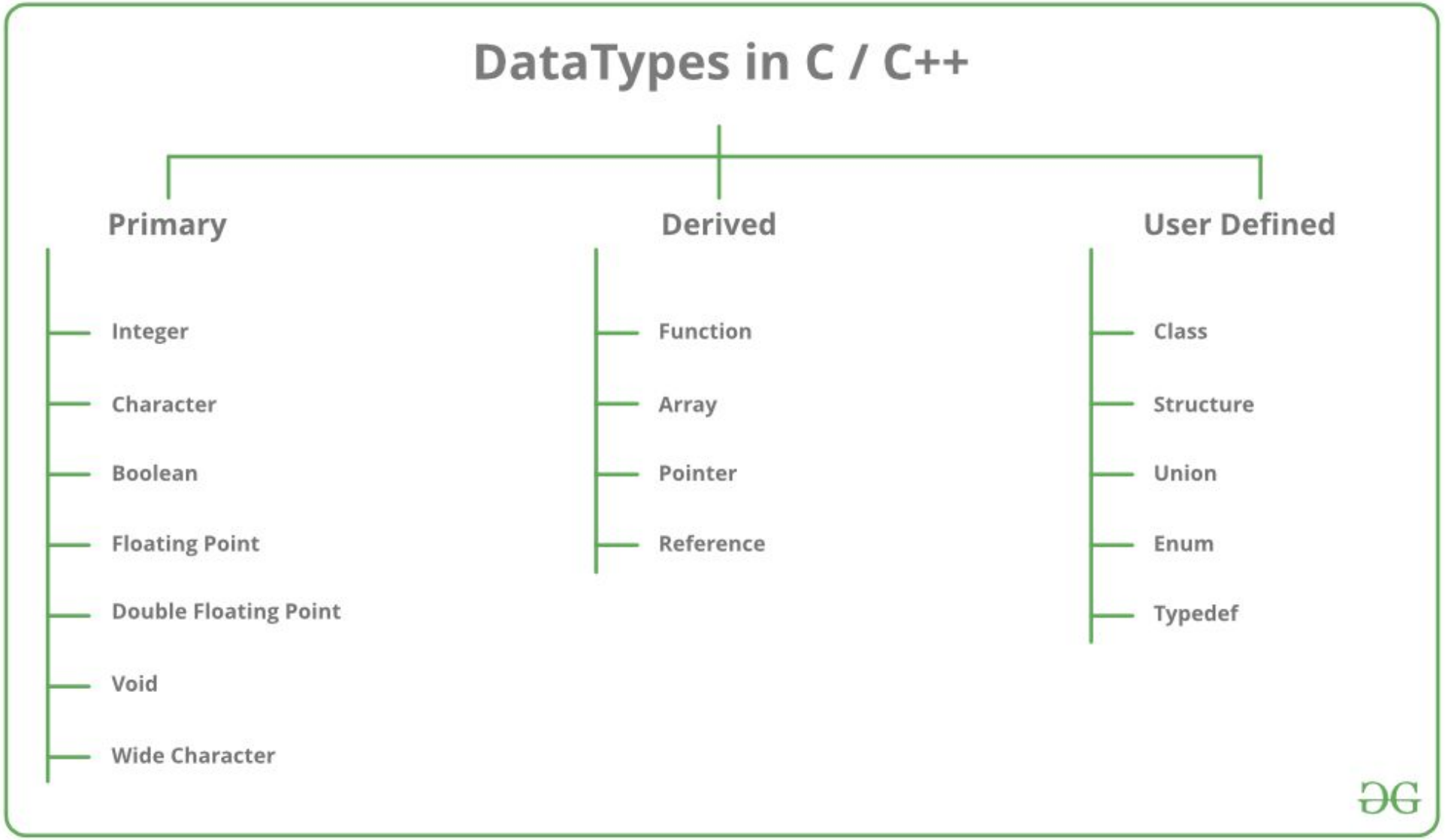
ARRAY & STRINGS

Module 4

Course Code: FEC205 Course Name: C Programming

Academic year-2023-2024 Sem-II

Kirti Suryawanshi.

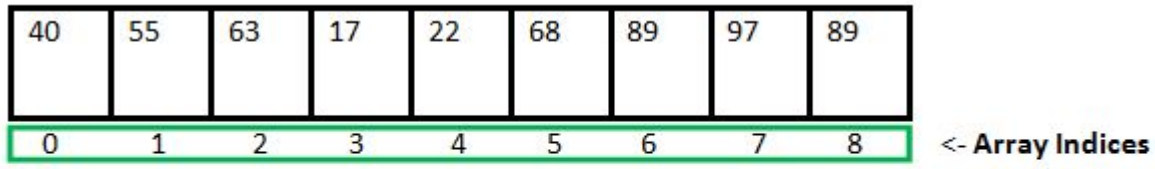

DataTypes in C / C++
Primary
Integer
Character
Boolean
Floating Point
Double Floating Point
Void
Wide Character
Derived
Function
Array
Pointer
Reference
User Defined
Class
Structure
Union
Enum
Typedef

WHAT ARE ARRAYS?

C language provides a capability that **enables the user to design a set of similar data types**, called array.

An array in C/C++ or in any programming language is a collection of similar data items stored at contiguous memory locations and elements can be accessed randomly using indices of an array.

They can be used to store collection of primitive data types such as int, float, double, char, etc of any particular type. To add to it, an array in C/C++ can store derived data types such as the structures, pointers etc.



- However, there are situations in which we would want to store more than one value at a time in a single variable.
- For example, suppose we wish to arrange the percentage marks obtained by 100 students in ascending order. In such a case, we have two options to store these marks in memory:
  - (a) Construct 100 variables to store percentage marks obtained by 100 different students, i.e., each variable containing one student's marks.
  - (b) Construct one variable (called array or subscripted variable) capable of storing or holding all the hundred values.
- An array is a collective name given to a group of 'similar quantities'. These similar quantities could be percentage marks of 100 students, or salaries of 300 employees, or ages of 50 employees.

  What is important is that the quantities must be 'similar'. Each member in the group is referred to by its position in the group.

  For example, assume the following group of numbers, which represent percentage marks obtained by five students.

  per = { 48, 88, 34, 23, 96 }

per = { 48, 88, 34, 23, 96 }

- However, in C, the fourth number is referred as per[3]. This is because, in C, the counting of elements begins with 0 and not with 1. Thus, in this example per[3] refers to 23 and per[4] refers to 96.
- In general, the notation would be per[i], where, i can take a value 0, 1, 2, 3, or 4, depending on the position of the element being referred. **Here per is the subscripted variable (array), whereas i is its subscript.**
- Thus, an array is a collection of similar elements. These similar elements could be all ints, or all floats, or all chars, etc. Usually, the **array of characters is called a 'string',** whereas an array of ints or floats is called simply an array.
- Remember that all elements of any given array must be of the same type, i.e., we cannot have an array of 10 numbers, of which 5 are ints and 5 are floats.

PROGRAM TO FIND AVERAGE MARKS OBTAINED BY A CLASS OF 30 STUDENTS IN A TEST.

```
# include <stdio.h>
int main( )
{
int avg, sum = 0 ;
int i ;
int marks[5] ; // array declaration
for(i=0;i<=5;i++)
{
printf("Enter marks");
scanf("%d",&marks[i]) ; /* store data
in array */
}
```

```
for ( i = 0 ; i <= 5 ; i++ )
sum = sum + marks[ i ] ; /* read data
from an array*/
avg = sum / 6 ;
printf ( "Average marks = %d\n", avg ) ;
return 0 ;
}
```

ARRAY INITIALIZATION

- So far we have used arrays that did not have any values in them to begin with. We managed to store values in them during program execution.
- Let us now see how to initialize an array while declaring it.

Following are a few examples that demonstrate this:

```
int num[ 6 ] = { 2, 4, 12, 5, 45, 5 } ;
int n[ ] = { 2, 4, 12, 5, 45, 5 } ;
float press[ ] = { 12.3, 34.2, -23.4, -11.3 } ;
```

- Note the following points carefully:

(a) Till the array elements are not given any specific values, they are supposed to contain garbage values.

(b) If the array is initialised where it is declared, mentioning the dimension of the array is optional as in the 2nd and 3rd examples above

ARRAY DECLARATION

- To begin with, like other variables, an array needs to be declared so that the compiler will know what kind of an array and how large an array we want. In our program, we have done this with the statement:

```
int marks[ 30 ] ;
```

- Here, **int** specifies the **type of the variable**, just as it does with ordinary variables.
- word **marks** specifies the **name of the variable**.
- The [30] however is new. The number **30 tells how many elements** of the type int will be in our array. This number is often called **the 'dimension' of the array**. The bracket ([ ]) tells the compiler that we are dealing with an array.

ACCESSING ELEMENTS OF AN ARRAY

- This is done with **subscript,** the number in the brackets following the array name. This number specifies the element's position in the array.
- All the array elements are numbered, starting with 0. Thus, marks[2] is not the second element of the array, but the third.
- In our program, we are using the variable i as a subscript to refer to various elements of the array. This variable can take different values and hence can refer to the different elements in the array in turn. This ability to use variables to represent subscripts is what makes arrays so useful.

ENTERING DATA INTO AN ARRAY

- Here is the section of code that places data into an array:

The for loop causes the process of asking for and receiving a student's marks from the user to be repeated 30 times.

- The first time through the loop, i has a value 0, so the scanf( ) function will cause the value typed to be stored in the array element marks[0], the first element of the array.
- This process will be repeated until i becomes 29. This is last time through the loop, which is a good thing, because there is no array element like marks[30].

CONT…

- In scanf( ) function, we have used the “address of” operator (&) on the element marks[i] of the array.
- In so doing, we are passing the address of this particular array element to the scanf( ) function, rather than its value; which is what scanf( ) requires.

READING DATA FROM AN ARRAY

- The balance of the program reads the data back out of the array and uses it to calculate the average.
- The for loop is much the same, but now the body of the loop causes each student's marks to be added to a running total stored in a variable called sum.
- When all the marks have been added up, the result is divided by 30, the number of students, to get the average.

```
for ( i = 0 ; i <= 29 ; i++ )
 sum = sum + marks[ i ] ;
avg = sum / 30 ;
printf ( "Average marks = %d\n", avg ) ;
```

CHANGE VALUE OF ARRAY ELEMENTS

```
int mark[5] = {19, 10, 8, 17, 9}

// make the value of the third element to -1
mark[2] = -1;

// make the value of the fifth element to 0
mark[4] = 0;
```

SOME POINTS ON ARRAY

(a) An array is a collection of similar elements.

(b) The first element in the array is numbered 0, so the last element is 1 less than the size of the array.

(c) An array is also known as a subscripted variable.

(d) Before using an array, its type and dimension must be declared.

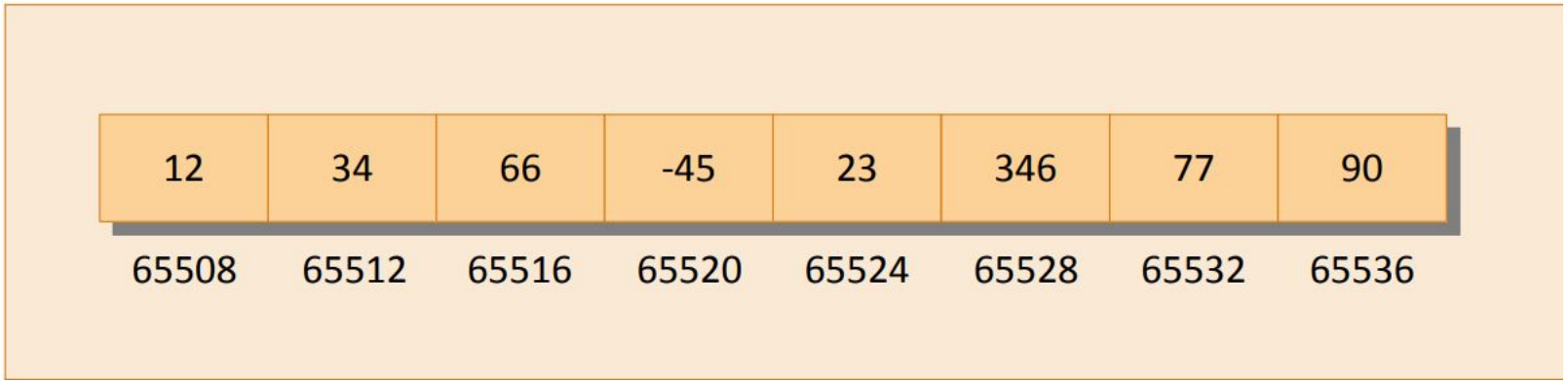
(e) However big an array, its elements are always stored in contiguous memory locations. This is a very important point which we would discuss in more detail later on.

ARRAY ELEMENTS IN MEMORY

- Consider the following array declaration:

  int arr[8] ;

- What happens in memory when we make this declaration? 32 bytes get immediately reserved in memory, 4 bytes each for the 8 integers (under TC/TC++ the array would occupy 16 bytes as each integer would occupy 2 bytes).
- since the array is not being initialized, all eight values present in it would be garbage values.
- Whatever be the initial values, all the array elements would always be present in contiguous memory locations. This arrangement of array elements in memory is shown in Figure



ACCESS ELEMENTS OUT OF ITS BOUND!

- Suppose you declared an array of 10 elements. Let's say,

- int testArray[10];
- You can access the array elements from testArray[0] to testArray[9].

- Now let's say if you try to access testArray[12]. The element is not available. This may cause unexpected output (undefined behavior). Sometimes you might get an error and some other time your program may run correctly.

- Hence, you should never access elements of an array outside of its bound.

PASSING ARRAY ELEMENTS TO A FUNCTION

- Array elements can be passed to a function by calling the function by value, or by reference.
- In the call by value, we pass values of array elements to the function, whereas in the call by reference, we pass addresses of array elements to the function. These two calls are illustrated below.

CALL BY VALUE USING ARRAY

```
# include <stdio.h>
void display ( int ) ;
 int main( )
 {
       int i ;
        int marks[ ] = { 55, 65, 75, 56, 78, 78, 90 } ;
       for ( i = 0 ; i <= 6 ; i++ )
        display ( marks[ i ] ) ;
       return 0 ;
}
void display ( int m ) {
 printf ( "%d ", m ) ;
}
```

55 65 75 56 78 78 90

CALL BY REFERENCE USING ARRAY

```
/* Demonstration of call by reference */
 # include<stdio.h>
 void disp ( int * ) ;
int main( )
{
      int i ;
      int marks[ ] = { 55, 65, 75, 56, 78, 78, 90 } ;
      for ( i = 0 ; i <= 6 ; i++ )
      disp ( &marks[ i ] ) ;
      return 0 ;
}
void disp ( int *i )
{
      printf ( "%d ", *i ) ;
}
```

OUTPUT

55 65 75 56 78 78 90

C PROGRAM TO READ AN ARRAY OF 10 INTEGER AND FIND SUM OF ALL EVEN NUMBERS.

```
#include <stdio.h>
int main()
{
    int a[8],i,e=0,o=0;
    printf("Enter 8 Numbers:\n");
    for(i=0;i<8;i++)
        scanf("%d",&a[i]);
    for(i=0;i<8;i++)
    {
        if(a[i]%2==0)
            e=e+a[i];
        else
            o=o+a[i];
    }
    printf("\nSum of Even Numbers = %d",e);
    printf("\nSum of Odd Numbers  = %d",o);
    return 0;
}
```

SORT THE ARRAY

```c
#include<stdio.h>
int main()
{
    int a[5]={6,12,0,18,11};
    int i,j,swap,min;
    int n=5;
    for(i=0; i<n; i++)
    {
        for(j=i+1 ;j<n; j++)
        {
            if(a[j]<a[i])
            {
                swap = a[i];
                a[i] = a[j];
                a[j] = swap;
            }
        }
    }
    for (i = 0; i < n; i++)
        printf("%d\t", a[i]);
    return 0;
}
```

MATRIX MULTIPLICATION

```c
#include<stdio.h>

#include<stdlib.h>

int main(){

int
a[10][10],b[10][10],mul[10][10],r,c,i,j,k;


system("cls");

printf("enter the number of row=");

scanf("%d",&r);

printf("enter the number of column=");

scanf("%d",&c);

printf("enter the first matrix
element=\n");

for(i=0;i<r;i++)

{

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

{

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

}

}
```

```c
printf("enter the second matrix
element=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
scanf("%d",&b[i][j]);
}
}

printf("multiply of the matrix=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
mul[i][j]=0;
for(k=0;k<c;k++)
{
mul[i][j]+=a[i][k]*b[k][j];
}
}
}
```

```c
//for printing result
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
printf("%d\t",mul[i][j]);
}
printf("\n");
}
return 0;
}
```

TWO-DIMENSIONAL ARRAYS

The two-dimensional array is also called a matrix. Let us see how to create this array and work with it. Here is a sample program that stores roll number and marks obtained by a student side-by-side in a matrix.

```
# include<stdio.h>
 int main( )
 {
    int stud[ 4 ][ 2 ] ;
    int i, j ;
    for ( i = 0 ; i <= 3 ; i++ )
     {
    printf ( "Enter roll no. and marks" ) ;
    scanf ( "%d %d", &stud[ i ][ 0 ], &stud[ i ][ 1 ] ) ;
    }
    for ( i = 0 ; i <= 3 ; i++ )
    printf ( "%d %d\n", stud[ i ][ 0 ], stud[ i ][ 1 ] ) ;
return 0 ;
}
```

CONT..

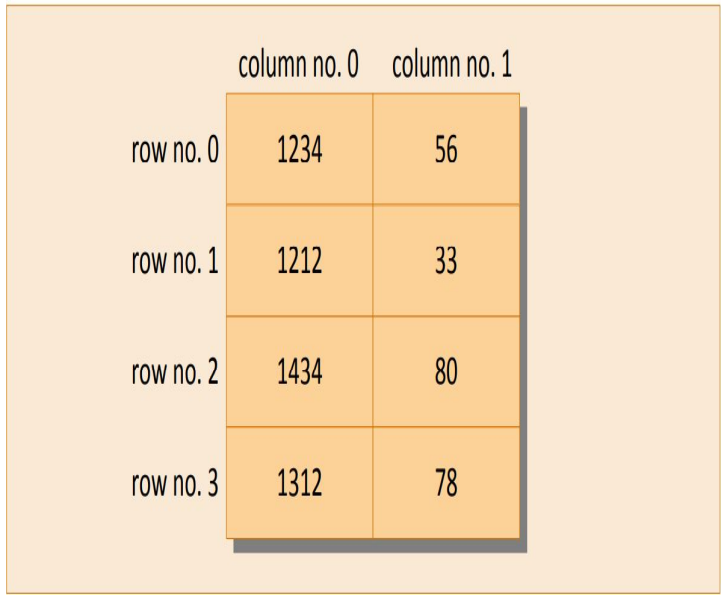
- There are two parts to the program—in the first part, through a for loop, we read in the values of roll no. and marks, whereas, in the second part through another for loop, we print out these values.
- Look at the scanf( ) statement used in the first for loop:

```
scanf ( "%d %d", &stud[ i ][ 0 ], &stud[ i ][ 1 ] );
```

- In stud[i][0] and stud[i][1], the first subscript of the variable stud, is row number which changes for every student. The second subscript tells which of the two columns are we talking about—the zeroth column which contains the roll no. or the first column which contains the marks.
- In our sample program, the array elements have been stored row-wise and accessed row-wise. However, you can access the array elements column-wise as well. Traditionally, the array elements are being stored and accessed row-wise; therefore we would also stick to the same strategy

CONT..

- Remember the counting of rows and columns begin with zero. The complete array arrangement is shown in Figure



	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

- Thus, 1234 is stored in stud[0][0], 56 is stored in stud[0][1] and so on. The above arrangement highlights the fact that a two- dimensional array is nothing but a collection of a number of one- dimensional arrays placed one below the other.

INITIALIZING A MULTIDIMENSIONAL ARRAY

Initialization of a 2d array

```
// Different ways to initialize two-dimensional array
int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};
int c[][3] = {{1, 3, 0}, {-1, 5, 9}};
int c[2][3] = {1, 3, 0, -1, 5, 9};
```

THREE-DIMENSIONAL ARRAY

- We aren't going to show a programming example that uses a three dimensional array. This is because, in practice, one rarely uses this array. However, an example of initializing a three-dimensional array will consolidate your understanding of subscripts.

```
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 },
                            }
                        };
```

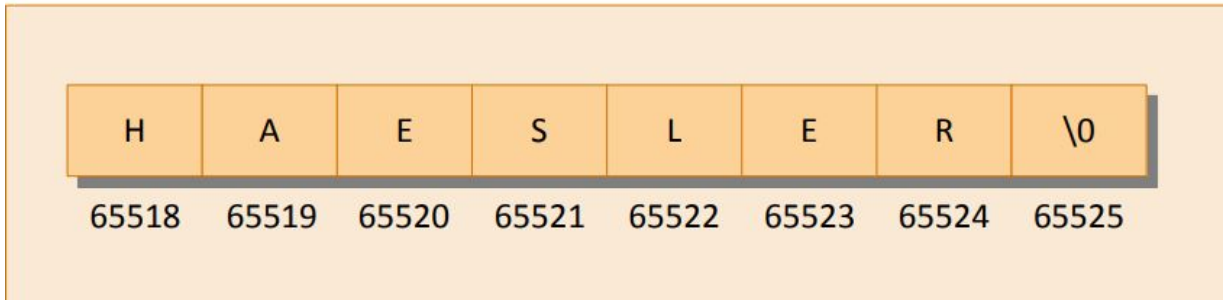
STRING

- The way a group of integers can be stored in an integer array, similarly a group of characters can be stored in a character array. Character arrays are many a time also called strings.
- A string constant is a one-dimensional array of characters terminated by a null (’\0’).

  char c[] = "c string";
- When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \0 at the end by default

CONT..

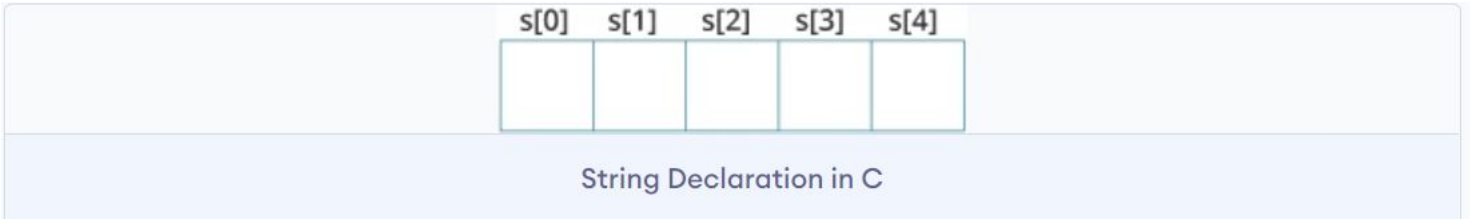
- Each character in the array occupies 1 byte of memory and the last character is always ’\0’.
- What character is this? It looks like two characters, but it is actually only one character, with the \ indicating that what follows it is something special.
- ’\0’ is called null character. Note that ’\0’ and ’0’ are not same. **ASCII value of ’\0’ is 0,** whereas ASCII value of ’0’ is 48.



HOW TO DECLARE A STRING?

- Here's how you can declare strings:

  char s[5];



- Here, we have declared a string of 5 characters.

HOW TO INITIALIZE STRINGS?

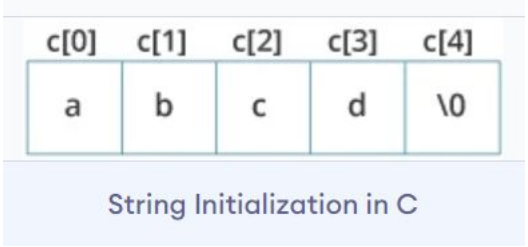
- You can initialize strings in a number of ways.

```
char c[] = "abcd";

char c[50] = "abcd";

char c[] = {'a', 'b', 'c', 'd', '\0'};

char c[5] = {'a', 'b', 'c', 'd', '\0'};
```



- Let's take another example:

```
char c[5] = "abcde";
```

Here, we are trying to assign 6 characters (the last character is '\0') to a char array having 5 characters. This is bad and you should never do this.

ASSIGNING VALUES TO STRINGS

- Arrays and strings are second-class citizens in C; they do not support the assignment operator once it is declared. For example,

```
char c[100];
c = "C programming";  // Error! array type is not assignable.
```

```
char c[] = {'a', 'b', 'c', 'd', '\0'};
```

READING THE STRING FROM USER

- You can use the scanf() function to read a string.
- The scanf() function reads the sequence of characters until encounters whitespace (space, newline, tab, etc.).

```c
#include <stdio.h>
int main()
{
    char name[20];
    printf("Enter name: ");
    scanf("%s",name);
    printf("Your name is %s.", name);
    return 0;
}
```

INPUT:
Entername: robin sharma
OUTPUT:
Your name is robin

CONT..

- Even though robin sharma was entered in the above program, only “robin" was stored in the name string. It's because there was a space after robin.

- Also notice that we have used the **code name** instead of &name with scanf().

```
scanf("%s", name);
```

- This is because name is **a char array, and array names decay to pointers in C.**
- Thus, the name in **scanf() already points to the address of the first element in the string,** which is why we don't need to use &.

HOW TO READ A LINE OF TEXT?

- You can use the fgets() function to read a line of string. And, you can use puts() to display the string.

```c
#include <stdio.h>
int main()
{
    char name[30];
    printf("Enter name: ");
    fgets(name, sizeof(name), stdin);  // read string
    printf("Name: ");
    puts(name);    // display string
    return 0;
}
```

INPUT:
Enter name: robin sharma
OUTPUT:
Name:robin sharma

CONT..

- Here, we have used fgets() function to read a string from the user.

  fgets(name, sizeof(name), stdlin); // read string

- The sizeof(name) results to 30. Hence, we can take a maximum of 30 characters as input which is the size of the name string.
- To print the string, we have used puts(name);.

Note: The gets() function can also be to take input from the user. However, it is removed from the C standard.
It's because gets() allows you to input any length of characters. Hence, there might be a buffer overflow.

PASSING STRINGS TO FUNCTIONS

- Strings can be passed to a function in a similar way as arrays.

```c
#include <stdio.h>
void displayString(char str[]);

int main()
{
    char str[10];
    printf("Enter string: ");
    fgets(str, sizeof(str), stdin);
    displayString(str);     // Passing string to a function.
    return 0;
}
void displayString(char str[])
{
    printf("String Output: ");
    puts(str);
}
```

Function	Use
strlen	Finds length of a string
strlwr	Converts a string to lowercase
strupr	Converts a string to uppercase
strcat	Appends one string at the end of another
strncat	Appends first n characters of a string at the end of another
strcpy	Copies a string into another
strncpy	Copies first n characters of one string into another
strcmp	Compares two strings
strncmp	Compares first n characters of two strings
strcmpi	Compares two strings by ignoring the case
stricmp	Compares two strings without regard to case (identical to strcmpi)
strnicmp	Compares first n characters of two strings without regard to case
strdup	Duplicates a string
strchr	Finds first occurrence of a given character in a string
strrchr	Finds last occurrence of a given character in a string
strstr	Finds first occurrence of a given string in another string
strset	Sets all characters of string to a given character
strnset	Sets first n characters of a string to a given character
strrev	Reverses string

STRLEN()

- The strlen() function calculates the length of a given string.
- The strlen() function takes a string as an argument and returns its length. The returned value is of type size_t (an unsigned integer type).
- It is defined in the <string.h> header file.
- Note that the strlen() function doesn't count the null character \0 while calculating the length

```
#include <stdio.h>
#include <string.h>
int main()
{
    char a[20]="Program";
    char b[20]={'P','r','o','g','r','a','m','\0'};

    // using the %zu format specifier to print size_t
    printf("Length of string a = %zu \n",strlen(a));
    printf("Length of string b = %zu \n",strlen(b));
    return 0;
}
```

OUTPUT:
Length of string a = 7
Length of string b = 7

Note: The correct way to print size_t variables is use of “%zu”. In “%zu” format, z is a length modifier and u stand for unsigned type.

STRCPY()

- The function prototype of strcpy() is:

  char* strcpy(char* destination, const char* source);

- The strcpy() function copies the string pointed by source (including the null character) to the destination.
- The strcpy() function also returns the copied string.
- The strcpy() function is defined in the string.h header file.

EXAMPLE

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

int main() {
  char str1[20] = "C programming";
  char str2[20];

  // copying str1 to str2
  strcpy(str2, str1);

  puts(str2); // C programming

  return 0;
}
```

OUTPUT:
C programming

STRCAT()

- The function definition of strcat() is:

```
char *strcat(char *destination, const char *source)
```

- strcat() arguments

As you can see, the strcat() function takes two arguments:

destination - destination string

source - source string

- **The strcat() function concatenates the destination string and the source string, and the result is stored in the destination string.**

EXAMPLE

```
#include <stdio.h>
#include <string.h>
int main() {
   char str1[100] = "This is " ;
   char  str2[100] =  "Cprogramming ";
   // concatenates str1 and str2
   // the resultant string is stored in str1.
   strcat(str1, str2);
   puts(str1);
   puts(str2);
   return 0;
}
```

OUTPUT:
This is Cprogramming
Cprogramming

When we use strcat(), the size of the destination string should be large enough to store the resultant string. If not, we will get the segmentation fault error.

STRCMP()

- The strcmp() compares two strings character by character. If the strings are equal, the function returns 0.
- The function prototype of strcmp() is:

  int strcmp (const char* str1, const char* str2);

- **Return Value from strcmp()**

Return ValueRemarks

0 if strings are equal

>0 if the first non-matching character in str1 is greater (in ASCII) than that of str2.

<0 if the first non-matching character in str1 is lower (in ASCII) than that of str2.

EXAMPLE

```
#include <stdio.h>
#include <string.h>
int main() {
  char str1[] = "abcd", str2[] = "abCd", str3[] = "abcd";
  int result;
  // comparing strings str1 and str2
  result = strcmp(str1, str2);
  printf("strcmp(str1, str2) = %d\n", result);
  // comparing strings str1 and str3
  result = strcmp(str1, str3);
  printf("strcmp(str1, str3) = %d\n", result);
  return 0;
}
```

OUTPUT:
strcmp(str1, str2) = 1
strcmp(str1, str3) = 0

In the program,

strings str1 and str2 are not equal. Hence, the result is a non-zero integer.
strings str1 and str3 are equal. Hence, the result is 0.

POINTERS (BASIC PROGRAM OF POINTER)

```
# include<stdio.h>
 int main( )
{ int i = 3, *x ;
float j = 1.5, *y ;
char k = 'c', *z ;
printf ( "Value of i = %d\n", i ) ;
printf ( "Value of j = %f\n", j ) ;
printf ( "Value of k = %c\n", k ) ;
x = &i ;
y = &j ;
 z = &k ;
printf ( "Original address in x = %u\n", x ) ;
printf ( "Original address in y = %u\n", y ) ;
printf ( "Original address in z = %u\n", z ) ;
x++ ;
y++ ;
z++ ;
printf ( "New address in x = %u\n", x ) ;
printf ( "New address in y = %u\n", y ) ;
printf ( "New address in z = %u\n", z ) ;
return 0 ; }
```

CONT..

- Output:

Value of i = 3

Value of j = 1.500000

Value of k = c

Original address in x = 65524

Original address in y = 65520

Original address in z = 65519

New address in x = 65528

New address in y = 65524

New address in z = 65520

EXAMPLE OF ARRAY USING POINTER

```
#include <stdio.h>
int main( )
 { int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
int i, *j ;
j = &num[ 0 ] ; /* assign address of zeroth element */
for ( i = 0 ; i <= 5 ; i++ )
{
printf ( "address = %u ", j ) ;
printf ( "element = %d\n", *j ) ;
j++ ; /* increment pointer to point to next location */
}
 return 0 ;
 }
```

The output of this program would be:
address = 65512 element = 24 address = 65516 element = 34 address = 65520 element = 12 address = 65524 element = 44 address = 65528 element = 56 address = 65532 element = 17

ARRAYS OF POINTERS: (TO STRINGS)

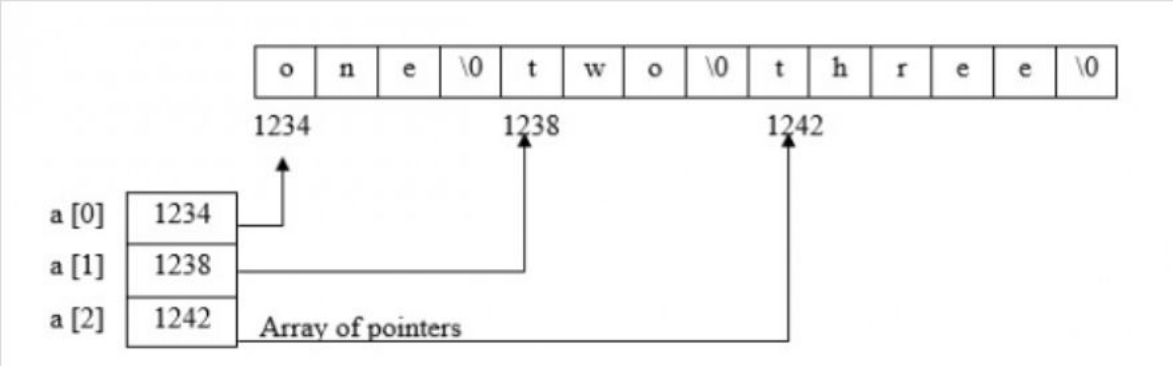
- It is an array whose elements are pointers to the base add of the string.
- It is declared and initialized as follows −

```
char *a[ ] = {"one", "two", "three"};
```

- Here, a[0] is a pointer to the base add of string "one".

  a[1] is a pointer to the base add of string "two".

  a[2] is a pointer to the base add of string "three".



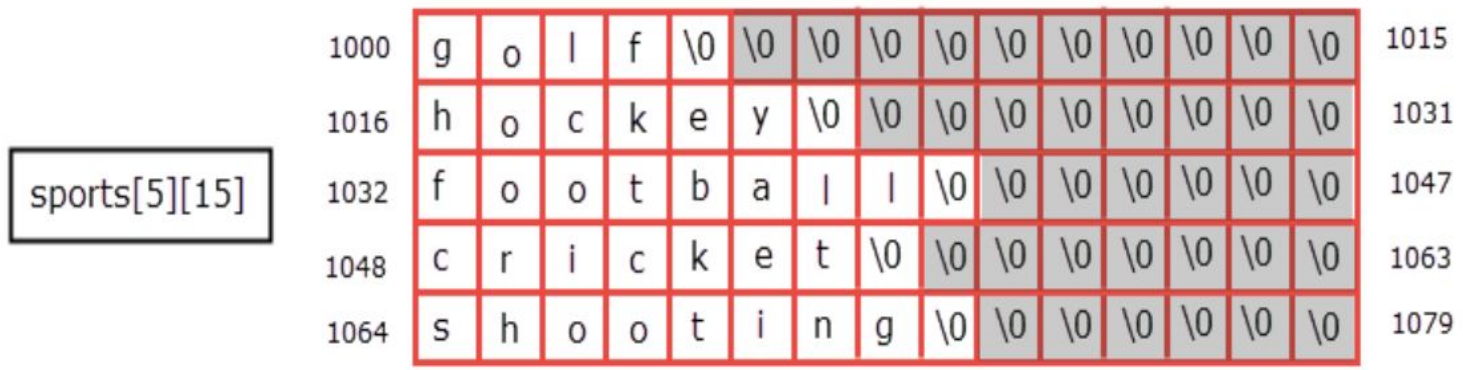
ADVANTAGES

- The advantages of array of pointers are explained below −
- Unlink the two dimensional array of characters, in array of strings and in array of pointers to strings, there is no fixed memory size for storage.
- The strings occupy only as many bytes as required hence, there is no wastage of space.

CONT..

```
char sports[5][15] = {
                        "golf",
                        "hockey",
                        "football",
                        "cricket",
                        "shooting"
                    };
```

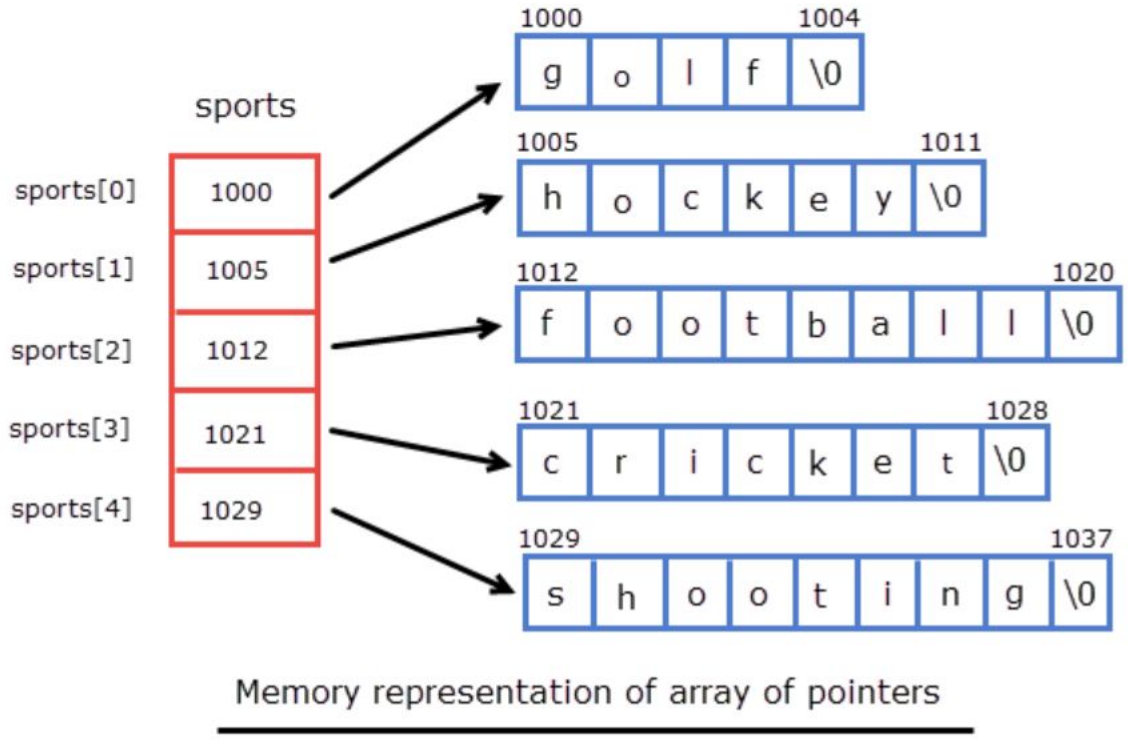
The `sports` array is stored in the memory as follows:



CONT…

```
char *sports[] = {
                    "golf",
                    "hockey",
                    "football",
                    "cricket",
                    "shooting"
                  };
```

Here is how an array of pointers to string is stored in memory.

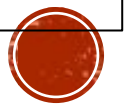


Memory representation of array of pointers

EXAMPLE

```
#include<stdio.h>
#include<string.h>
void main(){
   //Declaring string and pointers, for loop variable//
   int i;
   char *a[5]={"One","Two","Three","Four","Five"};
   //Printing values within each string location using for loop//
   printf("The values in every string location are : \n");
```

```
for(i=0;i<5;i++)
   {
      printf("%s\n",a[i]);
   }
   //Printing addresses within each string location using for loop//
   printf("The address locations of every string values are : \n");
   for(i=0;i<5;i++)
{
      printf("%d\n",a[i]);
   }
}
```



OUTPUT

The values in every string location are :

One

Two

Three

Four

Five

The address locations of every string values are :

-2140237824

-2140237820

-2140237816

-2140237810

-2140237805

EXAMPLE 2

LET'S CONSIDER ANOTHER EXAMPLE.
STATED BELOW IS A C PROGRAM DEMONSTRATING THE CONCEPT OF ARRAY OF POINTERS TO STRING −

```
#include<string.h>
void main(){
   //Declaring string and pointers//
   char string[10]="TutorialPoint";
   char *pointer = string;
   //Printing the string using pointer//
   printf("The string is : ");
   while(*pointer!='\0'){
      printf("%s",*pointer);
      pointer++;
   }
}
```

Output:
The string is: TutorialPoint