AMIT LEARNING

SESSION 5 ARRAY

Preprocessor

1 Preprocessors

2 Macros

3 File Inclusion

4 Preprocessor Constants

5 Commonly Used Macros

CONTENTS

Preprocessors

Pre-processor commands start with “#” which may optionally be surrounded by spaces and tabs

- The preprocessor allows us to :
  - Include files
  - Define, test and compare constants
  - Write macros
  - Debug

Definitions

- It is a better way to write the literal constants, as it increase the code readability and maintainability.
- #define Phi 3.14
- #define Num_Of_Emp10
- #define Emp_Salary100
- #define Total_Salary Num_Of_Emp* Emp_Salary
- These definitions are simply handled as a text replacement operation done by the preprocessor before the compilation of the file.

Macros

- Text replacement for expressions
- #define MAX(a,b) (a)>(b)?(a):(b)
- No semicolon at the end.
- No space after the macro name and before left parentheses.
- If more than one line is needed, mark each line end with a \ .
- Dummy text replacement.
- Can be used in Wrapping functions.
- Reduce context-switching time but increases code size, but functions, on the other hand, waste some time in context-switch, but reduces code size.
- Both (macros and functions) provide single point of code change.

Macros

- Parameterized Macros
  - Macro that is able to insert given objects into its expansion.This gives the macro some of the power of a
    - function.
    - *#define add(x,y) ((x)+(y))*
- Example:
  - *main()*
  - *{*
    - *int i=S, j=7, result=0; result= add(i, j);*

      printf("result = %d", result);
  - *}*

Macros

- Using Macros to Define Used Constants to improve readability and maintainability.
  - *#define MAX_STUDENTS_COUNT   20*
  - *#define PAI   3.14F*
  - *#define INTER_PLATFORM_MESSAGE_1   OxAA*
- Example:
  - *main()*
  - *{*
    - *int Students_List [MAX_STUDENTS_COUNT];*
  - *}*

Macros

- Writing Multiline Macro

```
#define MULT(x,y)\
  int i =0;\
int j =0;\
  i=x; \
  j=y; \
  x=i*j
```

Macros

Use Macros to replace tiny functions or a bulk of code that are called or used so many times like inside a loop to save calling time and improve performance.

Always surround all symbols inside your macro with parenthesis to avoid any future mistakes due to Using this macro.

Macros

- Example:

```
 #define MULT(x, y) x*y
  inti=2, j=3;
  result= MULT(i+5, j);
```

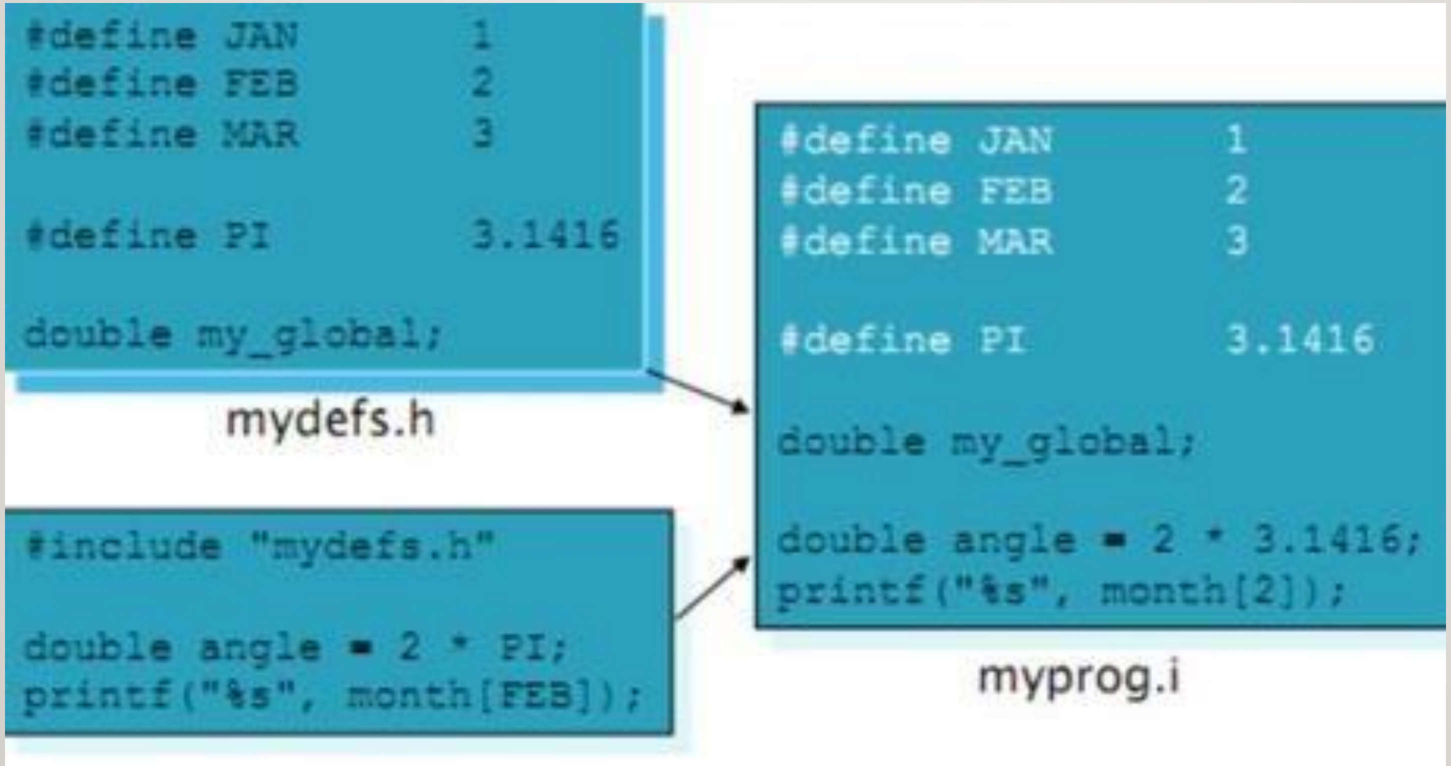
- You expect result= 21
- Actual result result= 17

Lab #1

- Implement a program that takes 2 input numbers i.e. x and y; and do the following:
  - Check value of given x is less that a predefined value
  - Check value of given y is less that a predefined value
  - Print the result of x + y
  - Print the result of x * y
  - Print the result of (x + y) * x
- Use Macros to hold the predefined values of x and y
- Use Macro to hold addition formula
- Use Macro to hold multiplication formula

File inclusion

The #include directive causes the pre-processor to "edit in" the entire contents of another file



```
#define JAN        1
#define FEB        2
#define MAR        3

#define PI         3.1416

double my_global;
```

mydefs.h

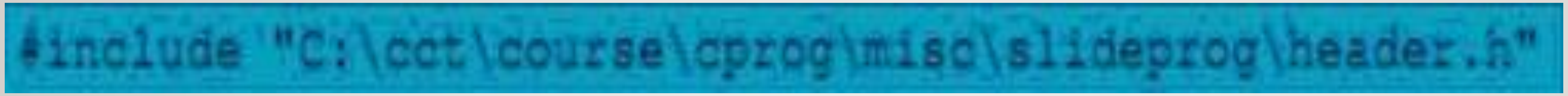
```
#include "mydefs.h"

double angle = 2 * PI;
printf("%s", month[FEB]);
```

```
#define JAN        1
#define FEB        2
#define MAR        3

#define PI         3.1416

double my_global;

double angle = 2 * 3.1416;
printf("%s", month[2]);
```

myprog.i

File inclusion

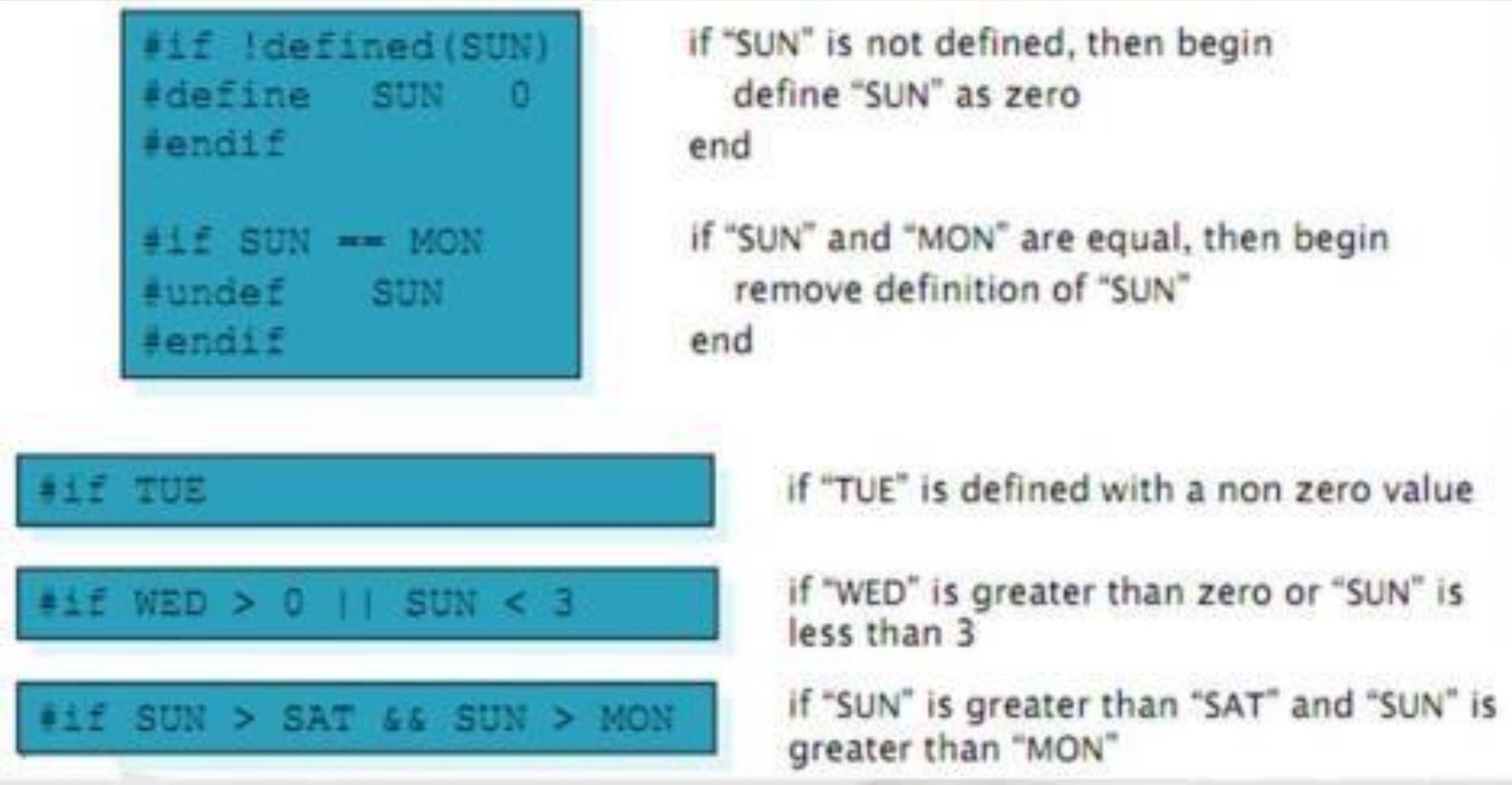
- Pathnames:
  - Full pathnames may be used, although this is not recommended



```
#include "C:\cct\course\cprog\misc\slideprog\header.h"
```

Preprocessor Constants

- Preprocessor Constants :
  - Constants may be created, tested and removed



```
#if !defined(SUN)
#define   SUN   0
#endif
```

if "SUN" is not defined, then begin
  define "SUN" as zero
end

```
#if SUN == MON
#undef    SUN
#endif
```

if "SUN" and "MON" are equal, then begin
  remove definition of "SUN"
end

```
#if TUE
```

if "TUE" is defined with a non zero value

```
#if WED > 0 || SUN < 3
```

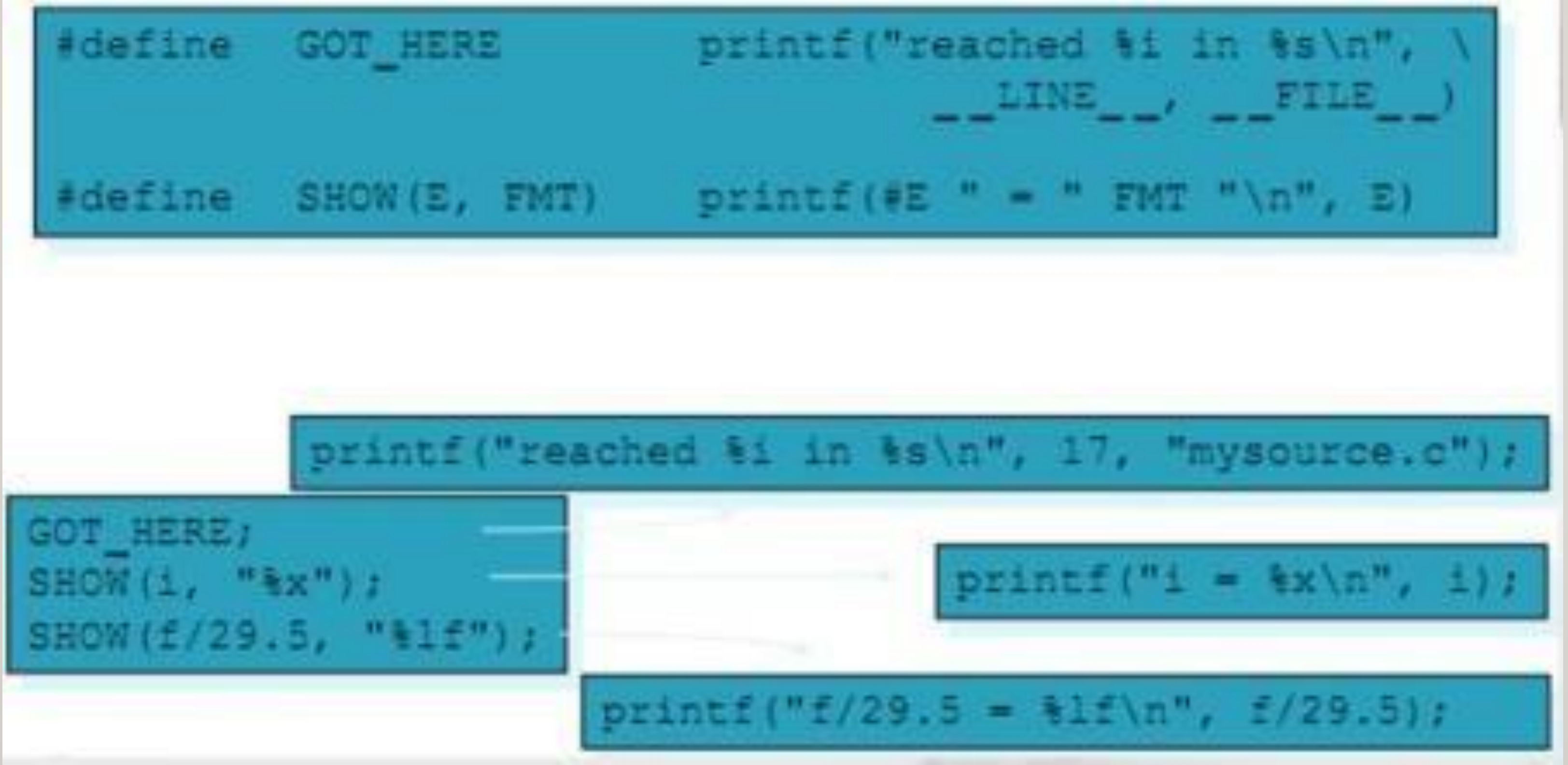
if "WED" is greater than zero or "SUN" is less than 3

```
#if SUN > SAT && SUN > MON
```

if "SUN" is greater than "SAT" and "SUN" is greater than "MON"

Preprocessor Constants

- Debugging
  - Several extra features make the pre-processor an indispensable debugging tool



```
#define   GOT_HERE        printf("reached %i in %s\n", \
                                    __LINE__, __FILE__)

#define   SHOW(E, FMT)    printf(#E " = " FMT "\n", E)
```

```
printf("reached %i in %s\n", 17, "mysource.c");
```

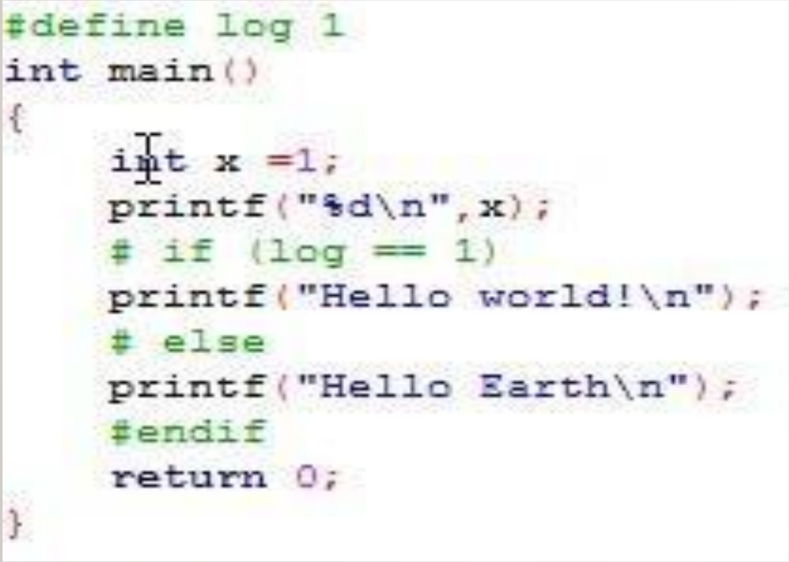
```
GOT_HERE;
SHOW(i, "%x");
SHOW(f/29.5, "%lf");
```

```
printf("i = %x\n", i);
```

```
printf("f/29.5 = %lf\n", f/29.5);
```

The #if Statement

- It is a preprocessor keyword.
- Normally it is used to add or delete some parts of code based on a certain condition.
- The #if only deals with definition as it is evaluated before the compilation of the program

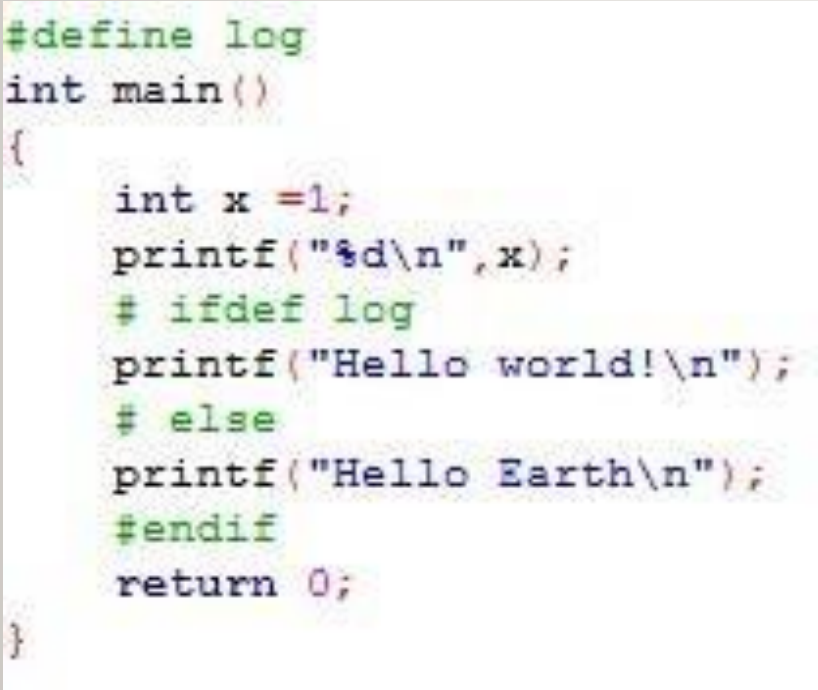


```
#define log 1
int main()
{
    int x =1;
    printf("%d\n",x);
    # if (log == 1)
    printf("Hello world!\n");
    # else
    printf("Hello Earth\n");
    #endif
    return 0;
}
```

"Hello World" will be printed

The #ifdef Statement

- It is a preprocessor keyword.
- Normally it is used to add or delete some parts of code based on the existence of a certain definition.

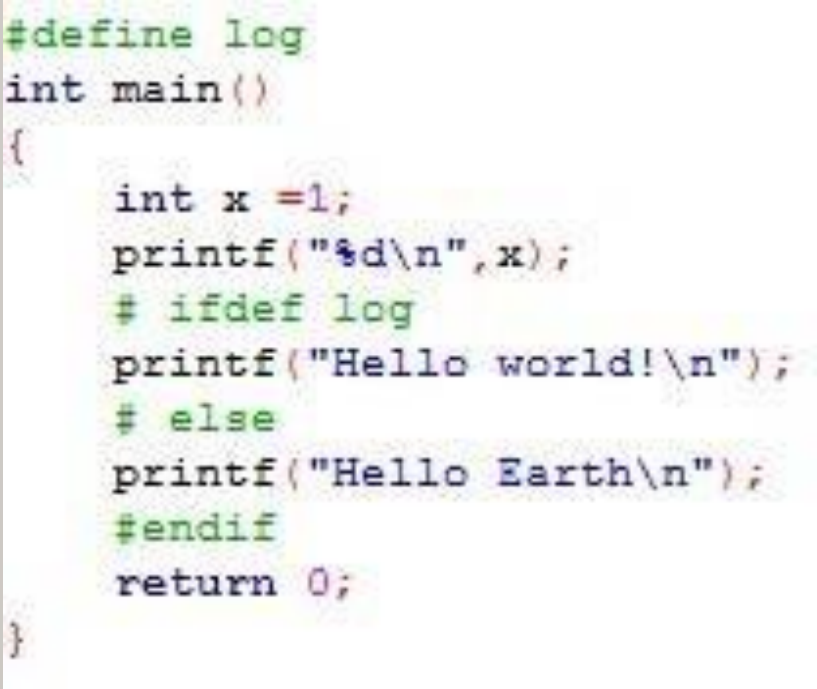


```
#define log
int main()
{
    int x =1;
    printf("%d\n",x);
    # ifdef log
    printf("Hello world!\n");
    # else
    printf("Hello Earth\n");
    #endif
    return 0;
}
```

“Hello World” will be printed

The #ifndef Statement

- It is a preprocessor keyword.
- Normally it is used to add or delete some parts of code based on the inexistence of a certain definition.



```
#define log
int main()
{
    int x =1;
    printf("%d\n",x);
    # ifdef log
    printf("Hello world!\n");
    # else
    printf("Hello Earth\n");
    #endif
    return 0;
}
```

“Hello Earth” will be printed

Lab #2

- Create DebugHeader.h file, with a predefined macro i.e. IS_DEBUG_MODE having a value defining whether the software runs in debug mode or not
- In the main file, include DebugHeader.h
- In the main function, use the preprocessor keyword #if to print whether the software running in debug mode or not

Lab #2

- Create DebugHeader.h file, with a predefined macro i.e. IS_DEBUG_MODE having a value defining whether the software runs in debug mode or not
- In the main file, include DebugHeader.h
- In the main function, use the preprocessor keyword #if to print whether the software running in debug mode or not

Lab # 3

- Create DebugHeader.h file, with apredefined macro“DEBUG_MODE_ACTIVE”
- If DEBUG_MODE_ACTIVE is defined, this indicated that debug mode is running
- In the main file, include DebugHeader.h
- In the main function, use the preprocessor keyword #ifdef to print whether the software running in debug mode or not

Commonly Used Macros

- Set Bit in a byte:

  X=X|=(1<<n)

- Clear Bit in a byte:

  X=X &(~(1<<n))

- Toggle Bit in a byte:

  X=X^(1<<n)

Commonly Used Macros

- Return max out of two numbers

```
        #define MAX(A,B) ((A)> (B) ? (A) : (B))
```

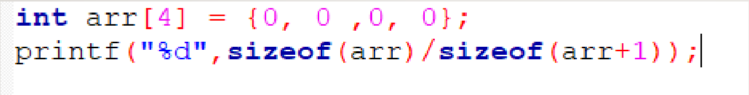
- Return min out of two numbers

```
        #define MIN(X,Y) ((X) < (Y) ? (X) : (Y))
```

```
void arr[100];
```

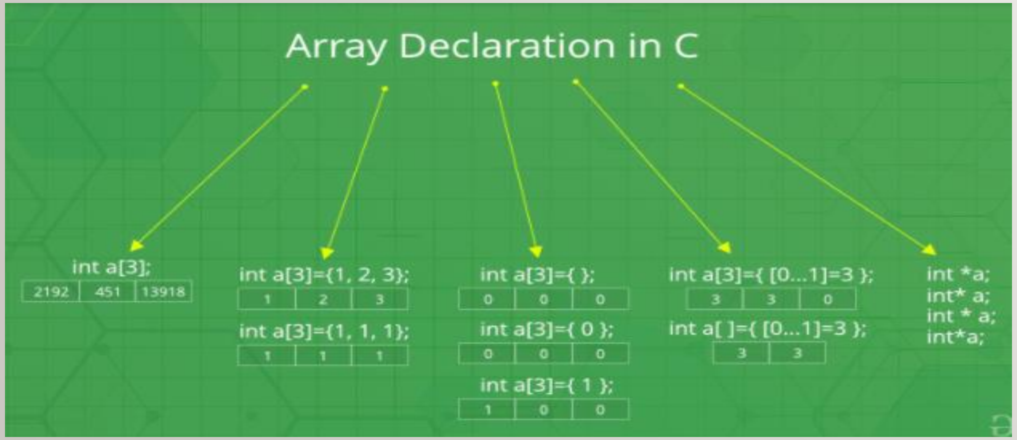
WHAT IS ARRAY

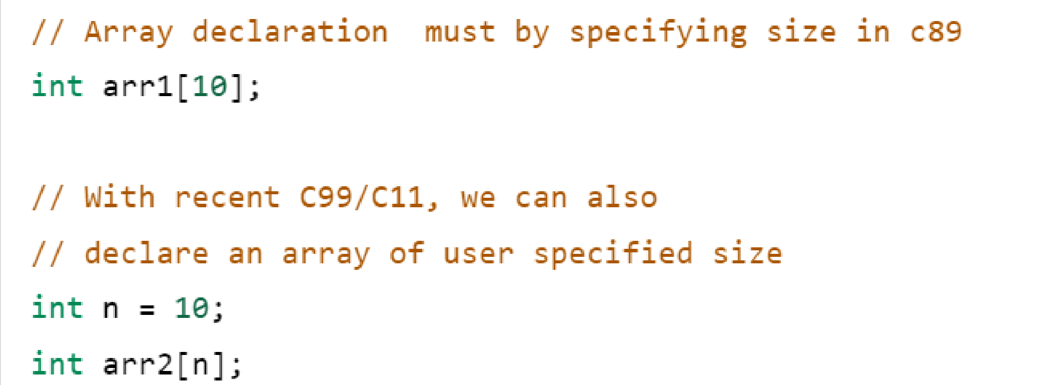
- An array in C or be it in any programming language is a collection of similar data items stored at contiguous memory locations and elements can be accessed randomly using indices of an array. They can be used to store collection of primitive data types such as int, float, double, char, etc of any particular type. Excpt void arr[] make compiler error.
- Calculate number of element:



```
int arr[4] = {0, 0 ,0, 0};
printf("%d",sizeof(arr)/sizeof(arr+1));
```

- **Why do we need arrays?**
  We can use normal variables (v1, v2, v3, ..) when we have a small number of objects, but if we want to store a large number of instances, it becomes difficult to manage them with normal variables. The idea of an array is to represent many instances in one variable.
- **Array declaration in C:**





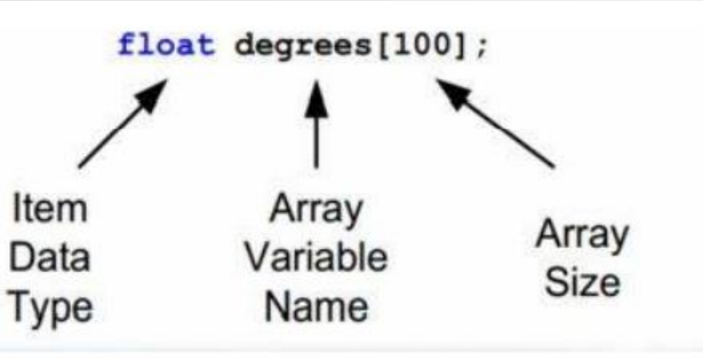
```
// Array declaration  must by specifying size in c89
int arr1[10];

// With recent C99/C11, we can also
// declare an array of user specified size
int n = 10;
int arr2[n];
```

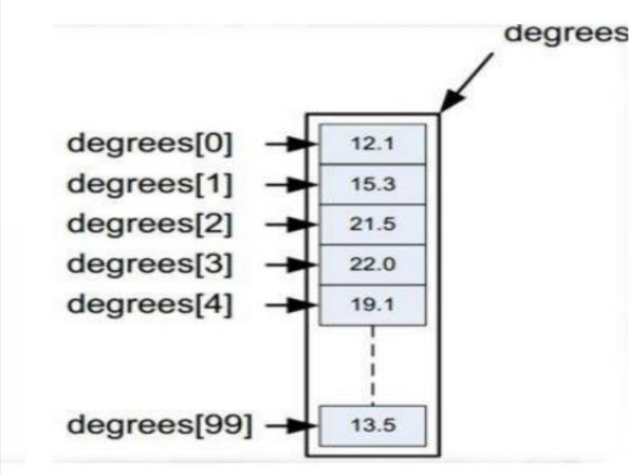
- ➢ Advantages of an Array in C:

1.Random access of elements using array index.
2.Use of less line of code as it creates a single array of multiple elements.
3.Easy access to all the elements.
4.Traversal through the array becomes easy using a single loop.
5.Sorting becomes easy as it can be accomplished by writing less line of code.

- ➢ Disadvantages of an Array in C:

1.Allows a fixed number of elements to be entered which is decided at the time of declaration. Unlike a linked list, an array in C is not dynamic.
2.Insertion and deletion of elements can be costly since the elements are needed to be managed in accordance with the new memory allocation.



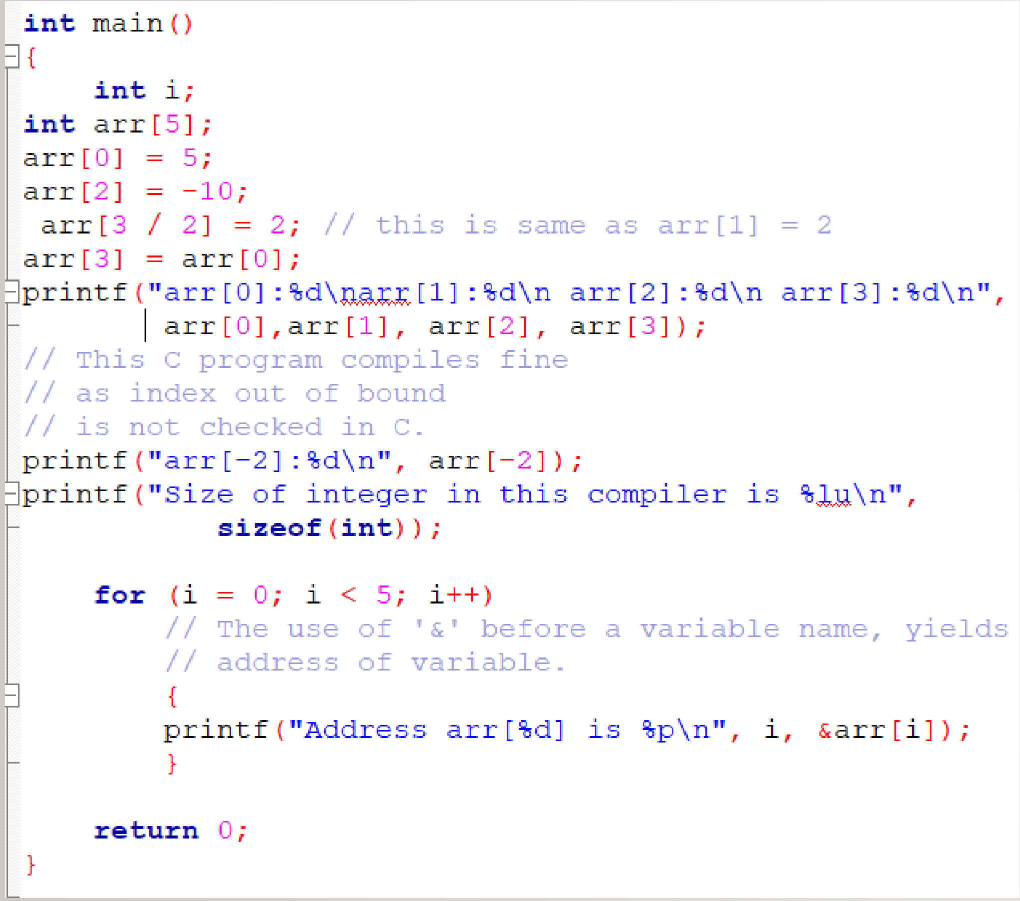


WHAT IS POINTER

1 Introduction to pointer

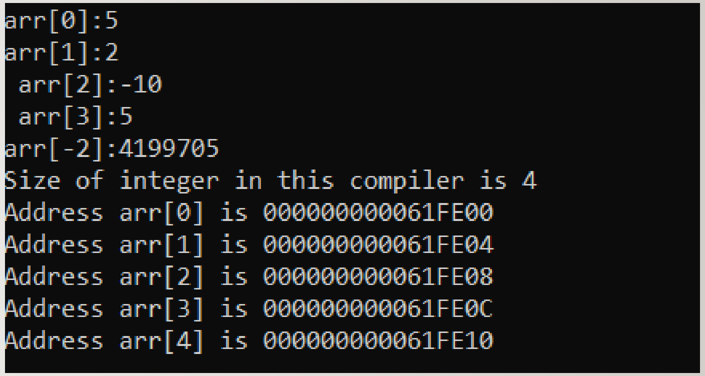
1.1 Introduction to pointer

CODE



```
int main()
{
    int i;
int arr[5];
arr[0] = 5;
arr[2] = -10;
 arr[3 / 2] = 2; // this is same as arr[1] = 2
arr[3] = arr[0];
printf("arr[0]:%d\narr[1]:%d\n arr[2]:%d\n arr[3]:%d\n",
        arr[0],arr[1], arr[2], arr[3]);
// This C program compiles fine
// as index out of bound
// is not checked in C.
printf("arr[-2]:%d\n", arr[-2]);
printf("Size of integer in this compiler is %lu\n",
           sizeof(int));

    for (i = 0; i < 5; i++)
        // The use of '&' before a variable name, yields
        // address of variable.
        {
        printf("Address arr[%d] is %p\n", i, &arr[i]);
        }

    return 0;
}
```

OUTPUT



```
arr[0]:5
arr[1]:2
 arr[2]:-10
 arr[3]:5
arr[-2]:4199705
Size of integer in this compiler is 4
Address arr[0] is 000000000061FE00
Address arr[1] is 000000000061FE04
Address arr[2] is 000000000061FE08
Address arr[3] is 000000000061FE0C
Address arr[4] is 000000000061FE10
```

DESCRIPTION

- In this example discuss array.

- In c programming language, arrays are classified into **two types**:
- **Single Dimensional Array (One Dimensional Array).**
  - The C99 standard allows variable sized arrays .But, unlike the normal arrays, variable sized arrays cannot be initialized.
- **Multi Dimensional Array(2D) and (3D).**

1 Introduction to Array

1.3 Array Arithmetic and related to pointer.

- Most of the time, pointer and array accesses can be treated as acting the same, the major exceptions being:

1) the sizeof operator

sizeof(array) returns the amount of memory used by all elements in array.

sizeof(pointer) only returns the amount of memory used by the pointer variable itself.

2) the & operator

&array is an alias for &array[0] and returns the address of the first element in array.

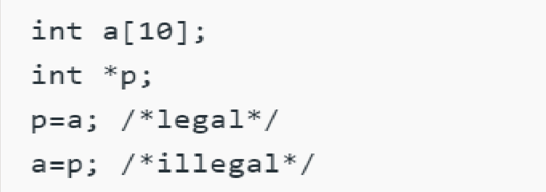
&pointer returns the address of pointer.

3) a string literal initialization of a character array

char array[] = “abc” sets the first four elements in array to ‘a’, ‘b’, ‘c’, and ‘\0’.

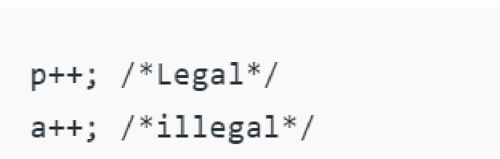
char *pointer = “abc” sets pointer to the address of the “abc” string (which may be stored in read-only memory and thus unchangeable).

4) Pointer variable can be assigned a value whereas array variable cannot be.



```
int a[10];
int *p;
p=a; /*legal*/
a=p; /*illegal*/
```

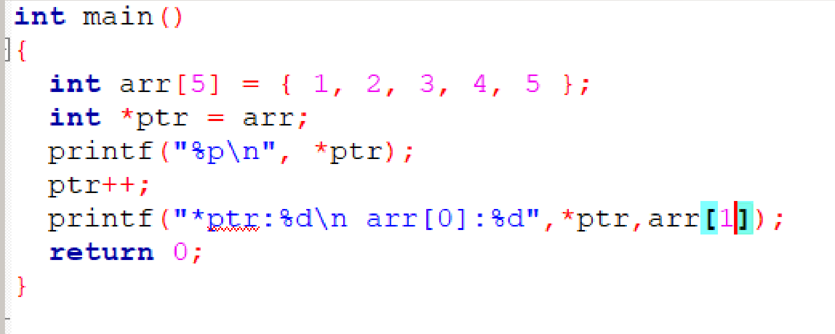
5) Arithmetic on pointer variable is allowed.



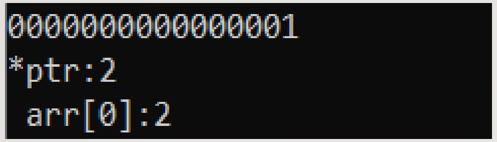
```
p++; /*Legal*/
a++; /*illegal*/
```

1 Introduction to Array

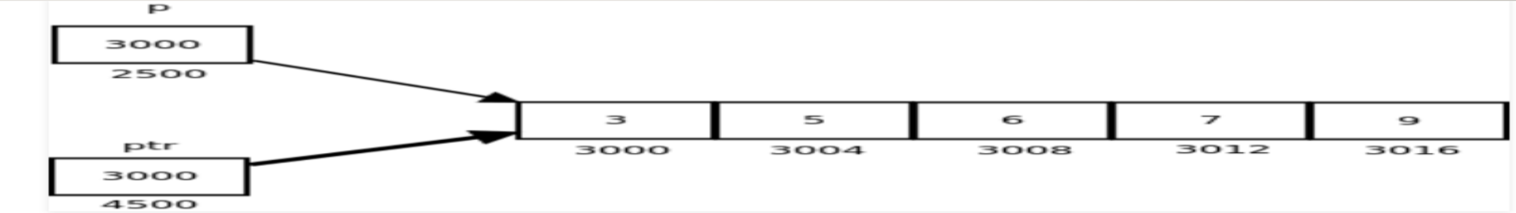
1.3 Array Arithmetic and related to pointer.



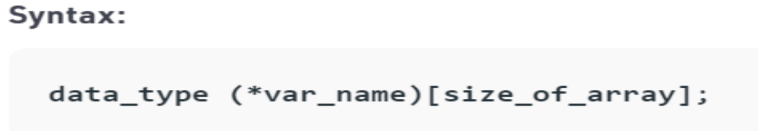
```
int main()
{
    int arr[5] = { 1, 2, 3, 4, 5 };
    int *ptr = arr;
    printf("%p\n", *ptr);
    ptr++;
    printf("*ptr:%d\n arr[0]:%d",*ptr,arr[1]);
    return 0;
}
```



```
0000000000000001
*ptr:2
 arr[0]:2
```



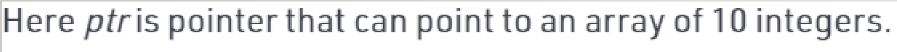
- In this program, we have a pointer ptr that points to the 0th element of the array. Similarly, we can also declare a pointer that can point to whole array instead of only one element of the array. This pointer is useful when talking about multidimensional arrays.



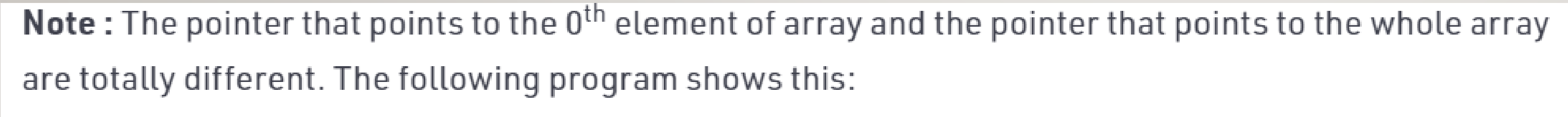
Syntax:

```
data_type (*var_name)[size_of_array];
```

- Example:
- Int(*ptr)[10];



Here *ptr* is pointer that can point to an array of 10 integers.

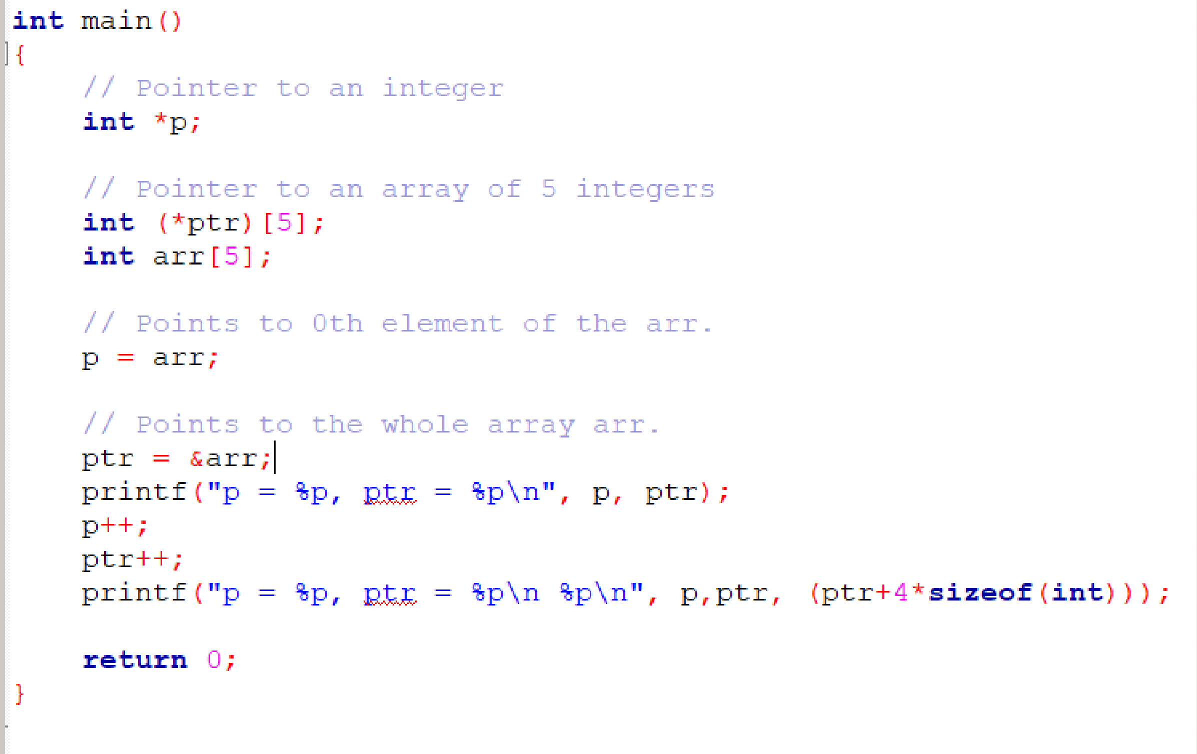


Note : The pointer that points to the 0th element of array and the pointer that points to the whole array are totally different. The following program shows this:

1 Introduction to pointer

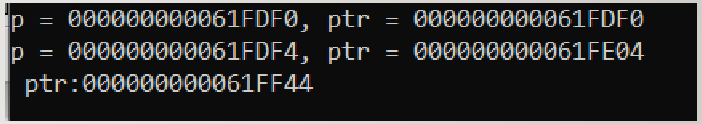
1.3 Array Arithmetic and related to pointer.

CODE



```
int main()
{
    // Pointer to an integer
    int *p;

    // Pointer to an array of 5 integers
    int (*ptr)[5];
    int arr[5];

    // Points to 0th element of the arr.
    p = arr;

    // Points to the whole array arr.
    ptr = &arr;
    printf("p = %p, ptr = %p\n", p, ptr);
    p++;
    ptr++;
    printf("p = %p, ptr = %p\n %p\n", p,ptr,  (ptr+4*sizeof(int)));

    return 0;
}
```

OUTPUT



```
p = 000000000061FDF0, ptr = 000000000061FDF0
p = 000000000061FDF4, ptr = 000000000061FE04
 ptr:000000000061FF44
```

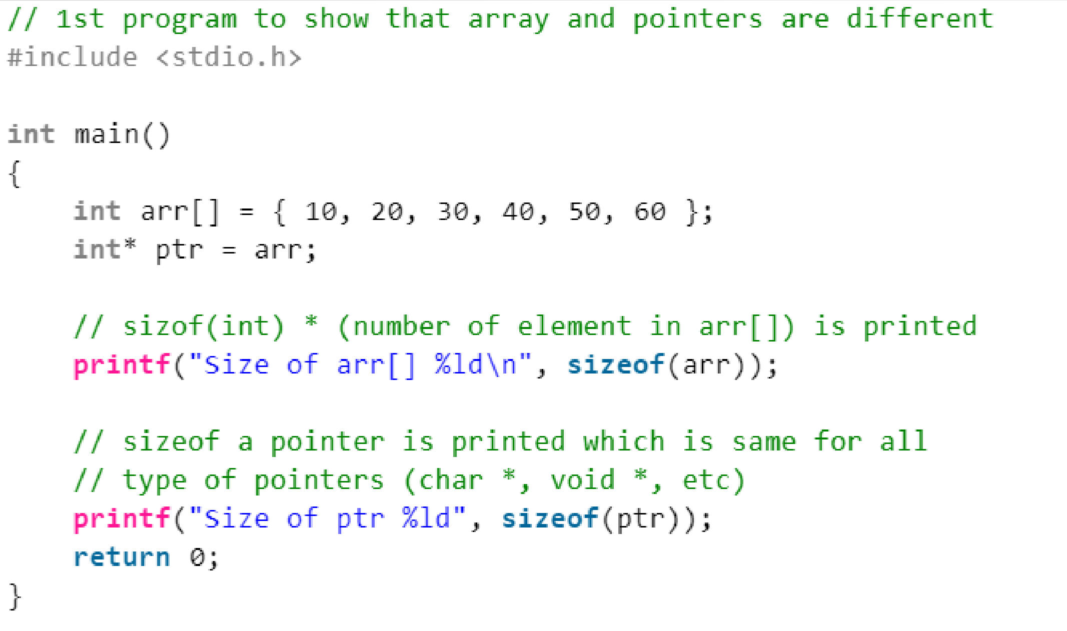
DESCRIPTION

- In this example discuss pointer to integer and pointer to array of 5 integer.

1 Introduction to pointer

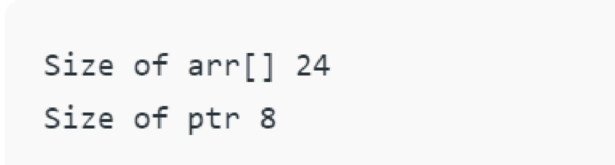
1.3 Array Arithmetic and related to pointer.

CODE



```
// 1st program to show that array and pointers are different
#include <stdio.h>

int main()
{
    int arr[] = { 10, 20, 30, 40, 50, 60 };
    int* ptr = arr;

    // sizof(int) * (number of element in arr[]) is printed
    printf("Size of arr[] %ld\n", sizeof(arr));

    // sizeof a pointer is printed which is same for all
    // type of pointers (char *, void *, etc)
    printf("Size of ptr %ld", sizeof(ptr));
    return 0;
}
```

OUTPUT



```
Size of arr[] 24
Size of ptr 8
```

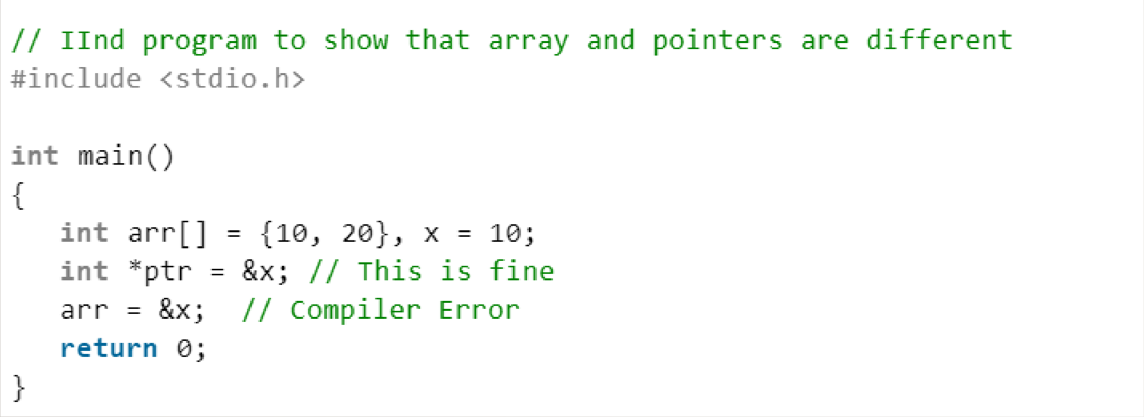
DESCRIPTION

- In this example discuss size of array and size of pointer.

1 Introduction to pointer

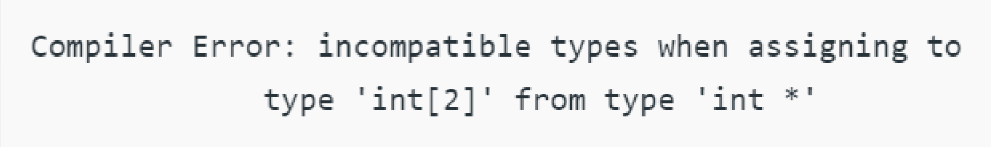
1.3 Array Arithmetic and related to pointer.

CODE



```
// IInd program to show that array and pointers are different
#include <stdio.h>

int main()
{
   int arr[] = {10, 20}, x = 10;
   int *ptr = &x; // This is fine
   arr = &x;  // Compiler Error
   return 0;
}
```

OUTPUT



```
Compiler Error: incompatible types when assigning to
        type 'int[2]' from type 'int *'
```

DESCRIPTION

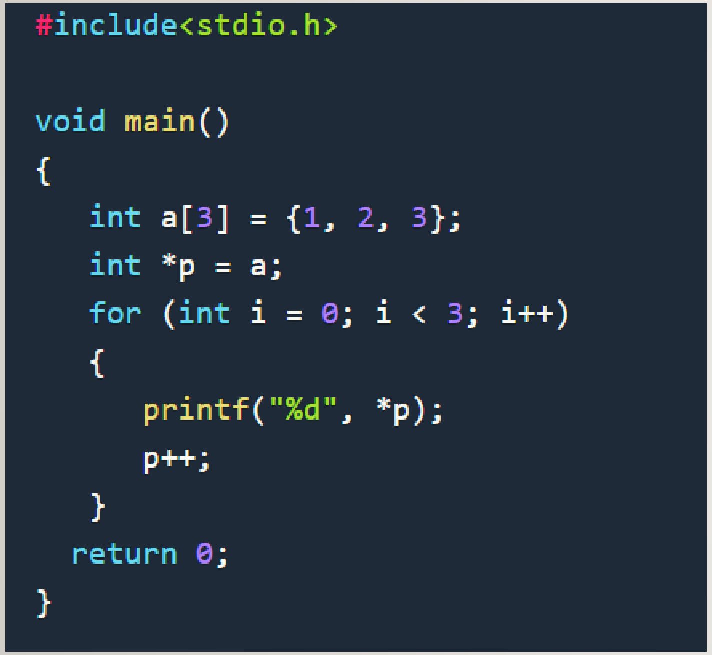
- In this example discuss you cannot assign anything to array name.

WHAT IS POINTER

1 Introduction to pointer

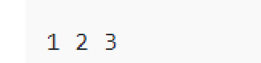
1.3 Array Arithmetic and related to pointer.

CODE



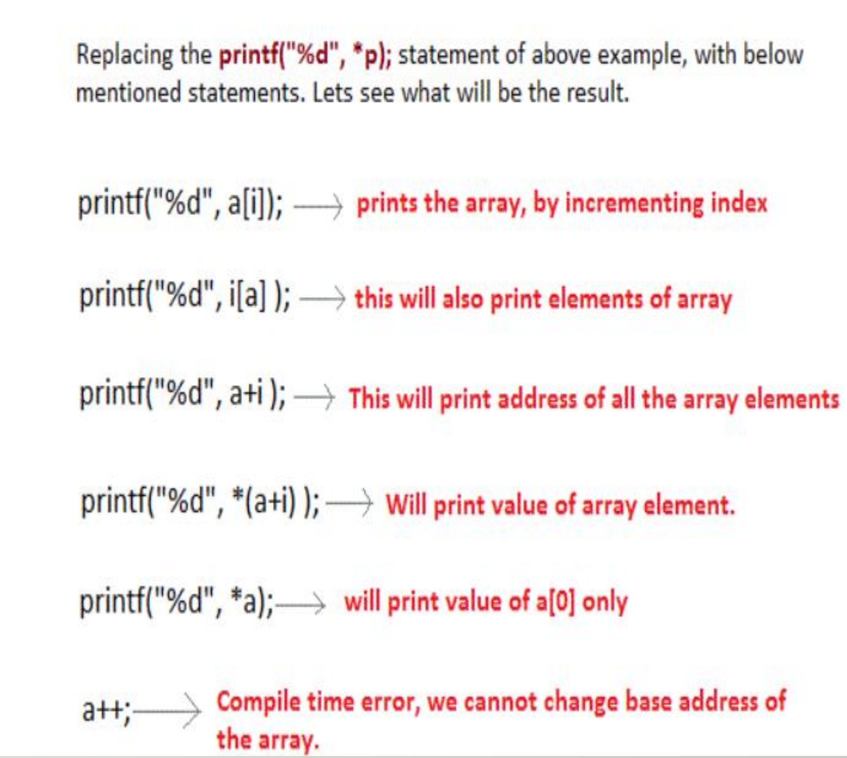
```c
#include<stdio.h>

void main()
{
   int a[3] = {1, 2, 3};
   int *p = a;
   for (int i = 0; i < 3; i++)
   {
      printf("%d", *p);
      p++;
   }
  return 0;
}
```

OUTPUT



```
1 2 3
```

DESCRIPTION



Replacing the **printf("%d", *p);** statement of above example, with below mentioned statements. Lets see what will be the result.

printf("%d", a[i]); → prints the array, by incrementing index

printf("%d", i[a]); → this will also print elements of array

printf("%d", a+i); → This will print address of all the array elements

printf("%d", *(a+i)); → Will print value of array element.

printf("%d", *a); → will print value of a[0] only

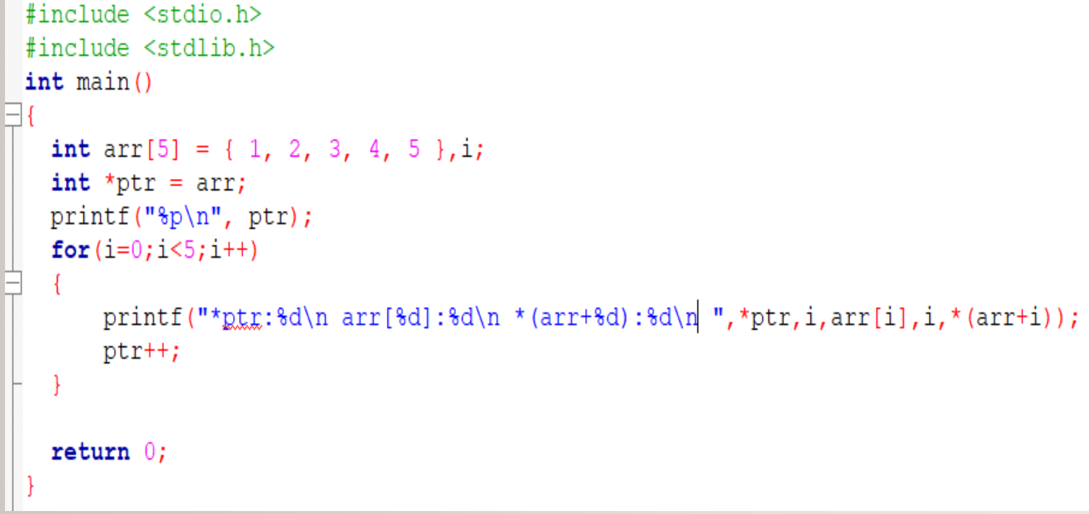
a++; → Compile time error, we cannot change base address of the array.

WHAT IS POINTER

1 Introduction to pointer

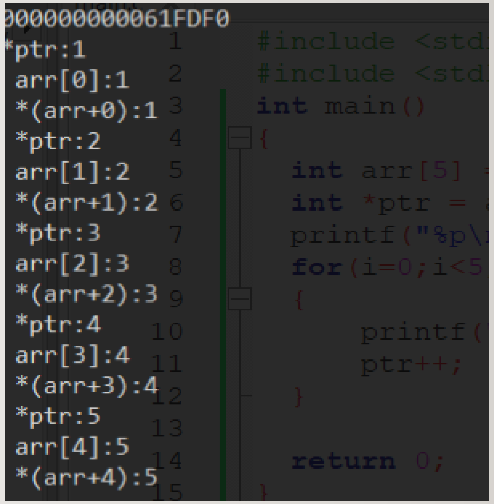
1.3 Array Arithmetic and related to pointer.

CODE



```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int arr[5] = { 1, 2, 3, 4, 5 },i;
  int *ptr = arr;
  printf("%p\n", ptr);
  for(i=0;i<5;i++)
  {
      printf("*ptr:%d\n arr[%d]:%d\n *(arr+%d):%d\n ",*ptr,i,arr[i],i,*(arr+i));
      ptr++;
  }

  return 0;
}
```

OUTPUT



```
000000000061FDF0
*ptr:1
 arr[0]:1
 *(arr+0):1
 *ptr:2
 arr[1]:2
 *(arr+1):2
 *ptr:3
 arr[2]:3
 *(arr+2):3
 *ptr:4
 arr[3]:4
 *(arr+3):4
 *ptr:5
 arr[4]:5
 *(arr+4):5
```

DESCRIPTION

- In this example discuss you cannot assign anything to array name.

1 Introduction to Array

1.4 Passing array to function.

- **Function call by value in C:**
- **By pass element of array.**
- **How to pass an array by value in C ?**

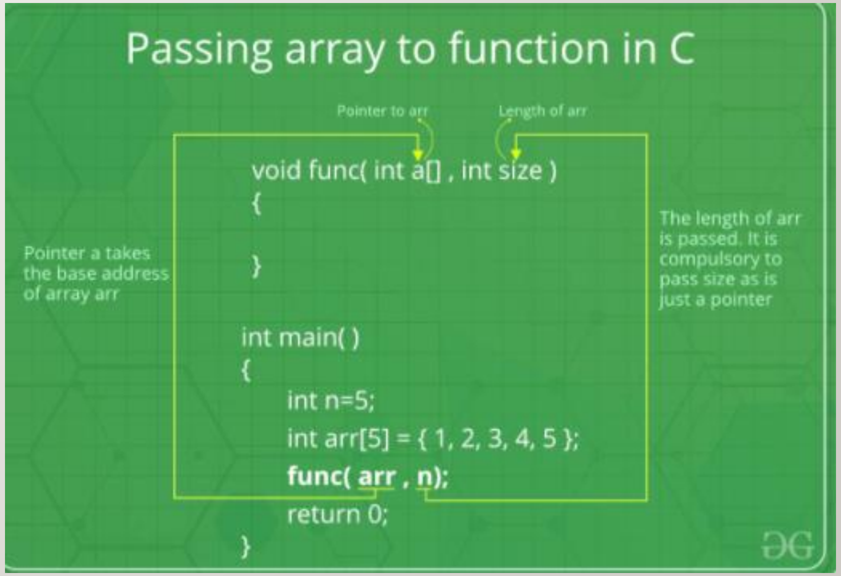
In C, array name represents address and when we pass an array, we actually pass address and the parameter receiving function always accepts them as pointers **How to pass array by value, i.e., how to make sure that we have a new copy of array when we pass it to function?**

This can be done by wrapping the array in a structure and creating a variable of type of that structure and assigning values to that array.

Check this link:

https://www.geeksforgeeks.org/pass-array-value-c/

- **Function call by reference in C**
- **Array to pass function call by reference.**



Passing array to function in C

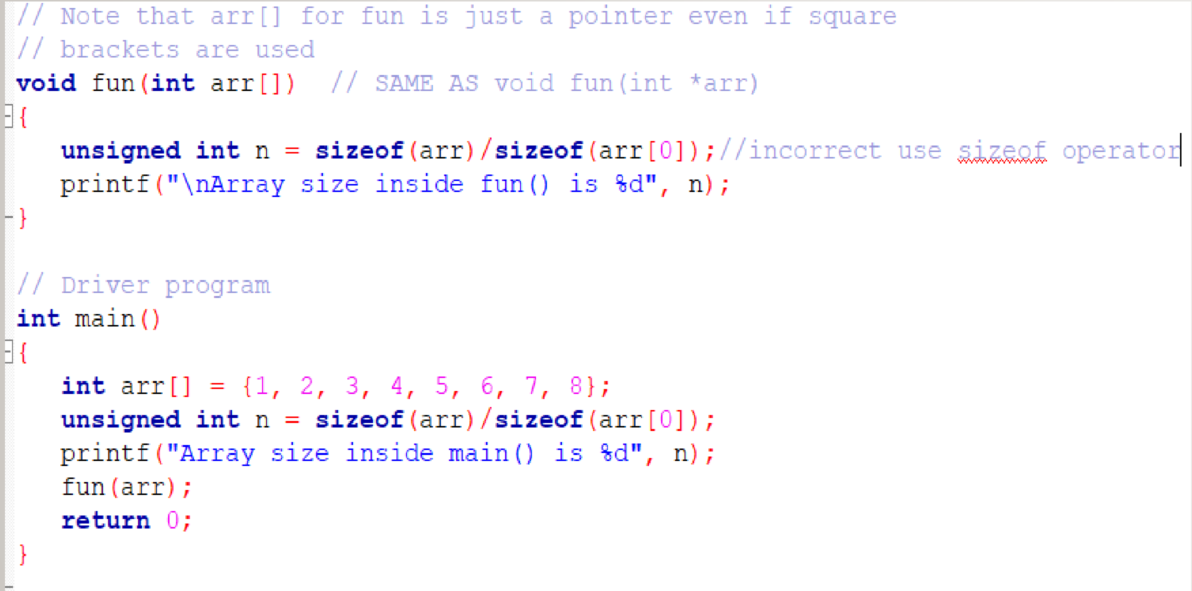
Pointer to arr
Length of arr
Pointer a takes the base address of array arr
The length of arr is passed. It is compulsory to pass size as is just a pointer

```
void func( int a[] , int size )
{

}

int main( )
{
    int n=5;
    int arr[5] = { 1, 2, 3, 4, 5 };
    func( arr , n);
    return 0;
}
```

1 Introduction to pointer

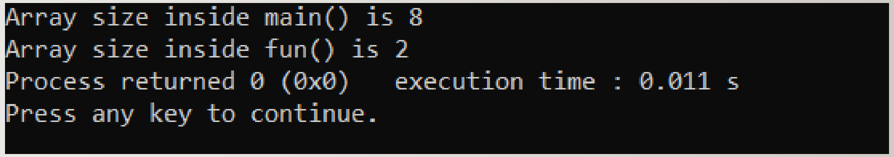
1.4 Passing array to function(call by reference)

CODE



```c
// Note that arr[] for fun is just a pointer even if square
// brackets are used
void fun(int arr[])  // SAME AS void fun(int *arr)
{
   unsigned int n = sizeof(arr)/sizeof(arr[0]);//incorrect use sizeof operator
   printf("\nArray size inside fun() is %d", n);
}

// Driver program
int main()
{
   int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
   unsigned int n = sizeof(arr)/sizeof(arr[0]);
   printf("Array size inside main() is %d", n);
   fun(arr);
   return 0;
}
```

OUTPUT



```
Array size inside main() is 8
Array size inside fun() is 2
Process returned 0 (0x0)   execution time : 0.011 s
Press any key to continue.
```

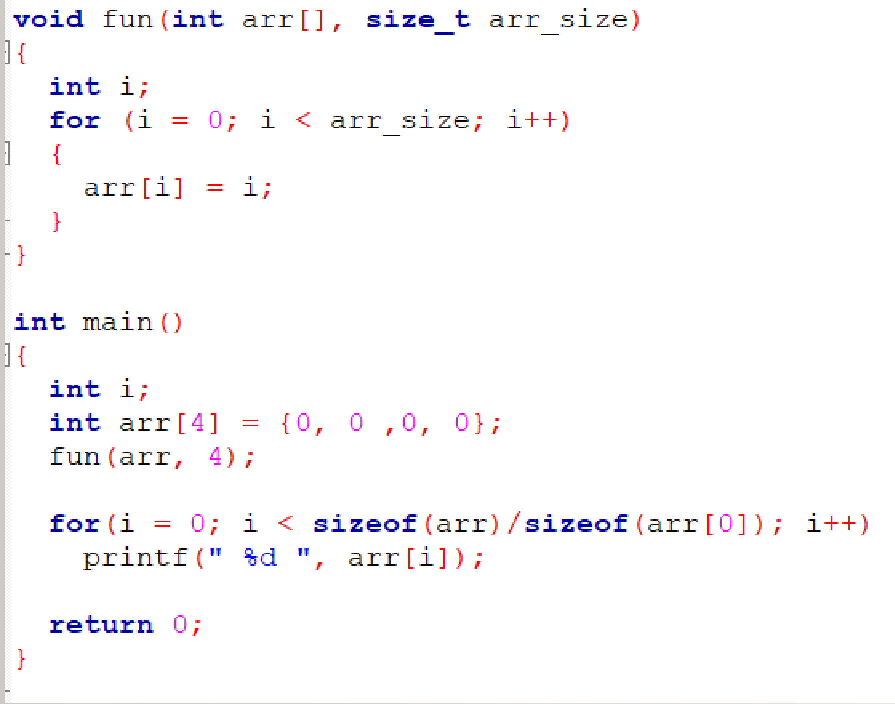
DESCRIPTION

- In this example discuss you call by refernce.

1 Introduction to pointer

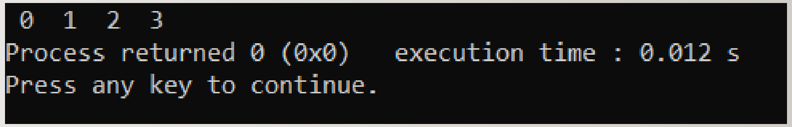
1.4 Passing array to function(call by reference)

CODE



```
void fun(int arr[], size_t arr_size)
{
   int i;
   for (i = 0; i < arr_size; i++)
   {
     arr[i] = i;
   }
}

int main()
{
   int i;
   int arr[4] = {0, 0 ,0, 0};
   fun(arr, 4);

   for(i = 0; i < sizeof(arr)/sizeof(arr[0]); i++)
     printf(" %d ", arr[i]);

   return 0;
}
```

OUTPUT



```
 0  1  2  3
Process returned 0 (0x0)   execution time : 0.012 s
Press any key to continue.
```

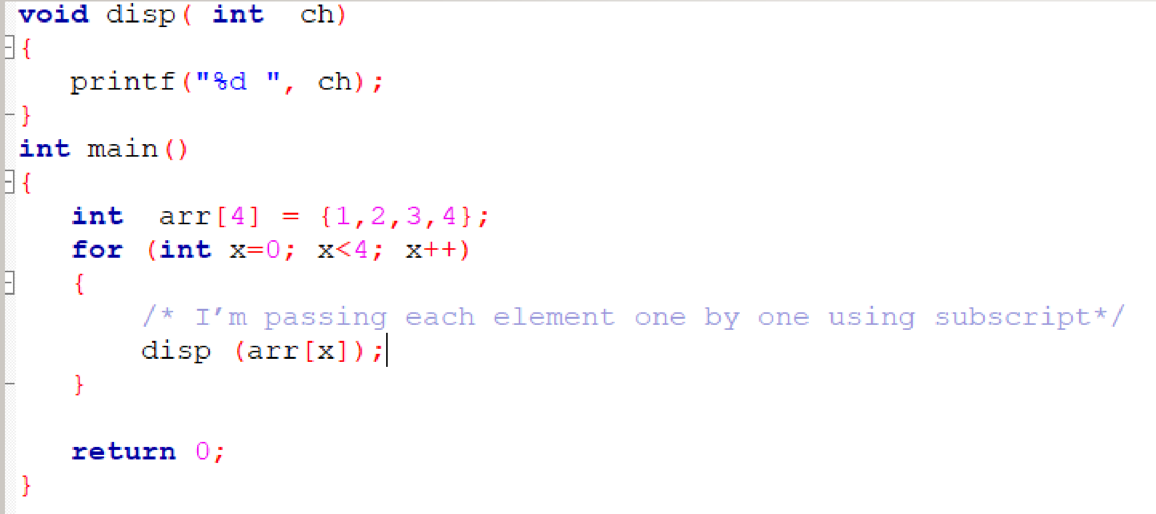
DESCRIPTION

- In this example discuss you cannot assign anything to array name.

1 Introduction to pointer

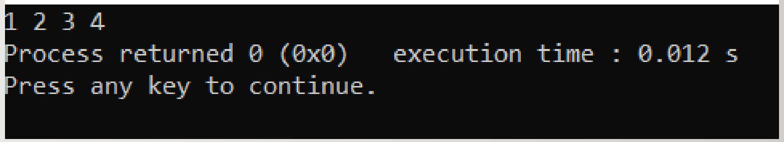
1.4 Passing array to function(call by value)

CODE



```
void disp( int  ch)
{
    printf("%d ", ch);
}
int main()
{
    int  arr[4] = {1,2,3,4};
    for (int x=0; x<4; x++)
    {
        /* I’m passing each element one by one using subscript*/
        disp (arr[x]);
    }

    return 0;
}
```

OUTPUT



```
1 2 3 4
Process returned 0 (0x0)   execution time : 0.012 s
Press any key to continue.
```

DESCRIPTION

- In this example discuss you cannot assign anything to array name.

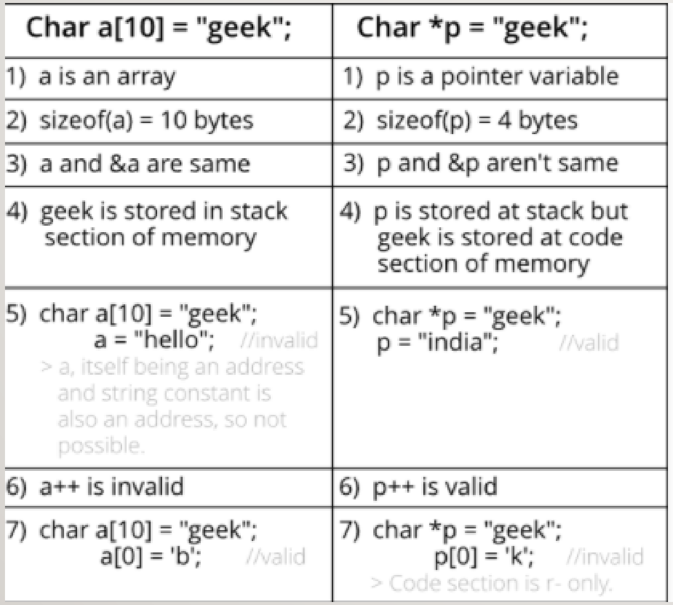
1.5 Storage for Strings in C

➢ What is string?

- String is a set of several consecutive characters; each character can be represented in C language by a (char) data type.

- This means that string value can be represented two ways:

1. Array of character.
2. Pointer of character.



Char a[10] = "geek";	Char *p = "geek";
1) a is an array	1) p is a pointer variable
2) sizeof(a) = 10 bytes	2) sizeof(p) = 4 bytes
3) a and &a are same	3) p and &p aren't same
4) geek is stored in stack section of memory	4) p is stored at stack but geek is stored at code section of memory

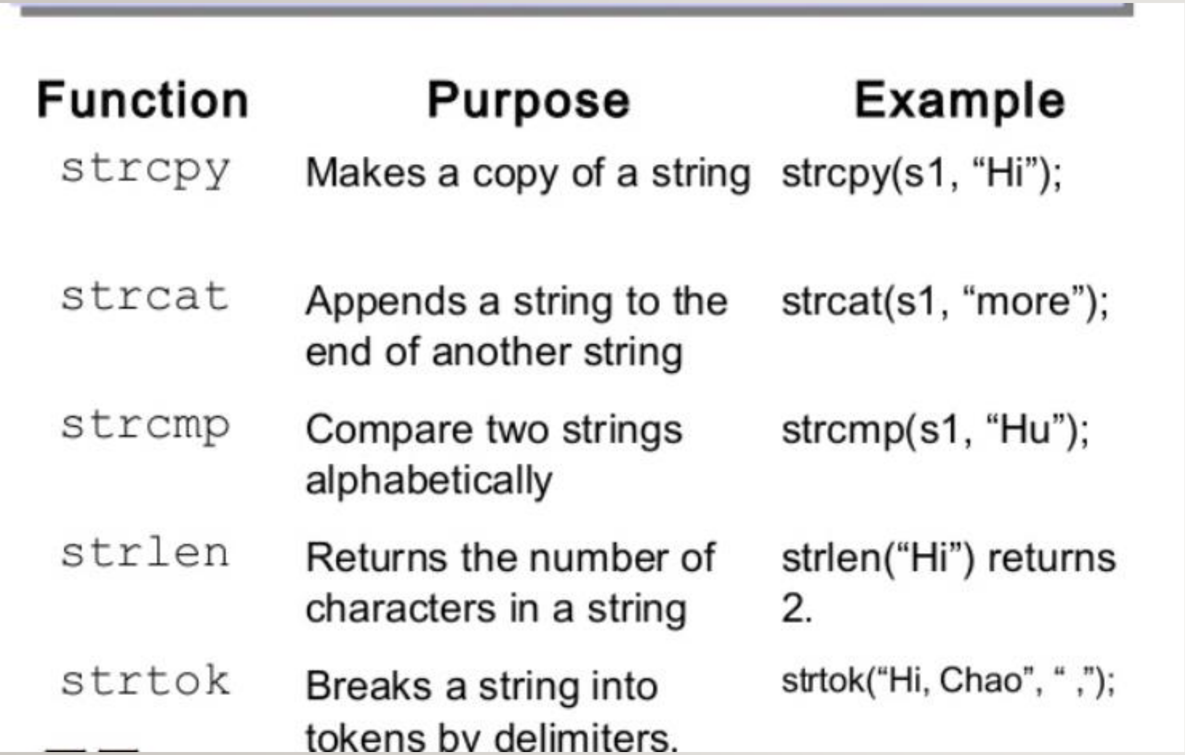
5) char a[10] = "geek";
a = "hello"; //invalid
> a, itself being an address and string constant is also an address, so not possible.
	5) char *p = "geek";
	p = "india"; //valid

6) a++ is invalid	6) p++ is valid

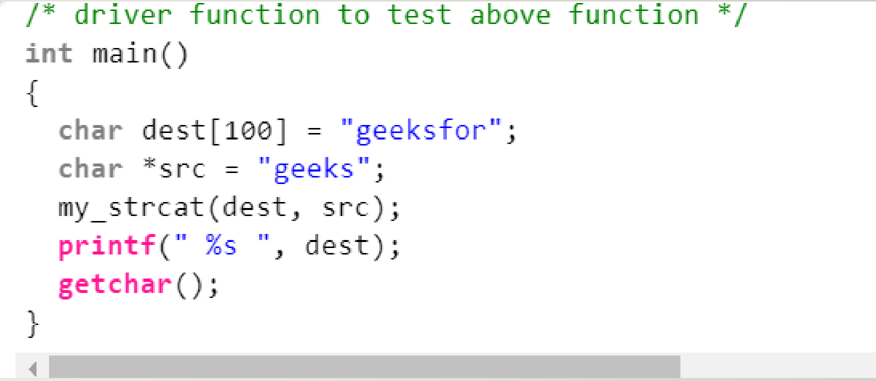
7) char a[10] = "geek";
a[0] = 'b'; //valid
	7) char *p = "geek";
	p[0] = 'k'; //invalid
	> Code section is r- only.

1 Introduction to Array

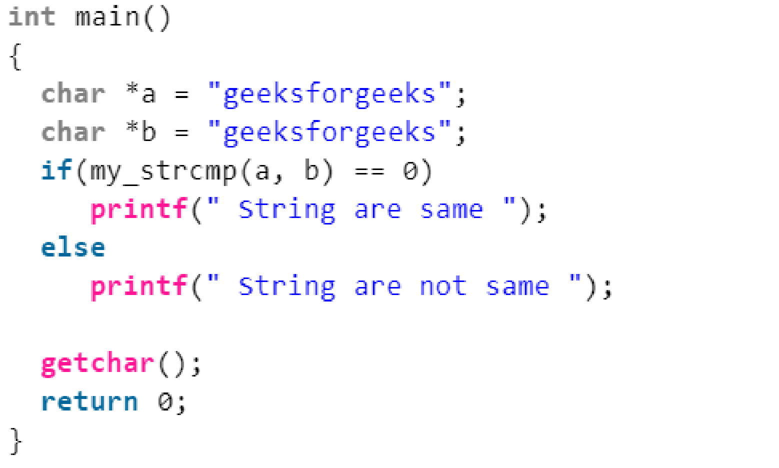
1.5 Storage for Strings in C(string library)



Function	Purpose	Example
strcpy	Makes a copy of a string	strcpy(s1, “Hi”);
strcat	Appends a string to the end of another string	strcat(s1, “more”);
strcmp	Compare two strings alphabetically	strcmp(s1, “Hu”);
strlen	Returns the number of characters in a string	strlen(“Hi”) returns 2.
strtok	Breaks a string into tokens bv delimiters.	strtok(“Hi, Chao”, “ ,”);



```
/* driver function to test above function */
int main()
{
  char dest[100] = "geeksfor";
  char *src = "geeks";
  my_strcat(dest, src);
  printf(" %s ", dest);
  getchar();
}
```

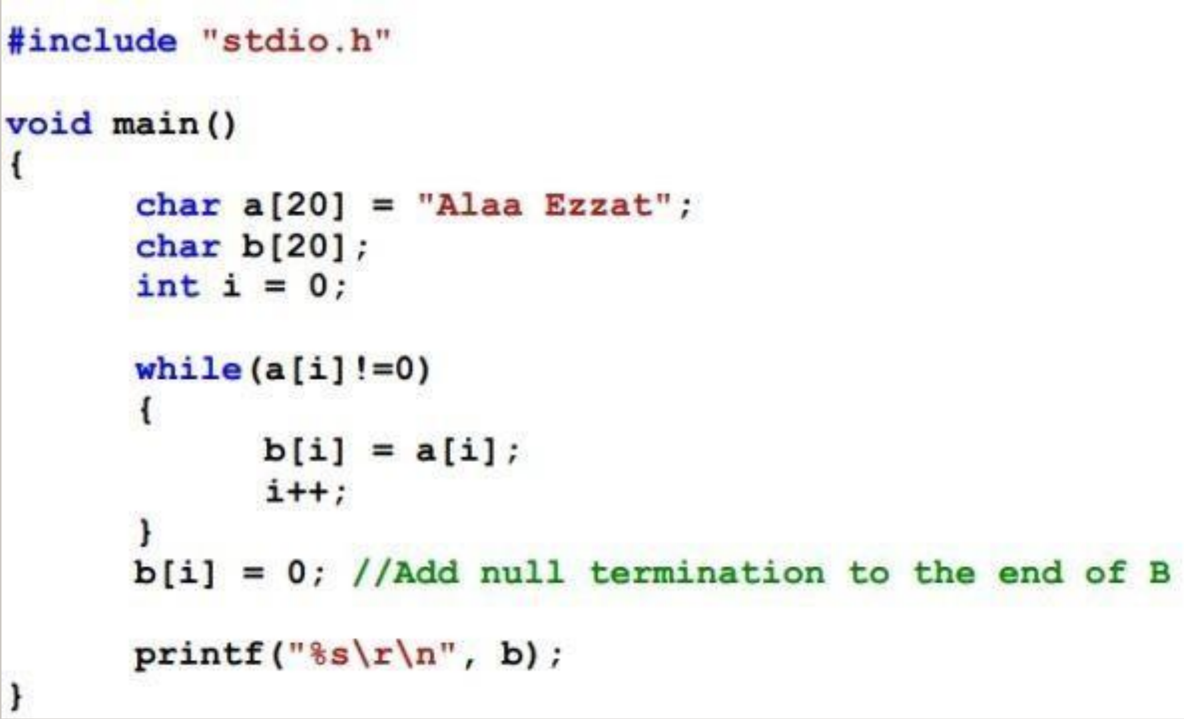


```
int main()
{
  char *a = "geeksforgeeks";
  char *b = "geeksforgeeks";
  if(my_strcmp(a, b) == 0)
     printf(" String are same ");
  else
     printf(" String are not same ");

  getchar();
  return 0;
}
```

1 Introduction to Array

1.5 Storage for Strings in C(string library) strcpy function

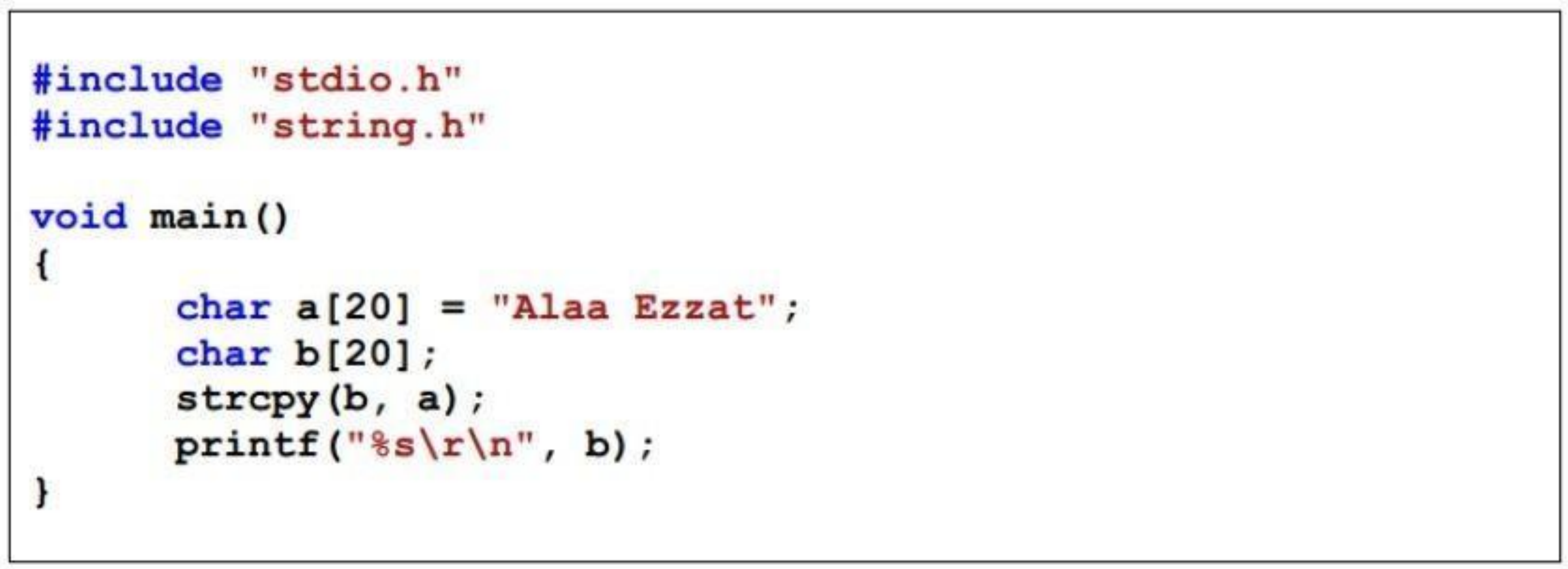
- There is another solution to above problem using **strcpy** function. **strcpy** takes both the destination and the source strings and performs the coping operation internally.



```
#include "stdio.h"

void main()
{
      char a[20] = "Alaa Ezzat";
      char b[20];
      int i = 0;

      while(a[i]!=0)
      {
            b[i] = a[i];
            i++;
      }
      b[i] = 0; //Add null termination to the end of B

      printf("%s\r\n", b);
}
```

1 Introduction to Array

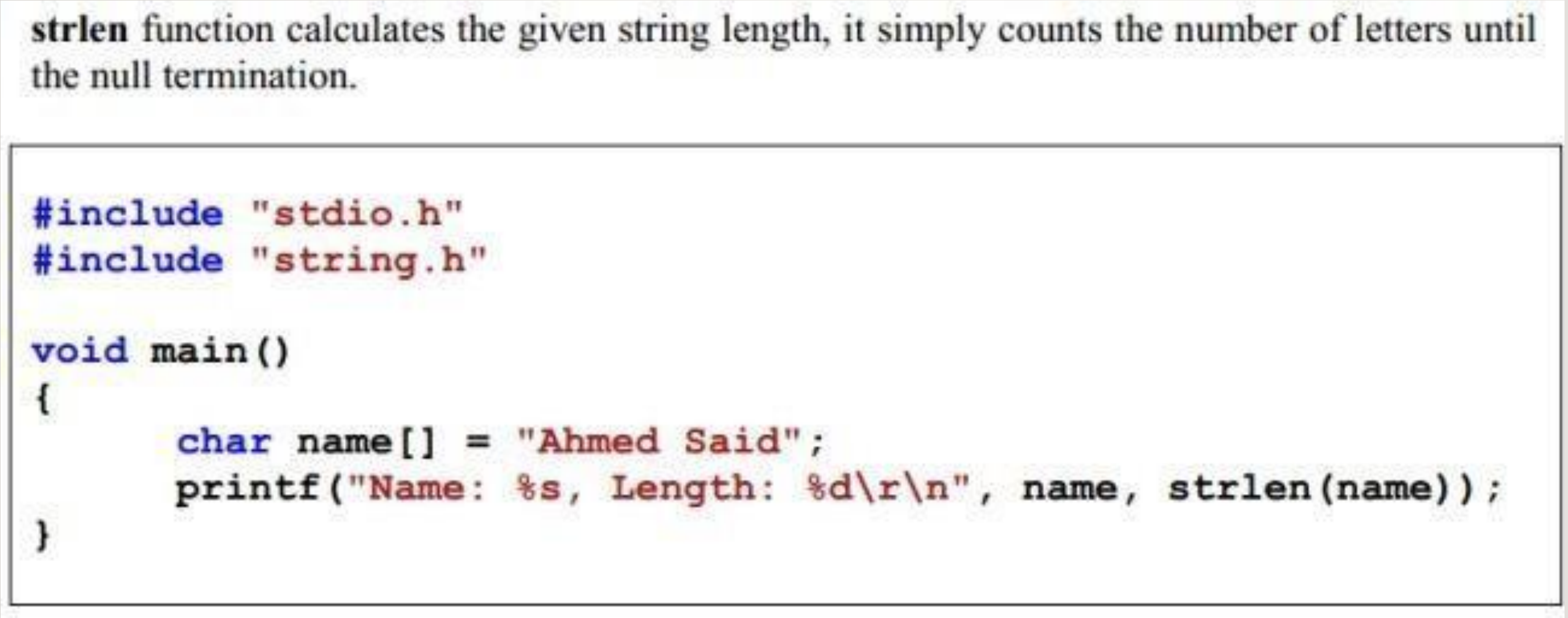
1.5 Storage for Strings in C(string library) strcpy function



```
#include "stdio.h"
#include "string.h"

void main()
{
        char a[20] = "Alaa Ezzat";
        char b[20];
        strcpy(b, a);
        printf("%s\r\n", b);
}
```

1 Introduction to Array

1.5 Storage for Strings in C(string library) strlen function



strlen function calculates the given string length, it simply counts the number of letters until the null termination.

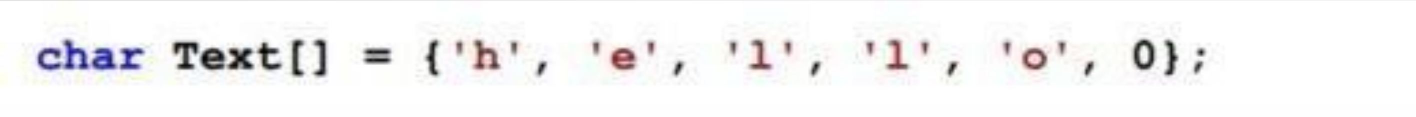
```
#include "stdio.h"
#include "string.h"

void main()
{
      char name[] = "Ahmed Said";
      printf("Name: %s, Length: %d\r\n", name, strlen(name));
}
```

1 Introduction to Array

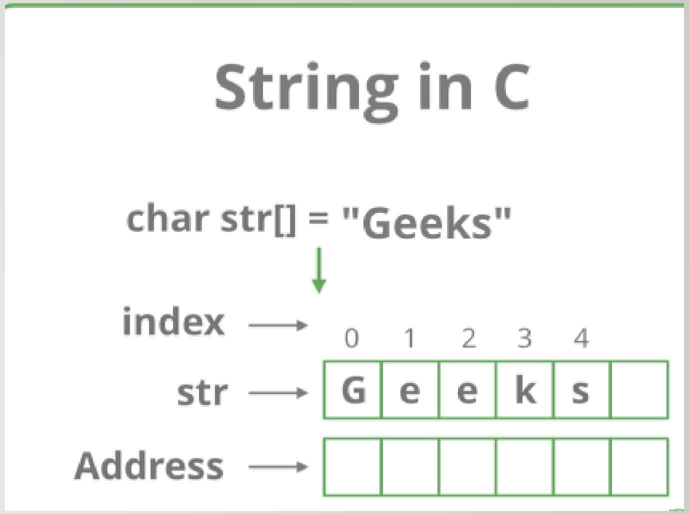
1.5 Storage for Strings in C

I. Array of character:

- Example:



```
char Text[] = {'h', 'e', 'l', 'l', 'o', 0};
```



- Above code shows how to store "hello" string letters in array.
- The last letter is sited with zero value (null termination),
- Text="world"; //error cannot assign anything to array.
- important to tell the computer that the string is terminated before this item.
- the sixth place is used to hold the null termination.
- In printf, "%s" is used to inform the program that it will print a string value.
- read by user Scanf("%s",text) or gets(text)
- display in screen Printf("%s",text) or puts(text)
- Important: printf uses the null termination to
  - end the printing operation. If the null termination is not used, the program continues printing the following memory contents until it reach a zero.

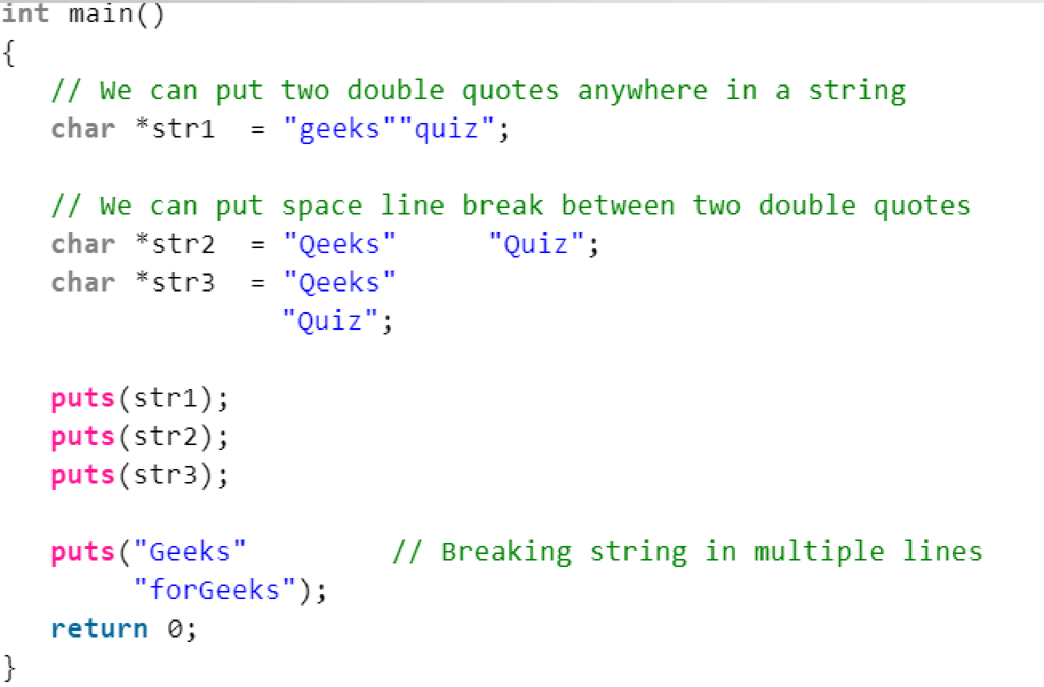
1 Introduction to Array

1.5 Storage for Strings in C

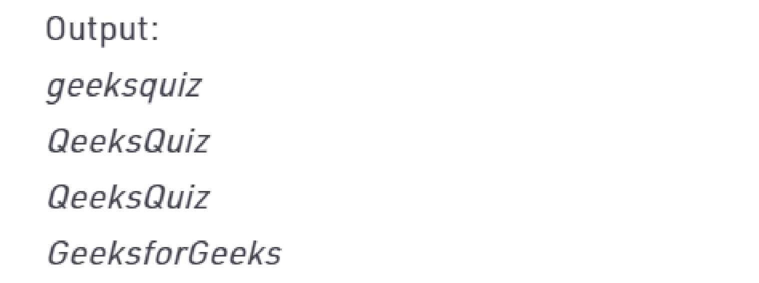
2.Pointer of character.
Pointers are very helpful in handling character arrays with rows of varying lengths.

- Example:

Char *m=“hello”;
Hello saved in .rodata so cannot change of hello if make *m=‘a’ this make run time error if you want to change hello make P =“geek” and m point to first character in h



```
int main()
{
   // We can put two double quotes anywhere in a string
   char *str1  = "geeks""quiz";

   // We can put space line break between two double quotes
   char *str2  = "Qeeks"      "Quiz";
   char *str3  = "Qeeks"
                 "Quiz";

   puts(str1);
   puts(str2);
   puts(str3);

   puts("Geeks"         // Breaking string in multiple lines
        "forGeeks");
   return 0;
}
```

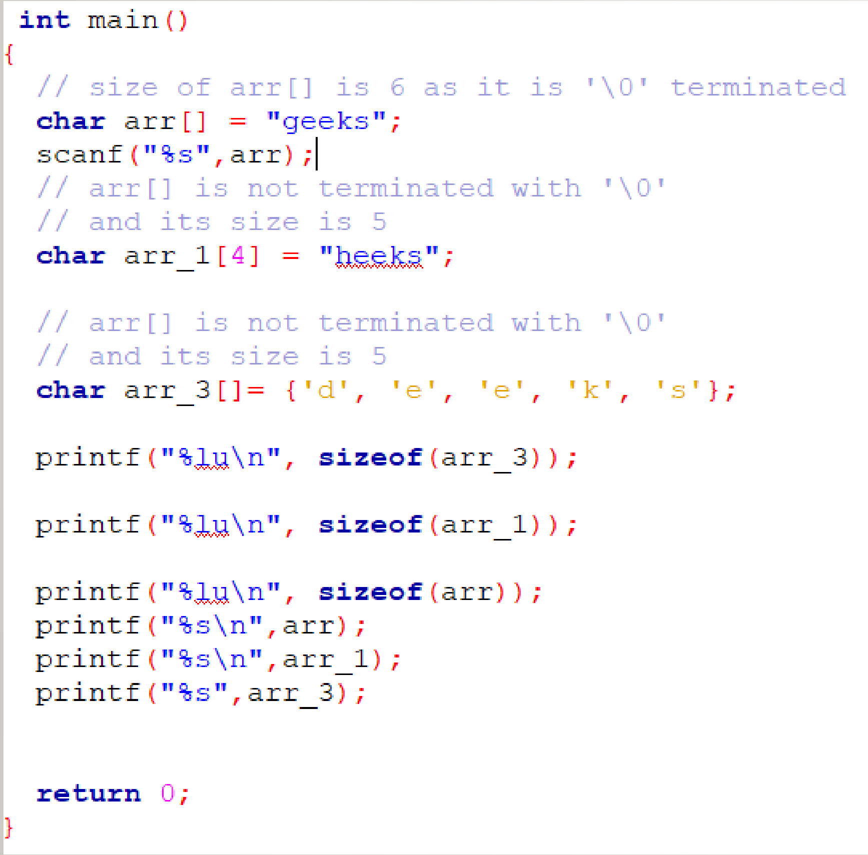


```
Output:
geeksquiz
QeeksQuiz
QeeksQuiz
GeeksforGeeks
```

1 Introduction to pointer

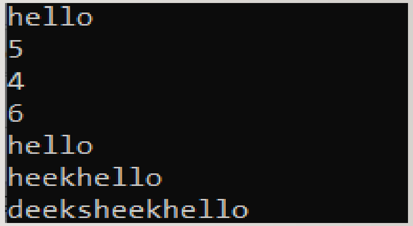
1.5 Storage for Strings in C (array of character)

CODE



```
int main()
{
  // size of arr[] is 6 as it is '\0' terminated
  char arr[] = "geeks";
  scanf("%s",arr);
  // arr[] is not terminated with '\0'
  // and its size is 5
  char arr_1[4] = "heeks";

  // arr[] is not terminated with '\0'
  // and its size is 5
  char arr_3[]= {'d', 'e', 'e', 'k', 's'};

  printf("%lu\n", sizeof(arr_3));

  printf("%lu\n", sizeof(arr_1));

  printf("%lu\n", sizeof(arr));
  printf("%s\n",arr);
  printf("%s\n",arr_1);
  printf("%s",arr_3);


  return 0;
}
```

OUTPUT



```
hello
5
4
6
hello
heekhello
deeksheekhello
```

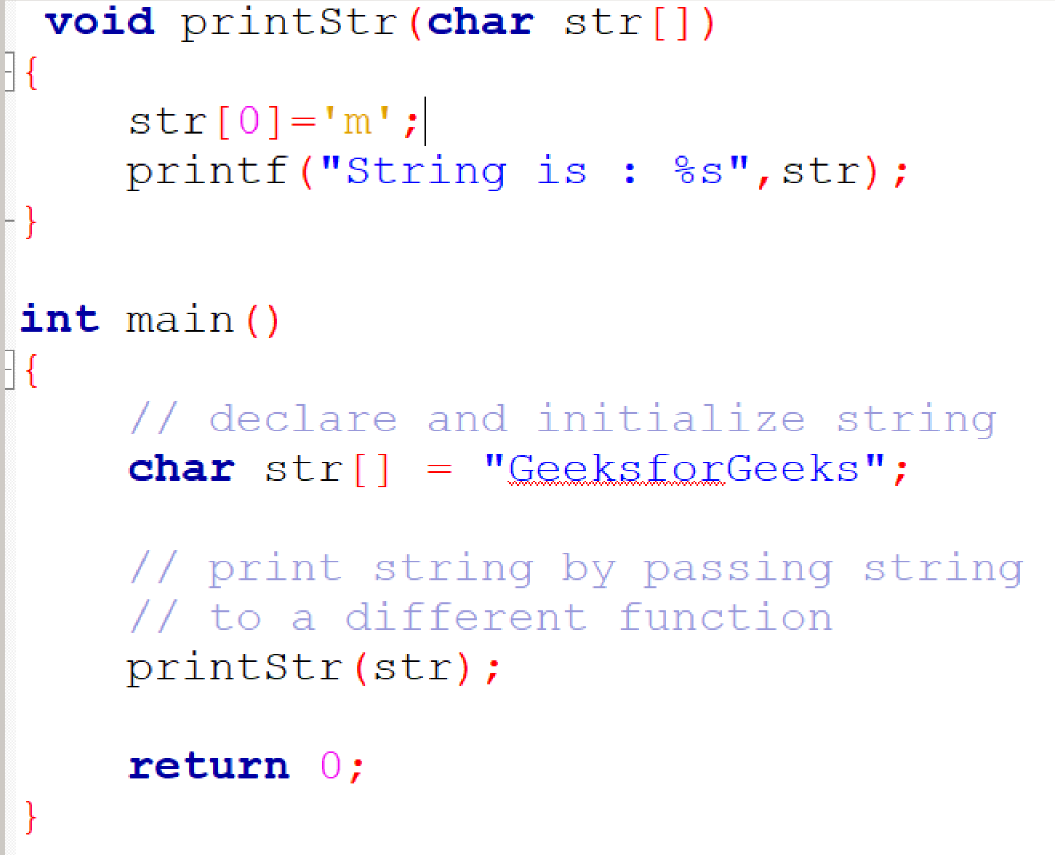
DESCRIPTION

- In this example discuss string .

1 Introduction to pointer

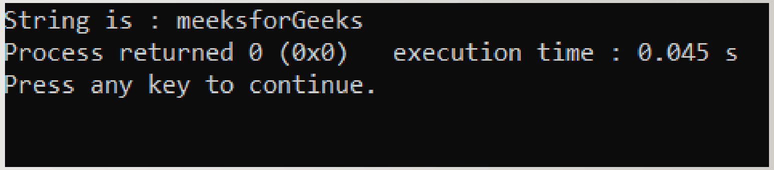
1.5 Storage for Strings in C (array of character) pass string to function.

CODE



```c
 void printStr(char str[])
{
    str[0]='m';
    printf("String is : %s",str);
}

int main()
{
    // declare and initialize string
    char str[] = "GeeksforGeeks";

    // print string by passing string
    // to a different function
    printStr(str);

    return 0;
}
```

OUTPUT



```
String is : meeksforGeeks
Process returned 0 (0x0)   execution time : 0.045 s
Press any key to continue.
```

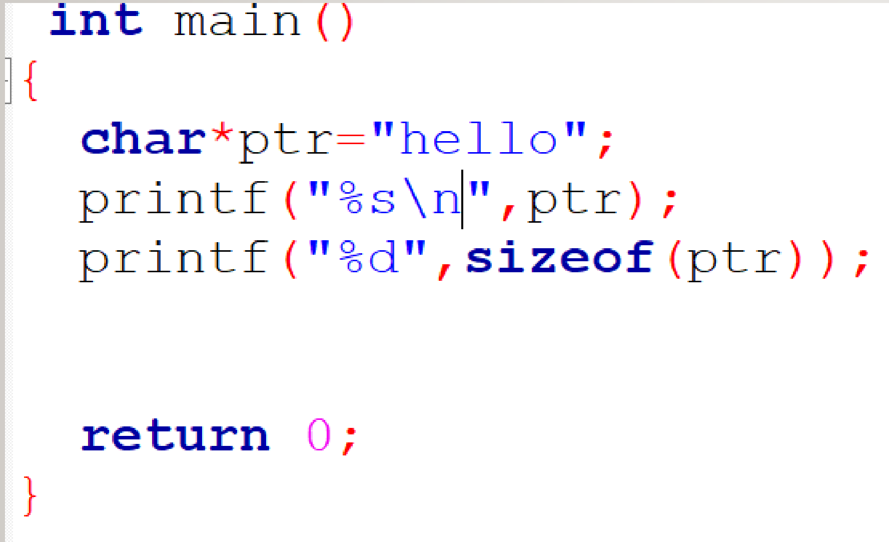
DESCRIPTION

- In this example discuss string and you can change string in another function.

1 Introduction to pointer

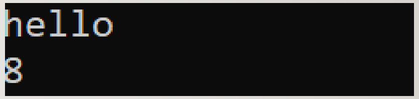
1.5 Storage for Strings in C (character of pointer)

CODE



```
 int main()
{
   char*ptr="hello";
   printf("%s\n",ptr);
   printf("%d",sizeof(ptr));


   return 0;
}
```

OUTPUT



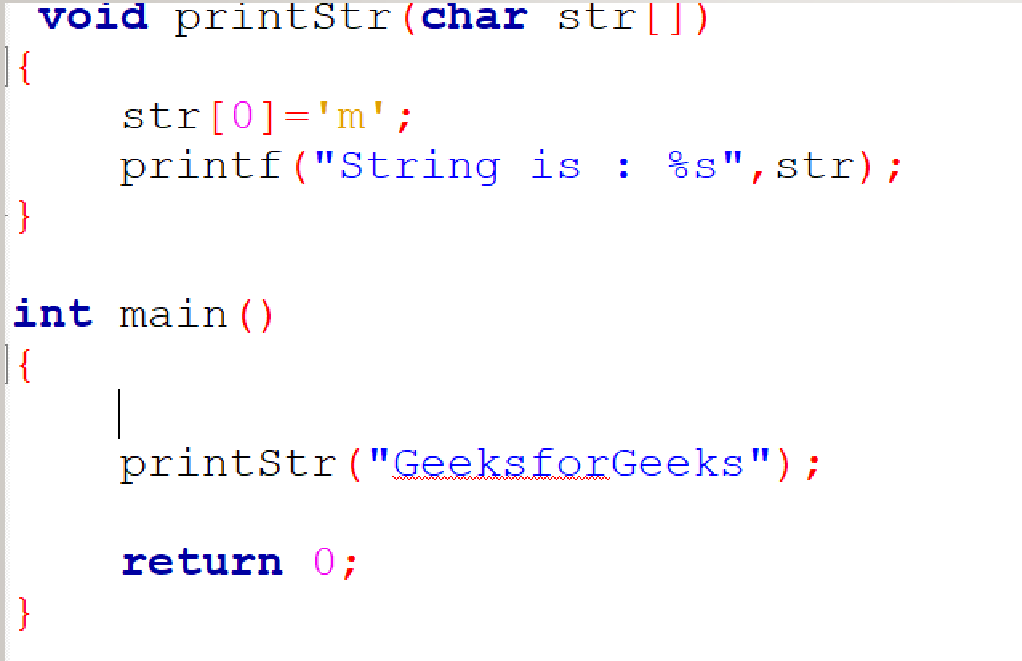
```
hello
8
```

DESCRIPTION

- In this example discuss character of pointer .

1.5 Storage for Strings in C (pointer of character) pass string to function.

CODE



```
 void printStr(char str[])
{
    str[0]='m';
    printf("String is : %s",str);
}

int main()
{

    printStr("GeeksforGeeks");

    return 0;
}
```

OUTPUT

DESCRIPTION

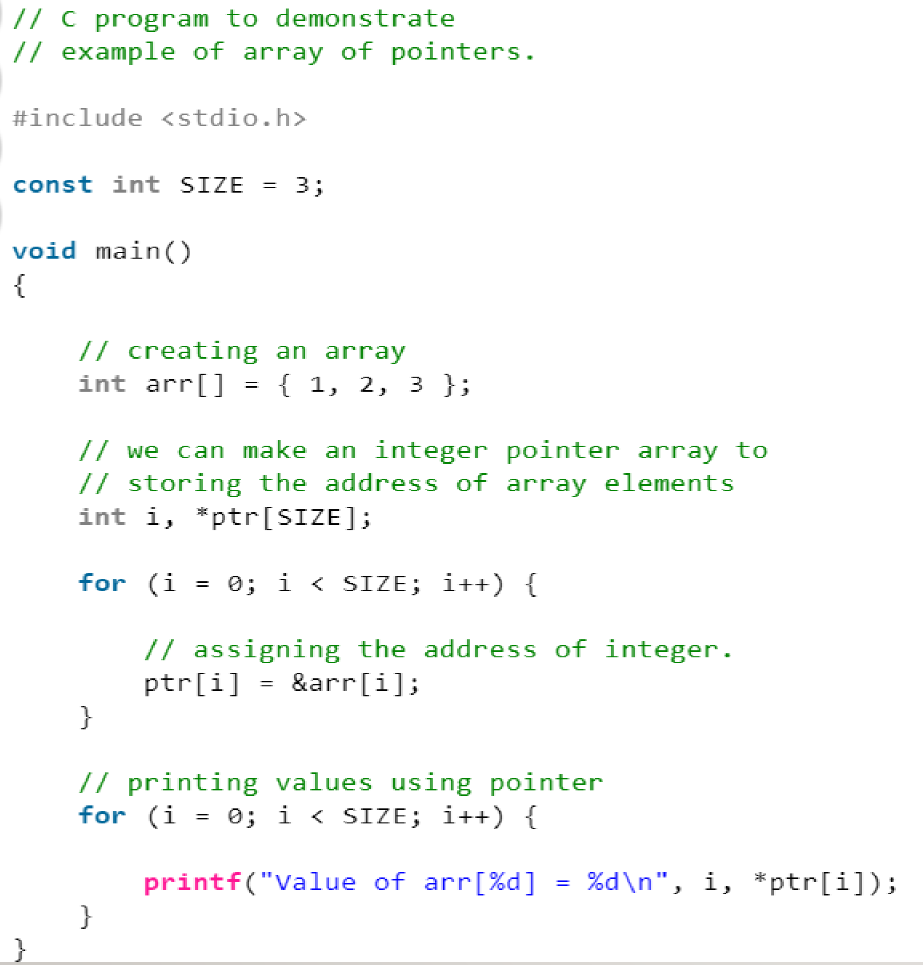
- In this example discuss string by pointer of character so cannot change that passed to function and make run time error.

WHAT IS POINTER

1 Introduction to pointer

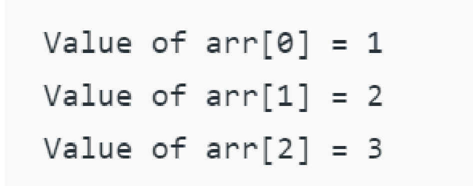
1.5 Storage for Strings in C (pointer of character) pass string to function.

CODE



```
// C program to demonstrate
// example of array of pointers.

#include <stdio.h>

const int SIZE = 3;

void main()
{

    // creating an array
    int arr[] = { 1, 2, 3 };

    // we can make an integer pointer array to
    // storing the address of array elements
    int i, *ptr[SIZE];

    for (i = 0; i < SIZE; i++) {

        // assigning the address of integer.
        ptr[i] = &arr[i];
    }

    // printing values using pointer
    for (i = 0; i < SIZE; i++) {

        printf("Value of arr[%d] = %d\n", i, *ptr[i]);
    }
}
```

OUTPUT



```
Value of arr[0] = 1
Value of arr[1] = 2
Value of arr[2] = 3
```

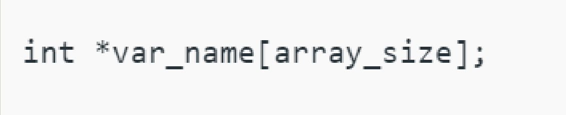
DESCRIPTION

- In this example discuss string by pointer of character so cannot change that passed to function and make run time error.

1.6 Array of pointer and array of pointer to function

- Declaration array of pointer :
- is an array of the pointer variables. It is also known as pointer arrays.
- We can make separate pointer variables which can point to the different values or we can make one integer array of pointers that can point to all the values.

Syntax:



```
int *var_name[array_size];
```

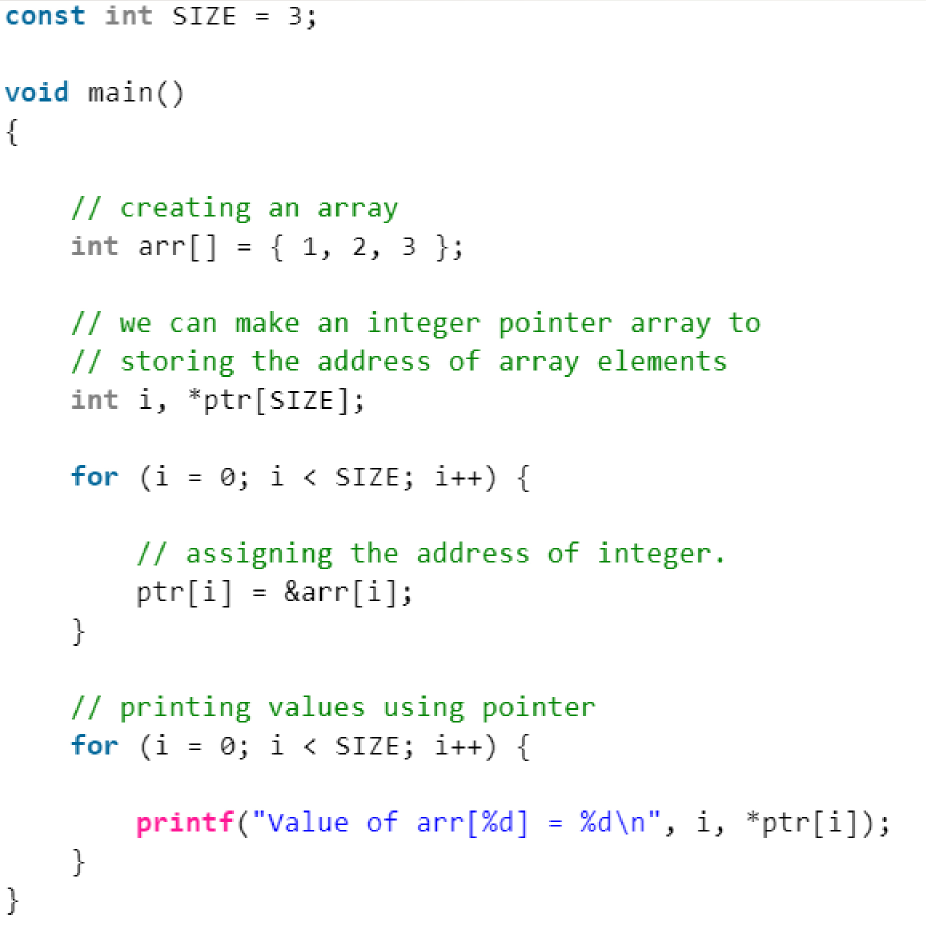
- Declaration of an array of pointers:

Int*ptr[10];

1 Introduction to pointer

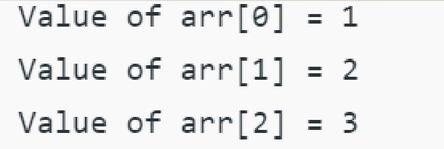
1.5 Storage for Strings in C (pointer of character) pass string to function.

CODE



```
const int SIZE = 3;

void main()
{

    // creating an array
    int arr[] = { 1, 2, 3 };

    // we can make an integer pointer array to
    // storing the address of array elements
    int i, *ptr[SIZE];

    for (i = 0; i < SIZE; i++) {

        // assigning the address of integer.
        ptr[i] = &arr[i];
    }

    // printing values using pointer
    for (i = 0; i < SIZE; i++) {

        printf("Value of arr[%d] = %d\n", i, *ptr[i]);
    }
}
```

OUTPUT



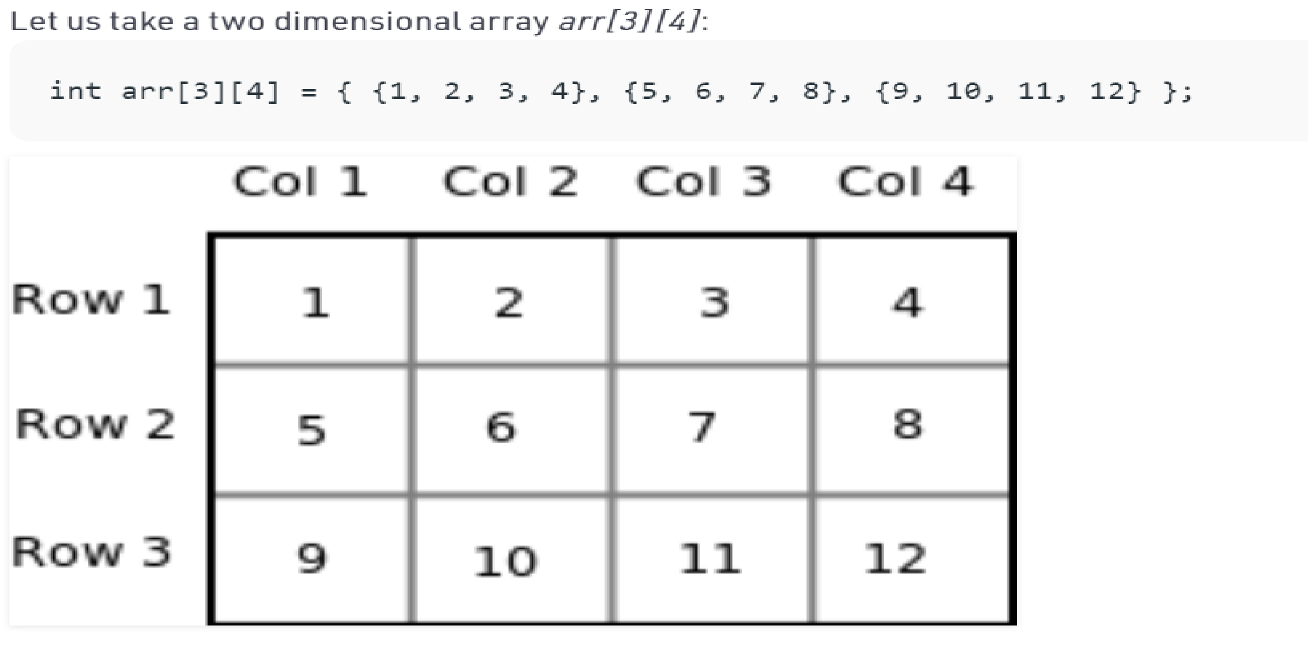
```
Value of arr[0] = 1
Value of arr[1] = 2
Value of arr[2] = 3
```

DESCRIPTION

- In this example discuss array of pointer each element in pointer point to element in another array.

1 Introduction to Array

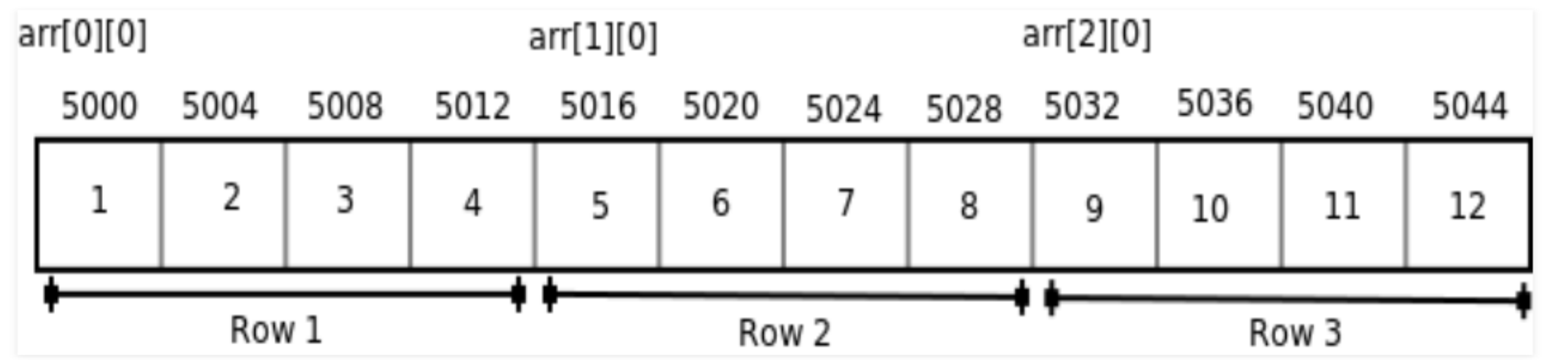
1.7 Initialization of a multidimensional arrays in C(2D)



Let us take a two dimensional array *arr[3][4]*:

```
int arr[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
```

	Col 1	Col 2	Col 3	Col 4
Row 1	1	2	3	4
Row 2	5	6	7	8
Row 3	9	10	11	12



5000	5004	5008	5012	5016	5020	5024	5028	5032	5036	5040	5044
1	2	3	4	5	6	7	8	9	10	11	12

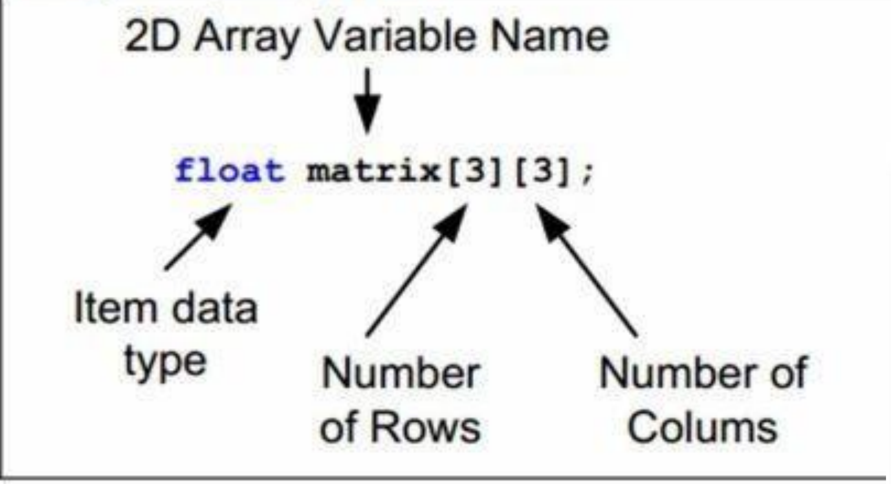
1 Introduction to Array

1.7 Initialization of a multidimensional arrays in C(2D)

2D array is suitable to hold tabular information for example, it can be used to store 4 students degrees in 6 subjects or to represent (m x n) matrix.

To define an array variable containing 100 (**float**) values:

```
float matrix[3][3];
```

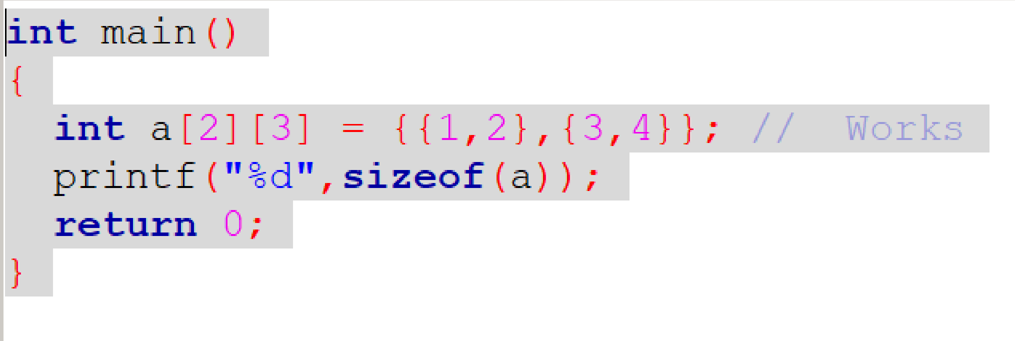


- The array float matrix**[3][3]** can store total (3*3) = 6 elements.
- Size of matrix is 36. each size of element in matrix is 4byte.

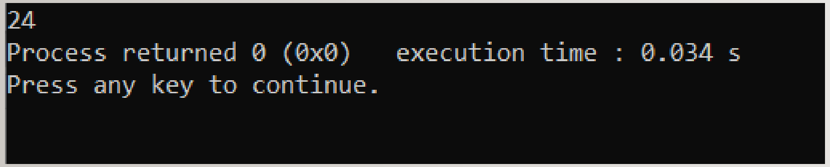
1 Introduction to Array

1.7 Initialization of a multidimensional arrays in C(2D)

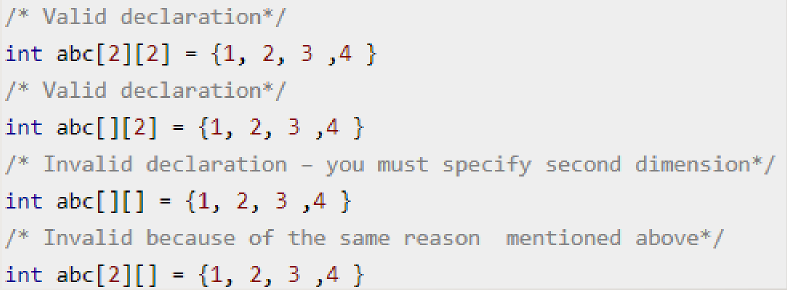
- initialization of a multidimensional arrays can have left most dimension as optional. Except the left most dimension, all other dimensions must be specified.
  For example, following program fails in compilation because two dimensions are not specified



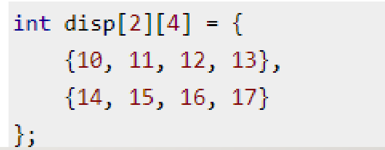
```
int main()
{
   int a[2][3] = {{1,2},{3,4}}; //  Works
   printf("%d",sizeof(a));
   return 0;
}
```



```
24
Process returned 0 (0x0)   execution time : 0.034 s
Press any key to continue.
```



```
/* Valid declaration*/
int abc[2][2] = {1, 2, 3 ,4 }
/* Valid declaration*/
int abc[][2] = {1, 2, 3 ,4 }
/* Invalid declaration – you must specify second dimension*/
int abc[][] = {1, 2, 3 ,4 }
/* Invalid because of the same reason  mentioned above*/
int abc[2][] = {1, 2, 3 ,4 }
```

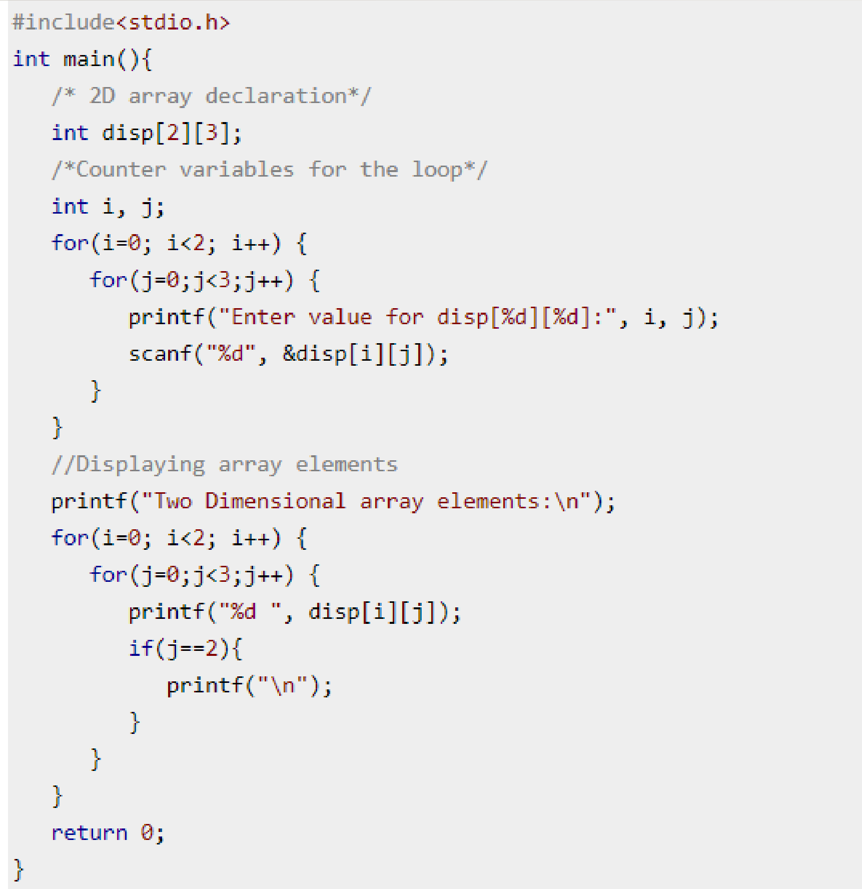


```
int disp[2][4] = {
    {10, 11, 12, 13},
    {14, 15, 16, 17}
};
```

1 Introduction to pointer

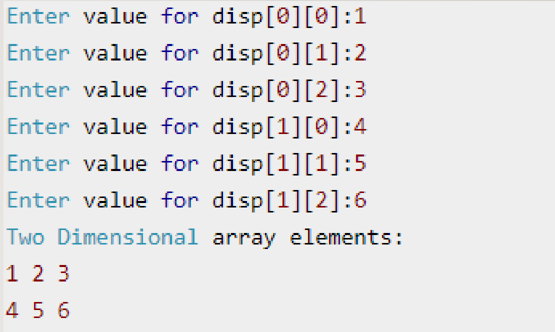
1.7 Initialization of a multidimensional arrays in C(2D) array of integer

CODE



```
#include<stdio.h>
int main(){
   /* 2D array declaration*/
   int disp[2][3];
   /*Counter variables for the loop*/
   int i, j;
   for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
         printf("Enter value for disp[%d][%d]:", i, j);
         scanf("%d", &disp[i][j]);
      }
   }
   //Displaying array elements
   printf("Two Dimensional array elements:\n");
   for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
         printf("%d ", disp[i][j]);
         if(j==2){
            printf("\n");
         }
      }
   }
   return 0;
}
```

OUTPUT



```
Enter value for disp[0][0]:1
Enter value for disp[0][1]:2
Enter value for disp[0][2]:3
Enter value for disp[1][0]:4
Enter value for disp[1][1]:5
Enter value for disp[1][2]:6
Two Dimensional array elements:
1 2 3
4 5 6
```

DESCRIPTION

- In this example discuss 2D array

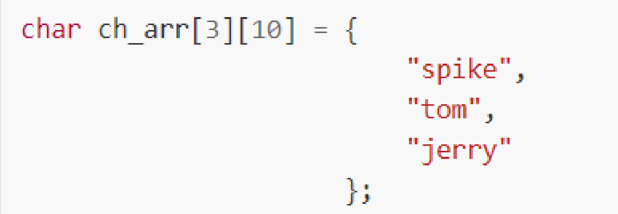
1.7 Initialization of a multidimensional arrays in C(2D) strings in c

- declare strings in c by two ways first way :

1. Array of pointer.
2. Array of array of character.

1.Array of array of character is (2D) array

- Example:

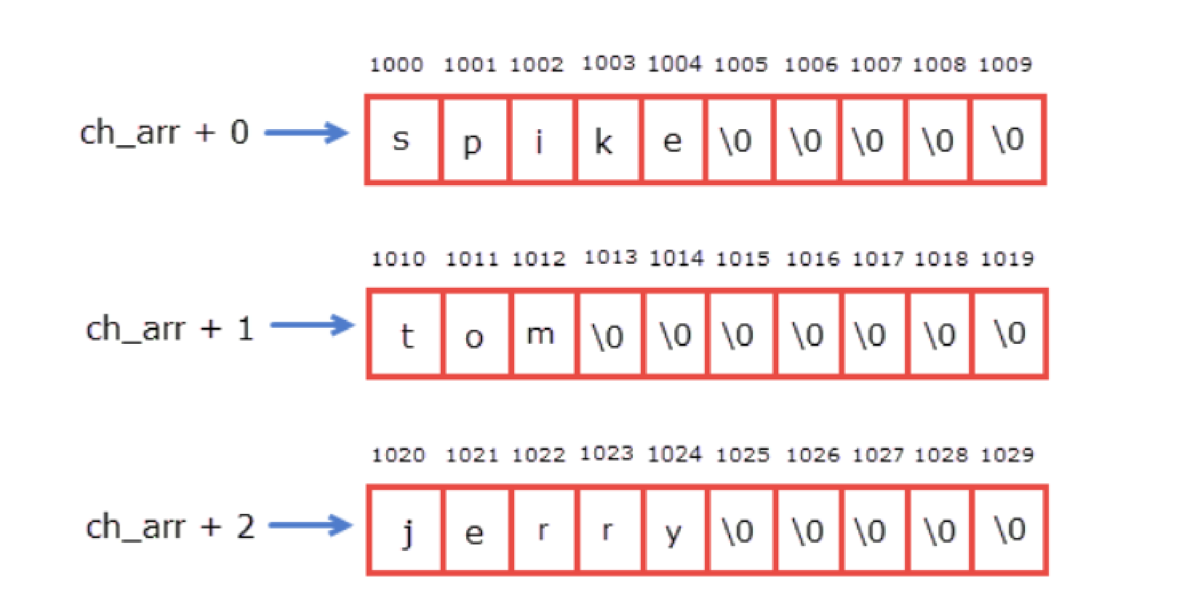


```
char ch_arr[3][10] = {
                         "spike",
                         "tom",
                         "jerry"
                     };
```

- Size of cha_arr is (3*10)*sizeof(char).

1 Introduction to Array

1.7 Initialization of a multidimensional arrays in C(2D) strings in c



ch_arr + 0 →

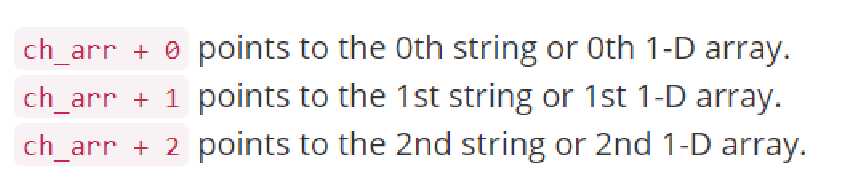
1000	1001	1002	1003	1004	1005	1006	1007	1008	1009
s	p	i	k	e	\0	\0	\0	\0	\0

ch_arr + 1 →

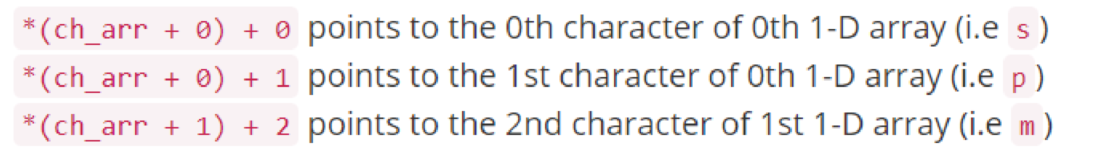
1010	1011	1012	1013	1014	1015	1016	1017	1018	1019
t	o	m	\0	\0	\0	\0	\0	\0	\0

ch_arr + 2 →

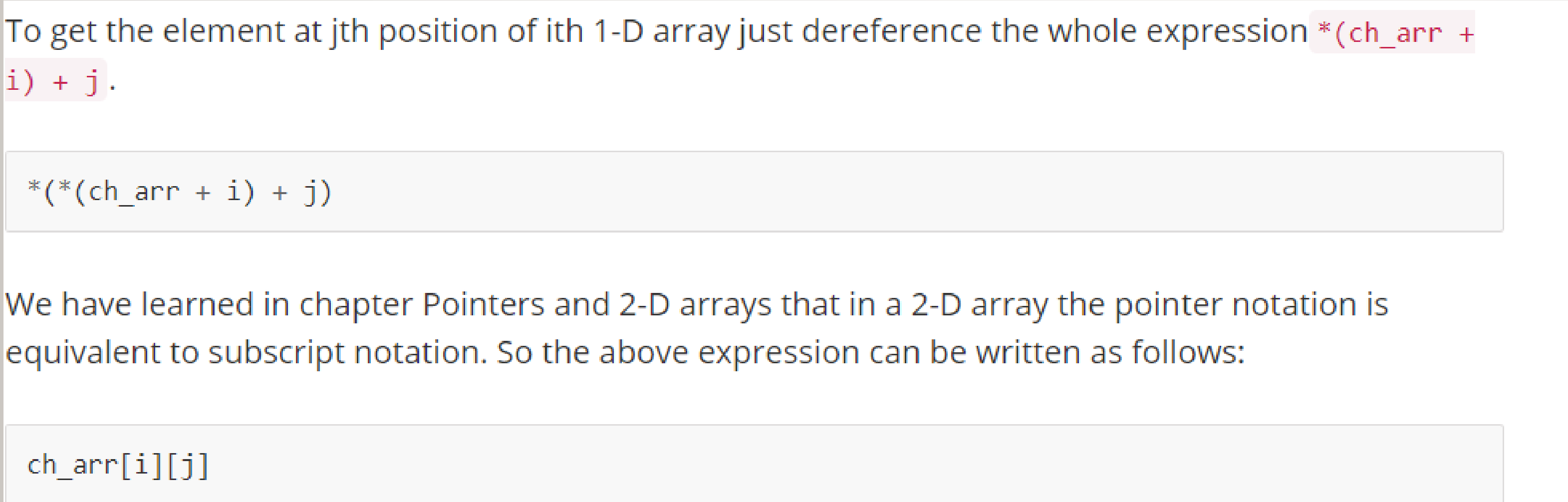
1020	1021	1022	1023	1024	1025	1026	1027	1028	1029
j	e	r	r	y	\0	\0	\0	\0	\0



`ch_arr + 0` points to the 0th string or 0th 1-D array.
`ch_arr + 1` points to the 1st string or 1st 1-D array.
`ch_arr + 2` points to the 2nd string or 2nd 1-D array.



`*(ch_arr + 0) + 0` points to the 0th character of 0th 1-D array (i.e `s`)
`*(ch_arr + 0) + 1` points to the 1st character of 0th 1-D array (i.e `p`)
`*(ch_arr + 1) + 2` points to the 2nd character of 1st 1-D array (i.e `m`)



To get the element at jth position of ith 1-D array just dereference the whole expression `*(ch_arr + i) + j`.

```
*(*(ch_arr + i) + j)
```

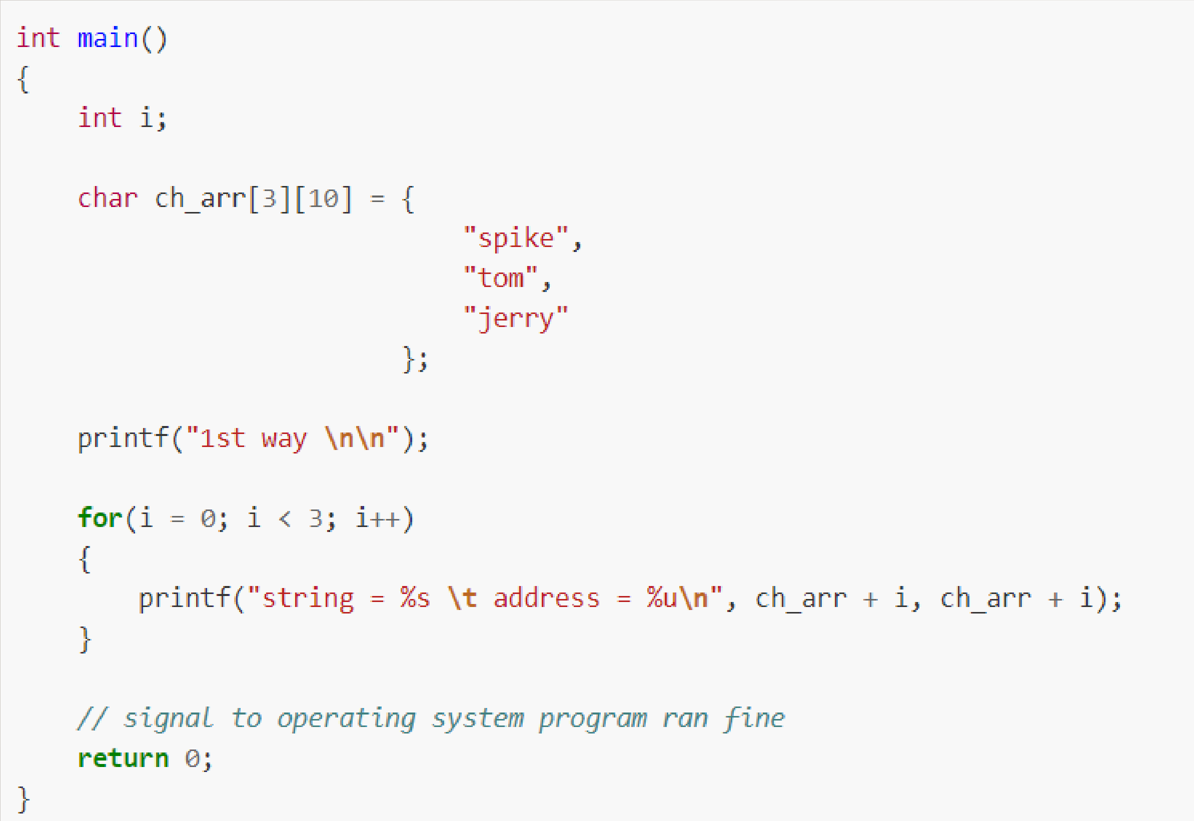
We have learned in chapter Pointers and 2-D arrays that in a 2-D array the pointer notation is equivalent to subscript notation. So the above expression can be written as follows:

```
ch_arr[i][j]
```

1 Introduction to pointer

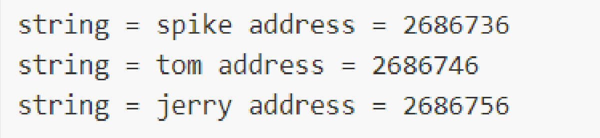
1.7 Initialization of a multidimensional arrays in C(2D) array of string

CODE



```c
int main()
{
    int i;

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

    printf("1st way \n\n");

    for(i = 0; i < 3; i++)
    {
        printf("string = %s \t address = %u\n", ch_arr + i, ch_arr + i);
    }

    // signal to operating system program ran fine
    return 0;
}
```

OUTPUT



```
string = spike address = 2686736
string = tom address = 2686746
string = jerry address = 2686756
```

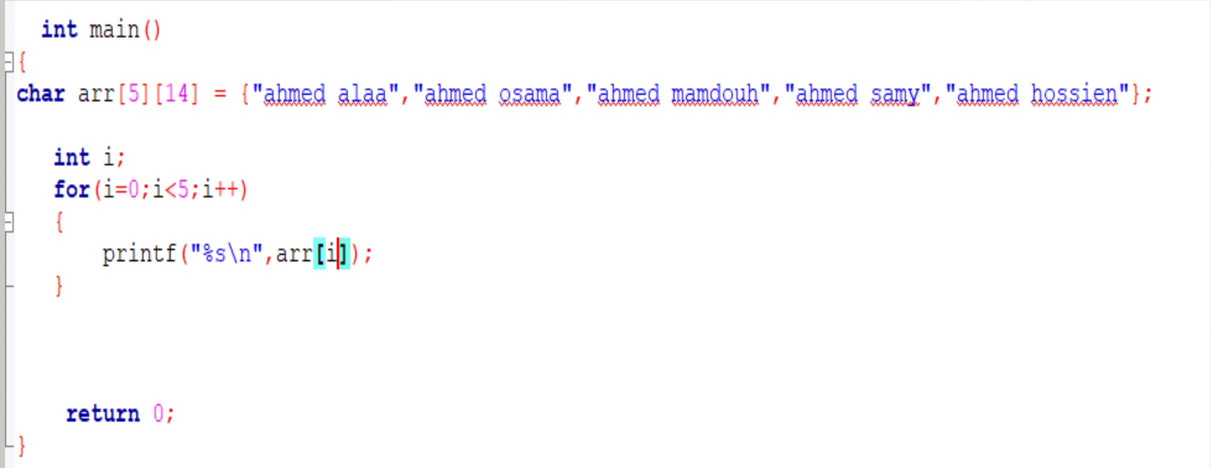
DESCRIPTION

- In this example discuss the ch_arr+i point address of first element in string also string.

1 Introduction to pointer

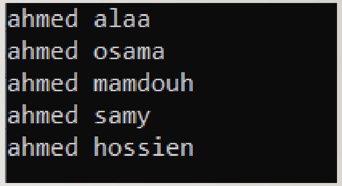
1.7 Initialization of a multidimensional arrays in C(2D) array of string

CODE



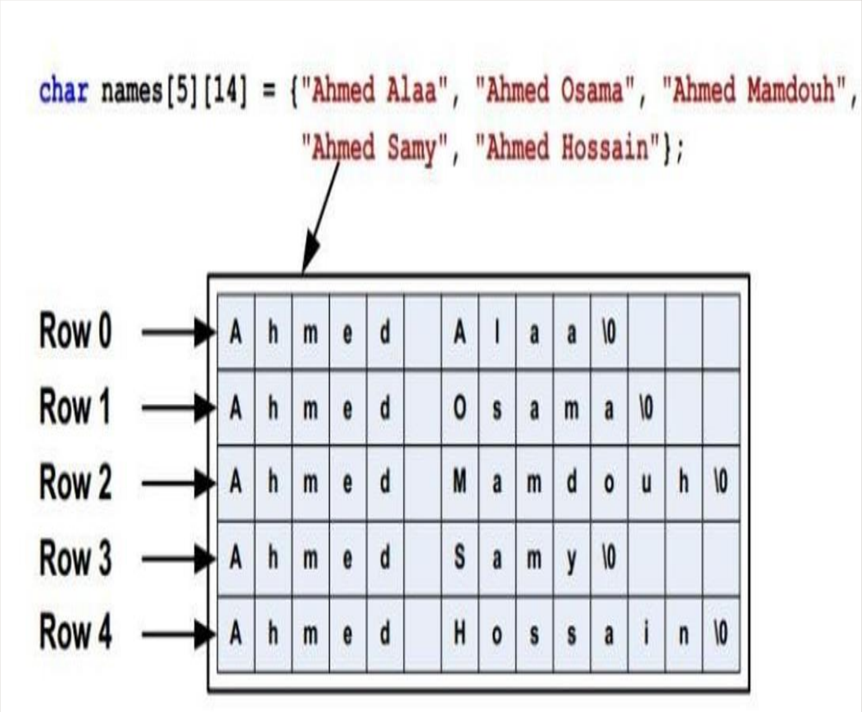
```
  int main()
{
char arr[5][14] = {"ahmed alaa","ahmed osama","ahmed mamdouh","ahmed samy","ahmed hossien"};

   int i;
   for(i=0;i<5;i++)
   {
       printf("%s\n",arr[i]);
   }



    return 0;
}
```

OUTPUT



```
ahmed alaa
ahmed osama
ahmed mamdouh
ahmed samy
ahmed hossien
```

DESCRIPTION



```
char names[5][14] = {"Ahmed Alaa", "Ahmed Osama", "Ahmed Mamdouh",
                     "Ahmed Samy", "Ahmed Hossain"};
```

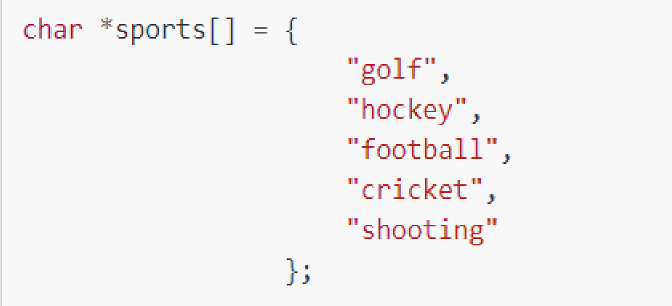
Row 0	A	h	m	e	d		A	l	a	a	\0			
Row 1	A	h	m	e	d		O	s	a	m	a	\0		
Row 2	A	h	m	e	d		M	a	m	d	o	u	h	\0
Row 3	A	h	m	e	d		S	a	m	y	\0			
Row 4	A	h	m	e	d		H	o	s	s	a	i	n	\0

1 Introduction to Array

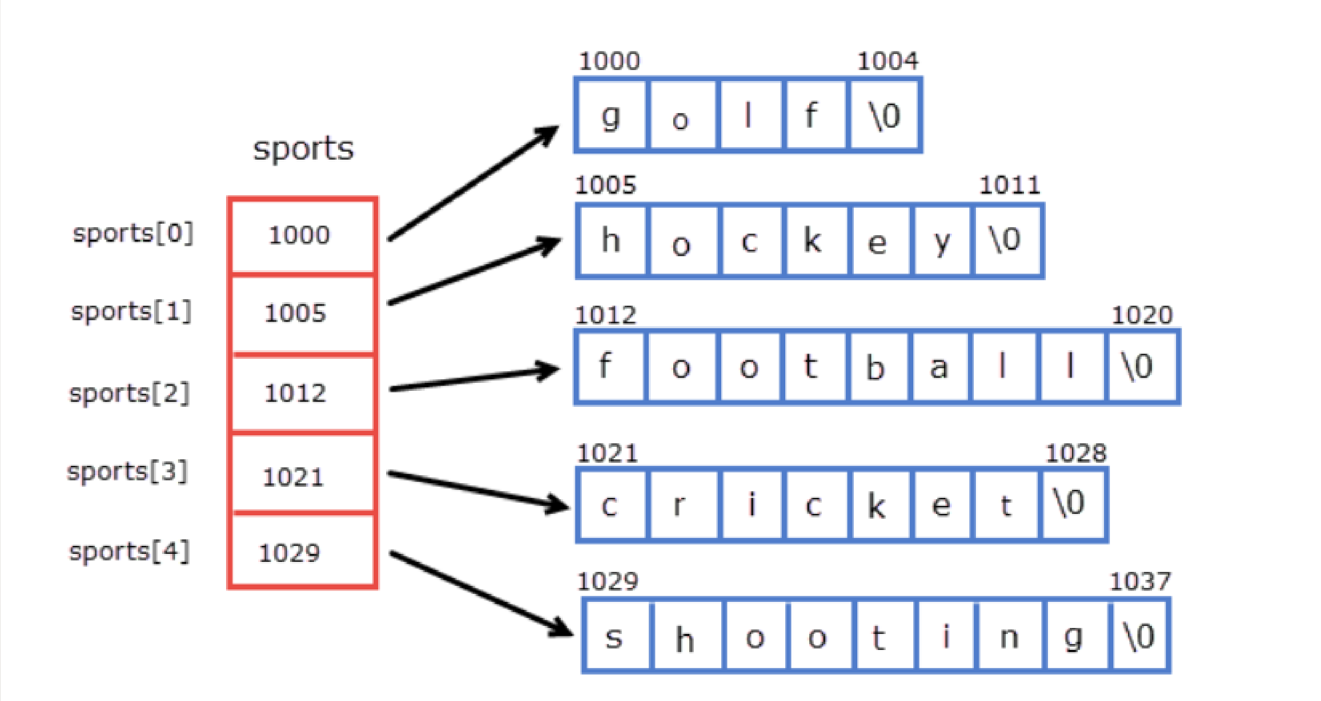
1.7 Array of pointer strings in c

2.An array of pointers to strings:
is an array of character pointers where each pointer points to the first character of the string or the base address of the string. Let's see how we can declare and initialize an array of pointers to strings.
The sport[0] point to base address of string”golf”, sport[1] point to base address of string “hockey”;

- Example:



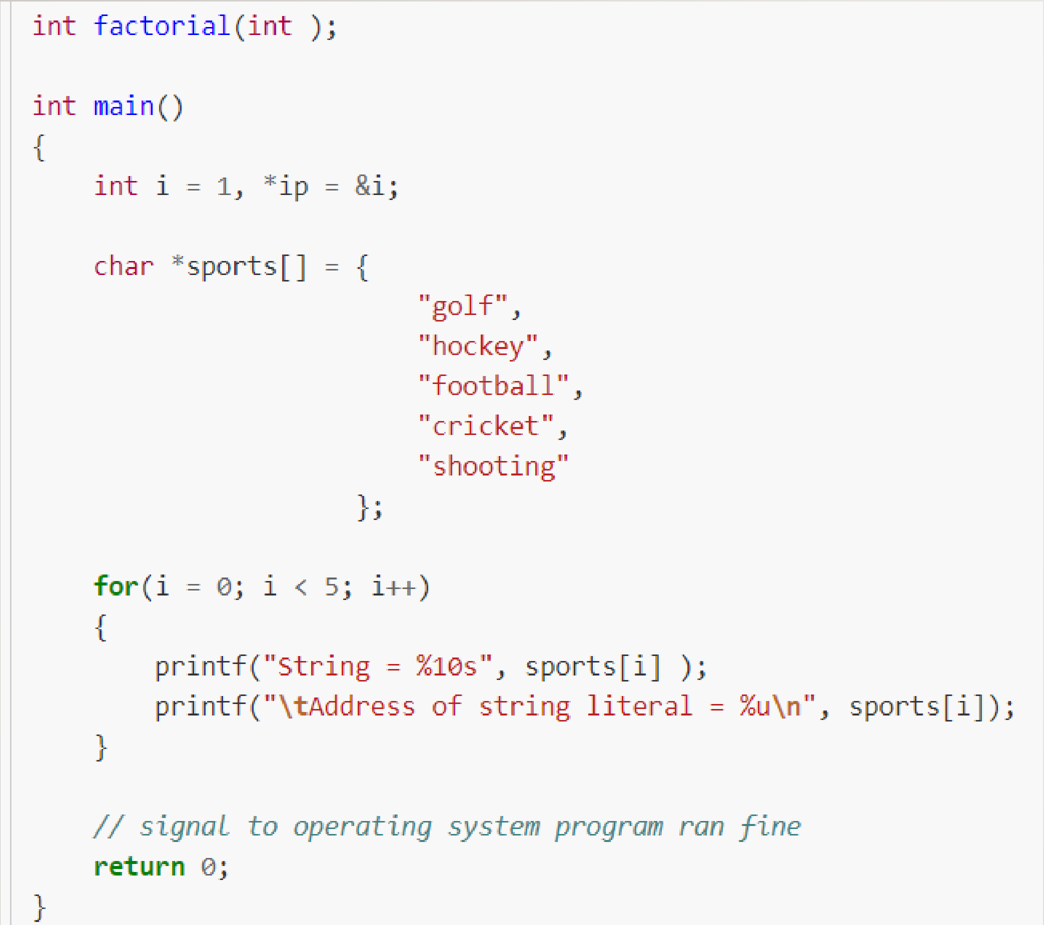
```
char *sports[] = {
                    "golf",
                    "hockey",
                    "football",
                    "cricket",
                    "shooting"
                 };
```



1 Introduction to pointer

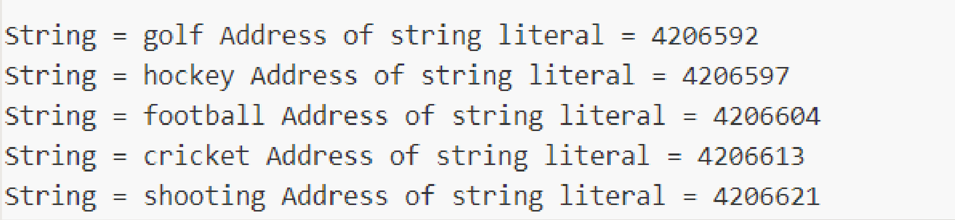
1.7 Array of pointer strings in c

CODE



```
int factorial(int );

int main()
{
    int i = 1, *ip = &i;

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

    for(i = 0; i < 5; i++)
    {
        printf("String = %10s", sports[i] );
        printf("\tAddress of string literal = %u\n", sports[i]);
    }

    // signal to operating system program ran fine
    return 0;
}
```

OUTPUT



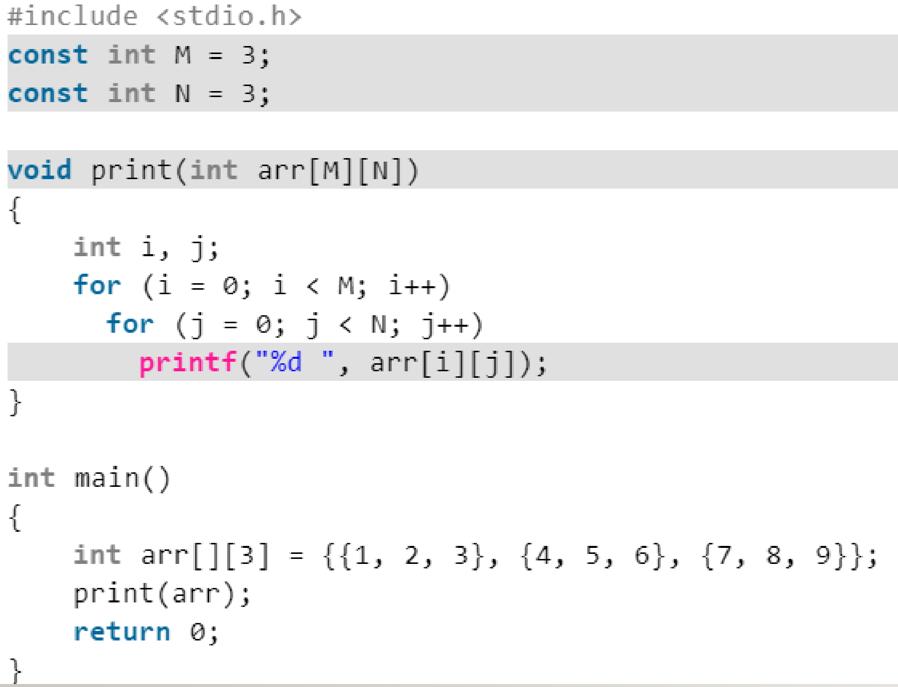
```
String = golf Address of string literal = 4206592
String = hockey Address of string literal = 4206597
String = football Address of string literal = 4206604
String = cricket Address of string literal = 4206613
String = shooting Address of string literal = 4206621
```

DESCRIPTION

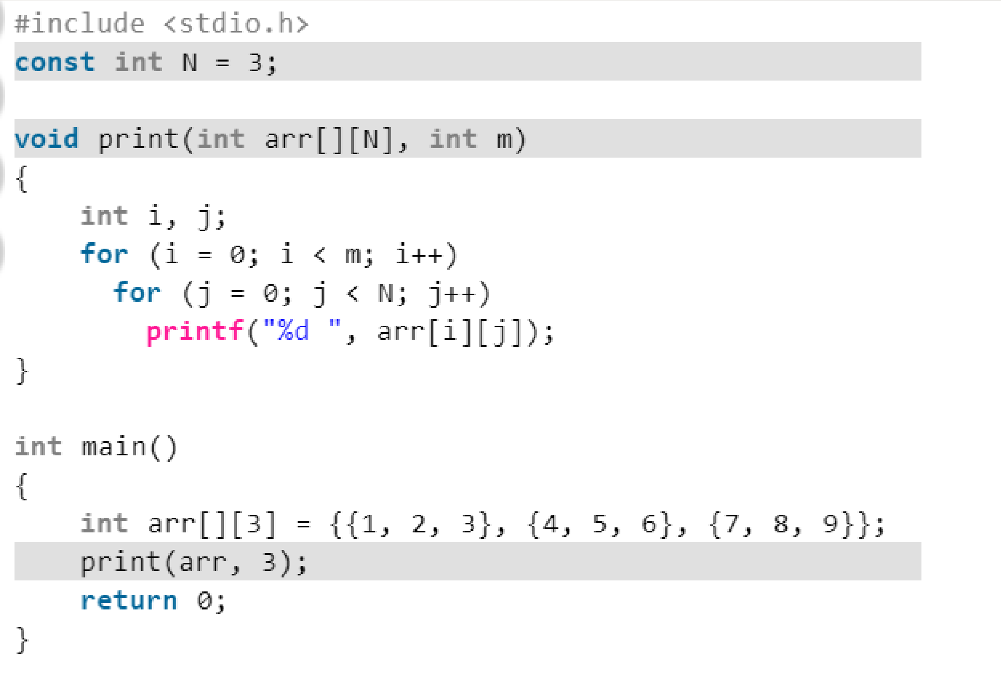
- In this example discuss Array of pointer strings in c

1.8 Initialization of a multidimensional arrays in C(2D) pass to function

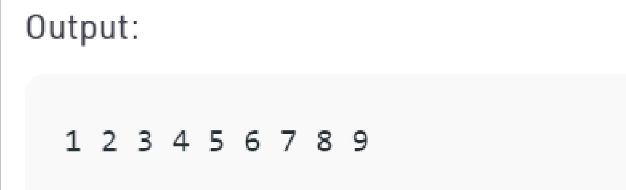
- When both dimensions are available globally (either as a macro or as a global constant).
- When only second dimension is available globally (either as a macro or as a global constant).



```c
#include <stdio.h>
const int M = 3;
const int N = 3;

void print(int arr[M][N])
{
    int i, j;
    for (i = 0; i < M; i++)
      for (j = 0; j < N; j++)
        printf("%d ", arr[i][j]);
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    print(arr);
    return 0;
}
```

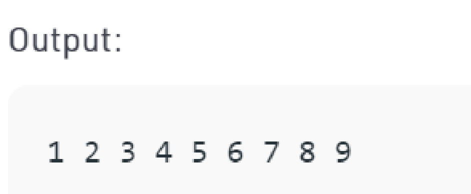


```c
#include <stdio.h>
const int N = 3;

void print(int arr[][N], int m)
{
    int i, j;
    for (i = 0; i < m; i++)
      for (j = 0; j < N; j++)
        printf("%d ", arr[i][j]);
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    print(arr, 3);
    return 0;
}
```



Output:

```
1 2 3 4 5 6 7 8 9
```

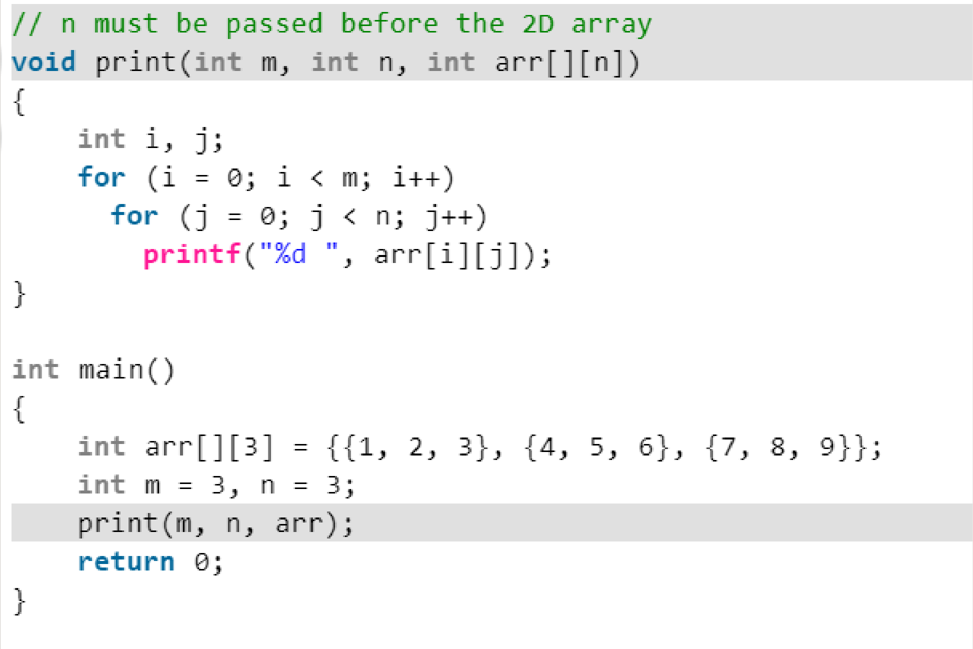


Output:

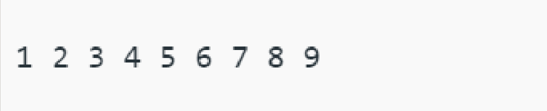
```
1 2 3 4 5 6 7 8 9
```

1.8 Initialization of a multidimensional arrays in C(2D) pass to function

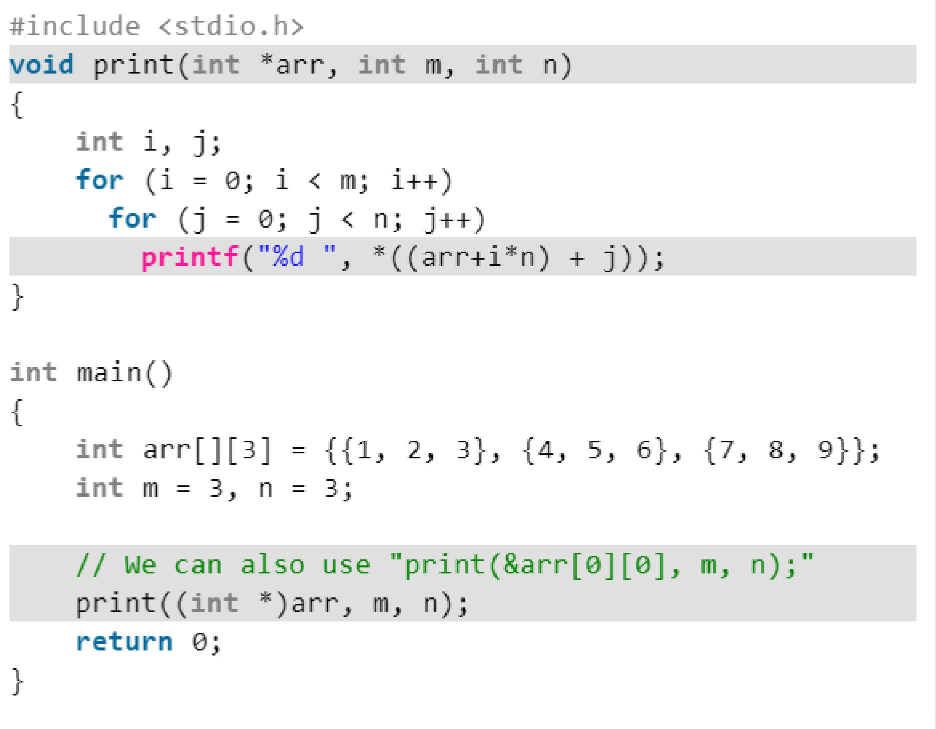
- C language supports variable sized arrays to be passed simply by specifying the variable dimensions.
- In this method, we must typecast the 2D array when passing to function.



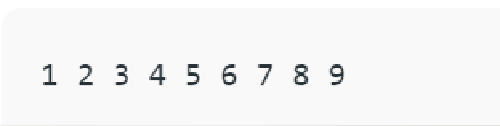
```c
// n must be passed before the 2D array
void print(int m, int n, int arr[][n])
{
    int i, j;
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        printf("%d ", arr[i][j]);
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int m = 3, n = 3;
    print(m, n, arr);
    return 0;
}
```



```
1 2 3 4 5 6 7 8 9
```



```c
#include <stdio.h>
void print(int *arr, int m, int n)
{
    int i, j;
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        printf("%d ", *((arr+i*n) + j));
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int m = 3, n = 3;

    // We can also use "print(&arr[0][0], m, n);"
    print((int *)arr, m, n);
    return 0;
}
```

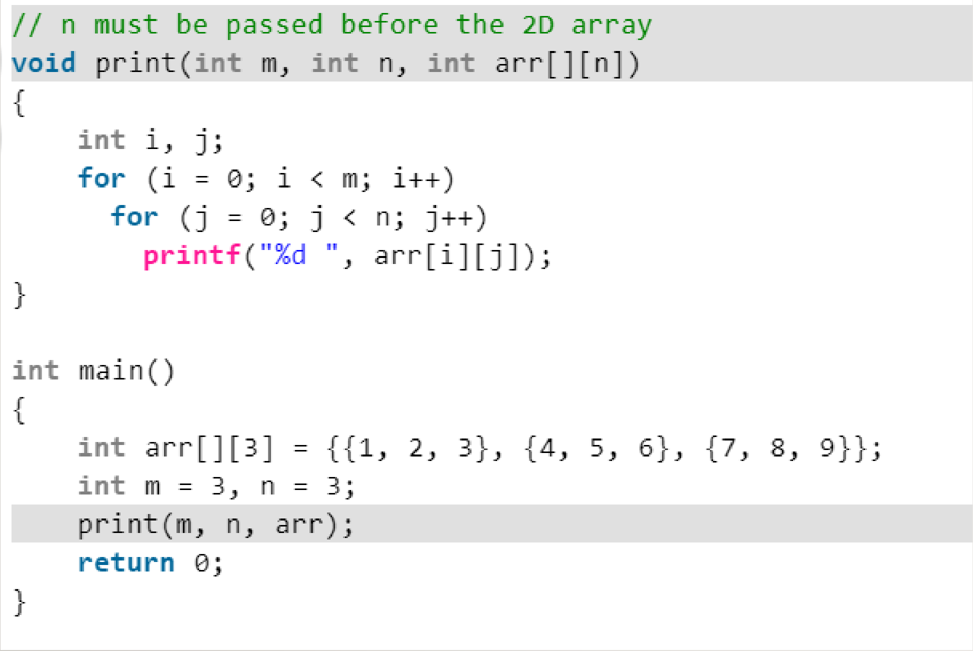


```
1 2 3 4 5 6 7 8 9
```

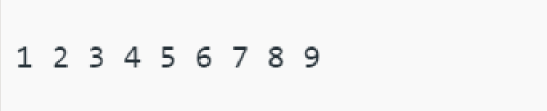
1 Introduction to pointer

1.8 Initialization of a multidimensional arrays in C(2D) pass to function

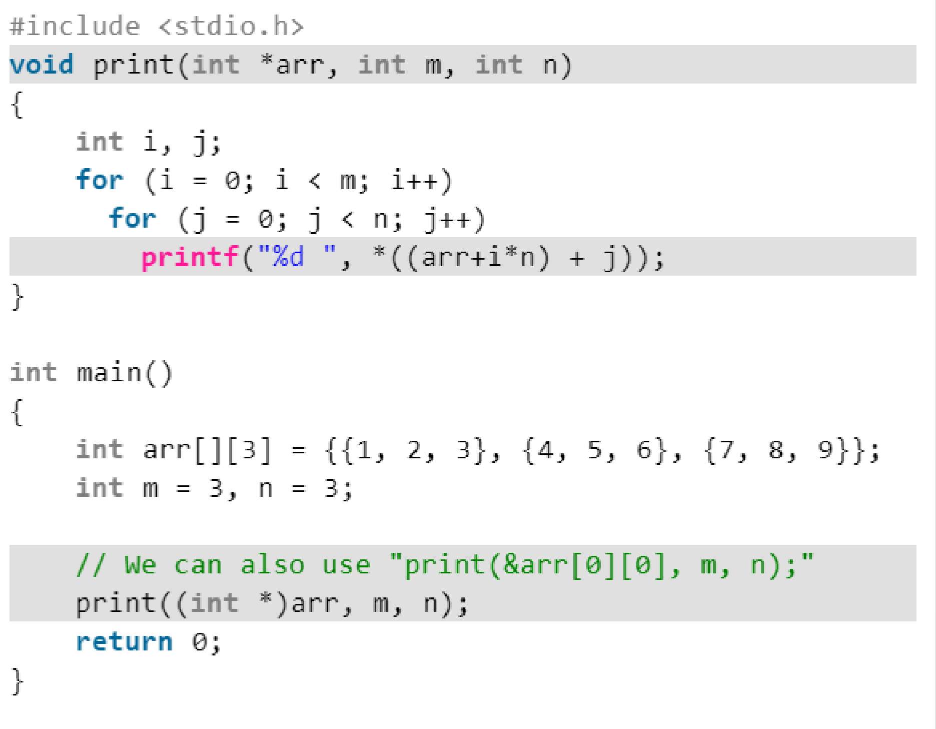
- C language supports variable sized arrays to be passed simply by specifying the variable dimensions.
- In this method, we must typecast the 2D array when passing to function.



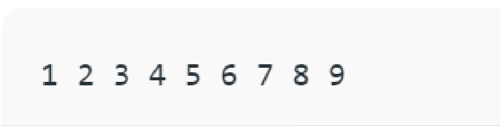
```c
// n must be passed before the 2D array
void print(int m, int n, int arr[][n])
{
    int i, j;
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        printf("%d ", arr[i][j]);
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int m = 3, n = 3;
    print(m, n, arr);
    return 0;
}
```



```
1 2 3 4 5 6 7 8 9
```



```c
#include <stdio.h>
void print(int *arr, int m, int n)
{
    int i, j;
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        printf("%d ", *((arr+i*n) + j));
}

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int m = 3, n = 3;

    // We can also use "print(&arr[0][0], m, n);"
    print((int *)arr, m, n);
    return 0;
}
```



```
1 2 3 4 5 6 7 8 9
```

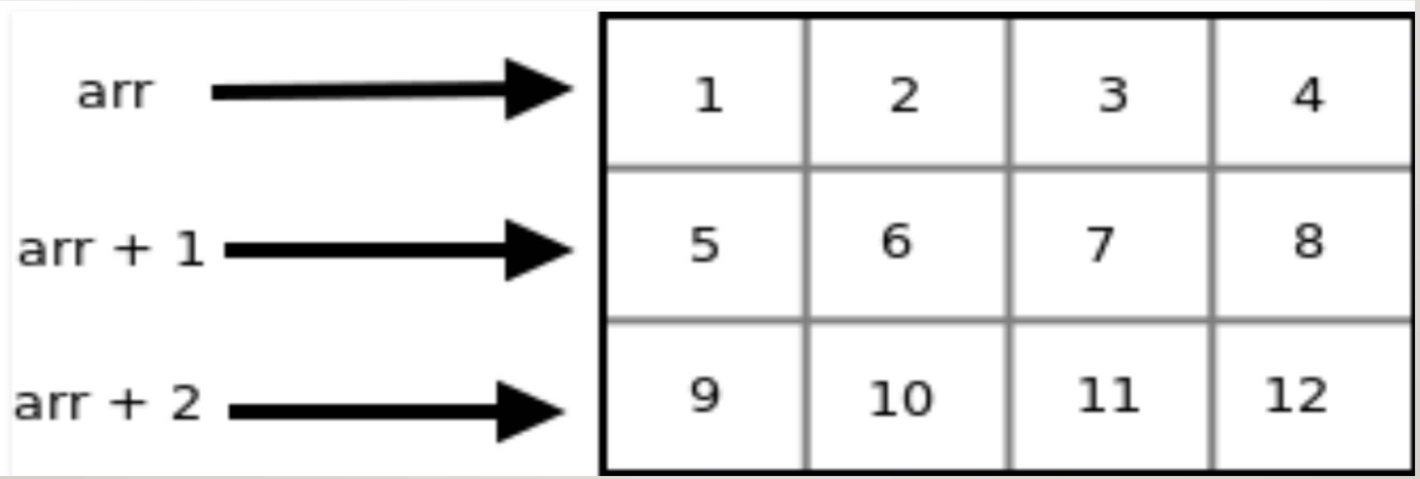
1.9 Pointer and related to (2D)

➢ pointer and related to (2D):

- In a two dimensional array, we can access each element by using two subscripts, where first subscript represents the row number and second subscript represents the column number. The elements of 2-D array can be accessed with the help of pointer notation also. Suppose arr is a 2-D array, we can access any element *arr[i][j]* of the array using the pointer expression ***(*(arr + i) + j)**.

➢ Example:

- Int arr[3][4]={{1,2,3,4},{4,5,6,7},{9,10,11,12}};



1	2	3	4
5	6	7	8
9	10	11	12

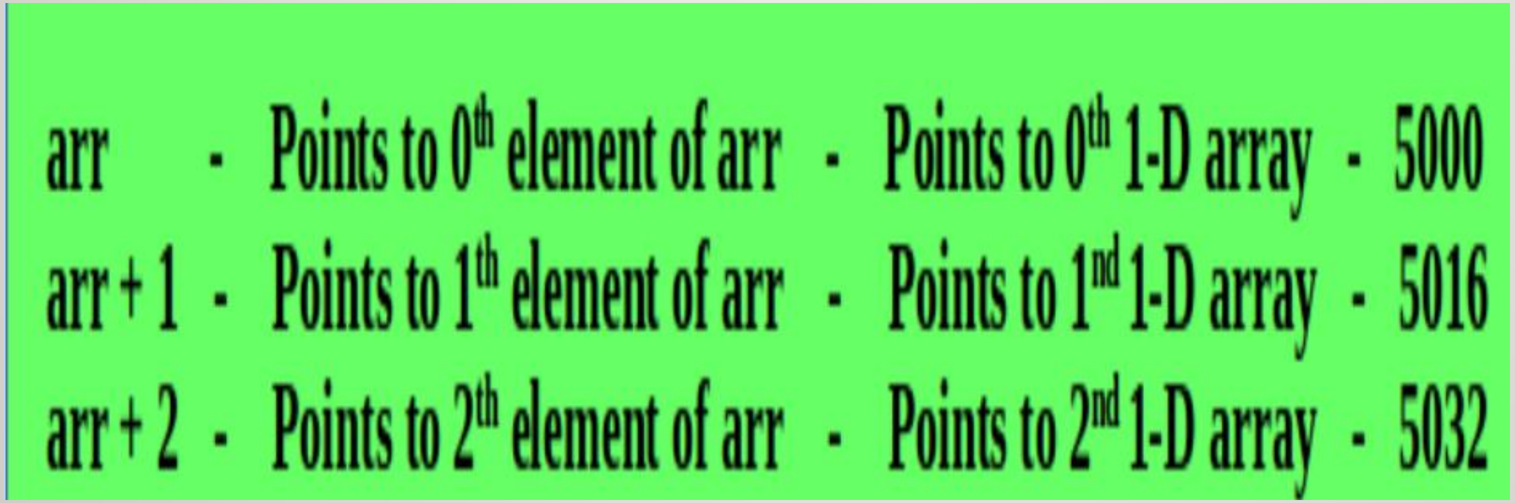
WHAT IS POINTER

1 Introduction to pointer

1.9 Pointer and related to (2D)

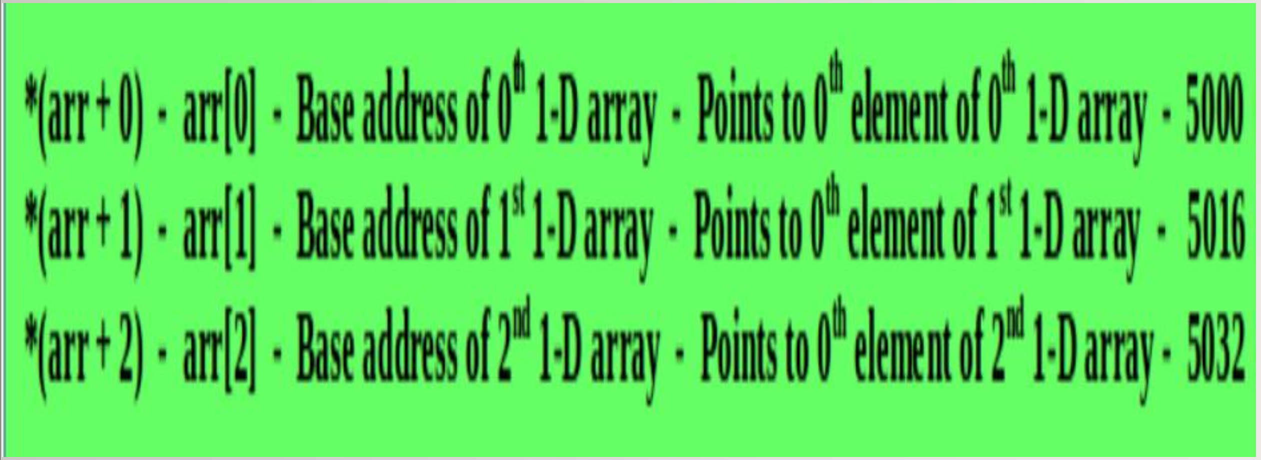
- We know that the name of an array is a constant pointer that points to 0th 1-D array and contains address 5000. Since *arr* is a 'pointer to an array of 4 integers', according to pointer arithmetic the expression arr + 1 will represent the address 5016 and expression arr + 2 will represent address 5032.

- So we can say that *arr* points to the 0th 1-D array, *arr + 1* points to the 1st 1-D array and *arr* + 2 points to the 2nd 1-D array.



arr	-	Points to 0th element of arr	-	Points to 0th 1-D array	-	5000
arr + 1	-	Points to 1th element of arr	-	Points to 1nd 1-D array	-	5016
arr + 2	-	Points to 2th element of arr	-	Points to 2nd 1-D array	-	5032

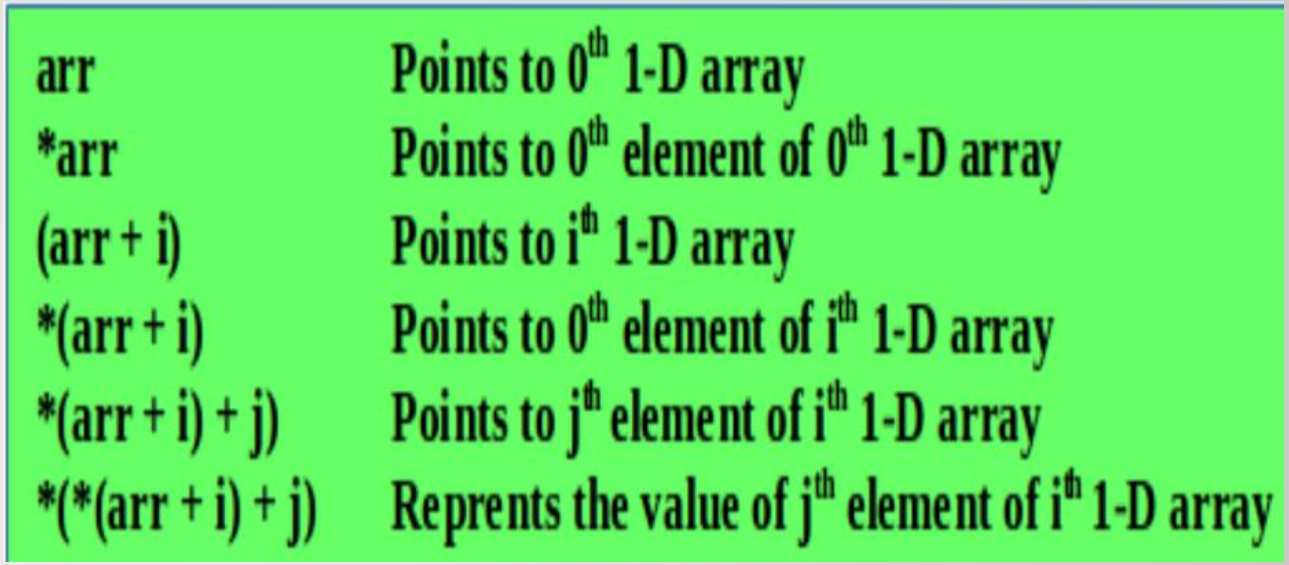
1.9 Pointer and related to (2D)



*(arr + 0) - arr[0] - Base address of 0th 1-D array - Points to 0th element of 0th 1-D array - 5000

*(arr + 1) - arr[1] - Base address of 1st 1-D array - Points to 0th element of 1st 1-D array - 5016

*(arr + 2) - arr[2] - Base address of 2nd 1-D array - Points to 0th element of 2nd 1-D array - 5032



arr	Points to 0th 1-D array
*arr	Points to 0th element of 0th 1-D array
(arr + i)	Points to ith 1-D array
*(arr + i)	Points to 0th element of ith 1-D array
*(arr + i) + j)	Points to jth element of ith 1-D array
((arr + i) + j)	Reprents the value of jth element of ith 1-D array

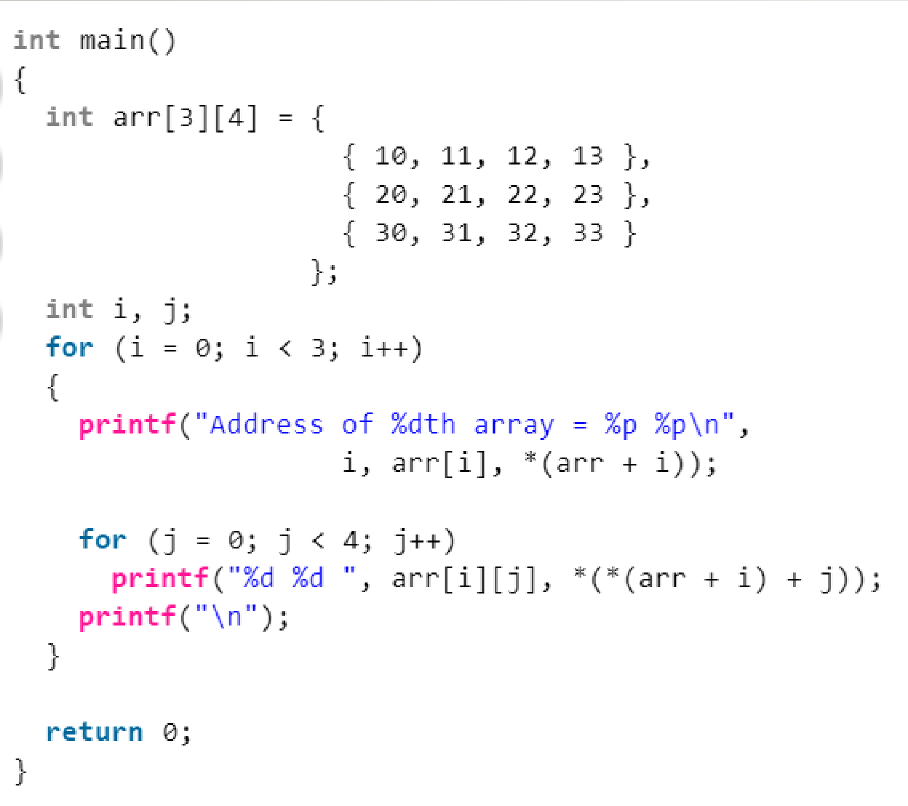
- Since arr + i points to i^{th} element of *arr*, on dereferencing it will get i^{th} element of *arr* which is of course a 1-D array. Thus the expression **(arr + i)* gives us the base address of i^{th} 1-D array.
- We know, the pointer expression **(arr + i)* is equivalent to the subscript expression *arr[i]*. So **(arr + i)* which is same as *arr[i]* gives us the base address of i^{th} 1-D array.

WHAT IS POINTER

1 Introduction to pointer

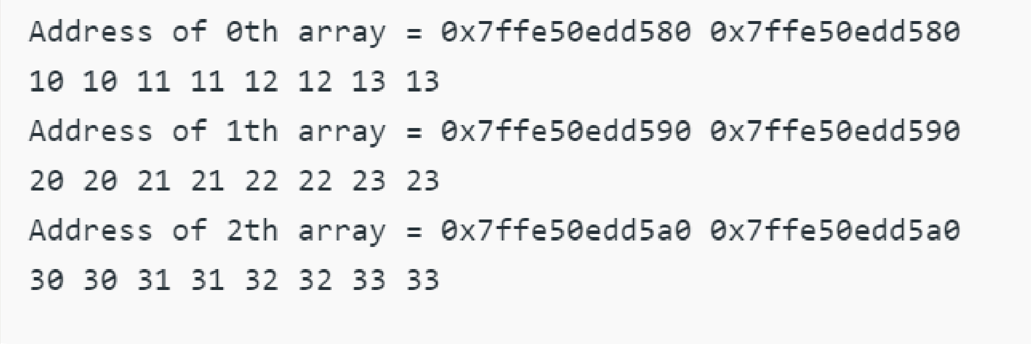
1.9 Pointer and related to (2D)

CODE



```
int main()
{
  int arr[3][4] = {
                    { 10, 11, 12, 13 },
                    { 20, 21, 22, 23 },
                    { 30, 31, 32, 33 }
                  };
  int i, j;
  for (i = 0; i < 3; i++)
  {
    printf("Address of %dth array = %p %p\n",
                    i, arr[i], *(arr + i));

    for (j = 0; j < 4; j++)
      printf("%d %d ", arr[i][j], *(*(arr + i) + j));
    printf("\n");
  }

  return 0;
}
```

OUTPUT



```
Address of 0th array = 0x7ffe50edd580 0x7ffe50edd580
10 10 11 11 12 12 13 13
Address of 1th array = 0x7ffe50edd590 0x7ffe50edd590
20 20 21 21 22 22 23 23
Address of 2th array = 0x7ffe50edd5a0 0x7ffe50edd5a0
30 30 31 31 32 32 33 33
```

DESCRIPTION

- In this example discuss 2D array