First Year First Semester Course

M.Tech. (CS) [Batch 2021-23]

Lecture #06

# Introduction to Programming

C: Arrays and Strings


Course Instructor:

**Dr. Monidipa Das**

**DST-INSPIRE Faculty**

**Machine Intelligence Unit (MIU), Centre for Artificial Intelligence and Machine Learning (CAIML)**

**Indian Statistical Institute (ISI) Kolkata, India**

# Arrays



- **Derived** data type
- A fixed-size sequenced collection of elements of the same (homogenous) data type
- All the data items constituting the group share the same name.

```
int x[10];
```

- Individual elements are accessed by specifying the index.

| X[0] | X[1] | X[2] | X[3] | X[4] | X[5] | X[6] | X[7] | X[8] | X[9] |
|---|---|---|---|---|---|---|---|---|---|
| 12 | 30 | 5 | 9 | 17 | 14 | 11 | 6 | 23 | 19 |

**x is a 10-element 1D array**

- Arrays can be of one-dimensional (1D), two-dimensional (2D), or even multi-dimensional (MD)

# Declaring Arrays



- Like the variables of fundamental data types, the arrays must be declared before these are used in a program

- **General syntax (for 1D array):**

*<type> <array-name> [<size>];*

- **Example:**

*array-name*

```
int marks[20];
```

/* marks is an array containing maximum 20 integers. */

# How an array is stored in memory?



- Starting from a given memory location, the successive array elements are allocated space in consecutive memory locations.
- Let


**x**: starting address of the array in memory

**k**: number of bytes allocated per array element

Element **a[i]** is allocated memory location at address **x + i*k**

```
int marks[7];
```

| *Array Index →* | marks[0] | marks[1] | marks[2] | marks[3] | marks[4] | marks[5] | marks[6] |
|---|---|---|---|---|---|---|---|
| *Array Element →* | 52 | 86 | 68 | 71 | 78 | 56 | 90 |
| *Memory Address →* | 602124 | 602128 | 602132 | 602136 | 602140 | 602144 | 602148 |

# Accessing Array Elements



- A particular element of the array can be accessed by specifying two things:
  - Name of the array
  - Index (relative position) of the element in the array
- In C, the **index** of an array **starts from zero**.
- **Example:**

```
int x[10];
```

  The 1st element of the array **x** can be accessed as **`x[0]`**, 6th element as **`x[5]`**, etc.

- The array index must evaluate to an integer between 0 and $n - 1$ where $n$ is the number of elements in the array.

$$x[i + 2] = 18;$$
$$y[3 * i - j] = x[10 - i] + 6;$$

# Initialization of Arrays



- **General form: (for 1D Array)**

  <type> <array_name>[<size>] = {<list of values>};

- **Examples:**

```
int marks[5] = {72, 83, 65, 80, 76};
char name[4] = {'A', 'm', 'i', 't'};
```

Compile time initialization

- **Some special cases:**
  - If the number of values in the list is less than the number of elements, the remaining elements are automatically set to zero.
  - The size may be omitted. The compiler automatically allocates enough space for all initialized elements.

Run time initialization

```
int sum[50];
for(i=0;i<50;i++)
          sum[i]=0;
```

```
float price[10];
for(i=0;i<10;i++)
  scanf("%f",&price[i]);
```

# Remember



```
int a[10], b[10];
```

- **You cannot**
  - use = to assign one array variable to another

    ```
    a = b;  /* a and b are arrays */
    ```
    X
  - use == to directly compare array variables

    ```
    if (a = = b) …
    ```
    X
  - directly scanf or printf arrays

    ```
    printf ("……", a);
    ```
    X

# Accessing Array



```
int a[20], b[20];
```

- **Reading the elements one at a time**

```
for (i=0; i<20; i++)
        scanf("%d", &a[i]);
```

  - The ampersand (&) is necessary.
  - The elements can be entered all in one line (space separated) or in different lines.

- **Copying the elements of one array to another**
  - By copying individual elements

```
for (i=0; i<20; i++)
        b[i] = a[i];
```

# Accessing Array [contd.]



```
int a[25];
```

- **Printing Array: (Traversing one dimensional array)**

  by printing one element at a time.

  – The elements can be printed one per line

```
for (i=0; i<25; i++)
        printf("\n %d", a[i]);
```

  – All the elements can be printed in one line (starting with a new line)

```
printf ("\n");
for (i=0; i<25; i++)
        printf ("%d ", a[i]);
```

# Examples



- **Find the minimum of a set of 10 numbers**

```
#include <stdio.h>
int main(){
        int a[10], i, min;

        printf(“Enter 10 values: \n");

        for (i=0; i<10; i++)
                scanf ("%d", &a[i]);
        min = 99999;
        for (i=0; i<10; i++){
                if (a[i] < min)
                        min = a[i];
        }
        printf ("\nThe minimum value is %d", min);
        return 0;
}
```

Array Declaration

Reading Array Element

Accessing Array Element

# Merging Two One Dimensional Arrays



```
#include<stdio.h>
#define MAXSIZE 10
//This program merges two 1D arrays into a 3rd one.
int main(){
            int i,a[MAXSIZE],b[MAXSIZE],c[2*MAXSIZE],m,n;
            printf("Enter the count of elements in the 1st array (maximum 10 allowed): ");
            scanf("%d",&m);
            printf("\nEnter the elements of the 1st array:\n");
            for(i=0;i<m;i++)
                        scanf("%d",&a[i]);
            printf("\nEnter the count of elements in the 2nd array (maximum 10 allowed): ");
            scanf("%d",&n);
            printf("\nEnter the elements of the 2nd array:\n");
            for(i=0;i<n;i++)
                        scanf("%d",&b[i]);
            for(i=0;i<m;i++)
                        c[i]=a[i];     /*copying the elements of the 1st array into the 3rd array*/
            for(i=0;i<n;i++)
                        c[m+i]=b[i]; /*copying the elements of the 2nd array into the 3rd array*/
            printf("\nThe array generated after the merging operation is:\n");
            for(i=0;i<m+n;i++)
                        printf("%d ",c[i]);
            return 0;
}
```

# Merging Two One Dimensional Arrays



**Output:**

```
Enter the count of elements in the 1st array (maximum 10 allowed): 3

Enter the elements of the 1st array:
12
46
80

Enter the count of elements in the 2nd array (maximum 10 allowed): 4

Enter the elements of the 2nd array:
22
68
35
91

The array generated after the merging operation is:
12 46 80 22 68 35 91
```

# Two-Dimensional (2D) Arrays



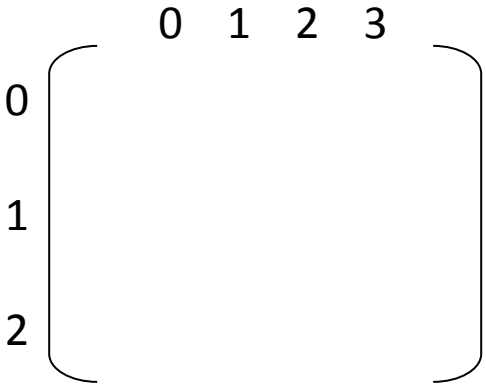
- Useful in a situation where a table of values will have to be handled
- **Declaration:**

*<type> <array name> [<row size>][<column size>];*

```
int b[3][4];
```

Starts from b[0][0] upto b[2][3]

**Logical view**




3 × 4 matrix

Physical view of a 2D array is much different from its logical view because they are stored linearly in memory

# Memory Allocation for 2D Array



- **Row major**

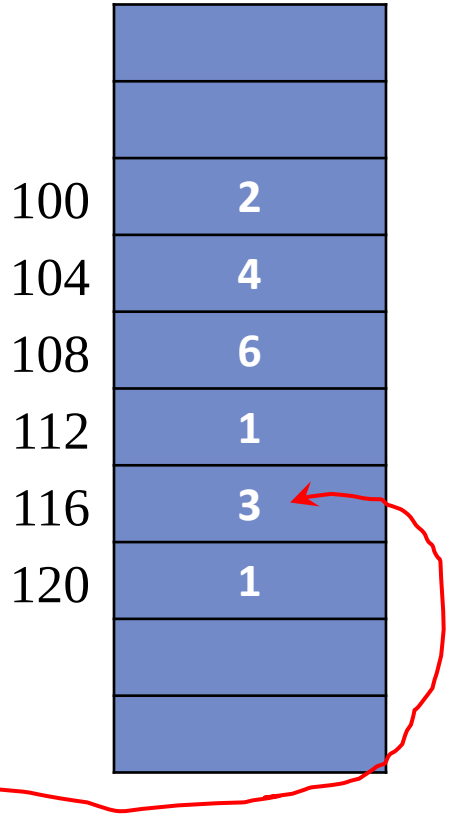
$$\begin{pmatrix} 2 & 4 & 6 \\ 1 & 3 & 1 \end{pmatrix}$$

```
int A[NROW][NCOL];
```

Address computation for A[r][c]

- Indexing starts at 0
- Base + size-of-int[r*NCOL+c]

Address of A[1][1] = 100+4*[1*3+1] = 116



# Memory Allocation for 2D Array [contd.]



- **Column major**

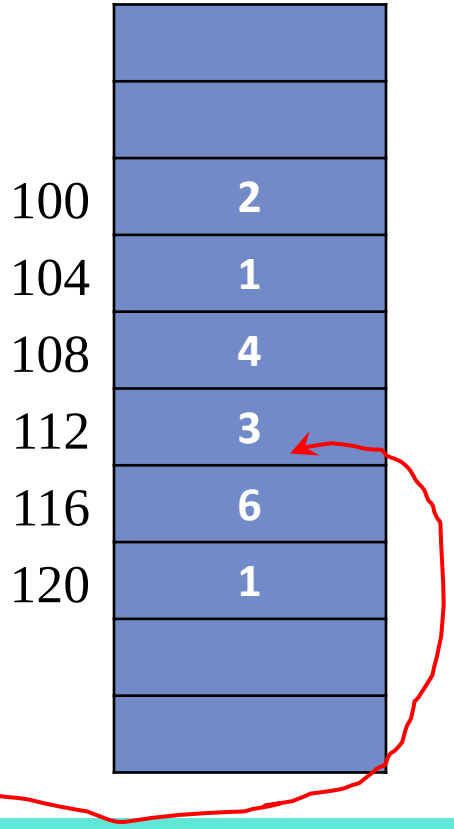
$$\begin{pmatrix} 2 & 4 & 6 \\ 1 & 3 & 1 \end{pmatrix}$$

```
int A[NROW][NCOL];
```

Address computation for A[r][c]

- Indexing starts at 0
- Base + size-of-int[c*NROW+r]

Address of A[1][1] = 100+4*[1*2+1] = 112



# Initializing 2D Arrays



- Compile time initialization

```
int table[2][3]={0,0,0,1,1,1};
int table[2][3]={{0,0,0},{1,1,1}};
```

Special cases:

```
int table[][3]={{0,0,0},{1,1,1}};
int table[][3]={{1,1},{2}};
int arr[3][5]={{0},{0},{0}};
int arr[3][5]={0,0};
```

- Run time initialization

```
int arr[3][5];
for(i=0;i<3;i++)
    for(j=0;j<5;j++)
        arr[i][j]=0;
```

```
float tab[3][5];
for(i=0;i<3;i++)
  for(j=0;j<5;j++)
    scanf("%f",&tab[i][j]);
```

# Accessing 2D Array



```
int arr[3][4];
```

- **Reading the elements one at a time**

```
for (i=0; i<3; i++)
        for (j=0; j<4; j++)
                scanf("%d", &arr[i][j]);
```

- **Printing 2D Array:**
  – The elements can be printed in matrix/table form

```
for (i=0; i<3; i++){
        for (j=0; j<4; j++)
                printf("%d ", arr[i][j]);
        printf("\n");
}
```

# Searching an Array



- **Searching:**
  - Check if a given element (***key***) occurs in the array.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 56 | 24 | 68 | 45 | 90 | 18 | 86 | 75 |

Is 68 present in the array?

Is 39 present in the array?

- Linear search, Binary search

# Linear Search



- Basic idea:
  - Start at the beginning of the array.
  - Inspect every element to see if it matches the key.

- Time complexity:
  - A measure of how long an algorithm takes to run.
  - If there are $n$ elements in the array:
    - **Best case:** match found in first element (1 search operation)
    - **Worst case:** no match found, or match found in the last element ($n$ search operations)
    - **Average:** $(n + 1) / 2$ search operations

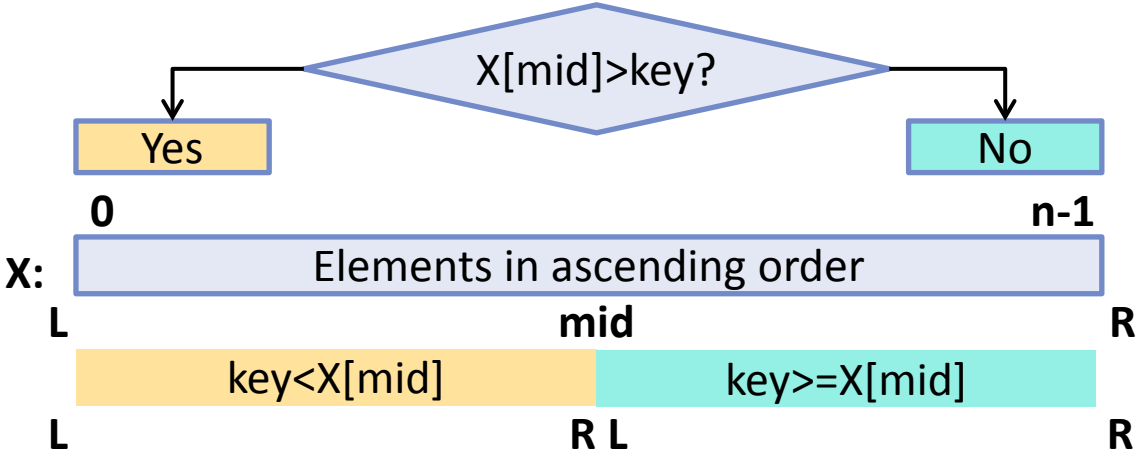
# Linear Search



```
/* If key appears in a[0..size-1], print its location pos, where a[pos-1] == key. Else print unsuccessful search */
#include <stdio.h>
#include <stdlib.h> /* for exit() function */
#define SIZE 100
int main()
{           int size,a[SIZE],key,i,pos;
            printf("Enter the number of elements: ");
            scanf("%d",&size);
            if(size>SIZE) { /* size is a variable, SIZE is not!! */
                        printf("Array Size error!!! I am exiting .... \n");
                        exit(0);
            }
            printf("Enter the elements: ");
            for(i=0;i<size;i++)
                        scanf("%d",&a[i]);
            printf("\nEnter the key element: ");
            scanf("%d",&key);
            for(pos=-1,i=0;i<size;i++) { /* initializing pos as unsuccessful search*/
                        if(a[i]==key) {
                                    pos=i;
                                    break;}
            }
            (pos==-1)? printf("Unsuccessful search\n"):printf("The element is present at position %d \n",pos+1);
            return 0;}
```

# Binary Search: The Basic Strategy

Binary search works ***if the array is sorted***.




- Look at [(**L**+**R**)/2].
- Move **L** or **R** to the middle depending on the test.
- Repeat search operation in the reduced interval.

**In every step, we reduce the number of elements to search in by half.**

# Binary Search [contd.]



```
#include<stdio.h>
//If key appears in x[0..size-1], prints its location pos where x[pos-1]==key. If not found, print "Search unsuccessful!"
int main ()
{           int x[100],size,key,i;
            int L, R, mid;
            printf("Enter the number of elements: ");
            scanf("%d",&size);
            printf("Enter the elements: ");
            for(i=0;i<size;i++)
                        scanf("%d",&x[i]);
            printf("Enter the key element: ");
            scanf("%d",&key);
            L = -1; R = size;
            while ( L+1 != R ) {
                        mid = (L + R) / 2;
                        if (x[mid] <= key)
                                    L = mid;
                        else R = mid;
            }
            if (L >= 0 && x[L] == key) printf("\nPosition of the key is: %d",L+1);
            else printf("\nSearch unsuccessful!");
            return 0;
}
```

**Typical Outputs:**

```
Enter the number of elements: 5
Enter the elements: 12 23 34 45 56
Enter the key element: 39

Search unsuccessful!
```

```
Enter the number of elements: 6
Enter the elements: 12
23
34
45
56
67
Enter the key element: 56

Position of the key is: 5
```

# Binary Search: Example



- **Sorted array**

| X: | 18 | 24 | 45 | 53 | 65 | 75 | 86 | 90 |
|---|---|---|---|---|---|---|---|---|

- **Trace**:

Search 53;

| | | | |
|---|---|---|---|
| L= -1; R= 8; | L+1!=R | mid=3; | x[3]<=53 |
| L= 3; R= 8; | L+1!=R | mid=5; | x[5]>53 |
| L= 3; R= 5; | L+1!=R | mid=4; | x[4]>53 |
| L= 3; R= 4; | L+1!=R →False! | | |

3 >= 0 && x[3] == 53 →True

Position of the key is L+1= 3+1=4

# Is binary search more efficient?



- Suppose there are 1000 elements.

- Linear search
  - If key is a member of x, it would require ***500 comparisons*** on the average.

- Binary search
  - after 1st compare, left with 500 elements.
  - after 2nd compare, left with 250 elements.
  - After ***at most 10 steps***, you are done.

If there are n elements in the array. Number of searches required: $\log_2 n$

# Strings



- Array of characters
- Usually one extra character is required to store the null character.
- The null character ('\0') indicates the end of the string.
- Declaration:

```
char <string_name>[size];
```

**Example:**

```
char name[30];
char address[80];
char city[20];
```

- Initialization:

```
char city[8]= "KOLKATA";
char city[8]= {'K', 'O', 'L', 'K', 'A', 'T', 'A', '\0'};
char city[]= {'K', 'O', 'L', 'K', 'A', 'T', 'A', '\0'};   /*Allowed */
char city[20]= "KOLKATA";   /* Allowed */
char city[7]= "KOLKATA";   /* Illegal */
```

```
char city[8];
city="KOLKATA";  /* Illegal */

char city1[8]="KOLKATA";
char city2[8];
city2=city1;  /* Illegal */
```

# Strings: Reading/Writing



```
char name[20];
scanf("%s",name); /* Reads a string */
printf("%s",name); /* Prints a string */
gets(name); /* Reads a string */
puts(name); /* Prints a string */
```

scanf("%s",line);
→ cannot be used to read a line containing more than one words

```
char name[20];
scanf("%5s",name);
```

| I | S | I | \0 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |

| K | O | L | K | A | \0 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |

# Strings: Reading/Writing [contd.]



## Reading/Writing a line of text:

```
char line[80];
scanf("%[^\n]s",line);
printf("%s",line);
```

%[. .] → *Edit set conversion code*

```
char line[80];
gets(line);
printf("%s",line);
```

/* or together can be written as: printf("%s",gets(line)); */

/*or puts(line); */

***gets*** → can read characters from the keyboard until a new-line character is encountered; it does not skip whitespaces.

# Strings: Reading/Writing [contd.]



- ***printf*** with format specification

```
printf("%15.6s", city);
```

w → Field width of 15 columns

d → Precision

```
printf("%-15.6s", city);
```

- → left-justified

```
printf("%*.*s", w, d, city);
```

Variable field width

Variable precision

# String Manipulation Examples



```
#include<stdio.h>
//This is a program for copying string1 to string2
int main()
{
        char str1[80], str2[80];
        int i;
        printf("Enter string1: ");
        gets(str1);
        for(i=0; str1[i]!='\0';i++)
                str2[i]=str1[i];
        str2[i]='\0';
        printf("\nThe string2 is: %s",str2);
        printf("\nThe number of characters is = %d\n",i);
        return 0;
}
```

**Output:**

```
Enter string1: Indian Statistical Institute, Kolkata

The string2 is: Indian Statistical Institute, Kolkata
The number of characters is = 37
```

# String Manipulation Examples [contd.]



```c
#include<stdio.h>
//Comparing two strings
int main()
{
        char str1[80], str2[80];
        int i=0;
        printf("Enter string1: ");
        gets(str1);
        printf("Enter string2: ");
        gets(str2);
        while(str1[i]==str2[i] && str1[i]!= '\0'&& str2[i]!= '\0')
                i++;
        if(str1[i]=='\0' && str2[i]=='\0')
                printf("\n\nThe strings are EQUAL.\n");
        else
                printf("\n\nThe strings are NOT EQUAL.\n");
        return 0;
}
```

**Output:**

```
Enter string1: Kolkata
Enter string2: Kolkata


The strings are EQUAL.
```

```
Enter string1: Kolkata
Enter string2: KOLKATA


The strings are NOT EQUAL.
```

# String Manipulation Examples [contd.]



```c
#include<stdio.h>
//Concatinating two strings (with a space in between).
int main()
{
        char str1[50], str2[50], str3[100];
        int i,j;
        printf("Enter string1: ");
        gets(str1);
        printf("Enter string2: ");
        gets(str2);
        for(i=0;str1[i]!='\0';i++)
                str3[i]=str1[i];
        str3[i]=' ';
        for(j=0;str2[j]!='\0';j++)
                str3[i+j+1]=str2[j];
        str3[i+j+1]='\0';
        printf("\n\nThe concatenated string is: %s\n",str3);
        return 0;
}
```

**Output:**

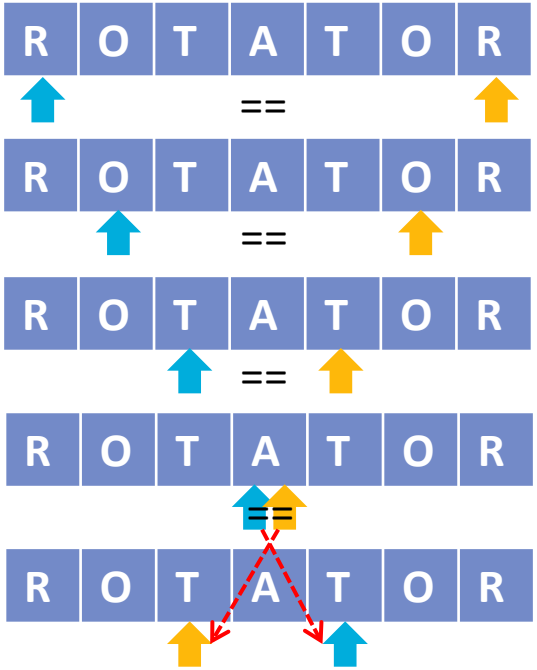
```
Enter string1: ISI
Enter string2: Kolkata


The concatenated string is: ISI Kolkata
```

# String: More Example



- Check whether a text is a palindrome or not

- MOM
- NOON
- LEVEL
- ROTATOR



# Check whether a text is a palindrome or not



```c
#include <stdio.h>
#define MAXLEN 100
main()
{           char text[MAXLEN];
            int i=0,j,len;
            printf("Enter a text: ");
            scanf("%s",text);
            while(text[i++]!='\0'); /* count the length of the text */
            len=i-1; /* length is excluding null character */
            printf("Length of the text is %d\n",len);
            i=0; j=len-1;
            while(text[i]==text[j]) {
                        i++; j--;
                        if(i>j)
                                    break;
            }
            (i>j)? printf("%s is a palindrome.\n",text) : printf("%s is NOT a palindrome.\n",text);
            return 0;
}
```

**Output:**

```
Enter a text: India
Length of the text is 5
India is NOT a palindrome.
```

```
Enter a text: ROTATOR
Length of the text is 7
ROTATOR is a palindrome.
```

```
Enter a text: Rotator
Length of the text is 7
Rotator is NOT a palindrome.
```

# Arithmetic Operations on Characters



- It is possible to perform arithmetic operations on the character constants and variables

```
x= 'a'+1;
```

- Character constants can be used in relational expression

```
ch>='A' && ch<= 'Z'
```

→ tests whether character contained in the variable ***ch*** is a uppercase letter

- Converting character digit to its equivalent integer value:

```
x= ch - '0';
```

- Converting string of digits `str` into integer value `x`:

```
x= atoi(str);
```

**#include<stdlib.h>**

```
strnum="2021";
year= atoi(strnum);
```

# Strings: Library Functions



- Header file is string.h

- Syntax

```
#include <string.h>
```

- Most frequently used library function:

  strcmp (to compare between two strings)
  strcat (to concatenate one string after another)
  strcpy (to copy one string to another)
  ….

# Table of Strings



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| S | H | I | B | P | U | R | | |
| K | H | A | R | A | G | P | U | R |
| S | I | N | G | A | P | O | R | E |
| B | A | R | A | N | A | G | A | R |

```
char location[][10]=
        {
                "SHIBPUR",
                "KHARAGPUR",
                "SINGAPORE",
                "BARANAGAR"
        };
```

Accessing the location names:

location[i-1] → i-th location name

location[0] → SHIBPUR
location[1] → KHARAGPUR
location[2] → SINGAPORE
location[3] → BARANAGAR

# Questions?