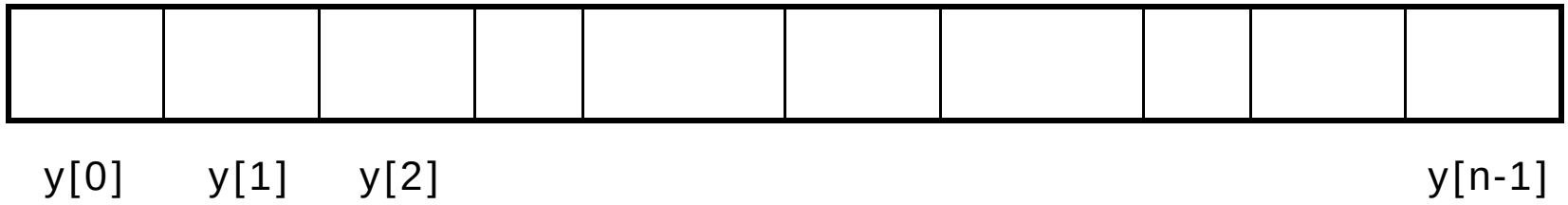
ARRAYS

Arrays

- Suppose, you need to store years of 100 cars. Will you define 100 variables?

```
int y1, y2,…, y100;
```

- An array is an indexed data structure to represent several variables having the same data type: `int y[100];`



Example Case

- You are required to store the following data for the Health Ministry Department of India:
  - Name of states and union territories along with capitals
  - Population of each
  - Literacy rate of each
- Compute and display the state with highest population and literacy rate.

Arrays (cont'd)

- An *element* of an array is accessed using the **array name** and an **index or subscript**,

  for example: `y[5]` which can be used like a variable
- In C, the subscripts always start with 0 and increment by 1, so `y[5]` is the sixth element

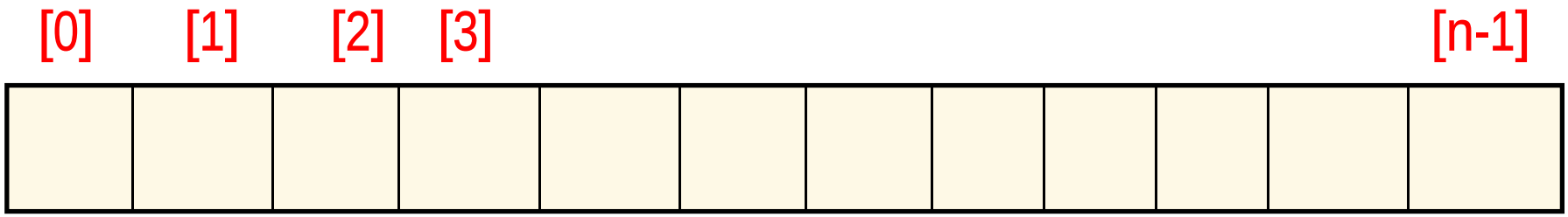
Notion of an array

- Array
  - Homogeneous collection of variables of same type.
  - Group of consecutive memory locations.
  - Linear and indexed data structure.
- To refer to an element, specify
  - Array name
  - Position number (Index)

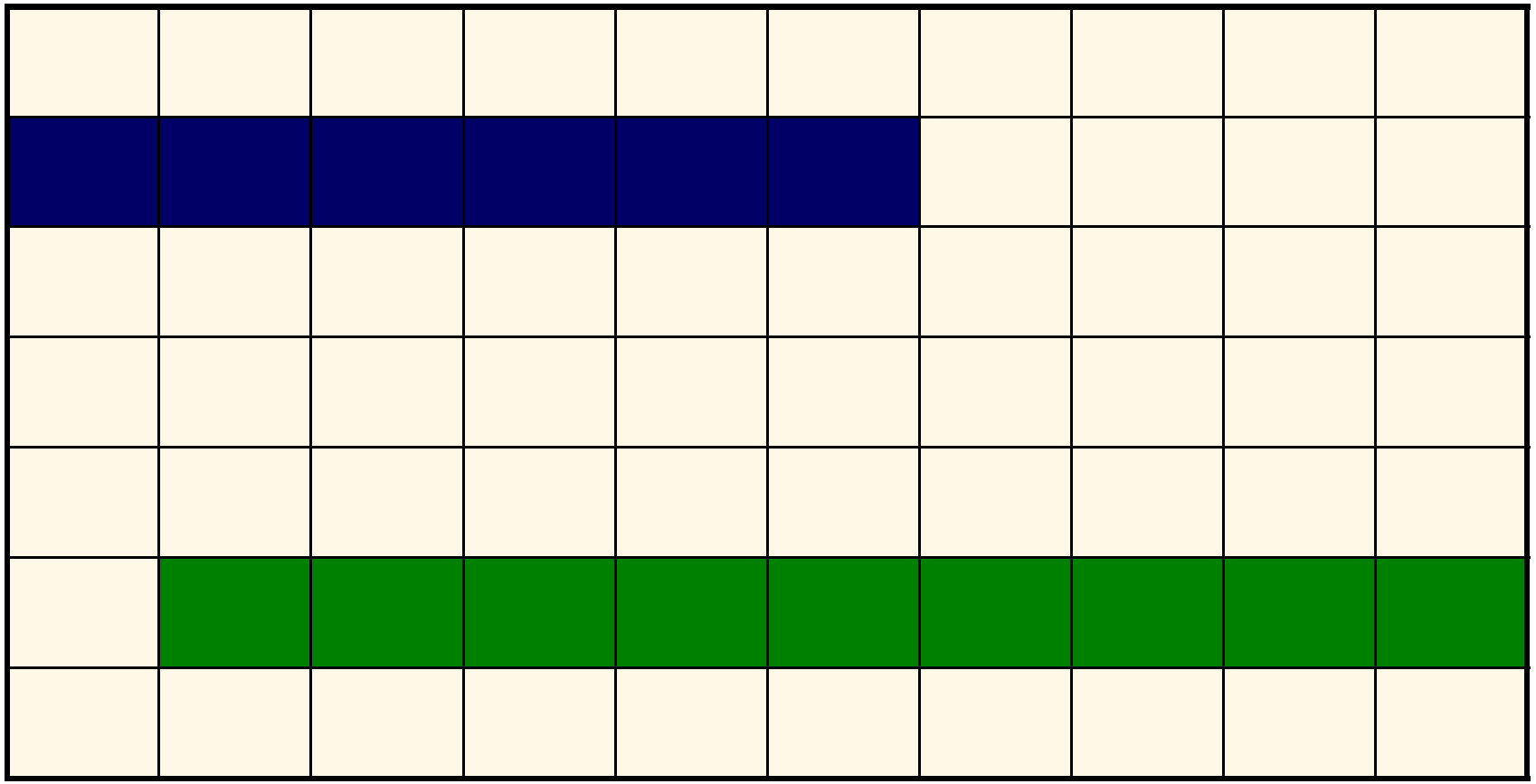
Single Dimension Arrays (1D)

Array Format

- Elements of an array can be integers, floats, characters etc.
- All the elements share a common name with an index called subscript.
- In an array of n elements:



Memory Layout



Declaration

- When declaring arrays, specify
  - Data type of array (integers, floats , characters…..)
  - Name of the array.
  - Size: number of elements

```
array_type   array_name[ size ]   ;
```

  - Example:
    - » int student[10] ;
    - » float my_array [300] ;

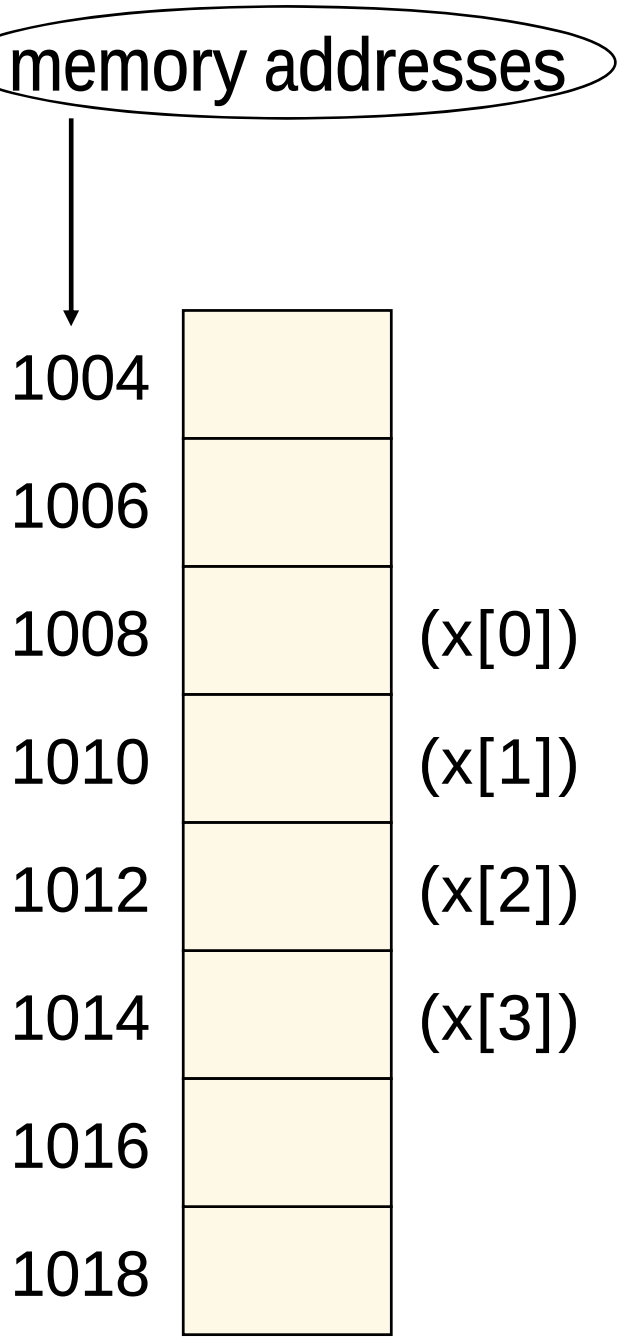
Example

- For example,

```
int   x   [ 4 ] ;
```

Square bracket

- An array of integers of 4 elements.
- Note that the starting memory address is determined by the operating system (just like that of simple variable).
- Contiguous memory locations are allocated.



Examples

- Integer array of 20 elements:

```
int  array_1  [ 20 ];
```

- Character array of 50 elements:

```
char array_2 [ 50 ];
```

- Float array of 100 elements:

```
float array_3 [ 100 ];
```

sizeof()

- The amount of storage required to hold an array is directly related to its data type and the size.

- Example

  Total size in bytes for a 1 D array:

  total bytes = sizeof(data type) * size of array

  int a [7] ;

  total_bytes = 2 * 7

  = 14 bytes in memory

Exercise

```
int main(void) {
float f1[10] ;
char c1[10] ;
printf(“%d”, sizeof(f1));
printf(“%d”, sizeof(c1));
return 0;
}
```

Output:

40

10

Example 1(Method 1: Array initialization using for loop)

Write a program to read 10 integers from the user and display them.

```
int main( void) {
 int  digit[10 ] , t  ;
 printf(“ \n Enter the value of 10 integers “ ) ;
 / * for reading 10 integers from the user using scanf    * /
            for( t = 0, t < 10 ; t + + )
                scanf( “ %d”,  &digit [ t ] );
 /* for displaying the integers using printf */
            for( t = 0, t < 10 ; t + + )
                 printf( “ %d”, digit [ t ] );
return 0;}
```

Example 2

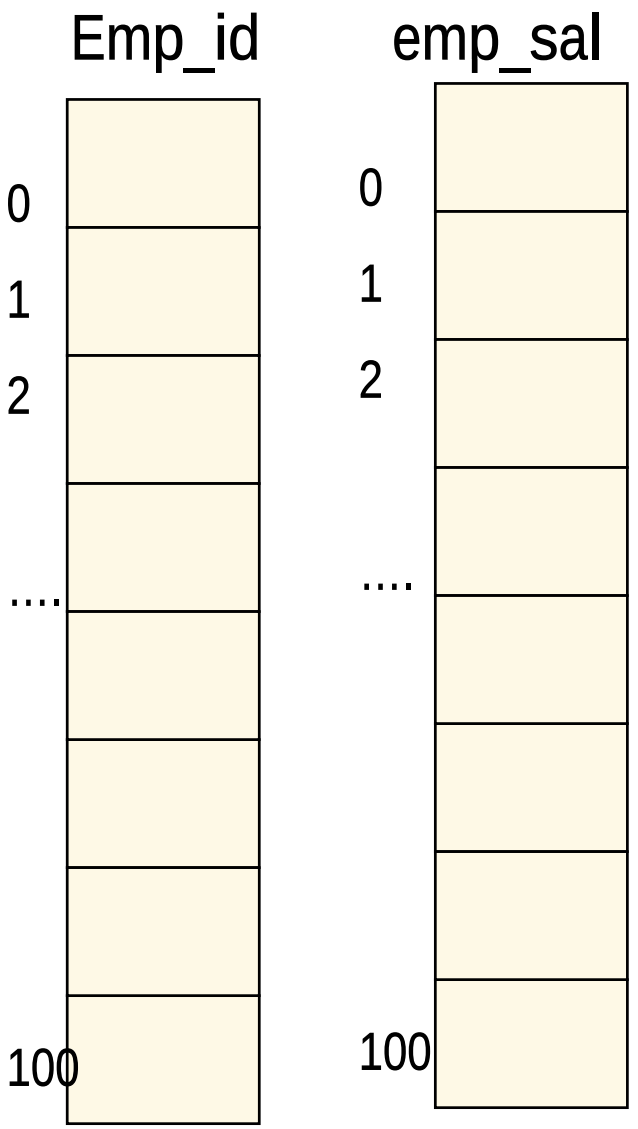
- Write a program to read 10 integers and add 5 to odd elements and 10 to even elements.

Exercise 1

- In an organization 100 employee are there. Write a program to store the employee id and salary of all the employees. Then input the employee id, and display the salary of the employee.

Solution

- Steps
  - Declare two arrays int emp_id[100] and float emp_sal[100]
  - Input the value of both the arrays from the user.
  - Input the emp_id for which salary is to be displayed.
  - Search the above id in array emp_id[ ] and get the index if match found
  - In the second array display the salary for the same index.



Arrays - Why?

```
main()
{
    float sal1,sal2,sal3;
    scanf (“%f”,&sal1);
    scanf (“%f”,&sal2);
    scanf (“%f”,&sal3);
}
```

```
main()
{
        int sal[3], i=0;
        while (i < 3)
        {
           scanf (“%f”,&sal[i];
        }
}
```

Imagine having 100 variables…...

Initializing Integer arrays (method 2)

- Another way of initializing the elements of the array is as follows:

  » int array [ ] = { 2, 20, -30, 10, 100 } ;

•The array size need not be specified explicitly when initial values are included as a part of array declaration.

•Array size will automatically be set equal to the number of initial values specified with in definition.

Initializing Integer arrays (method 3)

- int digits [10] = { 3, 30, 300} ;

- All individual array elements that are not assigned explicit initial values will automatically be set to zero.

- digits [0] = 3 digits [1] = 30 digits [2]= 300

  digits [3] = 0 digits [4]= 0 digits[5]= 0

  digits [7] = 0 digits [8] =0 digits [9] =0

Initializing Arrays (method 4)

Initialize each array element separately

```
id[0]=1234;

id[2]=28;

id[3]=2;

id[4]=888;

id[5]=827;
```

Basics of character array

- If you are required to store a group of character like your name, city , or your college name, or any word or text you need to define a array of characters.

- A char variable can hold a SINGLE character only like

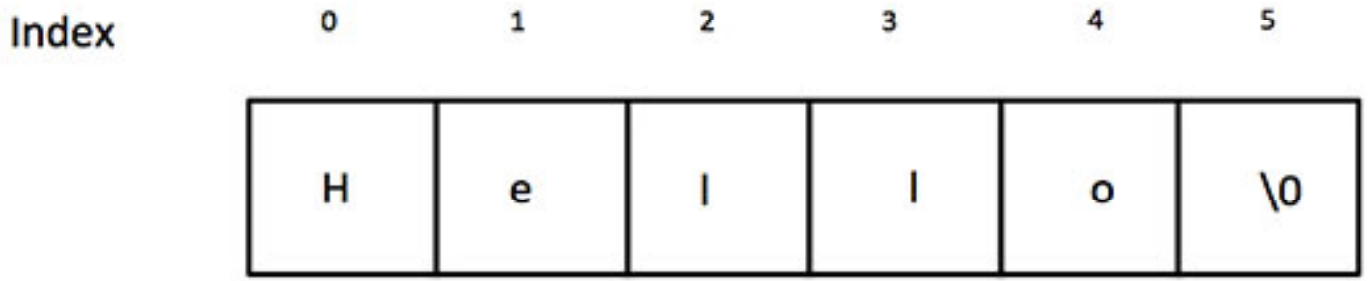
  char c = ‘A’ ;

  char c1=‘B’;

- What if you need to store “Sachin Tendulkar” or “MUMBAI” a string.

Character Arrays

- To hold a single string you need to declare a single dimension character array
  - » char str [11] ;
- When declaring a character array to hold a string(group of characters), one need to declare the array to be one character longer than the largest string that it will hold
- Example above array str[11] will hold 10 characters and a NULL character ('\0') at the end
- Example: char str[6];



Strings: #include<string.h>

- A special kind of array is an array of characters ending in the null character \0 called string arrays

- When declaring a string don’t forget to leave a space for the null character which is also known as the string terminator character

Initializing array with a string (Method 1):

```
char arr[] = {'c','o','d','e','\0'};
```

- In the above declaration/initialization, we have initialized array with a series of character followed by a ‘\0’ (null) byte. The null byte is required as a terminating byte when string is read as a whole.

Initializing array with a string (Method 2):

- char arr[] = "code";

- Here we neither require to explicitly wrap single quotes around each character nor write a null character. This is the advantage of using double quotes.

How To Enter value ?

```
/* program to read user's name and display it */
int main( )
{
char str [10 ] ;
printf( "Enter your name ");
scanf(" %s ", str);
printf( " Your name is : %s", str);
return 0;
}
```

No need to put &

- In case of character arrays, name of the array is the starting address of the array

  char str[10];

  &str [0] => str

Example

```
int main () {
   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
   printf("Greeting message: %s\n", greeting );
   return 0;
}
```

No need to put &

Output:

Greeting message: Hello

Disadvantage of %s

- While reading a string using %s format specifier, it does not scan the string after the space bar.

- For example

  Schin Tendulkar

  It will scan only “ Sachin ”.

  It will IGNORE any space or tab space

How to scan a complete text of words?

gets( )

- gets(argument string) : Collects a string of characters terminated by new line character ‘\n’ from the standard input stream (stdin).

- It allows to input spaces, tabs and copies all the character till new line into the argument string and append a NULL character ‘\0’ at the end of it.

- For example

```
char str [ 20 ] ;
gets( str ) ;
```

puts( )

- puts(argument string):

  It displays the string of characters from argument string to standard output till NULL character and appends a new line character at the end.

- For example

  puts (str) ;

Example

```
void main( )
    {
    char str [10 ] ;
    printf( “Enter your name “);
    scanf(“ %s ”, str);
    printf( “ Your name is : %s”, str);
    }
```

if you typed “New York” output will be “New”?

```
  void main( )
      {
      char name[10];
      printf(“ What is your first and last name?”);
      gets(name); /*gets function waits until the viewer hits the Enter
                    key before it cuts off the input*/
      puts(name); //This prints the viewer’s input on the screen
      }
```

String Handling Functions

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

strcpy(s1, s2);	Copies string s2 into string s1.
strcat(s1, s2);	Concatenates string s2 onto the end of string s1.
strlen(s1);	Returns the length of string s1.
strcmp(s1, s2);	Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.

strcpy(dest, source);

strrev(): to reverse the string

```
void main () {
   char str1[12] = "Hello";
   char str2[12] = "World";
   char str3[12];
   int  len ;
   /* copy str1 into str3 (Be sure that dest is large enough to hold the
     result!)*/
   strcpy(str3, str1);
   printf("strcpy( str3, str1) :  %s\n", str3 );
   /* concatenates str1 and str2 (This function takes the second string and
     appends it to the first string, overwriting the null character at the end
     of the first string with the first character of the second string) */
   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 );     }
```

Output

```
strcpy( str3, str1) :  Hello
strcat( str1, str2):   HelloWorld
strlen(str1) :  10
```

String Comparison

- Frequently, we may wish to compare strings.
- For integers, floating-point numbers, and other such data types, we can use simple comparison operators such as:

```
int x=9, y=27;
if (x < y) { … }
if (y < x) { … }
if (y == x) { … }
```

String Comparison

- If we were to do this with strings…

  char string1[20] = “Cat.”;

  char string2[20] = “Dog.”;

  if (string1 < string2) { … }
- This is a valid comparison; however it does not compare the strings “Cat.” and “Dog.”
- As the name of an array is the memory address of its first element, this comparison statement tests to see if the memory address string1 begins at is less than the memory address string2 begins at.

String Comparison

- Thus, C provides the **strcmp** function to compare strings.
- This function takes two strings and returns an integer value.

  strcmp(s1, s2);

  – A negative integer if str1 is less than str2.
  – Zero if str1 equals str2.
  – A positive integer if str1 is greater than str2.

String Comparison

- Every character (symbol) is actually a number, as defined by the ASCII code.
- As such, these numbers are what is actually compared when characters are being compared
- This is fairly transparent as “A” is less than “B” which is less than “C”, etc…
- However, it is important to realize that “Z” is less than “a”.

String Comparison

- The uppercase letters, in the ASCII code, have lower codes than the lowercase letters.
- Thus, if we were to call:

  strcmp(“Zebra”,“tiger”);
- This function would return a negative value as ‘Z’ is less then ‘t’, thus “Zebra” is less than “tiger”.

String Comparison

- So what would be the result of the following comparison function calls?
  - strcmp(“hello”,“jackson”); -2
  - strcmp(“red balloon”,“red car”); -1
  - strcmp(“blue”,“blue”); 0
  - strcmp(“AARDVARK”,“AARDVARKC”); -67

String Comparison

- So what would be the result of the following comparison function calls?
  - strcmp(“first”,“first”); 0
  - strcmp(“firsta”,“first”); 97
  - strcmp(“first”,“firsta”); -97
  - strcmp(“first”,“second”); -13

```
void main ()
{  char str1[15];
   char str2[15];
   int ret;
   strcpy(str1, "abcdef");
   strcpy(str2, "ABCDEF");
   ret = strcmp(str1, str2);
   if(ret < 0)
   { printf("str1 is less than str2");
   }
   else if(ret > 0)  {
     printf("str2 is less than str1");   }
   else {
      printf("str1 is equal to str2");   }
   }
```

Output: str2 is less than str1

Summary: String

- In C programming, array of characters is called as strings.
- A string is a sequence of characters that is treated as a single data item.
- A string is terminated by null character /0.
- A string is declared as:

  char string1[20];

  – This declares a string named **`string1`** which can hold strings from length 0 to length 19
- All strings must end with a null character.
- When a string is enclosed in double quotes, i.e. “Hello there”, a null character is automatically inserted.
  – The double quotes mean “the string is whatever is inside the quotes followed by a null character.”

WAP to Search an element in Array

```
void main() {
   int a[30], ele, num, i;
   printf("\nEnter no of elements :");
   scanf("%d", &num);
   printf("\nEnter the values :");
   for (i = 0; i < num; i++) {
      scanf("%d", &a[i]);
   }
   //Read the element to be
     searched
   printf("\nEnter the elements to be
     searched :");
   scanf("%d", &ele);
   //Search starts from the zeroth
     location
   i = 0;
   while (i < num && ele != a[i]) {
      i++;
   }
   //If i < num then Match found
   if (i < num) {
      printf("Number found at the
     location = %d", i + 1);
   } else {
      printf("Number not found");
   }
}
```

WAP to add all Elements of the Array

```
int main() {
   int i, arr[50], sum, num;
    printf("\nEnter no of elements :");
   scanf("%d", &num);
    //Reading values into Array
   printf("\nEnter the values :");
   for (i = 0; i < num; i++)
      scanf("%d", &arr[i]);
    //Computation of total
   sum = 0;
   for (i = 0; i < num; i++)
      sum = sum + arr[i];
```

```
//Printing of all elements of array
  for (i = 0; i < num; i++)
     printf("\na[%d]=%d", i, arr[i]);

  //Printing of total
  printf("\nSum=%d", sum);
```