Variables, Data types and Constants

Variable

Data is stored in a computer's memory. The memory system comprises of uniquely numbered cells called memory addresses. We need to know the address where something is stored in order to retrieve it and work on it. A programming language frees us from keeping track of these memory addresses by substituting names for them. These names are called variables. **Variables are descriptive names for the memory addresses.**

Before we use a variable in C we must declare it. We must identify what kind of information will be stored in it. This is called defining a variable. Variables must be declared at the start of any block of code, but most are found at the start of each function (including main function)

All variable definitions must include two things variable name and its data type. Some rules to be followed in naming a variable in C are, it must start with an alphabet and can contain letter, underscores and digits. Both uppercase and lowercase letters can be used. Spaces should not be used in a variable name. Certain keywords like *int, float, , if, while ,switch etc* cannot be used as variable names.

Identifier

Identifiers are names that are used in programs for functions, parameters, variables, constants, and types etc. An **identifier** consists of a sequence of letters, digits, and underscores that does not begin with a digit. An identifier cannot be a reserved keyword.

Key Differences between Identifier and Variable

Both an identifier and a variable are the names allotted by users to a particular entity in a program. The identifier is only used to identify an entity uniquely in a program at the time of execution whereas, a variable is a name given to a memory location that is used to hold a value.

Variable is only a kind of identifier, other kinds of identifiers are function names, class names, structure names, etc. So it can be said that all variables are identifiers whereas, vice versa is not true.

Constant

Constants refer to the fixed values that do not change during the execution of a program. A "constant" is a number, character, or character string that can be used as a value in a program. There may be a situation in programming that the value of certain variables to remain constant during execution of a program. There are three ways for defining a constant in C. First method is by using #define statements. . It is used to define constants as follows

```
#define TRUE 1
#define FALSE 0
#define PI 3.1415
```

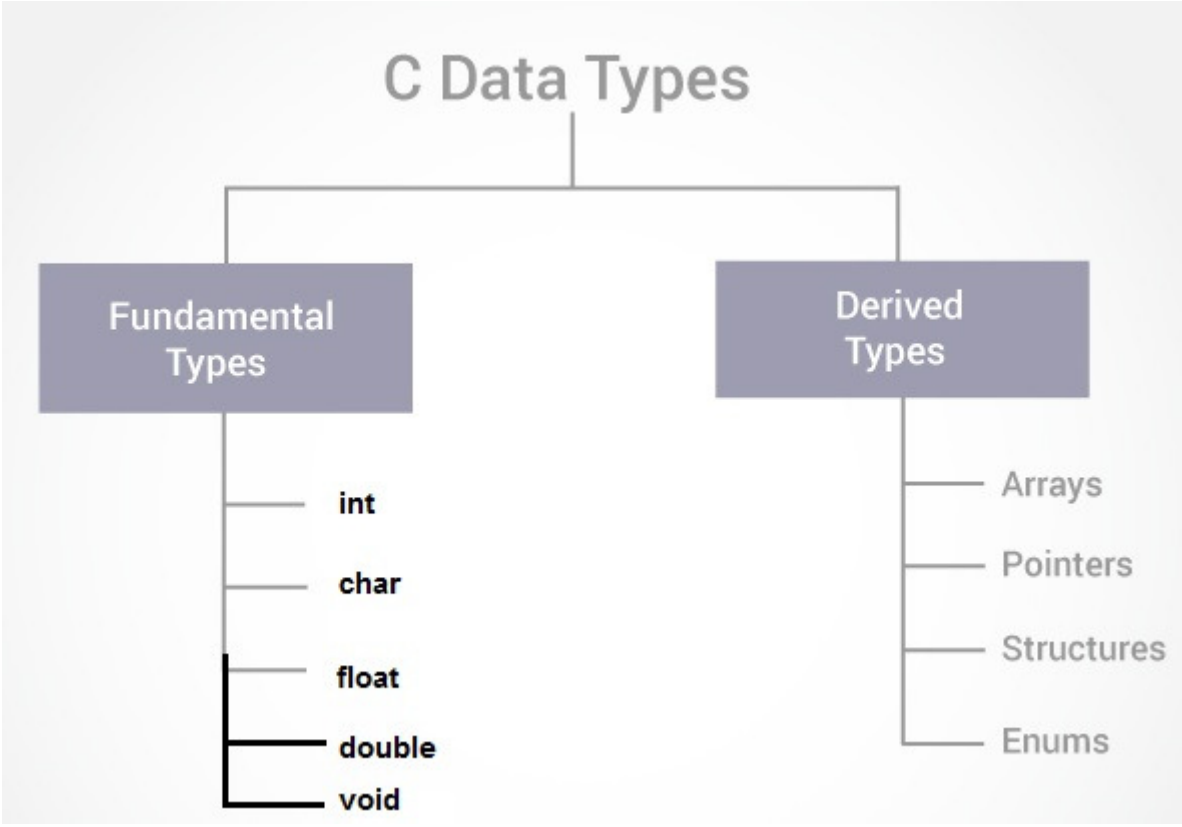
wherever the constant appears in a program, the compiler replaces it by its value.

The second method is by using the const keyword in conjunction with a variable declaration.

```
const float pie =3.147;
const int radius =4;
const char c = 'A';
```

Data types

Data types simply refer to the type and size of data associated with variables and functions. In C programming, variables should be declared before it can be used. Similarly, a function also needs to be declared before use.



Fundamental Data Types.

int - Integers are whole numbers that can have both positive and negative values but no decimal values. Example: 0, -5, 10. The size of int is either 2 bytes (In older PC's) or 4 bytes.

int a=5

float - Floating type variables can hold real numbers such as: 2.34, -9.382, 5.0 etc. You can declare a floating point variable in C by using either float or double keyword. The size of float (single precision float data type) is 4 bytes.

float f=5.66

double - a double-precision floating point value. The size of double (double precision float data type) is 8 bytes.

double f=6.6666

char - a single character. The size of character variable is 1 byte.

char test = 'h';

void - valueless special purpose type which used for representing function with no return values

Derived Data Types

Derived data types are object types which are aggregates of one or more types of basic data types.

Array: An array is a collection of the same basic types of data that is contiguous in memory.

int a[10]

Pointer: Pointer is variable which can hold the memory address of any variable.

int *p=&a;

Structure: Structures are collection of different data types, but with the concept of representing all contained information as a single object.

C Qualifiers

Qualifiers alter the meaning of base data types to yield a new data type.

Size qualifiers: Size qualifiers alter the size of a basic type. There are two size qualifiers, long and short.

size of long int is atleast 4 bytes.

size of long double is 10 bytes.

short is no longer than int, which is no longer than long.

Sign qualifiers: Integers and floating point variables can hold both negative and positive values. However, if a variable needs to hold positive value only, **unsigned** data types are used. For example:

unsigned int positiveInteger;

There is another qualifier **signed** which can hold both negative and positive only. However, it is not necessary to define variable signed since a variable is signed by default.

Constant qualifiers : An identifier can be declared as a constant. To do so const keyword is used.

const int cost = 20;

The value of cost cannot be changed in the program.

Volatile qualifiers: A variable should be declared volatile whenever its value can be changed by some external sources outside the program. Keyword volatile is used for creating volatile variables

Operators in C

C language offers many types of operators. They are,

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Bit wise operators
5. Conditional operators (ternary operators)
6. Increment/decrement operators

7. Assignment operators
8. Special Operators

1. Arithmetic operators

C Arithmetic operators are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus in C programs.

Arithmetic Operators/Operation	Example
+ (Addition)	A+B
– (Subtraction)	A-B
* (multiplication)	A*B
/ (Division)	A/B
% (Modulus)	A%B

2. Relational operators
   Relational operators are used to find the relation between two variables. i.e. to compare the values of two variables in a C program.

Operators	Example/Description
>	x > y (x is greater than y)
<	x < y (x is less than y)
>=	x >= y (x is greater than or equal to y)
<=	x <= y (x is less than or equal to y)
==	x == y (x is equal to y)
!=	x != y (x is not equal to y)

3. Logical operators: These operators are used to perform logical operations on the given expressions. There are 3 logical operators in C language. They are, logical AND (&&), logical OR (||) and logical NOT (!).

Operators	Example/Description
&& (logical AND)	(x>5)&&(y<5) It returns true when both conditions are true
|| (logical OR)	(x>=10)||(y>=10) It returns true when at-least one of the condition is true
! (logical NOT)	!((x>5)&&(y<5)) It reverses the state of the operand "((x>5) && (y<5))" If "((x>5) && (y<5))" is true, logical NOT operator makes it false

4. Bit wise operators
   These operators are used to perform bit operations. Decimal values are converted into binary values which are the sequence of bits and bit wise operators work on these bits.
   Bit wise operators in C language are & (bitwise AND), | (bitwise OR), ~ (bitwise NOT), ^ (bitwise XOR), << (left shift) and >> (right shift).

5. Conditional operators
   Conditional operators return one value if condition is true and returns another value is condition is false.
   This operator is also called as ternary operator.
   Syntax : (Condition? true_value: false_value);
   Example : (A > 100 ? 0 : 1);
   In above example, if A is greater than 100, 0 is returned else 1 is returned. This is equal to if else conditional statements.

6. Increment operators
   Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.
   Syntax:
   Increment operator: ++var_name; (or) var_name++;
   Decrement operator: – -var_name; (or) var_name – -;
   Example:
   Increment operator : ++ i ; i ++ ;
   Decrement operator : – – i ; i – – ;
7. Assignment operator
   = is the assignment operator in C. The RHS value is copied to LHS of the assignment operator.

8. Special Operators
   Below are some of the special operators that the C programming language offers.

Operators	Description
&	This is used to get the address of the variable. Example : &a will give address of a.
*	This is used as pointer to a variable. Example : * a where, * is pointer to the variable a.
Sizeof ()	This gives the size of the variable. Example : size of (char) will give us 1.

Precedence and associatively of operators in C

Precedence of operators

If more than one operator is involved in an expression, C language has a predefined rule of priority for the operators. This rule of priority of operators is called operator precedence.

In C, precedence of arithmetic operators(*, %, /, +, -) is higher than relational operators(==, !=, >, <, >=, <=) and precedence of relational operator is higher than logical operators(&&, || and !).

Associativity of operators

If two operators of same precedence (priority) are present in an expression, **Associativity** of operators indicates the order in which they execute.

The table below shows all the operators in C with precedence and associativity.

Note: Precedence of operators decreases from top to bottom in the given table.

Operator	Meaning of operator	Associativity
() []	Functional call Array element reference	Left to right
! ~ + - ++ -- & * sizeof	Logical negation Bitwise(1 's) complement Unary plus Unary minus Increment Decrement Dereference Operator(Address) Pointer reference Returns the size of an object	Right to left
* / %	Multiply Divide Remainder	Left to right
+ -	Binary plus(Addition) Binary minus(subtraction)	Left to right
<< >>	Left shift Right shift	Left to right
< <= > >=	Less than Less than or equal Greater than Greater than or equal	Left to right
== !=	Equal to Not equal to	Left to right
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
|	Bitwise OR	Left to right
&&	Logical AND	Left to right
||	Logical OR	Left to right
?:	Conditional Operator	Left to right
=	Assignment operator	Right to left

ARRAY

An array is a collection of data that holds fixed number of values of same type. For example: if you want to store marks of 100 students, you can create an array for it.

```
float marks[100];
```

The above marks array can store a maximum of 100 elements of the type int

The size and type of arrays cannot be changed after its declaration, trying to read/write an element from the array beyond its limits causes errors in the program (Array out of bound errors)

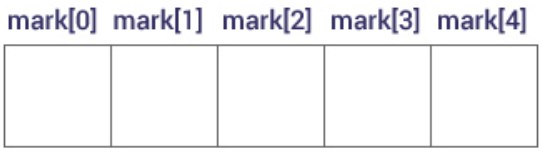
Arrays are of two types:

1. One-dimensional arrays
2. Multidimensional arrays

Elements of an Array and How to access them?

You can access elements of an array by indices.

Suppose you declared an array `mark` as above. The first element is `mark[0]`, second element is `mark[1]` and so on.



Few key notes:

- Arrays have 0 as the first index not 1. In this example, `mark[0]`
- If the size of an array is `n`, to access the last element, `(n-1)` index is used. In this example, `mark[4]`

How to insert and print array elements?

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

// insert different value to third element
mark[3] = 9;

// take input from the user and insert in third element
scanf("%d", &mark[2]);
```

```
•
•  // print first element of an array
•  printf("%d", mark[0]);
•
•  // print ith element of an array
•  printf("%d", mark[i-1]);
```

Basic Array operations.

Array insertion

Deleting an element from array

Array searching

Array sorting

Study the programs for array operations, which we have done in the lab,

Also study the programs for passing arrays to functions, ie array operation using function.

Matrix is an example for 2 dimensional array.

Eg : int A[10]1[10]

This array can store a maximum of 100 elements and the array index varies from A[0][0] to A[99][99]

Study the various matrix operations that we have done in the lab.

Read and print a matrix

Row sum of matrix

Colum sum of matrix

Sum of diagonal elements of a matrix

Transpose of a matrix

Matrix Addition

Matrix multiplication

Functions in C

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main ()**, and all the most trivial programs can define additional functions. The main advantage of function is that they make the programming simple. Once a function is declared and defined, the user can call the same function with different parameters and hence making the programming simple.

There are basically two types of functions in C.

1. Standard Library functions or built in functions
   These functions are provided by the C language itself. Eg: clrscr(), printf(), scanf(),getch(),strlen(), strcpy() etc….
2. User Defined Functions
   User can define their own functions.

There are basically three parts for a user defined function.

1) Function declaration (Function prototype)
   A function declaration tells the computer about a function's name, return type, and no of parameters and their type. Usually we declare function at the top of the main programme.
   Eg: int sum (int, int);

2).Function calling

   Usually we make the function call form the main programme.

   Eg: s=sum(2,3)

   The parameters which are provided in the function calling statements are called actual parameters, here 2 and 3. When the function is called the programme control is transferred to the function definition.

3. Function Defenition.

   A function definition provides the actual body of the function. That is it contains the code for what the function should do. The parameters which occur in the function definition is known as formal parameters.

   Eg: int sum (int a, int b)

```
{
int sum;
sum=a+b
return (sum);
}
```

And whenever a function call occurs, the values of the actual parameters are copied to the formal parameters. Here a and be are the formal parameters and when the function sum(2,3) is called, a gets the value of and b gets the value of 3.

This type of function calling is known as **call by value method**, where the value of actual parameters gets copied to formal parameters. And **whatever the changes that we make to the formal parameters are not going to reflected back to the actual parameters.** Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. In the above example variables sum, a and be are local variables and not visible from any other functions, including main function.

When the codes in the function definition are completed the program control is transferred back to the main programme.

There are two types of function calling mechanism in C.
Call by value and call by reference.

The call by value method is already discussed above. In call by reference method instead of passing the value, the reference or the address of the variable is passed. And this address is copied to the formal parameters. So the formal parameters should be pointers so that they can store the addresses. The main concept behind the call by reference method is that since the address of the variable is passed instead of value, whatever the changes that the function makes at this address, will be visible from any functions.
Eg:

```
#include<stdio.h>
void swap(int *x, int *y);    ## function decleration
 void  main () {
   int a = 100;
   int b = 200;
   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );
   swap(&a, &b);   ## function calling, &a-> address of a
   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );
}
```

```
 void swap(int *x, int *y) ## function definition , x and y are pointer variables

{
```

```
   int temp;
   temp = *x;    ## * operator gets the value saved at a particular address
   *x = *y;      ## put y into x
   *y = temp;    ## put temp into y

   return;
}
```

Output

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :200
After swap, value of b :100
```

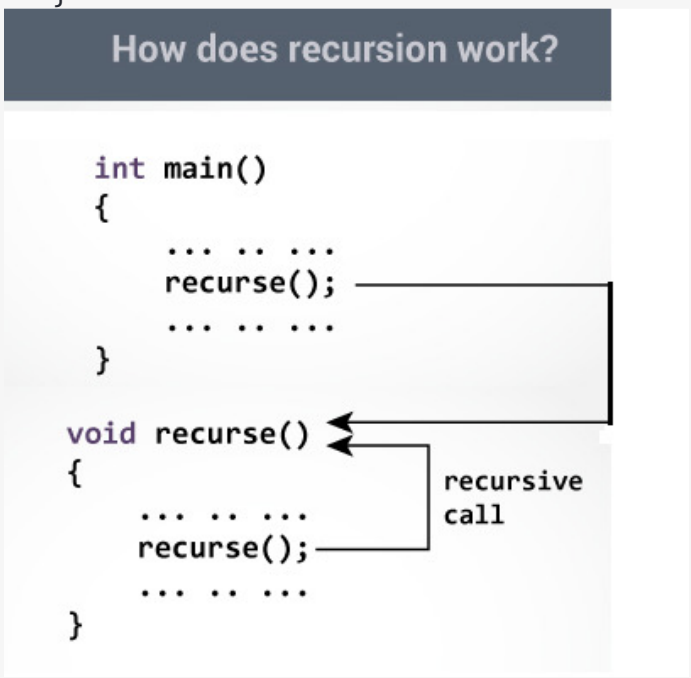
If we were using the call by value method , when we are printing the valued of a and b form the main programme, the values would not be swapped.

Recursion

A function that calls itself is known as a recursive function. And, this technique is known as recursion.

How recursion works?

```
int main()
{
   ... .. ...
   recurse();
   ... .. ...
}
void recurse()
{
   ... .. ...
   recurse();
   ... .. ...
}
```



The recursion continues until some condition is met to prevent it.To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call and other doesn't.

Example: Factorial of a Number Using Recursion

```
#include <stdio.h>
int factorial(int n);
void  main()
{
    int n;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    printf("Factorial=%d”,factorial(n));
    }
int factorial(int n)
{
    if (n >= 1)
        return n*factorial(n-1);## recursive call,
    else
        return 1;
}
```

The above program works in the following way.
factorial(3)
return (3*factorial(2)) ##since 3>1, return n * factorial of(n-1)= return 3* factorial(2)
return (3* return(2*factorial(1)))
return(3*return(2*1))
return(3*2)
=6

Study the programs for passing arrays and strings(char array) into functions, which we have discussed in the class.

Entry controlled loops and Exit controlled loops in C

Entry controlled Loop are those which check the condition at the time of entry. ***For loop*** and the ***while loop*** are the example of the entry controlled loop.

In Entry control loop when the execution first enters into the block of the loop where it checks the condition if the condition is true then the body of the loop will be executed else if the condition is false it will not read the body of the loop and exits from the loop.

In **Exit control** loop the condition is checked at the time of exit from the loop. Exit control loop is do-while loop.

In exit control loop first its runs the body of the loop then check the condition at the time of exit of the loop. First it performs the body of the loop then checks the condition if the condition is correct it will return to the entry of the loop otherwise it will exit.

Note: The main difference between entry control and exit control loop is that, Even if the condition is false exit control loop or the do-while loop will run it once whereas the entry control loop will not read the body as in exit control loop it checks the condition at the time of exit.

Counter controlled loop in C

> For-loops are counter-controlled, meaning that they are normally used whenever the number of iterations is known in advance.

Nested loop in C

A loop inside another loop is called a nested loop. The depth of nested loop depends on the complexity of a problem. We can have any number of nested loops as required.

Examples: Two dimensional array operations, different pattern printing etc

2 D Matrix Multiplication Program

```
#include <stdio.h>
void main()
{
int m, n, p, q, c, d, k, sum = 0;
int first[10][10], second[10][10], multiply[10][10];

printf("Enter the number of rows and columns of first matrix\n");
```

```
scanf("%d%d", &m, &n);
printf("Enter the elements of first matrix\n");

for (  c = 0 ; c < m ; c++ )
for ( d = 0 ; d < n ; d++ )
scanf("%d", &first[c][d]);

printf("Enter the number of rows and columns of second matrix\n");
scanf("%d%d", &p, &q);

if ( n != p )
printf("Matrices with entered orders can't be multiplied with each other.\n");
else
  {
printf("Enter the elements of second matrix\n");

for ( c = 0 ; c < p ; c++ )
for ( d = 0 ; d < q ; d++ )
scanf("%d", &second[c][d]);

for ( c = 0 ; c < m ; c++ )
    {
for ( d = 0 ; d < q ; d++ )
      {
for ( k = 0 ; k < p ; k++ )
        {
sum = sum + first[c][k]*second[k][d];
        }

multiply[c][d] = sum;
sum = 0;
      }
    }

printf("Product of entered matrices:-\n");

for ( c = 0 ; c < m ; c++ )
    {
for ( d = 0 ; d < q ; d++ )
printf("%d\t", multiply[c][d]);

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

Strings in C

In C programming, array of characters is called a string. A string is terminated by a null character ‘/0’

Declaration of strings

Before we actually work with strings, we need to declare them first.

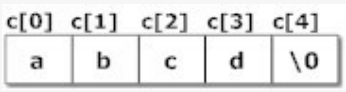
Strings are declared in a similar manner as arrays. Only difference is that, strings are of char type.

Eg: `char s[5];`

Initialization of strings

In C, string can be initialized in a number of different ways.

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

     OR,

char c[50] = "abcd";

     OR,

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

     OR,

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

OR

Char c[5];

C=”abcd”
```

And the array will be saved as 

Reading the Strings:

```
char c[20];

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

or

```
#include<stdio.h>
int main()
{
char name[30];
printf("Enter name: ");
gets(name);//Function to read string from user.
printf("Name: ");
puts(name);//Function to display string.
return0;
}
```

String Handling Functions in C

Function	Work of Function
strlen()	Calculates the length of string
strcpy()	Copies a string to another string
strcat()	Concatenates(joins) two strings
strcmp()	Compares two string
strlwr()	Converts string to lowercase
strupr()	Converts string to uppercase

Difference between scanf and gets

scanf terminates when it encounters white spaces.

gets can read the entire sentences

POINTER

A pointer is a variable which contains the address in memory of another variable. We can have a pointer to any variable type.

The ***address of operator, &*** gives the ``address of a variable".

The ***indirection*** or dereference operator ***** gives the ``contents of an object ***pointed to*** by a pointer".

We can declare a pointer to an integer variable as

```
int a=5;
int  *p;
p=&a
```

Now the pointer variable p is pointing to address of the integer variable a.

We can print the value of p as ***printf("%x",p)*** . If ***a*** was at the address 1000H, it will print 1000H. (we use %x to denote that the p stores an address(Hexadecimal))

Now applying the indirection operator *, we can get the content of the address pointed by p. Hence ***printf("%d",*p)*** will be printing 5, the value of a.

if we write *p=6, then now the value of a will be changed to 6.

Pointer Arithmetic

The following are valid pointer arithmetic operations

1. Addition of integer to a pointer

p=p+1 will increment the value of p by one. Hence if p was an integer pointer and p was initially pointing to 1000H, p=p+1 or ++p or p++ will change the value of p to 1002 H, since the size of an integer is 2 bytes.

if p was a pointer to a floating point value, then ++p will make the address to be incremented by 4 bytes. (ie if initiall p was 1000H, the p++ will make p to point to 1004 H)

if p was a pointer to a character variable then address will be incremented by 1 byte

2. Subtraction of integer to a pointer

Decrementing the pointer values also will be done in the same fashion.

3. Subtracting one pointer from another of the same type

If we have two pointers p1 and p2 of base type pointer to int with addresses 1002 and 1000 respectively, then p2 - p1 will give 1, since the size of int type is 2 bytes. If you subtract p2 from p1 i.e p1 - p2 then answer will be negative i.e -1.

4. comparision of pointers

we can compare two pointers using relational operator. You can perform six different type of pointer comparison <, >, <=, >=, == and !

This is known as pointer arithmetic or address arithmetic

Relation between pointers and arrays

Pointers and arrays in C are closely related.

The basic idea is that any array name in C is a pointer to its base address.

Lets consider the integer array a

int a[10];

by definition a=a[0] (Array name is the pointer to its base address (starting address))

Hence a+1 will point to a[1], a+2 will point to a[2] and so on.

*a(a+1) will get the value at a[1]

So we can write *(a+1) in place of a[1]

Therefore any array operations can be done using pointers as described below.

Write the program for array operation using pointers, that we have done in lab.

Also when passing array to function, we are passing array name, since array name is a pointer, passing array to function uses pass by reference method, not pass by value.

Matrix Multiplication

```
# include <stdio.h>
# include <conio.h>
void main()
{
  int mata[10][10], matb[10][10], matc[10][10] ;
  int i, j, k, row1, col1, row2, col2 ;
  clrscr() ;
  printf("Enter the order of first matrix : ") ;
  scanf("%d %d", &row1, &col1) ;
  printf("\nEnter the order of second matrix : ") ;
  scanf("%d %d", &row2, &col2) ;
  if(col1 == row2)
  {
     printf("\nEnter the elements of first matrix : \n\n") ;
     for(i = 0 ; i < row1 ; i++)
     {
          for(j = 0 ; j < col1 ; j++)
          {
                 scanf("%d", &mata[i][j]) ;
           }
     }
     printf("\nEnter the elements of second matrix : \n\n") ;
     for(i = 0 ; i < row2 ; i++)
     {
         for(j = 0 ; j < col2 ; j++)
          {
                scanf("%d", &matb[i][j]) ;
          }
     }

    for(i = 0 ; i < row1 ; i++)
    {
      for(j = 0 ; j < col2 ; j++)
      {
        matc[i][j] = 0 ;
        for(k = 0 ; k < col1 ; k++)
          matc[i][j] = matc[i][j] + mata[i][k] * matb[k][j] ;
      }
    }

    printf("\nThe resultant matrix is : \n\n") ;
    for(i = 0 ; i < row1 ; i++)
    {
      for(j = 0 ; j < col2 ; j++)
      {
        printf("%d \t", matc[i][j]) ;
      }
      printf("\n") ;
    }
  }
  else
    printf("\nMatrix Multiplication is not possible ...") ;
  getch() ;
}
```