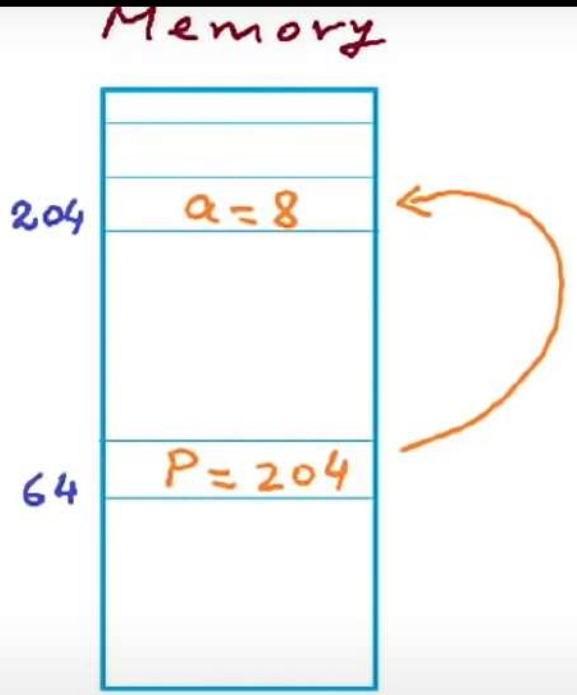
Pointers - variables that store address of another variable

```
int a;
int *p;
p = &a;
a = 5;
Print p      // 204
Print &a     // 204
Print &p     // 64
print *p     // 5   ⇒ derefrencing
*p = 8
Print a      // 8
```



```c
#include<stdio.h>
int main()
{
    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)
    // 1025 = 00000000 00000000 00000100 00000001
}
```

00000001 | 1

Pointers in C / C++ [Full Course]

```c
#include<stdio.h>
int main()
{
    int a = 1025;
    int *p;
    p = &a;
    printf("size of 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
}
```

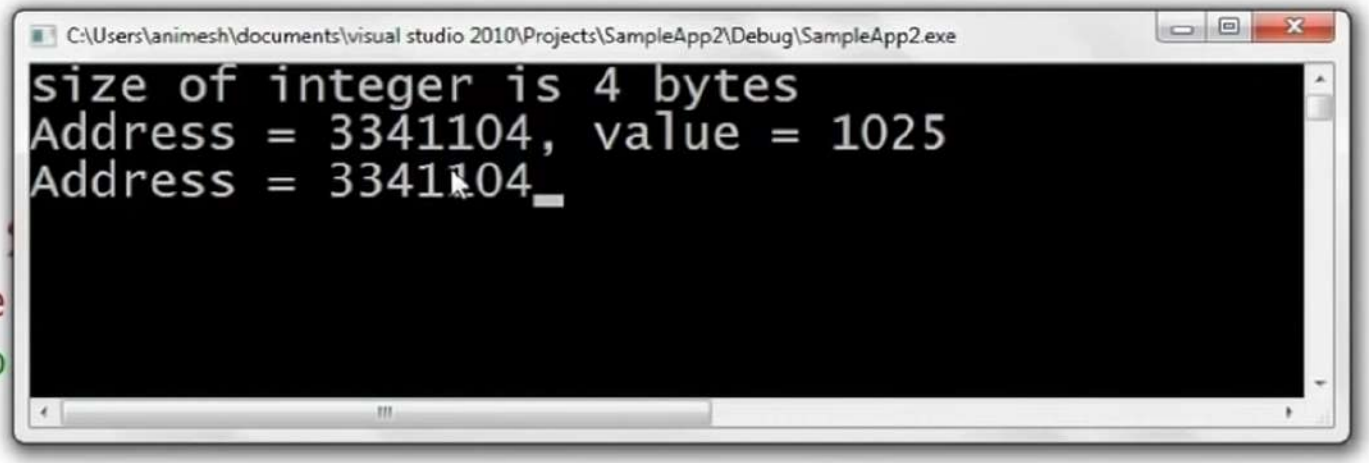
```
C:\Users\animesh\documents\visual studio 2010\Projects\SampleApp2\Debug\SampleApp2.exe
size of integer is 4 bytes
Address = 4456036, value = 1025
Address = 4456040, value = -858993460
size of char is 1 bytes
Address = 4456036, value = 1
Address = 4456037, value = 4
```

30:17 / 3:47:22 • Pointer types, pointer arithmetic, void pointers

```c
#include<stdio.h>
int main()
{
    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);
    // Void pointer - Genric pointer
    void *p0;
    p0 = p;
    printf("Address = %d, value = %d\n",p0,*p0);
}
```

Error: expression must be a pointer to a complete object type

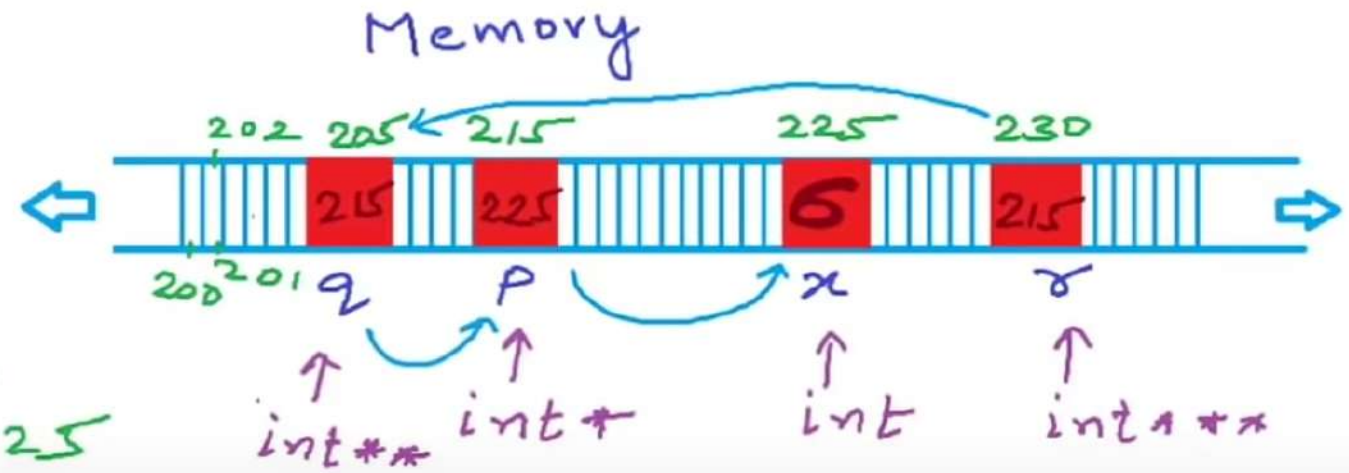
```c
#include<stdio.h>
int main()
{
    int a = 1025;
    int *p;
    p = &a;
    printf("size of integer is
    printf("Address = %d, value
    // Void pointer - Genric po
    void *p0;
    p0 = p;
    printf("Address = %d",p0);
}
```



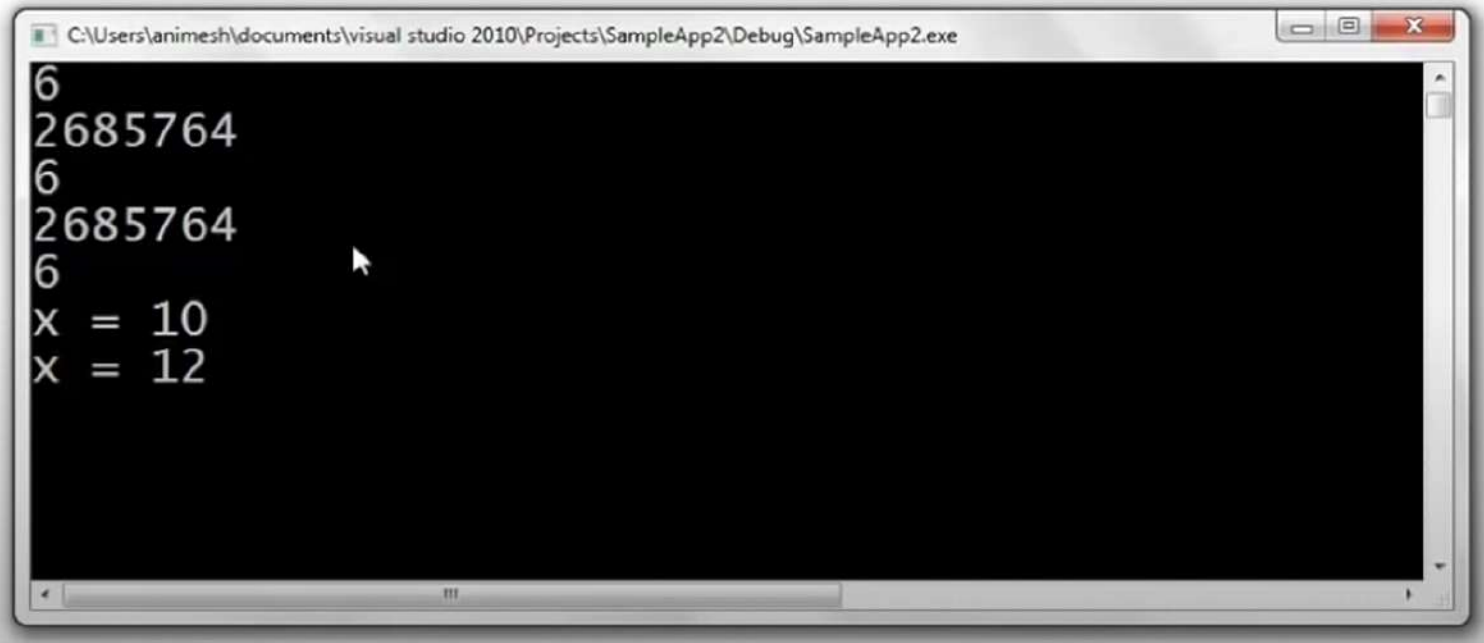
```c
#include<stdio.h>
int main()
{
    int a = 1025;
    int *p;
    p = &a;
    printf("size of integer is %d bytes\n",sizeof(int));
    printf("Address = %d, value = %d\n",p,*p);
    // Void pointer - Genric pointer
    void *p0;
    p0 = p;
    printf("Address = %d",p0);
    printf("Address = %d",p0+1);
}
```

```c
#include<stdio.h>
int main()
{
    int x = 5;
    int* p = &x;
    *p = 6;
    int** q = &p;
    int*** r = &q;
    printf("%d\n",*p); // 6
    printf("%d\n",*q); // 225
    printf("%d\n",*(*q)); // 6
    printf("%d\n", *(*r)) ; // 225
    printf("%d\n",*(*(*r))); // 6
}
```

Pointer to pointer

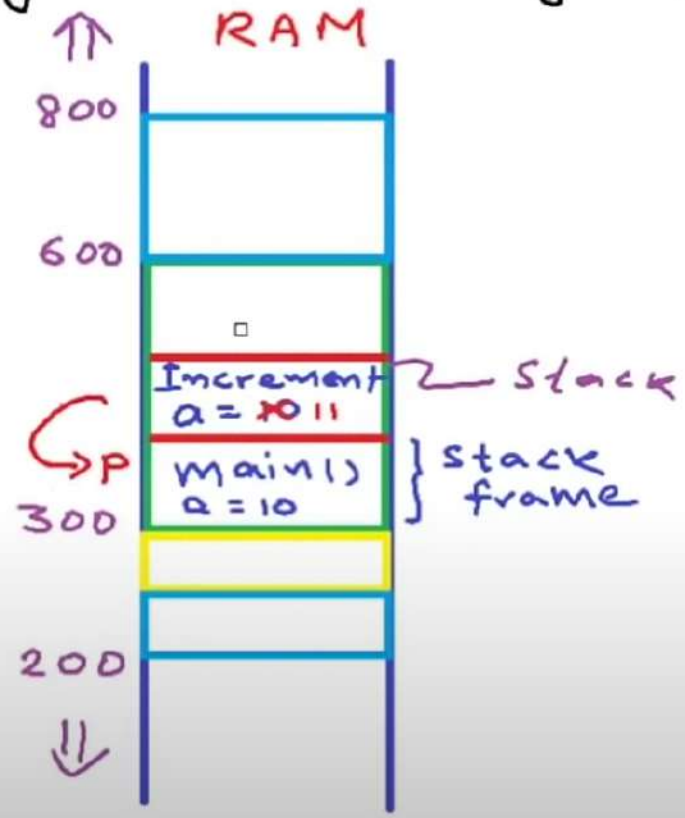


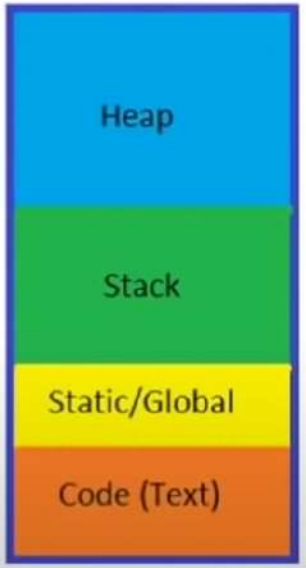
```c
// pointers to pointers
#include<stdio.h>
int main()
{
    int x = 5;
    int* p = &x;
    *p = 6;
    int** q = &p;
    int*** r = &q;
    printf("%d\n",*p);
    printf("%d\n",*q);
    printf("%d\n",**q);
    printf("%d\n",**r);
    printf("%d\n",***r);
    ***r = 10;
    printf("x = %d\n",x);
    **q = *p +2;
    printf("x = %d\n",x);
}
```



Pointers as function arguments - Call by reference

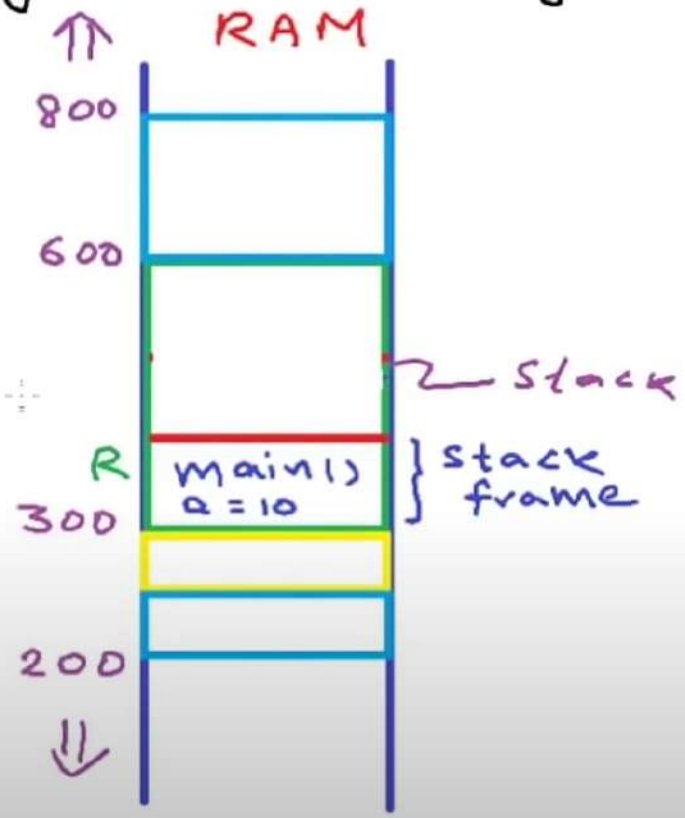
```c
#include<stdio.h>
void Increment(int a)
{
    a = a+1;
}
int main()
{
    int a;
    a = 10;
    Increment(a);
    printf("a = %d",a);
}
```

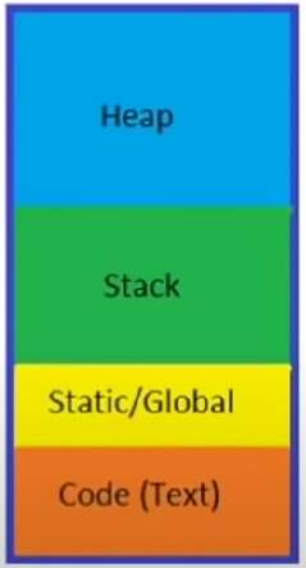




Pointers as function arguments - Call by reference

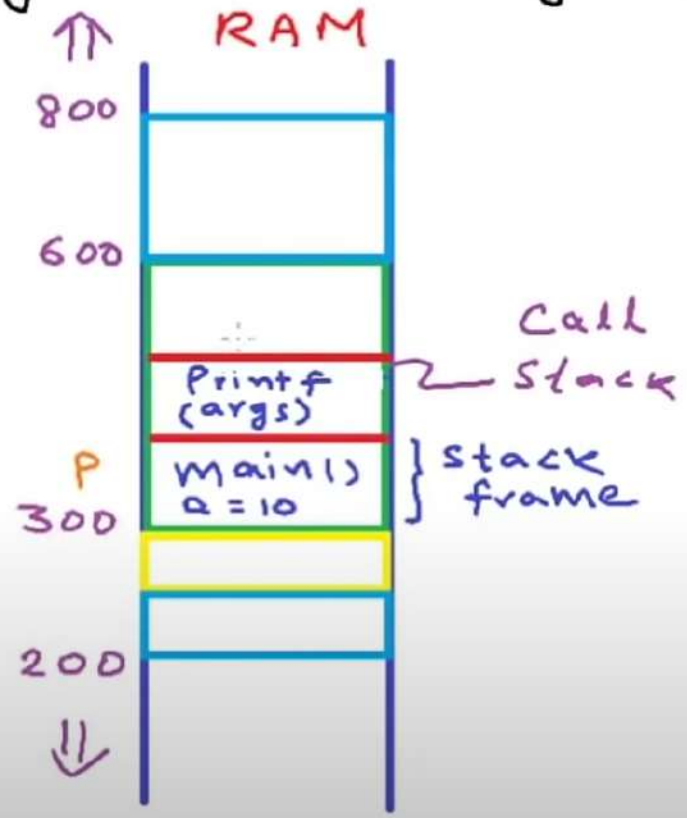
```c
#include<stdio.h>
void Increment(int a)
{
    a = a+1;
}
int main()
{
    int a;
    a = 10;
    Increment(a);
    printf("a = %d",a);
}
```

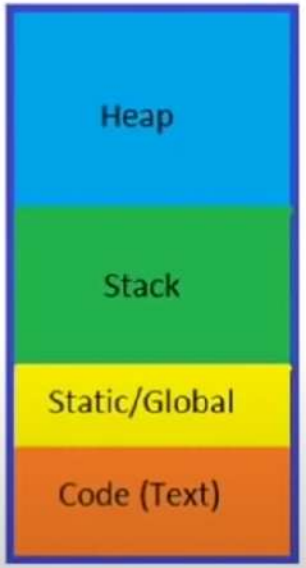




Pointers as function arguments - Call by reference

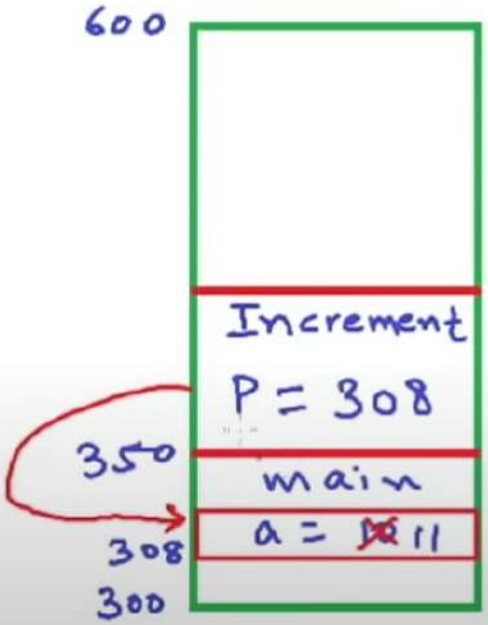
```c
#include<stdio.h>
void Increment(int a)
{
    a = a+1;
}
int main()
{
    int a;
    a = 10;
    Increment(a);
    printf("a = %d",a);
}
```

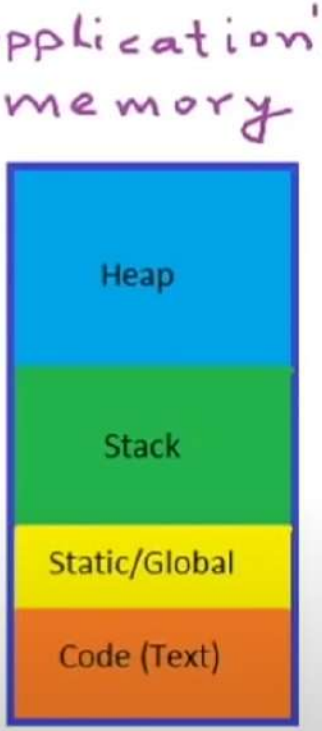




Pointers as function arguments - Call by reference

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





Pointers and Arrays

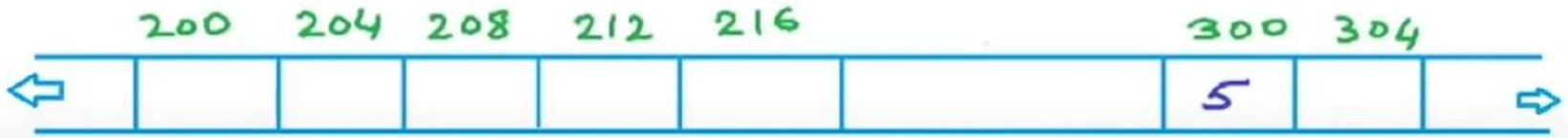
int A[5]

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

int → 4 bytes

A → 5 × 4 bytes
= 20 bytes

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




```
Print p  // 304
Print *p
```

Pointers and Arrays

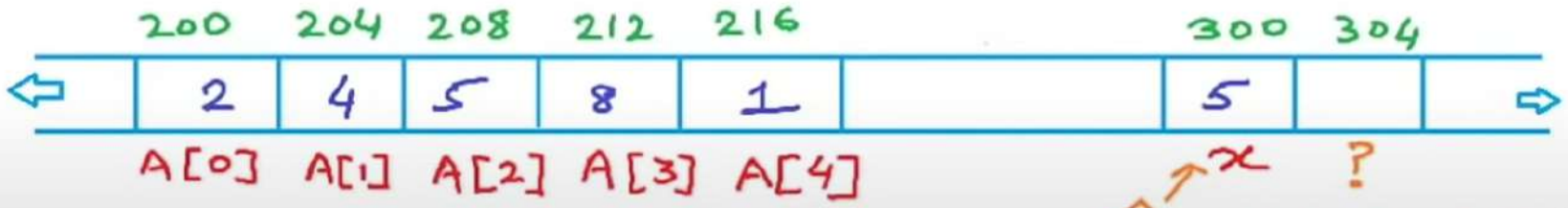
int A[5]

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

int → 4 bytes

A → 5 × 4 bytes
= 20 bytes

int A[5]
int *p
P = & A[0]
Print P // 200
Print *p // 2



Print P+2 //208
Print *(P+2) // 5

Pointers and Arrays

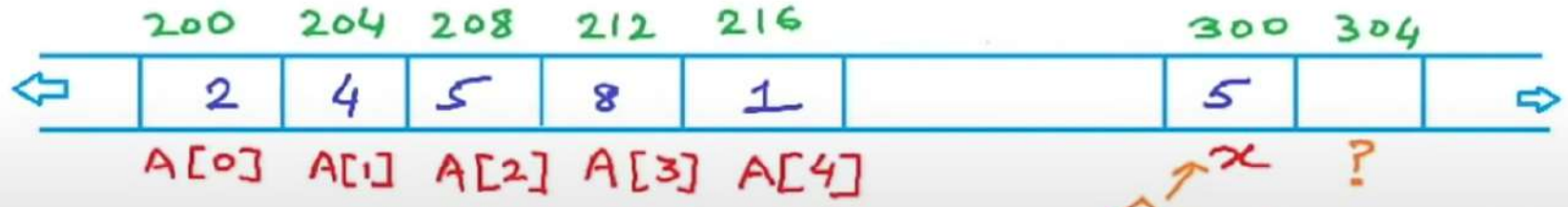
int A[5]

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

int → 4 bytes

A → 5 × 4 bytes
= 20 bytes

```
int A[5]
int *p
P = A
Print A // 200
Print *A // 2
```




```
Print A+1 // 204
Print *(A+1) // 4
```

Pointers and Arrays

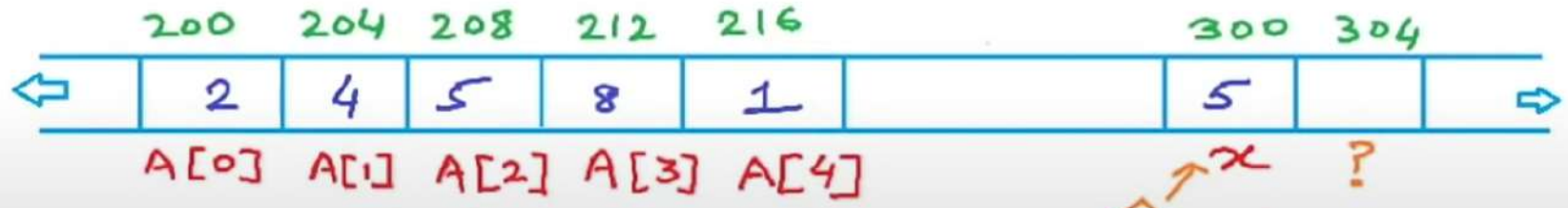
int A[5]

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

int → 4 bytes

A → 5 × 4 bytes
= 20 bytes

```
int A[5]
int *p
P = A
Print A // 200
Print *A // 2
```



Element at index i –

Address – &A[i] or (A + i)

Value – A[i] or *(A+i)

```
Print A+1 // 204
Print *(A+1) // 4
```

Pointers and Arrays

int A[5]

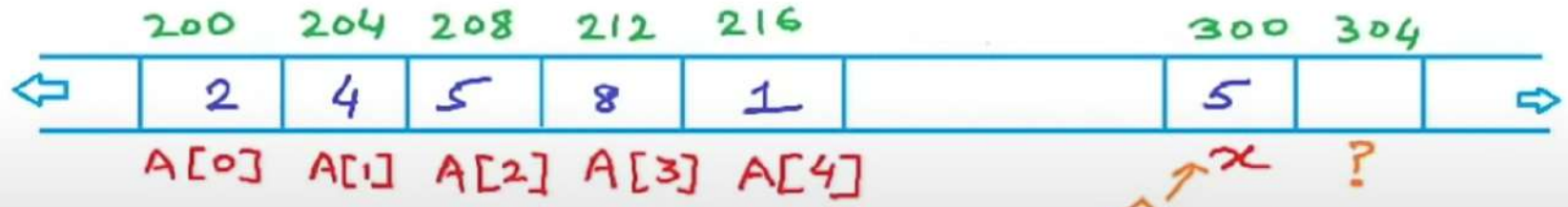
A[0]
A[1]
A[2]
A[3]
A[4]

int → 4 bytes

A → 5 × 4 bytes
= 20 bytes

A gives us base address

int A[5]
int *p
P = A
Print A // 200
Print *A // 2



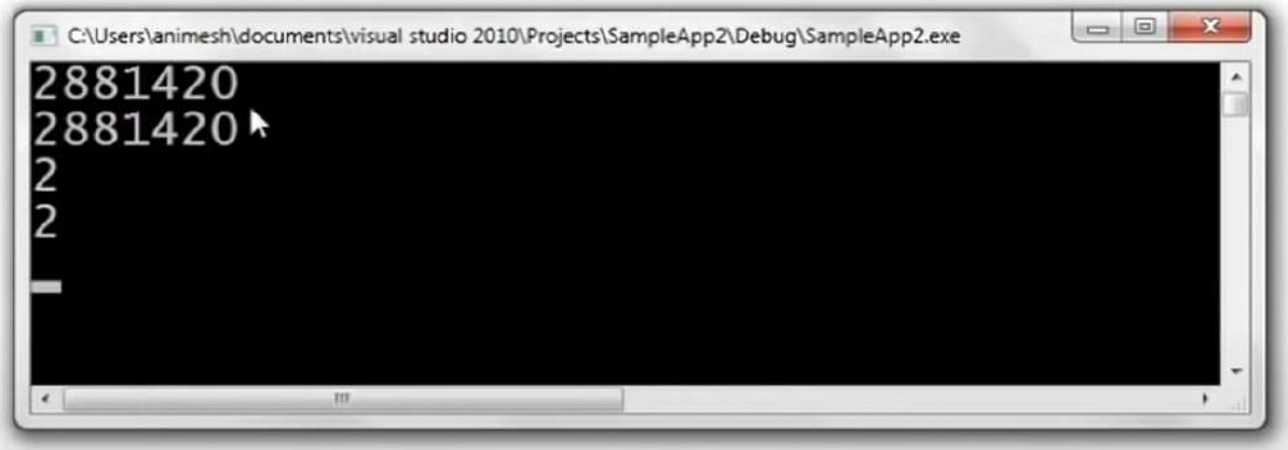
Element at index i –

Address – &A[i] or (A+i)

Value – A[i] or *(A+i)

Print A+1 // 204
Print *(A+1) // 4

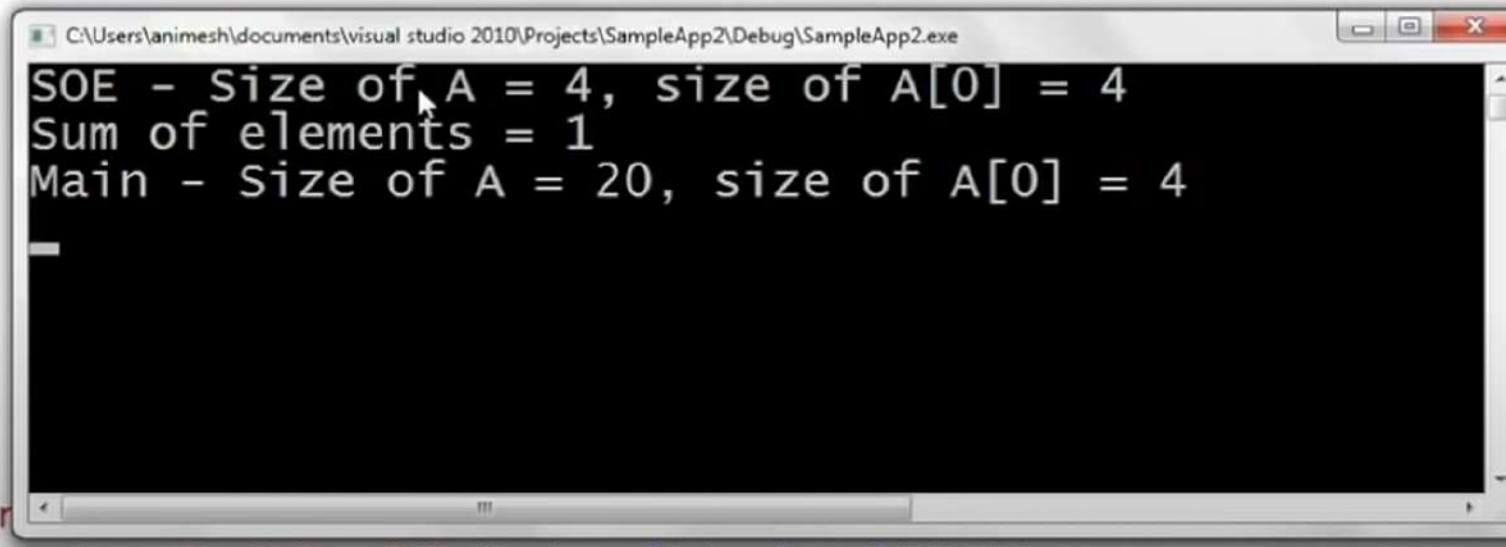
```c
// Pointers and Arrays
#include<stdio.h>
int main()
{
    int A[] ={2,4,5,8,1};
    printf("%d\n",A);
    printf("%d\n",&A[0]);
    printf("%d\n",A[0]);
    printf("%d\n",*A);
}
```



```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));
    }
}
```

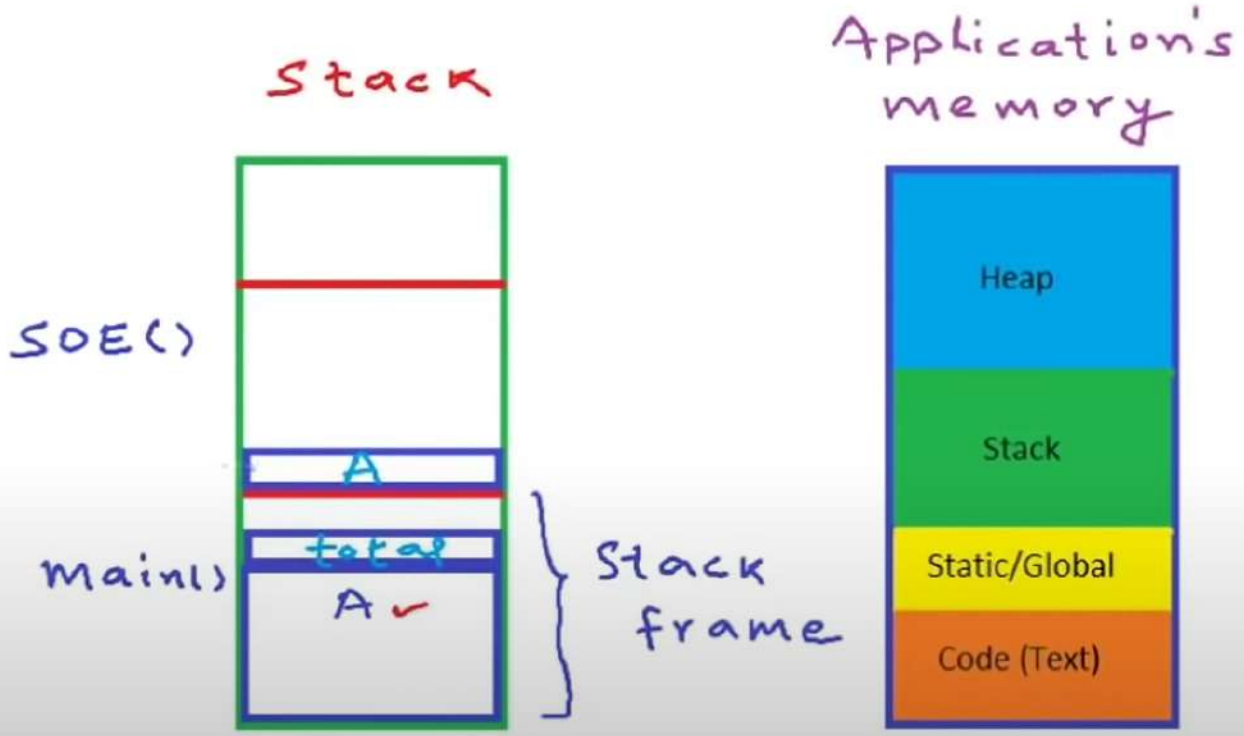
```c
// Arrays as function arguments
#include<stdio.h>
int SumOfElements(int A[], int size)
{
    int i, sum = 0;
    for(i = 0;i< size;i++)
    {
        sum+= A[i];
    }
    return sum;
}
int main()
{
    int A[]= {1,2,3,4,5};
    int size = sizeof(A)/sizeof(A[0]);
    int total = SumOfElements(A,size);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

```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]));
}
```



Arrays as function arguments

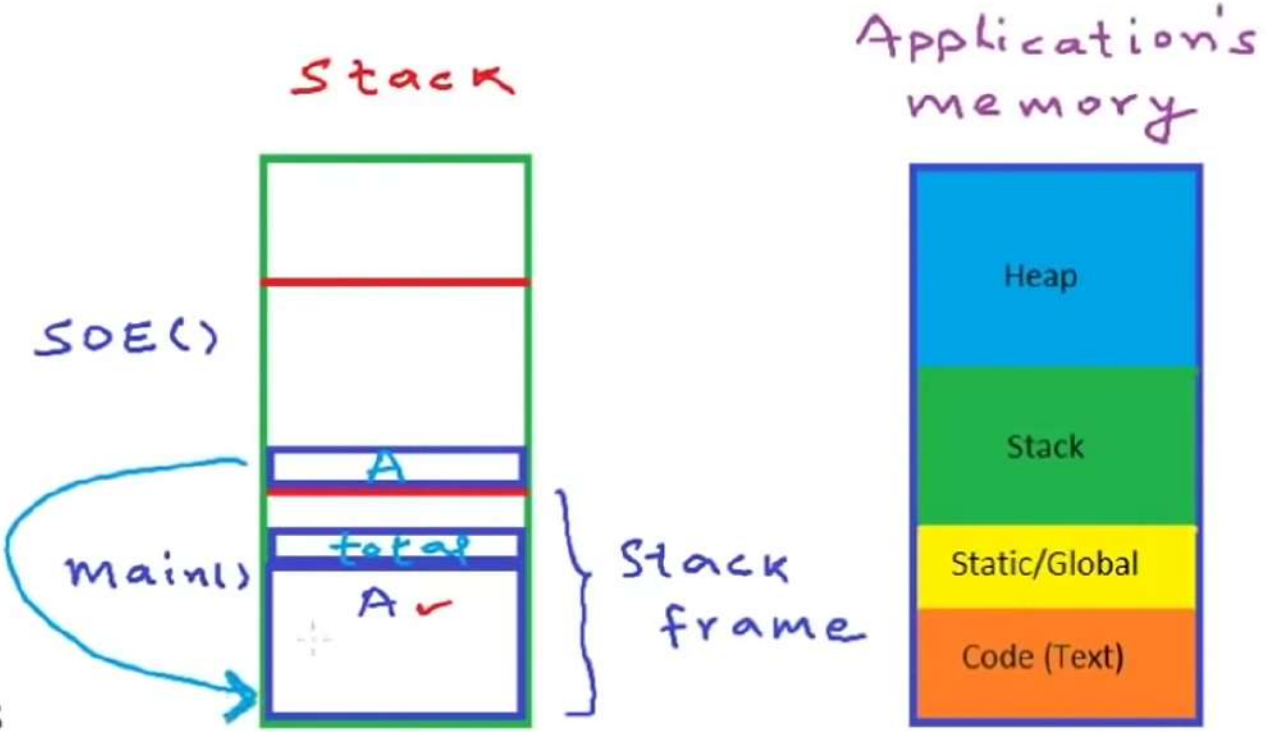
```c
#include<stdio.h>
int SumOfElements(int A[])
{
    int i, sum = 0;
    int size = sizeof(A)/sizeof(A[0]);
    for(i = 0;i< size;i++)
    {
        sum+= A[i];
    }
    return sum;
}
int main()
{
    int A[]= {1,2,3,4,5};
    int total = SumOfElements(A);
    printf("Sum of elements = %d\n",total);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



array it just creates a pointer
to the data type

Arrays as function arguments

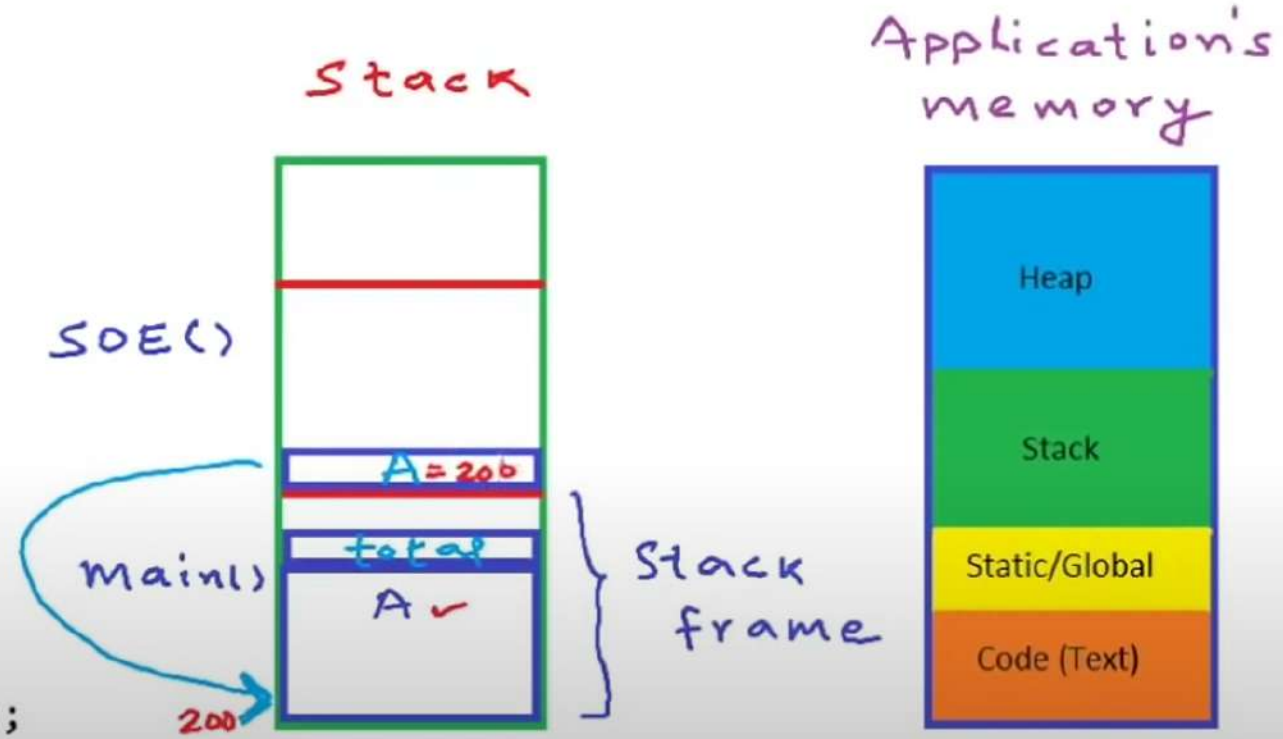
```c
#include<stdio.h>
int SumOfElements(int A[])
{
    int i, sum = 0;
    int size = sizeof(A)/sizeof(A[0]);
    for(i = 0;i< size;i++)
    {
        sum+= A[i];
    }
    return sum;
}
int main()
{
    int A[]= {1,2,3,4,5};
    int total = SumOfElements(A);
    printf("Sum of elements = %d\n",total);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



and the compiler just copies the address
of the first element in the array of the

Arrays as function arguments

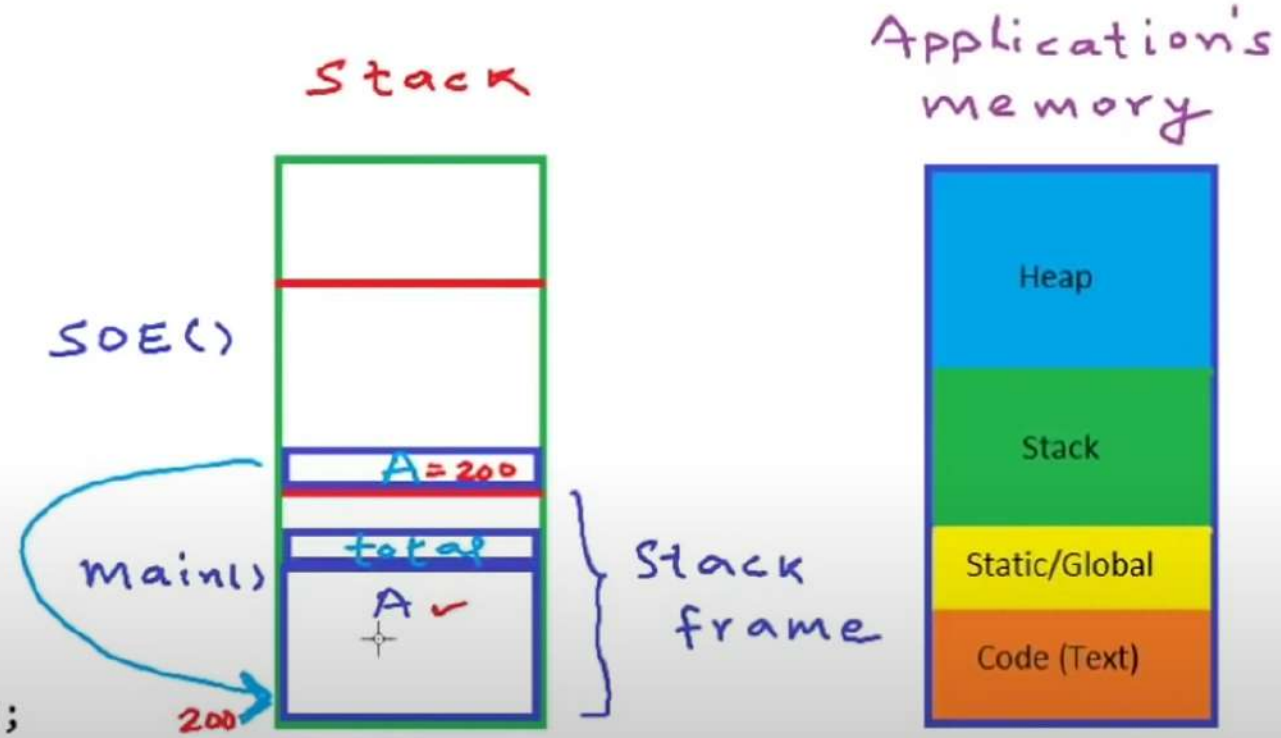
```c
#include<stdio.h>
int SumOfElements(int A[])
{
    int i, sum = 0;
    int size = sizeof(A)/sizeof(A[0]);
    for(i = 0;i< size;i++)
    {
        sum+= A[i];
    }
    return sum;
}
int main()
{
    int A[]= {1,2,3,4,5};
    int total = SumOfElements(A);
    printf("Sum of elements = %d\n",total);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



Arrays as function arguments

```c
#include<stdio.h>
int SumOfElements(int A[])
{
    int i, sum = 0;
    int size = 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",total);
}
```

(int* A)



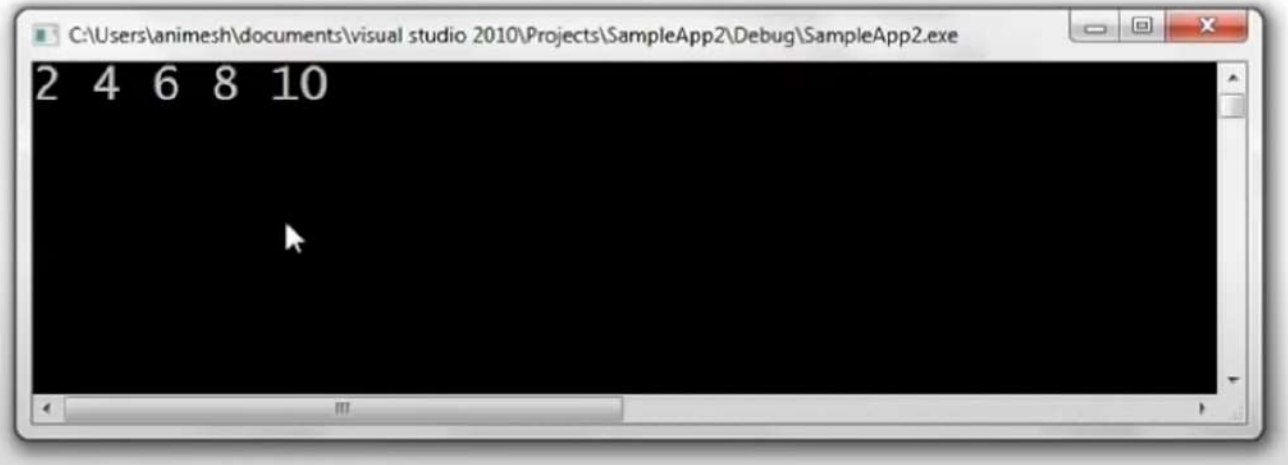
```c
#include<stdio.h>
int SumOfElements(int* A, int size)// "int* A" or "int A[]"  ..it's the same..
{
    int i, sum = 0;
    printf("SOE - Size of A = %d, size of A[0] = %d\n",sizeof(A),sizeof(A[0]));
    for(i = 0;i< size;i++)
    {
        sum+= A[i];
    }
    return sum;
}
int main()
{
    int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

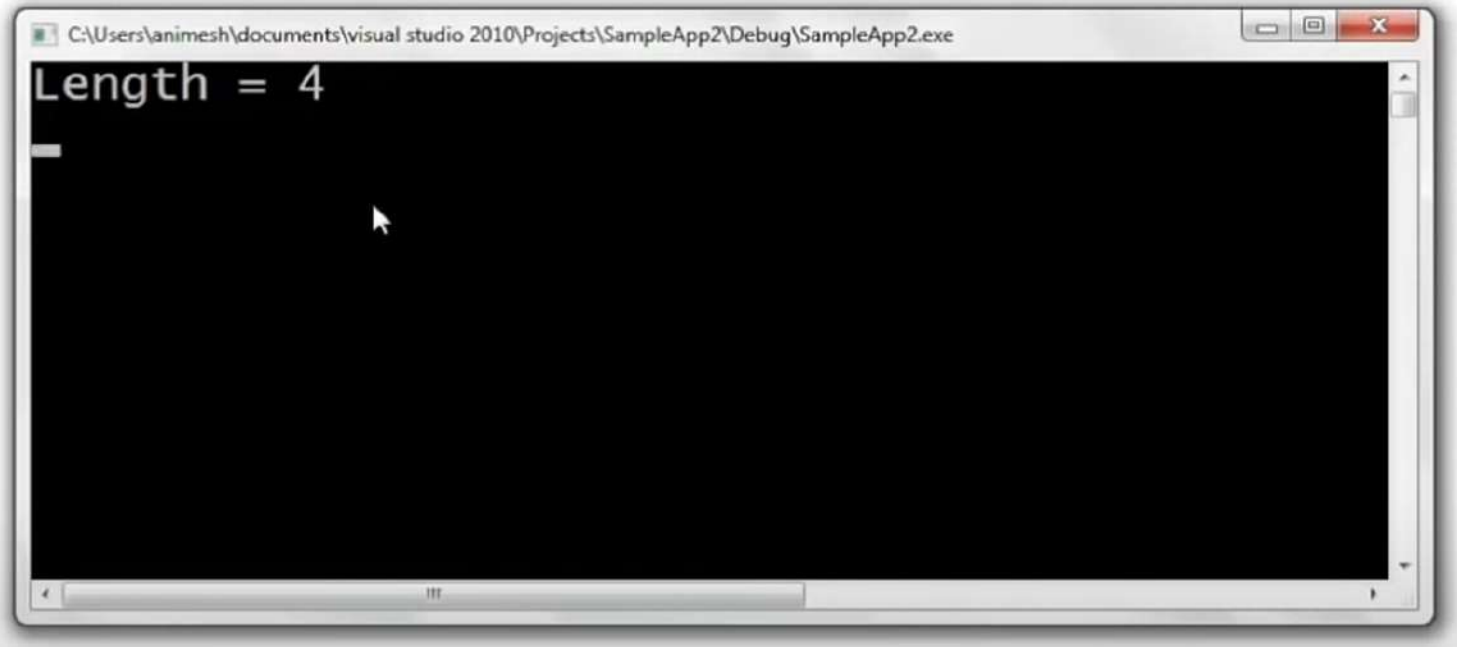
```
SOE - Size of A = 4, size of A[0] = 4
Sum of elements = 15
Main - Size of A = 20, size of A[0] = 4
```

```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]));
}
```

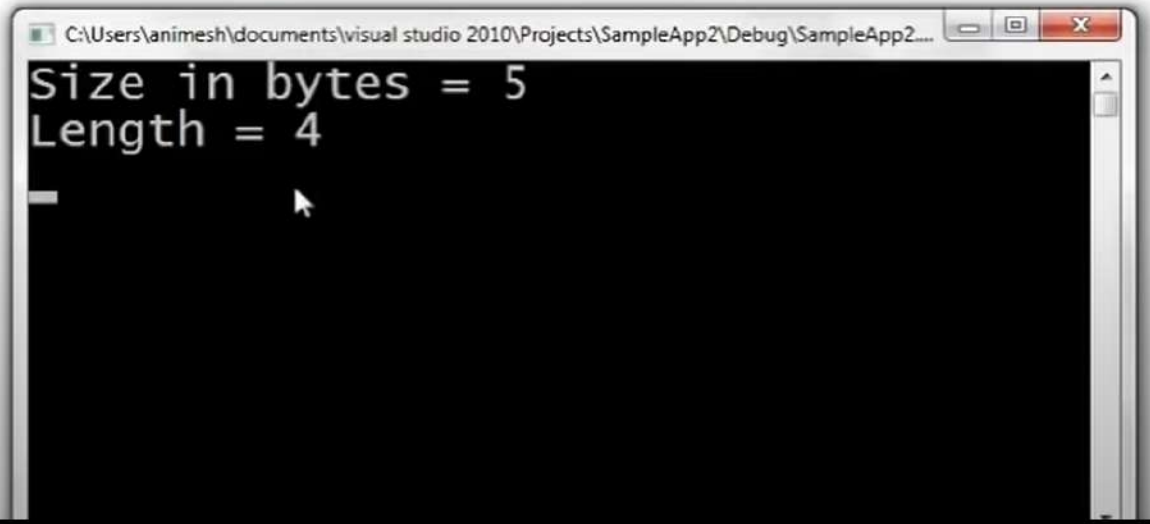
```c
#include<stdio.h>
void Double(int* A, int size)// "int* A" or "int A[]"  ..it's the same..
{
    int i, sum = 0;
    for(i = 0;i< size;i++)
    {
        A[i] = 2*A[i];
    }
}
int main()
{
    int A[]= {1,2,3,4,5};
    int size = sizeof(A)/sizeof(A[0]);
    int i;
    Double(A,size);
    for(i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



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



```c
//character arrays and pointers
#include<stdio.h>
#include<string.h>
int main()
{
    char C[] = "JOHN";
    printf("Size in bytes = %d\n",sizeof(C));
    int len = strlen(C);
    printf("Length = %d\n",len);
}
```

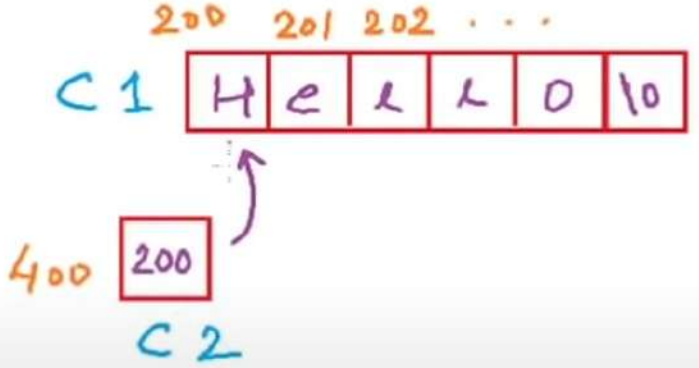


```c
//character arrays and pointers
#include<stdio.h>
#include<string.h>
int main()
{
    char C[5] = {'J','O','H','N','\0'};
    printf("Size in bytes = %d\n",sizeof(C));
    int len = strlen(C);
    printf("Length = %d\n",len);
}
```

Character arrays and pointers

2) Arrays and pointers are different types that are used in similar manner

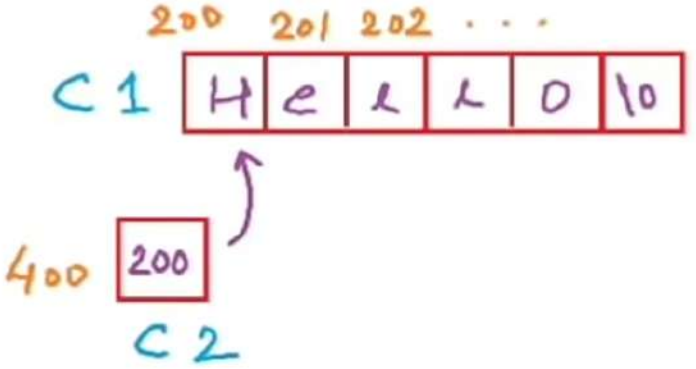
```
char c1[6] = "Hello";
char* c2;
c2 = c1;
```



2) Arrays and pointers are different types that are used in similar manner

```
char c1[6] = "Hello";
char* c2;
c2 = c1;
Print c2[1]; // l
c2[0] = 'A'; // "Aello"
```

c2[i] is *(c2+i)

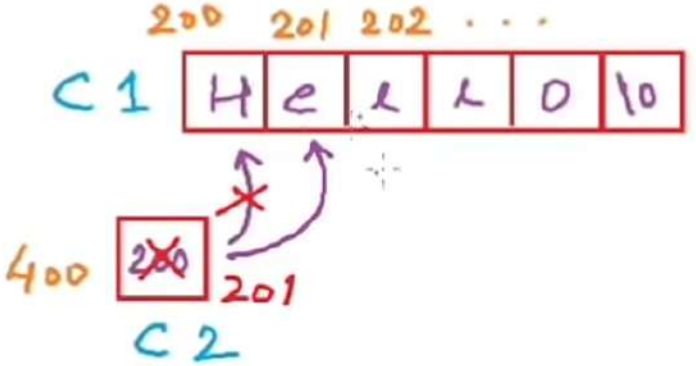


2) Arrays and pointers are different types that are used in similar manner

```
char c1[6] = "Hello";
char* c2;
c2 = c1; ✓
Print c2[1]; // l
c2[0] = 'A'; // "Aello"
```

c2[i] is *(c2+i)

c1[i] or *(c1+i)



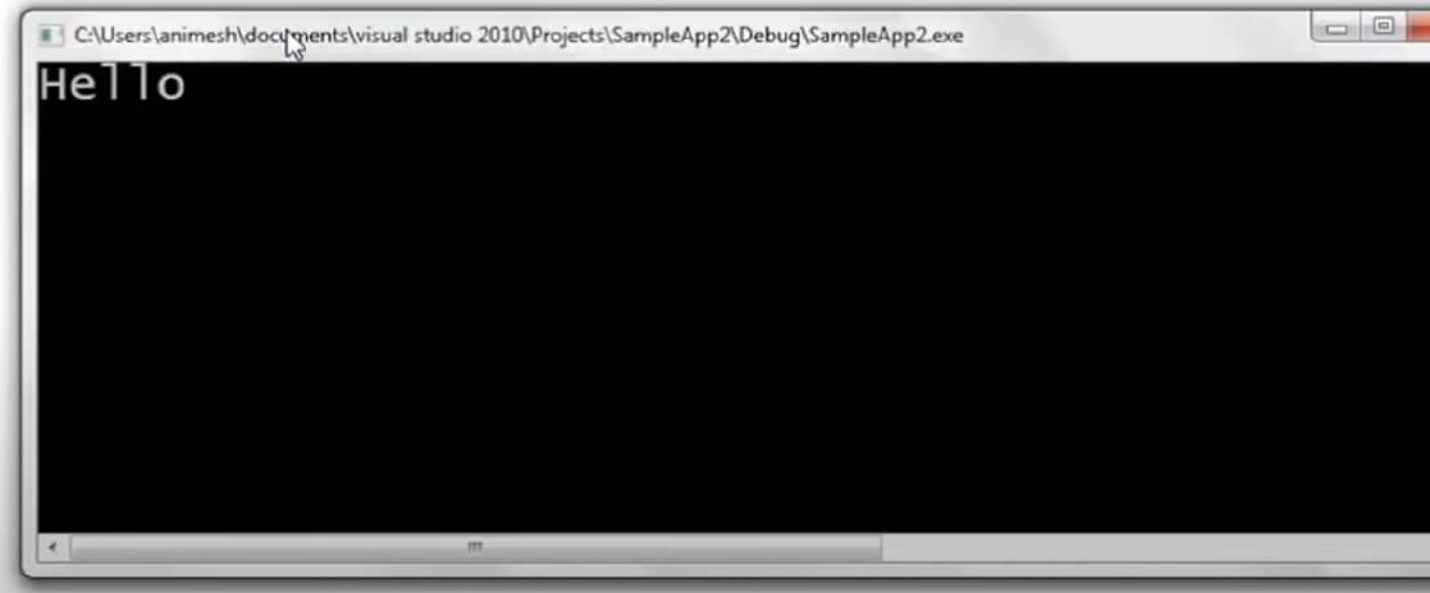
c1 = c2; ✗

c1 = c1+1; ✗

c2++;

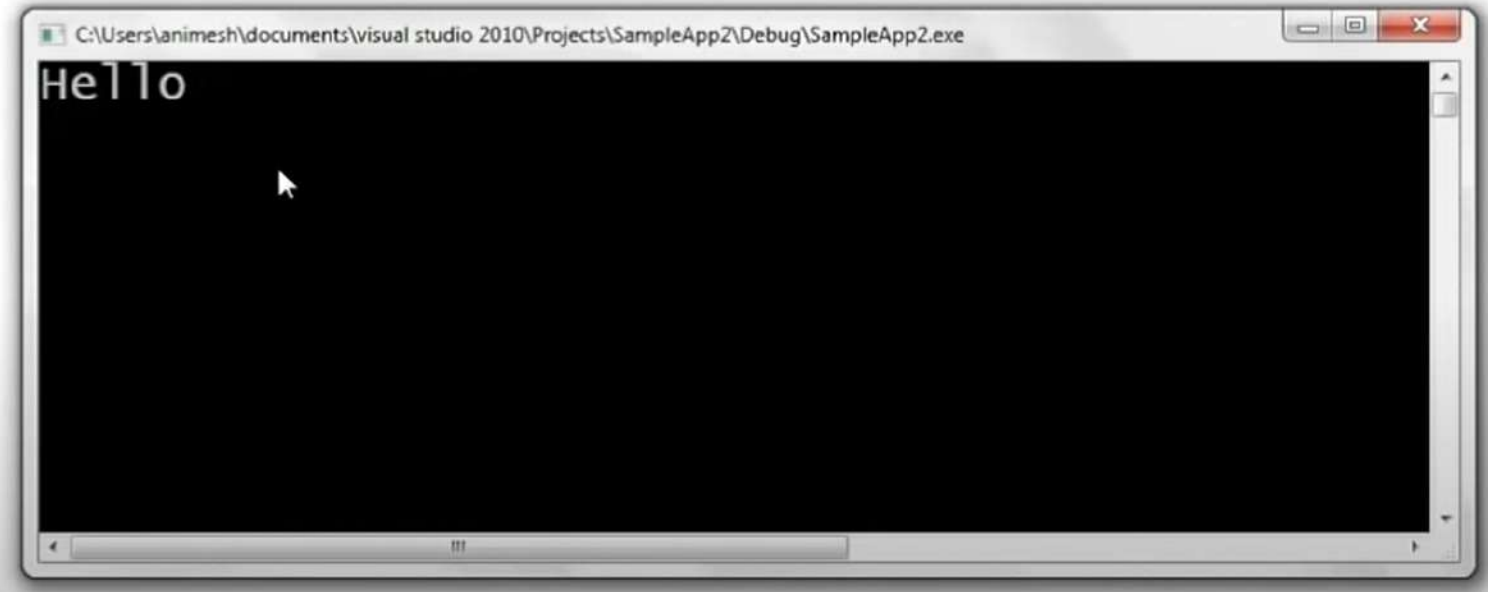
Pointers in C / C++ [Full Course]

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

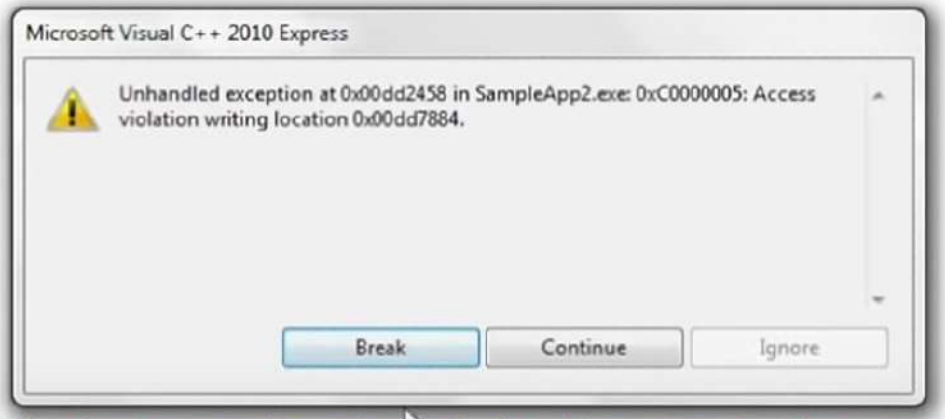


```c
//character arrays and pointers
#include<stdio.h>
void print(char* C)
{
    int i = 0;
    while(*(C+i)!= '\0')
    {
        printf("%c",C[i]);
        i++;
    }
    printf("\n");
}
int main()
{
    char C[20] = "Hello";
    print(C);
}
```

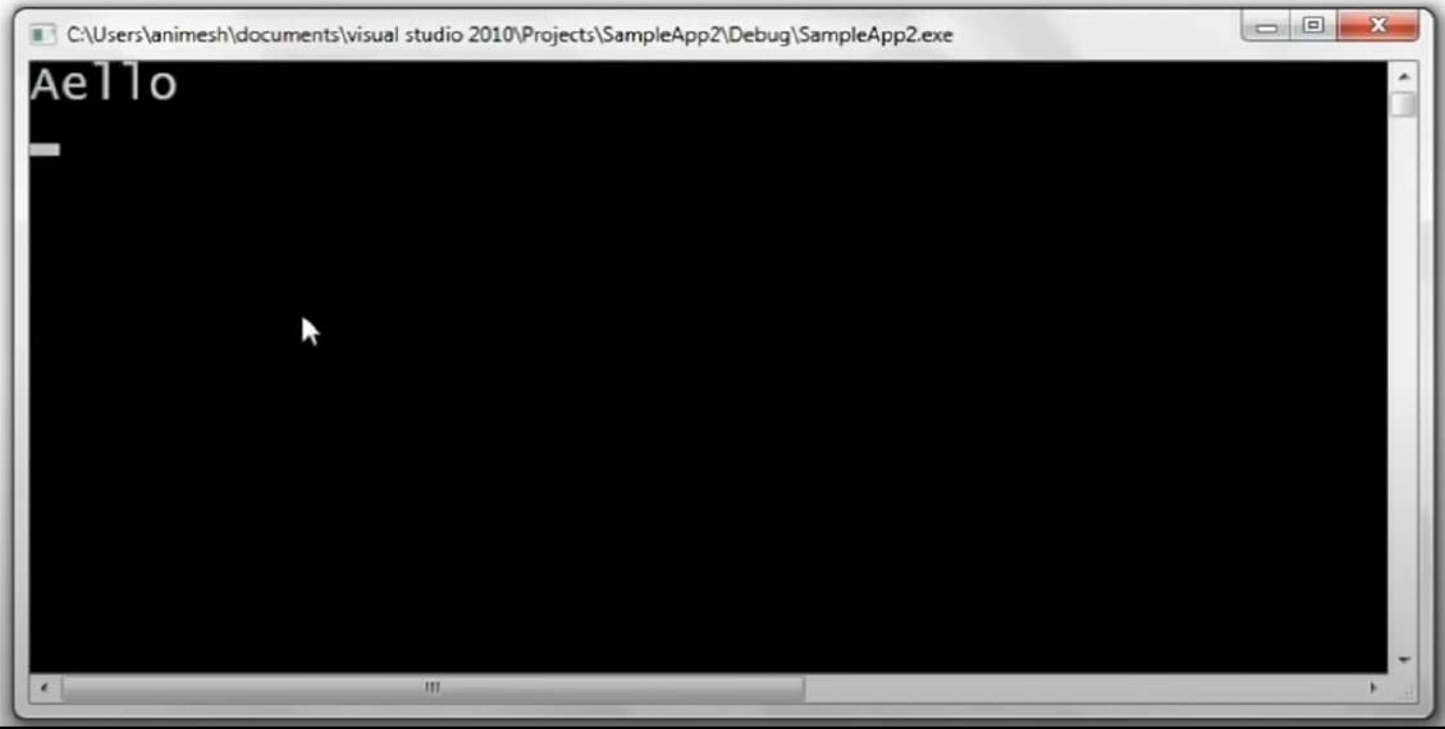
```c
//character arrays and pointers
#include<stdio.h>
void print(char* C)
{
    while(*C != '\0')
    {
        printf("%c",*C);
        C++;
    }
    printf("\n");
}
int main()
{
    char C[20] = "Hello";
    print(C);
}
```



```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
    C[0] = 'A';
    printf("Hello World");
    print(C);
}
```



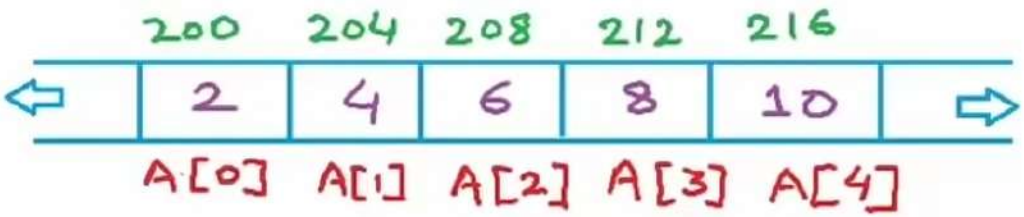
```c
//character arrays and pointers
#include<stdio.h>
#include<string.h>
void print(char *C)
{
    C[0] = 'A';
    while(*C != '\0')
    {
        printf("%c",*C);
        C++;
    }
    printf("\n");
}
int main()
{
    char C[20] = "Hello";
    print(C);
}
```



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

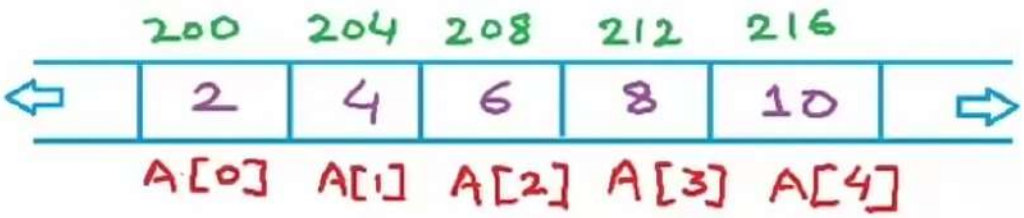
Pointers and multi-dimensional arrays

```
int A[5]

int *p = A;
Print p   // 200
Print *p  // 2
Print *(p+2) // 6
```



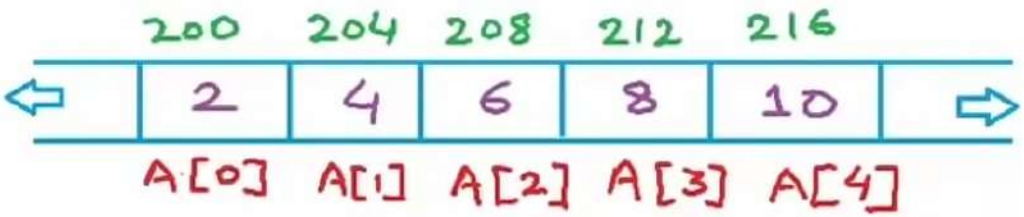
Pointers and multi-dimensional arrays

```
int A[5]

int *p = A;
Print A    // 200
Print *A   // 2
Print *(A+2) // 6
```



Pointers and multi-dimensional arrays

```
int A[5]
```




```
int *p = A;
Print A   // 200
Print *A  // 2
Print *(A+2) // 6
```

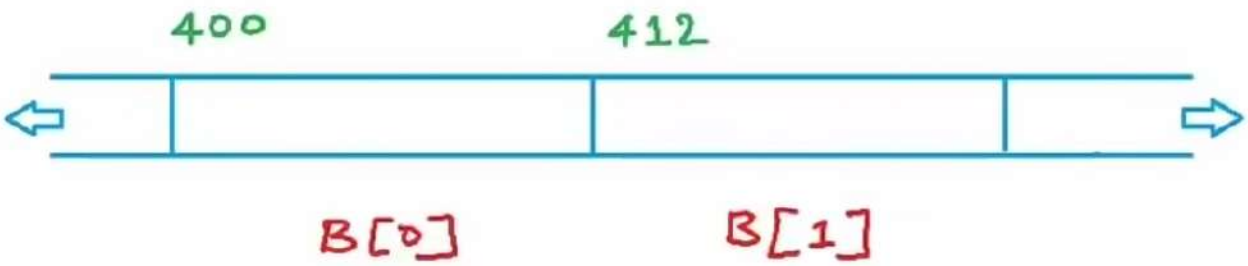
*(A+i) is same as A[i]

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

Pointers and multi-dimensional arrays

```
int B[2][3]
```

B[0]
B[1]
} → 1-D arrays of 3 integers




```
int *p = B;   X
```

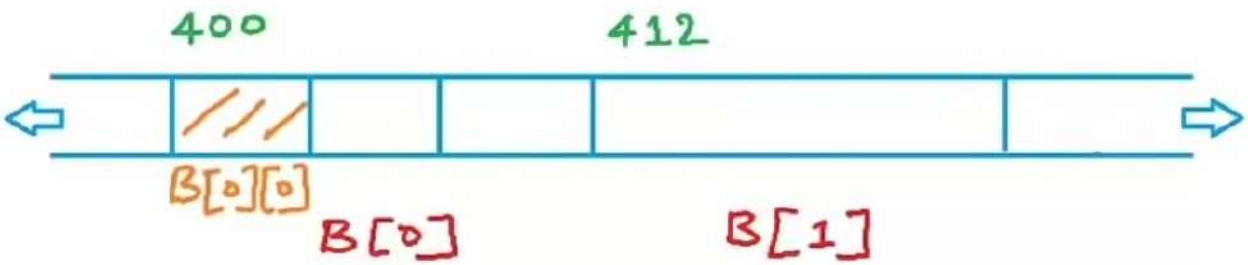
↓ will return a pointer to 1-D array of 3 integers

```
int (*p)[3]
```

Pointers and multi-dimensional arrays

```
int B[2][3]
```

B[0]
B[1]
} → 1-D arrays of 3 integers



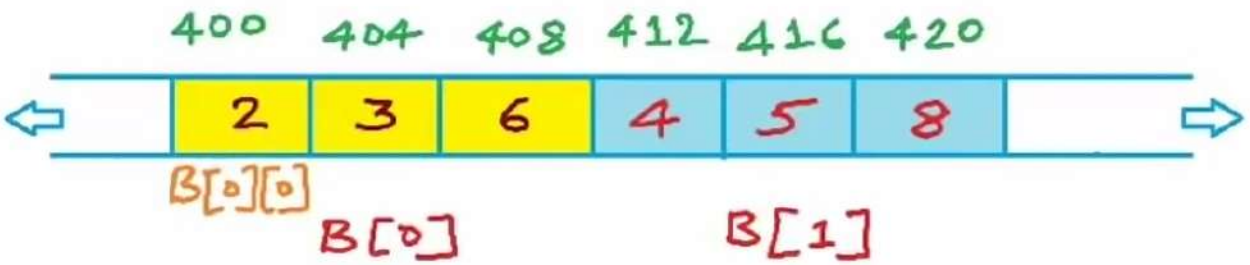

```
int (*p)[3] = B;
```

↓ will return a pointer to 1-D array of 3 integers

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

Pointers and multi-dimensional arrays

```
int B[2][3]

B[0]
B[1]   } → 1-D arrays of 3 integers
```

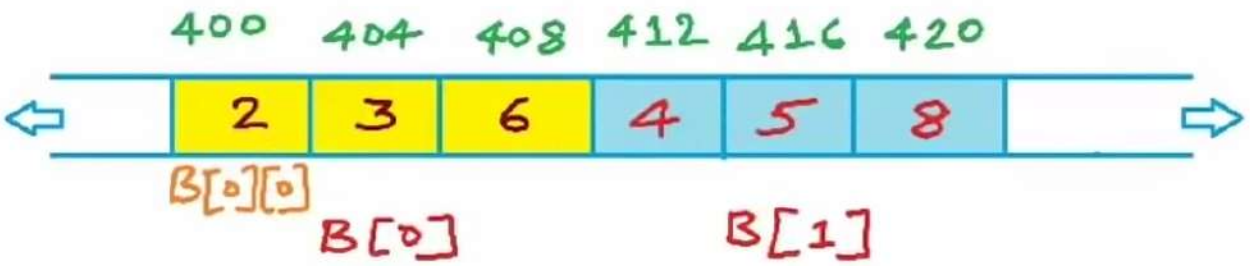



```
int (*p)[3] = B;

Print B or &B[0]          // 400
Print *B or B[0] or &B[0][0]   // 400
Print B+1                 // 400 + 12 = 412
      or
      &B[1]
```

Pointers and multi-dimensional arrays

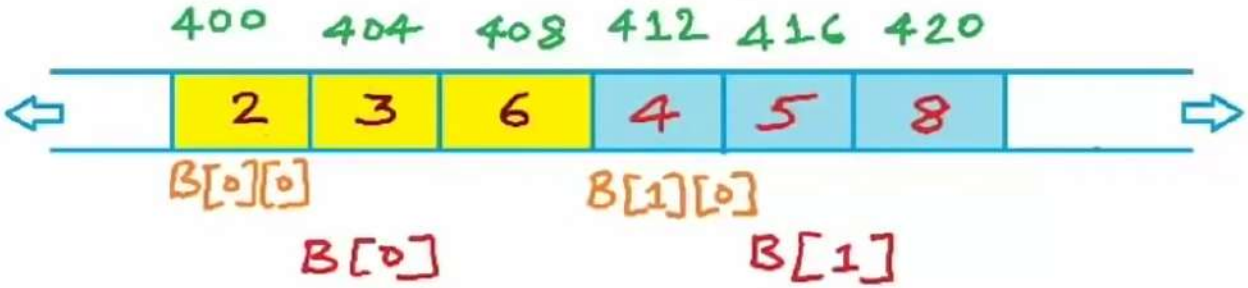
```
int B[2][3]
```

B[0]
B[1]
} → 1-D arrays of 3 integers

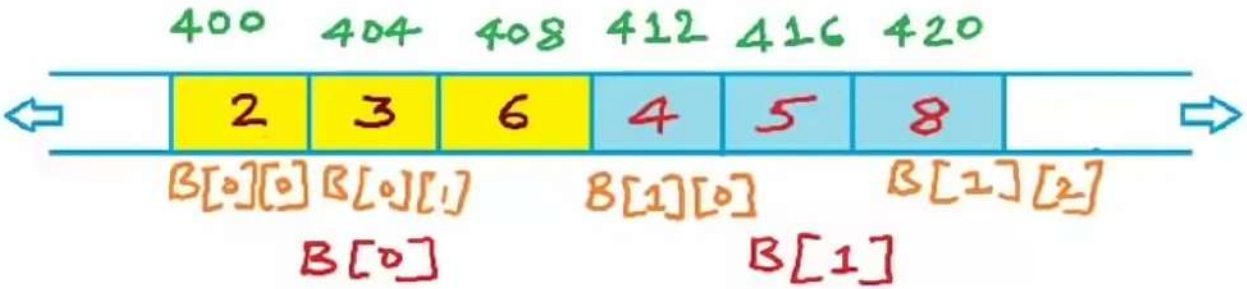



```
int (*p)[3] = B;
Print B or &B[0]          // 400
Print *B or B[0] or &B[0][0]   // 400
Print B+1 or &B[1]        // 412
Print *(B+1) or B[1] or &B[1][0]   // 412
```

```
int B[2][3]

B[0]
B[1]   } → 1-D arrays of 3 integers

int (*p)[3] = B;

Print B or &B[0]                    // 400
Print *B or B[0] or &B[0][0]        // 400
Print B+1 or &B[1]                  // 412
Print *(B+1) or B[1] or &B[1][0]    // 412
                  → returning int*
Print *(B+1)+2                      // 420
      ↓
     int*
```



```
int B[2][3]
B[0]
B[1]   } → 1-D arrays of 3 integers
```

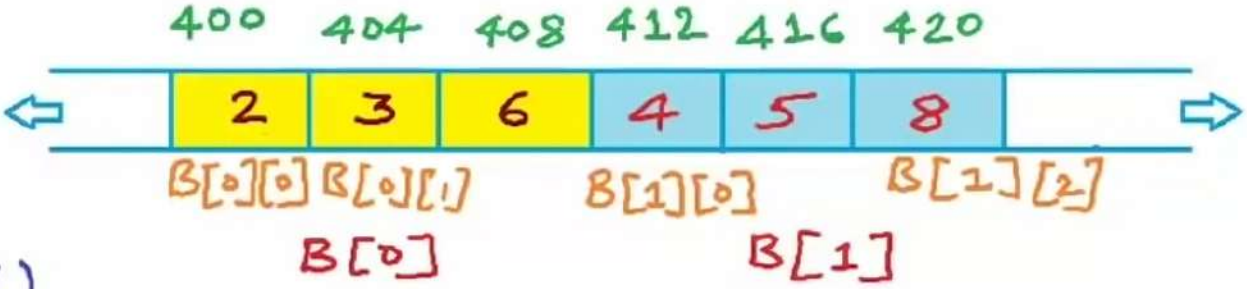



```
int (*p)[3] = B;
Print B or &B[0]                    // 400
Print *B or B[0] or &B[0][0]        // 400
Print B+1 or &B[1]                  // 412
Print *(B+1) or B[1] or &B[1][0]    // 412
Print *(B+1)+2 or B[1]+2 or &B[1][2]  // 420
Print *(*B+1)                       // 3
```

B[0][1]

```
int B[2][3]
```

For 2-D array



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

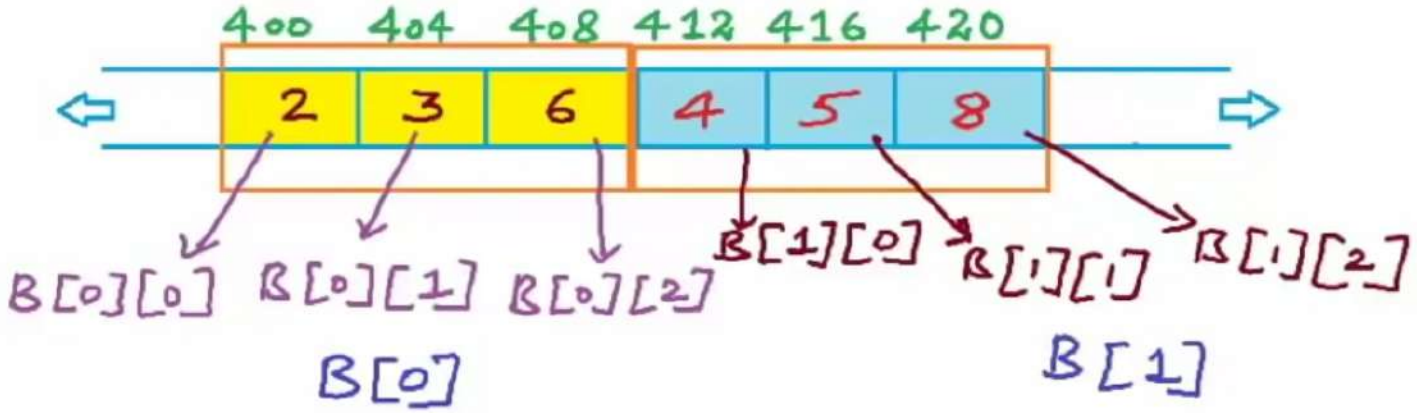
$$= *(*(B+i)+j)$$

Pointers and multi-dimensional arrays

```
int B[2][3]

int (*p)[3] = B; ✓
```

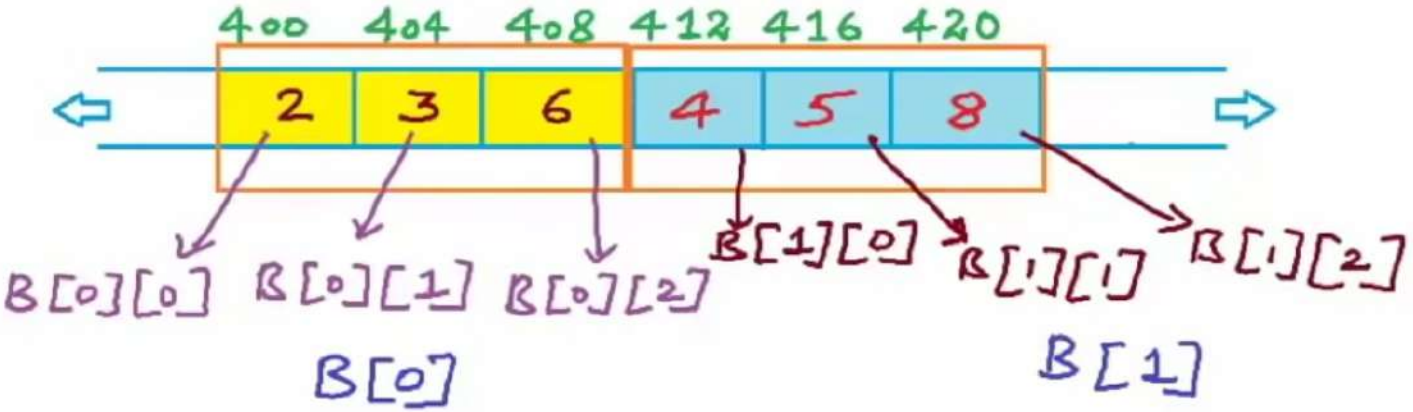
declaring pointer to 1-D array of 3 integers

```
int *p = B; X
```



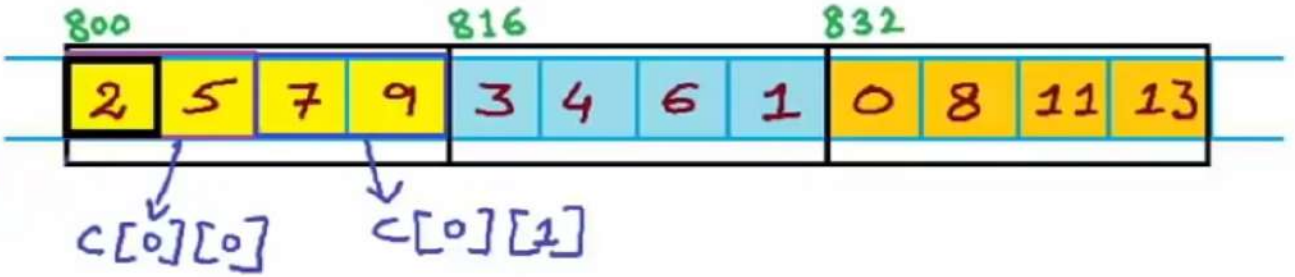
Pointers and multi-dimensional arrays

```
int B[2][3]

int (*P)[3] = B; ✓

Print B      //400
Print *B     //400
Print B[0]   //400
```



Pointers and multi-dimensional arrays

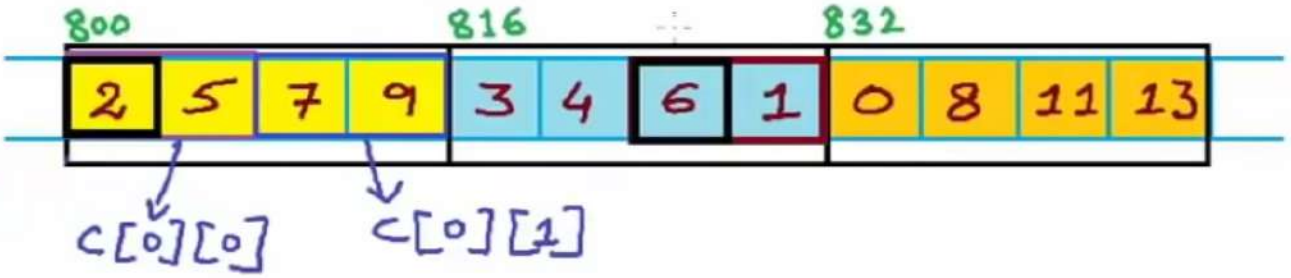
```
int C[3][2][2]
int (*p)[2][2] = C;
Print C      //800
   ↳ int (*)[2][2]
Print *C or C[0] or &C[0][0]
```



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

$$= *(*(*(C+i)+j)+k)$$

Pointers and multi-dimensional arrays



int c[3][2][2]

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

Print c // 800
↳ int (*)[2][2]

c[0] c[1] c[2]

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

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

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

int (*)[2]

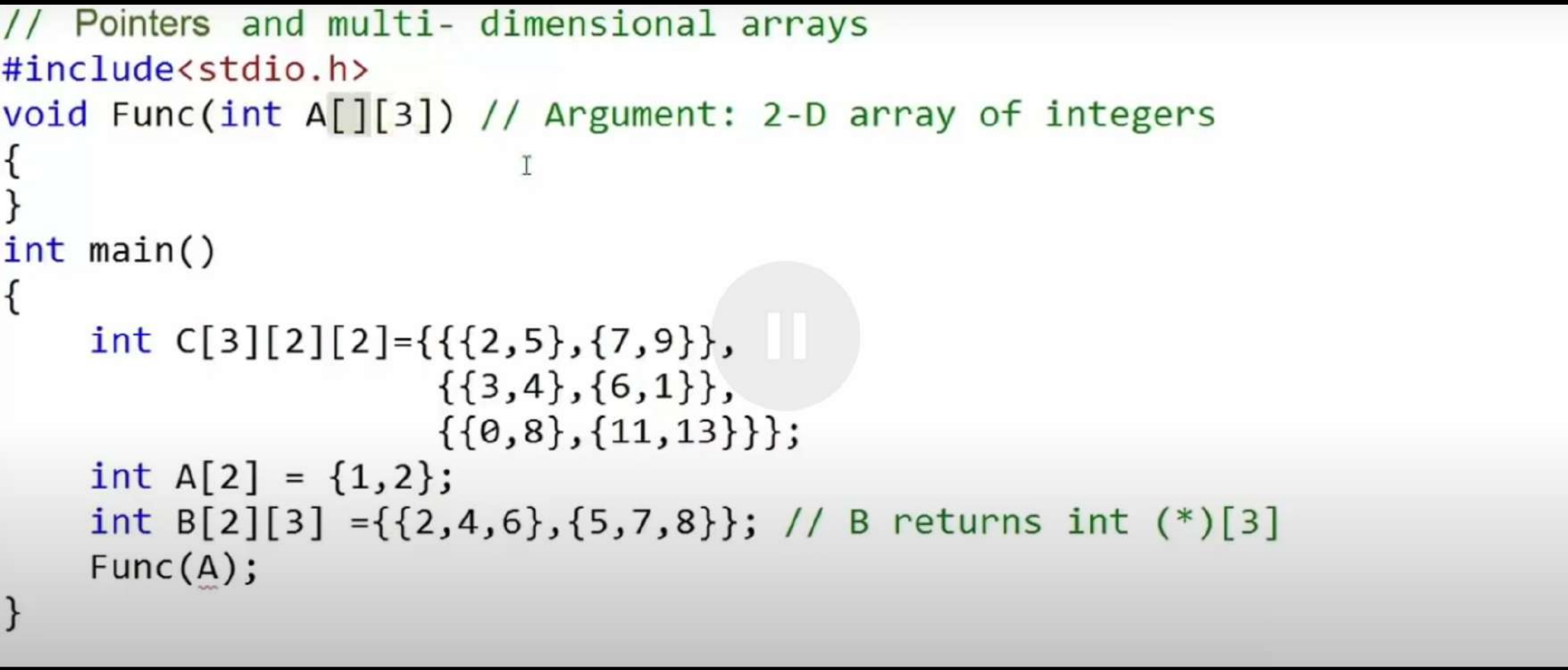
```
// Pointers   and multi- dimensional arrays
#include<stdio.h>
int main()
{
    int C[3][2][2]={{{2,5},{7,9}},
                    {{3,4},{6,1}},
                    {{0,8},{11,13}}};
    printf("%d %d %d %d", C, *C, C[0], &C[0][0]);
}
```

```
// 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\n", C, *C,
    printf("%d\n",*(C[0][0]+1));
}
```

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

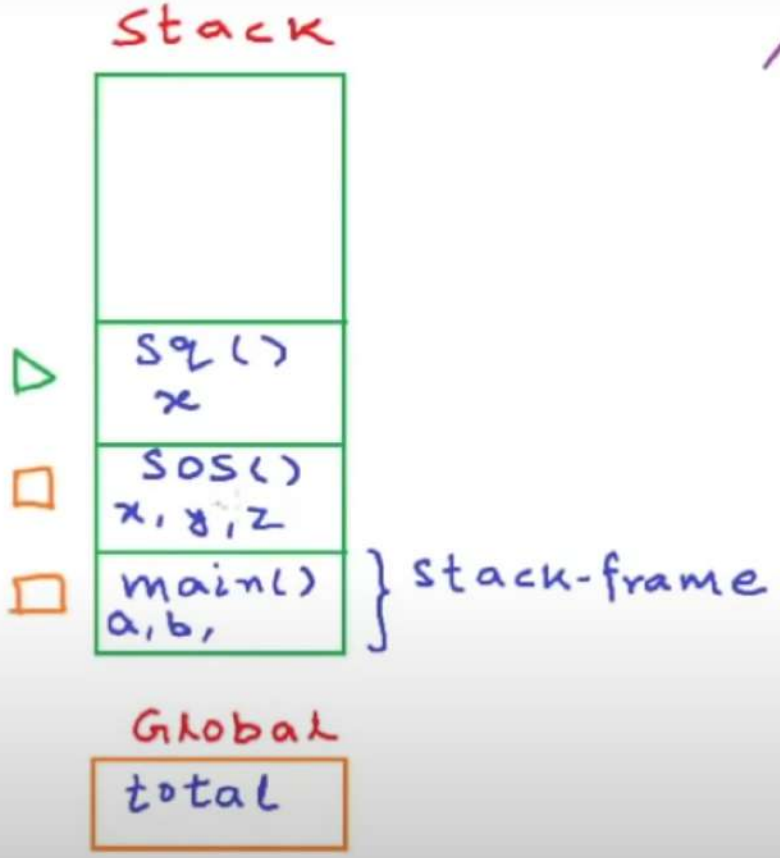
```
3078752 3078752 3078752 3078752
5
```

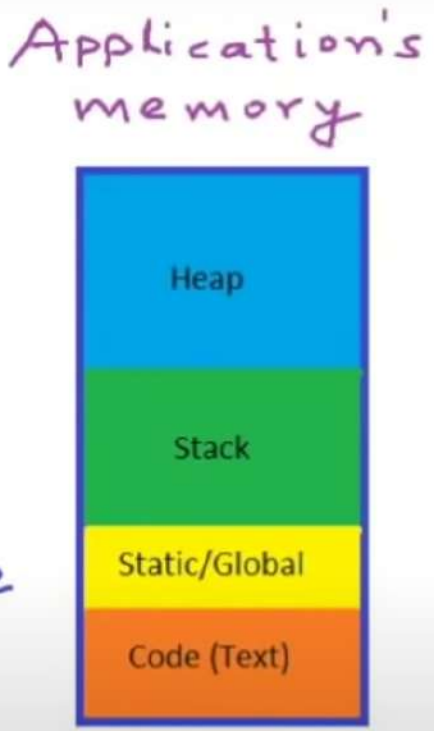
```c
// Pointers and multi- dimensional arrays
#include<stdio.h>
void Func(int (*A)[3]) // Argument: 2-D array of integers
{
}
int main()
{
    int C[3][2][2]={{{2,5},{7,9}},
                    {{3,4},{6,1}},
                    {{0,8},{11,13}}};
    int A[2] = {1,2};
    int B[2][3] ={{2,4,6},{5,7,8}};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



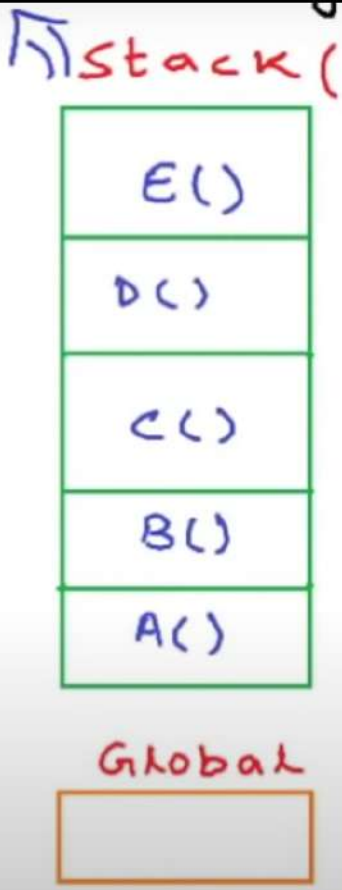
```c
// Pointers and multi- dimensional arrays
#include<stdio.h>
void Func(int A[][3]) // Argument: 2-D array of integers
{
}
int main()
{
    int C[3][2][2]={{{2,5},{7,9}},
                    {{3,4},{6,1}},
                    {{0,8},{11,13}}};
    int A[2] = {1,2};
    int B[2][3] ={{2,4,6},{5,7,8}};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

```c
#include<stdio.h>
int total;
int Square(int x)
{
    return x*x;  // x^2
}
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);
}
```

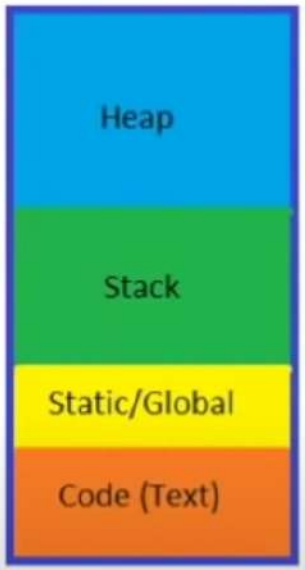




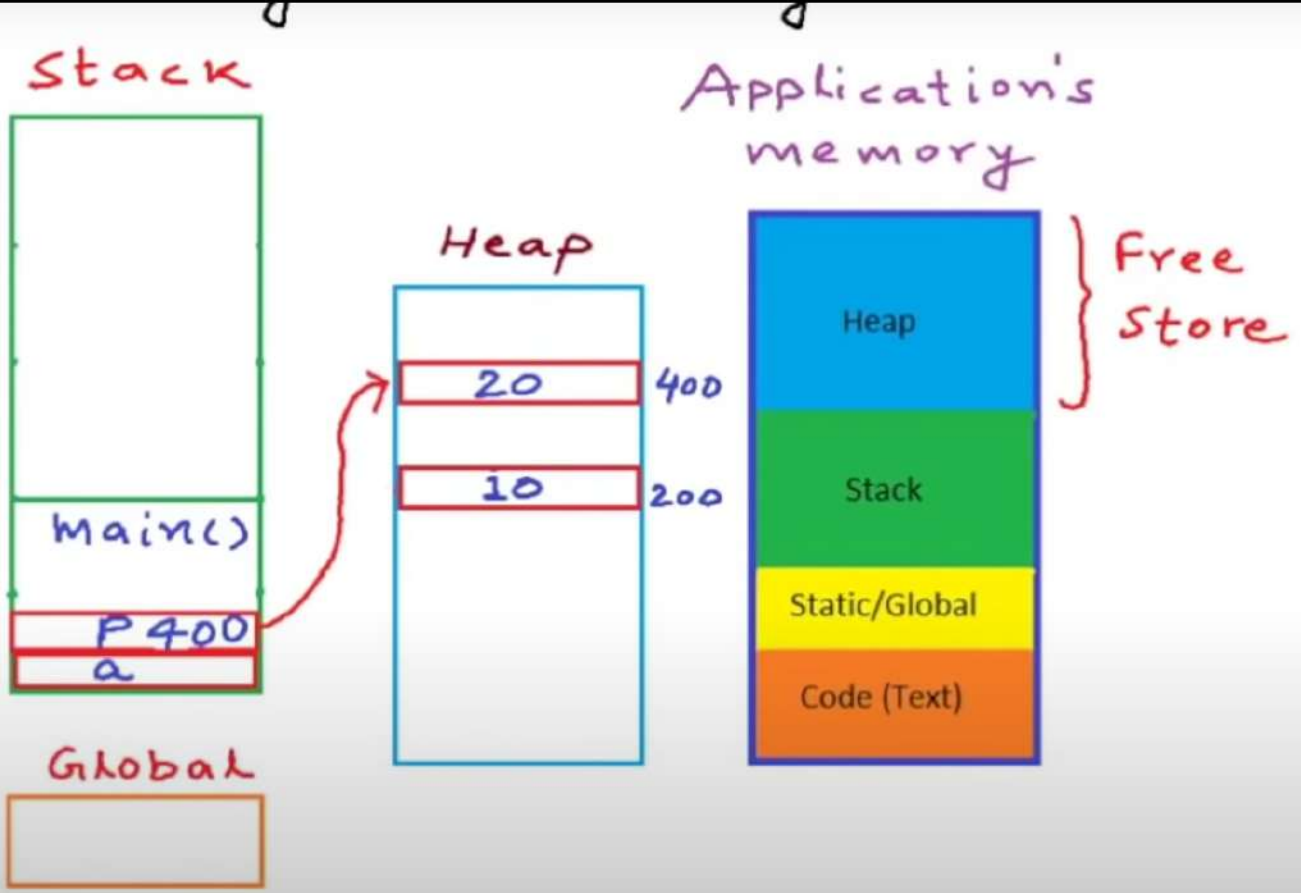
```c
#include<stdio.h>
int total;
int Square(int x)
{
    return x*x;  // x^2
}
int SquareOfSum(int x,int y)
{
    int z = Square(x+y);
    return z;  // (x+y)^2
}
int main()
{
    int a = 4, b = 8;
    total = SquareOfSum(a,b);
    printf("output = %d",total);
}
```



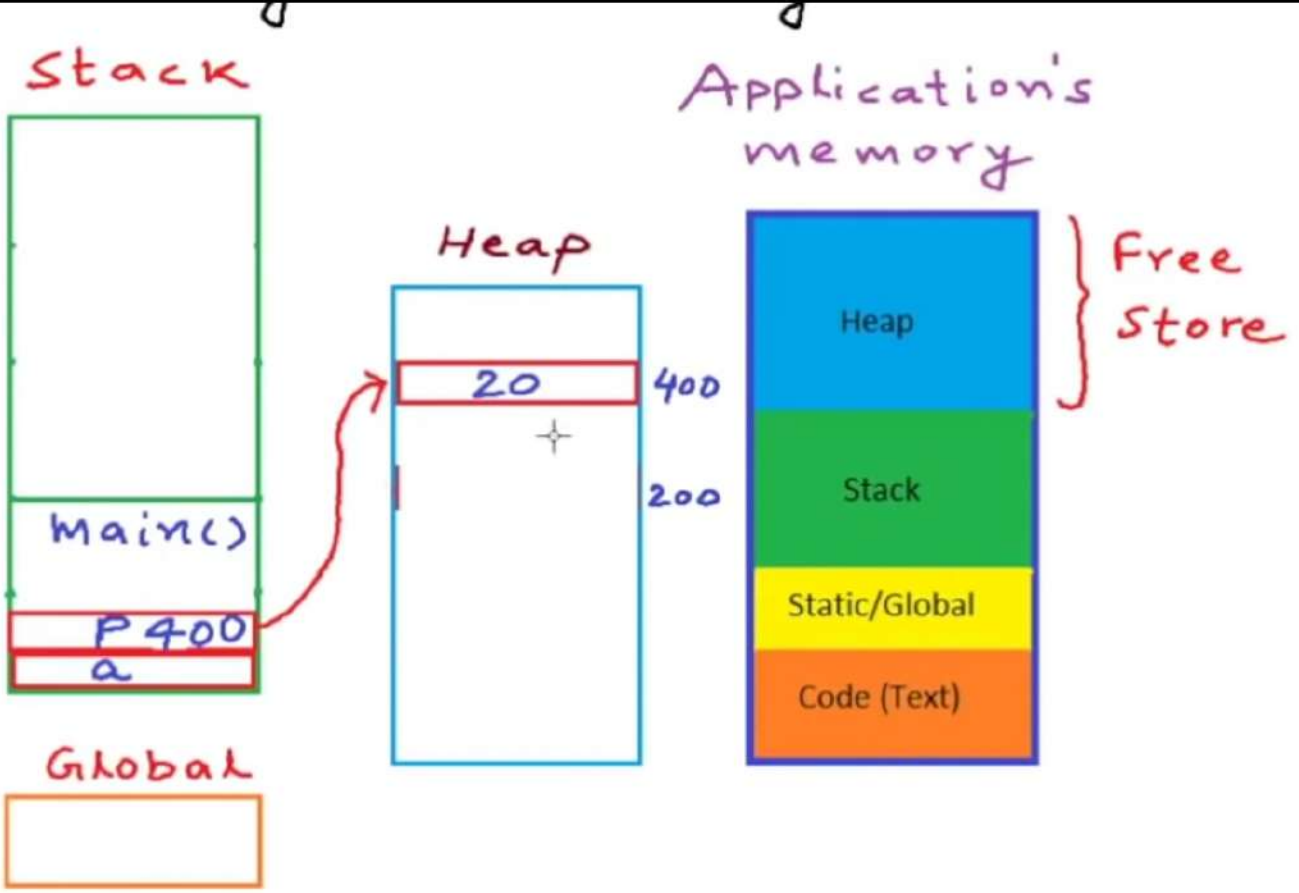
Stackoverflow

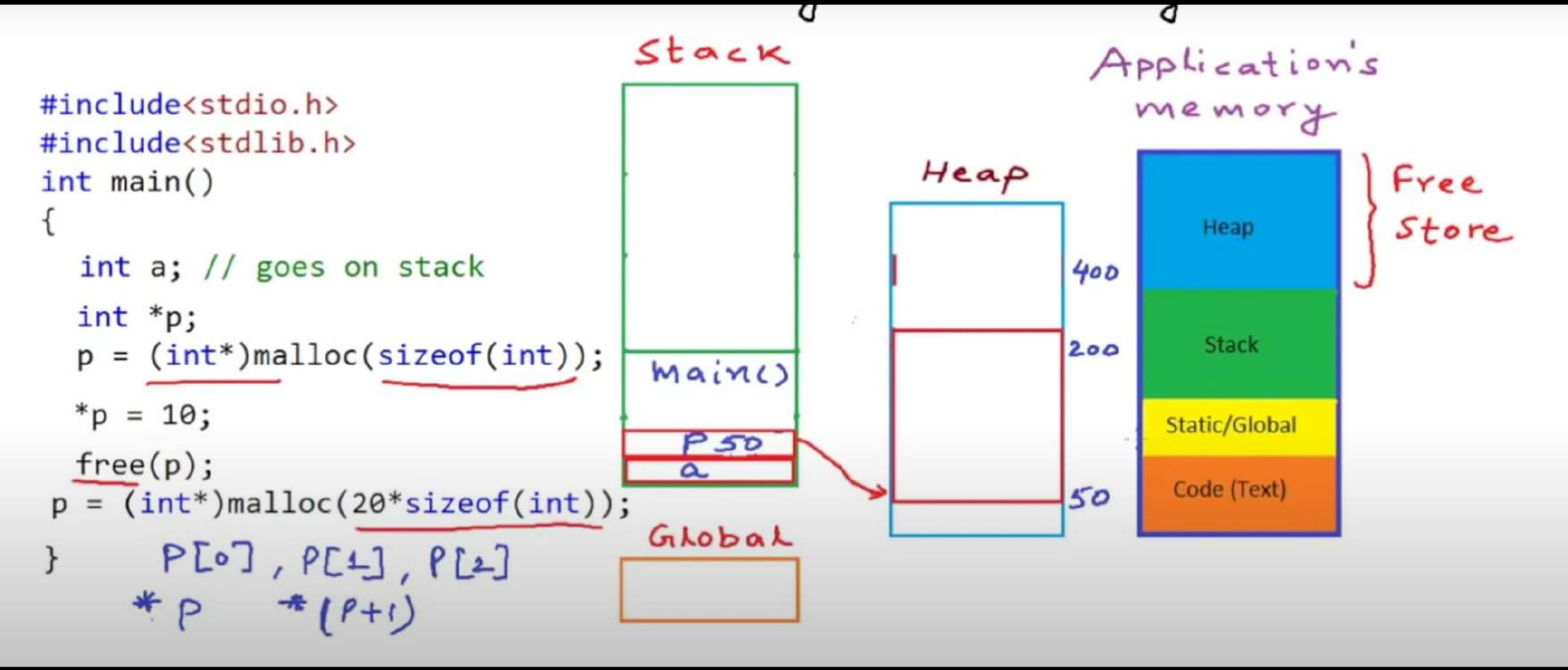


```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int a; // goes on stack
   int *p;
   p = (int*)malloc(sizeof(int));
   *p = 10;
   free(p);
   p = (int*)malloc(sizeof(int));
   *p = 20;
```



```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int a; // goes on stack
   int *p;
   p = (int*)malloc(sizeof(int));
   *p = 10;
   free(p);
   p = (int*)malloc(sizeof(int));
   *p = 20;
```




#include<stdio.h>
#include<stdlib.h>
int main()
{
int a; // goes on stack
int *p;
p = (int*)malloc(sizeof(int));
*p = 10;
free(p);
p = (int*)malloc(20*sizeof(int));
}
P[0], P[1], P[2]
*p *(p+1)
Stack
main()
P 50
a
Global
Heap
400
200
50
Application's memory
Heap
Stack
Static/Global
Code (Text)
Free Store

malloc, calloc, realloc, free

Allocate block of memory

malloc - void* malloc(size_t size)

unsigned int

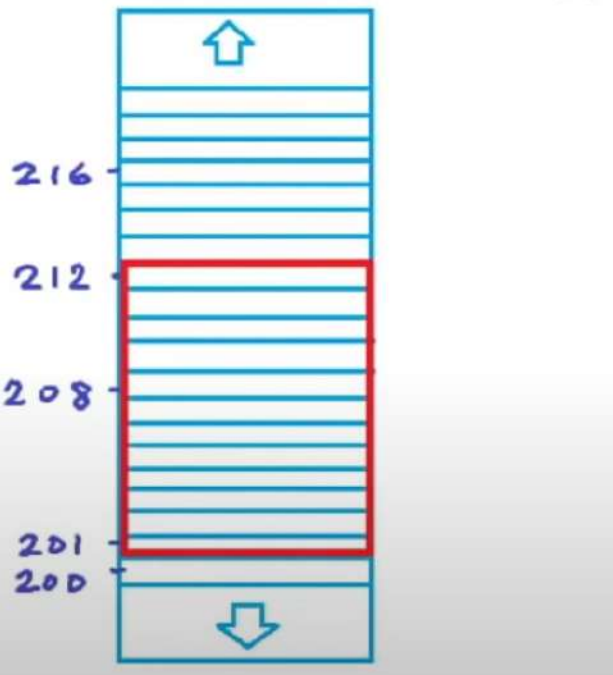
typecasting

```
int *P = (int *)malloc( 3 * sizeof(int))
```

no. of elements

size of one unit

```
Print P // 201
*P = 2
*(P+1) = 4
*(P+2) = 6
```



malloc, calloc, realloc, free

Allocate block of memory

unsigned int

malloc - void* malloc(size_t size)

typecasting

```
int *P = (int *)malloc(3 * sizeof(int))
```

Print P // 201

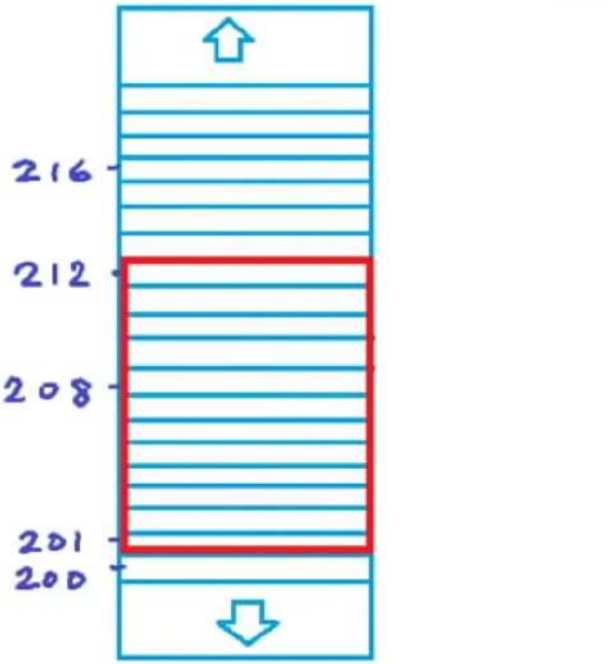
no. of elements

size of one unit

P[0] = 2

P[1] = 4

P[2] = 6



malloc, calloc, realloc, free

Allocate block of memory

malloc - void* malloc(size_t size)

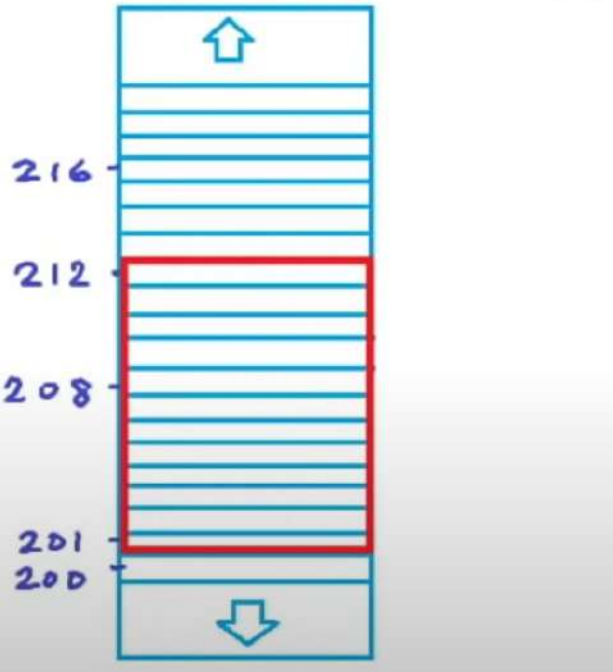
unsigned int

calloc - void* calloc(size_t num, size_t size)

```
int *p = (int *)calloc(3, sizeof(int));
```

no. of elements

size of data-type in bytes



malloc, calloc, realloc, free

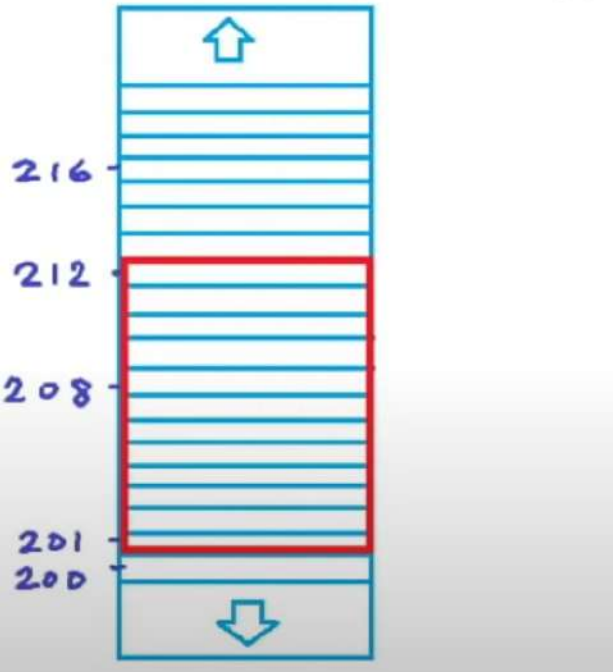
Allocate block of memory

unsigned int

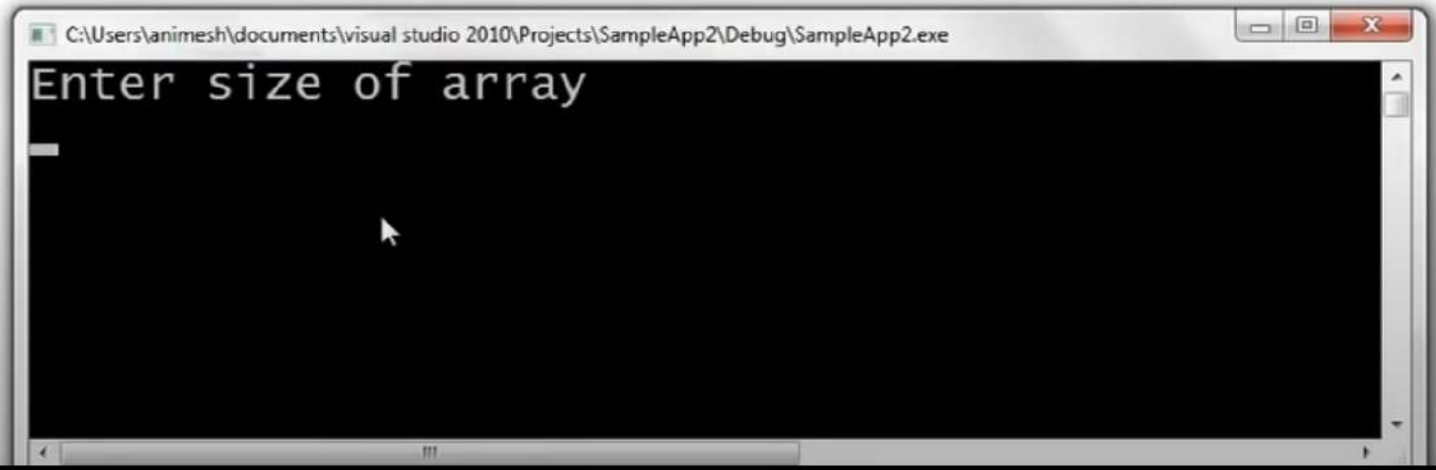
malloc - void* malloc(size_t size)

calloc - void* calloc(size_t num, size_t size)

realloc - void* realloc(void* ptr, size_t size)



```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int n;
    printf("Enter size of array\n");
    scanf("%d",&n);
    int *A = (int*)malloc(n*sizeof(int)); //dynamically allocated array
    for(int i = 0;i<n;i++)
    {
        A[i] = i+1;
    }
    for(int i =0;i<n;i++)
    {
        printf("%d ",A[i]);
    }
}
```



```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int n;
    printf("Enter size of array\n");
    scanf("%d",&n);
    int *A = (int*)calloc(n,sizeof(int)); //dynamically allocated array
    for(int i = 0;i<n;i++)
    {
        A[i] = i+1;
    }
    for(int i =0;i<n;i++)
    {
        printf("%d ",A[i]);
    }
}
```

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int n;
    printf("Enter size of array\n");
    scanf("%d",&n);
    int *A = (int*)calloc(n,sizeof(int)); //dynamically allocated array

    for(int i =0;i<n;i++)
    {
        printf("%d ",A[i]);
    }
}
```

```
C:\Users\animesh\documents\visual studio 2010\Projects\SampleApp2\Debug\SampleApp2.exe
Enter size of array
10
0 0 0 0 0 0 0 0 0 0
```

```c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int n;
    printf("Enter size of array\n");
    scanf("%d",&n);
    int *A = (int*)malloc(n*sizeof(int)); //dynamically allocated array

    for(int i =0;i<n;i++)
    {
        printf("%d ",A[i]);
    }
}
```

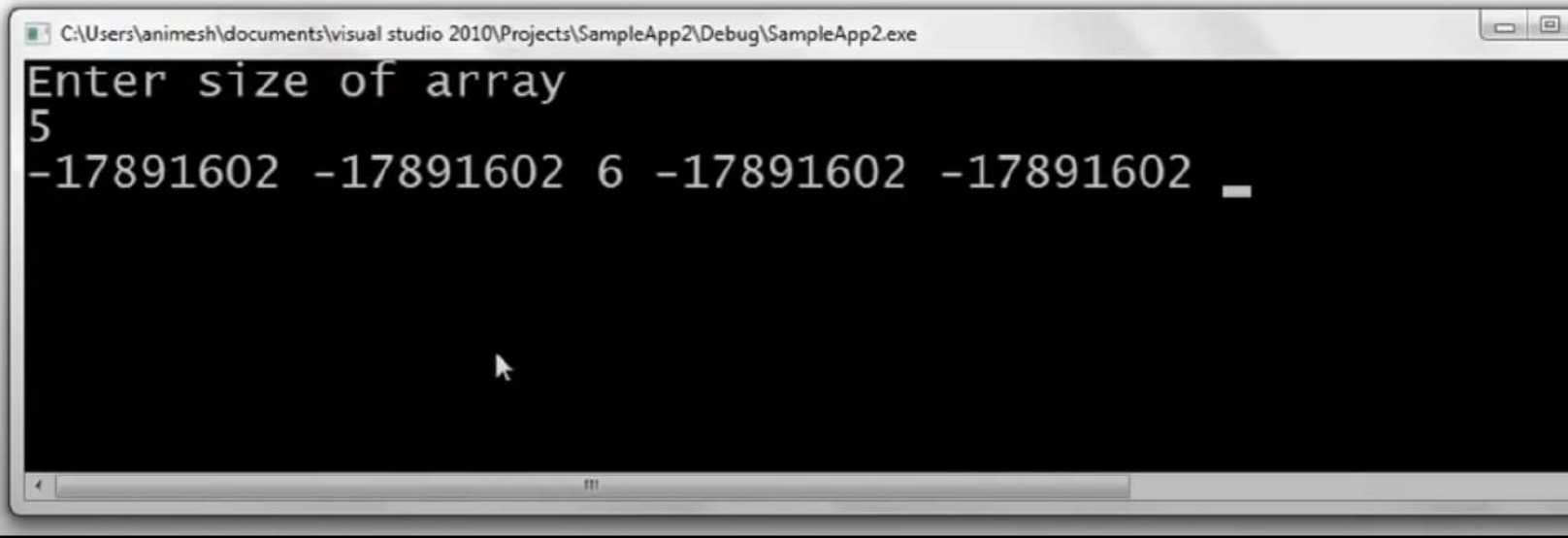
```
C:\Users\animesh\documents\visual studio 2010\Projects\SampleApp2\Debug\SampleApp2.exe
Enter size of array
10
-842150451 -842150451 -842150451 -842150451 -8
2150451 -842150451 -842150451
```

```c
int main()
{
    int n;
    printf("Enter size of array\n");
    scanf("%d",&n);
    int *A = (int*)malloc(n*sizeof(int)); //dynamically allocated array
    for(int i =0;i<n;i
    {
        A[i] = i+1;
    }
    free(A);
    for(int i = 0;i<n;
    {
        printf("%d ",A
    }
}
```

C:\Users\animesh\documents\visual studio 2010\Projects\SampleApp2\Debug\SampleApp2.exe

```
Enter size of array
5
-17891602 -17891602 -17891602 -17891602 -17891602
```

```c
int main()
{
    int n;
    printf("Enter size of array\n");
    scanf("%d",&n);
    int *A = (int*)malloc(n*sizeof(int)); //dynamically allocated array
    for(int i =0;i<n;i++)
    {
        A[i] = i+1;
    }
    free(A);
    A[2] = 6;
    for(int i = 0;i<n;i++)
    {
        printf("%d ",A[i]);
    }
}
```



```c
int main()
{
    int n;
    printf("Enter size of array\n");
    scanf("%d",&n);
    int *A = (int*)malloc(n*sizeof(int)); //dynamically allocated array
    for(int i =0;i<n;i++)
    {
        A[i] = i+1;
    }
    free(A);
    A[2] = 6; // value at address A+2
    for(int i = 0;i<n;i++)
    {
        printf("%d ",A[i]);
    }
}
```

```
C:\Users\animesh\documents\visual studio 2010\Projects\SampleApp2\Debug\SampleApp2.exe
Enter size of array
5
-17891602 -17891602 6 -17891602 -17891602
```

```c
int main()
{
    int n;
    printf("Enter size of array\n");
    scanf("%d",&n);
    int *A = (int*)malloc(n*sizeof(int)); //dynamically allocated array
    for(int i =0;i<n;i++)
    {
        A[i] = i+1;
    }
    free(A);
    A = NULL; // After free , adjust pointer to NULL
    for(int i = 0;i<n;i++)
    {
        printf("%d ",A[i]);
    }
}
```

```c
int main()
{
    int n;
    printf("Enter size of array\n");
    scanf("%d",&n);
    int *A = (int*)malloc(n*sizeof(int)); //dynamically allocated array
    for(int i =0;i<n;i++)
    {
        A[i] = i+1;
    }
    int *B = (int*)realloc(A, 2*n*sizeof(int));
    printf("Prev block address = %d, new address = %d\n",A,B);
    for(int i = 0;i<2*n;i++)
    {
        printf("%d\n",B[i]);
    }
}
```

```
int main()
{
    int n;
    printf("Enter size
    scanf("%d",&n);
    int *A = (int*)mall
    for(int i =0;i<n;i+
    {
        A[i] = i+1;
    }
    int *B = (int*)real
    printf("Prev block
    for(int i = 0;i<2*n
    {
        printf("%d\n",B
    }
}
```

C:\Users\animesh\documents\visual studio 2010\Projects\SampleApp2\Debug\SampleApp2.exe

```
Enter size of array
5
Prev block address = 9920128, new address = 9920128
1
2
3
4
5
-842150451
-842150451
-842150451
-842150451
-842150451
```

```c
int main()
{
    int n;
    printf("Enter size of array\n");
    scanf("%d",&n);
    int *A = (int*)malloc(n*sizeof(int)); //dynamically allocated array
    for(int i =0;i<n;i++)
    {
        A[i] = i+1;
    }
    int *B = (int*)realloc(A, 0);
    printf("Prev block address = %d, new address = %d\n",A,B);
    for(int i = 0;i<n;i++)
    {
        printf("%d\n",B[i]);
    }
}
```

```c
int main()
{
    int n;
    printf("Enter size of array\n");
    scanf("%d",&n);
    int *A = (int*)malloc(n*sizeof(int)); //dynamically allocated array
    for(int i =0;i<n;i++)
    {
        A[i] = i+1;
    }
    int *B = (int*)realloc(A, 0); //equivalent to free(A)
    printf("Prev block address = %d, new address = %d\n",A,B);
    for(int i = 0;i<n;i++)
    {
        printf("%d\n",B[i]);
    }
}
```

```c
int main()
{
    int n;
    printf("Enter size of array\n");
    scanf("%d",&n);
    int *A = (int*)malloc(n*sizeof(int)); //dynamically allocated array
    for(int i =0;i<n;i++)
    {
        A[i] = i+1;
    }
    int *A = (int*)realloc(A, 0); //equivalent to free(A)
    printf("Prev block address = %d, new address = %d\n",A,B);
    for(int i = 0;i<n;i++)
    {
        printf("%d\n",B[i]);
    }
}
```

```c
int main()
{
    int n;
    printf("Enter size of array\n");
    scanf("%d",&n);
    int *A = (int*)malloc(n*sizeof(int)); //dynamically allocated array
    for(int i =0;i<n;i++)
    {
        A[i] = i+1;
    }
    int *B = (int*)realloc(NULL, n*sizeof(int)); //equivalent to free(A)
    printf("Prev block address = %d, new address = %d\n",A,B);
    for(int i = 0;i<n;i++)
    {
        printf("%d\n",B[i]);
    }
}
```

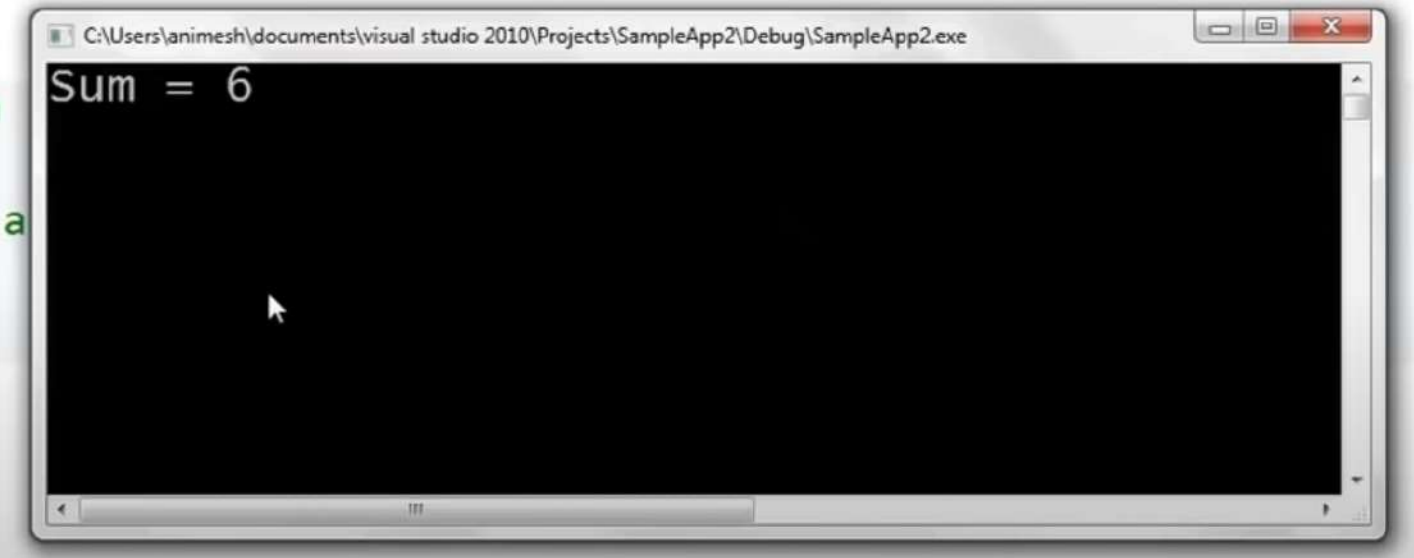
```c
// Pointers as function returns
#include<stdio.h>
#include<stdlib.h>
int Add(int* a,int* b){ // Called funtion
    // a and b are pointer to integers local to Add
    printf("Address of a in Add = %d\n",&a);
    printf("Value in a of Add (address of a of main) = %d\n",a);
    printf("Value at adress stored in a of Add = %d\n",*a);
    int c = (*a) + (*b);
    return c;
}
int main() {  //Calling function
    int a = 2, b =4;
    printf("Address of a in main = %d\n",&a);
    //Call by reference
    int c = Add(&a,&b); // a and b are integers local to Main
    printf("Sum = %d\n",c);
}
```

```c
// Pointers as function returns
#include<stdio.h>
#include<stdlib.h>
int Add(int* a,int* b){ // Called funtion
    // a and b are pointer to integers local to Add
    printf("Address of a in Add = %d\n",&a);
    printf("Value in a of Add
    printf("Value at adress s
    int c = (*a) + (*b);
    return c;
}
int main() {  //Calling funct
    int a = 2, b =4;
    printf("Address of a in m
    //Call by reference
    int c = Add(&a,&b); // a
    printf("Sum = %d\n",c);
}
```

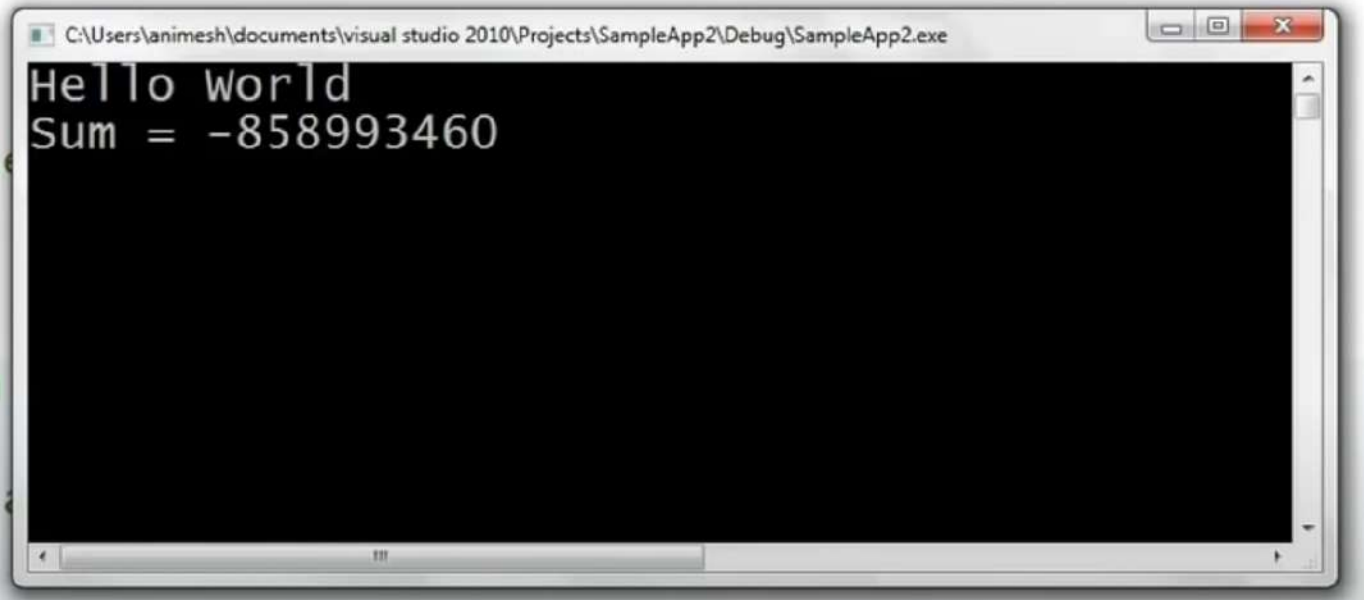
```
C:\Users\animesh\documents\visual studio 2010\Projects\SampleApp2\Debug\SampleApp2.exe
Address of a in main = 3537612
Address of a in Add = 3537372
Value in a of Add (address of a of main) = 3537612
Value at adress stored in a of Add = 2
Sum = 6
```

```c
// Pointers as function returns
#include<stdio.h>
#include<stdlib.h>
int* Add(int* a,int* b){ // Called funtion
    int c = (*a) + (*b);
    return &c;
}
int main() {  //Calling function
    int a = 2, b =4;
    printf("Address of a in main = %d\n",&a);
    int* ptr = Add(&a,&b); // a and b are integers local to Main
    printf("Sum = %d\n",*ptr);
}
```

```c
// Pointers as function returns
#include<stdio.h>
#include<stdlib.h>
int *Add(int* a,int* b){ // Called funtion  - returns pointer to integer
    int c = (*a) + (*b);
    return &c;
}
int main() {  //Calling function
    int a = 2, b =4;
    int* ptr = Add(&a,&b); // a a
    printf("Sum = %d\n",*ptr);
}
```

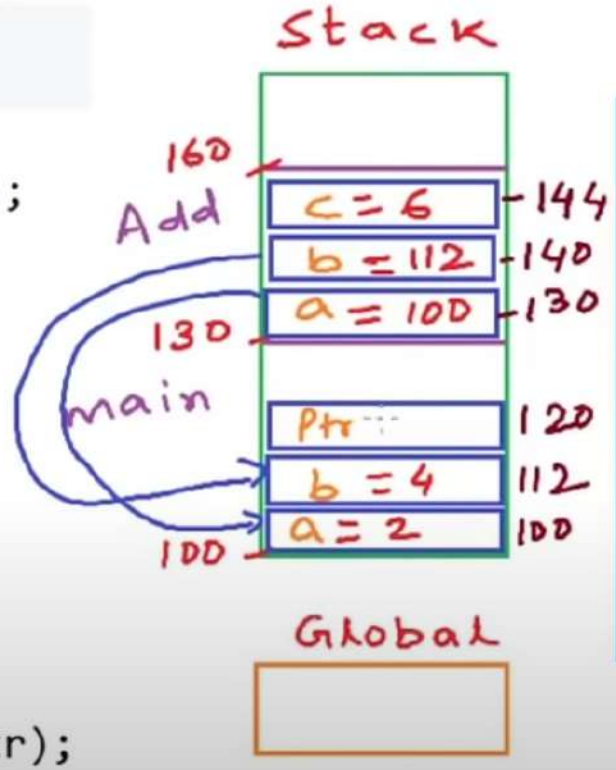


```c
// Pointers as function returns
#include<stdio.h>
#include<stdlib.h>
void PrintHelloWorld(){
    printf("Hello World\n");
}
int *Add(int* a,int* b){ // Calle
    int c = (*a) + (*b);
    return &c;
}
int main() {  //Calling function
    int a = 2, b =4;
    int* ptr = Add(&a,&b); // a
    PrintHelloWorld();
    printf("Sum = %d\n",*ptr);
}
```



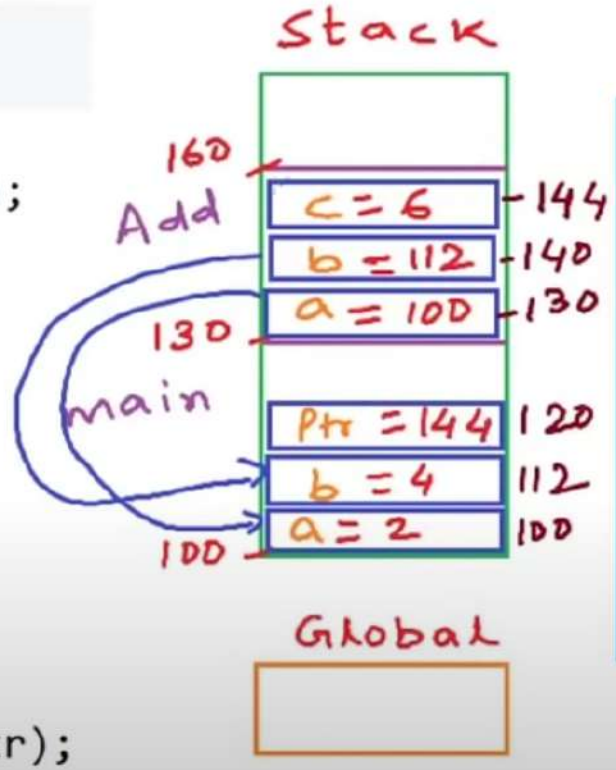
Pointers as function returns

```c
#include<stdio.h>
#include<stdlib.h>
void PrintHelloWorld(){
    printf("Hello World\n");
}
int *Add(int* a,int* b){
    int c = (*a) + (*b);
    return &c;
}
int main() {
  ✓int a = 2, b =4;
  ✓int* ptr = Add(&a,&b);
    PrintHelloWorld();
    printf("Sum = %d\n",*ptr);
}
```



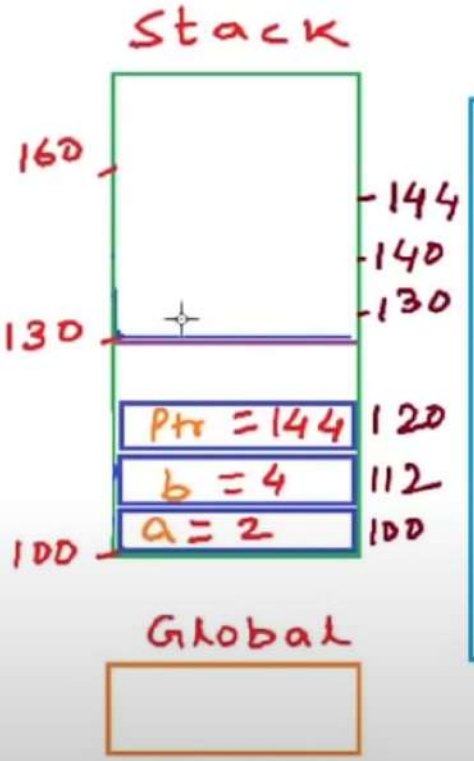
Pointers as function returns

```c
#include<stdio.h>
#include<stdlib.h>
void PrintHelloWorld(){
    printf("Hello World\n");
}
int *Add(int* a,int* b){
    int c = (*a) + (*b);
    return &c;
}
int main() {
  ✓int a = 2, b =4;
  ✓int* ptr = Add(&a,&b);
    PrintHelloWorld();
    printf("Sum = %d\n",*ptr);
}
```



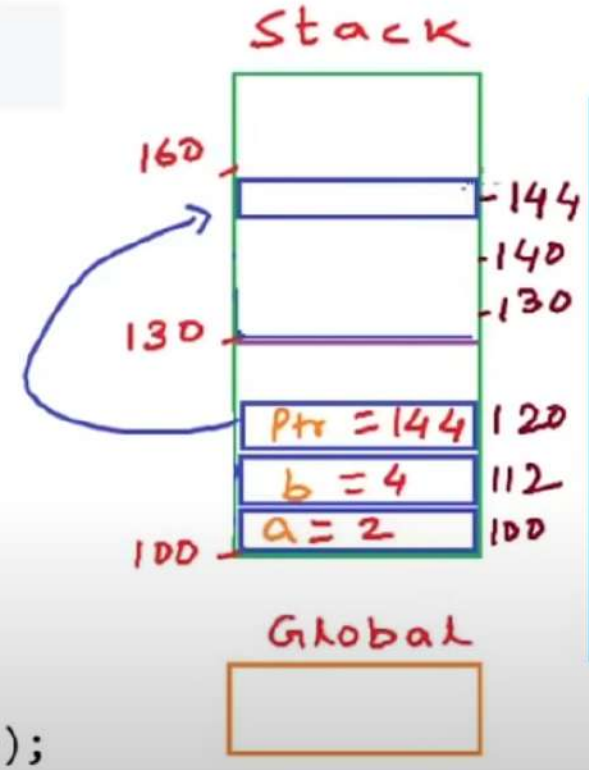
Pointers as function returns

```c
#include<stdio.h>
#include<stdlib.h>
void PrintHelloWorld(){
    printf("Hello World\n");
}
int *Add(int* a,int* b){
    int c = (*a) + (*b);
    return &c;
}
int main() {
  ✓int a = 2, b =4;
  ✓int* ptr = Add(&a,&b);
    PrintHelloWorld();
    printf("Sum = %d\n",*ptr);
}
```



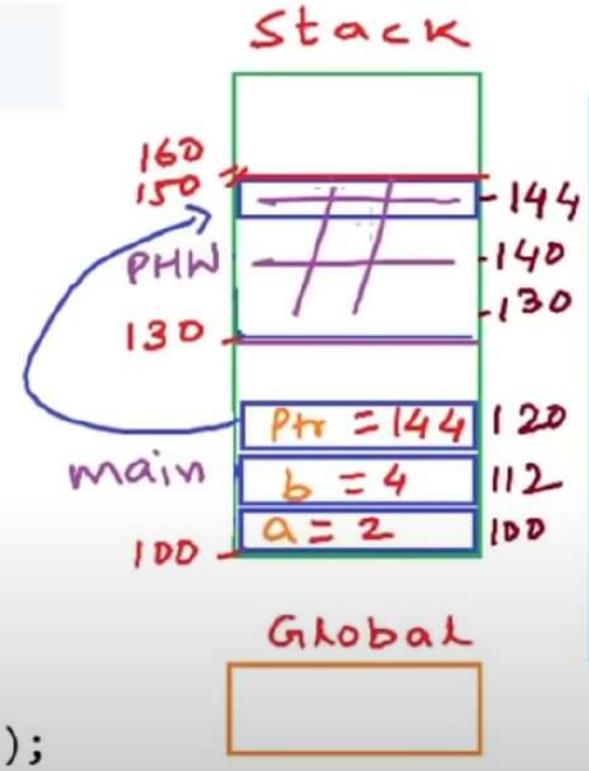
Pointers as function returns

```c
#include<stdio.h>
#include<stdlib.h>
void PrintHelloWorld(){
    printf("Hello World\n");
}
int *Add(int* a,int* b){
    int c = (*a) + (*b);
    return &c;
}
int main() {
  ✓int a = 2, b =4;
  ✓int* ptr = Add(&a,&b);
    PrintHelloWorld();
    printf("Sum = %d\n",*ptr);
}
```



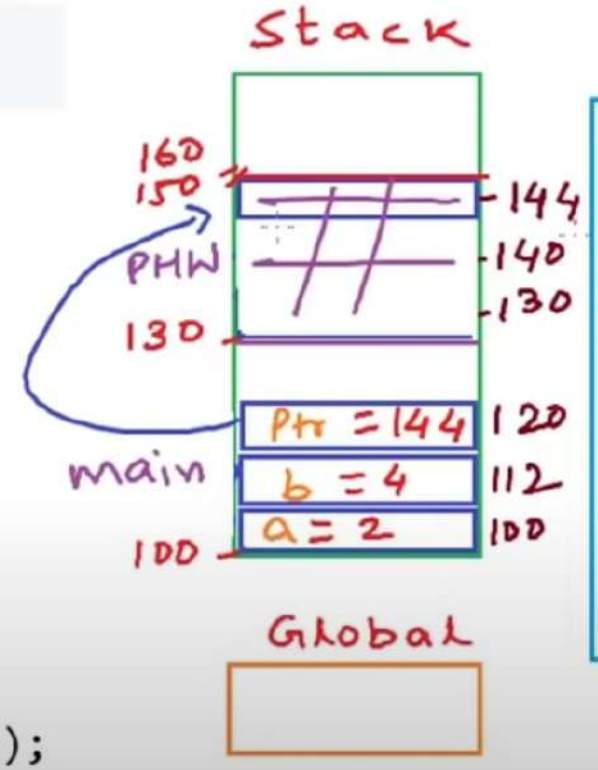
Pointers as function returns

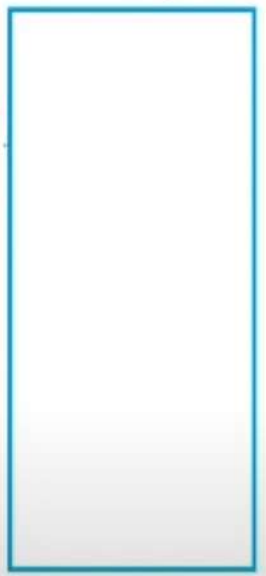
```c
#include<stdio.h>
#include<stdlib.h>
void PrintHelloWorld(){
    printf("Hello World\n");
}
int *Add(int* a,int* b){
    int c = (*a) + (*b);
    return &c;
}
int main() {
  int a = 2, b =4;
  int* ptr = Add(&a,&b);
  PrintHelloWorld();
    printf("Sum = %d\n",*ptr);
}
```



Pointers as function returns

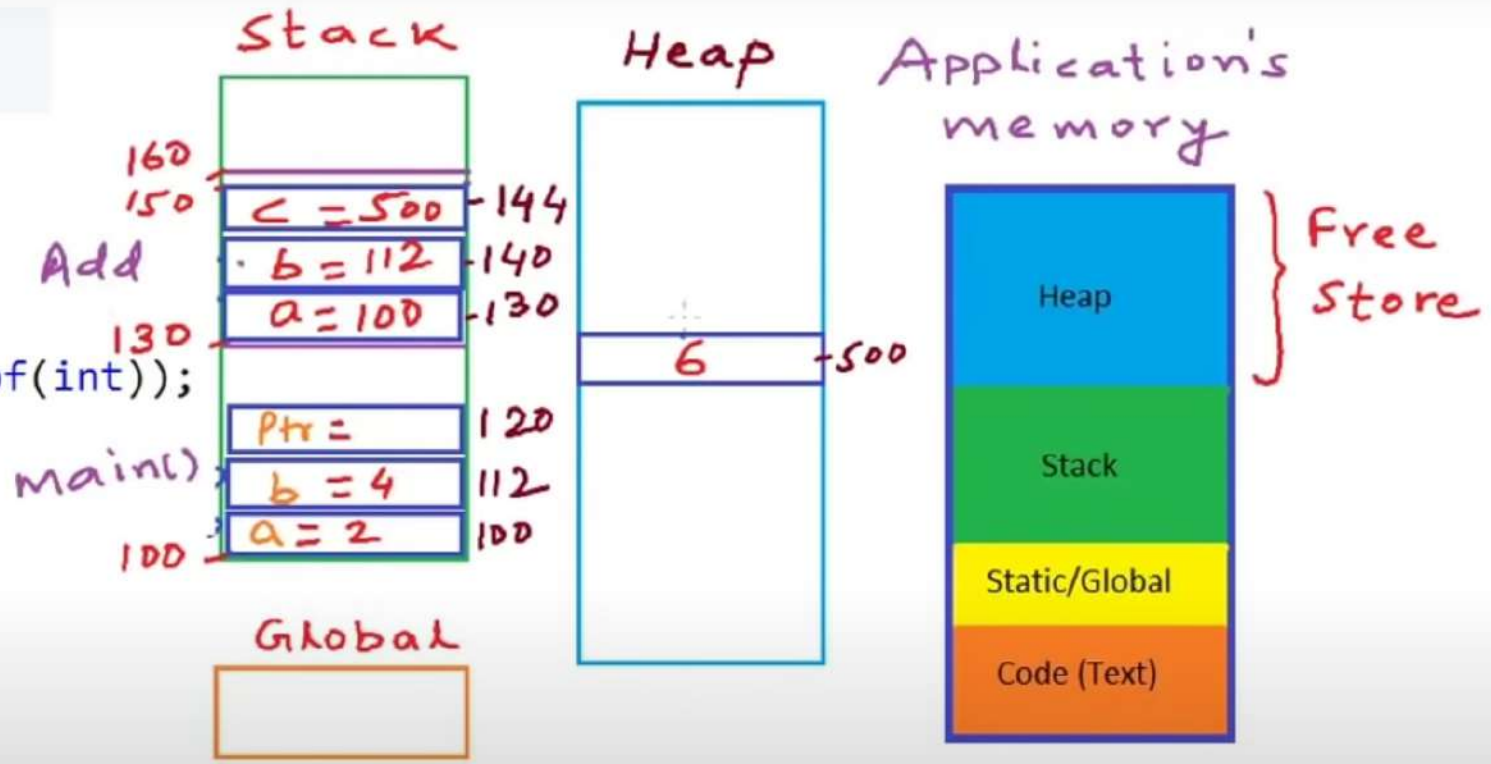
```c
#include<stdio.h>
#include<stdlib.h>
void PrintHelloWorld(){
    printf("Hello World\n");
}
int *Add(int* a,int* b){
    int c = (*a) + (*b);
    return &c;
}
int main() {
    int a = 2, b =4;
    int* ptr = Add(&a,&b);
    PrintHelloWorld();
    printf("Sum = %d\n",*ptr);
}
```



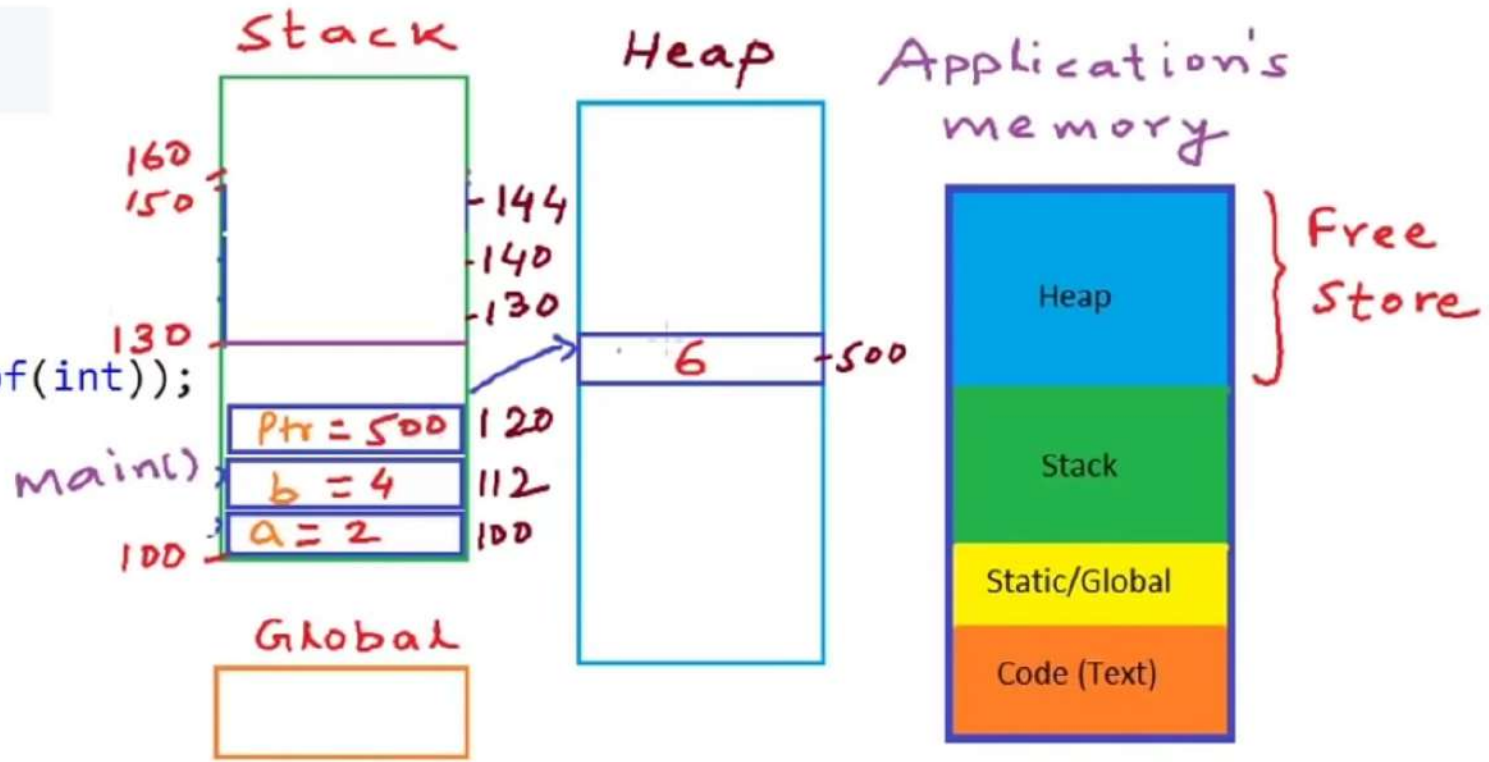


```c
// Pointers as function returns
#include<stdio.h>
#include<stdlib.h>
void PrintHelloWorld(){
    printf("Hello World\n");
}
int *Add(int* a,int* b){ // Called funtion  - returns pointer to integer
    int* c = (int*)malloc(sizeof(int));
    *c = (*a) + (*b);
    return c;
}
int main() {  //Calling function
    int a = 2, b =4;
    int* ptr = Add(&a,&b); // a and b are integers local to Main
    PrintHelloWorld();
    printf("Sum = %d\n",*ptr);
}
```

```c
#include<stdio.h>
#include<stdlib.h>
void PrintHelloWorld(){
    printf("Hello World\n");
}
int *Add(int* a,int* b){
  int* c = (int*)malloc(sizeof(int));
  *c = (*a) + (*b);
  return c;
}
int main() {
  int a = 2, b =4;
  int* ptr = Add(&a,&b);
  PrintHelloWorld();
  printf("Sum = %d\n",*ptr);
}
```

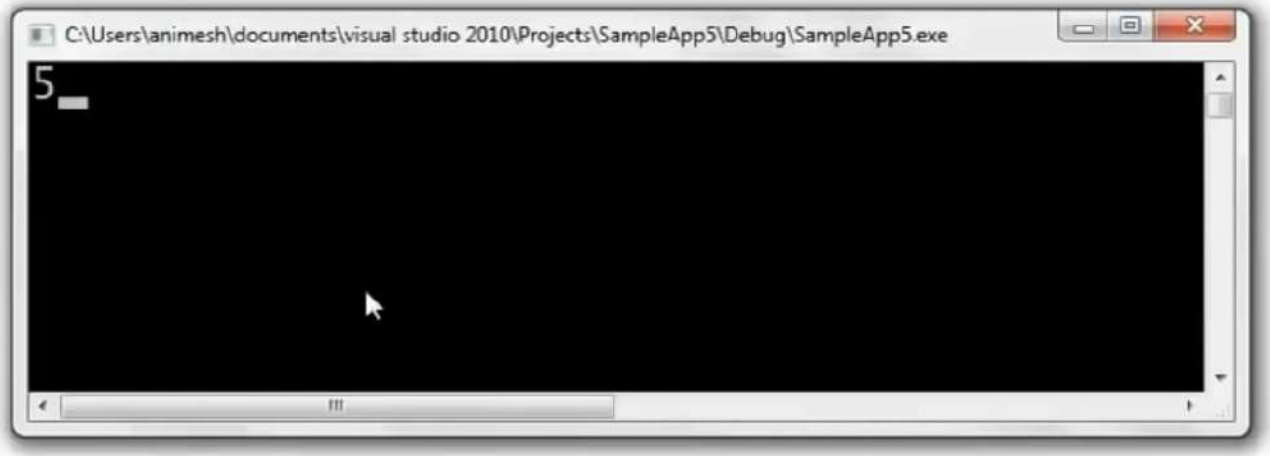


```c
#include<stdio.h>
#include<stdlib.h>
void PrintHelloWorld(){
    printf("Hello World\n");
}
int *Add(int* a,int* b){
  int* c = (int*)malloc(sizeof(int));
  *c = (*a) + (*b);
  return c;

}
int main() {
  int a = 2, b =4;
  int* ptr = Add(&a,&b);
  PrintHelloWorld();
  printf("Sum = %d\n",*ptr);
}
```



```c
//Function Pointers in C/C++
#include<stdio.h>
int Add(int a,int b)
{
    return a+b;
}
int main()
{
    int c;
    int (*p)(int,int);
    p = &Add;
    c = (*p)(2,3); //de-referencing and executing the function.
    printf("%d",c);
}
```

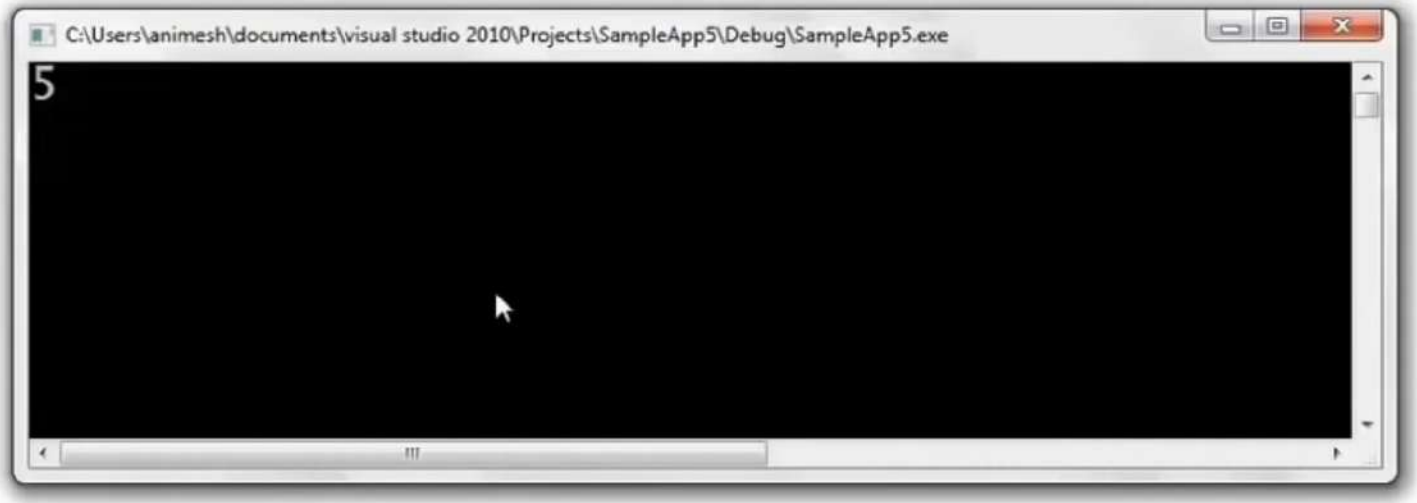
```c
//Function Pointers in C/C++
#include<stdio.h>
int Add(int a,int b)
{
    return a+b;
}
int main()
{
    int c;
    int (*p)(int,int);
    p = &Add;
    c = (*p)(2,3); //de-referencing and executing the function.
    printf("%d",c);
}
```



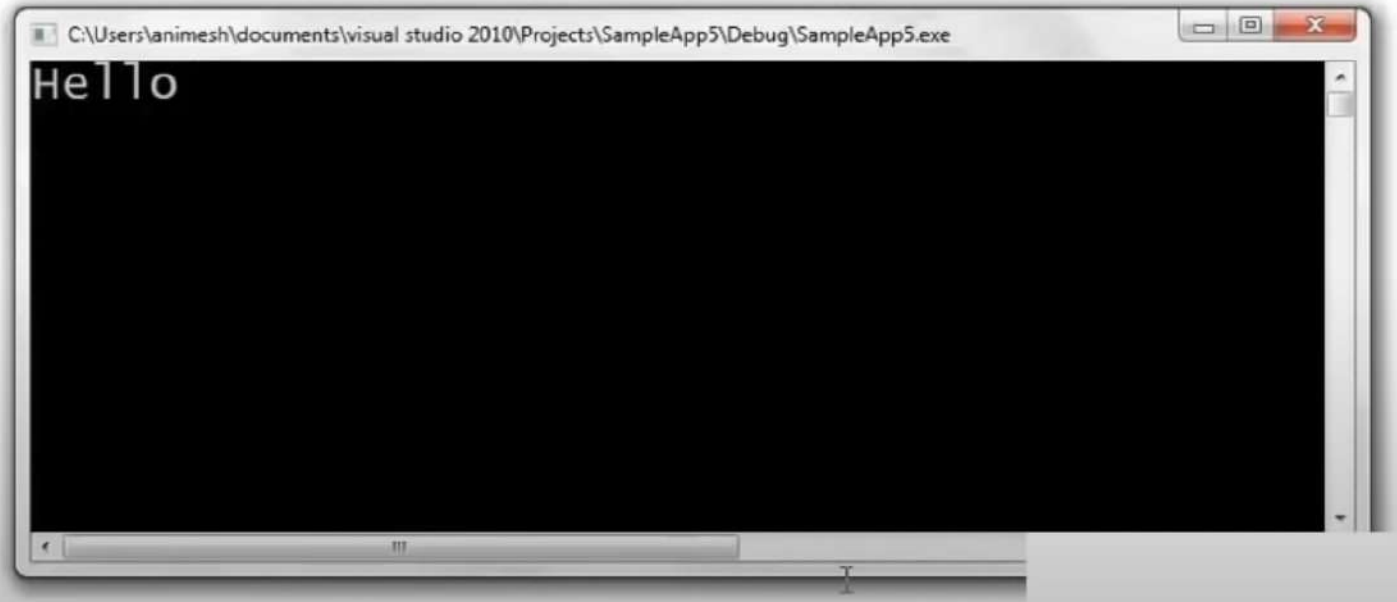
```c
//Function Pointers in C/C++
#include<stdio.h>
int Add(int a,int b)
{
    return a+b;
}
int main()
{
    int c;
    int (*p)(int,int);
    p = Add; // function name will return us pointer
    c = (*p)(2,3); //de-referencing and executing the function.
    printf("%d",c);
}
```

```c
//Function Pointers in C/C++
#include<stdio.h>
int Add(int a,int b)
{
    return a+b;
}
int main()
{
    int c;
    int (*p)(int,int);
    p = Add; // function name will return us pointer
    c = p(2,3); //de-referencing and executing the function.
    printf("%d",c);
}
```

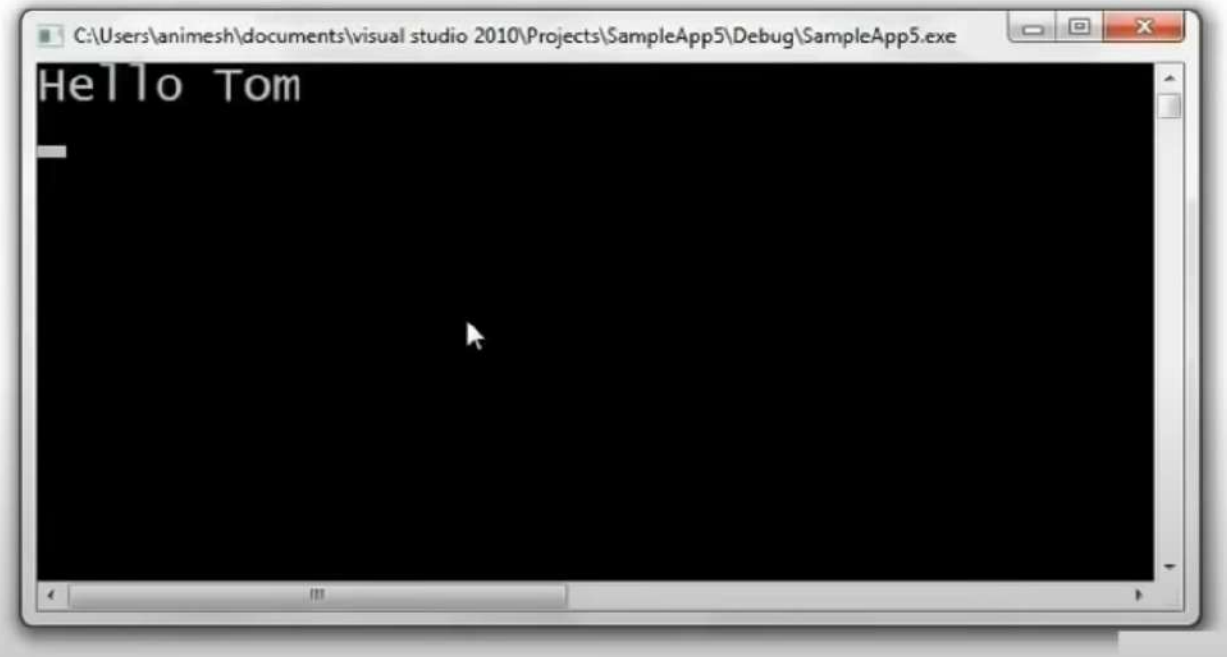
```c
//Function Pointers in C/C++
#include<stdio.h>
int Add(int a,int b)
{
    return a+b;
}
int main()
{
    int c;
    int (*p)(int,int);
    p = Add; // function name will return us pointer
    c = p(2,3); //de-referencing and executing the function.
    printf("%d",c);
}
```



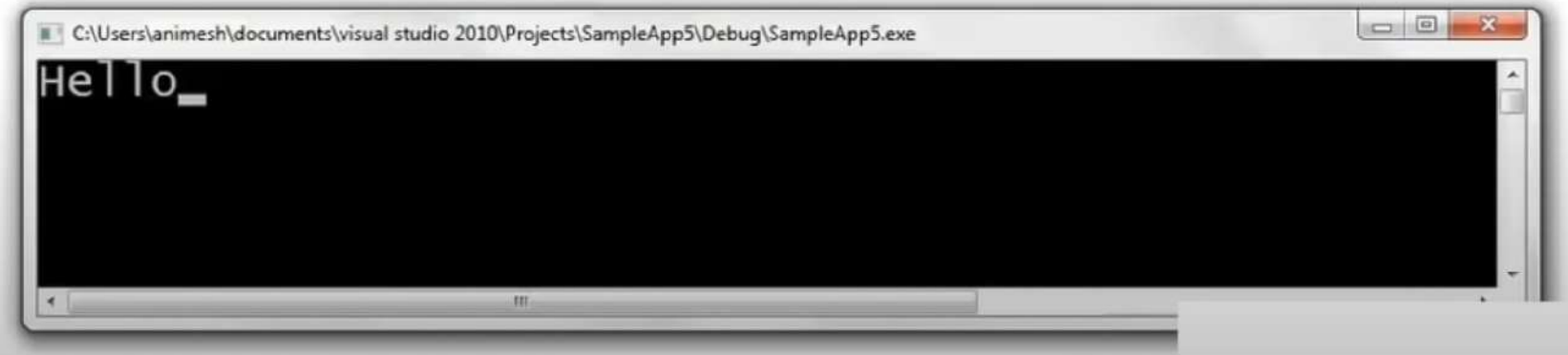
```c
//Function Pointers in C/C++
#include<stdio.h>
void PrintHello()
{
    printf("Hello\n");
}
int Add(int a,int b)
{
    return a+b;
}
int main()
{
    void (*ptr)();
    ptr = PrintHello;
    ptr();
}
```



```c
//Function Pointers in C/C++
#include<stdio.h>
void PrintHello(char *name)
{
    printf("Hello %s\n",name);
}
int Add(int a,int b)
{
    return a+b;
}
int main()
{
    void (*ptr)(char*);
    ptr = PrintHello;
    ptr("Tom");
}
```



```c
//Function Pointers and callbacks
#include<stdio.h>
void A()
{
    printf("Hello");
}
void B(void (*ptr)()) // function pointer as argument
{
    ptr(); //call-back function that "ptr" points to
}
int main()
{
    void (*p)() = A;
    B(p);
}
```

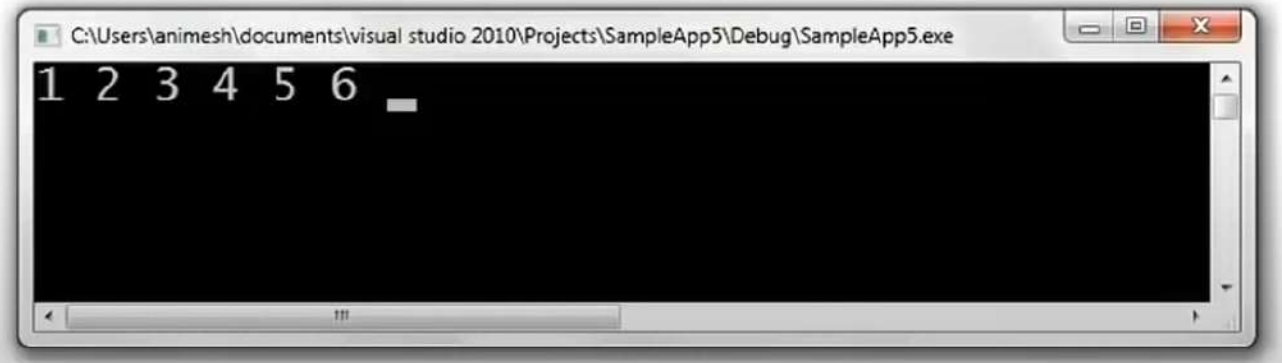


```c
//Function Pointers and callbacks
#include<stdio.h>
void A()
{
    printf("Hello");
}
void B(void (*ptr)()) // function pointer as argument
{
    ptr(); //call-back function that "ptr" points to
}
int main()
{
    B(A);
}
```

A 0x009913c0 A(void)

```c
#include<stdio.h>
int compare(int a,int b)
{
    if(a > b) return 1;
    else return -1;
}
void BubbleSort(int *A,int n,int (*compare)(int,int)) {
    int i,j,temp;
    for(i =0; i<n; i++)
        for(j=0; j<n-1; j++) {
            if(compare(A[j],A[j+1]) > 0) { //compare A[j] with A[j+1] and SWAP if needed
                temp = A[j];
                A[j] = A[j+1];
                A[j+1] = temp;
            }
        }
}
int main() {
    int i, A[] ={3,2,1,5,6,4};
    BubbleSort(A,6,compare);
    for(i =0;i<6;i++) printf("%d ",A[i]);
}
```

```c
#include<stdio.h>
int compare(int a,int b)
{
    if(a > b) return 1;
    else return -1;
}
void BubbleSort(int *A,int n,int (*compare)(int,int)) {
    int i,j,temp;
    for(i =0; i<n; i++)
        for(j=0; j<n-1; j++) {
            if(compare(A[j],A[j+1]) > 0) { //compare A[j] with A[j+1] and SWAP if needed
                temp = A[j];
                A[j] = A[j+1];
                A[j+1] = temp;
            }
        }
}
int main() {
    int i, A[] ={3,2,1,5,6,4};
    BubbleSort(A,6,compare);
    for(i =0;i<6;i++) printf("%d ",A[i]);
}
```



```c
//Function Pointers and callbacks
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int compare(const void* a, const void* b)
{
    int A = *((int*)a); // tyepcasting to int* and getting value
    int B = *((int*)b);
    return A-B;
}
int main() {
    int i, A[] ={-31,22,-1,50,-6,4}; // => {-1,4,-6,22,-31,50}
    qsort(A,6,sizeof(int),
}
```

```c
//Function Pointers and callbacks
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int compare(const void* a, cons
{
    int A = *((int*)a); // tyep
    int B = *((int*)b);
    return A-B;
}
int main() {
    int i, A[] ={-31,22,-1,50,-6,4}; // => {-1,4,-6,22,-31,50}
    qsort(A,6,sizeof(int),compare);
    for(i = 0;i<6;i++) printf("%d ",A[i]);
}
```