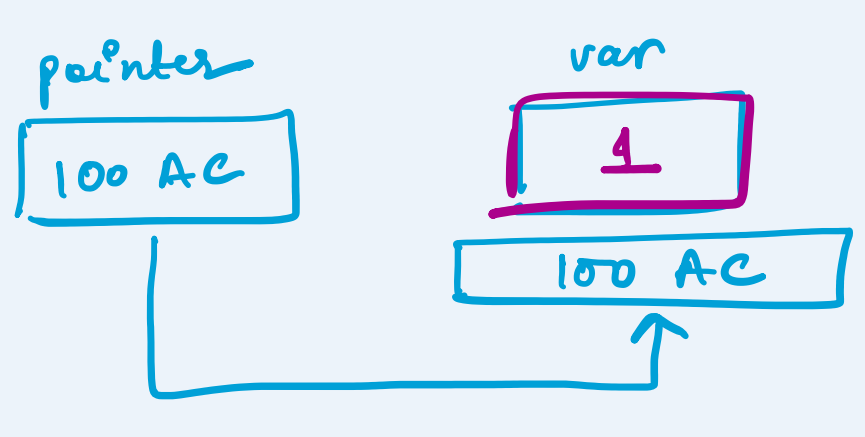
int *p;   int* p;

```
int * p;
x = 30;
p = &x;
cout << "value of x is" << *p;
```

ans → 30

```
int var= 5;
int * ip ;
ip= &var;
*ip = 1;
cout << var << endl;
```

→ 1

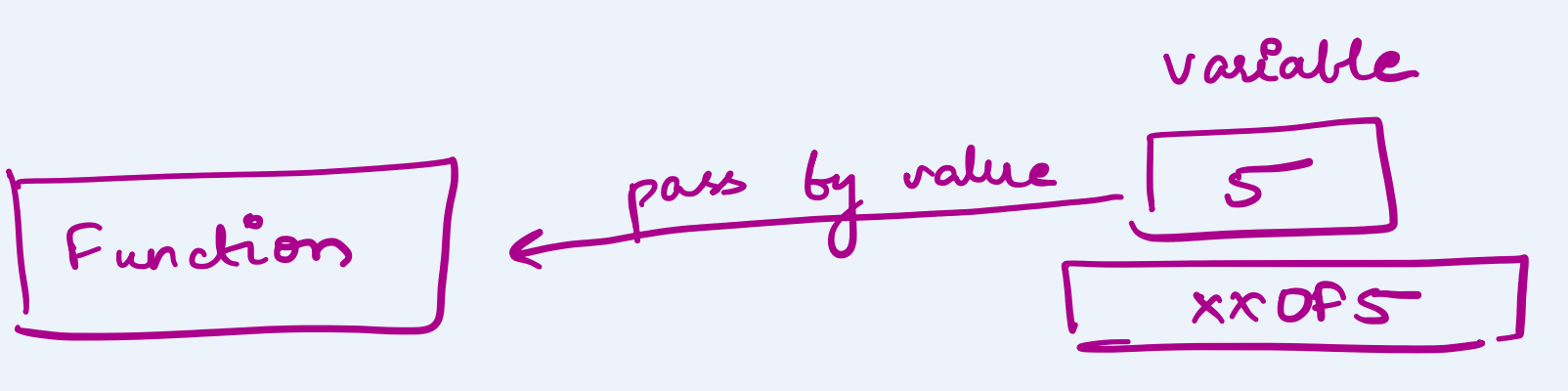


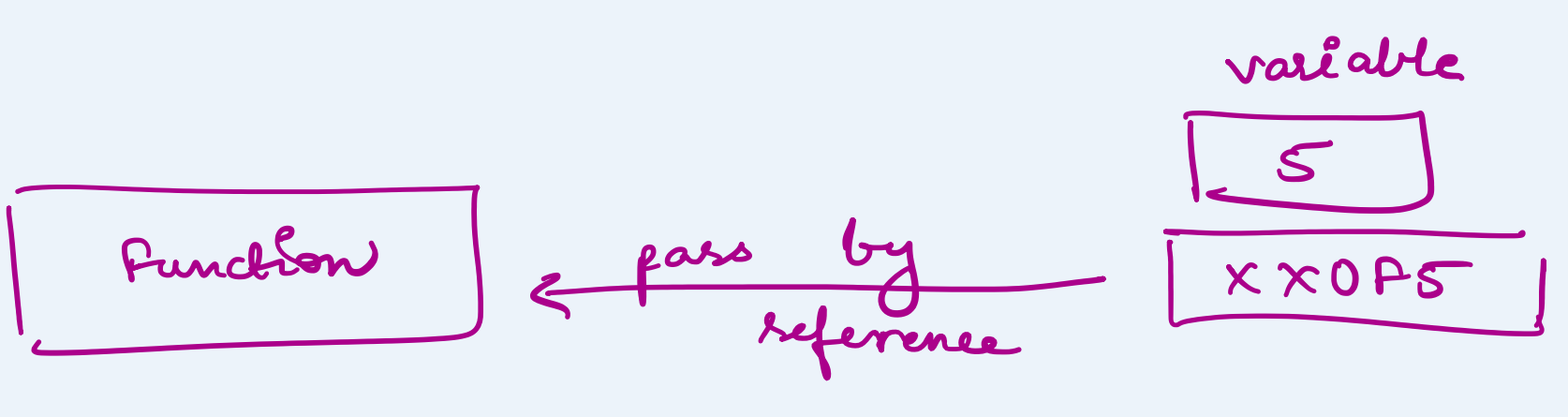
*a → 100 AC

Call by reference [using pointers]

pass by value | pass by reference

→ actual values of the arguements are not passed.
→ reference to the value is passed.





Dynamic memory allocation

→ allocation → new
→ deallocation → delete

new operator

→ allocates memory to a variable.

```
int * p;
p = new int;
*p = 45;
```

returns an address

pointerName = new datatype;

delete operator

delete pointerName;

```
int * p;
p= new int;
*p= 45;
delete p;
```

```
int * ptr;
int arr[5];
ptr = arr;
```

3rd ele

ptr	→	& arr[0]
ptr + 1	→	& arr [1]
ptr + 2	→	& arr [2]
ptr + 3	→	& arr [3]
ptr + 4	→	& arr [4]

*(ptr + 3)

ptr = & arr[2]

ptr - 2	→	& arr[0]
ptr - 1	→	& arr [1]
ptr + 1	→	& arr [3]
ptr + 2	→	& arr [4]