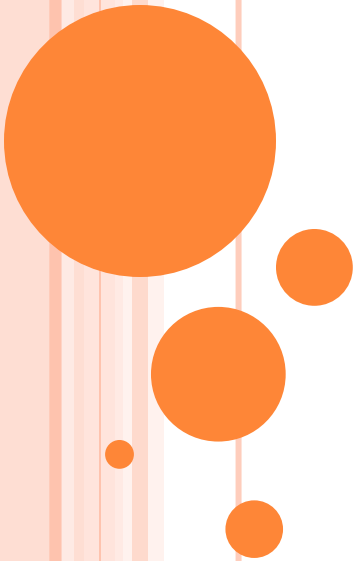


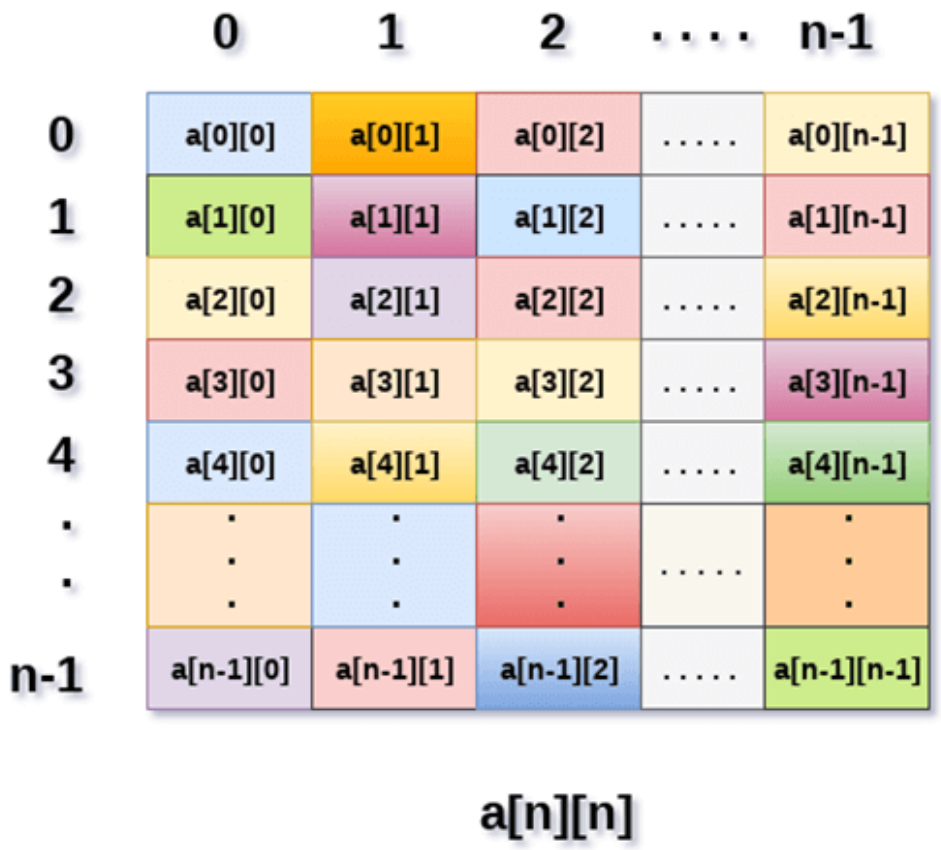
2D Array

2D Arrays

- 2D array can be defined as an **array of arrays**.
- The 2D array is organized as matrices which can be represented as the collection of rows and columns.
- However, 2D arrays are created to implement a relational database look alike data structure.

How to declare 2D Array

int arr[max_rows][max_columns];



Initializing an Array

There are two ways to initialize a two Dimensional arrays during declaration.

```
int disp[2][4] = {
    {10, 11, 12, 13},
    {14, 15, 16, 17}
};
```

OR

```
int disp[2][4] = { 10, 11, 12, 13, 14, 15, 16, 17};
```

Two-dimensional array example in C

```c
#include<stdio.h>
int main(){
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
//traversing 2D array
for(i=0;i<4;i++){
 for(j=0;j<3;j++){
   printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
 }//end of j
}//end of i
return 0;
}
```

Output

```
arr[0][0] = 1
arr[0][1] = 2
arr[0][2] = 3
arr[1][0] = 2
arr[1][1] = 3
arr[1][2] = 4
arr[2][0] = 3
arr[2][1] = 4
arr[2][2] = 5
arr[3][0] = 4
arr[3][1] = 5
arr[3][2] = 6
```

2D array example: Storing elements in a matrix and printing it

```
#include <stdio.h>
void main ()
{
   int arr[3][3],i,j;
   for (i=0;i<3;i++)
   {
      for (j=0;j<3;j++)
      {
         printf("Enter a[%d][%d]: ",i,j);
         scanf("%d",&arr[i][j]);
      }
   }
   printf("\n printing the elements ....\n");
   for(i=0;i<3;i++)
   {
      printf("\n");
      for (j=0;j<3;j++)
      {
         printf("%d\t",arr[i][j]);
      }
   }
}
```

Output

```
Enter a[0][0]: 56
Enter a[0][1]: 10
Enter a[0][2]: 30
Enter a[1][0]: 34
Enter a[1][1]: 21
Enter a[1][2]: 34

Enter a[2][0]: 45
Enter a[2][1]: 56
Enter a[2][2]: 78

 printing the elements ....

56      10      30
34      21      34
45      56      78
```

2D Array printing and calculating sum

```c
#include<stdio.h>
void main()
{
int a[2][3],i,j,sum;
printf("Enter the matrix values:");
for (i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&a[i][j]);
}
}
for(i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
for(i=0;i<2;i++)
{
sum=0;
for(j=0;j<3;j++)
{
sum=sum+a[i][j];

}
}
printf("sum is %d",sum);
}

```

Matrix Transpose

```c
#include<stdio.h>
void main()
{
int a[2][3],i,j;
printf("Enter the matrix values:");
for (i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&a[i][j]);
}
}
for(i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
printf("The transpose of matrix is :\n");
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
{
printf("%d\t",a[j][i]);

}
printf("\n");
}
```

```
ubuntu@ubuntu-Lenovo-ideapad-300-15ISK:~/Desktop$ ./a.out
Enter the matrix values:1 2 3 4 5 6
1       2       3
4       5       6
The transpose of matrix is :
1       4
2       5
3       6
ubuntu@ubuntu-Lenovo-ideapad-300-15ISK:~/Desktop$
```

```c
#include<stdio.h>
void main()
{
int a[2][3],b[3][2],i,j,c[3][2];
printf("Enter the matrix values:");
for (i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&a[i][j]);
}
}
for(i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
printf("The transpose of matrix is :\n");
/*Find the transpose of matrix*/
for(i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
c[j][i]=a[i][j];
}
printf("\n");
}
/*To print the transpose of matrix*/
for(i=0;i<3;i++)
{
for(j=0;j<2;j++)
{
printf("%d\t",c[i][j]);
}
printf("\n");
}
}
```

Matrix Addition

```c
#include<stdio.h>
void main()
{
int a[50][50],b[50][50],c[50][50],i,j;
printf("Enter the first matrix values:");
for (i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\nThe first matrix is:\n");
for(i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
printf("Enter the second matrix values:");
for (i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&b[i][j]);
}
}
printf("\nThe Second matrix is:\n");
for(i=0;i<2;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",b[i][j]);
}
printf("\n");
}
printf("\nThe sum of two matrix is :\n"
for(i=0;i<2;i++)
{
c[i][j]=0;
for(j=0;j<3;j++)
{
c[i][j]=a[i][j]+b[i][j];
printf("%d\t",c[i][j]);
}
printf("\n");
}

}
```

```
ubuntu@ubuntu-Lenovo-ideapad-300-15ISK:~/Desktop$ ./a.out
Enter the first matrix values:1 2 3 4 5 6

The first matrix is:
1       2       3
4       5       6
Enter the second matrix values:1 2 3 4 5 6

The Second matrix is:
1       2       3
4       5       6

The sum of two matrix is :
2       4       6
8       10      12
```

Matrix Multiplication

```c
#include<stdio.h>
void main()
{
int a[100][100],b[100][100],c[100][100],i,j,m,n,p,q,k;
printf("Enter rows and columns of first matrix");
scanf("%d%d",&m,&n);
printf("Enter the first matrix values:");
for (i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\nThe first matrix is:\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
printf("Enter rows and columns of second matrix");
scanf("%d%d",&p,&q);
printf("Enter the second matrix values:");
for (i=0;i<p;i++)
{
for(j=0;j<q;j++)
{
scanf("%d",&b[i][j]);
}
}
printf("\nThe Second matrix is:\n");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
{
printf("%d\t",b[i][j]);
}
printf("\n");
}
if (n!=p)
{
printf("\nThe multiplication is not possible :\n");
}
else
{
printf("\n The product of matrices is:\n");
for(i=0;i<m;i++)
{
c[i][j]=0;
for(j=0;j<q;j++)
{
for(k=0;k<m;k++)
{
c[i][j]=c[i][j]+a[i][k]*b[k][j];

}
printf("%d\t",c[i][j]);
}
printf("\n");
}
}
```

```
ubuntu@ubuntu-Lenovo-ideapad-300-15ISK:~/Desktop$ ./a.out
Enter rows and columns of first matrix 3 3
Enter the first matrix values: 1 2 3 1 2 1 4 0 2

The first matrix is:
1       2       3
1       2       1
4       0       2
Enter rows and columns of second matrix 3 2
Enter the second matrix values:1 2 3 1 1 0

The Second matrix is:
1       2
3       1
1       0

 The product of matrices is:
10      4
8       4
6       8
```

```
ubuntu@ubuntu-Lenovo-ideapad-300-15ISK:~/Desktop$ ./a.out
Enter rows and columns of first matrix 3 3
Enter the first matrix values:1 2 3 1 2 1 4 0 2

The first matrix is:
1       2       3
1       2       1
4       0       2
Enter rows and columns of second matrix 2 3
Enter the second matrix values: 1 2 3 4 5 6

The Second matrix is:
1       2       3
4       5       6

The multiplication is not possible :
```

ARRAYS

MODULE 3

- A fixed size sequenced collection of elements of the same data type.
- A collection of variables of the same data type that are referenced by a common name.

```
int rollno[64];
```

Different types of Array

- One Dimensional Arrays
- Two Dimensional Arrays

One Dimensional Arrays

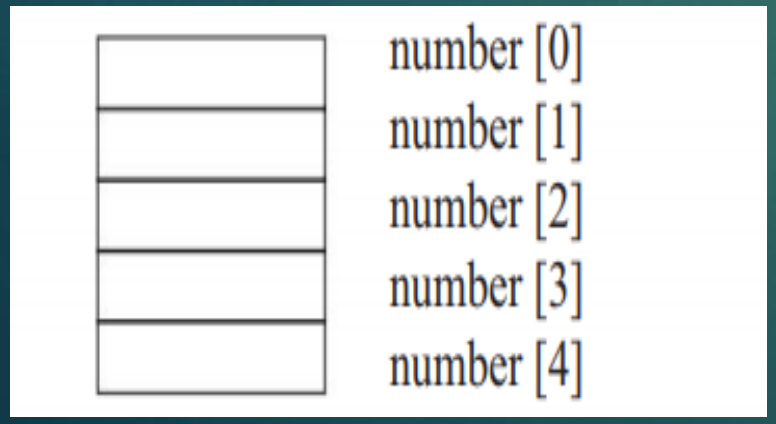
- A list of items can be given one variable name using only one subscript and such a variable is called a single subscripted variable or a one dimensional array.

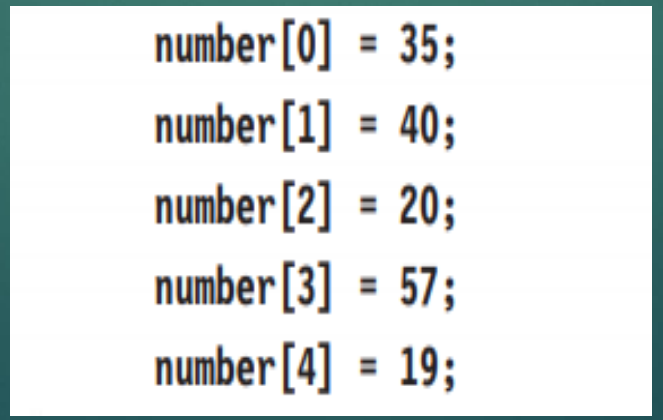
$$A = \frac{\sum_{i=1}^{n} x_i}{n}$$

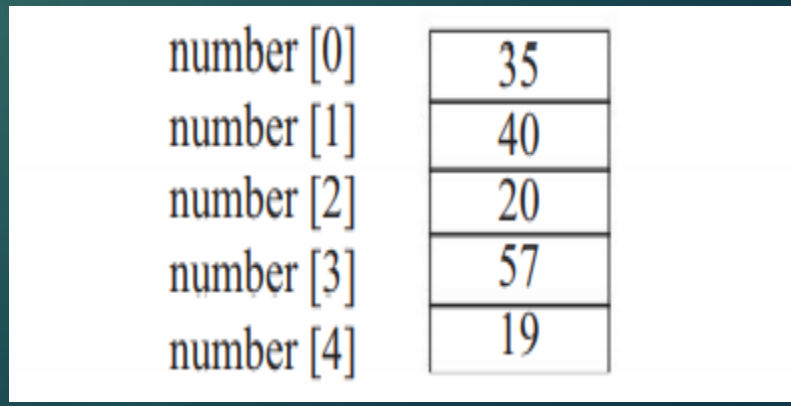
- Example: To calculate the average of n values of x.

- Set of five numbers (35,40,20,57,19) by any array variable number.
- Declare the variable number as

```
int number[5];
```







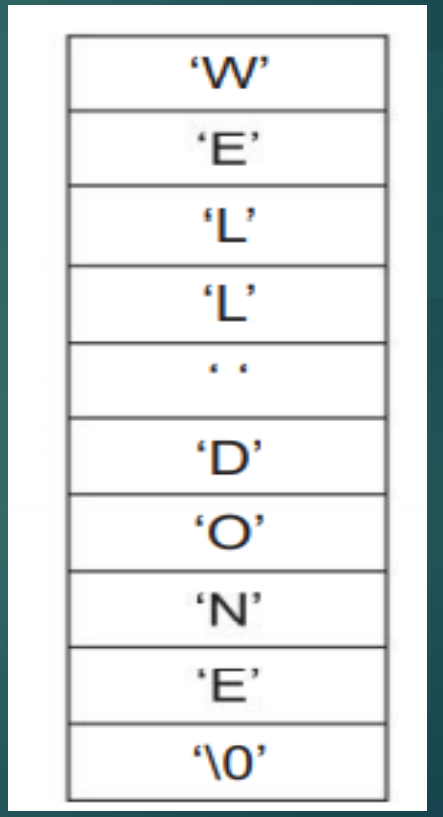
Declaration of One Dimensional Arrays

Data type variable-name[size];

Example : int group[10];

char name[10];

"WELL DONE"



Simple Program using Array

```
#include<stdio.h>
void main()
{
int avg,sum=0;
int i;
int marks[50];
for (i=0;i<=49;i++)
{
printf("enter marks:");
scanf("%d", &marks[i]);
}
for (i=0;i<=49;i++)
{
Sum = sum + marks[i];
Avg = sum/50;
printf("Average marks = %d", avg);
}
```

Initialization of array

- At compile time
- At run time

Compile time Initialization

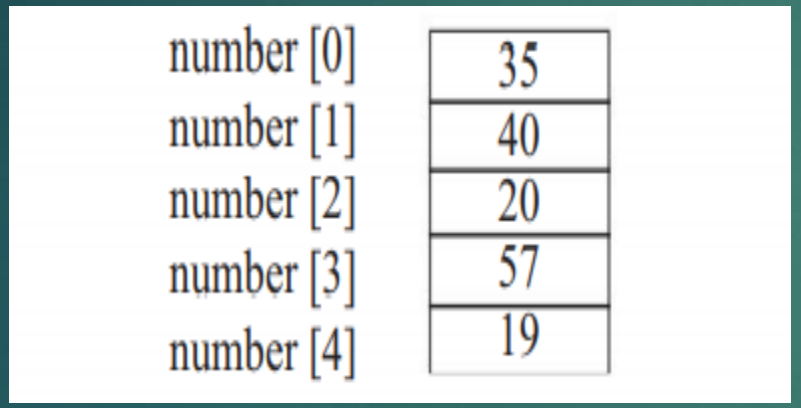
Data type array-name[size] = {list of variables};

Run Time Initialization

```
for (i=0;i<100;i++)
{
if i< 50
  a[i] = 0;
else
  a[i] =1;
}
```

Memory allocation and Accessing of Array

Base address = 2000



- Access first element a[0] = 35
- a[3]= 57

To access any element of the array at any time :

Base address + index * size of the data type

- To read 5 elements in an array and print the values.
- To read 5 elements and print the elements in reverse order.
- To find the sum and average of 5 marks using the concept of arrays.

Linear Search

```c
#include <stdio.h>
void main()
{
 int array[100], search, c, n,found;
 printf("Enter number of elements in array\n");
 scanf("%d", &n);
 printf("Enter %d integer(s)\n", n);
 for (c = 0; c < n; c++)
 scanf("%d", &array[c]);
 printf("Enter a number to search\n");
 scanf("%d", &search);
 for (c = 0; c < n; c++)
 {
 if (array[c] == search) /* If required element is found */
 {
 found=1;
 break;
 }
 }
 if (found == 1)
 printf("%d is present at location %d.\n", search, c+1);
 else
 printf("%d isn't present in the array.\n", search);
 }
```

STRING

MODULE 3 PART B

String

- Sequence of characters that is treated as a single data item..
- String is represented using double quotation marks.

Examples : “Hello world” , “xyz123@” , “Good”

- Strings in C are represented by array of characters.
- The end of the string is marked with a special character, the null character, which is simply the character with the ASCII value 0.
- ‘\0’ represents the end of the string. It is also referred as String terminator & Null Character

Declaration of string

- General form for declaration of a string variable:

```
char string_name[size];
```

Example:

```
char city[10];
char name[30];
```

Initialization of string

- Two forms are there:

Form1 : char city [9] = “NEW YORK”;

Size = 8+1

Form2: char city [9] = {‘N’,’E’,’W’,’ ‘,’Y’,’O’,’R’,’K’,’\0’};

Initialization of string

char city [ ] = “NEW YORK”;

Size = 8+1 (Automatically determined by compiler)

char city [ ] = {‘N’,’E’,’W’,’ ‘,’Y’,’O’,’R’,’K’,’\0’};

Size = 8+1 (Automatically determined by compiler)

Char string[10] = “GOOD”;

Other Declarations that results Error

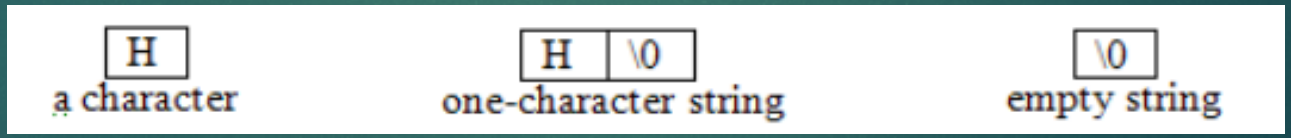
Example 1: *char string[3] = “good”;*

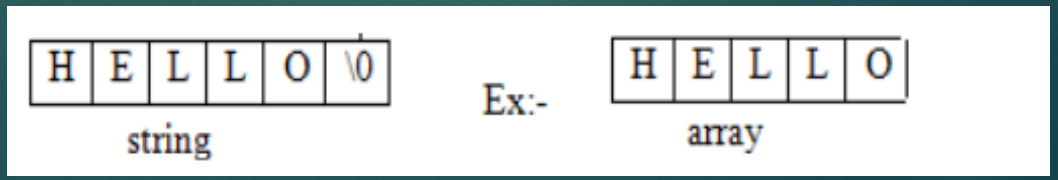
Example 2: *char string[5] ; //Cannot Separate the initialization from declaration*
string = “good”;

Example 3: *char s1[4] = “abc”;*
char s2[4];
s2 = s1; //Array name cannot be used as left operand of assignment operator

Storing the strings in memory

- A string is stored in array, the name of the string is a pointer to the beginning of the string.
- The character requires only one memory location.
- If we use one-character string it requires two locations.
- The difference is shown below,





Reading strings from terminal

- String can be read from the user by using three ways:

  a) scanf( ) function
  b) gets( ) function
  c) getchar( ) function

Using Scanf

- Used with %s format specification

```
char address[10]
scanf (“%s”, address);
```

Here don’t use “&” because name of string is a pointer to array.

The problem with scanf( ) is that it terminates its input on the first white space it finds.

```
#include<stdio.h>
void main()
{
char name[10];
printf(“Enter the name:”);
scanf(“%s”,name);
printf(“Name is %s”,name);
}
```

```
Enter the name: Dennis Richie
Name is Dennis
```

Using Scanf

- Used with %ws format specification

```
char address[10]
scanf (“%ws”, address);
```

If w is greater or equal than number of characters typed in, the entire string will be stored in string variable.

If w is less than number of characters typed in the string, the excess characters will be truncated and left unread.

```
#include<stdio.h>
void main()
{
char name[10];
printf(“Enter the name:”);
scanf(“%5s”,name);
}
```

Enter the name: Dennis Richie

```c
#include<stdio.h>
void main()
{
char name[30];
printf("Enter the name:");
scanf("%[^\n]", name);
 printf("%s", name);
}
```

```
Enter the name: Hello World
 Hello World
```

Using gets()

- gets( ) function takes the starting address of the string which will hold the input.
- string inputted using gets( ) is automatically terminated with a null character.
- The C gets function is used to read a line of text from a standard input device and store it in the String variable.
- When it reads the newline character, then the C gets function will terminate.

```c
#include<stdio.h>
void main()
{
char name[20];
printf("Enter the name:");
gets(name);
printf("Name is %s",name);
}
```

Enter the name: Dennis Richie
Name is Dennis Richie

Using getchar()

- Read successive single characters from the input and place them into a character array.
- Entire line of text can be read and stored in an array.
- Reading is terminated when the newline character is entered and the null character is placed at the end of the string.

```
char ch;
ch = getchar( );
```

```c
#include <stdio.h>
 void main( )
 {
 char line[81], character;
 int c;
 c = 0;
 printf("Enter text. Press <Return> at end\n");
 do
 {
 character = getchar();
 line[c] = character;
 c++;
 }
 while(character != '\n');
 c = c - 1;
 line[c] = '\0';
 printf("\n%s\n", line);
 }
```

```
Enter text. Press <Return> at end
sneha sreedevi

sneha sreedevi
```

Copy one string into another and count the number of characters copied

```c
#include <stdio.h>
 void main( )
 {

 int i;
 char string2[30],string1[30];
 printf("Enter a string \n");
 scanf("%s", string2);
 for( i=0 ; string2[i] != '\0'; i++)
 string1[i] = string2[i];
 string1[i] = '\0';
 printf("\n");
 printf("%s\n", string1);
 printf("Number of characters = %d\n", i );

 }
```

```
Sneha

Sneha
Number of characters = 5
```

Writing Strings To Screen

- Using printf()
- Using puts()
- Using putchar()

Using printf()

- Used with %s format specification

```
char address[10]
printf (“%s”, address);
```

Using puts()

- Used to print the strings including blank spac

puts(str);

Example:

char message[20]=“Hello world”;

puts(message);

```
#include <stdio.h>


void main()

{

char name[30];
puts("Enter a string ");
gets (name);
puts("Entered string is");
puts (name);
}
```

```
Enter a string
Ram is studying in fourth class
Entered string is
Ram is studying in fourth class
```

Using putchar( )

- To print a character on the screen.

  char ch = 'A';

  putchar (ch);

```c
#include <stdio.h>


void main()

{

char name[6] = "PARIS";
int i;
for (i=0; i<5; i++)
{
 putchar(name[i]);
putchar('\n');
}
}
```

```
P
A
R
I
S
```

```c
#include <stdio.h>

int main()

{

char string[] = "Welcome to the world of C Programming\n";

int i=0;

while(string[i]!='\0')

{

putchar(string[i]);

i++;

}

return 0;

}
```

```
Welcome to the world of C Programming
```

Putting Strings Together

- Just as we cannot assign one string to another directly, we cannot join two strings together by the simple arithmetic addition.
- That is, the statements such as

```
string3 = string1 + string2;
string2 = string1 + “hello”;
```

are not valid.

- The process of combining two strings together is called concatenation.

Comparison of Strings Together

- C does not permit the comparison of two strings directly. That is, the statements such as

```
if(name1 == name2)
if(name == “ABC”)
```

are not permitted.

- It is therefore necessary to compare the two strings to be tested, character by character.
- The comparison is done until there is a mismatch or one of the strings terminate into a null character, whichever occurs first.

```c
#include <stdio.h>

int main()

{

char str1[30];
char str2[30];
int i=0;
printf("Enter the string1");
gets(str1);
printf("Enter the string2");
gets(str2);
while(str1[i] == str2[i] && str1[i] != '\0'&& str2[i] != '\0')
{
 i = i+1;
}
 if (str1[i] == '\0' && str2[i] == '\0')
 printf("strings are equal\n");
 else
 printf("strings are not equal\n");
return 0;

}
```

```
Enter the string1 Pallavi Sneha
Enter the string2Pallavi Padmesh
strings are not equal
```

```
Enter the string1 Hello World
Enter the string2 Hello World
strings are equal
```

String Handling Functions

- C supports a number of string handling functions.
- All of these built-in functions are aimed at performing various operations on strings and they are defined in the header file string.h.
    - strlen( )
    - strcpy( )
    - strcat ( )
    - strcmp ( )

strlen()

- Counts and returns the number of characters in a string excluding null character.
- It takes the form

n = strlen(string)

Example:

```
char str1[ ] = “WELCOME”;
int n;
n = strlen(str1);
```

```c
#include <stdio.h>
#include<string.h>
int main( )
{
char string[50];
int length;
printf("Enter any string: ");
gets(string);
length=strlen(string);
printf("The length of string=%d", length);
return 0;
}
```

```
Enter any string: sneha sreedevi
The length of string=14
```

strcpy()

- This function is used to copy one string to the other.
- Its syntax is as follows:

  strcpy(string1,string2);

- where string1 and string2 are one-dimensional character arrays.
- This function copies the content of string2 to string1.

Example:

```
char str1[ ] = “WELCOME”;
char str2[ ] =”HELLO”;
 strcpy(str1,str2);
```

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

char city[15];
strcpy(city, "BANGALORE") ;
puts(city);
return 0;
}
```

```
BANGALORE
```

- A program to copy one string to another using strcpy() function

```c
#include<stdio.h>
#include<string.h>
int main()
{
char string1[30],string2[30];
printf("Enter first string:");
gets(string1);
printf("\nEnter second string:");
gets(string2);
strcpy(string1,string2);
printf("\nFirst string=%s",string1);
printf("\nSecond string=%s",string2);
return 0;
}
```

```
Enter first string: Hello World

Enter second string: Hai all

First string= Hai all
Second string= Hai all
```

strcat ( )

- This function is used to concatenate two strings. i.e., it appends one string at the end of the specified string.
- Its syntax as follows:

strcat(string1,string2);

- where string1 and string2 are one-dimensional character arrays.
- This function joins two strings together.

Example: Example:

```
char str1[20 ] = “HELLO”;
char str2[20] = ”WORLD”;
strcat(str1,str2);
```

```c
#include<stdio.h>
#include<string.h>
int main()
{
char string1[30],string2[15];
printf("\n Enter first string:");
gets(string1);
printf("\n Enter second string:");
gets(string2);
strcat(string1,string2);
printf("\n Concatenated string=%s",string1);
return 0;
}
```

```
Enter first string: Hai all

Enter second string:Welcome to C programming

Concatenated string= Hai allWelcome to C programming
```

strcmp ( )

- Compares two strings character by character (ASCII comparison) and returns one of three values {-1,0,1}.

Return value	Description
0	When both are equal
<0	If ASCII value of a character of the first string is less than the ASCII value of the character of the second string then function will return negative value
>0	If ASCII value of a character of the first string is greater than the ASCII value of the character of the second string then function will return positive value

Example :

```
int n;
char city[20] = “MADRAS”;
char town[20] = “MANGALORE”;
n = strcmp(city, town);

//ASCII value of D = 68
//ASCII value of N = 78
```

```c
#include<string.h>
int main( )
{
char a[100], b[100];
printf("Enter the first string\n");
gets(a);
printf("Enter the second string\n");
gets(b);
if( strcmp(a,b) == 0 )
printf("Entered strings are equal.\n");
else
printf("Entered strings are not equal.\n");
return 0;
}
```

```
Enter the first string
Delhi
Enter the second string
New Delhi
Entered strings are not equal.
```

strrev()

- strrev() function reverses a given string in C language.

```c
#include <stdio.h>
#include <string.h>
int main()
{
   char str[40]; // declare the size of character string
   printf (" \n Enter a string to be reversed: ");
   scanf ("%s", str);

   // use strrev() function to reverse a string
   printf (" \n After the reverse of a string: %s ", strrev(str));
   return 0;
}
```

```
Enter a string to be reversed: AMBULANCE

 After the reverse of a string: ECNALUBMA
```

Reverse of a string

```
#include<stdio.h>
#include<string.h>
int main()
{
int len,i,j,temp;
char string[50];
printf("Enter the string:");
scanf("%[^\n]",string);
len = strlen(string);
for(i=0;i<len/2;i++)
{
  temp = string[i];
  string[i]=string[len-i-1];
  string[len-i-1]=temp;
  }
  printf("Reverse of the string is %s", string);
  }
```

Table Of Strings

C	h	a	n	d	i	g	a	r	h
M	a	d	r	a	s				
A	h	m	e	d	a	b	a	d	
H	y	d	e	r	a	b	a	d	
B	o	m	b	a	y				

```
char city[ ] [ ]
      {
         "Chandigarh",
         "Madras",
         "Ahmedabad",
         "Hyderabad",
         "Bombay"
      } ;
```

Sorting a string

```
#include<stdio.h>
#include<string.h>
int main()
{
char str[10][50],temp[50];
 int i,j,n;
printf("Enter the no of Words to be entered:\n")
;
 scanf("%d",&n);
 printf("Enter the words:");
for(i=0;i<n;i++)   //Reading
scanf("%s[^\n]",str[i]);
//Sorting
for(i=0;i<n-1;i++)
 {
    for(j=i+1;j<n;j++)
   {
    if(strcmp(str[i],str[j])>0)
    {
      strcpy(temp,str[i]);
       strcpy(str[i],str[j]);
      strcpy(str[j],temp);
 }
  }
 }
```

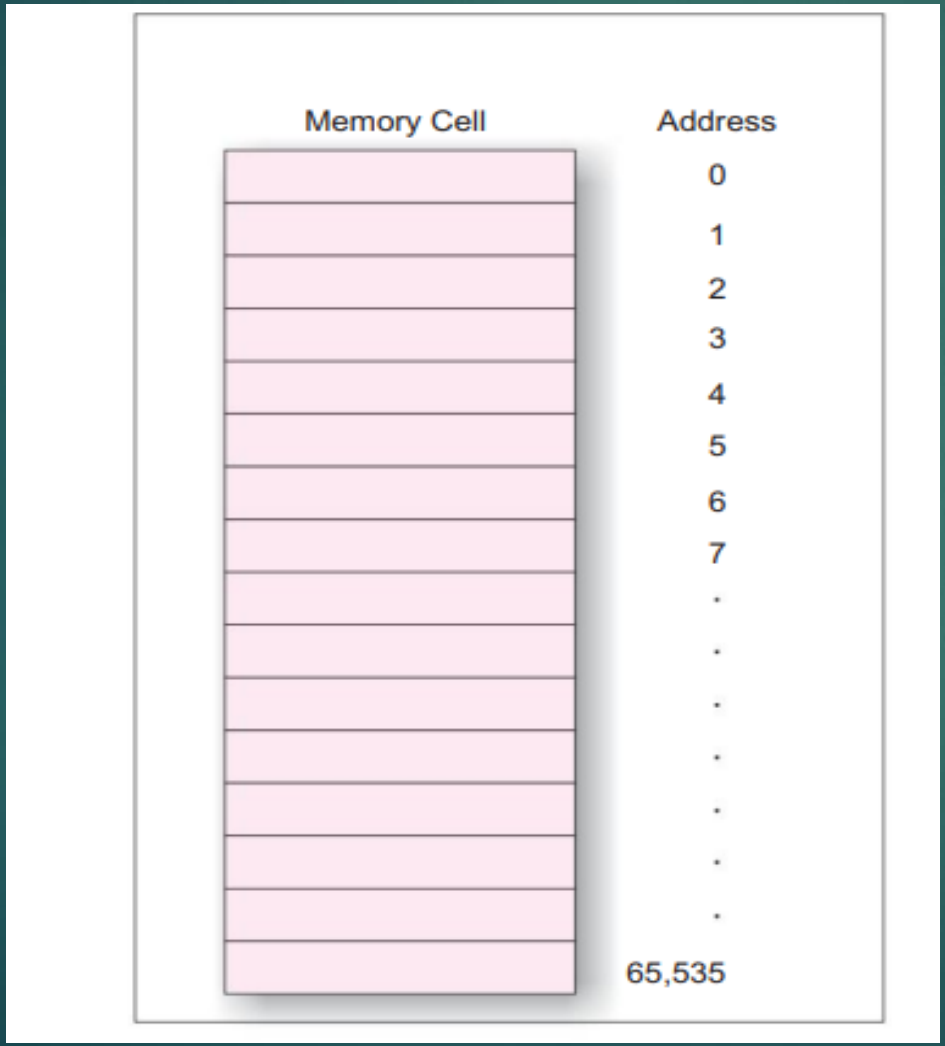
```
//Printing
printf("\nIn lexicographical order: \n");
  for(i=0;i<n;i++)
      puts(str[i]);
      return 0;
}
```

```
Enter the no of Words to be entered:
4
Enter the words:heap
stack
hello
queue

In lexicographical order:
heap
hello
queue
stack
```

POINTERS

MODULE 5

Memory organization

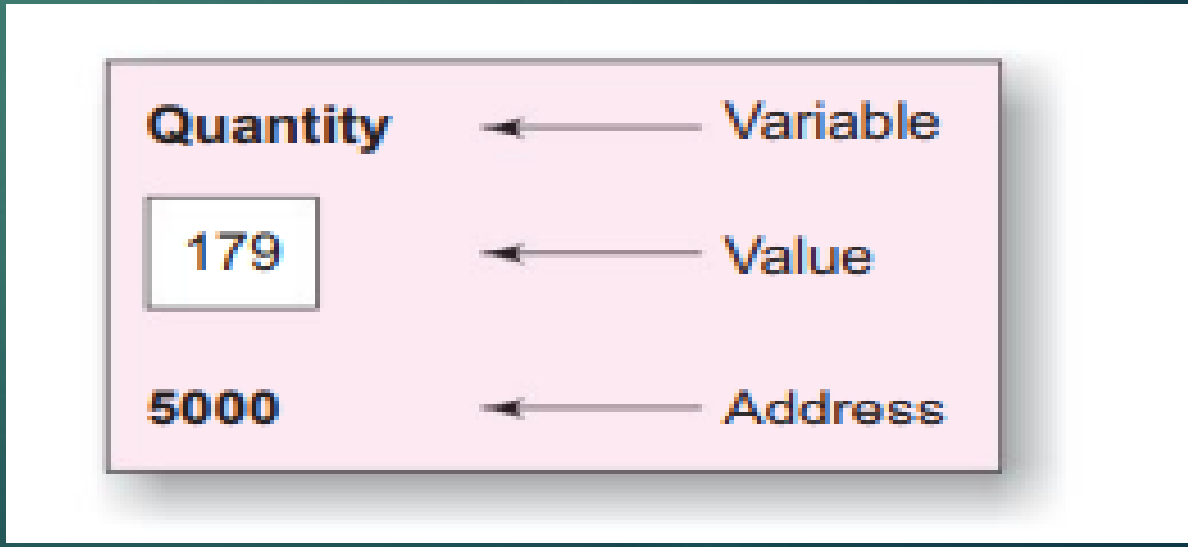


Pointer

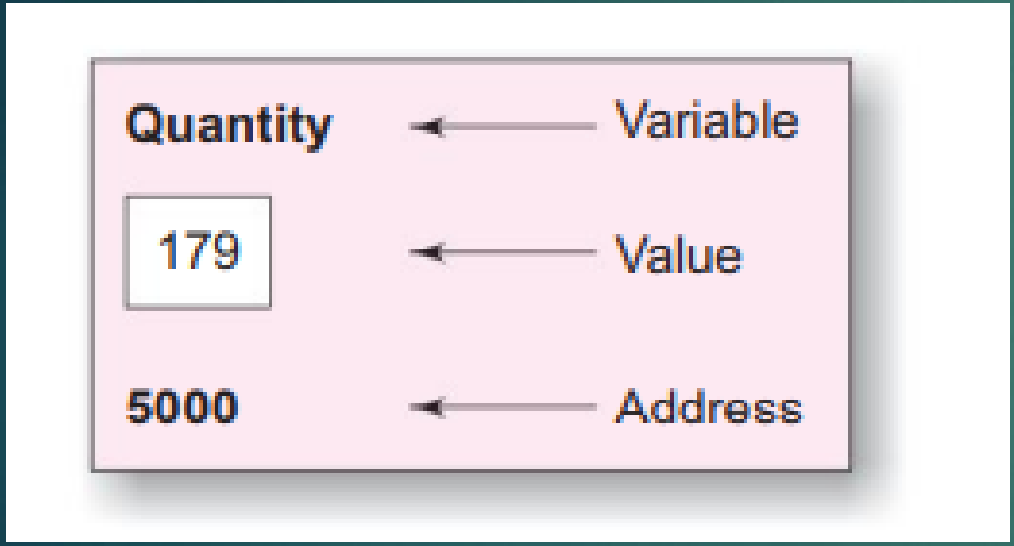
- Variable that holds the memory address of the location of another variable in the memory.
- It is a derived data type.

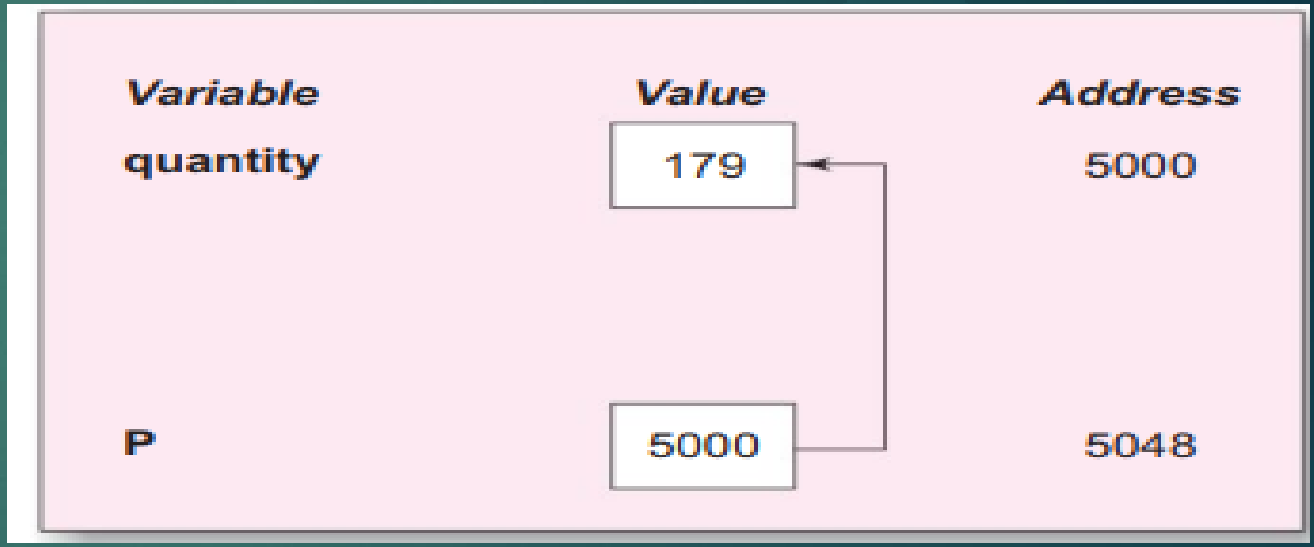
Example:

```
int Quantity = 179;
```



Pointer Variable





Declaration

syntax: *datatype * pointer name;*

Pointer variable
Holds the address of another variable that is of the specified datatype given

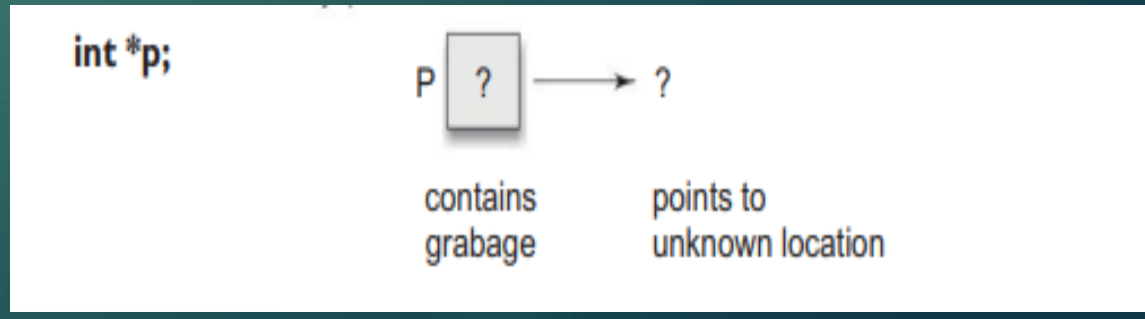
Pointer variable

datatype * pointer name

- * tells that the variable pointer name is a pointer variable
- Pointer name needs a memory location
- Pointer name points to variable of type data type.

Example : int *p; // integer pointer

float *p // float pointer



Declaration styles

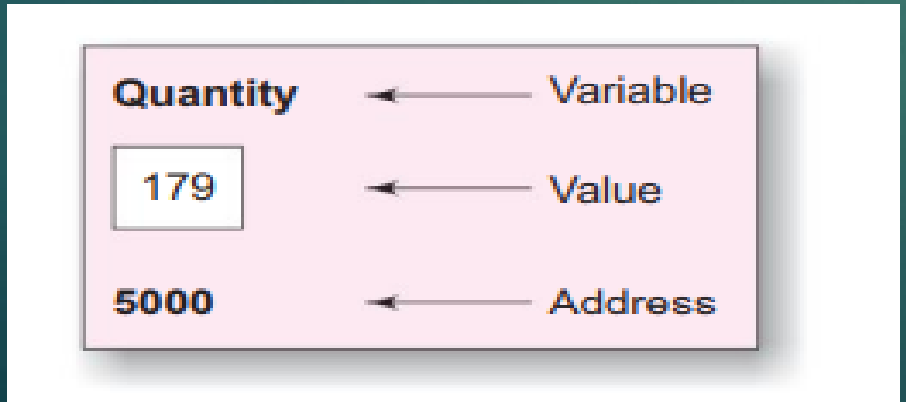
- Declared similarly as normal variables except for the addition of the unary operator(*).
- * can appear anywhere between type name and the pointer variable name.

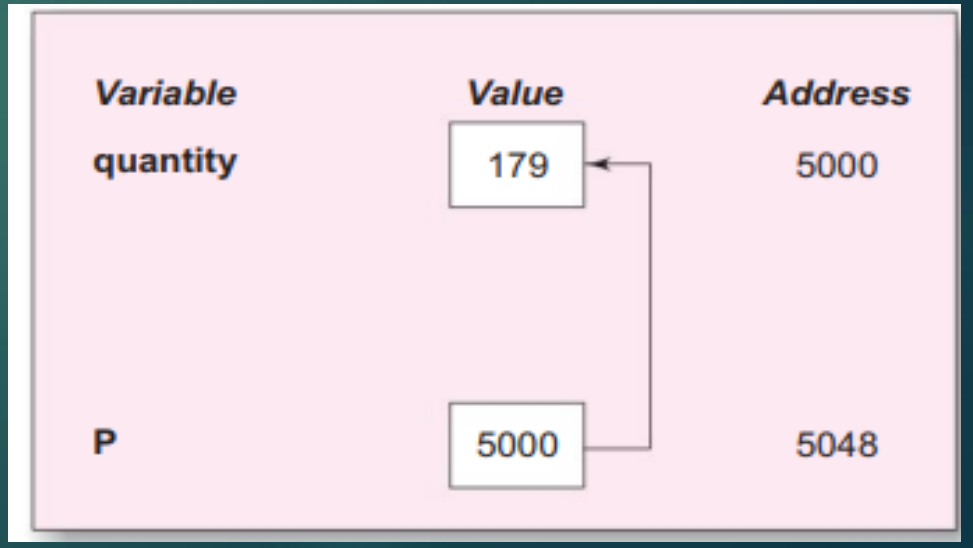
```
int*        p;    //Style 1
int* p;   //Style 2
int         * p;  // Style 3
```

Access the address of a variable

- Using & operator available in C.
- The operator & immediately preceding a variable returns the address of the variable associated with it.

Example : p = &quantity;





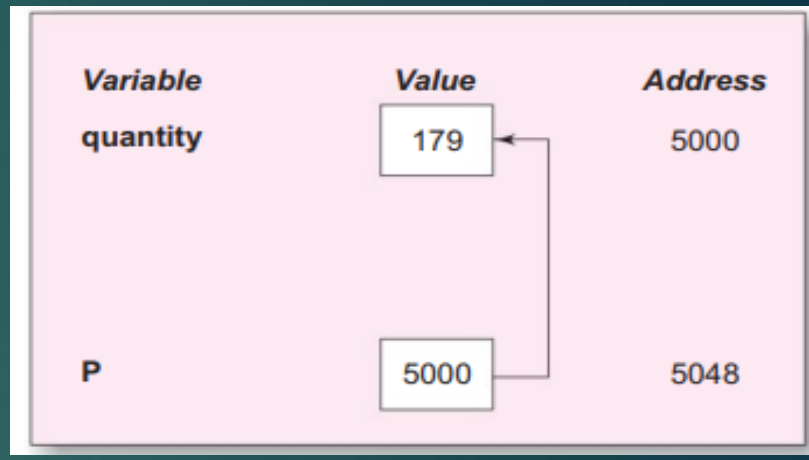
Initialize a pointer

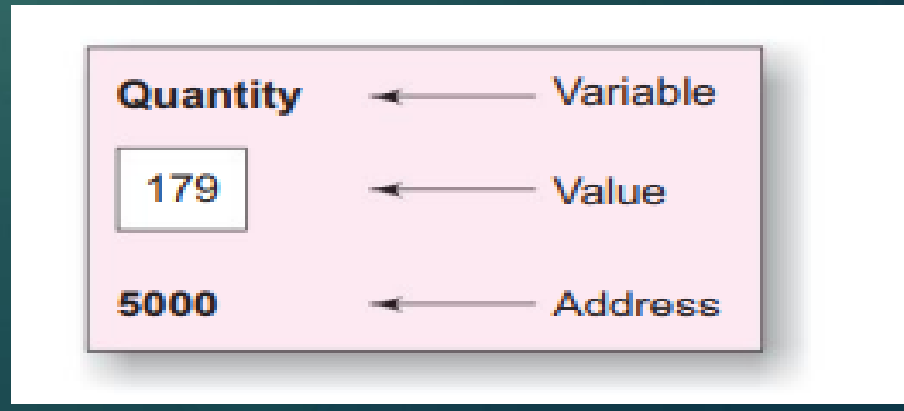
- Assigning address of a variable to a pointer variable.
- Use assignment operator for initialization.

Example: int quantity;

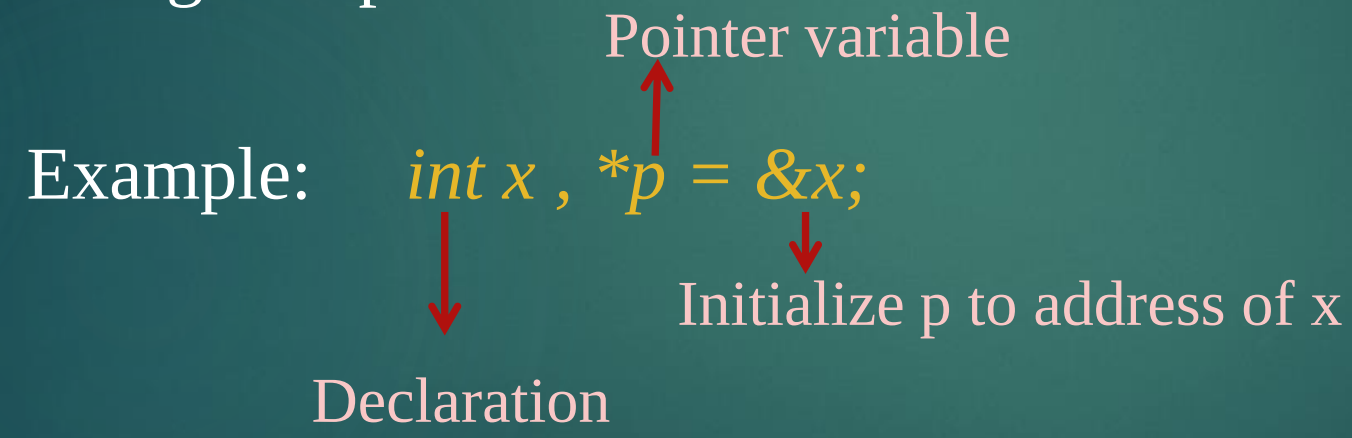
int *p; /*declaration*/

p = &quantity /*initialization*/





- Combination of declaration of data variable, declaration of pointer variable and initialization of pointer variable in single step.



- Declare pointer variable with an initial value of NULL or zero is also possible.

```
int *p = NULL;
int *p = 0;
```

Accessing a variable through its pointer

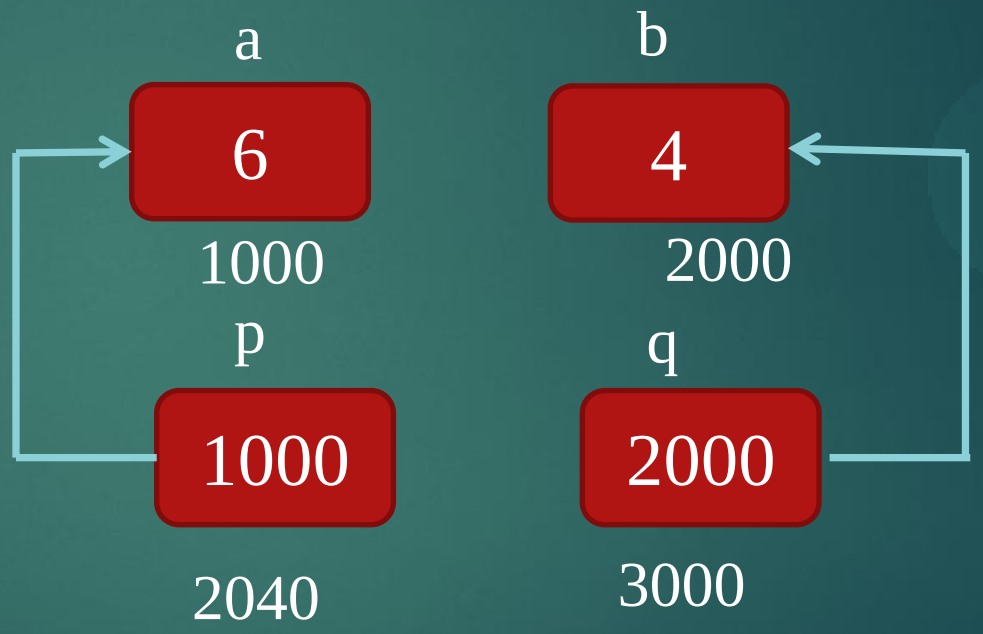
- Using unary operator * called indirection or dereferencing.

Example :

```
int quantity, *p, n;
quantity = 179;
p = &quantity; // address of variable quantity
n = *p; //Returns value at address
```

Example

```
int a= 6, b=4;
int *p,*q;
p=&a;
q=&b;
printf("Value of a = %d",*p);
```

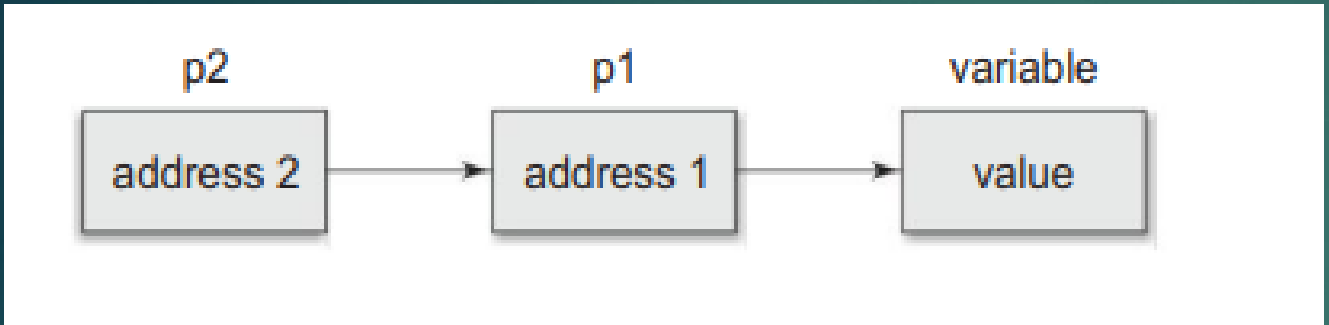


Example: Predict the output

```
int a= 6, b=4;
int *p,*q;
p=&a;
q=&b;
printf("value of a = %d",*p);
printf("address of a = %d",&a);
printf("address of a = %d",p);
printf("address of p = %d",&p);
```

```
void main()
{ int x, y;
int *ptr;
x = 10;
ptr = &x;
y = *ptr;
printf("Value of x is %d\n\n",x);
printf("%d is stored at addr %u\n", x, &x);
printf("%d is stored at addr %u\n", *&x, &x);
printf("%d is stored at addr %u\n", *ptr, ptr);
printf("%d is stored at addr %u\n", ptr, &ptr);
printf("%d is stored at addr %u\n", y, &y);
*ptr = 25;
printf("\nNow x = %d\n",x);
}
```

Chain of Pointers



- A variable that is a pointer to a pointer must be declared as:

*int **p2*

```
main ( )
 {
 int x, *p1, **p2;
 x = 100;
 p1 = &x;
p2 = &p1
printf (“%d”, **p2);
 }
```

Pointer Expressions

- C allows us to add integers to or subtract integers from pointers, as well as to subtract one pointer from another.

- Pointers can also be compared using the relational operators.

Sample Programs

- Program to compute the sum of two numbers using the concept of pointers
- Program to compute the largest of three numbers

Elements of array stored

```c

#include<stdio.h>

int main()
{
    int arr[5] = {1, 2, 3, 4, 5}, i;

    for(i = 0; i < 5; i++)
    {
        printf("Value of arr[%d] = %d\t", i, arr[i]);
        printf("Address of arr[%d] = %u\n", i, &arr[i]);
    }

    return 0;
}
```

```
Value of arr[0] = 1     Address of arr[0] = 1297118080
Value of arr[1] = 2     Address of arr[1] = 1297118084
Value of arr[2] = 3     Address of arr[2] = 1297118088
Value of arr[3] = 4     Address of arr[3] = 1297118092
Value of arr[4] = 5     Address of arr[4] = 1297118096
```

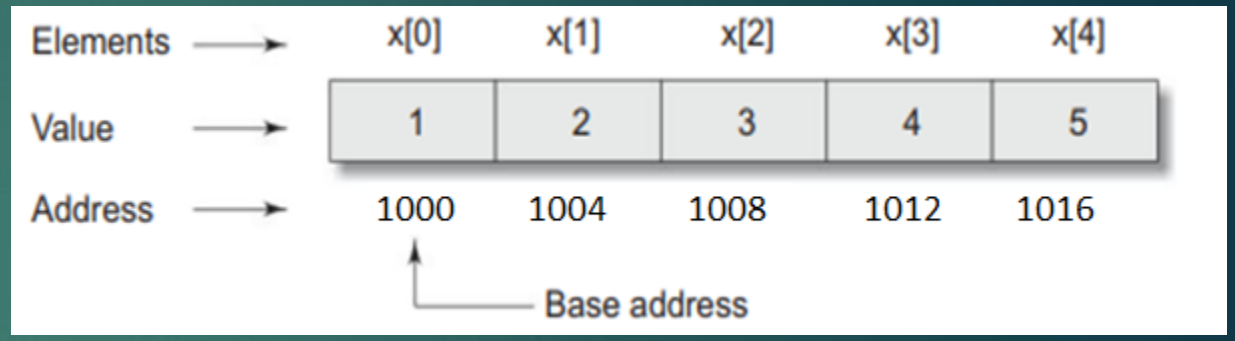
Using pointers to access elements and address of elements in an array

```c

#include<stdio.h>

int main()
{
    int arr[5] = {1, 2, 3, 4, 5}, i;

    for(i = 0; i < 5; i++)
    {
        printf("Value of a[%d] = %d\t", i, *(arr + i) );
        printf("Address of a[%d] = %u\n", i, arr + i );
    }

    return 0;
}
```

```
Value of a[0] = 1       Address of a[0] = 4063571264
Value of a[1] = 2       Address of a[1] = 4063571268
Value of a[2] = 3       Address of a[2] = 4063571272
Value of a[3] = 4       Address of a[3] = 4063571276
Value of a[4] = 5       Address of a[4] = 4063571280
```

Pointers and Arrays

```
int x[5] = {1, 2, 3, 4, 5};
int *p;
p = x; / p = &x[0];
```



Assigning 1-D array to a Pointer variable

```c
#include<stdio.h>

int main()
{
    int arr[5] = {1, 2, 3, 4, 5}, i;
    int *p;
    p = arr;
    for(i = 0; i < 5; i++)
    {
        printf("Value of a[%d] = %d\t", i, *(p + i) );
        printf("Address of a[%d] = %u\n", i, p + i );
    }
    return 0;
}
```

```
Value of a[0] = 1       Address of a[0] = 2432394160
Value of a[1] = 2       Address of a[1] = 2432394164
Value of a[2] = 3       Address of a[2] = 2432394168
Value of a[3] = 4       Address of a[3] = 2432394172
Value of a[4] = 5       Address of a[4] = 2432394176
```

Pointer Expressions

- C allows us to add integers to or subtract integers from pointers, as well as to subtract one pointer from another.

  Example: p1 + 4

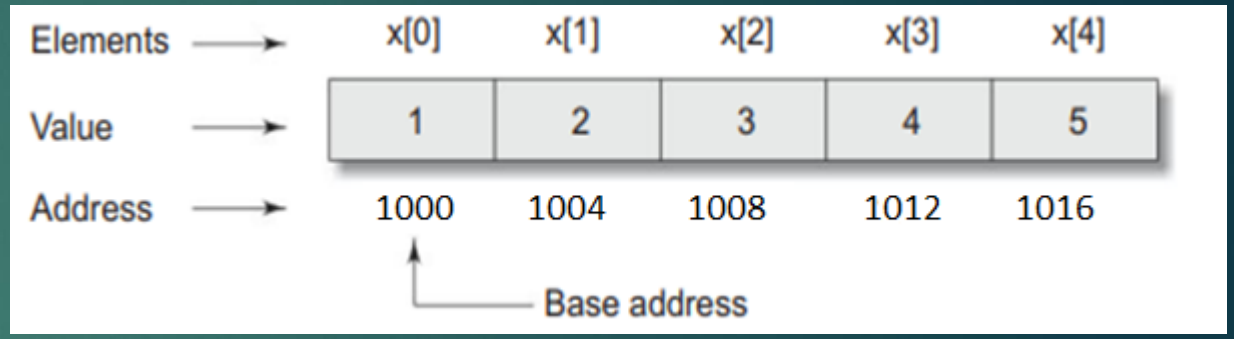
  p2 – 2

  p1 – p2

Increment/Decrement

- **Post Increment**

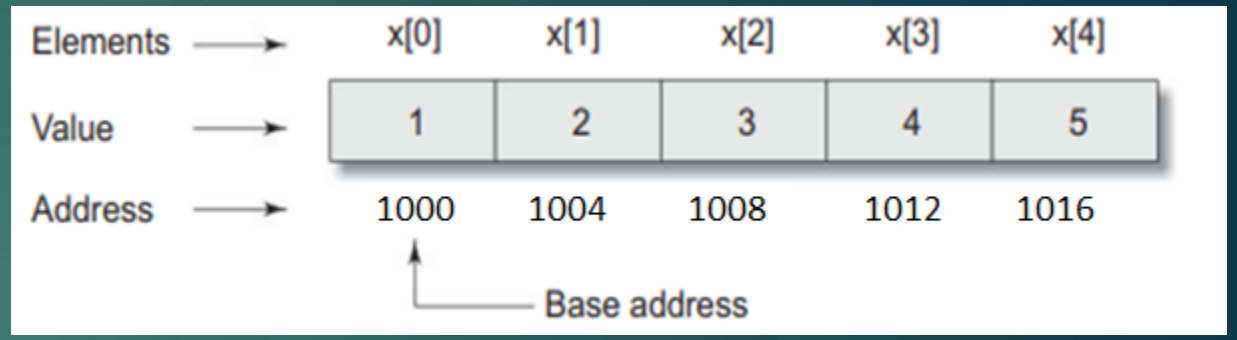
```
int x[5] = {1, 2, 3, 4, 5};
 int *p;
 p = x;
 p++ ;
printf(“%d”,*p++);
printf(“%d”,*p);
```



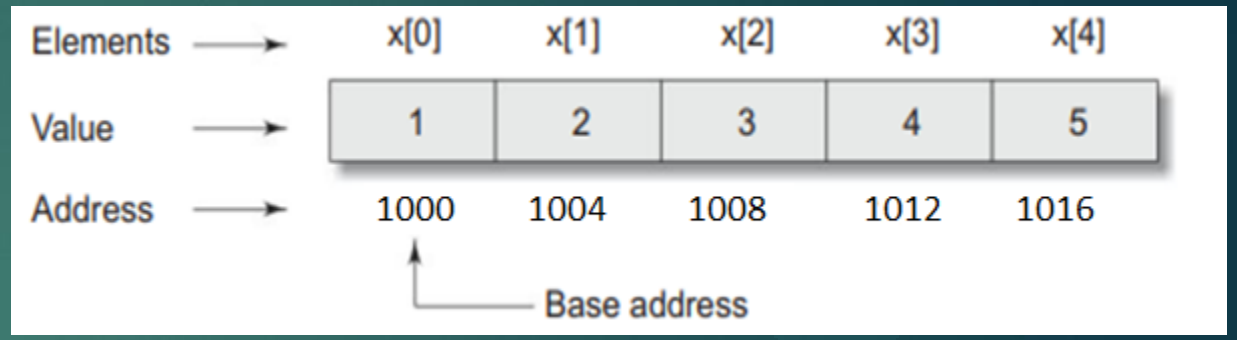
Increment/Decrement

- **Pre Increment**

```
int x[5] = {1, 2, 3, 4, 5};
 int *p;
 p = x;
 ++p ;
printf(“%d”,*++p);
printf(“%d”,*p);
```



```
int x[5] = {1, 2, 3, 4, 5};
 int *p;
 p = x;
 p + 2 ;
 p + 3;
printf("%d",*(p+2));
 printf("%d",*(p+3));
```



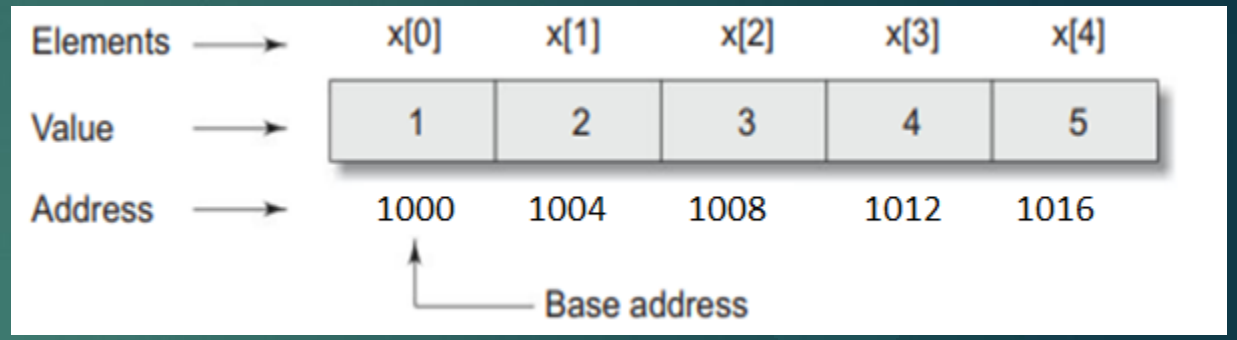
- Write a program using pointers to compute the sum of all elements stored in an array.

```
main()
 {
 int *p, sum, i;
 int x[5] = {5,9,6,3,7};
 i = 0;
 p = x; /* initializing with base address of x */
 printf("Element Value Address\n\n");
 while(i < 5)   or // for (i=x; i<=x+5;i++)
 {
 printf(" x[%d] %d %u\n", i, *p, p);
 sum = sum + *p; /* accessing array element */
 i++, p++; /* incrementing pointer */
 }
 printf("\n Sum = %d\n", sum);
 printf("\n &x[0] = %u\n", &x[0]);
 printf("\n p = %u\n", p);
 }
```

Using pointers to access elements and address of array

- *(p+0)
- *(p+1)
- *(p+2)
- *(p+3)
- *(p+4) → *(p+i)
- One dimensional array

(x+i) /(p+i)



- We can use a array names as pointers but assigning a new address to them is not possible.
- Example:

```
int main()
{
int a[]={1,2,3,4,5};
printf("%p",a++);
return 0;
}
```

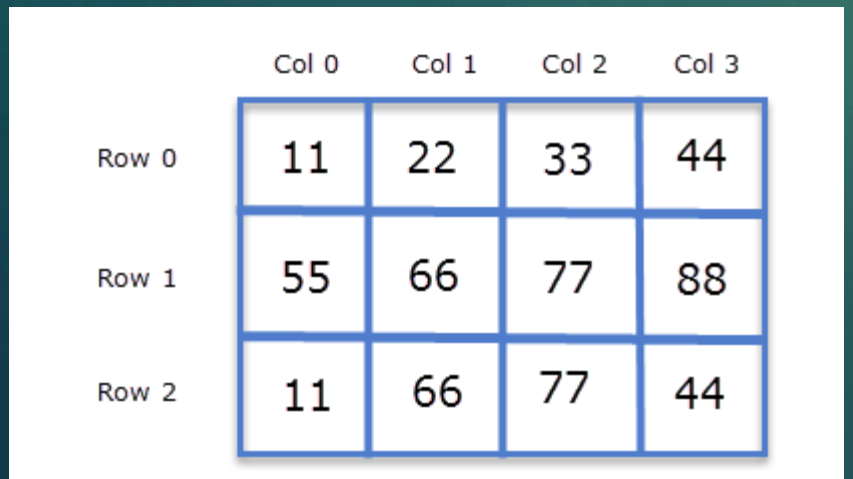
Pointer to an Array *int(*p)[5]*

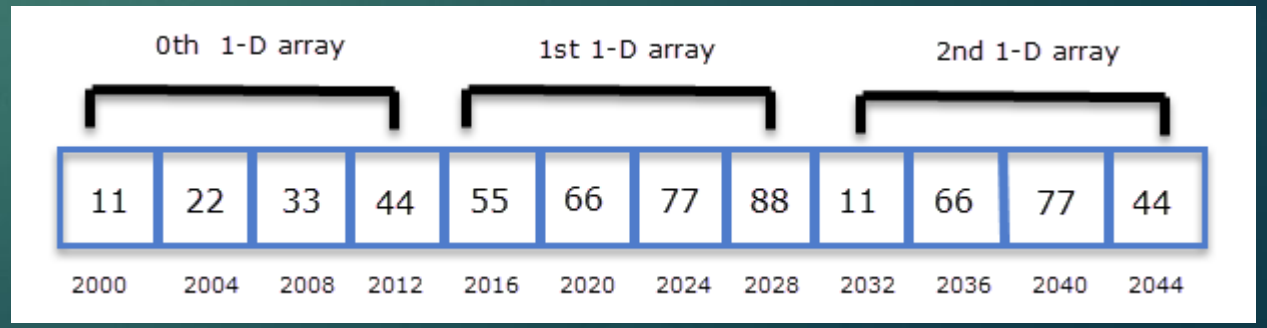
```c
#include<stdio.h>

int main()
{
    int *p; // pointer to int
    int (*parr)[5]; // pointer to an array of 5 integers
    int arr[5]; // an array of 5 integers

    p = arr;
    parr = arr;

    printf("Address of p = %u\n", p );
    printf("Address of parr = %u\n", parr );

    p++;
    parr++;

    printf("\nAfter incrementing p and parr by 1 \n\n");
    printf("Address of p = %u\n", p );
    printf("Address of parr = %u\n", parr );
    printf("Address of parr = %u\n", *parr );
return 0;
}
```

```
Address of p = 1282539200
Address of parr = 1282539200

After incrementing p and parr by 1

Address of p = 1282539204
Address of parr = 1282539220
Address of parr = 1282539220
```

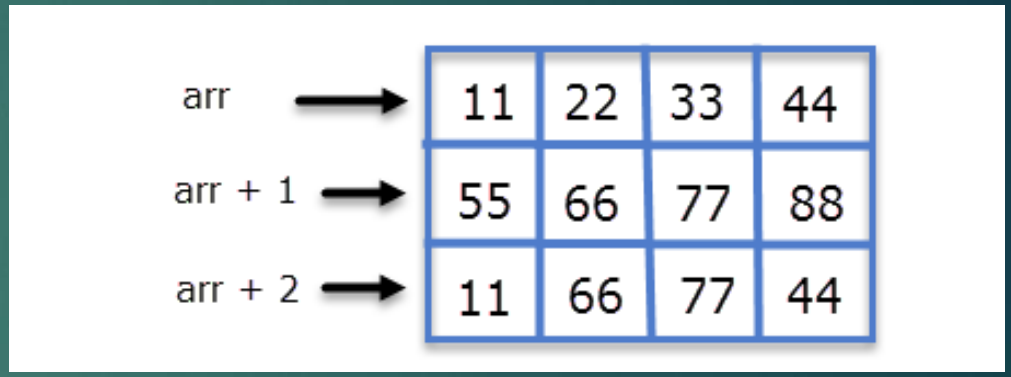
Pointers and 2D Arrays

```
int arr[3][4] = { {11,22,33,44},
                  {55,66,77,88},
                  {11,66,77,44} };
```





- arr points to 0th 1-D array.
  (arr + 1) points to 1st 1-D array.
  (arr + 2) points to 2nd 1-D array.



(arr + i) points to ith 1-D array
*(arr+i) points to the base address of the ith 1-D array.

- *(arr + i) points to the address of the 0th element of the 1-D array.
  *(arr + i) + 1 points to the address of the 1st element of the 1-D array
  *(arr + i) + 2 points to the address of the 2nd element of the 1-D array
- Hence we can conclude that:
- *(arr + i) + j points to the base address of jth element of ith 1-D array.
- On dereferencing *(arr + i) + j we will get the value of jth element of ith 1-D array.

(*(*(arr + i) + j))

```c
#include<stdio.h>

int main()
{
    int arr[3][4] = {
                        {11,22,33,44},
                        {55,66,77,88},
                        {11,66,77,44}
                    };

    int i, j;

    for(i = 0; i < 3; i++)
    {
        printf("Address of %d th array %u \n",i , *(arr + i));
        for(j = 0; j < 4; j++)
        {
            printf("arr[%d][%d]=%d\n", i, j, *( *(arr + i) + j) );
        }
        printf("\n\n");
    }
    return 0;
}
```

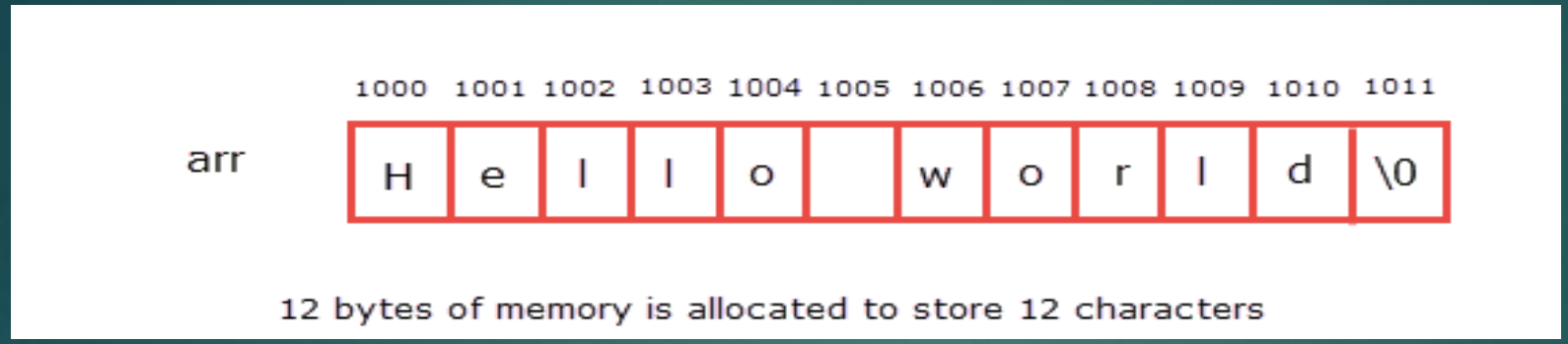
```
Address of 0 th array 3608971664
arr[0][0]=11
arr[0][1]=22
arr[0][2]=33
arr[0][3]=44


Address of 1 th array 3608971680
arr[1][0]=55
arr[1][1]=66
arr[1][2]=77
arr[1][3]=88


Address of 2 th array 3608971696
arr[2][0]=11
arr[2][1]=66
arr[2][2]=77
arr[2][3]=44
```

```c
#include<stdio.h>

int main()
{
    int arr[3][4] = {
                        {11,22,33,44},
                        {55,66,77,88},
                        {11,66,77,44}
                    };
    int i, j;
    int (*p)[4];
    p = arr;
    for(i = 0; i < 3; i++)
    {
        printf("Address of %d th array %u \n",i , *(p + i));
        for(j = 0; j < 4; j++)
        {
            printf("arr[%d][%d]=%d\n", i, j, *( *(p + i) + j) );
        }
        printf("\n\n");
    }
    return 0;
}
```

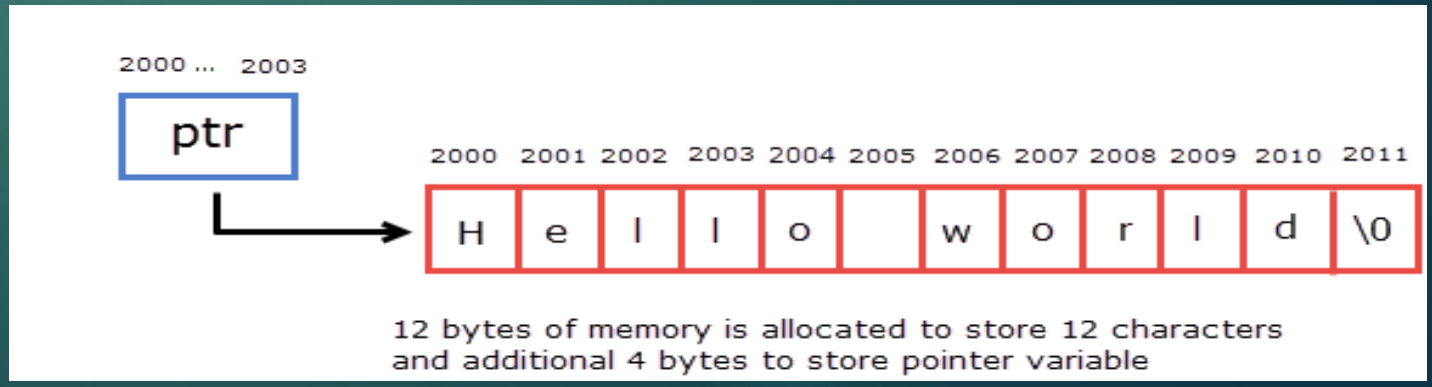
```
Address of 0 th array 2115621456
arr[0][0]=11
arr[0][1]=22
arr[0][2]=33
arr[0][3]=44


Address of 1 th array 2115621472
arr[1][0]=55
arr[1][1]=66
arr[1][2]=77
arr[1][3]=88


Address of 2 th array 2115621488
arr[2][0]=11
arr[2][1]=66
arr[2][2]=77
arr[2][3]=44
```

Pointer and character arrays

char arr[] = "Hello World"; // array version

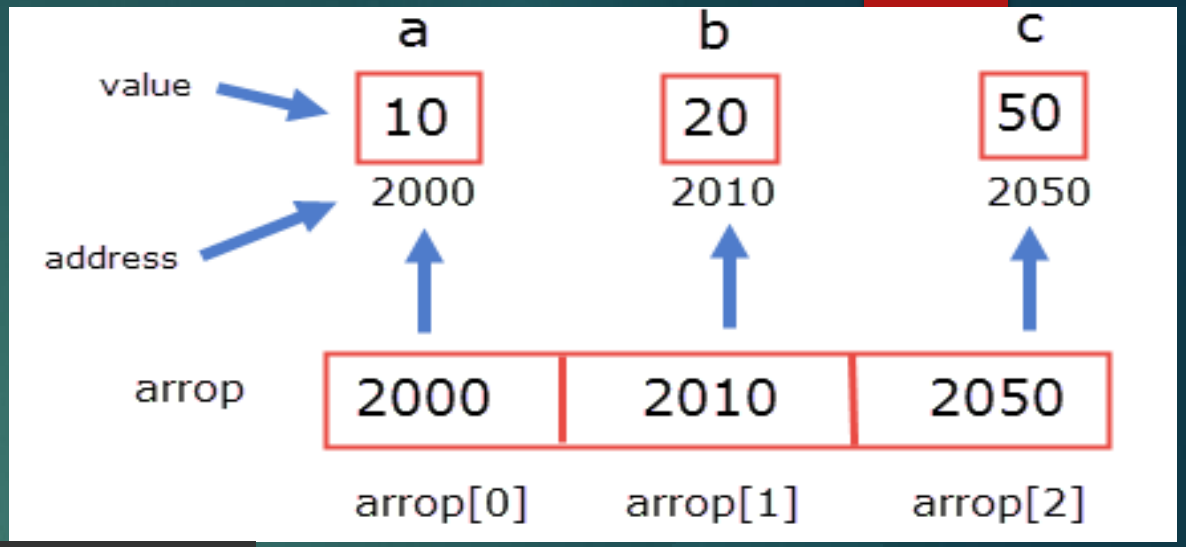


char ptr* = "Hello World"; // pointer version



Array of pointers

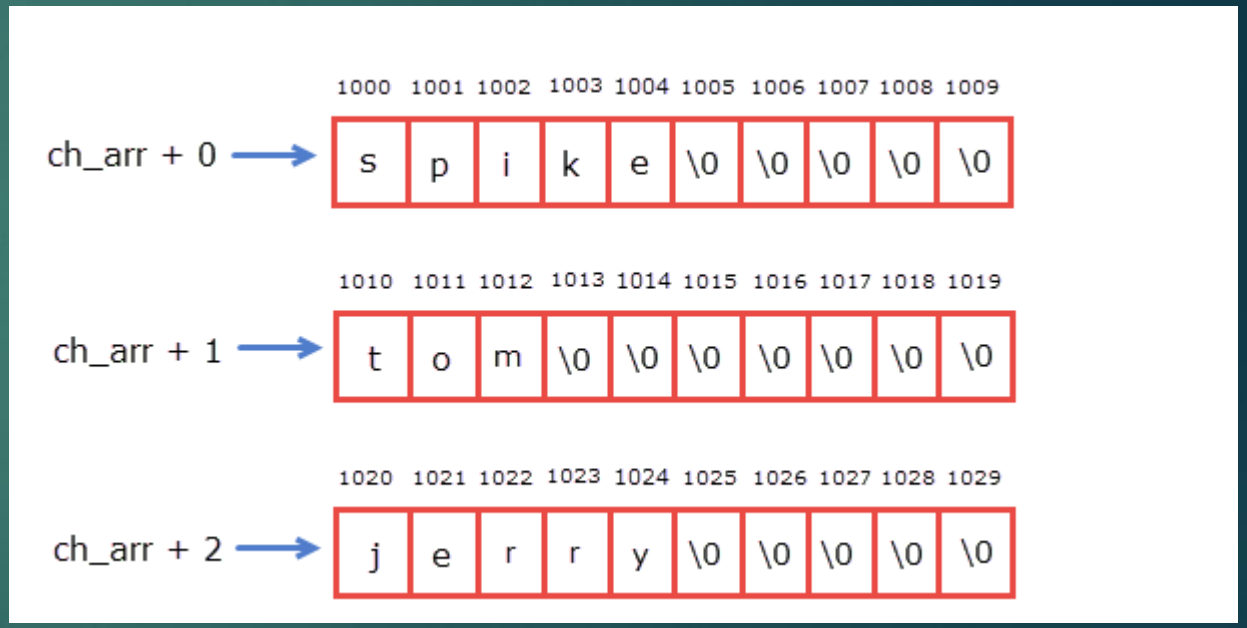
Syntax : datatype *array_name[size];




```c
#include<stdio.h>

int main()
{
    int *arrop[3];
    int a = 10, b = 20, c = 50, i;

    arrop[0] = &a;
    arrop[1] = &b;
    arrop[2] = &c;

    for(i = 0; i < 3; i++)
    {
        printf("Address = %d\t Value = %d\n", arrop[i], *arrop[i]);
    }

    return 0;
}
```

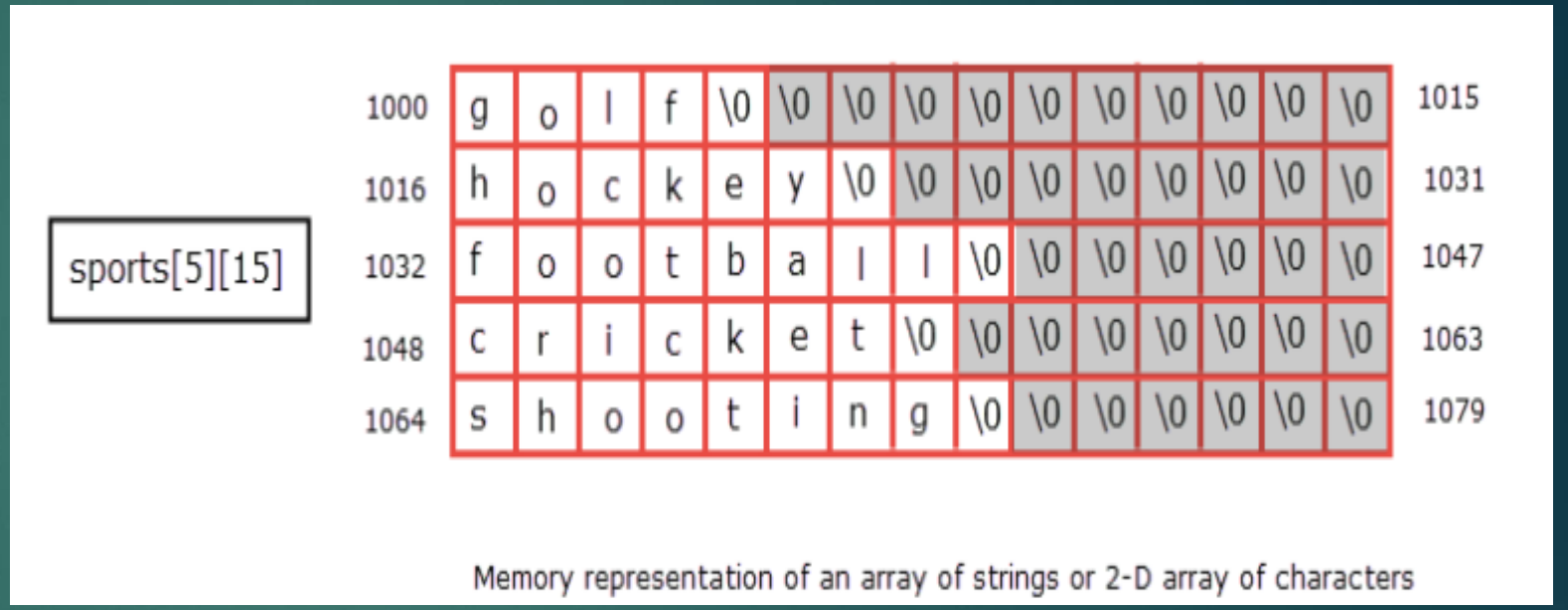
Array of Strings

```
char arr [3][10] = {
                    "spike",
                     "tom",
                     "jerry"
                      };
```



Array of Pointers to Strings

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



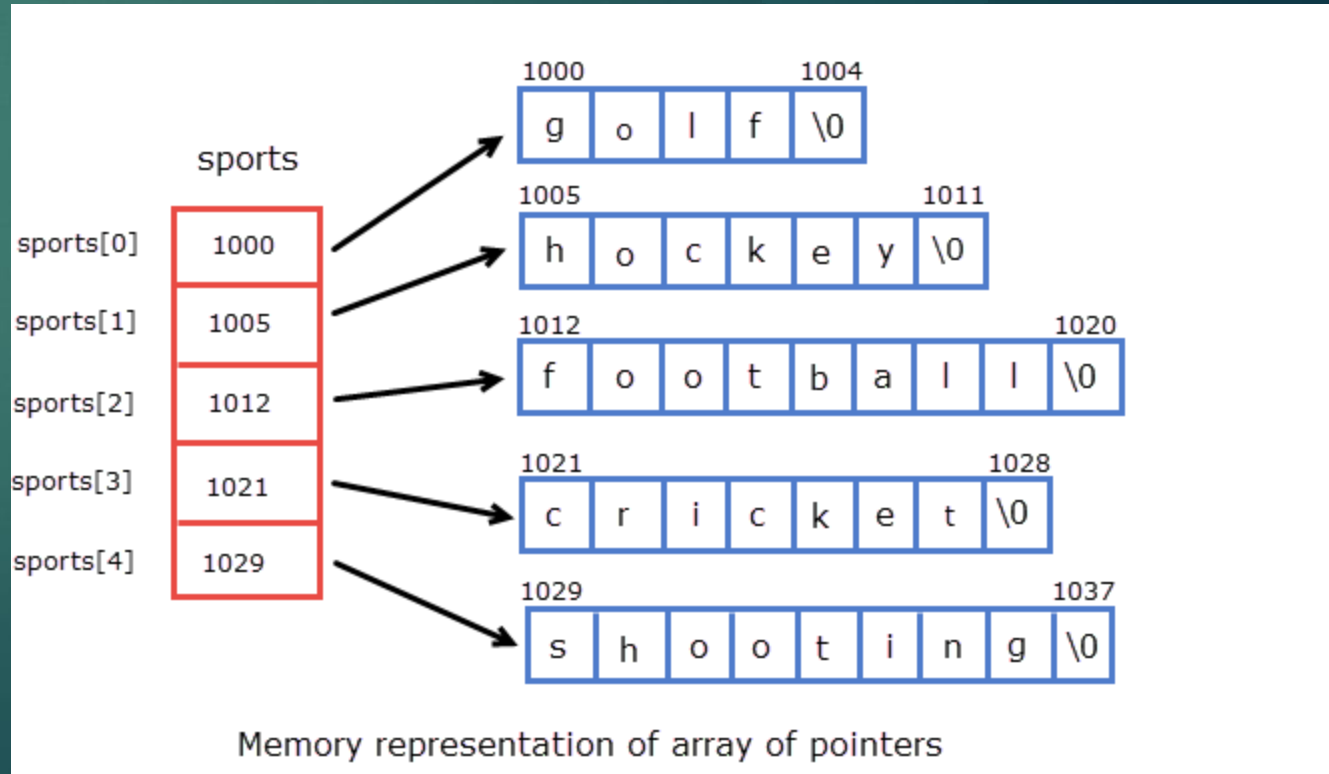
Memory representation of an array of strings or 2-D array of characters

Array of pointers:

- An array of character pointers where each pointer points to the first character of the string or the base address of the string.

- Declaration and Initialization:

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



Memory representation of array of pointers

```c
#include <stdio.h>

int main(void) {
  char *sports[5] = {
                      "golf",
                      "hockey",
                      "football",
                      "cricket",
                      "shooting"
                    };
  int r, c;
  for (r = 0; r < 5; r++) {
    c = 0;
    while(*(sports[r] + c) != '\0') {
      printf("sports[%d] is stored at %d\n and value is %c\n", r,(sports[r] + c),*(sports[r] + c));
      c++;
    }
    printf("\n");
  }

  return 0;
}
```

```
sports[0] is stored at 4196056
 and value is g
sports[0] is stored at 4196057
 and value is o
sports[0] is stored at 4196058
 and value is l
sports[0] is stored at 4196059
 and value is f
```

```
sports[1] is stored at 4196061
 and value is h
sports[1] is stored at 4196062
 and value is o
sports[1] is stored at 4196063
 and value is c
sports[1] is stored at 4196064
 and value is k
sports[1] is stored at 4196065
 and value is e
sports[1] is stored at 4196066
 and value is y
```

```
sports[2] is stored at 4196068
 and value is f
sports[2] is stored at 4196069
 and value is o
sports[2] is stored at 4196070
 and value is o
sports[2] is stored at 4196071
 and value is t
sports[2] is stored at 4196072
 and value is b
sports[2] is stored at 4196073
 and value is a
sports[2] is stored at 4196074
 and value is l
sports[2] is stored at 4196075
 and value is l
```

```
sports[3] is stored at 4196077
 and value is c
sports[3] is stored at 4196078
 and value is r
sports[3] is stored at 4196079
 and value is i
sports[3] is stored at 4196080
 and value is c
sports[3] is stored at 4196081
 and value is k
sports[3] is stored at 4196082
 and value is e
sports[3] is stored at 4196083
 and value is t
```

```
sports[4] is stored at 4196085
 and value is s
sports[4] is stored at 4196086
 and value is h
sports[4] is stored at 4196087
 and value is o
sports[4] is stored at 4196088
 and value is o
sports[4] is stored at 4196089
 and value is t
sports[4] is stored at 4196090
 and value is i
sports[4] is stored at 4196091
 and value is n
sports[4] is stored at 4196092
 and value is g
```