1- function prototype -signature (name ,parameters if exist,returntype-void)
//represent declaration for compiler //compile time

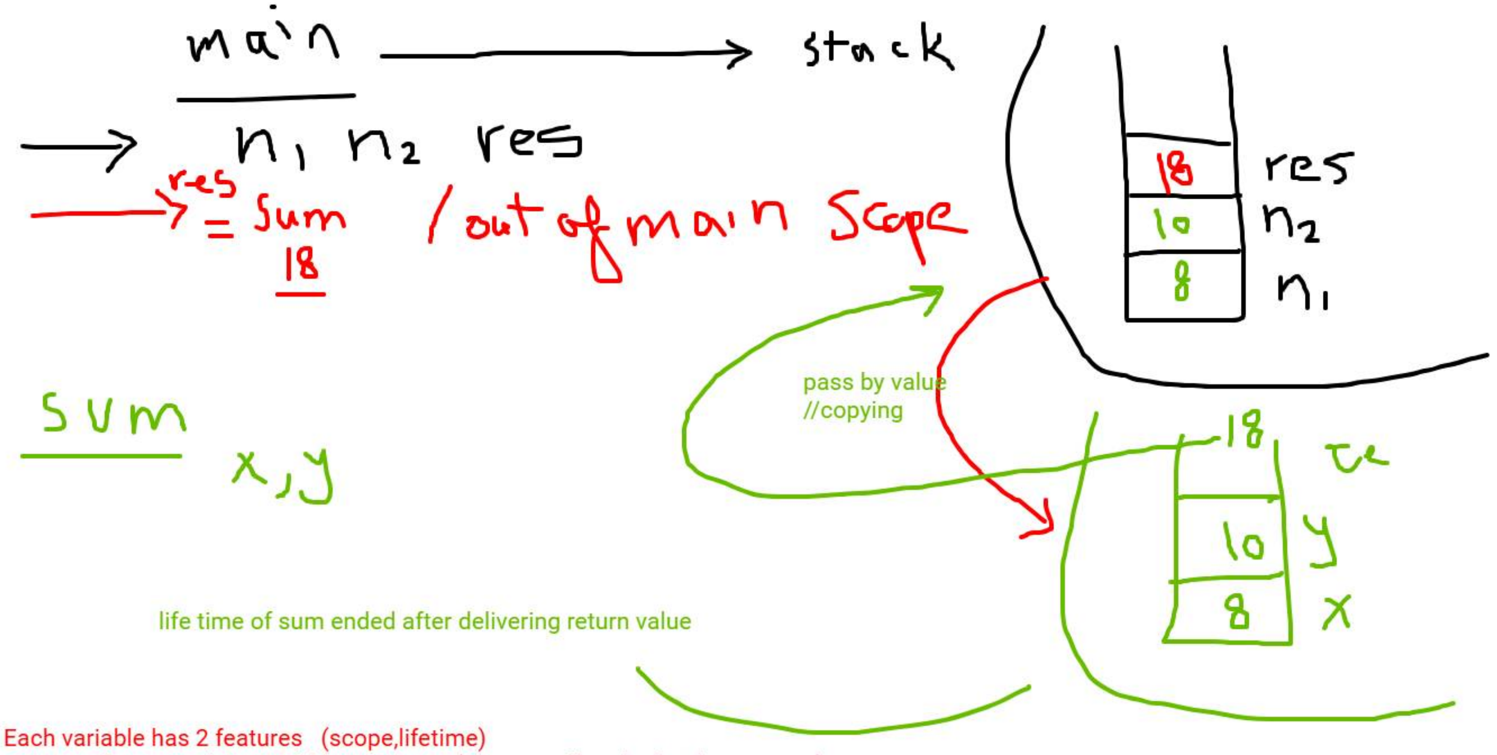
2-fn-Call inside main--->

3-Fn-implementation ---->signature with parameter names

ReturnType---->Void (doesn't return anything)
primitive types, struct types (fn doesn't return arrays)
Single value

Name--->represent Verb (verbal)

parameters---->any number, array, (by value, by ref)



Each variable has 2 features (scope,lifetime)
scope--->focus on this variable at run time (we can call or deal with var-name)
lifetime--->variable is removed from memory (pop from stack)

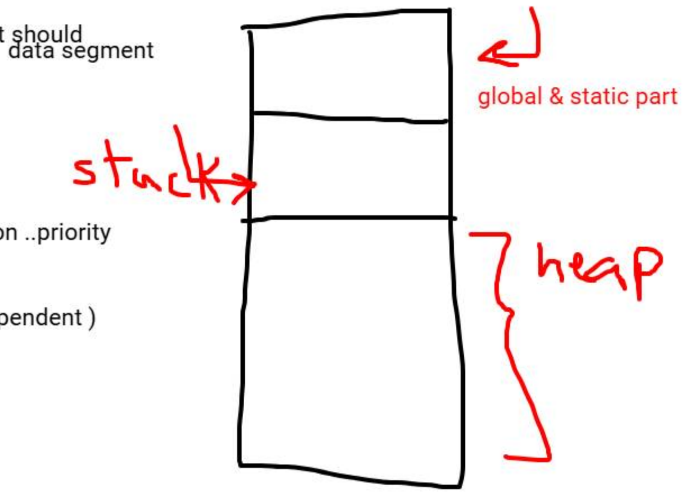
1- local variables ---scope (its function) life time with its function start and end.

Global Variable : variable declared before main or any other function that should work with this variable ...

scope :program itself (work ,seen in any other time)
lifetime :program itself (ended with end of program)

avoid using global variables (using const data or passing parameters)
//ambiguity   (if it has the same name as any local variable in any function ..priority goes for local)

//it allows dependence between functions (each function should be independent)



static local variable :declared inside function
initial value if first successive call for its function...

keep its value for each successive call

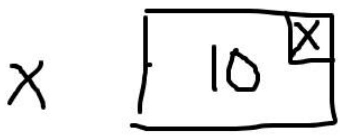
scope: inside function (can't be called outside this function)
lifetime :remain in memory all the program lifetime

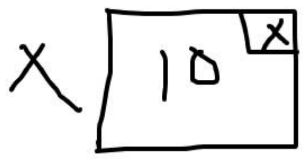
```
constexpr   type of modifiers --
const int x=10;              //     evaluate into runtime
constexpr int x=sum(5,6);//evaluate runtime 10;          //evaluate at compile time
consteval---->must run at compile time (error compiler) search??
int arr[x]={1,2};



//writing a function to run at compile time
constexpr int Multiply(int x,int y)
{ return x*y;}
int main()
{    //int n;  cin>>n;  //const in n=7;//run at runtime  constexpr int x=9;//at compiletime
    constexpr int res=Multply(7,x);    //evaluate at compile time(value evaluated from
compiletime)


}
```



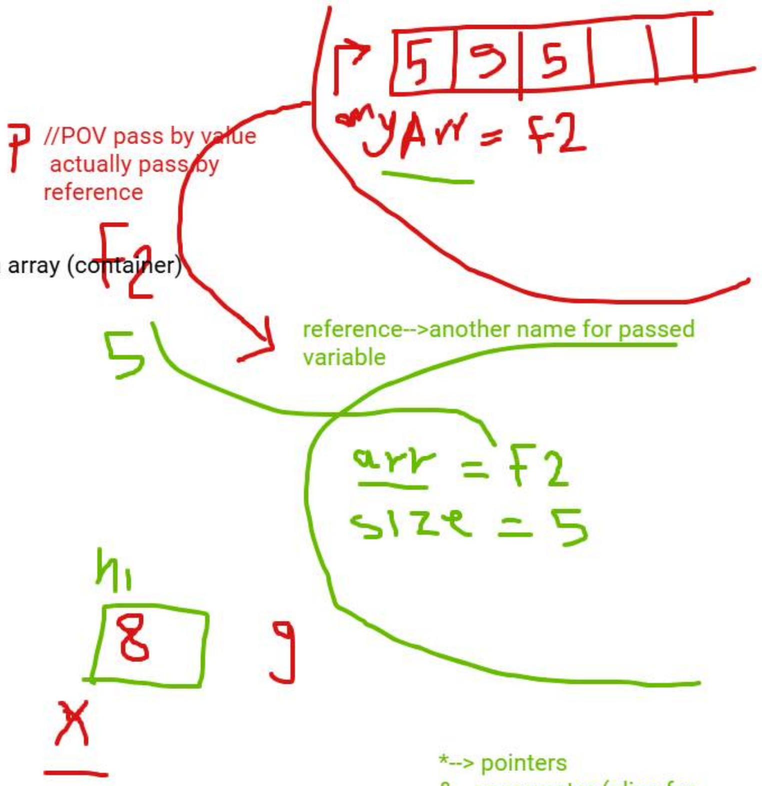


passing an array as a fn parameter.

```
int SumArr(const int [] arr,int size)
{   int sum=0;
    for(int i=0;i<size;i++)
      { sum+=arr[i];  }
     return sum;
}
int fillArr(int []arr,int size){cin>>arr[i];}//permit change main array (container)
int main()
{
     int myArr[5]={5,5,5,5,5};
     cout<<sumArr(myArr,5);
```

//POV pass by value
actually pass by
reference

reference-->another name for passed
variable



```
void takeinput(int &,int &);//pass by ref
int main()
{
    int n1,n2;
    takeinput(n1,n2);
    cout<<n1<<" and "<<n2<<endl;
 }
void takeinput(int &x,int &y){ cin>>x; cin>>y}
```

*--> pointers
&--->parameter (alias for
variable)

```
 int x=7;
cout<<&x;//show address of
var x in memory
```

Reference Type   :   when to use and when to avoid

// swap → by Ref

$\boxed{\text{As1}}$ ✓

Slicing--->std::subspan   SEarch
for it
As2-   replace values from index
0 to mid odf array by 1
mid to last array values by 0

```
int SumArr(std::span<int> arr)//safe access to passed arrays (support size tracking,slicing)
{   int sum=0;
   for(auto x: arr)//auto detect array size passed span from main fn //for (auto &x: arr) not read only can write too
      { sum+=x; }
    return sum;
}
int fillArr(int []arr,int size){cin>>arr[i];}//permit change main array (container)
int main()
{
    int myArr[5]={5,5,5,5,5};
    cout<<sumArr(myArr,5);
```

As 3-Recursive Function ---->inside Exam self study(Fibonacci ,print decimal value by its binary representation..

As4-connect Employee form with highlight menu (apply fn to employees then add to Menu)
Display --->show employee by index Display all-->show only employees that exist   New-->add by index
function that print menu ..
function that take choices from user // Main function has only Vr or Function names no cin or cout .....