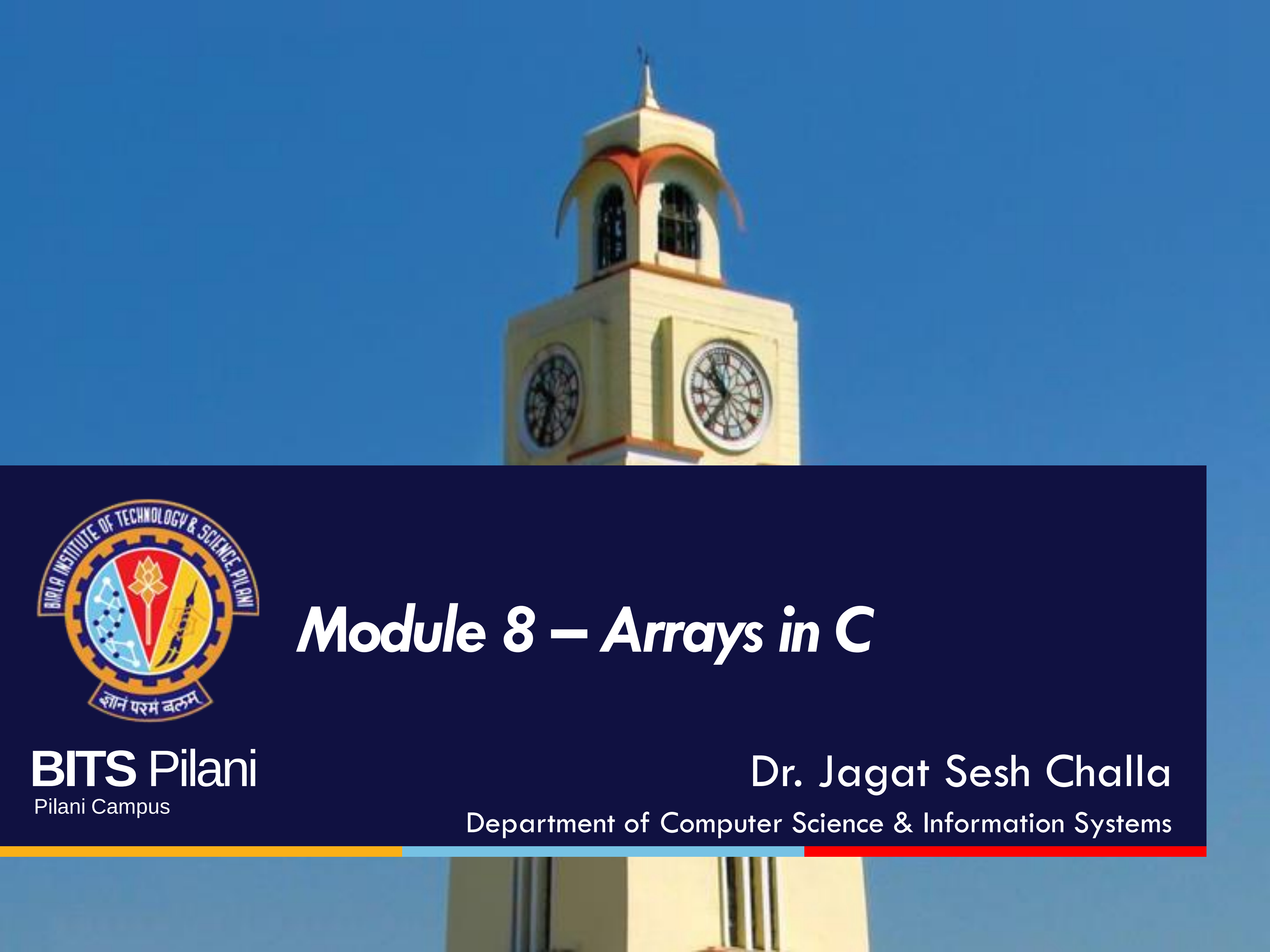


BITS Pilani
Pilani Campus

Module 8 – Arrays in C

Dr. Jagat Sesh Challa

Department of Computer Science & Information Systems

Module Overview

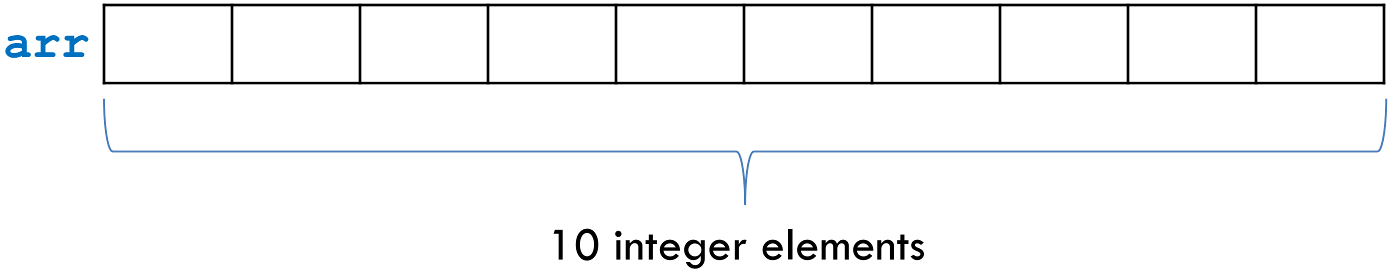
- Introduction to Arrays in C
- Arrays in Memory
- A few examples
- Passing arrays to functions
- Searching in an Array
- Selection Sort
- Binary Search
- Insert/Delete in an Array
- Character Arrays
- Multi-dimensional Arrays
- Matrix Addition and Multiplication

BITS Pilani
Pilani Campus

Intro to Arrays in C

What are Arrays?

- Array – fixed size sequenced collection of elements of the same data type.
- It is a derived data type.
- Example:
  - `int arr[10]` declares an array of 10 elements each of integer type



arr

10 integer elements

- Single name is used to represent a collection of items.
  - `arr` *represents 10 integer elements in the above array*
- Arrays are useful in processing multiple data items having a common characteristic
  - E.g.: set of numerical data, list of student names, etc.

Importance of Arrays

- Easier storage, access, and data management
- Easier to search
- Easier to organize data elements
- Useful to perform matrix operations
- Useful in databases
- Useful to implement other data structures

Defining Arrays

Syntax:

```
<Storage-class> <data-type> <array_name>[<size>]
```

Note: Storage-class is optional

Examples:

```
int count[100];
char name[25];
float cgpa[50];
```

Good practice:

```
#define SIZE 100
int count[SIZE];
```

Initializing arrays

Few valid ways of initializing arrays in C

```c
int intArray[6] = {1, 2, 3, 4, 5, 6};
```

1	2	3	4	5	6

```c
float floatArray[10] = {1.387, 5.45, 20.01};
```

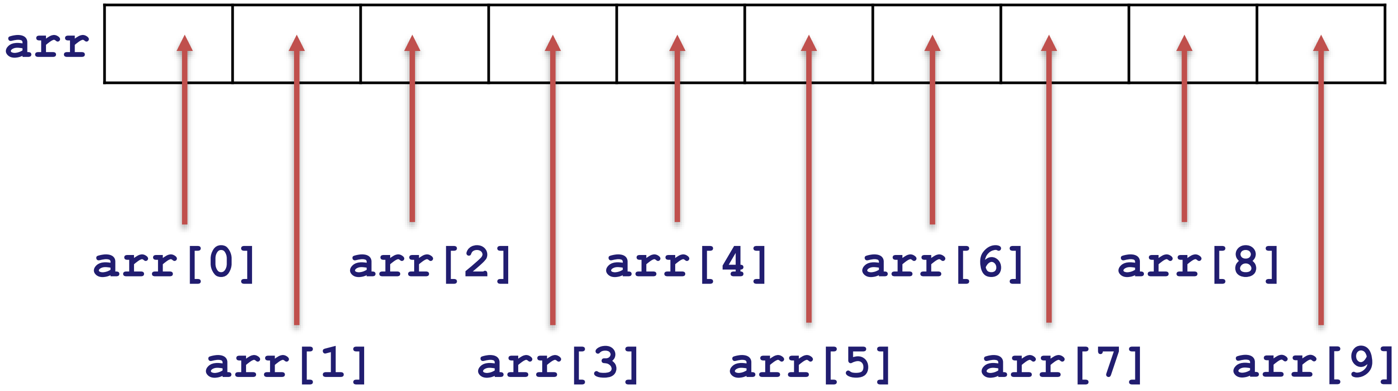
1.387	5.45	20.01	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```c
double fractions[] = {3.141592654,1.570796327,0.785398163};
```

3.141592654	1.570796327	0.785398163

Accessing elements of an array

```
int arr[10];
```



For an n-element array:

- The first element is accessed by `arr[0]`
- The second element is accessed by `arr[1]`
- The last element is accessed by `arr[n-1]`

Initializing Arrays During Program Execution

```
#include <stdio.h>
int main()
{
        int nums[10]; int i;
        for(i=0; i<10; i++)
        {
                /* Reading an array of elements */
                scanf(“%d”,&nums[i]);
        }
}
```

Example

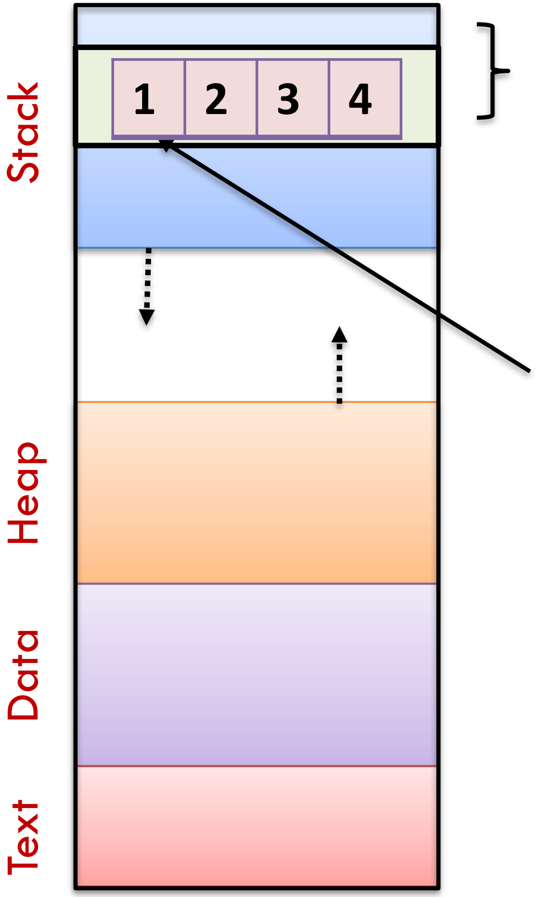
A program that takes 10 elements from the user and stores them in an array and then computes their sum.

```
#include <stdio.h>
int main(){
      int nums[10]; int sum=0; int i;
      for(i=0; i<10; i++){
            scanf(“%d”,&nums[i]);
      }
      for(i=0;i<10;i++){
            sum = sum + nums[i];
      }
      printf(“The sum is: %d”, sum);
}
```

BITS Pilani
Pilani Campus

Arrays in Memory

Arrays in memory



Frame allocated to *main()* in the stack

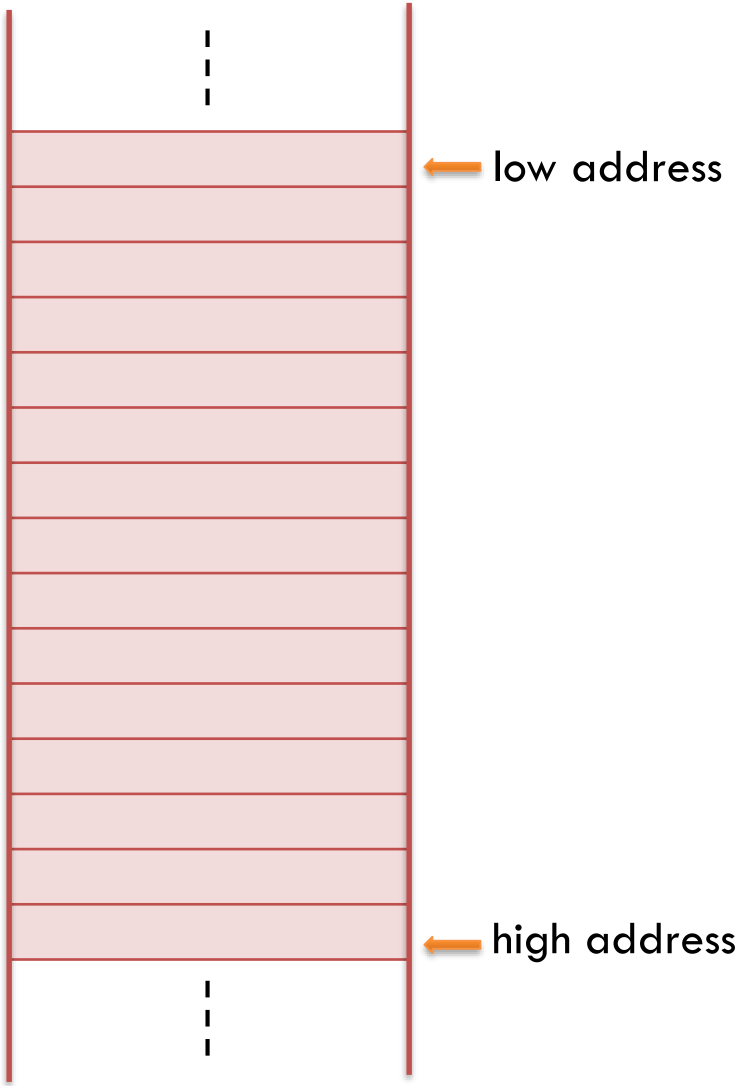
Array arr residing in the frame allocated to *main()*

- Elements of `arr` are stored in contiguous memory locations
- `arr` references the first element in the array.
- In other words, the variable `arr` contains the *address of the first location/element* of the 4 elements array that we have just defined.
- ***What is this address?***
- ***We will see***

Consider this program:

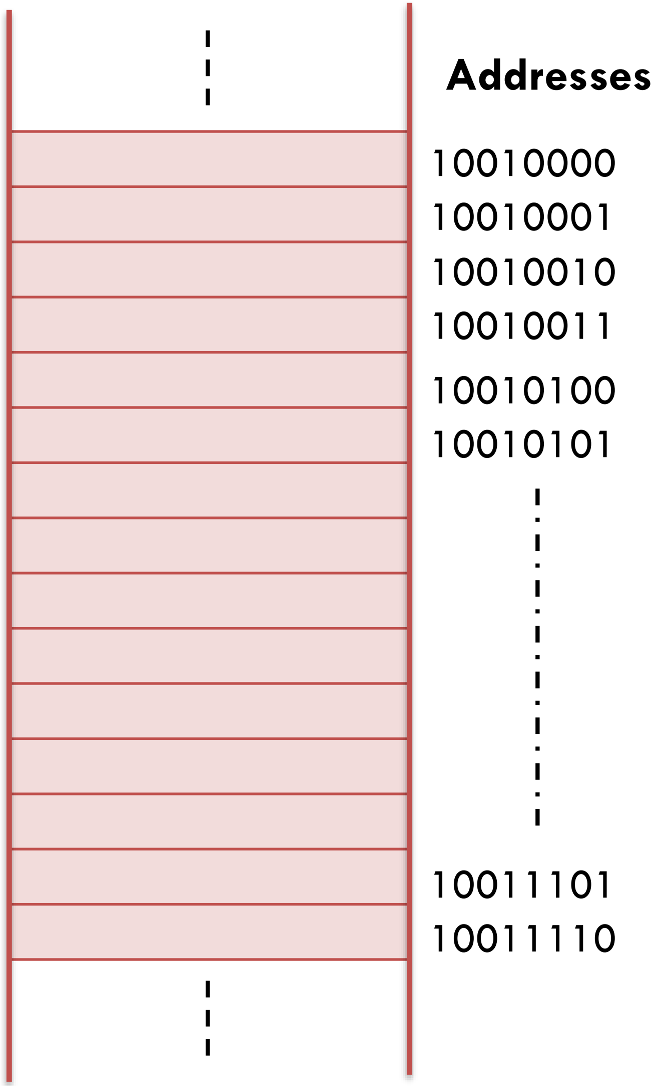
```
int main(){
        int arr[4] = {1, 2, 3, 4};
        …
        return 0;
}
```

Addresses in Main Memory



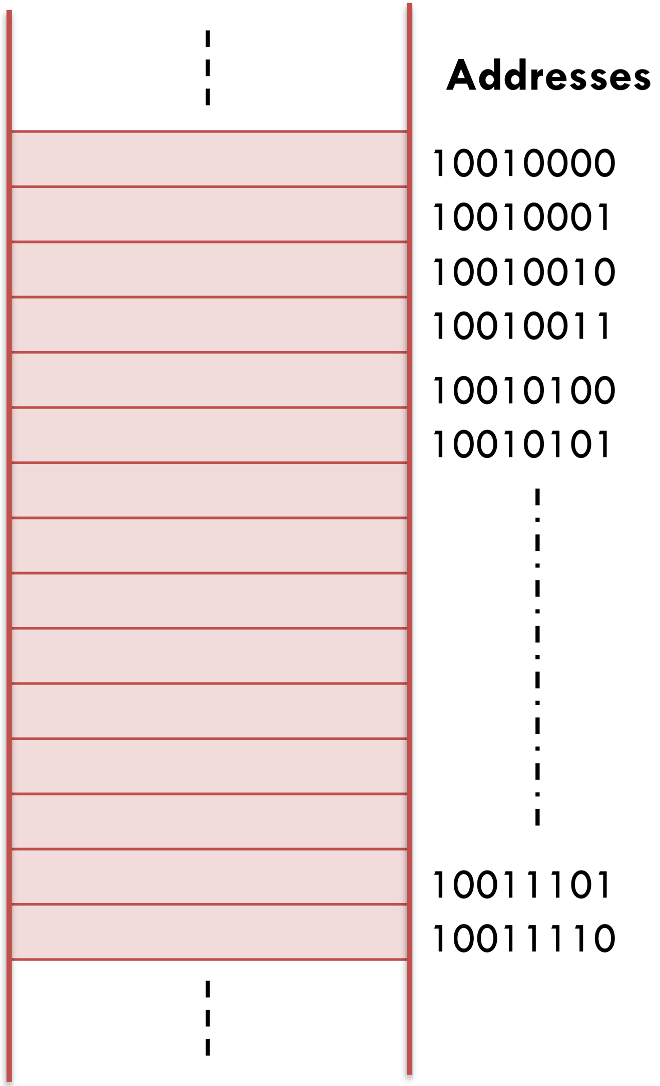
- Remember how our memory is actually organized
- It is like a table of contiguous memory locations, each having an address
  - Each address is represented by a string of bits
    - *typically 32 or 64 bits in modern computers*

Addresses in Main Memory



- Note: our memory is byte addressable.
  - every byte in the memory has an address,
  - In other words, size of each memory location is 1 byte.
- For Simplicity let us consider each address in the main memory to be represented by 8 bits.
- The addresses of the memory locations are depicted in the figure
- *Note:The addresses of two contiguous locations differ by 1 (in binary)*

How is an array stored in main memory?



Addresses

10010000
10010001
10010010
10010011
10010100
10010101

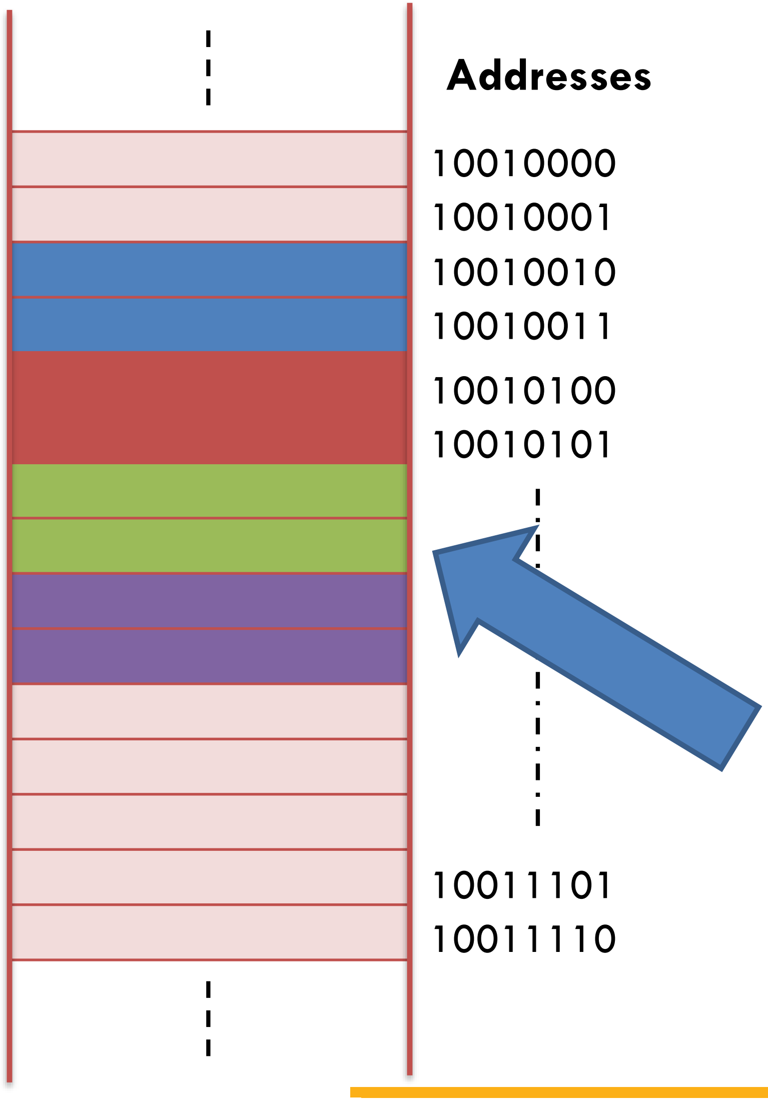
10011101
10011110

Consider this program:

```
int main(){
        int arr[4] = {1, 2, 3, 4};
        …
        return 0;
}
```

- Say each integer variable takes 2 bytes of memory
- To store an integer array of size 4, we need 4*2 = 8 bytes of memory or 8 contiguous locations in memory (given each location is of size 1 byte)

How is an array stored in main memory?



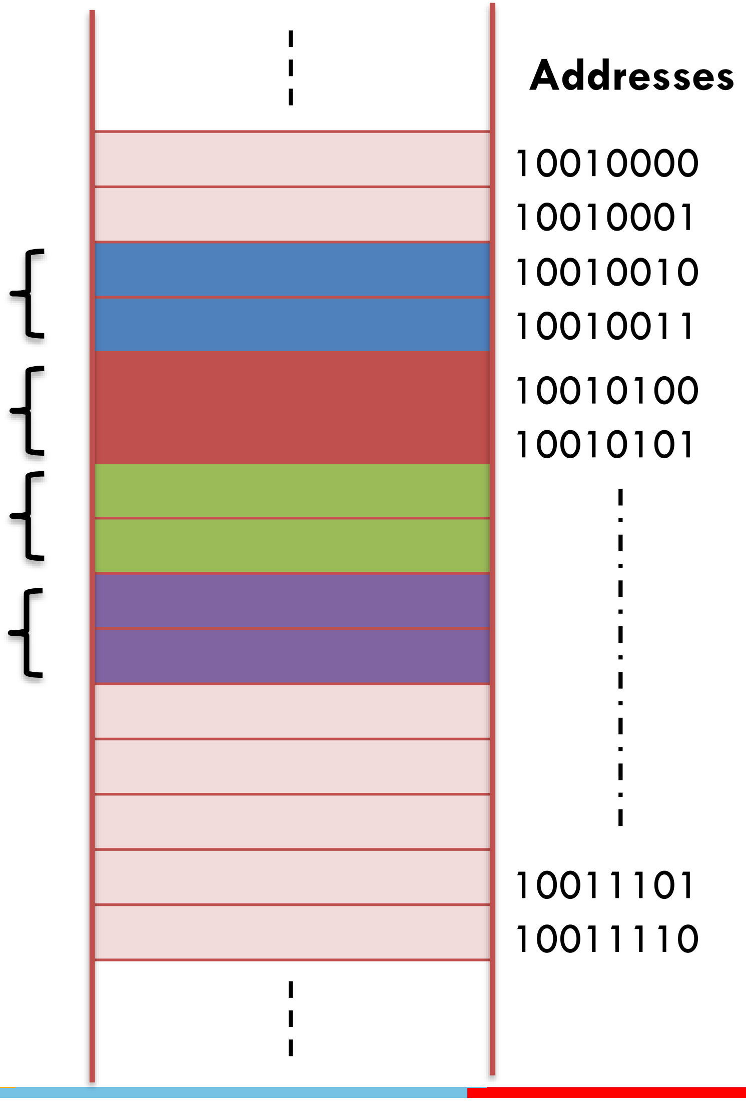
Consider this program:

```
int main(){
        int arr[4] = {1, 2, 3, 4};
        …
        return 0;
}
```

- Say each integer variable takes 2 bytes of memory
- To store an integer array of size 4, we need 4*2 = 8 bytes of memory or 8 contiguous locations in memory (given each location is of size 1 byte)

How is an array stored in main memory?

```
int arr[4] = {1, 2, 3, 4};
```



Stores first element of arr, accessed by **arr[0]**

Stores second element of arr, accessed by **arr[1]**

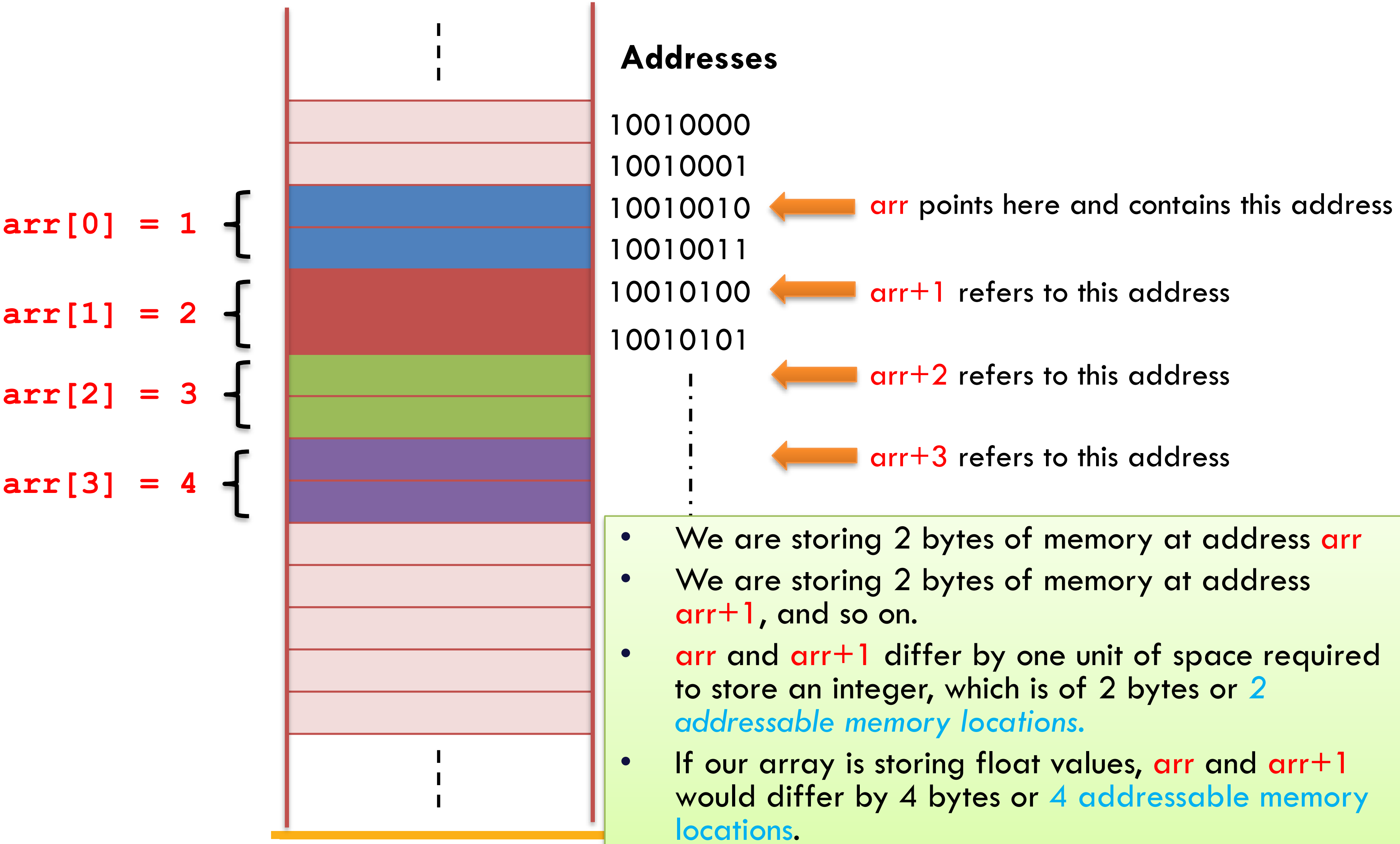
Stores third element of arr, accessed by **arr[2]**

Stores fourth element of arr, accessed by **arr[3]**

Addresses

10010000
10010001
10010010
10010011
10010100
10010101
10011101
10011110

How is an array stored in main memory?



arr[0] = 1

arr[1] = 2

arr[2] = 3

arr[3] = 4

Addresses

10010000

10010001

10010010 arr points here and contains this address

10010011

10010100 arr+1 refers to this address

10010101

arr+2 refers to this address

arr+3 refers to this address

- We are storing 2 bytes of memory at address arr
- We are storing 2 bytes of memory at address arr+1, and so on.
- arr and arr+1 differ by one unit of space required to store an integer, which is of 2 bytes or *2 addressable memory locations.*
- If our array is storing float values, arr and arr+1 would differ by 4 bytes or 4 addressable memory locations.

A few examples

Example 1

```
int main(){
int arr[6] = {1, 2, 3, 4, 5, 6};
for(int i=0;i<sizeof(arr)/sizeof(arr[0]); i++)
     printf("%d\t",arr[i]);
return 0;
}
```

```
Output?
1     2     3     4     5     6
```

Example 2

```
int main(){
int arr[6] = {1, 2, 3};
for(int i=0;i<sizeof(arr)/sizeof(arr[0]);i++)
     printf("%d\t",arr[i]);
return 0;
}
```

Output?

```
1    2    3    0    0    0
```

Example 3

```
int main(){
int arr[] = {1, 2, 3};
for(int i=0;i<sizeof(arr)/sizeof(arr[0]);i++)
     printf("%d\t",arr[i]);
return 0;
}
```

```
Output?
1    2    3
```

Example 4

```
int main(){
int arr[6];
arr[6] = {1, 2, 3, 4, 5, 6};
for(int i=0;i<sizeof(arr)/sizeof(arr[0]);i++)
     printf("%d\t",arr[i]);
return 0;
}
```

```
Output?
Compile-time error
```

What is the right way in such a scenario (declaration and initialization are separated) ?

Example 5

```
int main(){
    int arr[6];
    for (int i=0;i<6;i++)
        arr[i] = i;
    for (int i=0;i<6;i++)
        printf("%d\t",arr[i]);
    return 0;
}
```

Output?

```
0     1     2     3     4     5
```

Example 6

```
int main(){
    int arr[6];
    for (int i=0;i<6;i++)
        arr[i] = i;
    for (int i=0;i<9;i++)
        printf("%d\t",arr[i]);
    return 0;
}
```

```
Output?
0   1   2   3     4   5   3965    -8905  4872
```

(junk values)

Example 7

```
int main(){
    int arr[6];
    for (int i=0;i<8;i++)
        arr[i] = i;
    for (int i=0;i<6;i++)
        printf("%d\t",arr[i]);
    return 0;
}
```

Problems with arrays – No bounds checking!!

- There is no check in C compiler to see if the subscript used for an array exceeds the size of the array.
- Data entered with a subscript exceeding the array size will simply be placed in memory outside the array limit and lead to *unpredictable* results.
- It's the programmer's responsibility to take care.

Output?

```
*** stack smashing detected ***: ./a.out terminated
0     1     2     3     4     5  Aborted (core dumped)
```

Example 8

```
int main(){
    int arr[6]={0};
    arr[6]= 1;
     for (int i=0;i<7;i++)
        printf("%d\t",arr[i]);
    return 0;
}
```

Output?

```
*** stack smashing detected ***: ./a.out terminated
0       0       0       0       0       0       1
Aborted (core dumped)
```

Example 9

Write a C code which takes an int array of size SIZE and calculates (and displays) the average of all numbers

```
#include<stdio.h>
#include<stdlib.h>

#define SIZE 9
int main()
{
        int arr[SIZE], sum=0;
        for (int i=0;i<SIZE;i++)
                arr[i] =i;
        for (int i=0;i<SIZE;i++)
                sum = sum + arr[i];
        printf("AVG=%d\n",sum/SIZE);
        return 0;
}
```

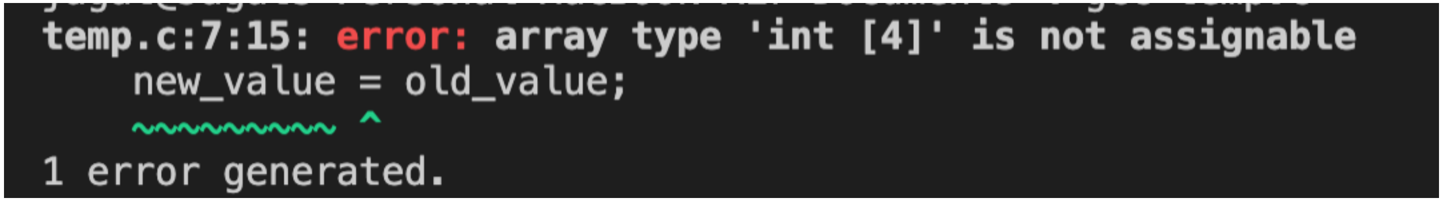
Copying Arrays

```
int old_value[5]={10,20,30,40,50};
int new_value[5];
```

How to copy the elements of **`old_value`** into **`new_value?`**

```
new_value = old_value;
```

It will not work. Why?



```
temp.c:7:15: error: array type 'int [4]' is not assignable
    new_value = old_value;
    ~~~~~~~~~ ^
1 error generated.
```

Question:

What is the correct way of copying arrays?

Sol:

Copy each individual element one by one

Home Exercise

Write a C program to find out the largest element in an array of integers. Assume N numbers are entered by the user (N is #defined)

innovate achieve lead

BITS Pilani
Pilani Campus

Passing Arrays to functions

Passing arrays to functions

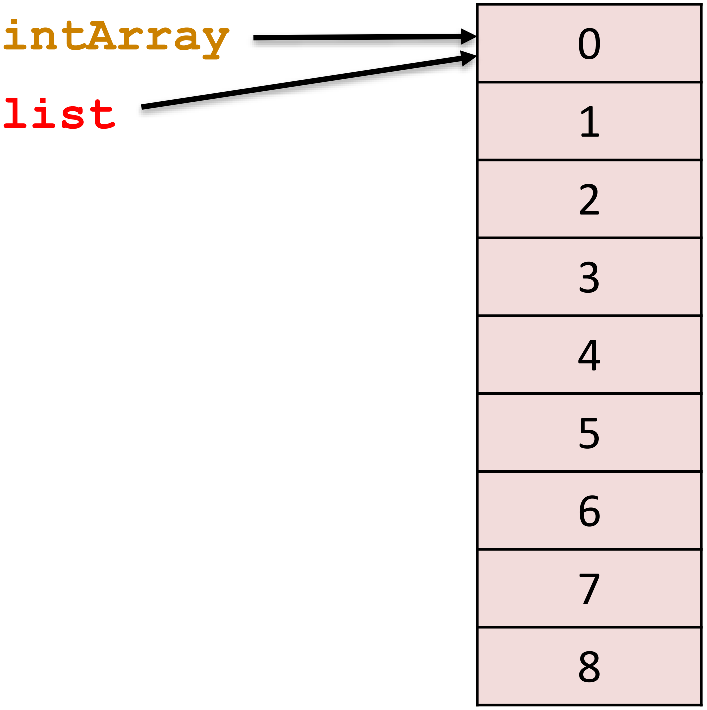
USING FUNCTIONS:

Write a C function that takes an int array of size SIZE and calculates (and displays) the average of all numbers

Solution

```
#define SIZE 9
int avg(int n, int list[]){
   int sum = 0;
   for (int i=0;i<n;i++)
      sum = sum + list[i];
   return sum/n;
}
int main(){
   int intArray[SIZE],
   int average;
   for (int i=0;i<SIZE;i++)
      intArray[i] = i;
   average=avg(SIZE, intArray);
   printf("Average=%d\n", average);
   return 0;
}
Output:
Average=4
```

When *avg()* is called, the address of the first element of the array `intArr` is copied into the array `list`.



0
1
2
3
4
5
6
7
8

Slight change…

```
int main(){
  int intArray[SIZE], average;
  for (int i=0;i<SIZE;i++){
    intArray[i] = i;
    printf(“%d\t”,intArray[i]);
  }

  average=avg(SIZE,intArray);
  printf("Average=%d\n", average);
  printf(“After calling avg…\n”);

  for (int i=0;i<SIZE;i++)
    printf(“%d\t”,intArray[i]);
  return 0;
}
```

```
int avg(int n, int list[])
{
   int sum = 0;
   for (int i=0;i<n;i++){
      list[i] = list[i]*2
      sum = sum + list[i];
   }
   return sum/n;
}
```

Output:

```
0  1  2  3  4  5  6  7  8
Average = 8
After calling avg…
0  2  4  6  8  10 12 14  16
```

Arguments to functions

Ordinary variables are **passed by value**

- *Values of the variables passed are copied into local variables of the function*

However… when an array is passed to a function

- Values of the array *are NOT passed*
- Array name interpreted as the address of the first element of the array is passed **[Pass by reference]**
- This value is captured by the corresponding function parameter, which becomes a **Pointer** to the first element of the array that was passed to it.

Therefore, altering **`list[i]`** within **`avg()`** altered the original values of **`intArray[i]`**.

Can we ‘return’ an array from a function?

If the array is defined inside the function, returning the array would give run-time error.

- Why?
- Try this! Next Slide.

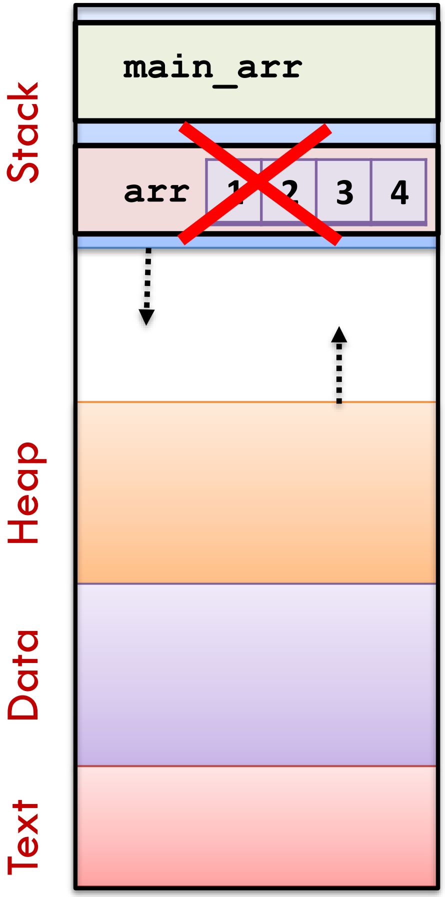
We can return dynamically allocated arrays.

- We will study about them when we study pointers.

We can also use global arrays in place of arrays defined inside functions.

- We don’t have to pass arrays into functions. Global arrays are accessible to all the functions in the program.

Let us try to explain with our memory diagram



Stack frame allocated to *main()* in stack

Stack Frame allocated to *f1()* in the stack

Memory allocated to our program

When f1() returns, the memory allocated to it is destroyed. So arr declared inside f1() does not exist anymore. Accessing arr in main function now gives an error.

Consider this program:

```
int f1(){
   int arr[4] = {1, 2, 3, 4};
   return arr;
}
int main(){
   int main_arr[] = f1();
   printf(”First ele is: %d”, main_arr[0]);
   return 0;
}
Output:
Error!
```

Example with global arrays

```
#define SIZE 4
int globArray[SIZE];
// int globalArray[SIZE] = {1,2,3,4} is also allowed
int avg(){
   int sum = 0;
   for (int i=0;i<n;i++)
      sum = sum + globArray[i];
   return sum/n;
}
```

```
int main(){
   int average;
   for (int i=0;i<SIZE;i++)
      globArray[i] = i;
   average=avg();
   printf("Average=%d\n", average);
   return 0;
}
```

Example (Worked on the Board)

Write a C function which accepts an array as the input and returns the index of the largest element in the array. Assume **N** numbers are entered by the user (**N is #defined**)

innovate achieve lead

BITS Pilani
Pilani Campus

Searching in an Array

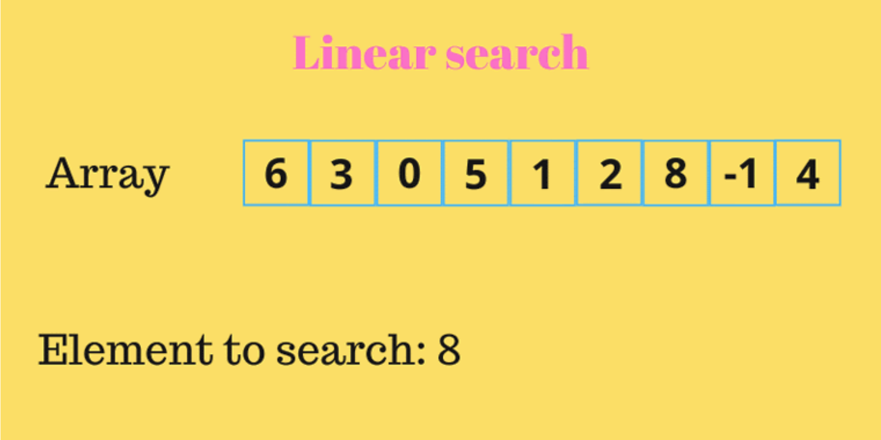
Linear search

Task: Search for an element **key** in the Array.

Each item in the array is examined until the desired item is found or the end of the list is reached

Algorithm:

```
1.  Read an array of N elements named arr[0…N-1] and search element key
2.  Repeat for i=0 to i=N-1
        If key equals to arr[i]
            Display element found and stop
3.  Display element not found
4.  Stop
```



Linear search

Array	6	3	0	5	1	2	8	-1	4

Element to search: 8

Linear Search – Implementation

```
#include <stdio.h>

int linearSearch(int arr[], int size, int key)
{
  // Function implementing linear search of key in array
  arr of size size

  int i = 0;

  for(i=0; i<size; i++){
      if(key == arr[i])
         return i; // element found at index i in the array
  }

  return -1; // element not found in the array
}
```

Linear Search – Implementation (contd.)

```
int main() {
  int arr[10],pos,key;
  printf("Enter Array elements:");
  for(index = 0; index<10; index++)
     scanf("%d",&arr[index]);
  printf("Enter search element");
  scanf("%d", &key);

  pos = linearSearch(arr,10,key);

  if (pos == -1)     printf("Element not found");
  else      printf(“Element found at index %d\n”,pos);

  return 0;
}
```

innovate achieve lead

BITS Pilani
Pilani Campus

Sorting – Selection Sort

Sorting

- Sorting refers to ordering data in an ***increasing*** or ***decreasing*** order
- Sorting can be done by
  - Names
  - Numbers
  - Records
  - etc.
- ***Sorting reduces the time for lookup (or search for an element)***
- Example: Telephone directory
  - Time to search for someone’s phone number
    - Directory sorted alphabetically vs. no ordering

Various Algorithms for Sorting

- Selection sort ← We would be studying in this course!
- Bubble sort
- Insertion sort
- Merge sort
- Quick sort
- Shell sort
- Bucket sort
- Radix sort
- and more . . .

Sorting (Selection Sort)

- Selection sort is a simple sorting algorithm.
- In this algorithm, the list (or an array) is divided into two parts:
  - ***The sorted part at the left***
  - ***The unsorted part at the right***
- *Initially, the sorted part at the left is empty, and the unsorted part at the right is the full list.*
- *In each iteration, the smallest element from the unsorted part of the array is added to the sorted part.*
- *The process continues until the unsorted part of the array is empty.*

Illustrating Selection Sort

14	33	27	10	35	19	42	44

Find the minimum element in the array and bring to the 0th index of the array

14	33	27	10	35	19	42	44

Element 10 is the minimum element in the array. To bring it the 0th index, we need to swap it with the element present at the 0th index

10	33	27	14	35	19	42	44

We can now see that the sorted portion of the array is from index 0 to 0, and the unsorted portion of the array is from index 1 to 7.

Illustrating Selection Sort (Cont.)

10	33	27	14	35	19	42	44

Now, we have to look at the array excluding the first element (element at 0th index). In the remaining (unsorted) portion of the array, again we will find the minimum element.

10	33	27	14	35	19	42	44

The minimum element in the unsorted portion of the array is 14 at index 3. We should swap it with the element at index 1.

10	14	27	33	35	19	42	44

We can now see that the sorted portion of the array is from index 0 to 1, and unsorted portion of the array is from index 2 to 7.

Illustrating Selection Sort (Cont.)

In the same way, we can continue this process. In the end, the sorted portion will be the entire array, and the unsorted portion will be empty.

10	14	27	33	35	19	42	44
10	14	19	33	35	27	42	44
10	14	19	27	35	33	42	44
10	14	19	27	33	35	42	44
10	14	19	27	33	35	42	44
10	14	19	27	33	35	42	44
10	14	19	27	33	35	42	44

Selection Sort: Implementation

```
#include <stdio.h>
void selectionSort(int arr[], int n) {
   int i, j, min;

   // One by one move the boundary of the unsorted subarray
   for (i = 0; i < n-1; i++){
      min = i; //minimum element in unsorted array

      for (j = i+1; j < n; j++){
         if (arr[j] < arr[min])  min = j;
      }

      // Swap the min element with the first element in the unsorted subarray
      int temp = arr[min];
      arr[min] = arr[i];
      arr[i] = temp;
   }
}
```

Does this function need to return anything?

Selection Sort: Implementation (contd.)

```
int main()
{
    int a[] = {24, 36, 20, 7, 42, 19};
    int size = sizeof(a) / sizeof(a[0]);
    selectionSort(a, size);
    return 0;
}
```

innovate achieve lead

BITS Pilani
Pilani Campus

Binary Search in an array

Binary Search

- Useful when the array is already sorted
- More efficient than linear search
- It performs lesser number of comparisons with the elements in the array, when compared to linear search
- Hence, very fast!

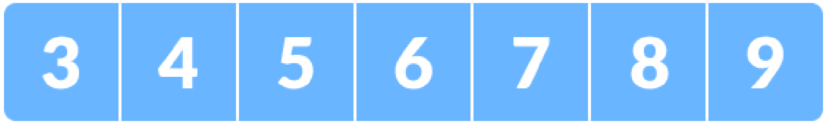
Binary Search Algorithm

Algorithm // Pre-condition: List must be sorted

- The desired item is first compared to the element in the middle of the list
- If the desired item is equal to the middle element:
  - No further searches are required
- If the desired item is greater than the middle element:
  - The left part of the list is discarded from any further search
- If the desired item is less than the middle element:
  - The right part of the list is discarded from any further search
- This process continues either until element is found or list reaches to singleton element

Illustrating Binary Search

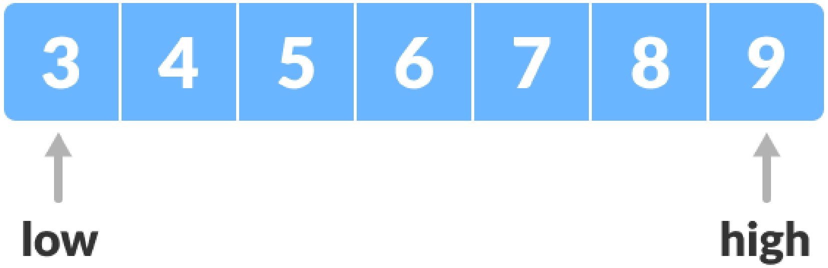
1. The array in which searching is to be performed is:



Initial Array(sorted)

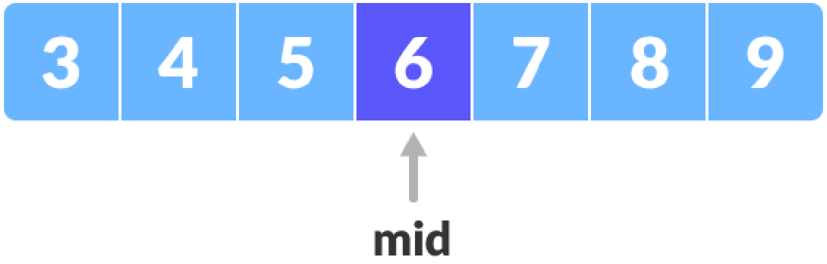
Let x = 4 be the element to be searched.

2. Set two pointers low and high at the lowest and the highest positions respectively.

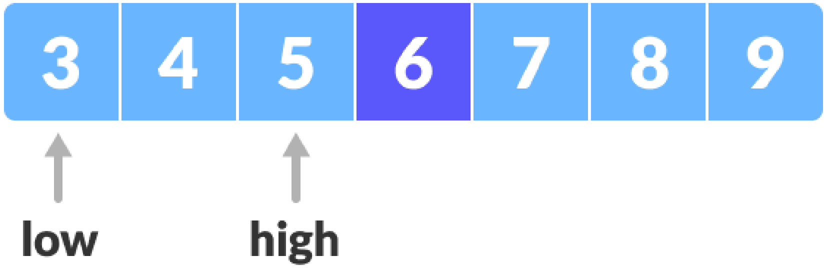


Illustrating Binary Search

3. Find the position of the middle element mid = (low+high)/2. The middle element is arr[mid] = 6.

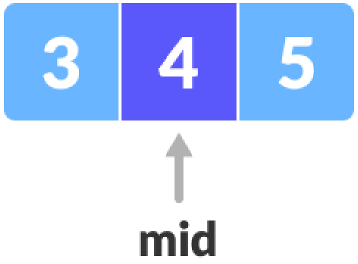


4. If x==arr[mid], then **return** mid.
5. Else If x > arr[mid], ***compare x with the middle element of the elements on the right side of mid***. This is done by setting low to low = mid + 1.
6. Else If x < arr[mid], ***compare x with the middle element of the elements on the left side of mid***. This is done by setting high to high = mid - 1.

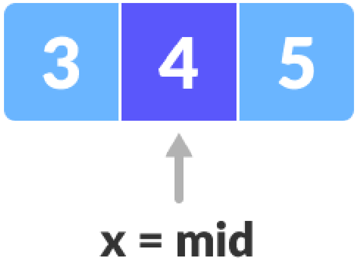


Illustrating Binary Search

7. Repeat steps 3 to 6 until low meets high.



8. x = 4 is found.



Binary Search - Code

```
#define Size 10
main(){            // Binary search implementation
  int arr[Size], index, upper, lower, key, mid;
  printf("Enter Array elements:");
  for(index = 0; index<Size; index++)
      scanf("%d", &arr[index]);
  printf("Enter search element");
  scanf("%d",&key);
  upper=Size-1;  lower=0;
  while(lower<=upper){
      mid=(lower+upper)/2;
      if(key>arr[mid])                    lower=mid+1;
      else if(key<arr[mid])               upper=mid-1;
      else{
            printf("Element found at location %d", mid);
            return;
      }
  }
  printf("Element no found");
}
```

Exercise: Re-write this program to do the searching part with a function call.

BITS Pilani
Pilani Campus

Insert/Delete in an Array

Handling changing number of elements in the array

Insert/Delete in an Array

- ***Arrays once declared they are of fixed size.***
- Example:

  `int arr1[10];` declares an array of size 10.

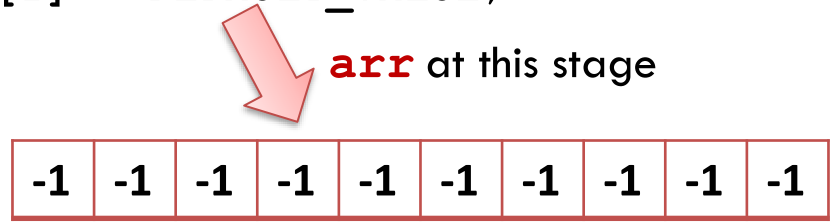
  We can't store 11 elements to this array.

- To enable insertion and deletion in an array
  - Choose `MAX_SIZE` and declare the array with it.
  - Insertions can be done as long as the number of elements of the array does not exceed `MAX_SIZE.`
  - Keep a `count` of actual number of elements present in the array
    - `count <= MAX_SIZE`
  - Some positions will remain vacant. We shall need to store a `DEFAULT_VALUE` in those positions.
  - Keep all occupied positions contiguous (Unoccupied positions as well)
    - If `MAX_SIZE` is 10, `count` is 6, then first 6 positions of the array should be occupied with some values and remaining 4 positions should have `DEFAULT_VALUE`.

Insert/Delete in an Array

- Choose **MAX_SIZE** and declare the array with it.
- Insertions can be done as long as the number of elements of the array does not exceed **MAX_SIZE.**
- Keep a **count** of actual number of elements present in the array
  - **count <= MAX_SIZE**
- Some positions will remain vacant. We shall need to store a **DEFAULT_VALUE** in those positions.
- Keep all occupied positions contiguous (Unoccupied positions as well)
  - If **MAX_SIZE** is 10, **count** is 6, then first 6 positions of the array should be occupied with some values and remaining 4 positions should have **DEFAULT_VALUE**.

```
#define MAX_SIZE 10
#define DEFAULT_VALUE -1
int main() {
// declare array with MAX_SIZE
int arr[MAX_SIZE];

// declare a variable to keep count of
number of elements in the array
int count = 0;

// fill all the positions with the
DEFAULT_VALUE
for(int i=0;i<MAX_SIZE;i++){
    arr[i] = DEFAULT_VALUE;
}
…  …  …
}
```



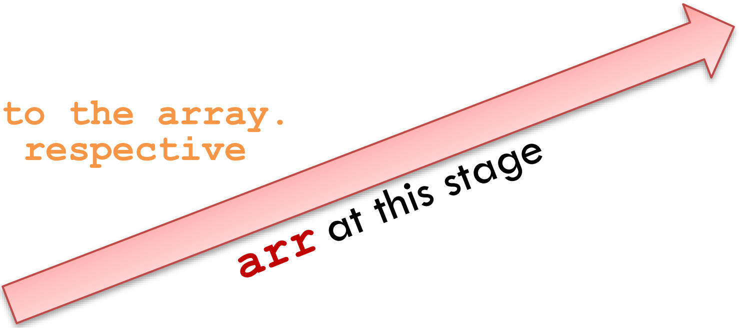
arr at this stage

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Insert in an Array

```
#define MAX_SIZE 10
#define DEFAULT_VALUE -1
int main() {
int arr[MAX_SIZE];
int count = 0;

for(int i=0;i<MAX_SIZE;i++){
    arr[i] = DEFAULT_VALUE;
}

// Insert 6 elements into the array.
Insert squares of their respective
index.
for (int i=0;i<6;i++){
        arr[i] = i*i;
        count+=1;
}

// Append an element at the end of
the array and increment count
arr[count]=12;
count+=1;

}
```

0	1	4	9	16	25	-1	-1	-1	-1



Element inserted here

0	1	4	9	16	25	12	-1	-1	-1

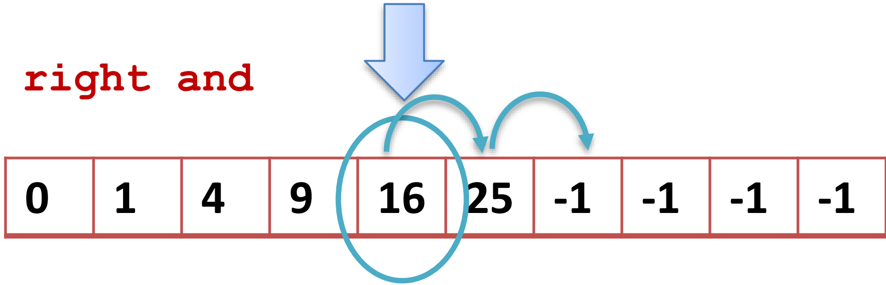
Insertion in a sorted array

```
#define MAX_SIZE 10
#define DEFAULT_VALUE -1
int main() {
int arr[MAX_SIZE];
int count = 0;

for(int i=0;i<MAX_SIZE;i++){
    arr[i] = DEFAULT_VALUE;
}

for (int i=0;i<6;i++){
        arr[i] = i*i;
        count+=1;
}

// Insert 10 into arr, while keeping arr sorted
// It should get inserted between 9 and 16.
// What should be done ?
// Shift 25, 16, by one place towards right and
// insert 10 in the place of 16.

}
```

0	1	4	9	16	25	-1	-1	-1	-1

arr at this stage

Insert 10 here after shifting elements to the right

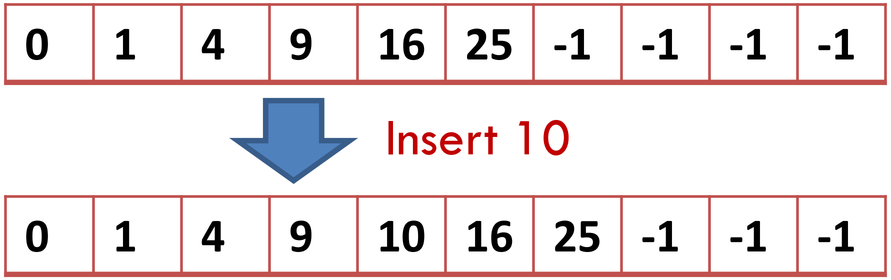


0	1	4	9	16	25	-1	-1	-1	-1

Insertion in a sorted array

```
#define MAX_SIZE 10
#define DEFAULT_VALUE -1
int main() {
…
…

int x=10; //element to be inserted
int i=0;

// Find the position to insert x
for (i=0; i<count; i++){
   if (arr[i]>=x)
        break;
}
// i is the position in arr where
   x should be inserted
```

```
// Now shift elements from last
   occupied position until i, one
   position to right
for (int j=count-1; j>i;j--){
   arr[j+1] = arr[j];
}
// loop exits when j=i, the
   position to insert x.

arr[j] = x; count++;
// x inserted at its position
}
```



0	1	4	9	16	25	-1	-1	-1	-1

Insert 10

0	1	4	9	10	16	25	-1	-1	-1

Delete in Array (Sorted or unsorted)

Delete is similar to insert.

Example: *Delete 9 from arr*

0	**1**	**4**	**9**	**10**	**16**	**25**	**-1**	**-1**	**-1**

Delete 9 from arr:

- shift 10,16,25 to one position towards left
- `arr[count-1] = DEFAULT_VAL`
- `count = count -1`

0	**1**	**4**	**10**	**16**	**25**	**-1**	**-1**	**-1**	**-1**

Exercise: Implement this operation

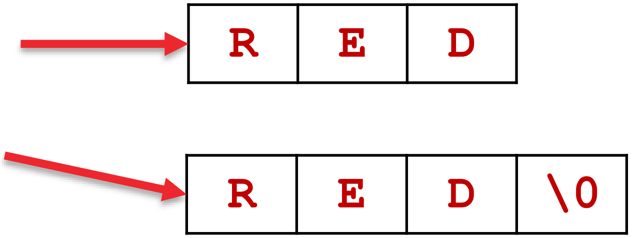
innovate achieve lead

BITS Pilani
Pilani Campus

Character Arrays

Character Arrays

```
char color[3] = "RED";
char color[ ] = "RED";
```



R	E	D

R	E	D	\0

Are they the same?

Character arrays are the way to represent **strings** in C.

Each string typically ends with a NULL character `"\0"`.

More about strings in Module 13.

Char Arrays

Write a C program which reads a 1D char array, converts all elements to uppercase, and then displays the converted array.

Hint: use toupper(ch) of <ctype.h>

Lower Case to Upper Case using char array

```
#include <stdio.h>
#include <ctype.h>
int main(){
  int size,i=0;
  char name[50];
  name[0]=getchar();
  while(name[i]!='\n')
  {
    i++;
    name[i]=getchar();
  }
  name[i]='\0';   ????
  size = i;
  printf("\nName is %s", name);
  for(i=0;i<size;i++)
    putchar(toupper(name[i]));
  return 0;
}
```

Char Arrays

Modify the previous code to use scanf()/printf() in place of getchar()/putchar()

Lower Case to Upper Case using char array

```
#include <stdio.h>
#include <ctype.h>
int main(){
  int size,i=0;
  char name[50];
  scanf(“%c ”,&name[0]);
  while(name[i]!='\n')
  {
    i++;
    scanf(“%c ”,&name[i]);
  }
  name[i]='\0';
  size = i;
  printf("\nName is %s",name);
  for(i=0;i<size;i++)
      printf(toupper(name[i]));
  return 0;
}
```

Char arrays – using gets/puts

```
int main()
{
        char c[20];
        int i=0;
        printf("Enter the Name");
        gets(c);
        printf("The name is %s",c);
        puts(c);
        return 0;
}
```

Note:

- It is not required to explicitly insert "\0" character at the end of each character array while using `gets()`.
- It automatically adds so.

innovate achieve lead

BITS Pilani
Pilani Campus

Multi-dimensional Arrays in C

Multi-dimensional Arrays

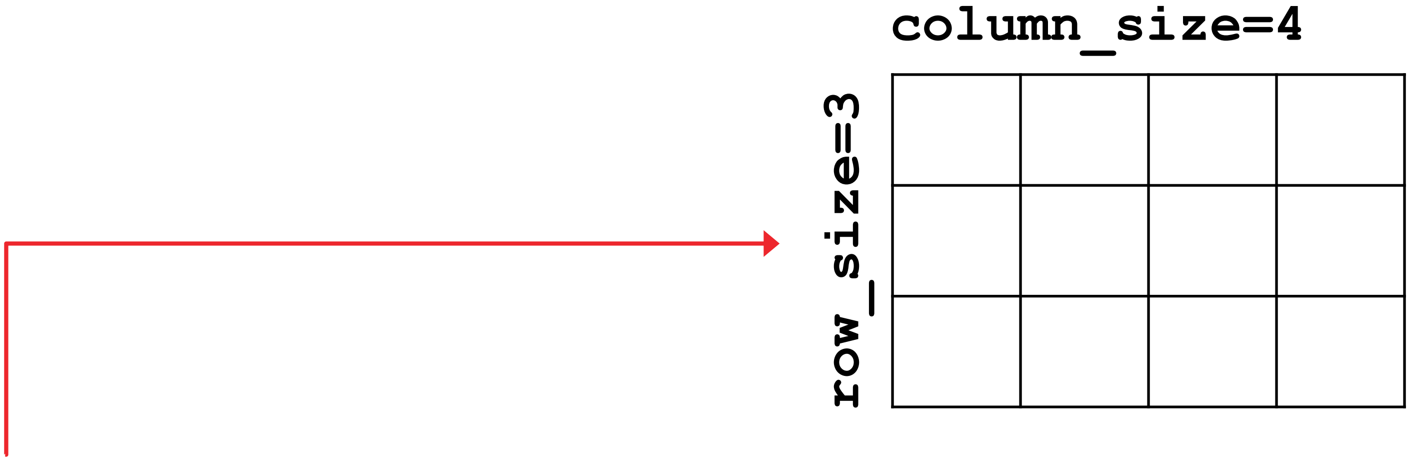
- C supports arrays of multiple dimensions
- A basic multi-dimensional array is a 2-D array
  - Also known as a matrix

Declaring 2-D Arrays - Syntax:

```
type variable_name[row_size][column_size];
```

```
row_size        →  Number of rows in the matrix
column_size     →  Number of columns in the matrix
```

Examples of 2-D Arrays



Examples:

```
int number[3][4]; /* 12 elements */
float number[3][2]; /* 6 elements */
char name[10][20]; /* 200 chars */
```

Initializing a 2-D Array

```
int a[2][3]={1,2,3,4,5,6};
int a[2][3]={{1,2,3}, {4,5,6}};
int a[][3]={{1,2,3}, {4,5,6}}
```

All are equivalent are produce the following array:

1	2	3
4	5	6

How will the values will be assigned in each case?

Incorrect Ways of Initializing 2-D Arrays

Following initializations are not allowed

```
int a[3][]={2,4,6,8,10,12};
int a[][]={2,4,6,8,10,12};
```

Note:

- If the first bracket pair is empty, then the compiler takes the size from the number of inner brace pairs
- If the second bracket pair is empty, the compiler throws a compilation error!

Accessing a 2-D Array

```
int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
for(i=0;i<3;i++)
{
        for(j=0;j<4;j++)
        {
            printf("%d",a[i][j]);
        }
        printf("\n");
}
```

Run time initialization of an array can be done in a similar way by changing ***`printf()`*** *to* ***`scanf()`*** *inside the j loop.*

innovate achieve lead

BITS Pilani
Pilani Campus

Memory Maps of 2-D Arrays

Storing 2-D Arrays in Memory

- 2-D arrays are stored in the memory as a linear sequence of variables
- Two methods for storing:
  - **Row major**
  - **Column major**

Row Major vs. Column Major

Example:

```
int a[3][3];
```

Row major storage will have this sequence in main memory:

```
    a          a+1        a+2        a+3        a+4
a[0][0], a[0][1], a[0][2], a[1][0], a[1][1],
   a+5        a+6        a+7        a+8
a[1][2], a[2][0], a[2][1], a[2][2]
```

Indicates addresses

Column major storage will have this sequence in main memory:

```
    a          a+1        a+2        a+3        a+4
a[0][0], a[1][0], a[2][0], a[0][1], a[1][1],
   a+5        a+6        a+7        a+8
a[2][1], a[0][2], a[1][2], a[2][2]
```

innovate achieve lead

BITS Pilani
Pilani Campus

Matrix Addition

Matrix Addition and Subtraction

Matrix Addition:

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 \\ 1+7 & 0+5 \\ 1+2 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 8 & 5 \\ 3 & 3 \end{bmatrix}$$

Matrix Subtraction:

$$\begin{bmatrix} 1 & 3 \\ 1 & 0 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 7 & 5 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1-0 & 3-0 \\ 1-7 & 0-5 \\ 1-2 & 2-1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ -6 & -5 \\ -1 & 1 \end{bmatrix}$$

Working with two dimensional Arrays (Matrix Addition & Subtraction)

Let `A[m][n]` and `B[p][q]` be two matrices.

Precondition: *m equals to p and n equals to q.*

Algorithm Steps:

```
1. Read two matrices A and B, and initialize C matrix to zero
2. Repeat (3) for i=0 to m-1
3. Repeat (3.a) for j=0 to n-1
    3.a) C[i][j] = A[i][j] + B[i][j]
4. Display C matrix
```

Matrix Addition: Code

```
#define ROW 10
#define COL 10
int main(){
   int M1[ROW][COL],M2[ROW][COL],M3[ROW][COL],i,j;
   int row1,col1,row2,col2;
   printf(“Enter row value for M1\n”);
   scanf(“%d”,&row1);
   printf(“Enter column value for M1\n”);
   scanf(“%d”,&col1);
   printf(“Enter row value for M2\n”);
   scanf(“%d”,&row2)
   printf(“Enter column value for M2\n”);
   scanf(“%d”,&col2)
```

Matrix Addition: Code (Contd.)

```
if(row1!=row2 || col1!=col2)
{
    printf(“Invalid Input: Addition is not possible”);
    return;
}
printf(“Enter data for Matrix M1\n”);
for(i=0;i<row1;i++)
{
    for(j=0;j<col1;j++)
    {
          scanf(“%d”,&M1[i][j]);
    }
    printf(“\n”);
}
```

Matrix Addition: Code (contd.)

```
printf(“Enter data for Matrix M2\n”);
for(i=0;i<row2;i++)
{
    for(j=0;j<col2;j++)
    {
           scanf(“%d”,&M2[i][j]);
    }
    printf(“\n”);
}
printf(“Addition of Matrices is as follows\n”);
for(i=0;i<row2;i++)
    for(j=0;j<col2;j++)
          M3[i][j]= M1[i][j] + M2[i][j]);
```

Matrix Addition: Code (contd.)

```
  // display the new matrix after addition
  for(i=0;i<row2;i++)
  {
      for(j=0;j<col2;j++)
      {
            printf(“%d”,M3[i][j]);
      }
      printf(“\n”);
  }
}
```

Matrix Subtraction can be done in a similar way

innovate achieve lead

BITS Pilani
Pilani Campus

Matrix Multiplication

Matrix Multiplication

$$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} \text{ X } \begin{bmatrix}10 & 11\\ 20 & 21\\ 30 & 31\end{bmatrix}$$

$$= \begin{bmatrix}1\text{x}10 + 2\text{x}20 + 3\text{x}30 & 1\text{x}11 + 2\text{x}21 + 3\text{x}31\\ 4\text{x}10 + 5\text{x}20 + 6\text{x}30 & 4\text{x}11 + 5\text{x}21 + 6\text{x}31\end{bmatrix}$$

$$= \begin{bmatrix}10+40+90 & 11+42+93\\ 40+100+180 & 44+105+186\end{bmatrix} = \begin{bmatrix}140 & 146\\ 320 & 335\end{bmatrix}$$

Matrix Multiplication M1xM2

Pre-condition: Number of columns in M1 should be equal to number of rows in M2

Algorithm Steps:

```
1. Read two matrices M1[m][n] and M2[n][r] and initialize
   another matrix M3[m][r] for storing result
2. Repeat for i=0 to m-1
       Repeat for j=0 to r-1
           M3[i][j] = 0
           Repeat for k=0 to n-1
               M3[i][j] += M1[i][k] * M2[k][j]
3. Print matrix M3
```

Matrix Multiplication: Code

```
#include <stdio.h>
#define row1 4
#define col1 3
#define row2 3
#define col2 4
#define row3 4
#define col3 4

void main()
{
   int M1[row1][col1],M2[row2][col2],M3[row3][col3];
   int i,j,k;
```

Matrix Multiplication: Code (contd.)

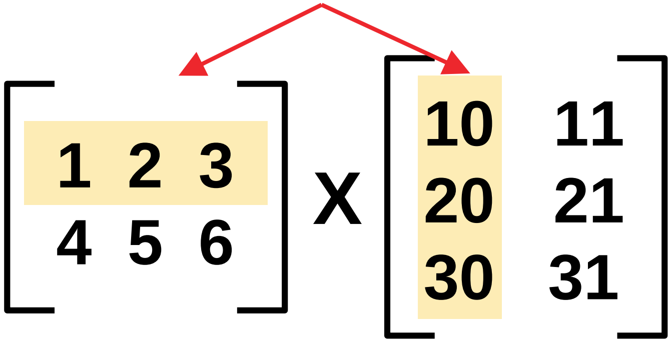
```
printf(“Enter data for Matrix M1\n”);
for(i=0;i<row1;i++)
{
     for(j=0;j<col1;j++)
     {
          scanf(“%d”,&M1[i][j]);
     }
     printf(“\n”);
}
```

Matrix Multiplication: Code (contd.)

```
printf(“Enter data for Matrix M2\n”);
for(i=0;i<row2;i++)
{
     for(j=0;j<col2;j++)
     {
          scanf(“%d”,&M2[i][j]);
     }
     printf(“\n”);
}
```

Matrix Multiplication: Code (contd.)

```
if (col1!= row2){
    printf("Multiplication is not possible");
    return;
}
for(i=0;i<row1;i++){
    for(j=0;j<col2;j++){
        M3[i][j] = 0;
        for(k=0;k<col1;k++){
            M3[i][j] += M1[i][k] * M2[k][j];
        }
    }
}
```



$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} X \begin{bmatrix} 10 & 11 \\ 20 & 21 \\ 30 & 31 \end{bmatrix}$$

Matrix Multiplication: Code (contd.)

```
 printf(“RESULT MATRIX IS\n ”);
 for(i=0;i<row3;i++){
     for(j=0;j<col3;j++){
         printf(“%d”,M3[i][j]);
     }
     printf(“\n”);
 }
 return;
}
```

innovate achieve lead

BITS Pilani
Pilani Campus

n-dimensional arrays: A glimpse

N-dimensional arrays

3D array: `int arr[2][2][3];`

4D array: `int arr[2][2][2][2];`

5D array: `int arr[2][2][2][2][2];`

…

Note:

- Elements are stored and accessed in a similar way as that of 2D arrays.
- They are also stored in Row Major format.

Exercises

Q. 1 Generate Fibonacci series using Array.

Q. 2 Write a program to find a binary equivalent of a decimal number using an array.

Q. 3 Write a program to find the transpose of a matrix.

innovate achieve lead

BITS Pilani
Pilani Campus

Thank you

Q & A