BBM 201 Data Structures
Hacettepe University

Lecture 3: Pointers and Arrays

Lecturer: Lale Özkahya

Resources:
Lecture Notes by Jason Zych, "Data Structures"
BBM201 Notes by Mustafa Ege and Sevil Şen
Lecture Videos
www.mycodeschool.com/videos/pointers-and-arrays

Outline

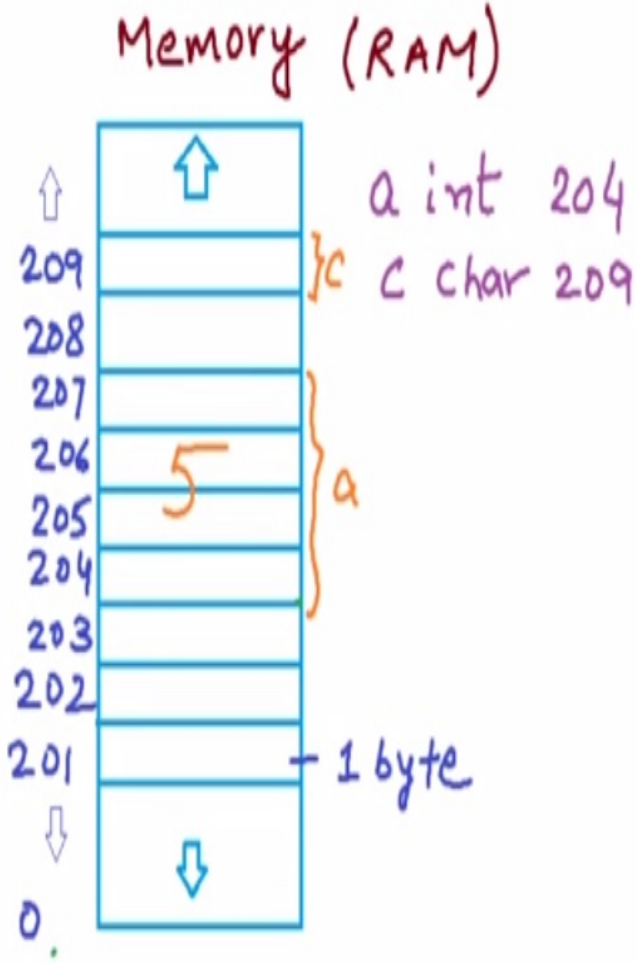
Introduction to pointers in C

int - 4 bytes

char - 1 byte

float - 4 bytes

```
int a;
char c;
a = 5;
```



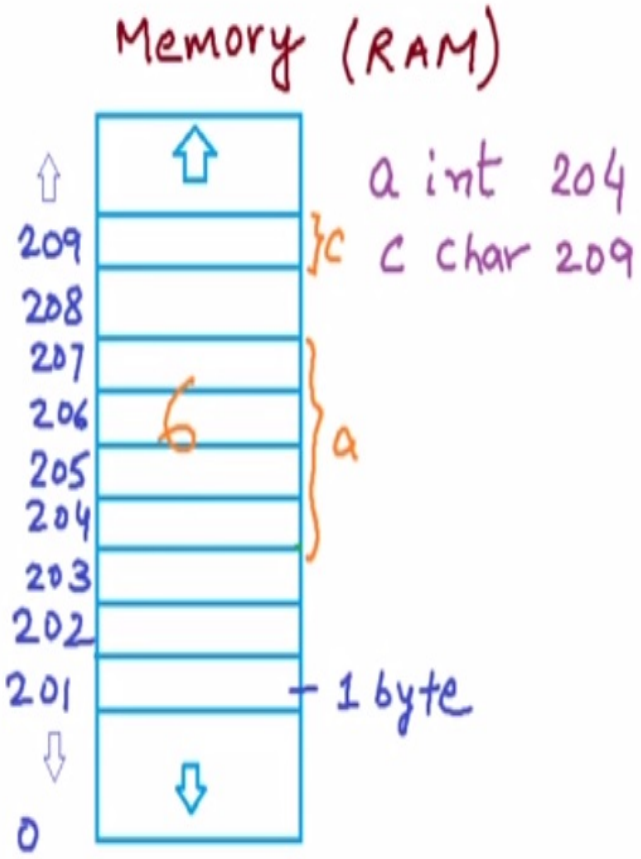
Introduction to pointers in C

int - 4 bytes

char - 1 byte

float - 4 bytes

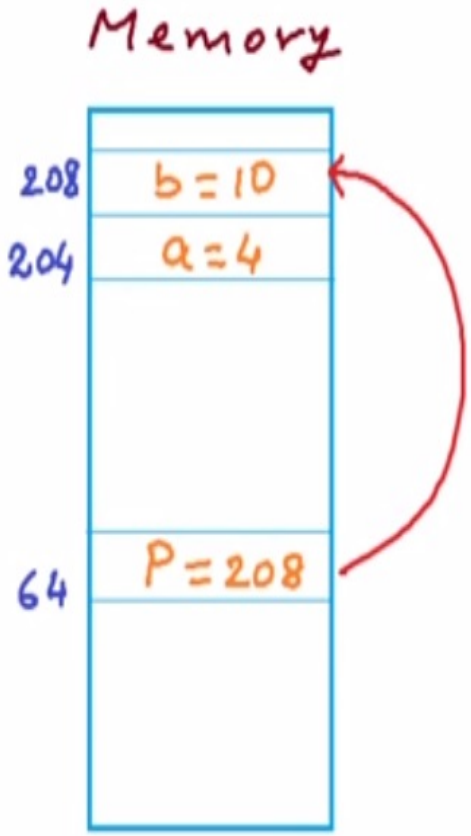
```
int a;
char c;
a = 5;
...
...
a++;
```



Introduction to pointers in C

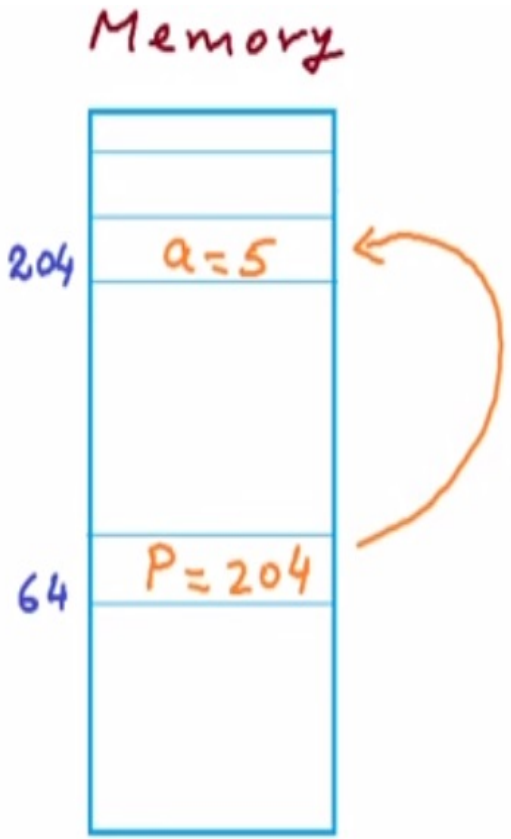
Pointers - variables that store address of another variable

```
int a;
int *p;
p = &a;
```



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  ⇒ dereferencing
```



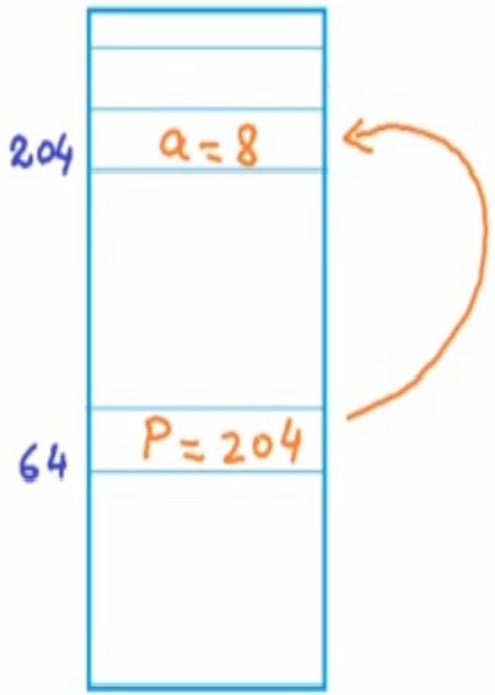
Pointers - variables that store address of another variable

P → address

*P → value at address

```
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
```

Memory



```c
#include<stdio.h>
int main()
{
    int a;  int *p;
    a = 10;
    p = &a; // &a =
    printf("Address
    printf("Value at
    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);
}
```

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

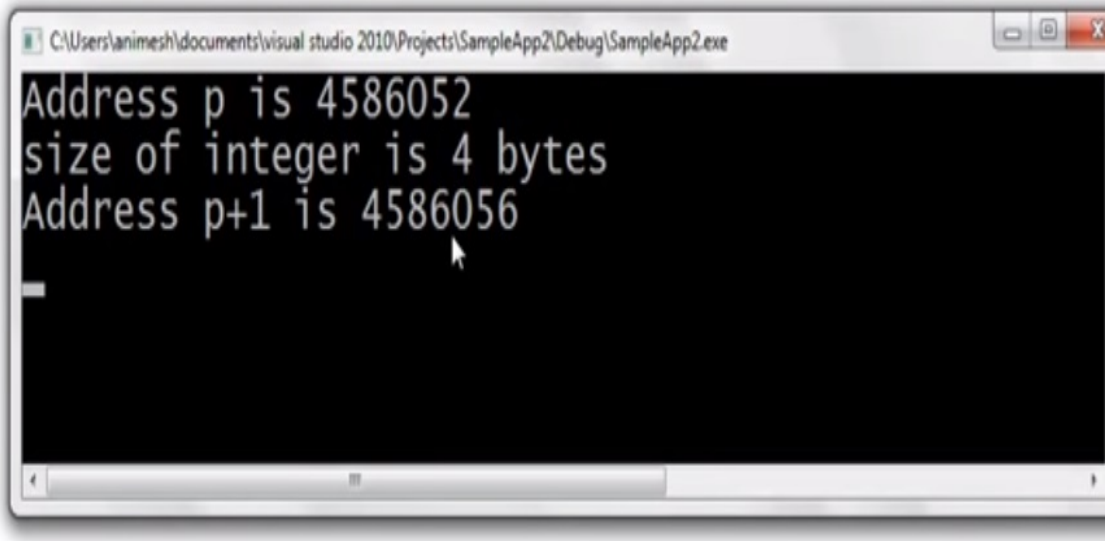
```
Address of P is 3144576
Value at p is 10
Address of P is 3144576
Value at p is 20
```

```c
#include<stdio.h>
int main()
{
    int a = 10;
    int *p  = &a;
    // p = &a; // &a = address of a
}
```

```c
#include<stdio.h>
int main()
{
    int a = 10;
    int *p;
    p = &a;
    // Pointer arithmetic
    printf("%d\n",p); // p is 2002
    printf("%d\n",p+1); // p+1 is ??
}
```

```c
#include<stdio.h>
int main()
{
    int a = 10;
    int *p;
    p = &a;
    // Pointer arithmetic
    printf("Address p is %d\n",p); // p is 2002
    printf("size of integer is %d bytes\n",sizeof(int));
    printf("Address p+1 is %d\n",p+1); // p+1 is 2006
}
```

```c
#include<stdio.h>
int main()
{
    int a = 10;
    int *p;
    p = &a;
    // Pointer arithmetic
    printf("Address p is %d\n",p); // p is 2002
    printf("size of integer is %d bytes\n",sizeof(int));
    printf("Address p+1 is %d\n",p+1); // p+1 is 2006
}
```



```c
#include<stdio.h>
int main()
{
    int a = 10;
    int *p;
    p = &a;
    // Pointer arithmetic
    printf("Address p is %d\n",p);
    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));
}
```

```c
#include<stdio.h>
int main()
{
    int a = 10;
    int *p;
    p = &a;
    // Pointer arithmetic
    printf("Address p is %d\n",p);
    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));
}
```

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

```
Address p is 2750832
value at address p is 10
size of integer is 4 bytes
Address p+1 is 2750836
value at address p+1 is -858993460
```

```c
#include<stdio.h>
int main()
{
    int a = 10;
    int *p;
    p = &a;
    // Pointer arithmetic
    printf("Address p is %d\n",p);
    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));
}
```

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

```
Address p is 2750832
value at address p is 10
size of integer is 4 bytes
Address p+1 is 2750836
value at address p+1 is -858993460
```

Outline

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
```

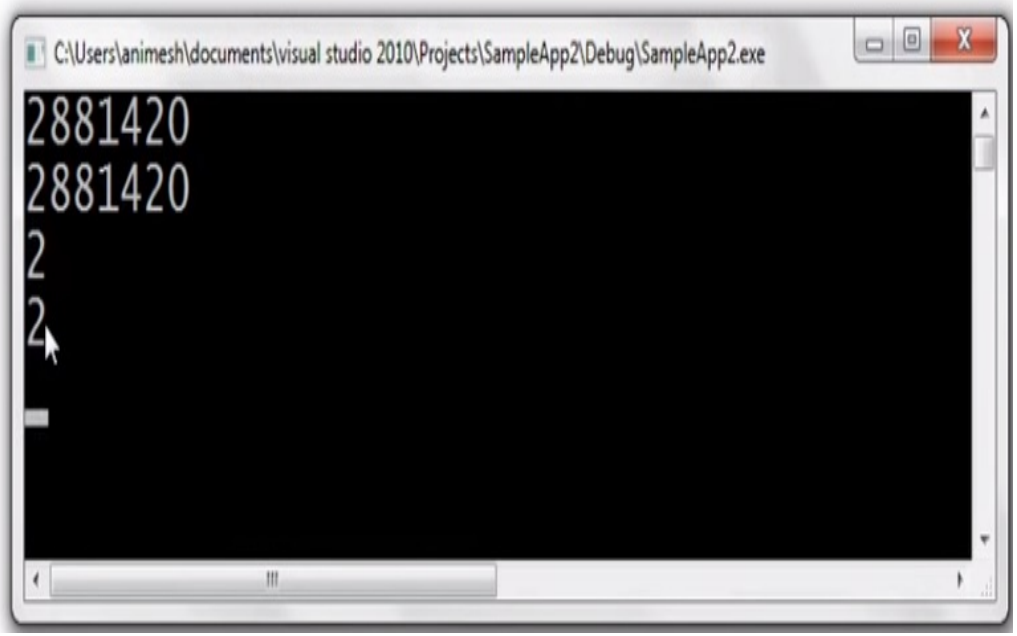
Element at index i –

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};
    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};
    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(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));
    }
}
```

```
// Pointers and Arrays
#incl
int m
{
    i
    i
    f
    {
    }
```

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

```
Address = 3734748
Address = 3734748
value = 2
value = 2
Address = 3734752
Address = 3734752
value = 4
value = 4
Address = 3734756
Address = 3734756
value = 5
value = 5
Address = 3734760
Address = 3734760
value = 8
value = 8
Address = 3734764
```

```c
// Pointers and Arrays
#include<stdio.h>
int main()
{
    int A[] ={2,4,5,8,1};
    int i;
    int *p = A;
    A++;
    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));
    }
```

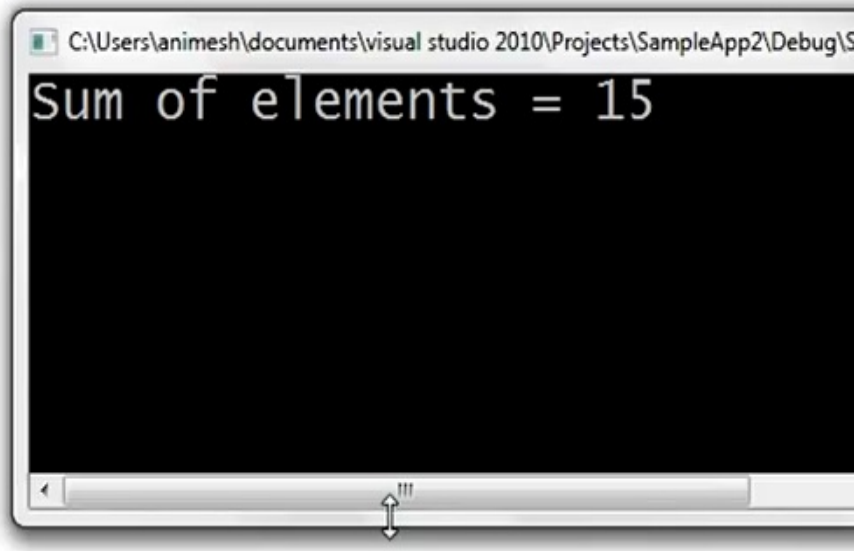
```c
// Pointers and Arrays
#include<stdio.h>
int main()
{
    int A[] ={2,4,5,8,1};
    int i;
    int *p = A;
    A++;// invalid
    p++ // valid
    for(int i = 0;i<5;i++)
    {
        printf("Address = %d\n",&A[i]);
        printf("Address = %d\n",A+i);
        printf("value = %d\n",A[i]);
    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

```c
// Pointers and Arrays
#include<stdio.h>
int main()
{
    int A[] ={2,4,5,8,1};
    int i;
    int *p = A;
    p++;
    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));
    }
```

Outline

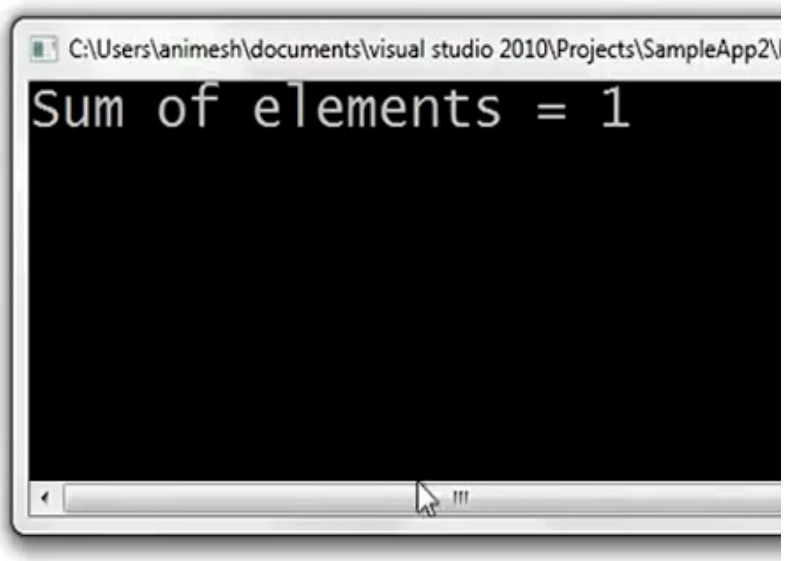
```c
// Arrays as function arguments
#include<stdio.h>
int SumOfElements(int A[], int size)
{
    int i, sum = 0;
    for(i = 0;i< size;i++)
    {
        sum+= A[i];
    }
    return sum;
}
int main()
{
    int A[]= {1,2,3,4,5};
    int size = sizeof(A)/sizeof(A[0]);
    int total = SumOfElements(A,size);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

```c
// 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
// Arrays as function arguments
#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);
```

```c
// Arrays as function arguments
#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);
```



```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",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);
    printf("Main - Size of A = %d, size of A[0] = %d",sizeof(A),sizeof(A[0]));
```

```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]));
```

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
```

Arrays as function arguments

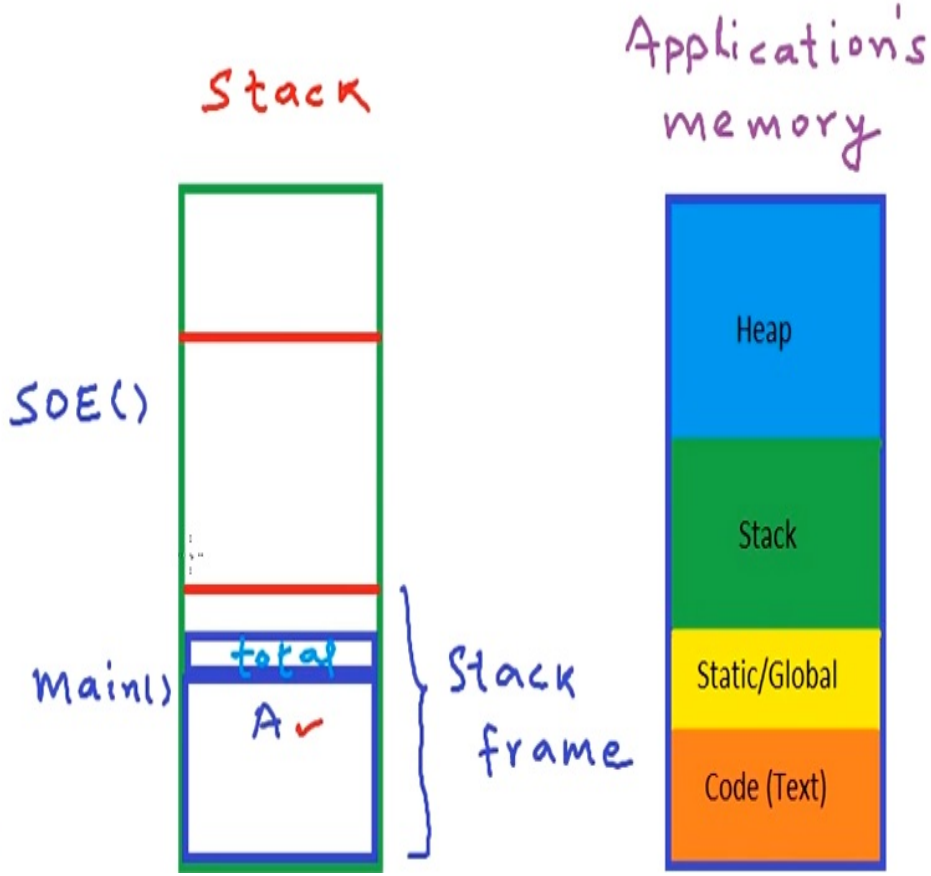
```c
#include<stdio.h>
int SumOfElements(int A[])
{
    int i, sum = 0;
    int size = sizeof(A)/sizeof(A[0]);
    for(i = 0;i< size;i++)
    {
        sum+= A[i];
    }
    return sum;
}
int main()
{
    int A[]= {1,2,3,4,5};
    int total = SumOfElements(A);
    printf("Sum of elements = %d\n",total);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

Stack

Application's memory

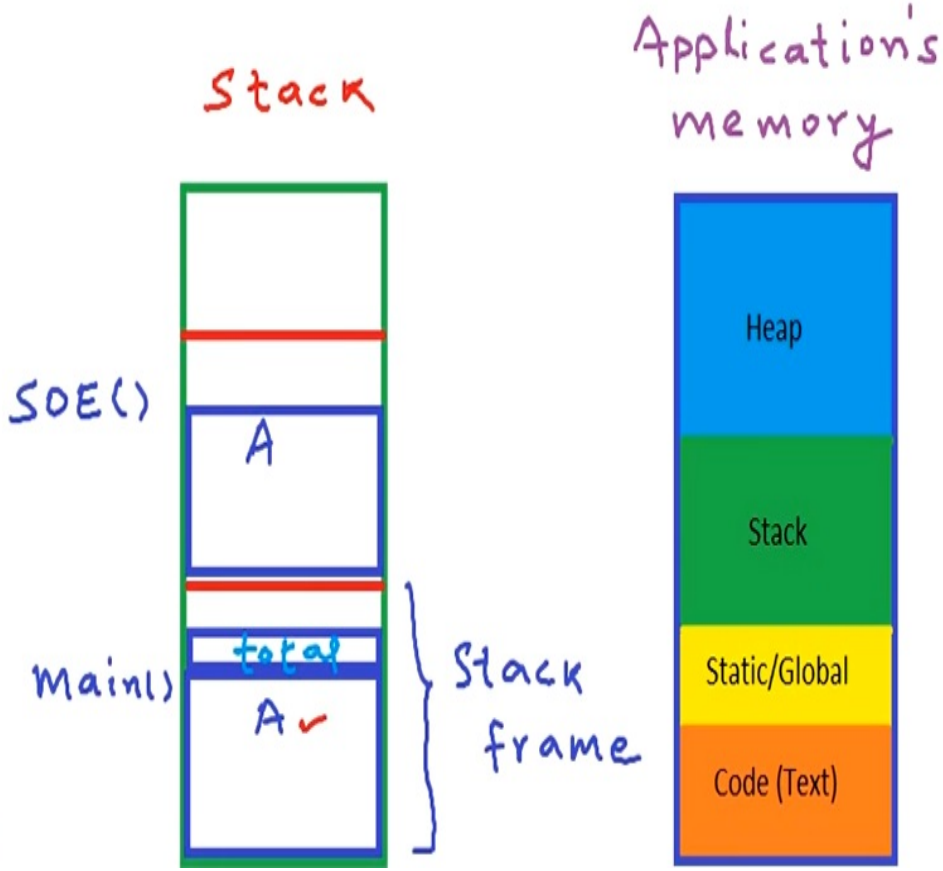
Arrays as function arguments

```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

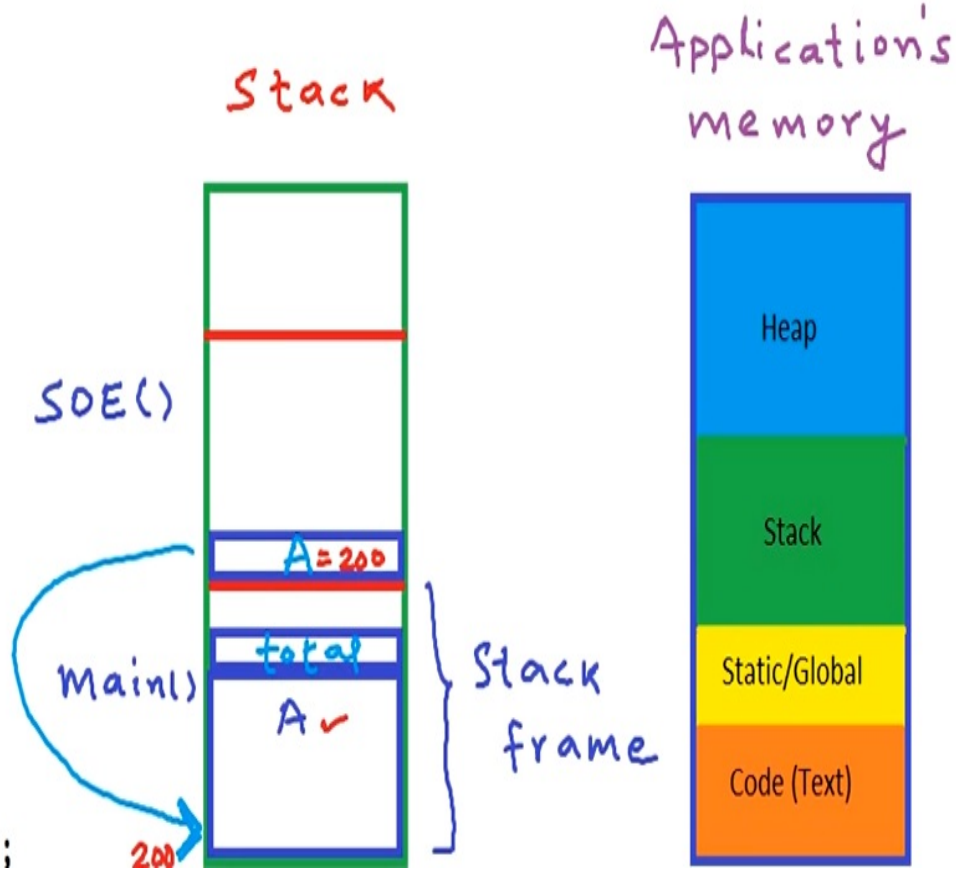
```c
#include<stdio.h>
int SumOfElements(int A[])
{
    int i, sum = 0;
    int size = sizeof(A)/sizeof(A[0]);
    for(i = 0;i< size;i++)
    {
        sum+= A[i];
    }
    return sum;
}
int main()
{
    int A[]= {1,2,3,4,5};
    int total = SumOfElements(A);
    printf("Sum of elements = %d\n",total);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



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)



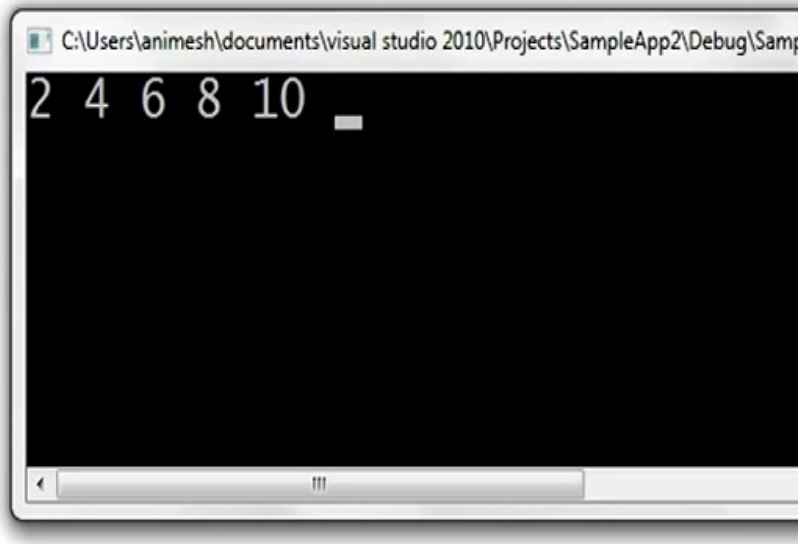
```c
#include<stdio.h>
int SumOfElements(int A[])
{
    int i, sum = 0;
    int size = sizeof(A)/sizeof(A[0]);
    printf("SOE - Size of A = %d, size of A[0] = %d\n",sizeof(A),sizeof(A[0]));
    for(i = 0;i< size;i++)
    {
        sum+= A[i];
    }
    return sum;
}
int main()
{
    int A[]= {1,2,3,4,5};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

```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",total);
    printf("Main - Size of A = %d, size of A[0] = %d\n",sizeof(A),sizeof(A[0]));
```

```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
#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]);
    }
```



Outline

Character arrays and pointers

String :- group of characters

eg:- "John"

"Hello World"

"I am feeling lucky"

Character arrays and pointers

1) How to store strings

Size of array $\geq$ no. of characters in string + 1

"John" Size ≥ 5

Character arrays and pointers

1) How to store strings

Size of array $\geq$ no. of characters in string + 1

"John" Size ≥ 5

Character arrays and pointers

1) How to store strings

Size of array ≥ no. of characters in string + 1

"John" Size ≥ 5

```
char c[8];
```

	0	1	2	3	4	5	6	7
c	J	O	H	N				

```
c[0] = 'J'; c[1] = 'O'; c[2] = 'H'; c[3] = 'N';
```

Character arrays and pointers

1) How to store strings

Size of array ≥ no. of characters in string + 1

"John" Size ≥ 5

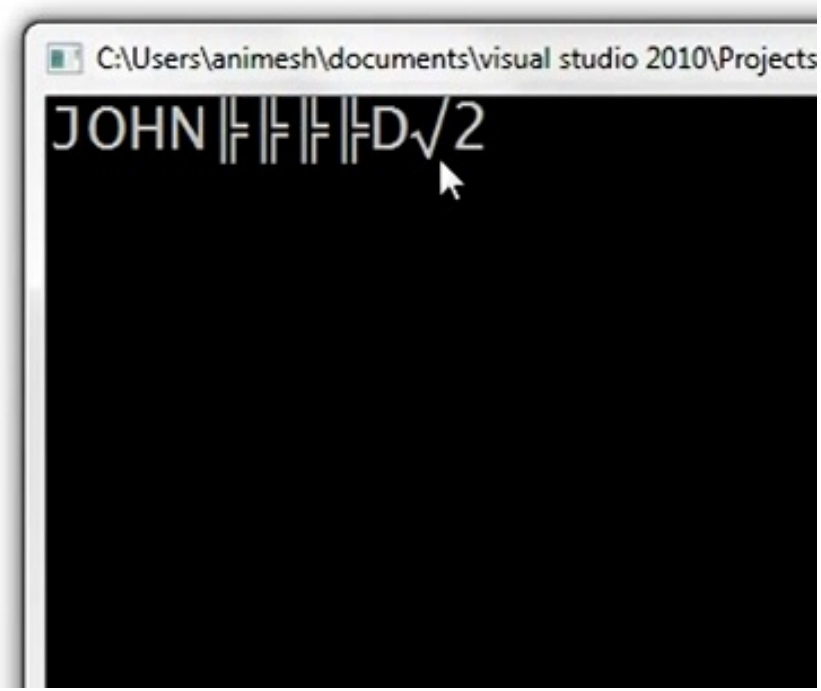
```
char c[8];
```

	0	1	2	3	4	5	6	7
c	J	O	H	N	\0	//	//	//

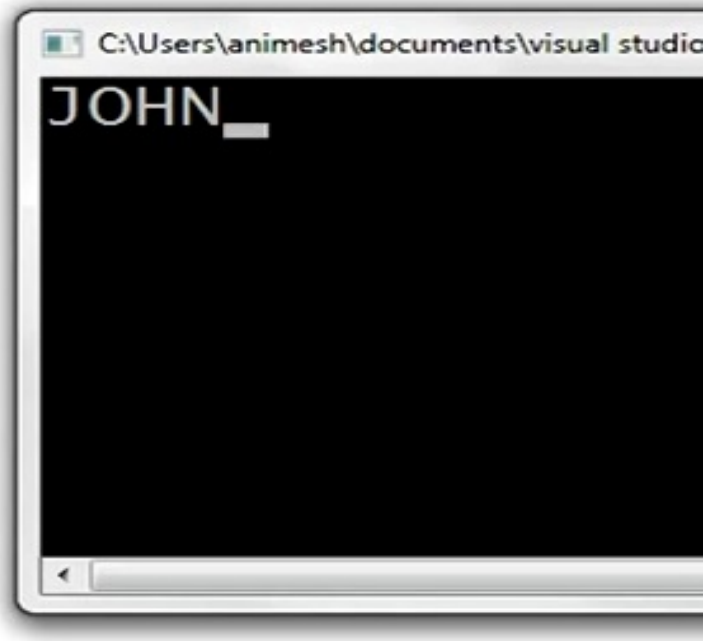
```
c[0] = 'J'; c[1] = 'O'; c[2] = 'H'; c[3] = 'N';
c[4] = '\0';
```

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

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



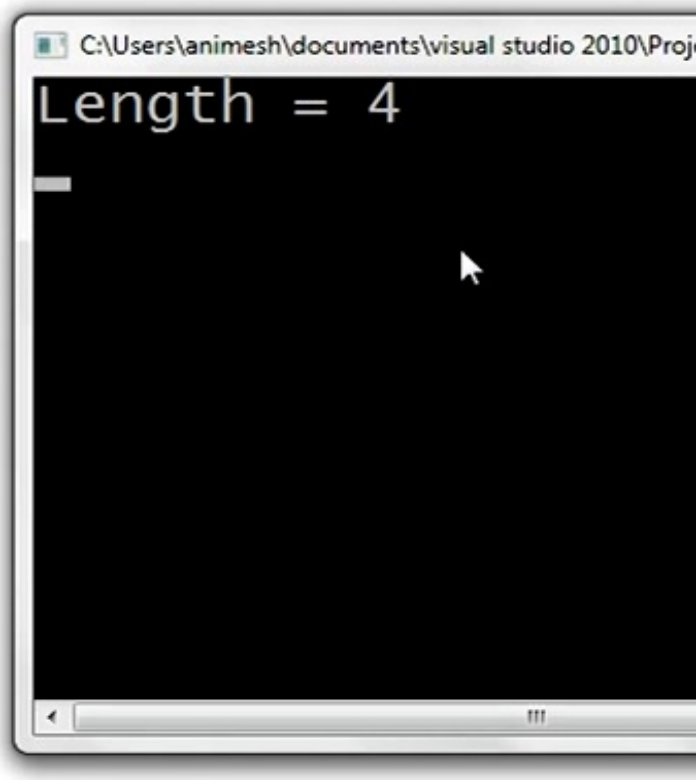
```c
//character arrays and pointers
#include<stdio.h>
int main()
{
    char C[5];
    C[0] = 'J';
    C[1] = 'O';
    C[2] = 'H';
    C[3] = 'N';
    C[4] = '\0';
    printf("%s",C);
}
```



```c
//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);
}
```

```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[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[20] = "JOHN";
    int len = strlen(C);
    printf("Length = %d\n",len);
}
```

```c
//character arrays and pointers
#include<stdio.h>
#include<string.h>
int main()
{
    char C[20];
    C = "JOHN";
    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[] = "JOHN";
    printf("Size in bytes = %d\n",sizeof(C));
    int len = strlen(C);
    printf("Length = %d\n",len);
}
```

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

```
Size in bytes = 5
Length = 4
```

```c
//character arrays and pointers
#include<stdio.h>
#include<string.h>
int main()
{
    char C[4] = "JOHN";
    printf("Size
    int len = strlen(C);
    printf("Length = %d\n",len);
}
```

Error: a value of type "const char [5]" cannot be used to initialize an entity of type "char [4]"

```c
//character arrays and pointers
#include<stdio.h>
#include<string.h>
int main()
{
    char C[5] = "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);
}
```

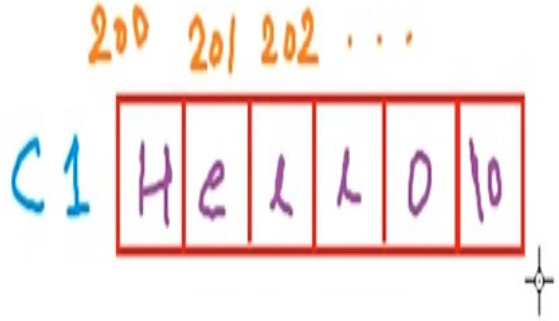
Character arrays and pointers

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

Character arrays and pointers

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

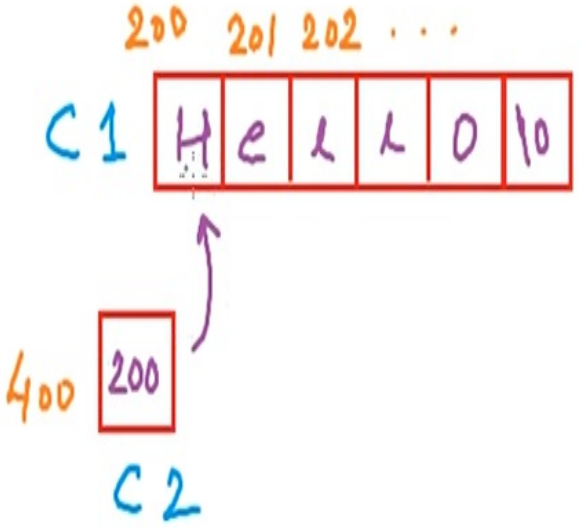
```
char c1[6] = "Hello";
```



Character arrays and pointers

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

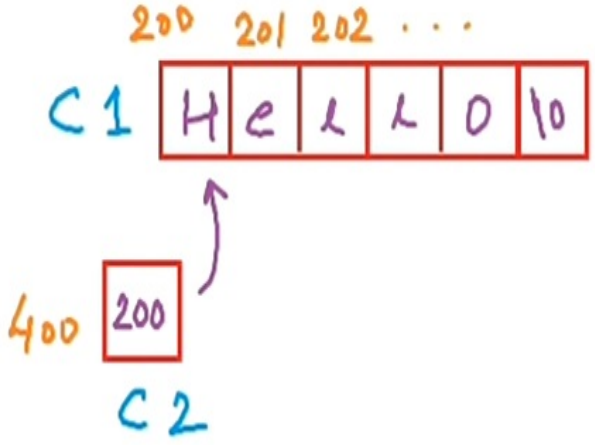
```
char c1[6] = "Hello";

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)

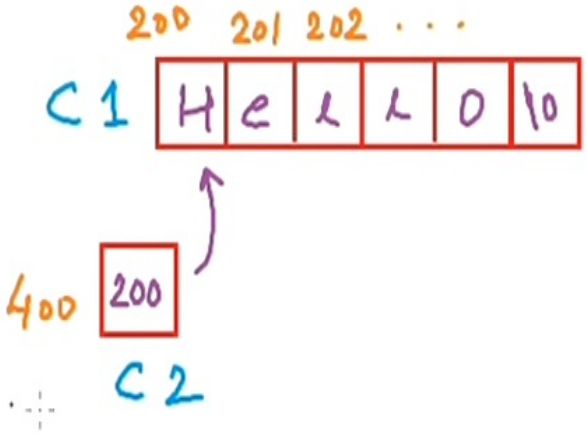


2) Arrays and pointers are different types that are used in similar manner

```
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)

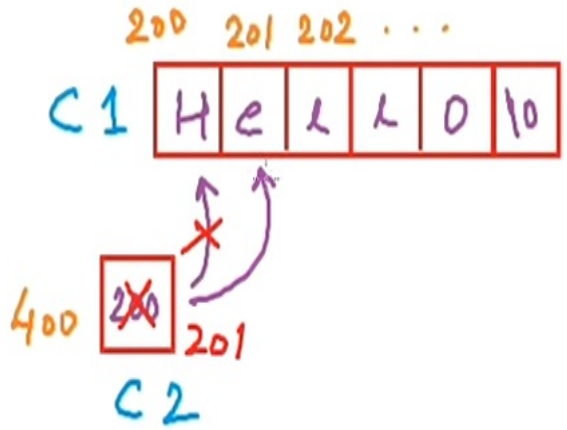


2) Arrays and pointers are different types that are used in similar manner

```
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; ✗

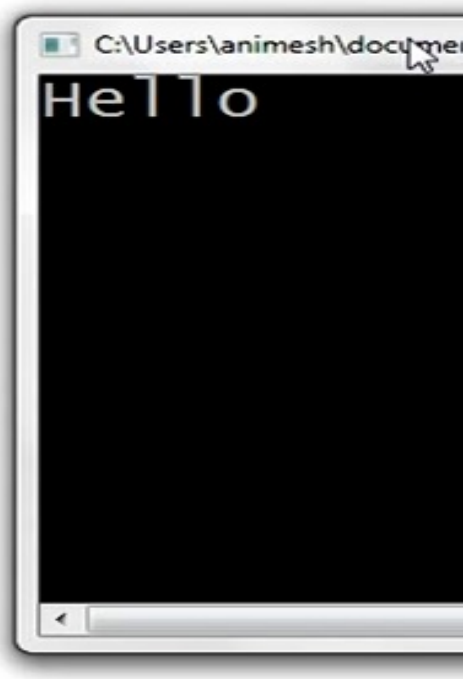
c2++;

Character arrays and pointers

3) Arrays are always passed to function by reference

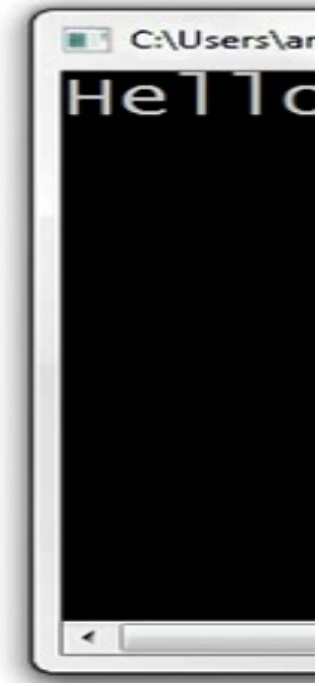
```c
//character arrays and pointers
#include<stdio.h>
void print(char* C)
{
    int i = 0;
    while(C[i] != '\0')
    {
        printf("%c",C[i]);
        i++;
    }
    printf("\n");
}
int main()
{
    char C[20] = "Hello";
    print(C);
}
```

```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)
{
    int i = 0;
    while(*(C+i)!= '\0')
    {
        printf("%c",C[i]);
        i++;
    }
    printf("\n");
}
int main()
{
    char C[20] = "Hello";
    print(C);
}
```

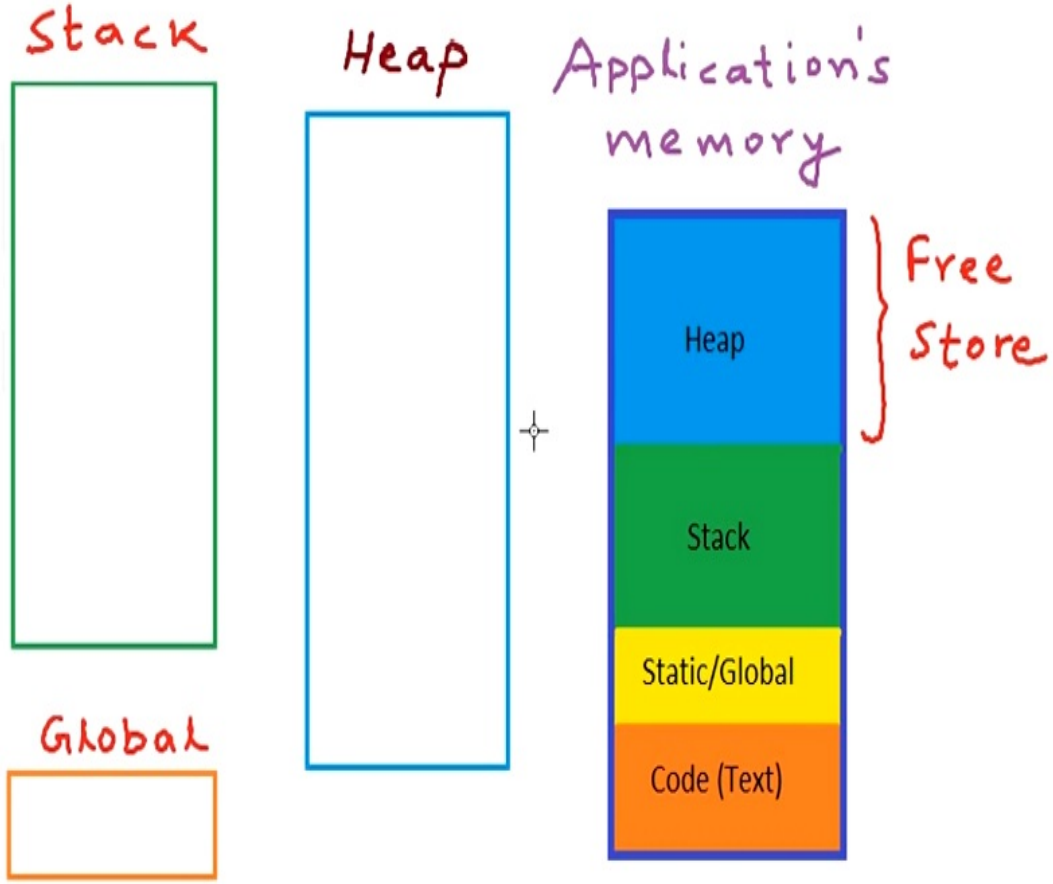
```c
//character arrays and pointers
#include<stdio.h>
void print(char* C)
{
    while(*C != '\0')
    {
        printf("%c",*C);
        C++;
    }
    printf("\n");
}
int main()
{
    char C[20] = "Hello";
    print(C);
}
```



Outline

Character arrays and pointers - part II

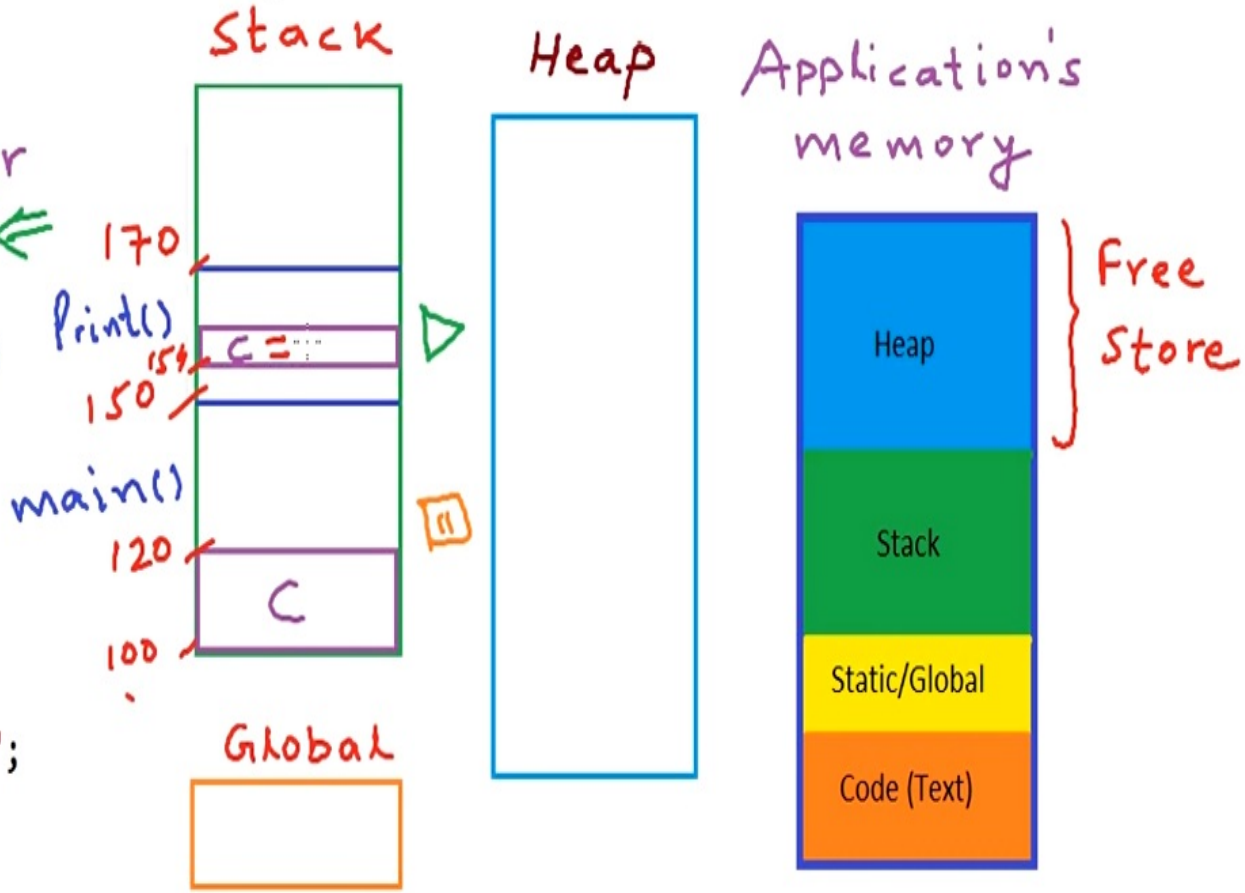
```c
#include<stdio.h>
#include<string.h>
void print(char *C)
{
    while(*C != '\0')
    {
        printf("%c",*C);
        C++;
    }
    printf("\n");
}
int main()
{
    char C[20] = "Hello";
    print(C);
}
```



Character arrays and pointers - part II

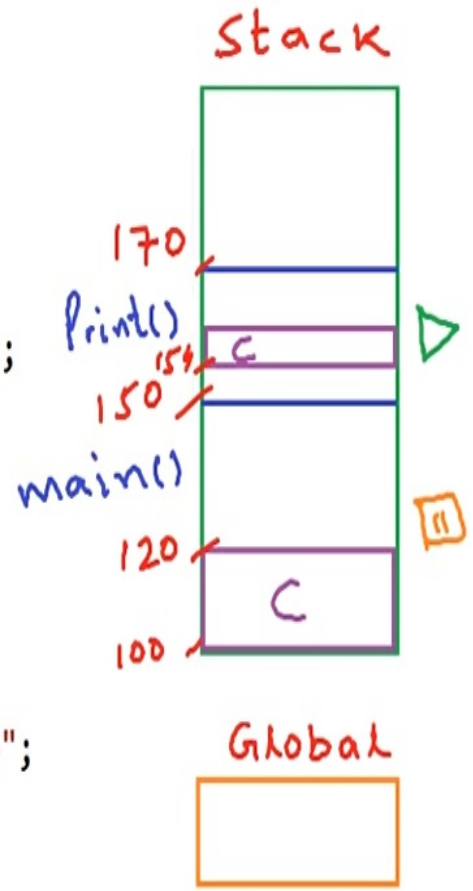
```
#include<stdio.h>
#include<string.h>
void print(char *C)
{
    while(*C != '\0')
    {
        printf("%c",*C);
        C++;
    }
    printf("\n");
}
int main()
{
    char C[20] = "Hello";
    print(C);
}
```

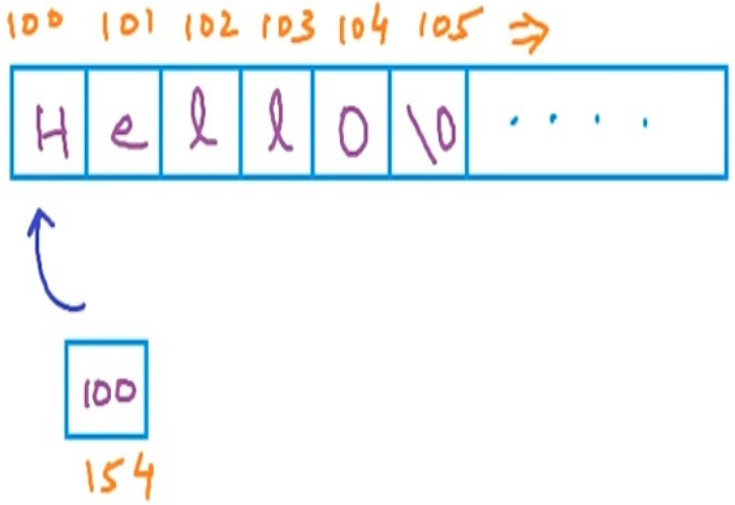
a, ptr



Character arrays and pointers - part II

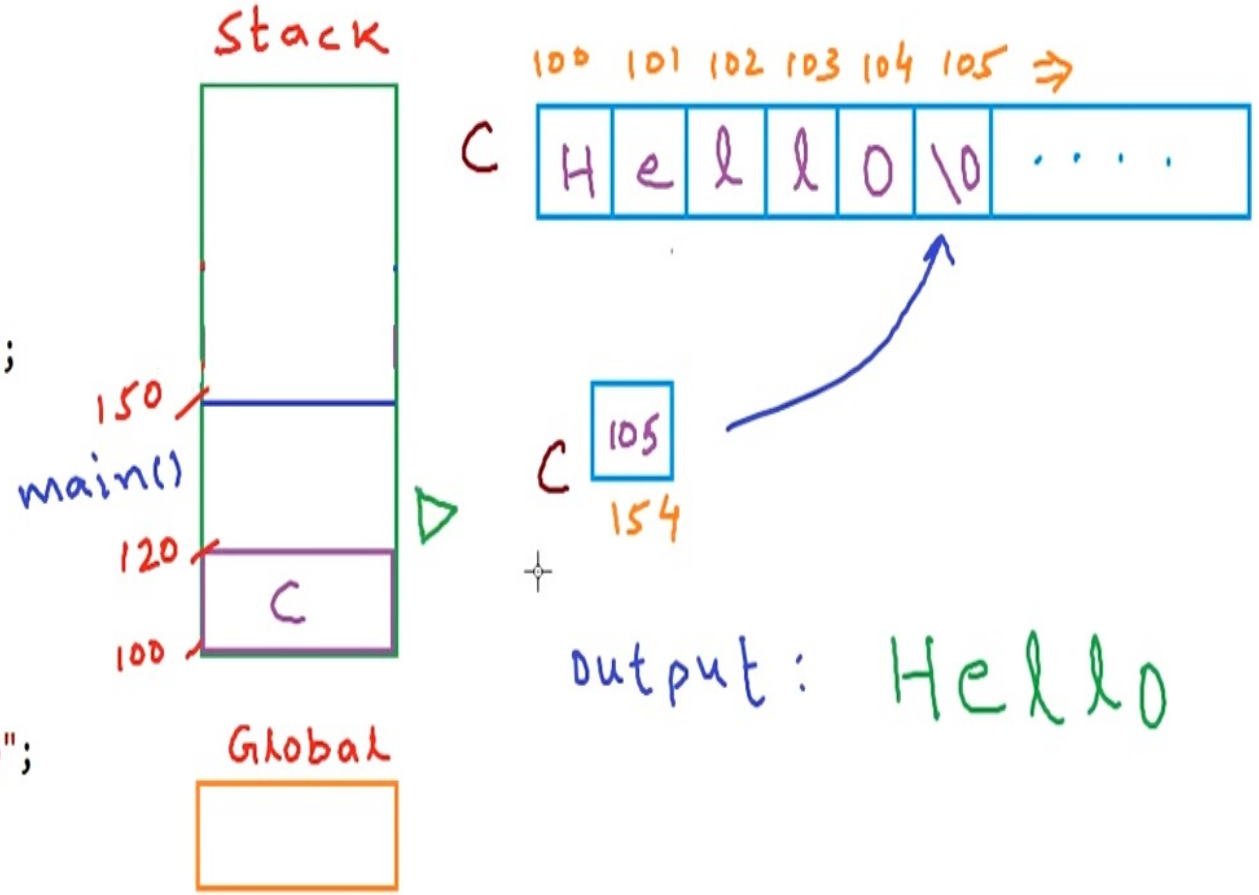
```
#include<stdio.h>
#include<string.h>
void print(char *C)
{
    while(*C != '\0')
    {
        printf("%c",*C);
        C++;
    }
    printf("\n");
}
int main()
{
    char C[20] = "Hello";
    print(C);
}
```





Character arrays and pointers - part II

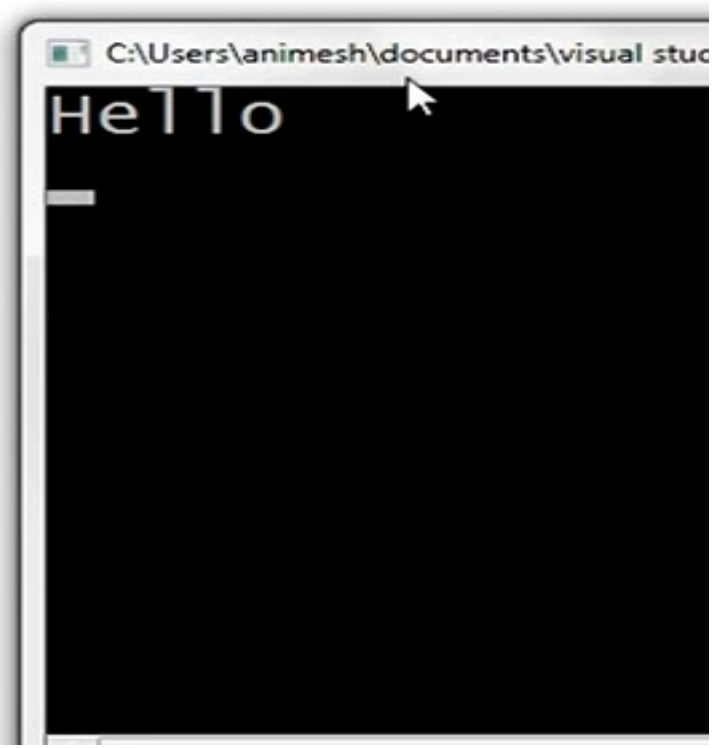
```
#include<stdio.h>
#include<string.h>
void print(char *C)
{
    while(*C != '\0')
    {
        printf("%c",*C);
        C++;
    }
    printf("\n");
}
int main()
{
    char C[20] = "Hello";
    print(C);
}
```



Output: Hello

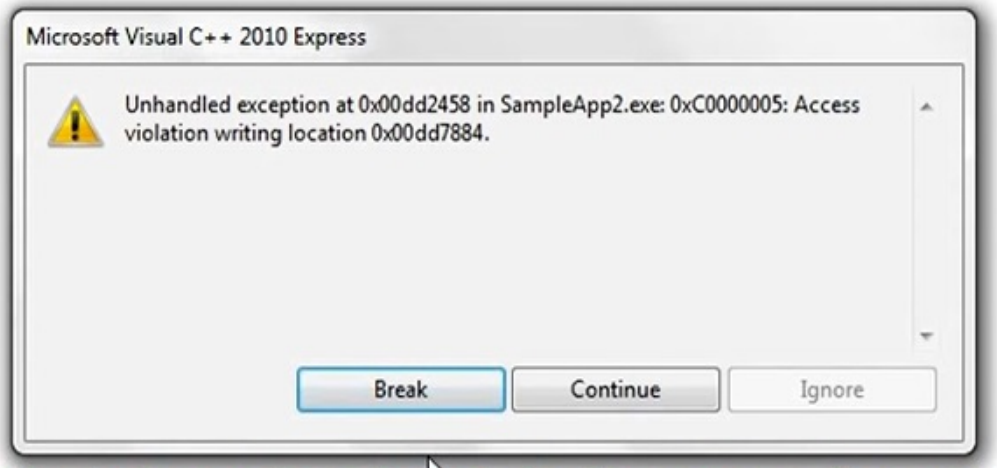
```c
//character arrays and pointers
#include<stdio.h>
#include<string.h>
void print(char *C)
{
    while(*C != '\0')
    {
        printf("%c",*C);
        C++;
    }
    printf("\n");
}
int main()
{
    char *C = "Hello";
    print(C);
}
```

```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 = "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
    printf("Hello World");
    print(C);
}
```

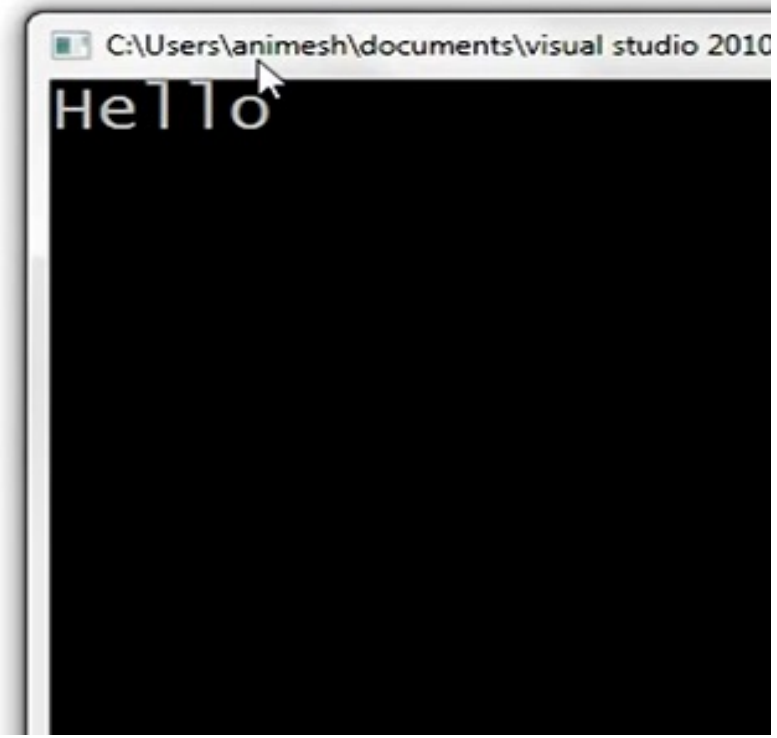
```c
//character arrays and pointers
#include<stdio.h>
#include<string.h>
void print(char *C)
{
    while(*C != '\0')
    {
        printf("%c",*C);
        C++;
    }
    printf("\n");
}
int main()
{
    //char C[20] = "Hello"; // string gets stored in the space for array
    char *C = "Hello"; // string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

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



Outline

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
```

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

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

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]

```
p = A;  ✓
A = p;  ✗
```

Pointers and multi-dimensional arrays

```
int A[5]
```

A[0], A[1] → int

⋮

200	204	208	212	216
2	4	6	8	10
A[0]	A[1]	A[2]	A[3]	A[4]

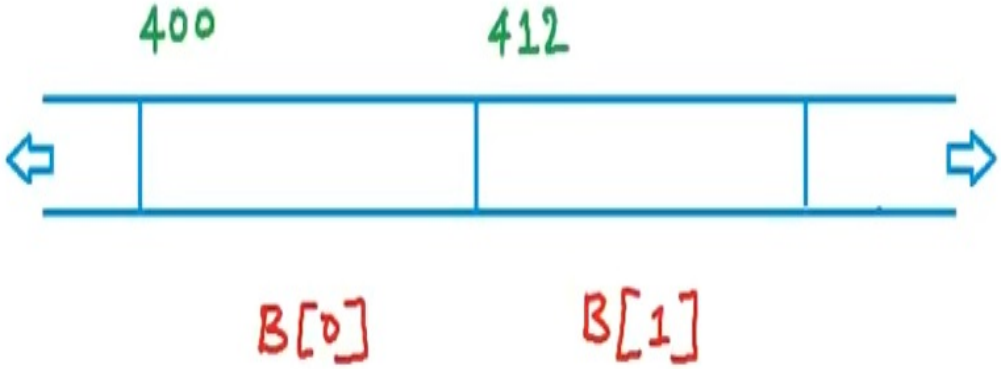
```
int B[2][3]
```

B[0], B[1] → 1-D arrays of 3 integers

Pointers and multi-dimensional arrays

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

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



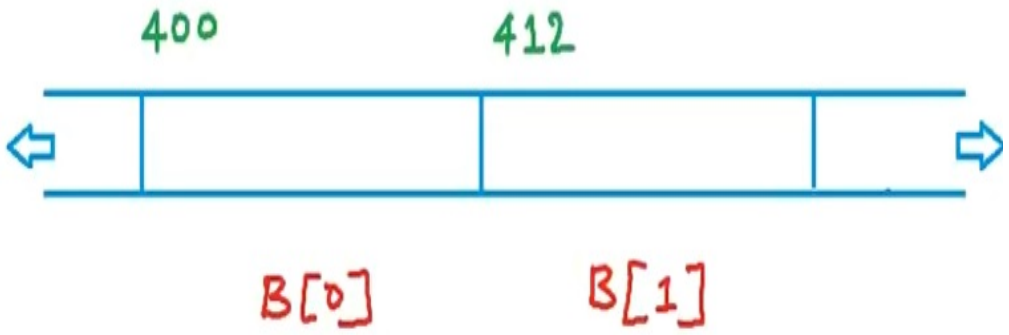

```
int *p = B;   X
```

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

Pointers and multi-dimensional arrays

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

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




```
int *p = B;   ✗
```

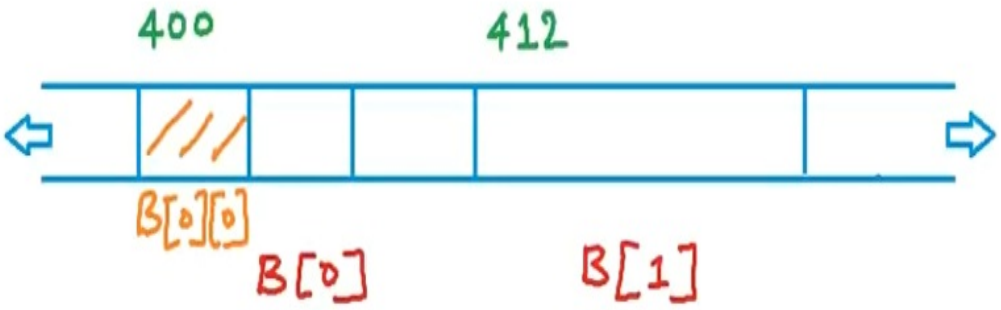
↓ will return a pointer to 1-D array of 3 integers

```
int (*p)[3] = B;   ✓
```

Pointers and multi-dimensional arrays

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

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




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

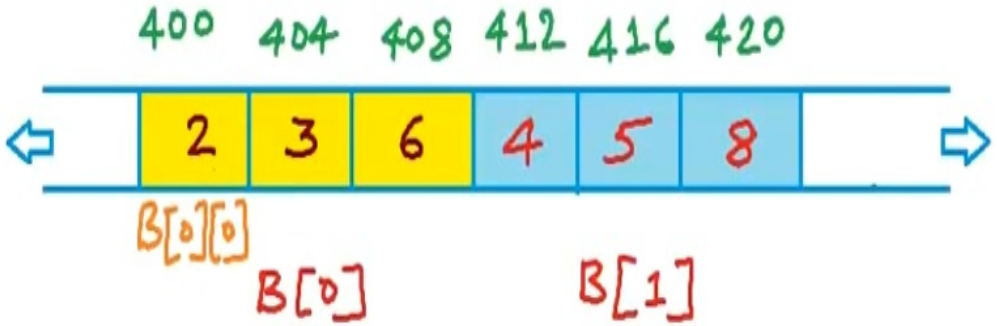
↓ will return a pointer to 1-D array of 3 integers

```
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

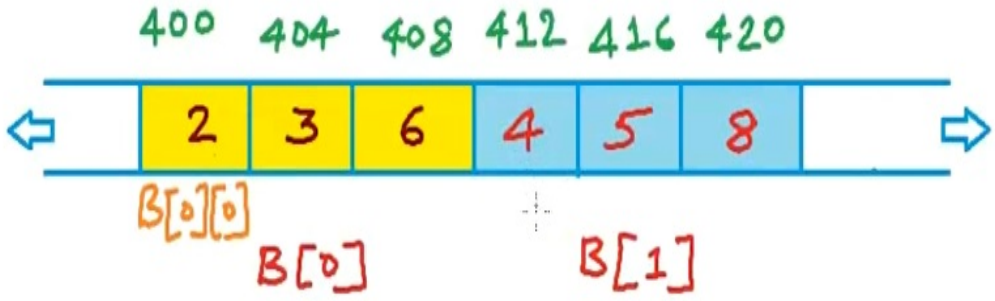



```
int (*p)[3] = B;

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

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

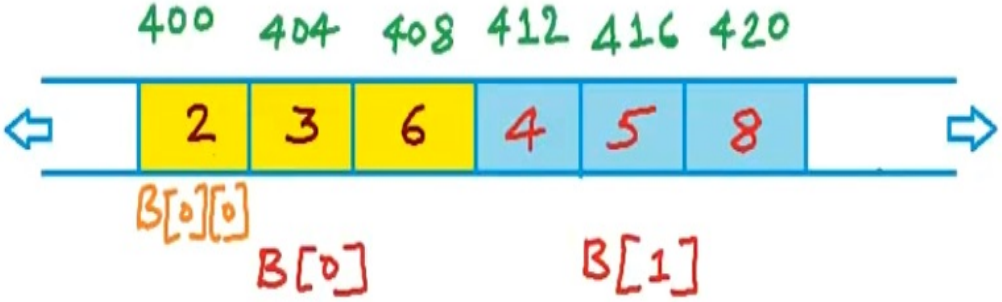
Print *B or B[0] or &B[0][0] // 400

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

Pointers and multi-dimensional arrays

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

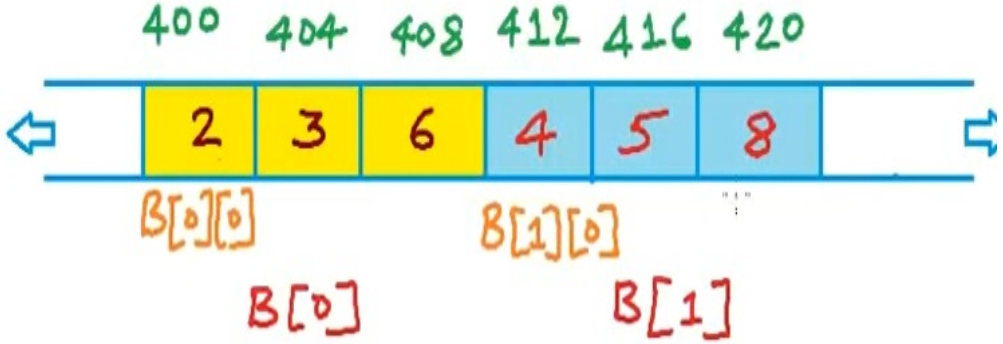
B[0]
B[1] } → 1-D arrays of 3 integers




```
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]
```

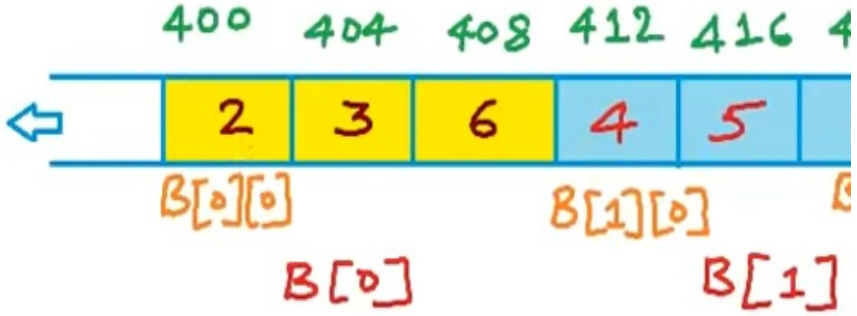
B[0]
B[1] } → 1-D arrays of 3 integers




```
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 or B[1]+2 or &B[1][2]   // 420
```

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

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



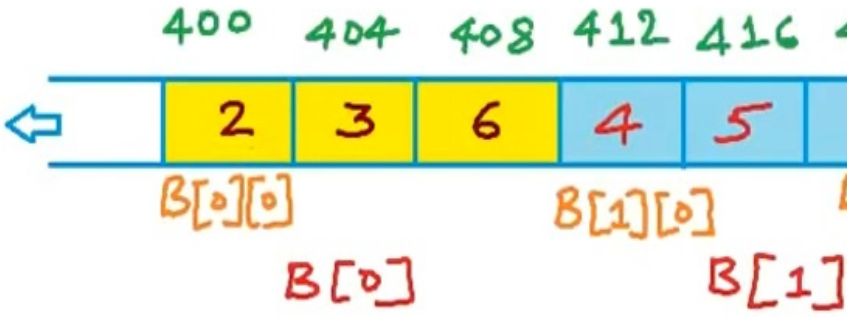

```
int (*p)[3] = B;
Print B or &B[0]          // 400
Print *B or B[0] or &B[0][0]   // 400
Print B+1 or &B[1]        // 412
Print *(B+1) or B[1] or &B[1][0]   // 412
Print *(B+1)+2 or B[1]+2 or &B[1][2]   // 420
Print *(*B+1)
```

B[0]

B → int (*)[3]
B[0] → 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)
```

$\underbrace{*B+1}_{\&B[0][1]}$

B → int (*)[3]
B[0] → 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+1) → B[0][1]

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

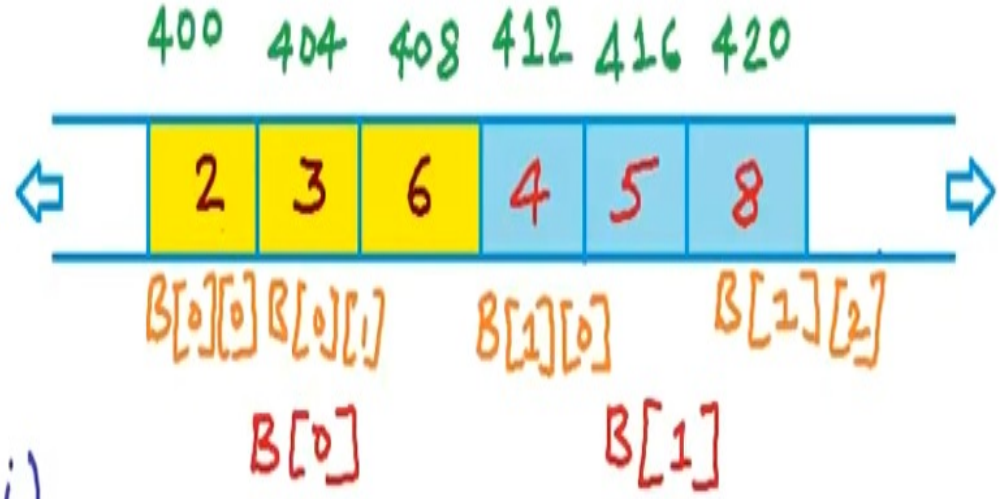
B[0]
B[1] } → 1-D arrays of 3 integers


```
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+1) → B[0][1]

int B[2][3]

For 2-D array

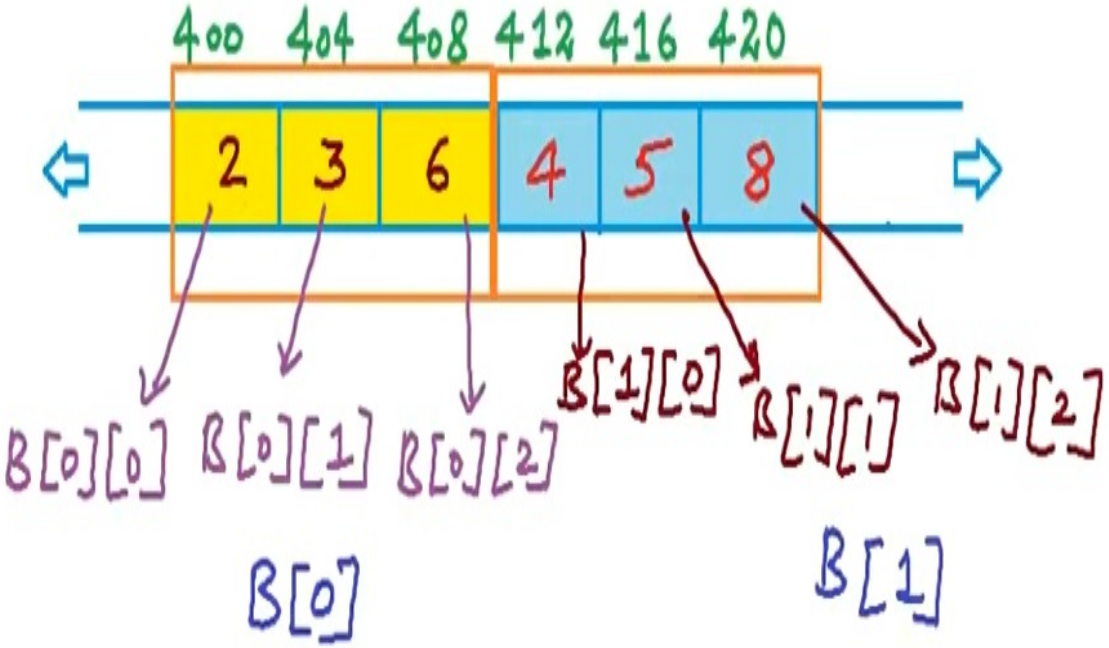


$$B[i][j] = *(B[i]+j)$$

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

Pointers and multi-dimensional arrays

int B[2][3]

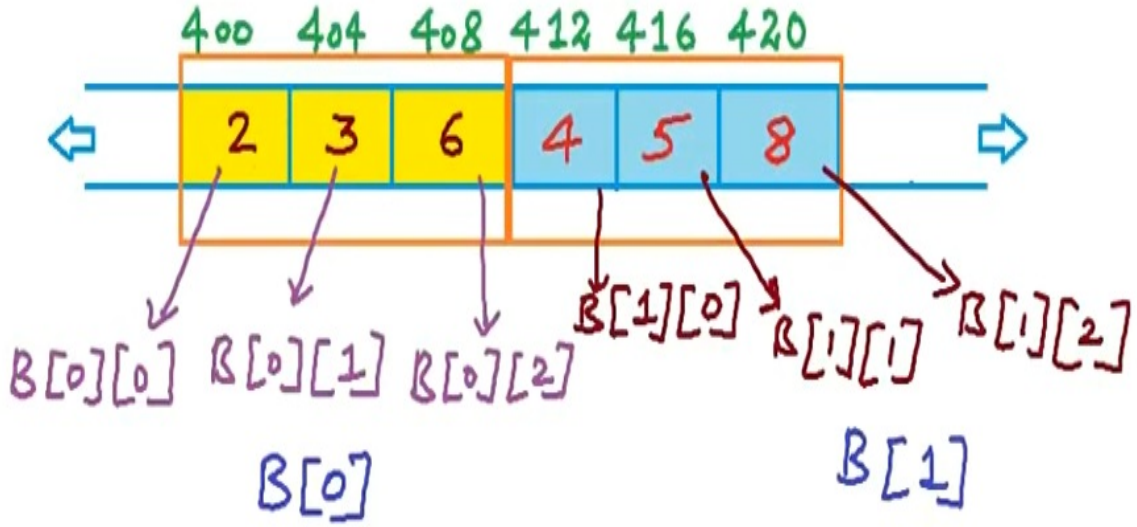


Pointers and multi-dimensional arrays

```
int B[2][3]

int (*P)[3] = B; ✓
```

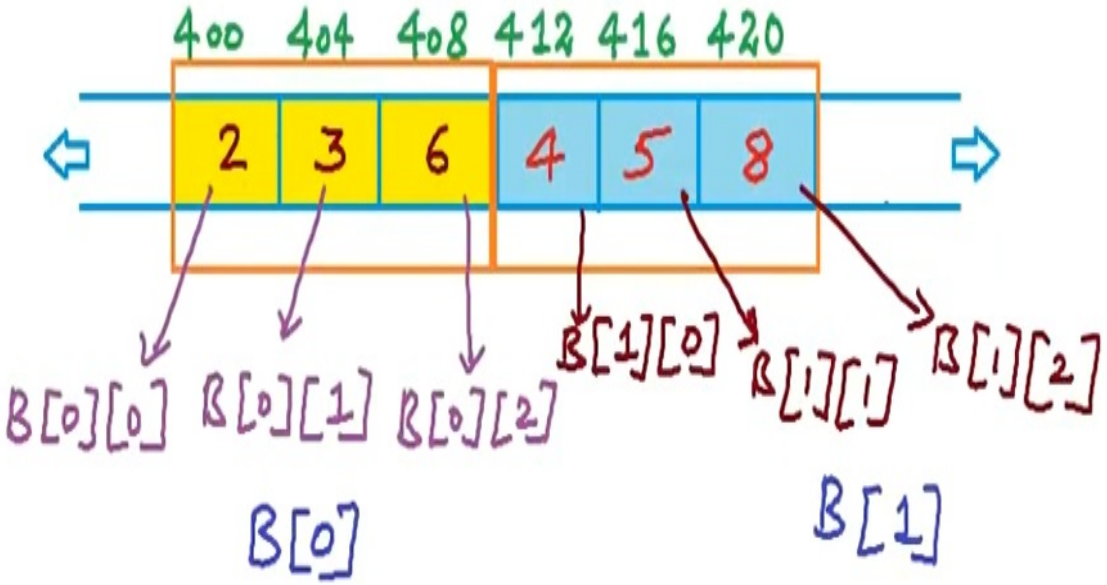
declaring pointer to 1-D array of 3 integers

```
int *P = B; ✗
```



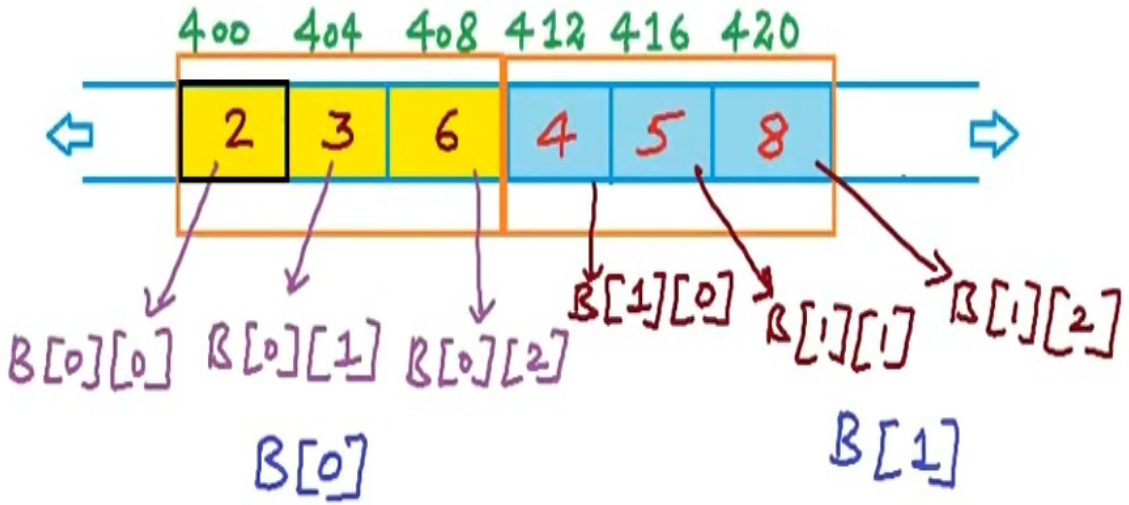
Pointers and multi-dimensional arrays

```
int B[2][3]

int (*P)[3] = B; ✓

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



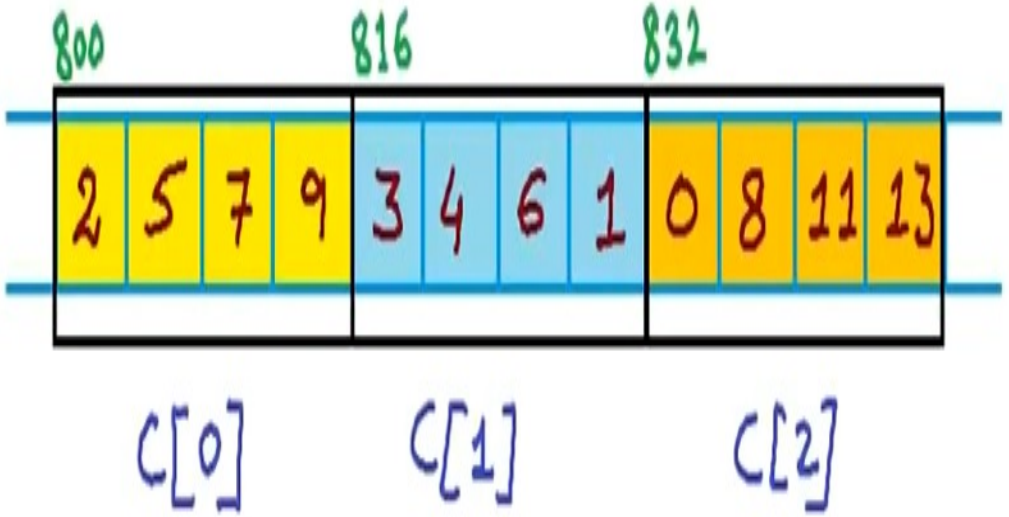
Pointers and multi-dimensional arrays

```
int B[2][3]

int (*P)[3] = B; ✓

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



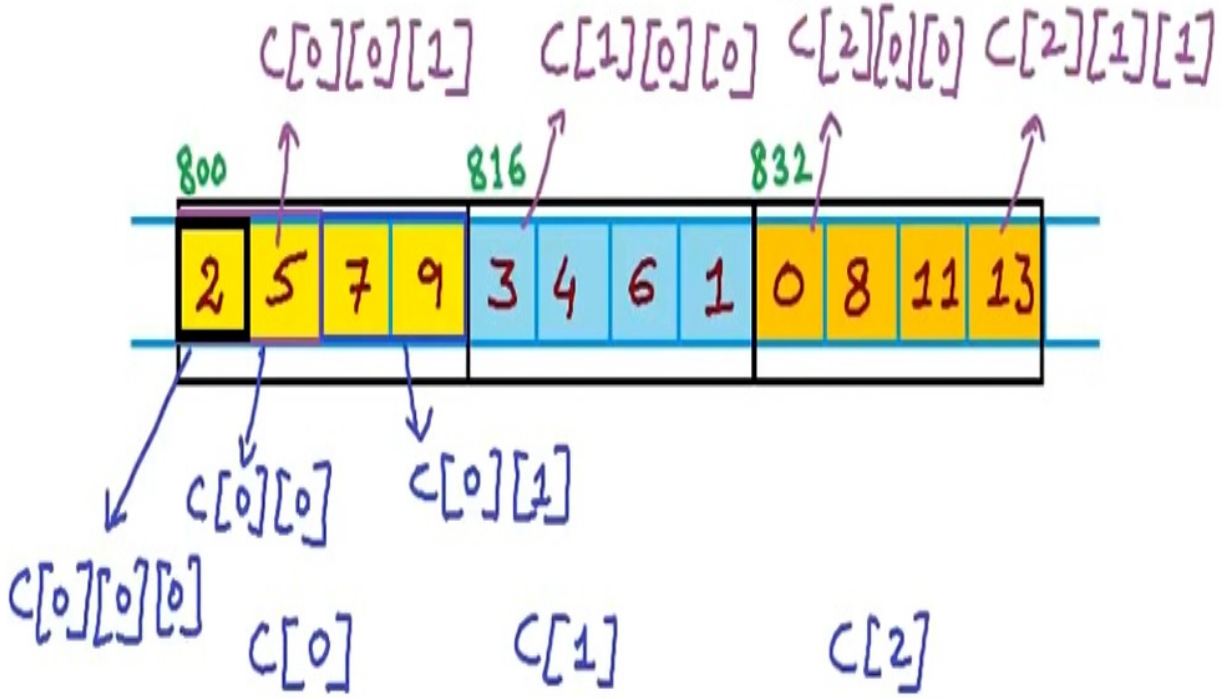
Pointers and multi-dimensional arrays

int c[3][2][2]



Pointers and multi-dimensional arrays

int c[3][2][2]



Pointers and multi-dimensional arrays

int C[3][2][2]

int (*P)[2][2] = C; ✓

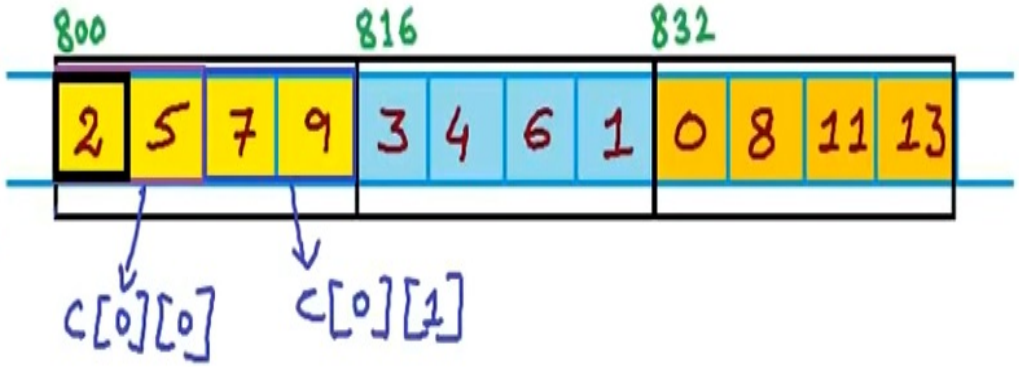
Print C // 800

↳ int (*)[2][2]

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

↓

int (*)[2]



Pointers and multi-dimensional arrays

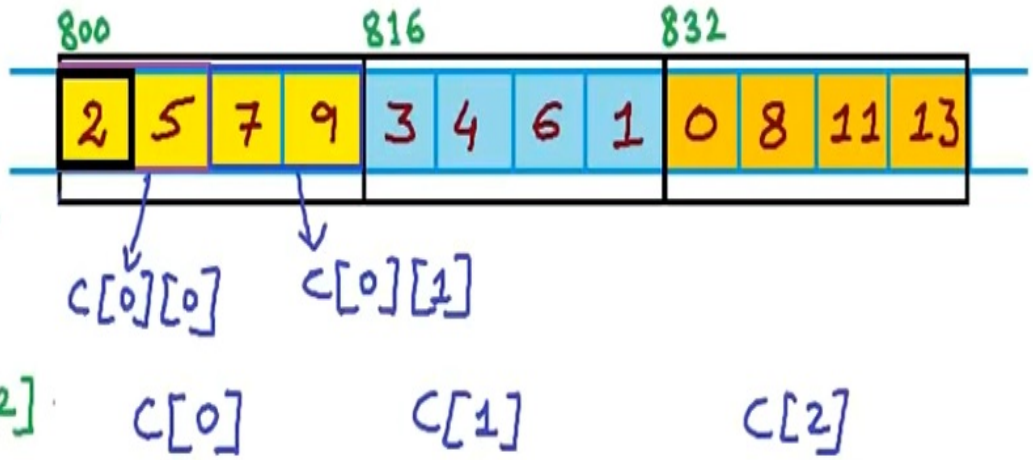
int C[3][2][2]

int (*P)[2][2] = C; ✓

Print C //800

↳ int (*)[2][2]

Print *C or C[0] or &C[0][0]



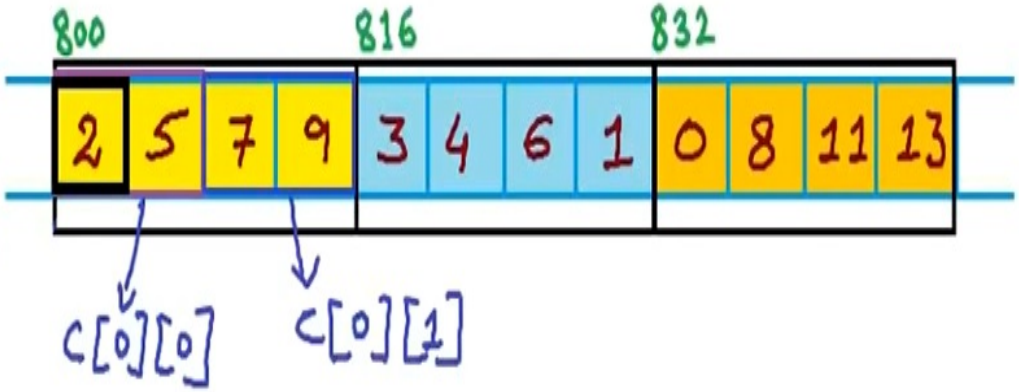
$$C[i][j][k] = *(C[i][j]+k) = *(*(C[i]+j)+k)$$

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

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]

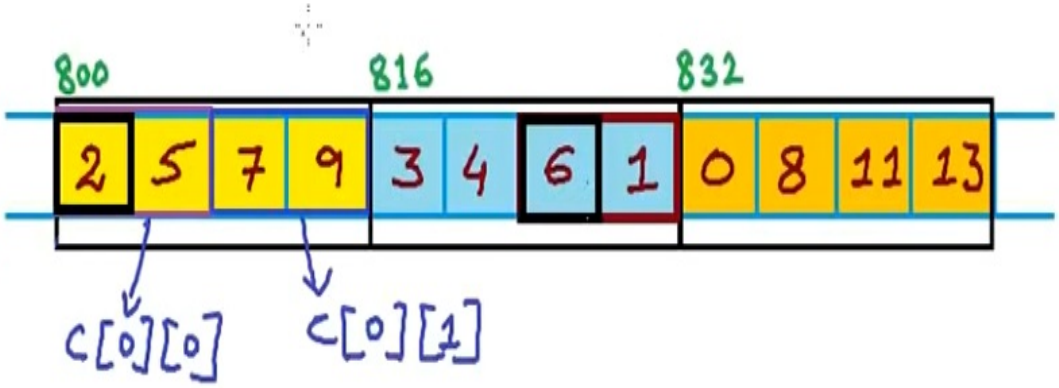
Print *C or C[0] or &C[0][0]

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

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

```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 4192172 4192172 4192172
```

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

```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\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) // Argument: 1-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}};
    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);
}
```

myc

```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(B);
}
```

```
// 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]
    int X[2][4];
    Func(X); // X returns int (*)[4]
```

```
// Pointers and multi- dimensional arrays
#include<stdio.h>
void Func(int (*A)[2][2]) // Argument: 2-D array of integer
{
}
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]
    int X[5][3];
    Func(C);
}
```