Arrays

ICT 4303

Contents

- Array as Abstract Data Type (ADT)
- Sparse Matrix Representation
- Transpose of a Sparse Matrix
- Representation of Multidimensional Arrays
- String ADT
- Pattern Matching

Abstract Data Type (ADT)

- What is Data Type?
  - A collection of *objects* and a set of *operations* that act on those objects.
- Examples?
- Data Abstraction
- An Abstract Data Type is a data type that is organized in such a way that the specification of the objects and of the operations on the objects is separated from the representation of the objects and the implementation of the operations.

Abstract Data Type (ADT)

- Specification
  - Names of every function, argument types, type of its results
  - What the function does
- Representation of Objects and Implementation of Operations

Array Definition

- An array is a collection of elements of the same type placed in contiguous memory locations that can be individually referenced by using an index to a unique identifier.
- Correct?
- Desirable?
- An array is a set of pairs, <*index, value*>, such that each index that is defined has a value associated with it. What type of values?
- Array as ADT: operations that can be performed on an array.

Array as ADT

- Objects:
  - A set of pairs, <*index, value*>, such that each index that is defined has a value associated with it. Index is a finite ordered set of one or more dimensions.
- Functions:
  - *Array* Create (j, A): Create an array ‘A’ of j-dimensions.
  - *Item* Retrieve (A, i): Returns the value at i^{th} position of array ‘A’.
  - *Array* Store (A, i, x): Stores ‘x’ at the i^{th} position of array ‘A’.

Array as ADT

- Let an array A of type ‘t’ has ‘n’ elements.

1. CREATE(A): Create an array A
2. INSERT(A,X): Insert an element X into an array A in any location
3. DELETE(A,X): Delete an element X from an array A
4. MODIFY(A,X,Y): modify element X by Y of an array A
5. TRAVELS(A): Access all elements of an array A
6. MERGE(A,B): Merging elements of A and B into a third array C

Arrays

- An array is a group of related data items that share a common name.
- A particular value in an array is indicated by writing an integer number called index number or subscript in a square brackets after the array name.
- The least value that an index can take in array is 0.

Array Declaration

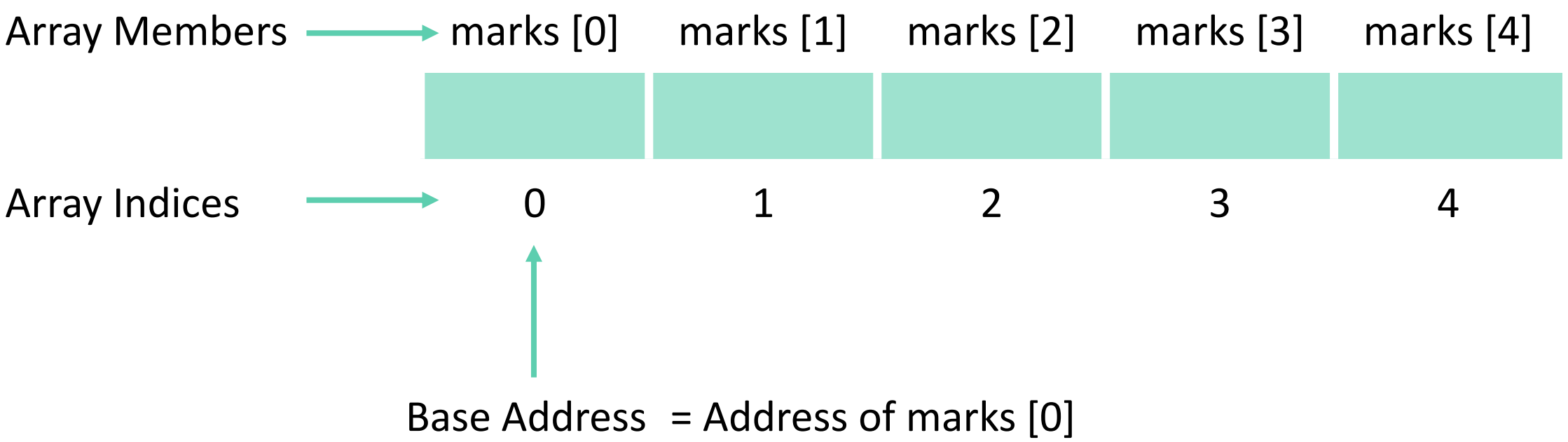
- Declaration (Single Dimensional Array): *DataType* ArrayName [ArraySize];
  - DataType is a valid data type (like int, float...).
  - ArrayName is a valid identifier.
  - ArraySize specifies how many elements the array must contain (>0).
  - ArraySize field is always enclosed in square brackets [ ] and takes static values.
- Example: float marks [5];

Address of marks [3] =?

Array Declaration

- int marks [5];

One Dimensional Array



One Dimensional Array

- A linear list of fixed number of data items of same type.
- These items are accessed using the same name using a single subscript. Example: marks [1], marks [3]
- A list of items can be given one variable name using only one subscript and such a variable is called a single-subscripted variable or a one-dimensional array.

Array Initialization

- Initializing one-dimensional array (compile time)

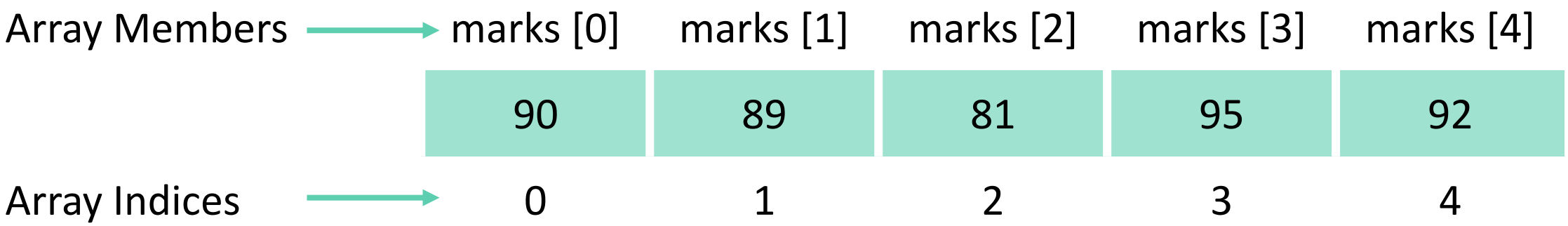
  DataType ArrayName [ArraySize] = {List of values};

- ArraySize : maximum number of elements and may be omitted.
- List of values: values separated by commas.

- Example: int number[3] = {0,1,2};
- This will declare the variable ‘number’ as an array of size 3 and will assign 0 to each element.

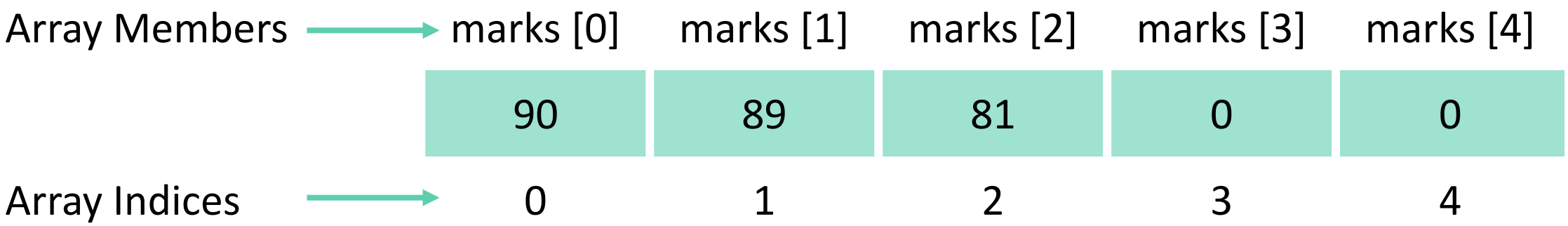
Array Initialization

- int marks [5] = {90, 89, 81, 95, 92};
- int marks [ ] = {90, 89, 81, 95, 92};



Array Initialization

- int marks [5] = {90, 89, 81};



Example: Print Array Elements

```
#include <iostream>
using namespace std;
int main() {
        int marks[5] = {90, 89, 81, 95, 92};
        cout << "The numbers are: ";
        for (i=0; i<5; i++) {
                cout << marks [i] << "  ";
                }
        return 0;
        }
```

Output:

The numbers are: 90 89 81 95 92

Example: Accept User Input and Store in Array

```
int main() {
        int marks[5];
        cout<< “Enter 5 numbers: ”<<endl;
        for (i=0; i<5; ++i) {
                cin >> marks [i] >> "  ";
                }
        cout << "The numbers are: ";
        for (i=0; i<5; i++) {
                cout << marks [i] << "  ";
                }
        return 0;
        }
```

Output:

Enter 5 numbers:

90

89

81

95

92

The numbers are: 90 89 81 95 92

Questions

- Write a CPP program to accept marks of 3 students in 5 subjects. Display the total marks obtained by each student.

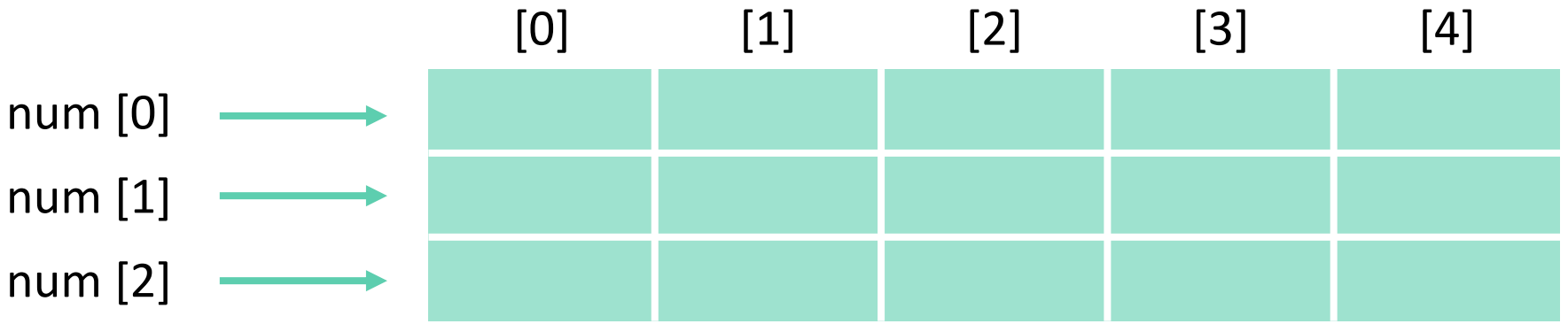
- Declare an array class. Define the member functions: storeNum(), displayNum(). Create an object of this class and call the member functions to store 5 numbers and display the array elements. (Suitable variables, data types and parameters for member functions, etc. should be taken into consideration.)

Two-Dimensional Arrays

- An ordered table of homogeneous elements. **Array-of-arrays**.
- It can be imagined as a two-dimensional table made of elements, all of them of a same uniform data type.
- It is generally referred to as **matrix**, of some rows and some columns.
- It is also called as a **two-subscripted variable**.
- Declaration: *DataType* ArrayName [RowSize] [Column Size];

Two-Dimensional Arrays

int num [3] [5];



Two-Dimensional Arrays: Example

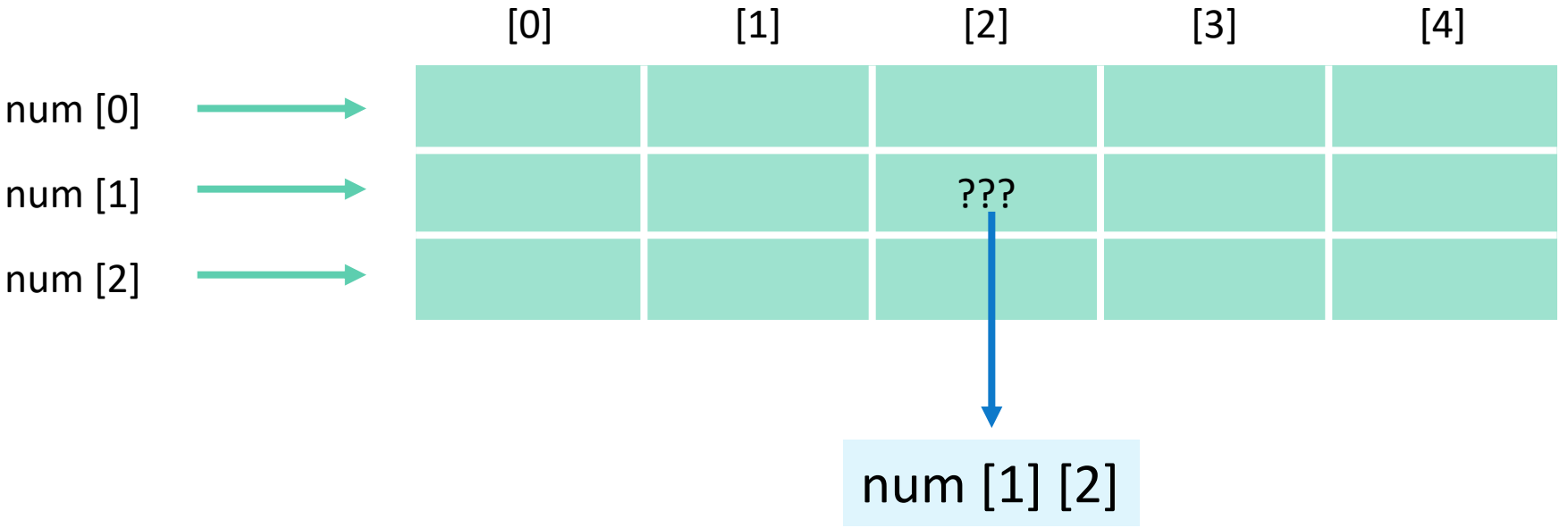
```
int marks [5][3];

float matrix [3][3];

char page [25][80];
```

- The first example tells that marks is a 2-D array of 5 rows and 3 columns.
- The second example tells that matrix is a 2-D array of 3 rows and 3 columns.
- Similarly, the third example tells that page is a 2-D array of 25 rows and 80 columns.

Two-Dimensional Arrays: Example




```
int x = num [1] [2];
cout<< The element in 2nd row 3rd column is: <<“ x” <<endl ;
```

Two-Dimensional Arrays: Example

```
const int WIDTH = 5;
const int HEIGHT = 3;
int Table [HEIGHT][WIDTH];
int n, m;
void main () {
        for (n=0; n<HEIGHT; n++){
                for (m=0; m<WIDTH; m++){
                        Table[n][m]=(n+1)*(m+1);
                        }
                }
}
```

Table

	0	1	2	3	4
0	1	2	3	4	5
1	2	4	6	8	10
2	3	6	9	12	15

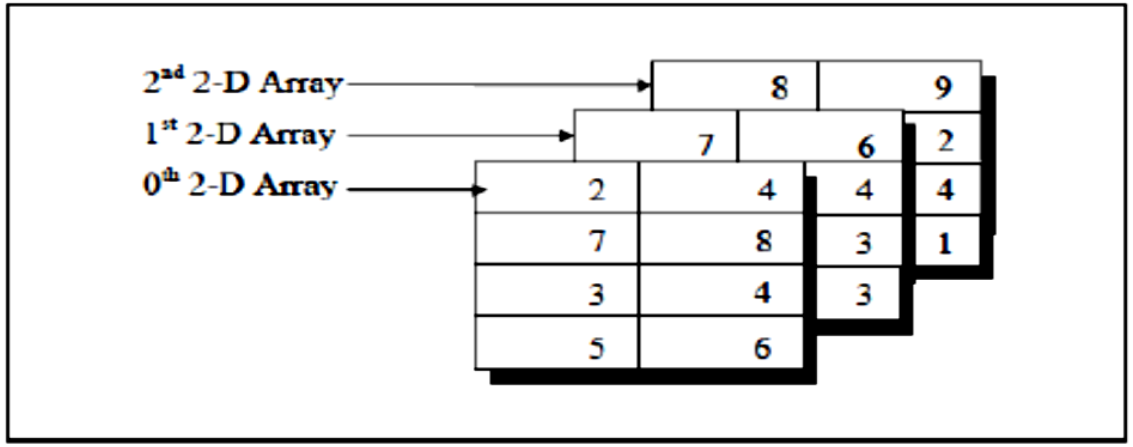
Read a matrix

```
void main(){
        int i, j, m, n, A[10][10];
        cout<<“Enter dimensions for array A: ”;
        cin>>m>>n;
        cout<<“\n Enter elements for A:\n”;
        for (i=0; i<m; i++){
                for (j=0; j<n; j++){
                        cin>> A[i][j];
                }
        }
        for (i=0; i<m; i++){
                for (j=0; j<n; j++){
                        cout<<“\t”<< A[i][j];
                        }
                cout<<“\n”;
                }
}
```

Multi-dimensional Arrays

```
int arr[3][4][2]= {
                    { { 2, 4 }, { 7, 8 }, { 3, 4 }, { 5, 6 } },
                    { { 7, 6 }, { 3, 4 }, { 5, 3 }, { 2, 3 } },
                    { { 8, 9 }, { 7, 2 }, { 3, 4 }, { 5, 1 }, }
                  } ;
```

A three-dimensional array can be thought of as an array of arrays of arrays.



Conceptual View

3-D Array Declaration and Initialization

- Declaration:

  DataType ArrayName [p][RowSize] [Column Size];

  int num [2] [3] [3];

- Initialization:

  DataType ArrayName [p][RowSize] [Column Size]={list of elements};

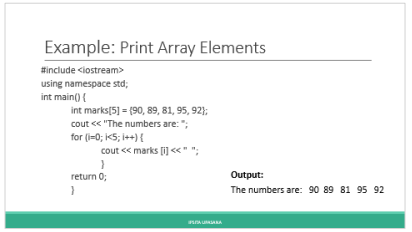
  int num [2] [3] [3] ={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18};

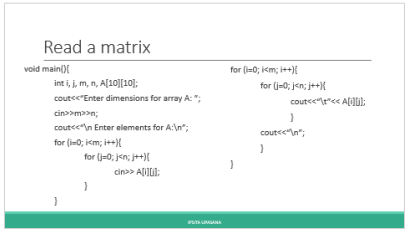
  int num [2] [3] [3] ={ {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}},
  {{10, 11, 12}, {13, 14, 15}, {16, 17, 18}} };

3-D Array Declaration and Initialization

```
int num [2] [3] [3] ={  {{1, 2, 3},        {4, 5, 6},        {7, 8, 9}},
                        {{10, 11, 12},   {13, 14, 15},   {16, 17, 18}}   };
```

i	j	k	Elements
0	0	0	1
0	0	1	2
0	0	2	3
0	1	0	4
0	1	1	5
0	1	2	6
0	2	0	7
0	2	1	8
0	2	2	9
1	0	0	10
1	0	1	11
1	0	2	12
1	1	0	13
1	1	1	14
1	1	2	15
1	2	0	16
1	2	1	17
1	2	2	18



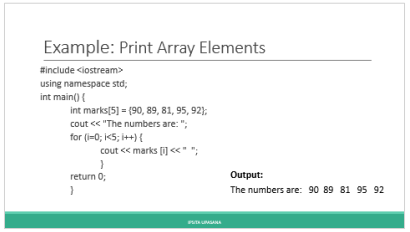


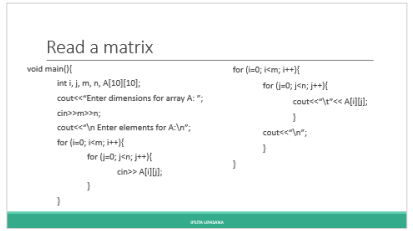
3-D Array Printing

1	2	3
4	5	6
7	8	9

10	11	12
13	14	15
16	17	18

3-D Array Printing






```
for (i=0; i<2; i++) {
        for (j=0; j<3; j++) { //rows
                for (k=0; k<3; k++) {//columns
                        cout<<"\t"<< A[i][j][k];
                }
                cout<<"\n";
        }
        cout<<"\n\n";
}
```

1	2	3	10	11	12
4	5	6	13	14	15
7	8	9	16	17	18

Arrays as Parameters

- Two-dimensional arrays can be passed as parameters to a function.
- How are they are passed?

Question:

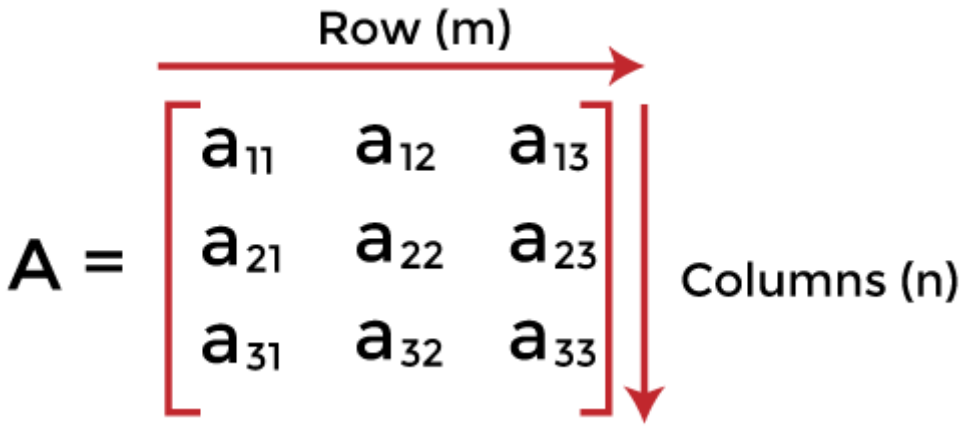
1. Matrix Addition
2. Matrix Multiplication
3. Transpose of a Matrix

Sparse Matrix

- Matrix:
  - A 2-D array of elements arranged in ‘m’ rows and ‘n’ columns.
  - $A_{m \times n}$: A matrix ‘A’ with ‘m’ rows and ‘n’ columns, called as : m x n matrix.
- Sparse Matrix:

  A matrix having majority of the elements equal to zero. The sparse matrix can be defined as the matrix that has a greater number of zero elements than the non-zero elements.

Sparse Matrix



	col 0	col 1	col 2	col 3	col 4	col 5
row 0	15	0	0	22	0	−15
row 1	0	11	3	0	0	0
row 2	0	0	0	−6	0	0
row 3	0	0	0	0	0	0
row 4	91	0	0	0	0	0
row 5	0	0	28	0	0	0

Sparse Matrix

- Representing a Sparse Matrix using traditional representation method wastes space. Any other issues?
- There are two ways to represent the sparse matrix:
- Array Representation
- Linked List Representation

In 2D array representation of sparse matrix, there are three fields used: row, column, and value.

Array Representation of Sparse Matrices

	col 0	col 1	col 2	col 3	col 4	col 5
row 0	15	0	0	22	0	−15
row 1	0	11	3	0	0	0
row 2	0	0	0	−6	0	0
row 3	0	0	0	0	0	0
row 4	91	0	0	0	0	0
row 5	0	0	28	0	0	0

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	−15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	−6
[7]	4	0	91
[8]	5	2	28

Question

	0	1	2	3
0	0	0	1	0
1	3	0	0	0
2	0	4	5	0
3	0	6	0	0

Transpose of a Sparse Matrix

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	−15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	−6
[7]	4	0	91
[8]	5	2	28

Transpose

	row	col	value
b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]	1	1	11
[4]	2	1	3
[5]	2	5	28
[6]	3	0	22
[7]	3	2	−6
[8]	5	0	−15

Transpose of a Sparse Matrix

Assign
A[i][j] to B[j][i]

place element <i, j, value>
in element <j, i, value>

For all columns i

For all elements in column j

Scan the array
"columns" times.
The array has
"elements" elements.

```
void transpose(term a[], term b[])
/* b is set to the transpose of a */
{
   int n,i,j, currentb;
   n = a[0].value;        /* total number of elements */
   b[0].row = a[0].col; /* rows in b = columns in a */
   b[0].col = a[0].row; /* columns in b = rows in a */
   b[0].value = n;
   if (n > 0 )  { /* non zero matrix */
      currentb = 1;
      for (i = 0; i < a[0].col; i++)
      /* transpose by the columns in a */
         for (j = 1; j <= n; j++)
         /* find elements from the current column */
            if (a[j].col == i) {
            /* element is in current column, add it to b */
               b[currentb].row = a[j].col;
               b[currentb].col = a[j].row;
               b[currentb].value = a[j].value;
               currentb++;
            }
   }
}
```

==> O(columns*elements)

Transpose of a Sparse Matrix

- Discussion: compared with 2D array representation
  - O(columns * elements) vs O(columns * rows)
  - elements => columns * rows when non-sparse which is equivalent to O(columns2 * rows)
- Problem: Scan the array “column” times
  - In fact, we can transpose a matrix represented as a sequence of triplets in O(columns + elements) time.
- Solution:
  - First, determine the number of elements in each column of the original matrix.
  - Second, determine the starting positions of each row in the transpose matrix.

Fast Transpose of a Sparse Matrix

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	−15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	−6
[7]	4	0	91
[8]	5	2	28

	[0]	[1]	[2]	[3]	[4]	[5]
rowTerms						

	[0]	[1]	[2]	[3]	[4]	[5]
startingPos						

startingPos [0]=1
startingPos [i] = startingPos [i-1] + rowTerms [i-1]

Fast Transpose of a Sparse Matrix

	row	col	value
a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	−15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	−6
[7]	4	0	91
[8]	5	2	28

Transpose

	[0]	[1]	[2]	[3]	[4]	[5]
startingPos	1	3	4	6	8	8

	row	col	value
b [0]			
b [1]			
b [2]			
b [3]			
b [4]			
b [5]			
b [6]			
b [7]			
b [8]			

Fast Transpose of a Sparse Matrix

```
void fast_transpose(term a[], term b[])
{
/* the transpose of a is placed in b */
   int row_terms[MAX_COL], starting_pos[MAX_COL];
   int i,j, num_cols = a[0].col, num_terms = a[0].value;
   b[0].row = num_cols;  b[0].col = a[0].row;
   b[0].value = num_terms;
   if (num_terms > 0) { /* nonzero matrix */
      for (i = 0; i < num_cols; i++)
         row_terms[i] = 0;
      for (i = 1; i <= num_terms; i++)
         row_terms[a[i].col]++;
      starting_pos[0] = 1;
      for (i = 1; i < num_cols; i++)
         starting_pos[i] =
                     starting_pos[i-1] + row_terms[i-1];
      for (i = 1; i <= num_terms; i++) {
         j = starting_pos[a[i].col]++;
         b[j].row = a[i].col;   b[j].col = a[i].row;
         b[j].value = a[i].value;
      }
   }
}
```

Strings

- A **string** is an array of characters.
- Any group of characters (except double quote sign) defined between double quotation marks is a **constant string**.
- Character strings are often used to build meaningful and readable programs.
- The common operations performed on strings are:
    - Reading and writing strings
    - Combining strings together
    - Copying one string to another
    - Comparing strings to another
    - Extracting a portion of a string, and so on.

String ADT

$S = s_0, s_1, \ldots, s_{n-1}$

If n = 0?

- The string ADT values are all sequences of characters up to a specified length.
- Properties
  - The component characters are from the ASCII character set
  - They are comparable in lexicographic order
  - They have a length, from 0 to the specified length.
- Operations on the string ADT include
  - Input
  - Output
  - Initialization and assignment
  - Comparison greater, equal, less
  - Determination of length
  - Concatenation
  - Accessing component characters and substrings

String ADT

```
class String
{
// objects: A finite ordered set of zero or more characters.
public:
   String(char *init, int m);
   //Constructor that initializes *this to string init of length m

   int operator==(String t);
   // if (the string  represented by *this equals t) return 1 (TRUE)
   // else return 0 (FALSE);

   int operator!();
   // if *this is empty then return 1 (TRUE); else return 0 (FALSE);

   int Length();
   // return the number of characters in *this

   String Concat(String t);
   // return a string whose elements are those of *this followed by those of t.

   String Substr(int i, int j);
   // return a string containing j characters of *this at positions i, i+1, ..., i+j-1
   // if these are valid positions of *this; otherwise, return the empty string.

   int Find(String pat);
   // return an index i such that pat matches the substring of *this that begins at position i.
   // Return -1 if pat is either empty or not a substring of *this
};
```

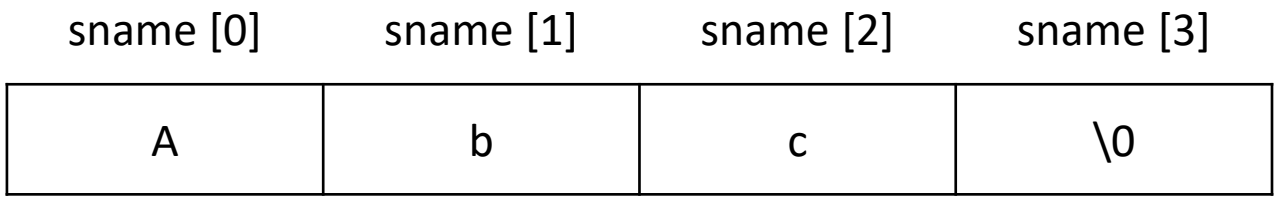
Strings

- Declaration and initialization

  char string_name[size];

  The size determines the number of characters in the string_name.

- char sname[5];
- char sname[] = “Abc”;



Example

```
#include<iostream>
void main(){
        char question[] = “Please enter your first name: ”;
        char greeting [] = “Hello, ”;
        char yourName [80];
        cout<<question;
        cin>>yourName;
        cout<<greeting<<yourName<<“!”;
}
```

Reading Multiple Lines

- Arguments in cin.get() function: cin.get(array_name, size, stop_char)
- The argument stop_char specifies the character that tells the function to stop reading.
- The default value for this argument is the newline (‘\n’) character, but if you call the function with some other character for this argument, the default will be overridden by the specified character.

Example

```
#include <iostream>
using namespace std;
int main(){
        const int len = 80;
        cout << "\nEnter a string:";
        char str[len];
        cin.get(str, len);
        cout<<"\n"<<str;
        return 0;
}
```

String Length

```
#include <iostream>
using namespace std;
int main(){
        char a[30];
        int i, c=0;
        cout<<"Enter a string:";
        cin.get(a,30);
        for(i=0;a[i]!='\0';i++)
                c++;
        cout<<"\nLength of the string '"<<a<<"' is "<<c;
        return 0;
}
```

String Concatenation

```
int main() {
        char str1[55],str2[25];
        int i=0,j=0;
        cout<<"\n Enter First String:";
        cin>>str1;
        cout<<"\n Enter Second String:";
        cin>>str2;
        while(str1[i]!='\0')
                i++;
        while(str2[j]!='\0'){
                str1[i]=str2[j];
                j++;
                i++;
                }
                str1[i]='\0';
        cout<<"\n Concatenated String is\n"<<str1;
        return 0;
}
```

Library functions: String Handling functions(built-in)

- These in-built functions are used to manipulate a given string.
- These functions are part of string.h header file.
  - strlen ()

    Gives the length of the string. eg: strlen(string)
  - strcpy ()

    Copies one string to other. eg: strcpy(Dstr1, Sstr2)
  - strcmp ()

    Compares the two strings. eg: strcmp(str1, str2)
  - strcat ()

    Concatenate the two strings. eg: strcat(str1, str2)

Library function: strlen()

- String length can be obtained by using the following function

```
n=strlen(string);
```

- This function counts and returns the number of characters in a string, where n is an integer variable which receives the value of the length of the string. The argument may be a string constant.

- Copying a String the Hard Way

The best way to understand the true nature of strings is to deal with them character by character.

Copying a String using for loop

- The copying is done one character at a time, in the Statement str2[j] = str1[j];
- The copied version of the string must be terminated with a null.
- However, the string length returned by strlen() **does not include the null**.
- We could copy one additional character, but it's safer to insert the null explicitly. We do this with the line str2[j] = '\0';
- If you don't insert this character, you'll find that the string printed by the program includes all sorts of unwanted characters following the string you want.
- The << just keeps on printing characters, whatever they are, until by chance it encounters a '\0'.

```
#include <iostream>
#include<string.h>
using namespace std;
int main()
{
char str1[ ] = "Manipal Institute of Technology";
char str2[100];
int j;
for(j=0 ; j<strlen(str1); j++)
    str2[j] = str1[j];
str2[j] = '\0';
cout << str1<<"\n"<<str2 << endl;
return 0;
}
```

Library function: strcpy()

- Copying a String the Easy Way

  strcpy(destination, source)

- The strcpy function works almost like a string assignment operator and assigns the contents of source to destination.
- Destination may be a character array variable or a string constant.

  strcpy(city, “DELHI”); will assign the string “DELHI” to the string variable city.

- Similarly, the statement strcpy(city1, city2);

  will assign the contents of the string variable city2 to the string variable city1.

- The size of the array city1 should be large enough to receive the contents of city2.

strcpy() Example

```
#include <iostream>
#include<string.h>
using namespace std;
int main()
{
        char str1[ ] = "Manipal Institute of Technology";
        char str2[100];
        strcpy(str2,str1);
        cout << str1<<"\n"<<str2 << endl;
        return 0;
}
```

Library function: strcmp()

- The **strcmp function** compares two strings identified by the arguments and has a value 0 if they are equal.
- If they are not, it has the numeric difference between the first nonmatching characters in the strings.

  strcmp(string1,string2);

  string1 and string2 may be string variables or string constants.

e.g., **strcmp("their", "there");** will return a value of –9 which is the numeric difference between ASCII "i" and ASCII "r". That is, "i" minus "r" wrt ASCII code is –9.

- If the value is negative, string1 is alphabetically above string2.

strcmp () Example

```
#include <iostream>
#include<string.h>
using namespace std;
int main()
{
        char str1[ ] = "Manipal Institute of Technology";
         char str2[ ] ="Manipal Institute of Technology";
         if(strcmp(str1,str2)!=0)
                cout<<"strings are not equal";
         else
                cout <<"Two strings are equal";
         return 0;
}
```

Library function: strcat()

- The **strcat function** joins two strings together.
- It takes the following form:

strcat (string1, string2);

string1 and string2 are character arrays.

- When the function **strcat** is excuted, string2 is appended to a string1.
- It does so by removing the null character at the end of string1 and placing string2 from there.
- The string at string2 remains unchanged.

strcat() Example

```
#include <iostream>
#include<string.h>
using namespace std;
int main()
{
        char str1[ ] = "Manipal";
        char str2[] =" Institute of Technology";
        strcat(str1,str2);
        cout<<"concatenated string is\n"<<str1;
        return 0;
}
```

Reading integers followed by sentences

```
#include <iostream>
using namespace std;
int main(){
        int n;
        char s1[10],s2[10];
        cout<<"Enter integer";
        cin>>n;
        fflush(stdin);
        gets(s1);
        fflush(stdin);
        gets(s2);
        cout<<"s1="<<s1<<endl;
        cout<<"s2="<<s2<<endl;
        return 0;
}
```

Reading integers followed by multiline input strings

```
#include <iostream>
using namespace std;

int main(){
        int n;
        char s1[10],s2[10];
        cout<<"Enter integer";
        cin>>n;
        fflush(stdin);
        cin.get(s1,10,'$');
        fflush(stdin);
        cin.get(s2,10,'$');
        cout<<"s1="<<s1<<endl;
        cout<<"s2="<<s2<<endl;
        return 0;
}
```

Pattern Matching

- If we have two strings ‘pattern’ and ‘string1’, the easy way to search for ‘pattern’ in ‘string1’ is to use the built-in fuction **strstr.**
- strstr(string1, pattern): returns null pointer if ‘pattern’ is not in ‘string1’.
- How to do it without using built-in function?

Insert Substring

Check the correctness of the code.

IPSITA UPASANA

```
#include <iostream>
#include<string.h>
#include<math.h>
using namespace std;
int main()
{
char a[10], b[4], c[10];
int pos, len_a, len_b, t=0, i=0, p;
int x, tot_size, o;
cout<<"Enter First String:";
cin>>a;
cout<<"Enter Second String:";
cin>>b;
cout<<"Enter the position where the item has to be inserted: ";
cin>>p;
pos=p-1;
len_a=strlen(a);
len_b=strlen(b);
// Copying the input string into another array
while(i <=len_a) {
                c[i]=a[i];
                i++;
              }
c[i]='\0';
tot_size = len_a+len_b;
o = pos+len_b; //making space for string b
// Adding the sub-string
for(i=pos;i<tot_size;i++){
                x = c[i];
                if(t<len_b) {
                            a[i] = b[t];
                            t=t+1;
                            }
                a[o]=x;
                o=o+1;
              }
cout<<a;
return 0;
}
```

Delete Substring

- Take a string and its substring as input.
- Put each word of the input string into the rows of 2-D array.
- Search for the substring in the rows of 2-D array.
- When the substring is found, then override the current row with next row, and so on, up to the last row.

Books

- Ellis Horowitz, Sartaj Sahni, Susan Anderson-Freed, Fundamentals of Data structures in C (2e), Silicon Press, 2008.
- Ellis Horowitz, Sartaj Sahni, Dinesh Mehta, Fundamentals of Data Structures in C++ (2e), Galgotia Publications, 2008.