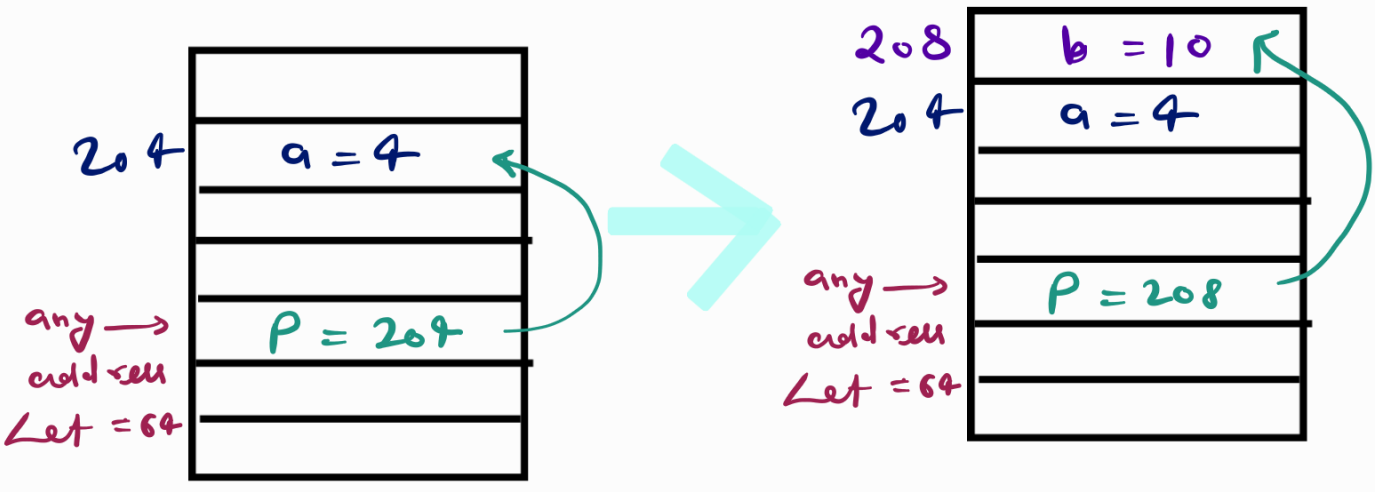
POINTERS

Bytes
int - 4
float - 4
char - 1

Basics

Stores address of another Variable




```
int a;
int* p;   or   int *p;
p = &a; →// address of a
a = 5;  //  ——————  ..  = 204
Print p → // 204
Print &a →// 204
print &p → // any address p is being Stored
print *p → // value @204 = 5
```

Dereferencing

You can also Change value of a

```
*p = 8;
print a;  // a = 8 now
```

Working with Pointers :

variables that store address of other variables

```
int a; //integer
int *p; //Pointer to integer
char c; // character
char *po //pointer to character
double d  //double
double *p1 //Pointer to double
```

Diff. type of Pointers

```
p = &a;
a = 8 ;
```

(*p) → 8

Some Examples

```
int a ;
int *p;
 p = &a;
print p ;  //  address of a
  "  *p ;  //  Value of a / Garbage Value
print &a ;  //  address of a
```

Modify Value of a Using pointer

*p = 12;

Value @ address where p is referencing = 12

DeReferencing → Access / Modify Value

```
int a;
 a = 5;
int *p;
 p = &a;
print p;   // address of a
 "  *p;   // a = 5
print &a;  // address of a
*p = 12;
print a;   // a = 12
```

```c
#include<stdio.h>
int main()
{
    int a;  int *p;
    a = 10;
    p = &a; // &a = address of a
    printf("Address of P is %d\n",p);
    printf("Value at p is %d\n",*p);
    int b = 20;
    *p = b; // Will the address in p change to point b
    printf("Address of P is %d\n",p);
    printf("Value at p is %d\n",*p);
}
```

Address of a
a = 10

address of a
value of a = 20

```
int a = 10
int *p;
p = &a;
Print p ; // address of a = 2002
Print p+1 ; //  ____  = 2006
___ p+2; //  ____  = 2010
```

Let
+4
+4

```
#include<stdio.h>
int main()
{
    int a = 10;
    int *p;
    p = &a;
    // Pointer arithmetic
    printf("Address p is %d\n",p); // address of a = 2004
    printf("value at address p is %d\n",*p);
    printf("size of integer is %d bytes\n",sizeof(int));
    printf("Address p+1 is %d\n",p+1);
    printf("value at address p+1 is %d\n",*(p+1));
}
```

Let
a = 10
4
= 2008
= Any Value (Garbage Value)

Pointer Types, Void Pointer

Pointer Arithmetic

int a = 1025


```
int *p;
p = &a ;
Print p ; // 200
```

point *p; // → 1025

```c
int a = 1025;
int *p;
p = &a;
printf("size of integer is %d bytes\n",sizeof(int));
printf("Address = %d, value = %d\n",p,*p);
char *p0;
p0 = (char*)p; // typecasting
printf("size of char is %d bytes\n",sizeof(char));
printf("Address = %d, value = %d\n",p0,*p0);
```

4

Any address, 1025

Let 2004

*

1

Same Prev. Address → 2004, 1

0000 0001

Char typecasting will lead to leaving all other 3 bytes

Only 1st byte will come.

```c
int a = 1025;
int *p;
p = &a;
printf("size of integer is %d bytes\n",sizeof(int));
printf("Address = %d, value = %d\n",p,*p);
printf("Address = %d, value = %d\n",p+1,*(p+1));
char *p0;
p0 = (char*)p; // typecasting
printf("size of char is %d bytes\n",sizeof(char));
printf("Address = %d, value = %d\n",p0,*p0);
printf("Address = %d, value = %d\n",p0+1,*(p0+1));
// 1025 = 00000000 00000000 00000100 00000001
```

4

2004, 1025

2008

Garbage Value

1

2004, 1

0 0 4 1

p+1 means 2005

*(p+1) means 4

```
po = (char*) p;
```

Make int type p to char type & then store in po.

Generic Pointer type: Void

```
po = p; // Valid here
```

↳ We can't dereference this means *po → throw error.

You can only print Address of Void Pointer.

Pointers to Pointers

```
int x = 5;
int *p;
p = &x; // p points to x;
*p = 6; // x = 6 now
```

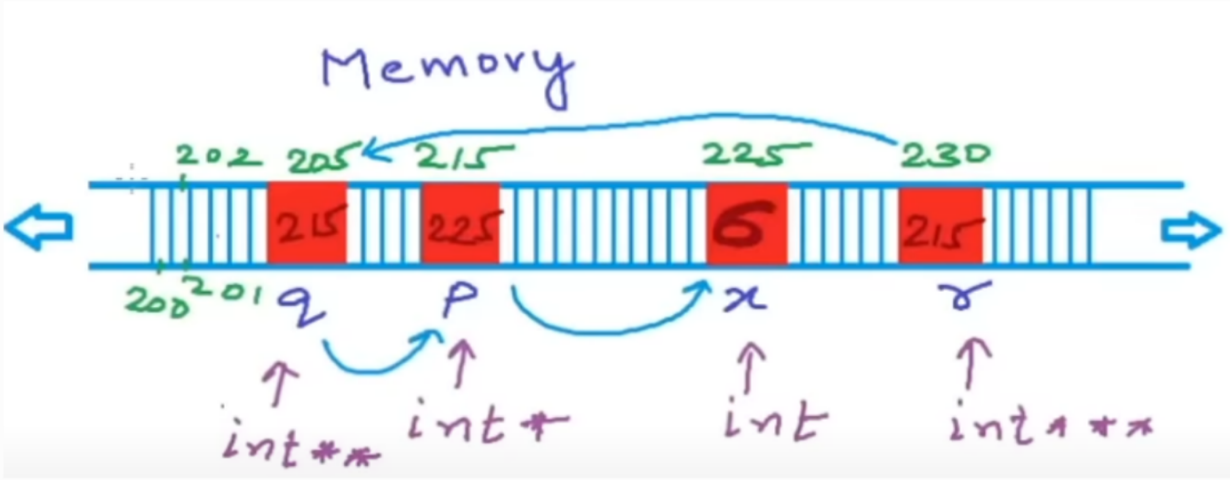
Let 225

```
int **q;
q = &p; // q points to Pointer p
```

225

itself @ 205

** because q is pointing to pointer of x mainly → So, q is pointing to x indirectly.

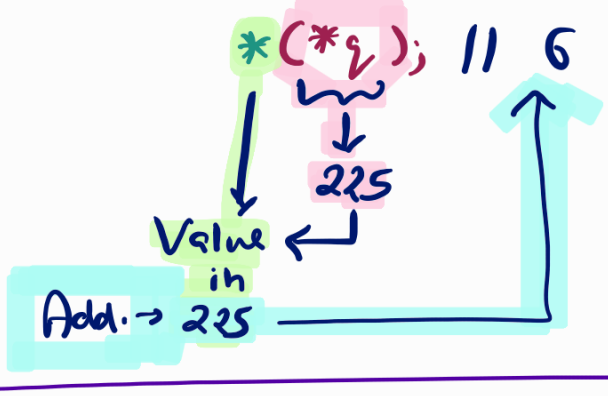



```c
#include<stdio.h>
int main()
{
    int x = 5;
    int* p = &x;
    *p = 6;
    int** q = &p;
    int*** r = &q;
}
```

Print *p; // 6

" *q; // = &x = 225

*(*q); // 6



Print *(*r);

→ 225

Print *(*(*r); → 6

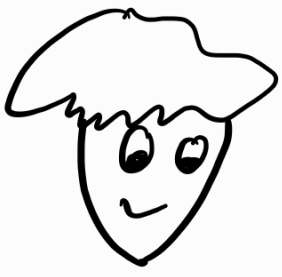
```c
// pointers to pointers
#include<stdio.h>
int main()
{
    int x = 5;
    int* p = &x;
    *p = 6;
    int** q = &p;
    int*** r = &q;
    printf("%d\n",*p);
    printf("%d\n",*q);
    printf("%d\n",**q);
    printf("%d\n",**r);
    printf("%d\n",***r);
    ***r = 10;
    printf("x = %d\n",x);
}
```

```
**q = *p + 2;
printf("x = %d\n", x);
```

```
C:\Users\animesh\do
6
4193620
6
4193620
6
x = 10
```

x = 12

Pointers as Func^n Arguments – Call by Reference



Albert forgets

Local Variable

Main

```c
#include<stdio.h>
void Increment(int a)
{
    a = a+1;
}
int main()
{
    int a;
    a = 10;
    Increment(a);
    printf("a = %d",a);
}
```

Function of type Void

x

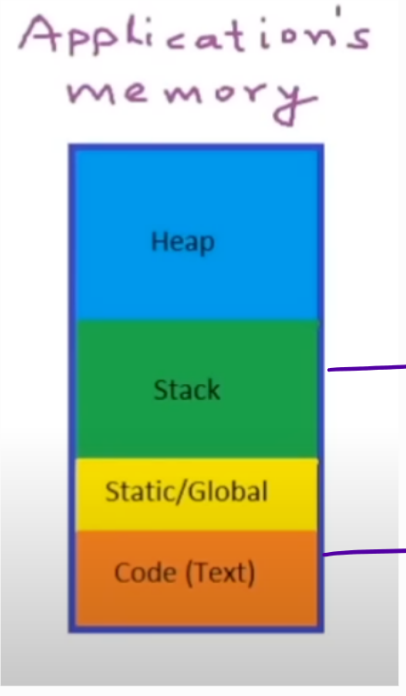
// a = 10

```c
#include<stdio.h>
void Increment(int a)
{
    a = a+1;
    printf("Address of variable a in increment = %d",&a);
}
int main()
{
    int a;
    a = 10;
    Increment(a);
    printf("Address of variable a in main = %d",&a);
    //printf("a = %d",a);
}
```

Let 4454460

Local Variables go in STACK

Here also 4454460




```c
#include<stdio.h>
void Increment(int a)
{
    a = a+1;
}
int main()
{
    int a;
    a = 10;
    Increment(a);
    printf("a = %d",a);
}
```

called func^n

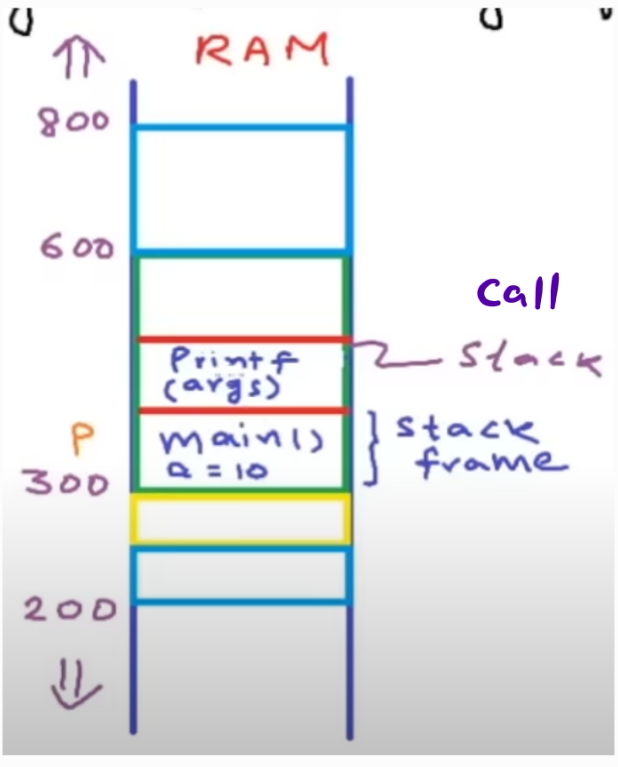
Formal Argument

calling func^n

Actual Argument

Lifetime of a Local Variable is till the funct. is executing.

Actual Argument is Mapped to formal Argument.



If we've x here then a will be copied to x.

```c
#include<stdio.h>
void Increment(int a)
{
    a = a+1;
}
int main()
{
    int a;
    a = 10;
    Increment(a);
    printf("a = %d",a);
}
```

x

x = x+1;

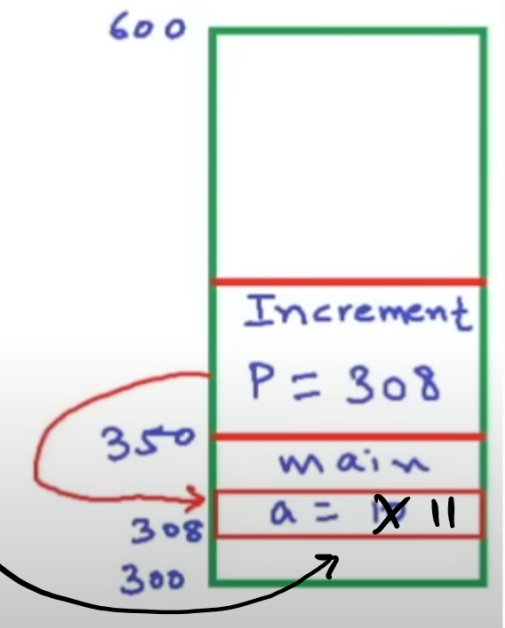
a → x

{Call by Value} will not work.

```c
#include<stdio.h>
void Increment(int *p)
{
    *p = (*p) + 1;
}
int main()
{
    int a;
    a = 10;
    Increment(&a);
    printf("a = %d",a);
}
```

↑ by 1

a=11 Now



This is Call by Reference.

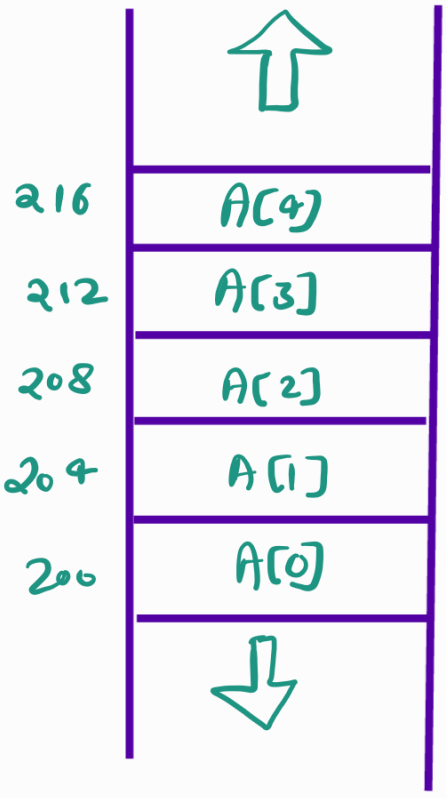
Pointers & Arrays :

```
int A[5];
```

A[0]
A[1]
A[2]
A[3]
A[4]

int → 4 bytes

A → 5 × 4
= 20 Bytes

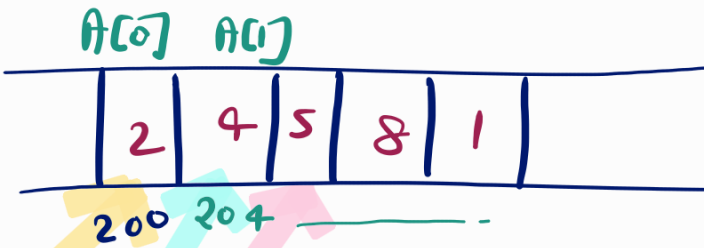


Let `int x = 5;` → @ Add ↓ 300

```
int *p;
p = &x ;//   300
Print *p ;//  5
p = p+1 ;  //  304
```

Print *p; // ? We don't Know

```
int A[5];
int *P
P = &A[0];
Print P   ;//200
Print *p; // 2
Print p+1; //204
Print *(p+1); // 4
Print *(p+2); // 5
```



Element at index i –

Address – &A[i] or (A+i)

Value – A[i] or *(A+i)

```c
// Pointers and Arrays
#include<stdio.h>
int main()
{
    int A[] ={2,4,5,8,1};
    int i;
    for(int i = 0;i<5;i++)
    {
        printf("Address = %d\n",&A[i]);
        printf("Address = %d\n",A+i);
        printf("value = %d\n",A[i]);
        printf("value = %d\n",*(A+i));
    }
}
```

How ?

```c
int A[] ={2,4,5,8,1};
int i;
int *p = A;
A++;// invalid
p++ // valid
for(int i = 0;i<5;i++)
{
    printf("Address = %d\n",&A[i]);
    printf("Address = %d\n",A+i);
    printf("value = %d\n",A[i]);
    printf("value = %d\n",*(A+i));
```

Don't increase Array A

→ only ↑ pointer

Arrays @ function Arguments:

```c
// Arrays as function arguments
#include<stdio.h>
int SumOfElements(int A[], int size)
{
    int i, sum = 0;
    for(i = 0;i< size;i++)
    {
        sum+= A[i];
    }
    return sum;
}
int main()
{
    int A[]= {1,2,3,4,5};
    int size = sizeof(A)/sizeof(A[0]);
    int total = SumOfElements(A,size);
    printf("Sum of elements = %d\n",total);
}
```

A[] size

sum ⟵ total

A[] size total in which we are using both of our Common Variables or Arguments

Total in main func^n will take Final Result from Sum from defined func^n

Sum will take Values from Main & Logic from defined func^n SumOfElements

Size of A in → SoE = 4

→ Main = 20 Why?

Array are being called by Reference bcz they are Big

Calling by value means creating a duplicate of original everytime.

```c
#include<stdio.h>
int SumOfElements(int A[])
{
    int i, sum = 0;
    int size = sizeof(A)/sizeof(A[0]);
    printf("SOE - Size of A = %d, size of A[0] = %d\n",sizeof(A),sizeof(A[0]));
    for(i = 0;i< size;i++)
    {
        sum+= A[i];
    }
    return sum;
}
int main()
{
    int A[]= {1,2,3,4,5};
    int total = SumOfElements(A);
    printf("Sum of elements = %d\n
    printf("Main - Size of A = %d, size of A[0] = %d\n",sizeof(A),sizeof(A[0]));
}
```

A here is a pointer to integer

while here A is array

```
C:\Users\animesh\documents\visual studio 2010\Projects\SampleApp2\Debug\SampleApp2.exe
SOE - Size of A = 4, size of A[0] = 4
Sum of elements = 1
Main - Size of A = 20, size of A[0] = 4
```

is wrong

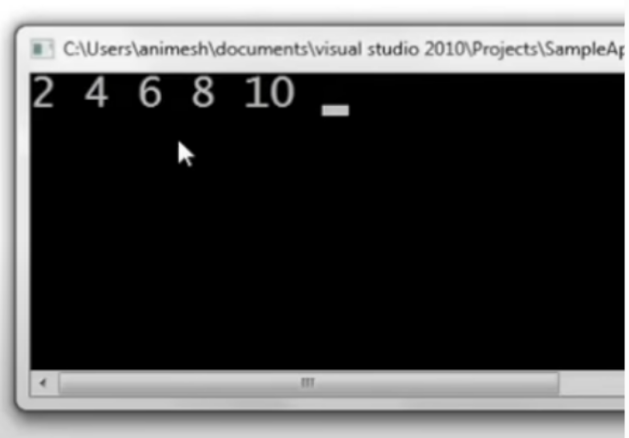
```c
#include<stdio.h>
int SumOfElements(int* A, int size)// "int* A" or "int A[]"  ..it's the same..
{
    int i, sum = 0;
    printf("SOE - Size of A = %d, size of A[0] = %d\n",sizeof(A),sizeof(A[0]));
    for(i = 0;i< size;i++)
    {
        sum+= A[i];
    }
    return sum;
}
int main()
{
    int A[]= {1,2,3,4,5};
    int size = sizeof(A)/sizeof(
    int total = SumOfElements(A,size);
    printf("Sum of elements = %d\n",total);
    printf("Main - Size of A = %d, size of A[0] = %d\n",sizeof(A),sizeof(A[0]));
}
```

```
C:\Users\animesh\documents\visual studio 2010\Projects\SampleApp2\Debug\SampleApp2.exe
SOE - Size of A = 4, size of A[0] = 4
Sum of elements = 15
Main - Size of A = 20, size of A[0] = 4
```

is correct

```c
#include<stdio.h>
int SumOfElements(int* A, int size)// "int* A" or "int A[]"  ..it's the same..
{
    int i, sum = 0;
    printf("SOE - Size of A = %d, size of A[0] = %d\n",sizeof(A),sizeof(A[0]));
    for(i = 0;i< size;i++)
    {
        sum+= A[i]; // A[i] is *(A+i)
    }
    return sum;
}
int main()
{
    int A[]= {1,2,3,4,5};
    int size = sizeof(A)/sizeof(A[0]);
    int total = SumOfElements(A,size); // A can be used for &A[0]
    printf("Sum of elements = %d\n",total);
    printf("Main - Size of A = %d, size of A[0] = %d\n",sizeof(A),sizeof(A[0]));
}
```

```c
#include<stdio.h>
void Double(int* A, int size)// "int* A" or "int A[]"  ..it's the same..
{
    int i, sum = 0;
    for(i = 0;i< size;i++)
    {
        A[i] = 2*A[i];
    }
}
int main()
{
    int A[]= {1,2,3,4,5};
    int size = sizeof(A)/sizeof(A[0]);
    int i;
    Double(A,size);
    for(i = 0;i< size;i++)
    {
        printf("%d ",A[i]);
    }
}
```



Double in main will take final result from double func^n!

by taking Value from Main func^n & logic from double func^n!