Character Arrays & Pointers

Strings

How to Store Strings?

Size of Array ≥ no. of Characters in String + 1.

"John" → Size → 5

Let Char C[8]

C

J	O	H	N	\0			

↑ Null character

```
C[0] = 'J';
C[1] = 'O';
C[2] = 'H';
C[3] = 'N';
C[4] = '\0';
```

to know that

↓

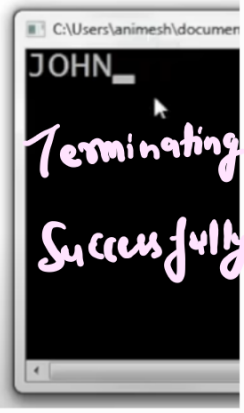
Now the String ends Here

Terminate String by a Null Character.

Use

include < String.h>

```
//character arrays and pointers
#include<stdio.h>
int main()
{
    char C[20];
    C[0] = 'J';
    C[1] = 'O';
    C[2] = 'H';
    C[3] = 'N';
    C[4] = '\0';
    printf("%s",C);
}
```



```
int len = strlen(C);   // 4
printf("Length = %d", len);
```

JOHN

```
Char C[20] = "JOHN"; // Null is always already stored.
```

```
X  C[4] = "JOHN"; // throws error
```

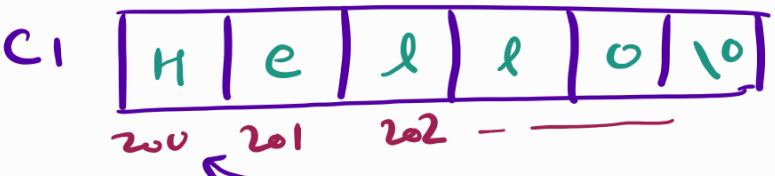
```
✓  C[5] = "JOHN"; // ✓
```

OR

```
C[5] = {'J', 'O', 'H', 'N', '\0'};
```

Arrays & Pointers are different types that are used in Similar Manner.

```
Char C1[6] = "Hello";
```




```
Char* C2;
```


```
C2 = C1;
```

→ Valid

Make C2 points to first element of Array

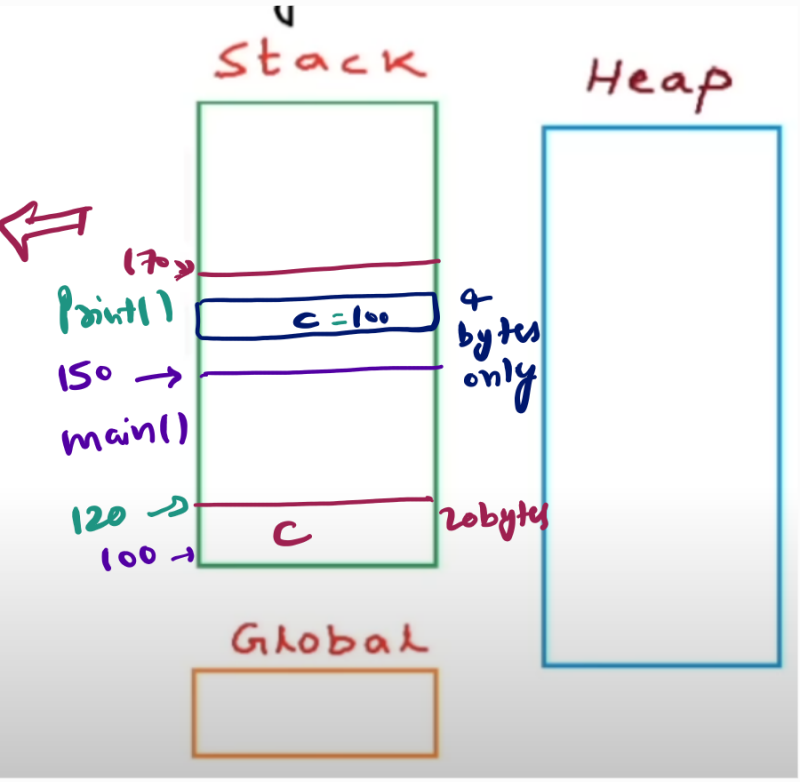
```
print C2[1]; // l
```

```
C2[0] = 'A'; // "Aello"
```

C2[i] is *(C2+i) // Same Syntaxes

C1[i] or *(C1+i)

```c
#include<stdio.h>
#include<string.h>
void print(char *C)
{
    while(*C != '\0')
    {
        printf("%c",*C);
        C++;
    }
    printf("\n");
}
int main()
{
    char C[20] = "Hello";
    print(C);
}
```



same var of type char in Main &
& type * char (Pointer to that char in func^n)

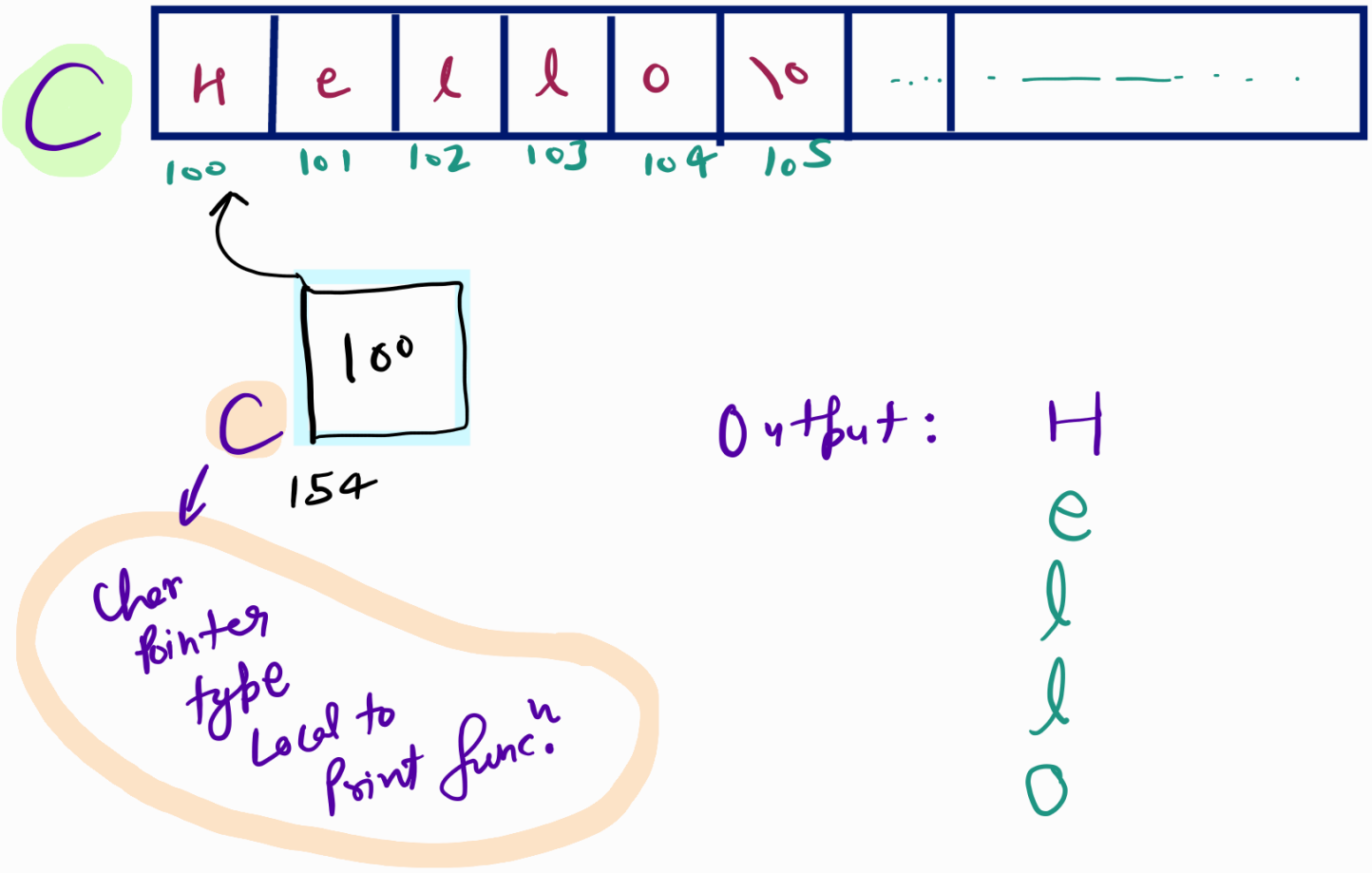
Value to char from main while logic —— .. from func^n.

C in main() & C in print() are different.

char *C

can be a or b anything

C of func^n will store Address 100 & stores it in pointer Variable.




```c
//character arrays and pointers
#include<stdio.h>
#include<string.h>
void print(char *C)
{
    while(*C != '\0')
    {
        printf("%c",*C);
        C++;
    }
    printf("\n");
}
int main()
{
    //char C[20] = "Hello"; // string gets stored in the space for array
    char *C = "Hello"; // string gets stored as compile time costant
    printf("Hello World");
    print(C);
}
```

If you print C[0] = A ; it will throw error

Now come back to char array

```c
int main()
{
    char C[20] = "Hello";
    print(C);
}
```

but ↓

```c
//character arrays and pointers
#include<stdio.h>
#include<string.h>
void print(char *C)
{
    C[0] = 'A'; //Already initialized or changed to A
    while(*C != '\0')
    {
        printf("%c",*C);
        C++;
    }
    printf("\n");
}
int main()
{
    char C[20] = "Hello";
    print(C);
}
```

So, output = Aello

If you just want to read a String & not write then ↓

using void print (const char *C)

↓

You can just read now & not write.

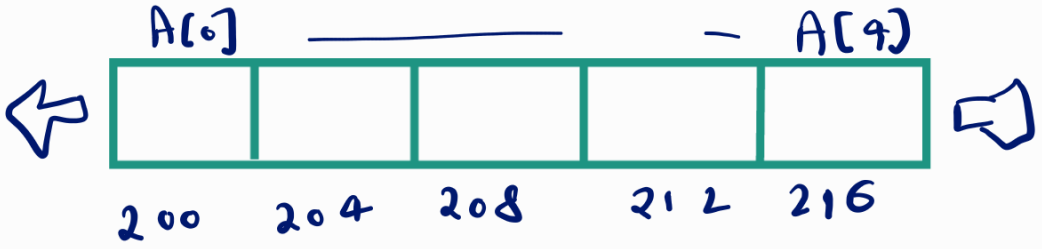
So, just by removing C[0] = A;

Code will work for Reading else it will throw error.

POINTERS & multi-dimensional Array

1D Array

int A[5];



`int *p = A;` //Dereferencing

Print A // 200
Print *A // 2
Print *(A+2) // 6

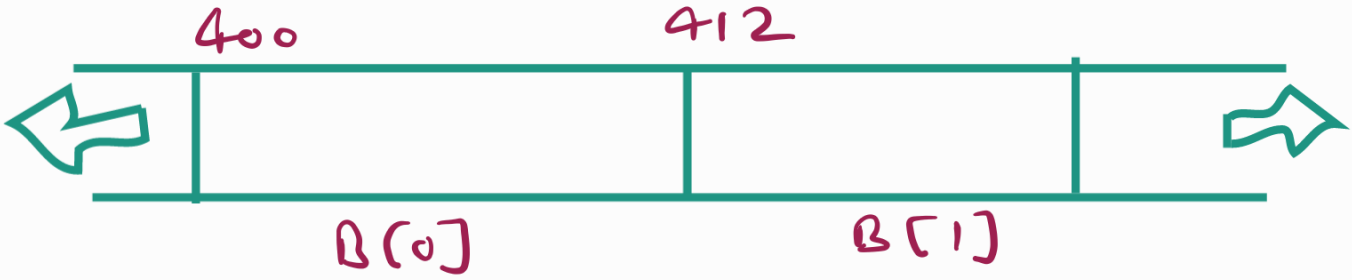
*(A+i) is same as A[i]
(A+i) ——— &A[i]

P = A; ✓
A = P; ✗ Compilation error.

int B[2][3]

B[0]
B[1]
B[2]

→ 1-D arrays of 3 integers each.



`int *p = B;` ✗ error

bcz B will return a POINTER to 1-D array of 3 integers

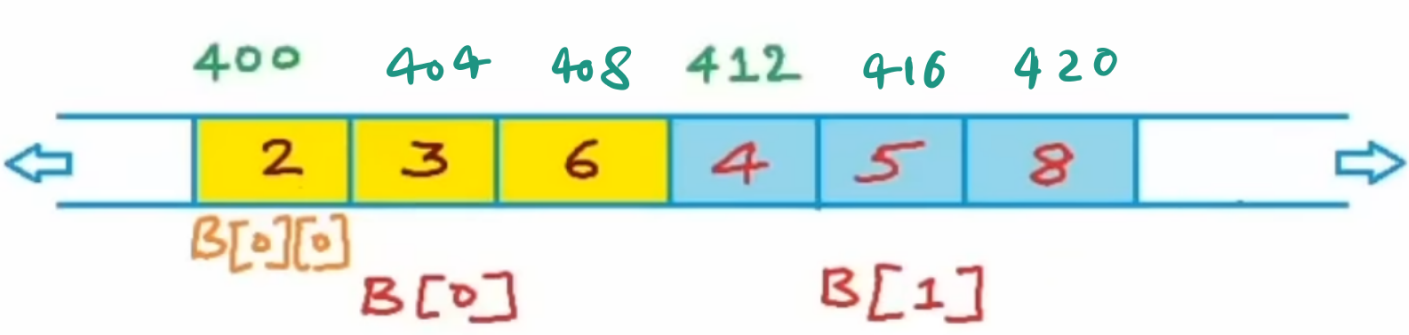
When will type of Pointer will matter?

↓

It matters @ when you dereference
(b) when you Pointer Arithmetic

Print B or &B[0]; // 400

Print *B or B[0] or &B[0][0]; // 400



Print B+1; // 400 + size of 1-D array of 3 integers (in bytes)

412

Print &B[1]

Print *(B+1) or B[1] or &B[1][0] // 412

↘ returning int *

Print *(B+1)+2 ; // 412+8 → 420

int *

Print B[1]+2 ;

Print *B[1][2] ;

Print *(*B+1)

*B[0][1]

B[0][1] = 3

B → int(*)[3]

B[0] → int *

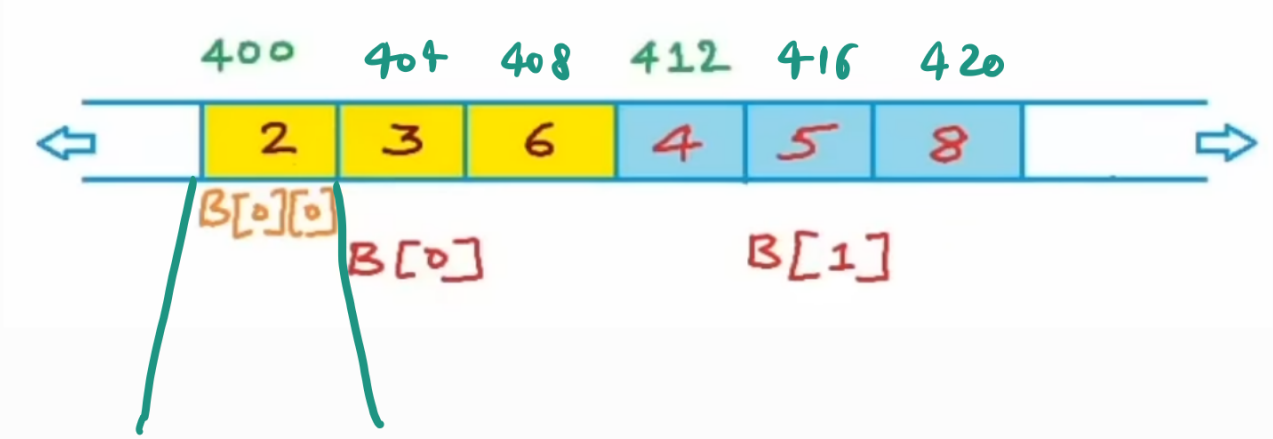
For 2D-Array

B[i][j] = *(B[i]+j)

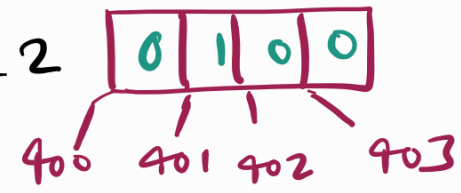
= *(*(B+i)+j)

Pointer with Multi-dimensional

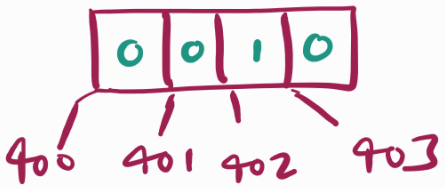
int B[2][3]



Value = 2



OR

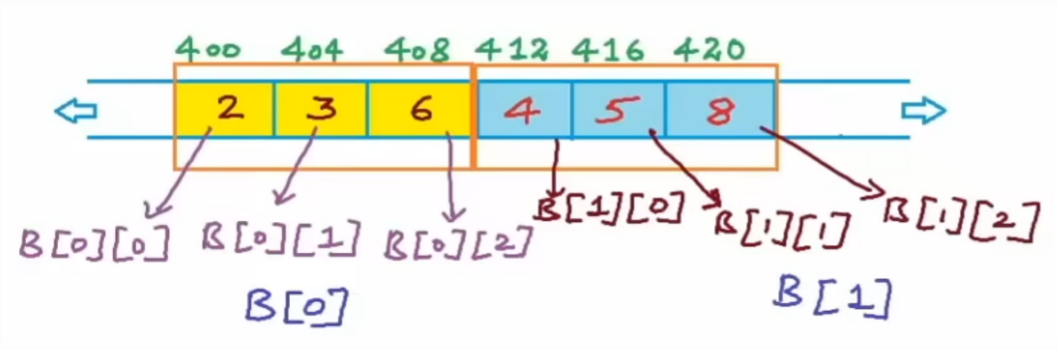


?

int B[2][3]

int (*P)[3] = B;

declaring Pointer to 1-D array of 3 integers.



○ int *P = B; ✗ bcz B will not return a Pointer to integer

B will return a Pointer to 1-D array of 3 integers.

```
Print  B   ; // 400
  "   *B   ; // 400
  "   B[0] ; // 400
      or
Print &B[0][0]; // 400
```

B[i][j] = *(B[i]+j)

& B[2][3] = *(B[2]+3);

Dereferencing to get Value of that integer.

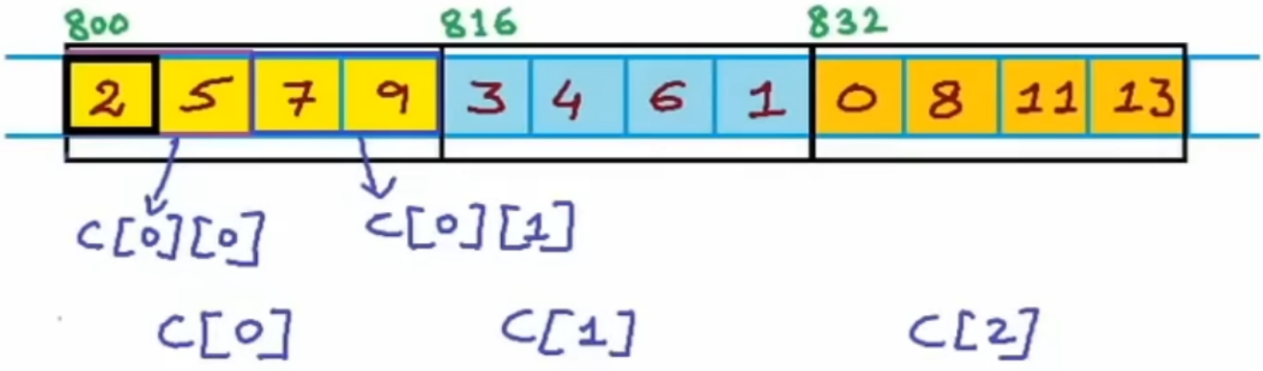
B[i][j] = *(B[i]+j) = *(*(B+i)+j)

B[i] : int *

*(B+i) : int *

3-D ARRAY

int C[3][2][2] → 3 - 2D Arrays of 2 elements each



int (*p)[2][2] = c;

Print c; // 800

↳ type → int *[2][2]

Print *c or c[0] or c[0][0]; // 800

↳ int (*)[2]

C[i][j][k] = *(C[i][j] + k) = *(*(c[i]+j)+k)

= *(*(*(c+i)+j)+k)

Print *(c[0][1] + 1) or c[0][1][1]; // 9

↓

(int*)

Print *(c[1]+1) or c[1][1] or c[1][1][0]; // 824 ?

int (*)[2]

```c
// Pointers and multi- dimensional arrays
#include<stdio.h>
int main()
{
    int C[3][2][2]={{{2,5},{7,9}},
                    {{3,4},{6,1}},
                    {{0,8},{11,13}}};
    printf("%d %d %d %d", C, *C, C[0], &C[0][0]);
}
```

C:\Users\animesh\Documents\Visual Studio 2010\Projects\SampleApp5\Debug\SampleApp5.exe

4192172

All will give address from where the Array starts.

C = *C = C[0] , & C[0][0]

or

C[0][0]
↓
C[0]
↓
*C
↓
C


```c
// Pointers and multi- dimensional arrays
#include<stdio.h>
int main()
{
    int C[3][2][2]={{{2,5},{7,9}},
                    {{3,4},{6,1}},
                    {{0,8},{11,13}}};
    printf("%d %d %d %d", C, *C, C[0], &C[0][0]);
}
```


C:\Users\animesh\Documents\Visual Studio 2010\Projects\SampleApp5\Debug\SampleApp5.exe

print ("%d \n", *([0][0]+1));

C[0][0][1] = 5

```c
// Pointers and multi- dimensional arrays
#include<stdio.h>
void Func(int (*A)[3]) // Argument: 2-D array of integers
{
}
int main()
{
    int C[3][2][2]={{{2,5},{7,9}},
                    {{3,4},{6,1}},
                    {{0,8},{11,13}}};
    int A[2] = {1,2};
    int B[2][3] ={{2,4,6},{5,7,8}}; // B returns int (*)[3]
    Func(A);
}
```

use int A[][3]

Let's declare here: X[2][4];

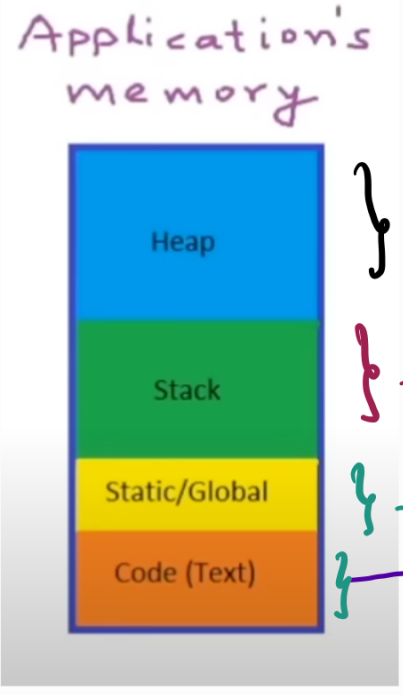
Func(X); // X returns int (*)[4]

If X[5][3];

Func(X); // X will be fine in this case.

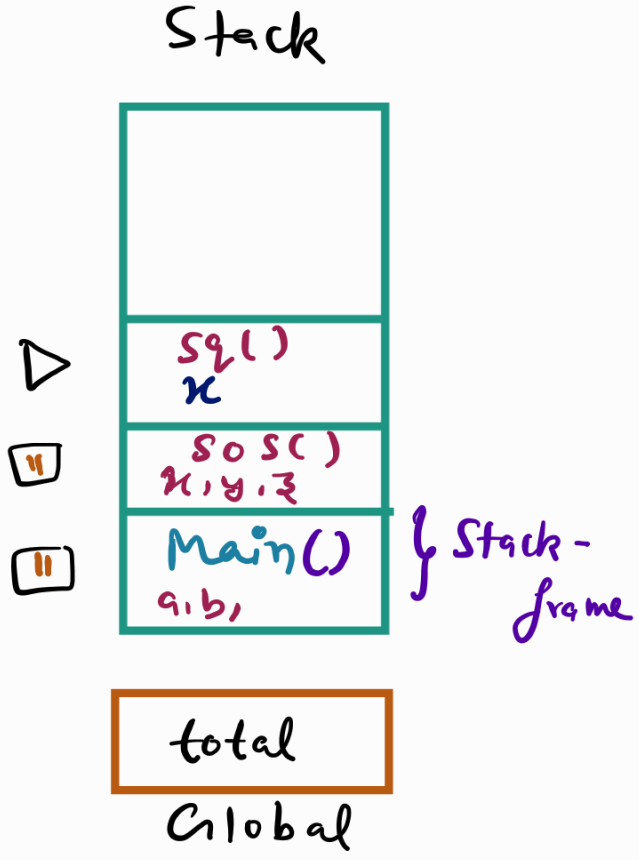
https://youtu.be/zuegQmMdy8M?si=HoPv6E6F0sQ8FlsP&t=7885 } Refer this

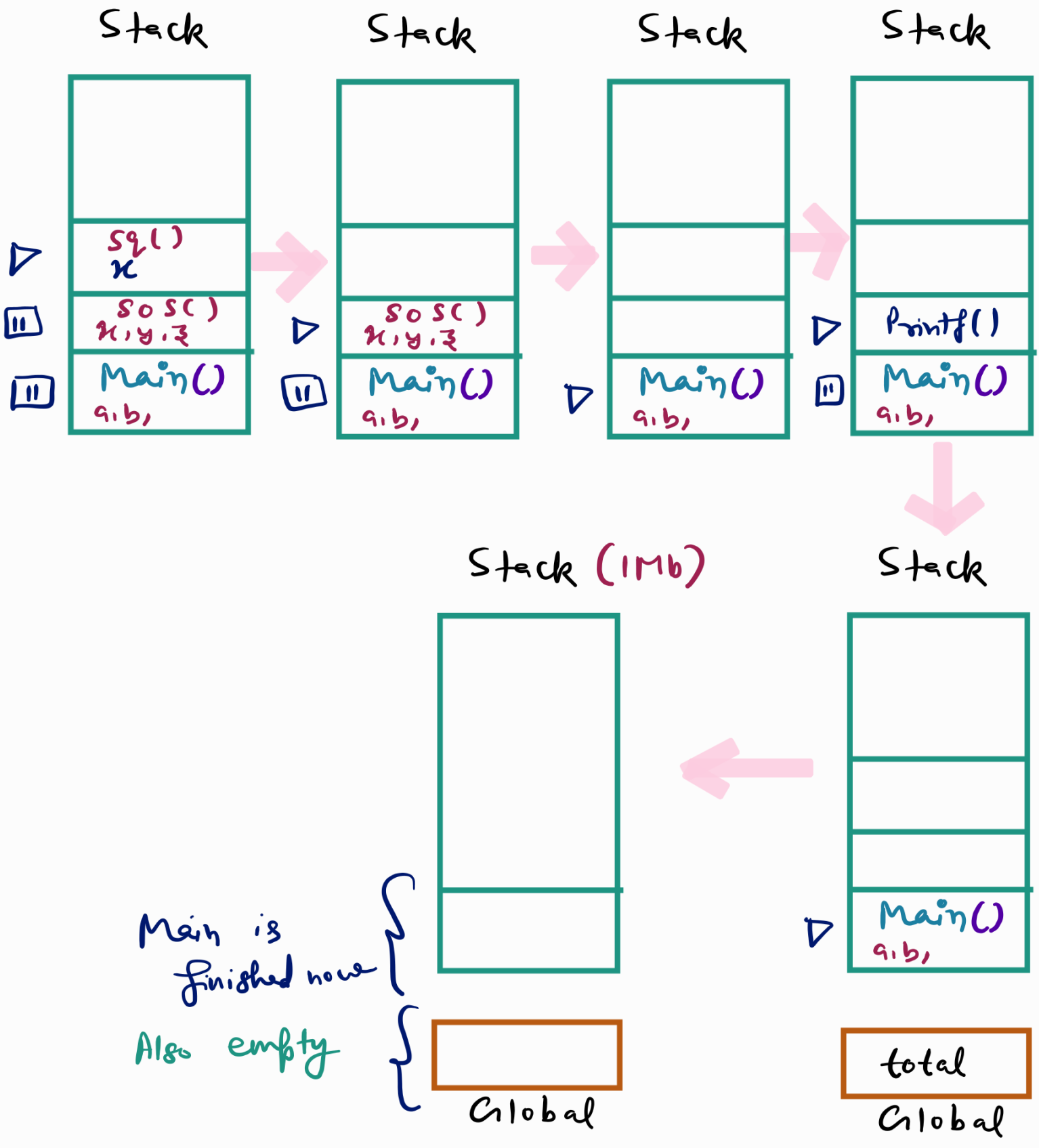
POINTERS & Dynamic Memory




```c
#include<stdio.h>
int total;
int Square(int x)
{
    return x*x;
}
int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return z; // (x+y)2
}
int main()
{
    int a = 4, b = 8;
    total = SquareOfSum(a,b);
    printf("output = %d",total);
}
```

SoS( )




Stack
Stack
Stack
Stack
Sq()
x
SOS()
x,y,z
Main()
a,b,
SOS()
x,y,z
Main()
a,b,
Main()
a,b,
Printf()
Main()
a,b,
Stack (1Mb)
Stack
Main()
a,b,
Main is finished now
Also empty
Global
total
Global

Memory Exhausted

Stack Overflow

↓

Possible Reasons

↓

Bad Recursions