Unit 4

Arrays, Strings and Pointers

Arrays

Introduction

- Array is a fixed size sequence collection of elements of the same data type.
- Example
  - List of employees in an organization.
- Arrays are a convenient way of grouping a lot of variables under a single variable name.
- Arrays are like pigeon holes or chessboards, with each compartment or square acting as a storage place.
- They can be one dimensional, two dimensional or more dimensional!
- Normally they are represented as

0	1	2	3	4
100	200	300	400	500

Declaration of one-dimensional arrays

- Like any other variable arrays must be declared before they are used.
- The general form of declaration is:
  - type variable-name[size];
    - The type specifies the type of the elements that will be contained in the array, such as int float or char
    - The size indicates the maximum number of elements that can be stored inside the array.
- Example
  - float height[50];
    - Declares the height to be an array containing 50 real elements.
    - Any subscripts 0 to 49 are valid.
- In C the array elements index or subscript begins with number zero.
  - height [0] refers to the first element of the array.
- An individual array element can be used anywhere that a normal variable with a statement such as
  - G = grade [5];

        - ♪ The statement assigns the value stored in the 5th index of the array to the variable g.

- More generally if **i** is declared to be an integer variable, then the statement
    - ✓ g=grades [**i**];
        - ♪ Will take the value contained in the element number I of the grades array to assign it to g.
- A value stored into an element in the array simply by specifying the array element on the left-hand side of the equals sign.
- In the statement
    - ✓ grades [100]=95;
        - ♪ The value 95 is stored into the element number 100 of the grades array.
- In addition to integer constants, integer valued expressions can also be inside the brackets to reference a particular element of the array.
- Example
    - ✓ if low and high were defined as integer variables, then the statement
        - ♪ next_value=sorted_data[(low+high)/2];
            - φ would assign to the variable next_value indexed by evaluating the expression (low+high)/2.
            - φ If low is equal to 1 and high were equal to 9, then the value of sorted_data[5] would be assigned to the next_value.
- Just as variables arrays must also be declared before they are used.
- The declaration of an array involves the type of the element that will be contained in the array such as **int**, **float**, **char** as well as maximum number of elements that will be stored inside the array.
- The C system needs this latter information in order to determine how much memory space to reserve for the particular array.
- The declaration int values[10]; would reserve enough space for an array called values that could hold up to 10 integers.

Storage of arrays

- The amount of storage required to hold an array is directly related to its type and size.
- The total size in bytes for an array is

  - ✓ Total bytes = sizeof(basetype) * length of array

Initialization of Arrays

- Initialize the elements in the array in the same way as the ordinary variables when they are declared.
- The general form of initialization off arrays is:
  - ✓ type array_name[size]={list of values};
- The values in the list are separated by commas, for example the statement
  - ✓ int number[3]={10,10,10};
    - ♪ Will declare the array size as a array of size 3
    - ♪ Will assign ten to each element
- If the number of values in the list is less than the number of elements, then only that many elements are initialized.
- The remaining elements will be set to zero automatically.
- The size of the array may be omitted while declaring.
- The compiler allocates enough space for all initialized elements.
  - ✓ int counter[]={1,1,1,1};
    - ♪ Will declare the array to contain four elements with initial values 1.
  - ✓ This approach works fine as long as we initialize every element in the array.

Operations on one-dimensional arrays

- The two most frequent operations performed are
  - ✓ Searching
  - ✓ Sorting

Sorting

- Process of arranging elements in a list according to their values in ascending or descending order.
- Sorted list is known as ordered list.
- A sorting algorithm is an algorithm that puts elements of a list in a certain order.

Sorting Algorithms

- Bubble sort
- Selection Sort
- Insertion sort
- Shell sort
- Merge sort
- Quick sort
- Radix Sort

Bubble Sort

- *Bubble sort* is a straightforward and simplistic method of sorting data that is used in computer science education.
- The algorithm starts at the beginning of the data set.
  - ✓ It compares the first two elements, and if the first is greater than the second, it swaps them.
  - ✓ It continues doing this for each pair of adjacent elements to the end of the data set.
  - ✓ It then starts again with the first two elements, repeating until no swaps have occurred on the last pass.

Searching

- It is the process of finding the location of the specified element in a list.
- The specified element is often called the search key.
- If the process of searching finds a match of the search key with a list element value, the search is said to be successful.
- Else it is known as an unsuccessful search.
- There are two kinds of search
  - ✓ Linear Search
  - ✓ Binary Search

Linear Search

- It is also known as sequential search.
- It operates by checking every element of a list one at a time in sequence until a match is found.
- Linear search runs in O(n).

Linear Search - Pseudocode

- For each item in the list:
  - Check to see if the item you're looking for matches the item in the list.
  - If it matches.
    - Return the location where you found it (the index).
  - If it does not match.
    - Continue searching until you reach the end of the list.
  - If we get here, we know the item does not exist in the list.
    - Return -1.

Binary Search

- It is an algorithm for locating the position of an element in a sorted list by checking the middle, eliminating half of the list from consideration, and then performing the search on the remaining half.
- If the middle element is equal to the sought value,
  - then the position has been found;
- otherwise, the upper half or lower half is chosen for search based on whether the element is greater than or less than the middle element.
- The method reduces the number of elements needed to be checked by a factor of two each time.

Binary Search - Pseudocode

```
min = 1;
max = N;
repeat
    mid = (min + max) / 2;
    if x > A[mid] then min = mid + 1
    else max = mid - 1;
until (A[mid] = x) or (min > max);
```

Arrays of more than one dimension

- There is no limit, in principle, to the number of indices which an array can have.
- Though there is a limit to the amount of memory available for their storage.

- An array of two dimensions could be declared as follows:
    - data-type array_name[row_size][column_size];

Two dimensional Arrays

- There are two subscripts used in declaring a two-dimensional array.
    - First subscript denotes the row and the second denotes the column.
    - C places each size in its own set of brackets.
- The two-dimensional array is can be seen in the memory as

	Column 0	Column 1	Column 2
	[0][0]	[0][1]	[0][2]
Row 0	**109**	**189**	**789**
	[1][0]	[1][1]	[1][2]
Row 1	**89**	**78**	**65**
	[2][0]	[2][1]	[2][2]
Row 2	**34**	**45**	**56**
	[3][0]	[3][1]	[3][2]
Row 3	**1**	**2**	**3**

- They are stored in a row-column matrix.
    - The left-index indicates row and right indicates column.
    - Right most index changes faster than the leftmost while accessing the elements in the array.
- The number of bytes required to store a two-dimensional array
    - Bytes = size of 1st index * size of 2nd index * sizeof(base type)

Initialization of two-dimensional arrays

- Like the one-dimension arrays, two-dimension arrays may be initialized by following their declaration with a list of initial values enclosed in braces
    - int table[2][3]={0,0,0,1,1,1};
        - Initializes the elements of first row to zero and second row to 1.
        - The initialization is done row by row.

- By surrounding each rows elements in braces
  - ✓ int table[2][3]={{0,0,0},{1,1,1}} ;
- Written in the form of a matrix
  - ✓
    ```
    int table[2][3]={
                         {0,0,0},
                         {1,1,1}
                       };
    ```
- When all the elements are initialized there is no need to specify the size of the first dimension.
  - ✓
    ```
    int table[ ][3]={
                         {0,0,0},
                         {1,1,1}
                    };
    ```
- Missing values will be initialized to 0.
  - ✓ int table[2][3]={{10},{10}};
    - ♪ The first element of each row is initialized to 10 while other elements are automatically initialized to 0.
    - ♪ It can also be written as
      - φ int table[2][3] = {10,10,10};

Multi-dimensional Array

- C allows us to have arrays of more than two dimensions also.
- The exact limit of the dimension is determined by the compiler.
- The general form of a multi-dimensional array is given as
  - ✓ date_type array_name[s_1][s_2][s_3]…..[s_n];

Dynamic Arrays

- An array created at compile time by specifying the size in the source code cannot be modified at run time.
- Dynamic arrays are created at run time using pointer variables and memory management functions malloc, calloc and realloc.

Programming Questions

- Write a program to accept two matrices and find their product.
- Write a program to find the trace and norm of a matrix of order m*n
- Write a C program to read a m*n two-dimensional array and exchange the elements of the first and the last row.
- Write a C program to sort N numbers using bubble sort technique.

Questions

- What is an array? Explain the syntax of declaring a two-dimensional array.

Strings

Introduction

- A string is a sequence of characters.
- Any sequence or set of characters defined within double quotation symbols is a constant string.
- Operations on strings are:
  - Reading string, displaying strings
  - Combining or concatenating strings
  - Copying one string to another.
  - Comparing string & checking whether they are equal
  - Extraction of a portion of a string

Declaring and initializing string variables

- C does not support strings as a data type.
- It allows the representation of strings as a character array.
- The general form of declaration is
  - char string_name[size];
    - The size determines the number of characters in the string.
- Initializing can be done in two ways
  - char month1[8] = {‘j’,’a’,’n’,’u’,’a’,’r’,’y’,’\0’};
  - char month1[8]=“january”;
- Whenever the size for a string is defined it has to be one more than the number of characters stored in the string.
- In the first method we must explicitly send the null terminator to the string.
- The null terminator is a character with value 0 present in ASCII and the Unicode character sets.
- It is often represented as the escape sequence ‘\0’ in the source code.
- C allows us to initialize without specifying the number of elements.
- The size of the array will be determined automatically based on the number of elements initialized.
  - char month1[ ] = {‘j’,’a’,’n’,’u’,’a’,’r’,’y’,’\0’};
  - char month1[ ]=“january”;

  - ✓ char month1[ ]={"january"};

- Whenever the size for a string is defined it has to be one more than the number of characters stored in the string.
- The size of the array can be declared to be much larger than the string size.
  - ✓ char month1[10]="january";
- The compiler
  - ✓ Creates a character array of size 10
  - ✓ Places the value of "january" in it
  - ✓ Terminates with the null character
  - ✓ Initializes all other elements to null
- The array cannot be declared with a smaller size than that of the string.
  - ✓ char month1[5]="january";
    - ♪ Will give a compile time error.
- Initialization and declaration should be in the same instruction.
- Example
  - ✓ char str[10];
  - ✓ str[10]="Hello";
    - ♪ Is not allowed.
- An array name cannot be used as the left-hand side operator of an assignment operator.
  - ✓ char str[10] = "Hello"; char str1[10];
  - ✓ str1 = str; // is not allowed

Reading strings from terminal

- The function ***scanf*** with ***%s*** format specification is needed to read the character string from the terminal.
  - ✓ char address[15];
  - ✓ scanf("%s",address);
- Scanf statement has a draw back
  - ✓ It just terminates the statement as soon as it finds a blank space.
  - ✓ Example

        - If the string ‘new york’ has been typed then only the string ‘new’ will be read and since there is a blank space after word “new” it will terminate the string.

- The scanf can be used without the ampersand symbol before the variable name since a string is being read.
- The field width can be specified using the specifier ***%ws*** for reading a specified number of characters from the string.
    - ***scanf(“%ws”,arr);***
- Possibilities
    - w is equal to or greater than the number of characters typed in
        - The entire string is stored in the string variable.
    - w is less than the number of characters in the string.
        - Excess characters will be truncated and left unread.
- Example
    - char arr[10];
    - scanf(“%4s”,arr);
        - Input is MCA

M	C	A	\0	?	?	?	?	?	?

        - Input is PESIT

P	E	S	I	\0	?	?	?	?	?

Arithmetic Operations on characters

- The characters can be manipulated as numbers are manipulated in C language.
- Whenever the system encounters the character data it is automatically converted into a integer value by the system.
- The integer value depends on the local character set of the system.
- Example
    - x=’a’;
    - printf(“%d\n”,x);
        - Will display 97 on the screen.
- Arithmetic operations can also be performed on characters.

- Example
    - x=’y’-1; is a valid statement.
        - The ASCII value of ‘z’ is 121 the statement the therefore will assign 120 to variable x.
- It is also possible to use character constants in relational expressions.
- Example
    - ch>’a’ && ch < = ’z’
        - will check whether the character stored in variable ch is a lower-case letter.
- A character digit can also be converted into its equivalent integer value.
- Example
    - a=character-‘1’;
        - where a is defined as an integer variable & character contains value 8
        - ASCII value of 8 – 56, ASCII value ‘1’ – 49
        - 56-49=7.
- C library function to converts a string of digits into their equivalent integer values.
    - variable_name = atoi(string)
- String conversion functions are stored in the header file ***stdlib.h.***

Putting strings together

- One string cannot be assigned to another string directly.
- One string cannot be joined with another string using a simple arithmetic addition.
- The process of combing two strings is known as ***concatenation***.

Comparison of two strings

- C does not permit direct comparison of two strings.
- The strings have to be compared character by character.
- The comparison is done until there is a mismatch or one of the strings terminates with a null value.

String handling functions

- C library supports a large number of string handling functions that can be used to array out many of the string manipulations such as:
  - ✓ Length (number of characters in the string).
  - ✓ strlen(string);
  - ✓ Concatenation (adding two are more strings)
  - ✓ Comparing two strings.
  - ✓ Substring (Extract substring from a given string)
  - ✓ Copy(copies one string over another)

String handling functions - strlen

- strlen(string)
  - ✓ Calculates the length of a string
  - ✓ Returns the number of characters in string without counting the terminating null character.
- Example
  - ✓ char string[50] =“This is a string”;
  - ✓ int i =strlen(string);

String handling functions - strcat

- strcat(dest, source)
  - ✓ Appends one string to another.
  - ✓ It appends one copy of the source to the dest.
- The length of the new string is strlen(dest)+strlen(source)
- Example
  - ✓ char str[30] =“This is a string”, str1[10] =“!!!”
  - ✓ strcat(str, str1);

String handling functions – strcmp

- strcmp(str1, str2)
  - ✓ Compares two strings without case sensitivity.
  - ✓ The string comparison starts with first character in each string and continues until the corresponding characters differ or until the end of the strings is reached.

- ✓ It returns one of the three values
  - ♪ < 0 if str1 < str2
  - ♪ ==0 if str1 == str2
  - ♪ > 0 if str1 > str2
- ✓ Example
  - ♪ int I;
  - ♪ I = strcmp("Hello","Hi!!");

String handling functions – strcpy

- strcpy(dest, source)
  - ✓ Copies the source string into destination string
  - ✓ It copies source to dest stopping after the terminating null character has been moved.
- Example
  - ✓ char str[10], str1[10] = "ABCDEF";
  - ✓ strcpy(str, str1);

String handling functions – strncpy

- strncpy(dest, source, maxlen)
  - ✓ Copies at most maxlen characters of source to dest
- Example
  - ✓ char str[10], str1[10]="abcde";
  - ✓ strncpy(str,str1,3);
  - ✓ str[3]='\0';
  - ✓ printf("%s\n",str);

String handling functions – strncmp

- strncmp(str, str1, maxlen)
  - ✓ Compares portions of two strings
  - ✓ Performs an unsigned comparison of str to str1
- Example
  - ✓ char str1[10]="aaab", str2[10]="aabb";
  - ✓ int c;
  - ✓ c=strncmp(str1, str2, 3);

String handling functions – strncat

- strncat(dest, source, maxlen)
  - ✓ Copies at most maxlen characters of source to the end of dest and then appends a null character
  - ✓ The maximum length of the resulting string is strlen(dest) + maxlen
- Example
  - ✓ char str[15] = "Hello, How ", str1[10] = "are you??"
  - ✓ strncat(str, str1,   8);

String handling functions – strstr

- strstr(str, s)
  - ✓ Finds the first occurrence of a substring in another string
  - ✓ On success it returns the position of the element in str where s begins.
  - ✓ On error it returns a null value.
- Example
  - ✓ char str[20] = "Hello how are you?", str1[10] ="how";
  - ✓ int pos;
  - ✓ pos =strstr(str, str1);

Table of strings

- A list of names is considered as a table of strings and a two-dimensional array is used to store the entire list.
- Example
  - ✓ Student[30][15]
    - ♪ It is used to store a list of 30 names each of whose length is not more than 15 characters

Programming Examples

- Write a program to sort an array of n names in ascending order.
- Write a C program to compare two strings without using string functions.
- Write a C program to input a string and print its reverse without using library functions.

Questions

- What is a character array? With an example explain how to input strings from keyboard.

Pointers

Introduction

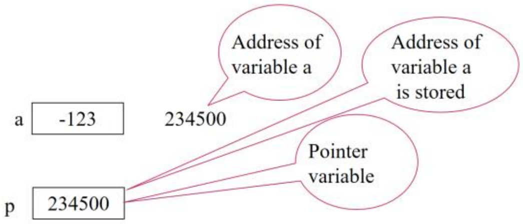
- Pointer is a derived data type in C.
- It is a constant or a variable that contains an address that is used to access the data.
- The address is the location of another object (another variable) in memory.
- Pointer normally **points to** another variable.

Benefits of Pointer

- They are more efficient in handling arrays and data tables.
- They can be used to return multiple values from a function using the function arguments.
- They permit references to functions which allows passing of functions as arguments to other functions.
- The use of pointer arrays to character strings results in saving of data storage space in memory.
- It allows C to support dynamic memory allocation.
- It reduces the length and complexity of a program.
- It provides an efficient tool for manipulating dynamic data structures.
- They fasten the execution speed and reduce the program's execution time.

Pointer Variables

- When the address of a variable is stored in another variable, the second is called pointer variable.
- Syntax
  - ✓ <data_type> *<name of variable>;
    - ♪ Data_type is the base type of the pointer
    - ♪ The pointer thinks that it is pointing to another variable of the same <data_type>
    - ♪ Name of variable is the name given to the pointer



Pointer Constants

- The pointer constant can only be used, they cannot be changed.
- A character constant is a value that is stored in a variable which is named and declared in the program.
- Example
  - ✓ char c=‘A’;
    - ♪ The variable ‘***c***’ has a name and an address.
    - ♪ The name ‘***c***’ has been created by the programmer.
- The address is the relative location of the variable with respect to the program’s memory space.
- The address of the variable is referred to as the pointer constant.
- The address of the variable ‘c’ is drawn from a possible set of address spaces that are available to program and have not stored anything.
- This cannot be changed by the programmer.
- The value may be different each time the program is run.

Pointer Value

- To save the value of a pointer constant it should be identifiable.
- The address operator extracts the address for a variable.
- Example
  - ✓ char a;
  - ✓ printf(“%p”,&a);

Pointer Operators

- &
  - ✓ Address operator
  - ✓ Unary operator

  - ✓ Returns the memory address of its operand.
  - ✓ Internal location of the variable
  - ✓ Example
    - ♪ m = &count;
      - φ m receives the address of count
- ▪ *
  - ✓ Unary Operator
  - ✓ Returns the value located at the address
  - ✓ Example
    - ♪ q=*m;
      - φ q receives the value at address m

Pointer declaration and definition

- ▪ Each variable has to be declared with its type.
- ▪ Since pointers contain addresses that belong to a separate data type they have to be declared as pointers.
- ▪ A pointer irrespective of the data type it is pointing to will occupy eight bytes of memory in a 64 bit machine.
  - ✓ data_type * identifier;
- ▪ The compiler understands that
  - ✓ * tells that identifier is a pointer variable.
  - ✓ Identifier needs a memory location.
  - ✓ Identifier points to a variable of the type data_type.

Initialization of pointer variables

- ▪ When a program starts, uninitialized pointers will have some unknown addresses in them.
  - ✓ They will have an unknown value that is interpreted as memory location.
- ▪ Always assign valid memory address to a pointer.
  - ✓ int a; //variable unknown

  - int *p = &a; //p has a valid address (address of a)
  - *p = 89;// a is assigned the value 89
- A pointer can be set to NULL.
- Null pointer contains address 0.
- Pointer Flexibility
  - The same pointer can be used to point to different data variables in different statements.

Pointer Expressions

- Pointer variables can be used in expressions.
  - Pointer Assignments
  - Pointer Conversions
  - Pointer Arithmetic
- Arithmetic operations can be implemented on the pointers.
- They can also be compared using relational operators.
- Pointers cannot be used in multiplication or division.

Pointer Assignment

- Use a pointer on the right-hand side of an assignment statement to assign its value to another pointer.
- Example
  - int a =10, b=20;
  - int *pa, *pb;
  - pa=&a;
  - pb=pa;
- To print the address - %p in printf

Pointer Conversion

- One type of pointer can be converted to another type.
  - Void to others
    - Void pointers can be assigned to any other type of pointer.
    - Any other type of pointer can be assigned to void pointers
    - Example

- φ void *p; int *ip;
- φ p=ip;
- φ ip=p;

Pointer Conversion

- Using explicit cast for other types of pointers
- Example
  - ✓ double x=1000.0001, y;
  - ✓ int *ip;
  - ✓ ip=(int *) &x;
  - ✓ y=*ip;
    - ♪ This may result in wrong values.
    - ♪ x=1000.000100 y=879609302.000000
- Convert an integer to pointer and a pointer to an integer.
  - ✓ Use explicit cast to do this.

Pointer Arithmetic

- Arithmetic operations can be implemented on the pointers.
- Only addition and subtraction can be implemented on pointers.
- Example
  - ✓ int a, sum=0,b;
  - ✓ int *p1, *p2;
  - ✓ a = *p1 * *p2;
  - ✓ sum = sum + *p1;
  - ✓ b = 5 - *p2 / *p1;
  - ✓ *p2 = *p2 + 15;
- Example
  - ✓ int *p1;
  - ✓ p1++;
    - ♪ If initially p1 is pointing to address 2000, p1++ will increment by 4 bytes since integer takes four bytes of memory.
    - ♪ The same holds for decrement too.

Pointer Arithmetic - Rules

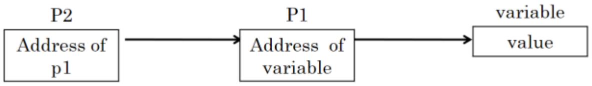
- Each time a pointer is incremented, it points to the memory location of the next element of its base type.
- Each time it is decremented it points to the memory location of the previous element.
- Two pointers cannot be added to each other.
- Only an integer can be added to a pointer.
- One pointer can be subtracted from another pointer.
- Find the number of objects of the base type that separate the two.

Pointer Comparison

- Compare two pointers using relational operators.
- They are useful only when the pointers point to a common object.

Chain of Pointers

- Also called multiple indirection.
- It is possible to make a pointer point to another pointer.
- It creates a chain of pointers.
- Example



Rules to pointer operations

- A pointer variable can be assigned the address of another variable.
- A pointer variable can be assigned the values of another pointer variable.
- A pointer variable can be initialized with NULL or zero value.
- A pointer variable can be prefixed or postfixed with increment or decrement operators.
- An integer value may be added to or subtracted from a pointer variable.
- When two pointers point to the objects of the same data types, they can be compared using the relational operators.
- A pointer variable cannot be multiplied by a constant. Two pointer variables cannot be added.
- A value cannot be assigned to an arbitrary address.
  - ✓ &a = 100 // illegal

Pointers and Arrays

- When an array is declared the compiler allocates a base address and sufficient amount of storage to contain all the elements of the array in contiguous memory locations.
- The base address represents the memory of the first element of the array.
- The compiler defines the array name as the constant pointer pointing to the first element.
- Example
  - ✓ int arr[4]={1,2,3,4};
  - ✓ Base address of arr is 2000.
- The name arr is defined as the constant pointer pointing to the first element arr[0].
  - ✓ Value of arr is 2000.
  - ✓ arr = &arr[0] = 2000
- If parr is declared as an integer pointer
  - ✓ parr = arr;
  - ✓ parr is pointing to the array arr.
- It is equivalent to
  - ✓ parr = &arr[0];
- Every value of arr can be accessed using the pointer parr.
  - ✓ parr = &arr[0]
  - ✓ parr+1 = &arr[1]
  - ✓ parr+2 = &arr[2]
  - ✓ parr+3 = &arr[3]
- **address of arr[index_no] = base address + (index_no * scale factor of data_type)**

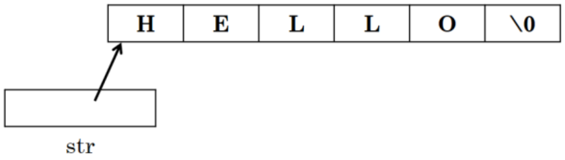
Pointers and two-dimensional arrays

- In one-dimensional array
  - ✓ *(arr+i) or *(parr+i)
    - ♪ Represents the element arr[i].
- In two-dimensional array

  - ✓ *(*(arr+i)+j) or *(*(parr+i)+j)
    - ♪ Represents the element arr[i][j]

Pointers and Strings

- C allows the creation of strings using pointer variables of the type ***char***.
- Example
  - ✓ char str[6] = “Hello”;
  - ✓ char *str = “Hello”;
    - ♪ This creates a string for the literal and then stores its address in the pointer variable ***str***.
- The pointer ***str*** points to the first character of the string “Hello”.



H	E	L	L	O	\0

- Run time assignments can be used to give values to the string pointers.
- Example
  - ✓ char *str1;
  - ✓ str1 = “Hello”;
    - ♪ The value can be printed using
      - φ printf(“%s”,str1);
- Since str1 is the name of the string the indirection operator need not be used.

Array of Pointers

- char name[5][25];
  - ✓ name is a table that contains a maximum of 5 strings of maximum 25 characters in length.
- To have a varying length string
  - ✓ char *name[5];
    - ♪ Declares name to be an array of 5 pointers to characters.
- Example
  - ✓ char *name[5] = {“ABC,”I SEM”,“MCA”, “PESIT”,”BANGALORE”};

A	B	C	/0						
I		S	E	M	/0				
M	C	A	/0						
P	E	S	I	T	/0				
B	A	N	G	A	L	O	R	E	/0

♪ Instead of 125 bytes of memory only 25 bytes of memory is allocated.

- To print out all the 5 names
  - ✓ for(i=0;i<5;i++)
  - ✓ printf("%s\t",name[i]);
- To access the jth in the ith name
  - ✓ *(name[i]+j)
- An array of integer pointers can also be created.
- An array of integer pointers will be nothing but a collection of addresses of integer variables.
- These addresses can be
  - ✓ addresses of isolated variables
  - ✓ addresses of array elements
  - ✓ any other addresses.

Pointers as Function Arguments

- When addresses are passed as arguments to a function, the parameters receiving the addresses should be pointers.
- This is referred to as call by reference.

Functions returning pointers

- Functions can also return a pointer back to the calling program.

Programming Examples

- Write a program to find the sum of five integers stored in an array using pointers.
- Write a C program to find the largest of two numbers using pointers.

Questions

- What is a pointer? How is it declared in a program?
- What is a pointer? Mention the advantages of pointers.
- What is a pointer? With the help of a diagram explain how to access a variable through its pointer.
- Briefly explain pointers and arrays.