***Arrays and Strings(cont.)***

DSA I SY (COMP) COEP ASHWINI MATANGE & SHRIDA KALAMKAR

***Pictorial representation***

***Two Dimensional Arrays***

**The table contains a total of 20 values, five in each line.** – **The table can be regarded as a matrix consisting of**

**four rows and five columns.**

**We have seen that an array variable can store a list of values.**

**Many applications require us to store a table of values.**

**C allows us to define such tables of items by using**

**two-dimensional arrays.**

**In 2-dimentional array, elements are arranged in row and column format.**

**Array having more than one subscript variable/index is called multidimensional array.**

**Multidimensional array is also called as matrix.**

**General form:**

**type array\_name [row\_size][column\_size];**

**Examples:**

**int marks[4][5]; // 4 rows 5 column float sales[12][25]; // 12 rows 25 columns double matrix[100][100]; // 100 rows 100 column**

***Declaring 2-D Arrays***

A two-dimensional array **a**, which contains three rows and four columns can be shown as follows −

int a[3][4];

Thus, every element in the array **a** is identified by an element name of the form **a[ i ][ j ]**, where 'a' is the name of the array, and 'i' and 'j' are the subscripts/indices that uniquely identify each element in 'a'.

***Declaring 2-D Arrays***

Similar to that for 1-D array, but use two indices/subscripts. – First indicates row, second indicates column. – Both the indices should be integer constant greater than 0 or expressions which evaluate to integer values.

Examples:

x[7][3] = 0; c[i][k] = a[i][j] \* b[j][k]; a = sqrt (a[j\*3][k]);

***Initializing Elements of a 2-D Array***

***Initialization:***

int stud[2][4] = { 1234, 56, 1212, 33, 1434, 80, 1312, 78 } ;

All values are assigned sequentially and row-wise.

Multidimensional arrays may be initialized by specifying bracketed values for each row.

int stud[4][2] = {

{ 1234, 56 }, { 1212, 33 }, { 1434, 80 }, { 1312, 78 } } ;

***Initializing Elements of a 2-D Array***

***Initializing and Accessing Elements of a 2-D Array***

#include <stdio.h> int main() {

int i, j; int a[3][2] = { 1, 4, 5, 2, 6, 5 }; /\* or you can initialize as int a[3][2] = { { 1, 4 }, { 5, 2 }, { 6, 5 }}; \*/ for (i = 0; i < 3; i++) { for (j = 0; j < 2; j++)

printf("%d ", a[i][j]); printf("\n"); }return 0; }

**Output :** 1 4 5 2 6 5 **Note-** for(i=0;i<row,i++) {

for(j=0;j<col,j++) {

printf("%d",a[i][j]); } }

It is important to remember that while initializing a 2-D array it is necessary to mention the second (column\_size) dimension, whereas the first dimension (row\_size) is optional.

Thus,int arr[2][3] = { 12, 34, 23, 45, 56, 45 } ; int arr[ ][3] = { 12, 34, 23, 45, 56, 45 } ; are perfectly acceptable,

whereas, int arr[2][ ] = { 12, 34, 23, 45, 56, 45 } ; int arr[ ][ ] = { 12, 34, 23, 45, 56, 45 } ; would never work.

***Initializing Elements of a 2-D Array***

An element in a two-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array.

For example − int val = a[2][3];

It will take the element from the 2nd row and 3rd cloumn of the array and assign its value to ‘val’ variable.

***Accessing Elements of a 2-D Array***

***Accessing Elements of a 2-D Array***

#include <stdio.h> int main () {

/\* an array with 5 rows and 2

columns\*/ int a[5][2] = { {0,0}, {1,2}, {2,4},

{3,6},{4,8}}; int i, j; /\* output each array element's value

\*/ for ( i = 0; i < 5; i++ ) { for ( j = 0; j < 2; j++ )

printf("a[%d][%d] : %d\n", i, j, a[i][j] );

}

} return 0; }Output: a[0][0]: 0 a[0][1]: 0 a[1][0]: 1 a[1][1]: 2 a[2][0]: 2 a[2][1]: 4 a[3][0]: 3 a[3][1]: 6 a[4][0]: 4 a[4][1]: 8

***How is a 2-D array is stored in memory?***

**Starting from a given memory location, the elements are stored row-wise in consecutive memory locations.**

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

• **c: number of columns**

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

**a[i][j]** → **is allocated memory location at address**

**x + (i \* c + j) \* k**

***How to read the elements of a 2-D array?***

**By reading them one element at a time for (i=0; i<nrow; i++)**

**for (j=0; j<ncol; j++)**

**scanf (“%f”, &a[i][j]);**

**The ampersand (&) is necessary.**

**The elements can be entered all in one line or in different lines.**

***How to print the elements of a 2-D array?***

**By printing them one element at a time.**

**for (i=0; i<nrow; i++)**

**for (j=0; j<ncol; j++)**

**printf (“\n %f”, a[i][j]);** – **The elements are printed one per line.**

**123.. for (i=0; i<nrow; i++)**

**for (j=0; j<ncol; j++)**

**printf (“%f”, a[i][j]);** – **The elements are all printed on the same line.**

**123456789**

**for (i=0; i<nrow; i++) {**

**for (j=0; j<ncol; j++)**

**printf (“%f ”, a[i][j]); printf (“\n”); }** – **The elements are printed nicely in matrix form.**

1 2 3 4 5 6 7 8 9

***Operations on a 2D Matrix***

Transpose

***Example: Matrix Addition***

Let A and B are two matrices of dimension M X N and S is the sum matrix(S = A + B).

• To add two matrices we have to add their corresponding elements. For example, S[i][j] = A[i][j] + B[i][j].

• Traverse both matrices row wise(first all elements of a row, then jump to next row) using two loops.

• For every element A[i][j], add it with corresponding element B[i][j] and store the result in Sum matrix at S[i][j].

***Example: Matrix Addition***

#include <stdio.h> int main() {

int rows, cols, rowCounter, colCounter; int firstmatrix[50][50], secondMatrix[50][50], sumMatrix[50][50]; printf("Enter Rows and Columns of Matrix\n"); scanf("%d %d", &rows, &cols);

printf("Enter first Matrix of size %dX%d\n", rows, cols); /\* Input first matrix\*/ for(rowCounter = 0; rowCounter < rows; rowCounter++) {

for(colCounter = 0; colCounter < cols; colCounter++) { scanf("%d", &firstmatrix[rowCounter][colCounter]); } }

***Example: Matrix Addition***

/\* Input second matrix\*/

printf("Enter second Matrix of size %dX%d\n", rows, cols); for(rowCounter = 0; rowCounter < rows; rowCounter++) {

for(colCounter = 0; colCounter < cols; colCounter++) { scanf("%d", &secondMatrix[rowCounter][colCounter]); } }/\* adding corresponding elements of both matrices sumMatrix[i][j] = firstmatrix[i][j] + secondMatrix[i][j] \*/

***Example: Matrix Addition***

for(rowCounter = 0; rowCounter < rows; rowCounter++) {

for(colCounter = 0; colCounter < cols; colCounter++) {

sumMatrix[rowCounter][colCounter] = firstmatrix[rowCounter][colCounter]

+ secondMatrix[rowCounter][colCounter]; } }printf("Sum Matrix\n"); for(rowCounter = 0; rowCounter < rows; rowCounter++) {

for(colCounter = 0; colCounter < cols; colCounter++) { printf("%d ", sumMatrix[rowCounter][colCounter]); }printf("\n"); }return 0; }

***Example: Matrix Multiplication***

#include<stdio.h> int main() {

int a[50][50], b[50][50], c[50][50], i, j, k, sum = 0, m, n, o, p; printf("\nEnter the row and column of first matrix"); scanf("%d%d", &m, &n); printf("\nEnter the row and column of second matrix"); scanf("%d %d", &o, &p); if(n != 0) {

printf("Matrix mutiplication is not possible"); printf("\nColumn of first matrix must be same as row of second matrix"); }

***Matrix Multiplication (Cond..)***

else {

//Read the matrix form keyboard(user) printf("\nEnter the First matrix->"); for(i=0;i<m;i++)

for(j=0;j<n;j++)

scanf("%d",&a[i][j]);

printf("\nEnter the Second matrix->"); for(i=0;i<o;i++)

for(j=0;j<p;j++)

scanf("%d",&b[i][j]);

***Matrix Multiplication (Cond..)***

//performing the matrix Multiplication

for(i = 0; i < m; i++) //row of first matrix

{

for(j = 0;j < p; j++) //column of second matrix

{

sum=0; for(k=0; k < n; k++)

sum=sum + a[i][k] \* b[k][j]; c[i][j] = sum; } } }

***Matrix Multiplication (Cond..)***

//Display the matrix Multiplication printf("\nThe multiplication of two matrix is\n"); for(i = 0;i < m; i++) {

printf("\n"); for(j = 0; j < p; j++) { printf("%d\t",c[i][j]); } }return 0; }

***Multi –dimensional Array***

eg: Last 3 months sale each of Mobiles and Laptops of 4 outlets of Croma

Sale[4][3][2]

In C programming, the one-dimensional array of characters are called strings, which is terminated by a null character ‘\0.

Last character is always ‘\0’

because it is the only way the functions that work with a string can know where the string ends.

*The ASCII value of null character('\0') is 0.*

*A string not terminated by a ‘\0’ is not really a string, but merely a collection of characters.*

Note that for individual characters, C uses single quotes, whereas for strings, it uses double quotes.

*For Example:*

*String constant : “CCourse" Character constant: ‘C'*

***String (Character Array)***

Examples:

char address[100]; char welcomeMessage[200]; char name[6];

*Note- C Does not provide support for boundary checking i.e. if we store more characters than the specified size in string declaration then C compiler will not give you any error.*

*Remember, name of an array also specifies the base address of an array.*

Syntax:

***char string\_name[SIZE\_OF\_STRING];***

***String Declaration in C***

char message[6] = {‘H', ‘e', ‘l', ‘l', ‘o', '\0'}; or

char message[] = “Hello"; *Note- In this type of initialization , we don’t need to put terminating null character at the end of string constant. C compiler will automatically inserts a null character('\0'), after the last character of the string literal*.

In C programming language, a string can be initialized at the time of declarations like any other variable in C. If we don't initialize an array then by default it will contain garbage value.

***String Initialization in C***

***String***

#include<stdio.h> int main() {

char stringArray[100]; // string declaration printf("Please write something: \n");

scanf("%s", stringArray); //%s is a format specifier for string

printf("You entered the string %s \n", stringArray); return 0; }

Note- Array name is by default **address of array**, so & not needed for scanf.

char str[]; // Invalid at declaration string size is must

char str[0]; // Invalid string size must not be -ve or zero value

char str[-1]; // Invalid

char str[10]; // Valid

char str[2]={'5','+','A','B'}; // Invalid can not initialized more than size of string

char str[]={'5','+','A','B'}; // Valid In initialization of the string the size is optional

***Few Points on Strings***

***Few Points on Strings***

– Each character of string is stored in consecutive memory location and occupy 1 byte of memory. – If string contains the double quote as part of string then we can use escape character to keep double quote as a part of string.

“Co\"ep" is an example of String – **char string1[] = "first";**

**string1** actually has 6 elements – It is equivalent to

**char string1[] = { 'f', 'i', 'r', 's', 't', '\0' };** – Can access individual characters

**string1[ 3 ] is character ‘s’** – If we don't specify the size of String then length is calculated automatically by the compiler. The length of the string will be one more than the number of characters in string literal/constant.

a character

a string

an empty

string

***Display a String***

void main( ) {

char name[ ] = “COEP" ; int i = 0 ; while ( i <= 3 ) {

printf ( "%c", name[i] ) ; i++ ; } }Note- **printf( )** doesn’t print the ‘\0’.

***Display a String***

void main( ) {

char name[ ] = “COEP" ; int i = 0 ; while ( name[i] != `\0’ ) {

printf ( "%c", name[i] ) ; i++ ; } }

***Input strings***

– Use scanf – char string2[4]; – **For reading individual characters**

**for(i=0;i<3;i++)**

**scanf("%c",&string2[i]); main( ) {**

**char name[10]; int i = 0 ; while ( i<9 ) {**

**scanf ( "%c", &name[i] ) ; i++ ; }name[9]=‘\0’; }**

***Input strings***

– Use scanf – char string2[4];

**scanf( "%s", string2 );**

• Array name is **address of array**, so & not needed for scanf

**main( ) {**

**char name[10]; int i = 0 ; printf(“Enter your name”) scanf ( "%s", name ) ; printf(“%s”, name);**

**}**

***Inputting strings***

While entering the string using **scanf( ) we must be cautious about** two things: 1) The length of the string should not exceed the dimension of the character array. This is because the C compiler doesn’t perform bounds checking on character arrays. Hence, if you carelessly exceed the bounds there is always a danger of overwriting something important.

2) **scanf() is not capable of receiving multi-word strings.** Therefore names such as ‘COEP COLLEGE’ would be unacceptable. The way to get around this limitation is by using the function **gets( ).**

***Execution of scanf ("%s", dept);***

• Whenever encountering a white space, the scanning stops and scanf places the null character at the end of the string.

• e.g., if the user types “MATH 1234 TR 1800,” the string “MATH” along with ‘\0’is stored into dept[].

***strings using gets() and puts()***

**void main( ) {**

**char name[10]; int i = 0 ; printf(“Enter name”) gets(name) ; //scanf(“%[^\n]s”, name);**

**// printf(“hello!\n%s”, name); puts(“hello!”); puts(name); }**

**Enter Name COEP COLLEGE hello! COEP COLLEGE**

***strings using gets() and puts()***

puts( ) can display only one string at a time.

Also, on displaying a string, unlike printf( ), puts( ) places the cursor on the next line.

gets( ) can receive a multi-word string.

gets( ) and puts( ) are available in stdio.h header file

***String Handling Standard Library Function***

C supports a wide range of functions that manipulate null- terminated strings.

The string can not be copied by the assignment operator ‘=’.– e..g, **str = “Test String”** is not valid.

C provides string manipulating functions in the “string.h” header file.

***Some String Functions from string.h***

**Function Purpose Example**

**strcpy() strcpy(s1, s2);**

Copies string s2 into string s1.

strcpy(s1, “Hi”);

**strcat() strcat(s1, s2);**

Appends a string s2 to the end of another string s1.

strcat(s1, “more”);

**strcmp() strcmp(s1, s2);**

Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2. Compares two strings character by character.

strcmp(s1, “Hi”);

***Some String Functions from string.h***

**Function Purpose Example**

**strlen()** Returns the number of characters in

a string.

strlen(“Hi”) returns 2.

**strchr() strchr(s1, ch);**

It searches for the first occurrence of a character ch in the string s1. **strstr() strstr(s1, s2);**

It searches for the first occurrence of a string s2 in another string s1.

Function strcpy copies the second string (source) into the first string (destination).

– strcpy(destination, source) – e.g., strcpy(dest, “test string”);

The null character is appended at the end automatically.

strcpy( ) goes on copying the characters in source string into the target string till it doesn't encounter the end of source string (‘\0’).

If source string is longer than the destination string, the overflow characters may occupy the memory space used by other variables.

to ***copy*** str2 to str1:

char str1[20]; char str2[]=“coep”;

**strcpy(str1, str2);**

**str1 = str2; /\* Will NOT work!! \*/**

***strcpy***

***Example***

#include<string.h> void main( ) {

char source[ ] = "Sayonara" ;

char target[20] ; strcpy ( target, source ) ; printf ( "\nsource string = %s", source ) ; printf ( "\ntarget string = %s", target ) ; }}The output would be...

source string = Sayonara target string = Sayonara

This function concatenates the source string at the end of the target string. (without space)

– **strcat(target,source);**

to ***add*** (*concatenate*) str2 to the end of str1:

– char str1[]=”coep”; – char str2[]=“pune”; – **strcat(str1, str2);**

• returns the value of str1 with str2 added to the end

–Make sure that str1 has room for str2 str1 becomes “coeppune”

***strcat***

***Example***

#include<string.h> void main( ) {

char source[ ] = "Folks!" ; char target[30] = "Hello" ; strcat ( target, source ) ; printf ( "\nsource string = %s", source ) ; printf ( "\ntarget string = %s", target ) ; }The output would be...

source string = Folks! target string = HelloFolks!

Use **strlen(string)**

– returns length of the string

This function counts the number of characters present in a string.

Function strlen is often used to check the length of a string (i.e., the number of characters before the null character).

char dest[6] = “Hello”;

i=strlen(dest);

i=5

***Finding the Length of a String***

***Example***

#include<string.h> void main( )

{

char arr[ ] = “COEP" ; int len1, len2 ; len1 = strlen ( arr ) ; len2 = strlen ( "Humpty Dumpty" ) ; printf ( "\nstring = %s length = %d", arr, len1 ) ; printf ( "\nstring = %s length = %d", "Humpty Dumpty", len2 ) ; }The output would be... string = COEP length = 4 string = Humpty Dumpty length = 13

**strcmp(string1, string2 )**

The comparison between two strings is done by comparing each corresponding character in them.

The two strings are compared character by character until there is a mismatch or end of one of the strings is reached, whichever occurs first.

If the two strings are identical, **strcmp( )** returns a value zero. If they’re not, it returns the numeric difference between the ASCII values of the first non-matching pairs of characters.

– “thrill” < “throw” since ‘i’ < ‘o’; – “joy” < joyous“;

***String Comparison (1/2)***

***String Comparison (2/2)***

**Relationship Returned Value Example** str1 < str2 Negative “Hello”< “Hi” str1 == str2 0 “Hi” = “Hi”

str1 > str2 Positive “Hi” > “Hello”

• e.g., we can check if two strings are the same by char str1[]=“coep”; char str2[]=“pune”;

if(strcmp(str1, str2) != 0)

printf(“The two strings are different!”);

***Example***

#include<string.h>

void main( )

{

char string1[ ] = "Jerry" ; char string2[ ] = "Ferry" ; int i, j, k ; i = strcmp ( string1, "Jerry" ) ; j = strcmp ( string1, string2 ) ; k = strcmp ( string1, "Jerry boy" ) ; printf ( "\n%d %d %d", i, j, k ) ; }The output would be...

0 4 -32

***Example***

#include <stdio.h> #include <string.h> int main () {

char str1[12] = "Hello"; char str2[12] = "World"; char str3[12]; int len ;

/\* copy str1 into str3 \*/ strcpy(str3, str1); printf("strcpy( str3, str1) : %s\n", str3 );

***Example***

/\* concatenates str1 and str2 \*/ strcat( str1, str2); printf("strcat( str1, str2): %s\n", str1 );

/\* total lenghth of str1 after concatenation \*/ len = strlen(str1); printf("strlen(str1) : %d\n", len );

return 0; }strcpy( str3, str1) : Hello strcat( str1, str2): HelloWorld strlen(str1) : 10

***Review***

Output

**printf(“%s”,name);**

**for(i=0;i<4;i++)**

**print("%c",name[i]);** –String Library functions (strcpy, strcat, strlen, strcmp)

Character array

Declaration of Character Array

Input

**for(i=0;i<4;i++)**

**scanf("%c“,&name[i]); scanf(“%s”,name); not read space gets(name); read the space scanf(“%[^\n]s”, name); read the space also**

***what is the o/p of following code?***

void main( ) {

char name[ ] = “College" ; int i = 0 ,n; //n=strlen(name); while ( name[i] != `\0’ ) { //while(i<n)

printf ( "%c", name[i] ) ; i++ ; } }

void main( ) {

char name[25] ; printf ( "Enter your full name " ) ; gets ( name ) ;//scanf(“%[^\n]s”,name); printf( "Hello!" ) ; printf(“%s”,name ) ; }***what is the o/p of following code?***

***What does this program do***

void main() {

char s[20]; int i,j,k,m; printf(“Enter a sentence”); gets(s); printf(“Enter position and no of charactor”); scanf(“%d%d”,&i,&j); for(k=0;s[k]!='\0';k++) { if(k==i-1) {

for(m=k;m<i+j-1;m++) printf("%c",s[m]); } } }

***Same program with little modification***

void main() {

char s[20]; int i,j,k; printf(“Enter a sentence”); gets(s); printf(“Enter position and no of charactor”); scanf(“%d%d”,&i,&j); k=i; while(k<j+i) {

printf("%c",s[k-1]); k++; } }**It made our code simpler**