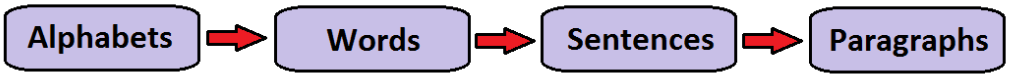
Topics Covered

What is C?

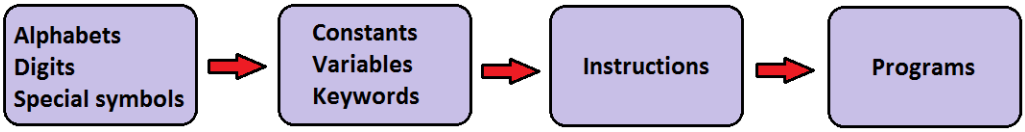
- C is a computer programming language that you can use to create lists of instructions for a computer to follow.
- C programming is widely used because of its efficiency and control.

How we can learn C?

- Steps in learning English language



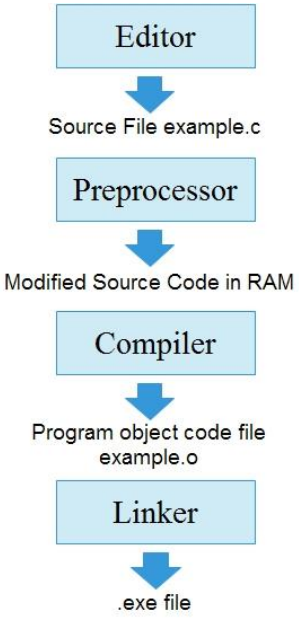
- Steps in learning Clanguage



Writing C programs

- A programmer uses a text editor to create ormodify files containing C code.
- Code is also known as source code.
- A file containing source code is called a source file.
- After a C source file has been created, the programmer must invoke the C compiler before the program can be executed (run).

Program Development



Stage 1:
Preprocessing:

- Performed by a program called the **preprocessor**
- Modifies the source code (in RAM) according topreprocessor directives orcommands embedded in the source code.

Stage 2:
Compilation:

- Performed by a program called the **compiler**
- Compiler translates the preprocessor-modified source codeinto **object code.**
- Object code is a set of instruction that is understood by a computer at the lowest hardware level.
- Checks for **syntax errors** and **warnings**
- Saves the object code to a disk file, if instructed todo so.
    - If any compiler errors are received, no object code file will begenerated.
    - An object code file will be generated if only warnings, not errors,are received

Stage 3:
Linking:

- Combines the program object code with other object code to produce the executable file.
- The other object code can come from the Run-Time library, other libraries, or object files that you have created.
- Saves the executable code to a disk file. On Windows system, that file is called **example.exe**

Let's look at a simple C program to find area of a circle: (example.c)

```
#include<stdio.h>        // C preprocessing directive #include includes 'stdio.h' header file
#include<conio.h>        //C preprocessing directive #include includes 'conio.h' header file
#define PI 3.1415        // PI is given a value which will be constant throughout the program.
int main()               // The execution of every C program starts from this main() function.
{
int radius;
float area;
printf("Enter the radius: ");
scanf("%d",&radius);
    area=PI*radius*radius;
printf("Area=%.2f",area);
getch();
return0;
}
```

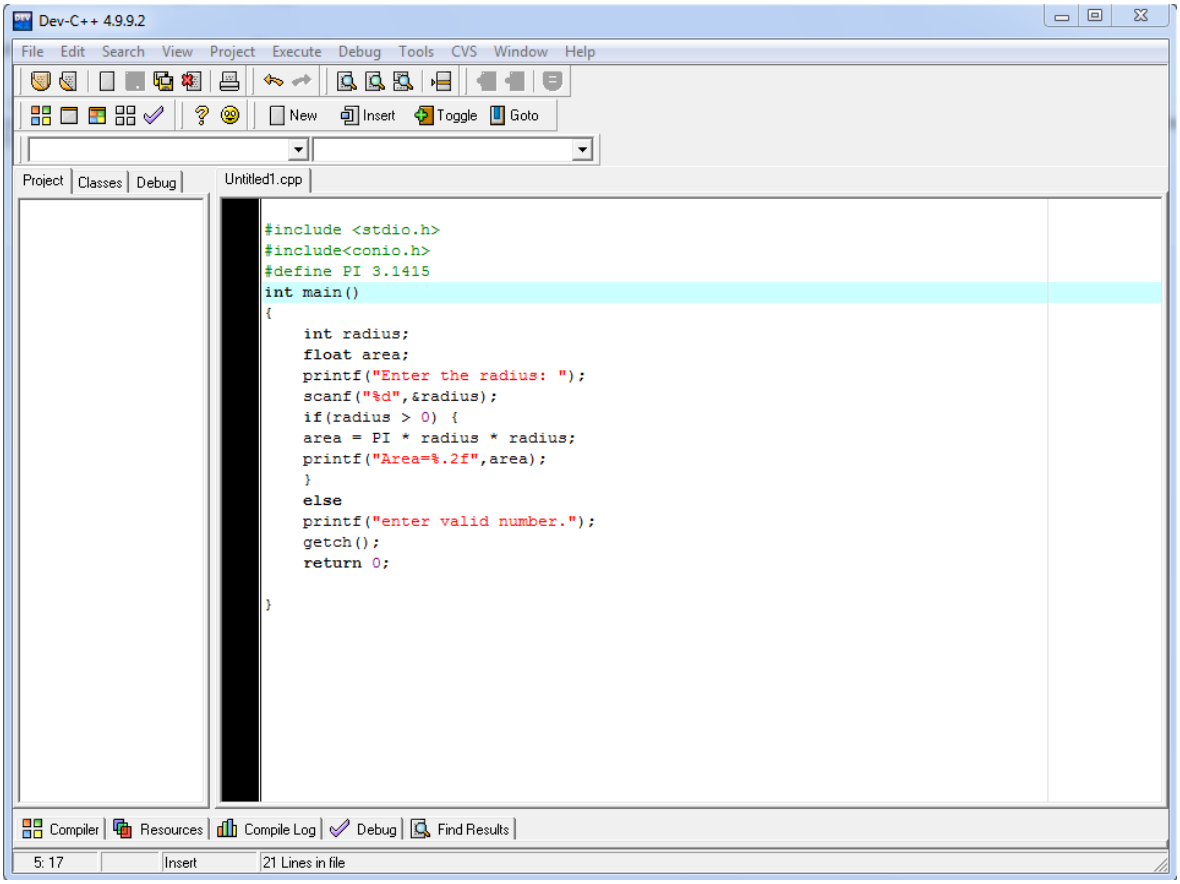
Output:
Enter the radius:3
Area=28.27

How to compile and run a program:

Here, we have used Dev C++ software for C programming.

You can download and install the software from:
http://sourceforge.net/projects/dev-cpp/files/Binaries/Dev-C%2B%2B%204.9.9.2/devcpp-4.9.9.2_setup.exe/download?use_mirror=kaz

In the programming canvas, you can write code as shown below:



Click on execute button in the tool bar and click compile to check the program. If it shows 0 errors and 0 warnings found then, click on run to run the program.

In the further part of the tutorial, you will be learning the syntax and semantics of C programming.

Preprocessors:

- The preprocessor provides the ability for the inclusion of header files, conditional compilation, define constants.
- The **C Preprocessor** is not part of the compiler, but is a separate step in the compilation process.
- Line that begins with # (pound) are called preprocessing directives.

In the above program, the first line is

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

where, stdio.h is the header file and the preprocessor replaces the above line with the contents of header file. The above line tells the compiler to include information about the standard input/output library to perform the task.

For example: Consider the printf() function, which is used to display data on the screen. The definition for this function is included in the header file i.e. a program code written to access your screen to allow data to be displayed on it.

```
#include<conio.h>
```

where, conio.h is the header file. The above line tells the compiler to include information about the console input/output library to perform the task.

#define statement:
#define directive causes the compiler to substitute *token-string* for each occurrence of *identifier* in the source file. Identifiers are the names given to variables, constants, functions and user-defined data.
Syntax:

```
#define identifier token_string
```

For e.g.:

In the above code to find area of circle, we have used

```
#define PI 3.1415
```

Here, the token string in above line 3.1415 is replaced in every occurrence of symbolic constant `PI`.

Variables and Data Types

- In C, variables are memory location in computer's memory to store data.
- Variable should be declared before it can be used in program.
- Data types are the keywords, which are used for assigning a type to a variable.

Data Types are of two types:

a. Fundamental Data Types

   i. Integer type
   ii. Floating type
   iii. Character type

b. Derived Data Types

   i. Arrays
   ii. Pointers
   iii. Structures
   iv. Enumeration

Syntax to declare a variable:

```
data_typevariable_name;
```

Rules for writing variable name in C

a. Variable name can be composed of letters (both uppercase and lowercase letters), digits and underscore '_' only.
b. The first letter of a variable should be either a letter or an underscore.

Integer Type:

- Keyword int is used for declaring the variable with integer type

```
int radius;
```

- `radius`is a variable of type integer.

Floating type:

- Variable of floating type can hold real numbers such as: 2.34, -9.382 etc.
- Float or double is used for declaring floating type variable. For example:

```
float area;
```

In our file example.c, variable area will store a real number, ie. area=28.27.

Character types:

- Keyword char is used for declaring the variable of character type. For example:

```
char var4='h';
```

Constants

Constants are the identifiers whose values can't be changed during the execution of a program. const keyword can be used to declare constant with specific type.

Integer constants:

Integer constants are the numeric constants (constant associated with number) without any fractional part or exponential part. There are three types of integer constants in C language: decimal constant (base 10), octal constant (base 8) and hexadecimal constant (base 16).

Decimal digits: 0 1 2 3 4 5 6 7 8 9
Octal digits: 0 1 2 3 4 5 6 7
Hexadecimal digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F
For e.g.:

```
constint var1= 10;
cons int var2= 0x7F;
```

Floating-point constants:

Floating point constants are the numeric constants that have either fractional form or exponent form.
For e.g.:

```
const float var1= -2.0;
```

Character constants:

Character constants are the constant which use single quotation around characters.
For e.g.:

```
const char var1= 'a';
```

String constants

String constants are the constants which are enclosed in a pair of double-quote marks.
For e.g.:

```
const char var1[]= “Hello”;
```

Operators in C programming

Operators are the symbol which operates on value or a variable. For example: + is an operator to perform addition. Different types of operators are as follows:

Arithmetic Operators:

We can use the following operators to do the arithmetic operations.

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division(modulo division)

In our example.c code, we have used multiplication operator(*) to get πr^2.
i.e.: `area=PI*radius*radius;`

Increment and Decrement Operators:

In C, ++ and -- are called increment and decrement operators respectively.
Letvar=5

```
var++;//a becomes 6
var--;//a becomes 5
++var;//a becomes 6
--var;//a becomes 5
```

Note:
When ++ is used as prefix(like: ++var), ++var will increment the value of *var* and then return it but, if ++ is used as postfix(like: var++), operator will return the value of operand first and then only increment it.

Assignment Operators:

The most common assignment operator is "=". This operator assigns the value present in right side to the left side.

```
area=PI*radius*radius;
```

Result of (PI*radius*radius) is assigned to variable `area`.
Using assignment operators, we can assign the values to operand as shown in below table:

Operator	Used as	Same as
=	a=b	a=b
+=	a+=b	a=a+b
-=	a-=b	a=a-b
*=	a*=b	a=a*b
/=	a/=b	a=a/b
%=	a%=b	a=a%b

Relational Operators:
Relational operators checks relationship between two operands. If the relation is true, it returns value 1 and if the relation is false, it returns value 0.

Relational operators are used in decision making and loops in C programming.

Operator	Meaning of Operator
==	Equal to
>	Greater than
<	Less than
!=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

In our example,

```
#include<stdio.h>
#define PI 3.1415
int main()
{
int radius;
float area;
printf("Enter the radius: ");
scanf("%d",&radius);
if(radius>0) {
    area=PI*radius*radius;
printf("Area=%.2f",area);
    }
    else {
printf("enter valid number.");
return0;
}
```

If radius is greater than 0 then only it will calculate the area of circle else it will display- enter valid number.

Logical Operators:
Logical operators are used to combine conditions containing relation operators.

Operator	Meaning of Operator
&&	Logical AND
||	Logical OR
!	Logical NOT

For e.g.:

```
main():
{
        int var1=40, var2=20, var3=15;
        if((var1>var2) && (var2>var3))
        //if both conditions are true then printf statement will be executed
        printf("AND executed");
        if((var1>40) || (var2==20))
        //Even if one condition is true printf statement will be executed
        printf("OR executed");
}
Output:
AND executed OR executed
```

Conditional Operator:

Conditional operator takes three operands and consists of two symbols- ? and : . Conditional operators are used for decision making in C.

```
var=(var>0)?10:-10;
```

In above statement, if*var* is greater than 0, value of *var*will be 10 but, if *var* is less than 0, value of *var* will be -10.

Bitwise Operators:

Bitwise operators are special types of operators that are used in programming the processor. A bitwise operator works on each bit of data, i.e. they are used in bit level programming. The bitwise operators are as follows:

Bitwise AND operator:

The output of bitwise AND is 1 if both the corresponding bits of operands is 1. If either of bit is 0 or both bits are 0, the output will be 0. In C Programming, bitwise AND operator is denoted by &.

In a bit pattern, to set a bit as 0, we can use AND operator.

For e.g.:
In a given 8 bit pattern, to set the 8th bit as 0 and remaining bits unchanged. We will be using the bitwise AND operation as shown below:

```
10001010        //hexadecimal value: 0x8A
```

To set 8th bit as 0, we will use bitwise AND operation using a 8-bit pattern where Most Significant Bit will be 0 and rest all bits 1 to keep remaining bits unchanged, i.e. 0111 1111.

```
  10001010      //hexadecimal value: 0x8A
& 0111 1111     //hexadecimal value: 0x7F
_________
  0000 1010     //hexadecimal value: 0x0A
```

Output: **0 0 0 0 1 0 1 0**

C code for above operation will be:

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

int main()
{
char a=0x8A,b=0x7F;
unsigned char c = a&b;
printf("Output=%x",c);
getch();
    return 0;
}
```

Output:0x0A

Bitwise OR Operator:

The output of bitwise OR operation is 1 if either of the bit is 1 or both the bits are 1. In C programming, bitwise OR operator is denoted by |.

In a bit pattern, to set a bit as 1, we can use OR operator.

For e.g.:

In a given 8 bit pattern, to set the 8th bit as 1 and remaining bits unchanged. We will be using the bitwise OR operation as shown below:

```
00001010        //hexadecimal value: 0x0A
```

To set 8th bit as 1, we will use bitwise OR operation using a 8-bit pattern where Most Significant Bitwill be 1 and rest all bits 0 to keep remaining bits unchanged, i.e. 1000 0000.

```
  00001010      //hexadecimal value: 0x0A
| 1000 0000     //hexadecimal value: 0x80
_________
  1000 1010     //hexadecimal value: 0x8A
```

Output : **1 0 0 0 1 0 1 0**

C code for above operation will be:

```
#include <stdio.h>
#include <conio.h>
int main()
{
 char a=0x0A,b=0x80;
 unsigned char c = a|b;
printf("Output=%x",c);
getch();
    return 0;
}
```

Output: 0x8A

Bitwise exclusive OR:

The output of bitwise XOR operation is 1 if the corresponding bits of two operands are opposite, i.e. if first operand is 1 then second operand should be 0. It is denoted by ^.

For e.g.:

The bitwise exclusive ORoperation of two hexadecimal numbers 0x0A and 0x14 will be:

```
0x0A = 00001010 //In Binary
0x14 = 00010100//In Binary

00001010
^00010100
 ________
00011110
```

C code for above operation will be:

```
#include <stdio.h>
#include <conio.h>
int main()
{
 char a=0x0A, b=0x14;
 unsigned char c = a^b;
printf("Output=%x",c);
getch();
   return 0;
}
```

Output: 0x1E

Bitwise complement:

It is a unary operator. It changes the corresponding bit of the operand to opposite bit, i.e., 0 to 1 and 1 to 0. It works on one operand only. It is denoted by ~.

```
0x0A =00001010//In Binary
```

Bitwise complement will be

```
~ 00001010
 ________
11110101
```

C code for above operation will be:

```
#include <stdio.h>
#include <conio.h>
int main()
{
 unsigned char c = 0x0A,d;
```

```
   d = ~c
printf("Output=%x",d);
getch();
   return 0;
}
```

Output: 0xF5

There are two shift operators in C, Right Shift and Left Shift Operator.

Right Shift Operator:

Right shift operator moves all bits towards the right by specified number of bits. It is denoted by >>.

For e.g.:

Binary representation of 0xF5 is`11110101`

0xF5>> 3 //right shift the number by 3 bits

`1111 0101>> 3`

`0001 1110`

i.e. 0x1E

C code for above operation will be:

```
#include <stdio.h>
#include <conio.h>
int main()
{
 unsigned char c = 0xF5,d;
   d = c >>3;
printf("Output=%x",d);
getch();
   return 0;
}
```

Left Shift Operator:

Left shift operator moves all bits towards the left by specified number of bits. It is denoted by <<.

For e.g.:

Binary representation of 0xF5 is`11110101`

`0xF5 <<3` //left shift the number by 3 bits

`1111 0101 << 3`

`10101000`

i.e. 0xA8

C code for above operation will be:

```
#include <stdio.h>
#include <conio.h>
int main()
{
   unsigned char c = 0xF5,d;
   d = c << 3;
printf("Output=%x",d);
getch();
   return 0;
}
```

Control Statements:

Control statementsare used by the programmer to specify one or more conditions to be evaluated or tested by the program.

If() statement:

The 'if()' statement checks whether the condition inside parenthesis ( ) is true or not. If the condition is true, statement/s inside the body of if statement is/are executed but if condition/s is/are false, statement/s inside body of 'if' is ignored.

The syntax is:

```
if (condition)
{
      statements;
}
```

For e.g:

```
if(radius>0)
{
    area=PI*radius*radius;
    printf("Area=%.2f",area);
}
```

If-else statement:

An **if-else** statement can be followed by an optional **else statement**, which executes when the logical condition is false.

The **if-else** statement is used to express decisions.

Formally the syntax is

```
if (condition)
{
      statements;
}
```

```
else
{
      statements;
}
```

For e.g.:

```
if(radius>0)
{
    area=PI*radius*radius;
    printf("Area=%.2f",area);
}
else
{
    printf("enter valid number.");
}
```

Nested if…else statements:

Nested if…else statements are usedwhen program requires more than one test conditions.You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).

The syntax of **else if** ladder is

```
if (condition)
   statement;
else if (condition)
   statement;
else if (condition)
   statement;
else if (condition)
   statement;
else
   statement;
```

For e.g.:
Let's write a program to compare the values of two variables.

```
if(a==b)
   printf("a is equal to b");
else if(a<b)
   printf("a is less than b");
else
   printf("a is greater than b");
```

Switch statement:
A **switch** statement allows a variable to be tested for equality against a list of values.
The syntax of switch statement is

```
switch (condition)
{
   case const-expr:
         statements;
         break;
```

```
        case const-expr:
              statements;
              break;
        default:
              statements;
              break;
    }
```

For e.g.:
Let's write a program to display the working of switch statement:

```
main()
{
char grade;
printf("Enter grade:");
scanf("%c",&grade);
switch(grade)
{
case'A':
printf("Excellent!\n");
break;
case'B':
printf("Very Good!\n");
break;

case'C':
printf("Well done\n");
break;
case'D':
printf("You passed\n");
break;
case'F':
printf("Better try again\n");
break;
default:
printf("Invalid grade\n");
}
printf("Your grade is  %c\n", grade );
}
```

Output:
Enter grade: A
Your grade is Excellent!

Conditional Loops:

There may be a situation, when you need to execute a block of code several number of times.A loop statement allows us to execute a statement or group of statements multiple times.

There are different types of conditional loops, which are as follows:

while loop:

This repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

The syntax of while is

```
while (condition)
{
   statements;
}
```

For e.g.:

```
x = 0;
while (x<10)
{
   x++;
   printf("%d\n",x);  //prints number from 0 to 9
}
```

In this loop, when the program counter reaches the curly bracket at the end it jumps back up to the beginning of the loop, which checks the condition again and decides whether to repeat the block another time, or stop and move to the next statement after the block.

do…while loop:

In do…while loop, the condition is tested at the end of the block instead of the beginning, so the block will be executed at least once. If the condition is true, program counter jumps back to the beginning of the block and execute it again.

A do..while loop is almost the same as a while loop except that the loop body is assured to execute at least once.

The syntax of **do while** is

```
do
{
   Statements;
} while (condition);
```

For e.g.:

```
intx = 0;
do
{
   printf("%d\n", x);
   x++;
} while (x>10);
```

In this case the condition is false still it will execute the statement inside do..while once, i.e., 0 will be displayed as output.

for loop:

A for loop allows code to be repeatedly executed.

The syntax of **for** is

```
for( initialization ; condition ; variable update )
{
   Statements;
}
```

The variable initialization allows you to either declare a variable and give it a value or give a value to an already existing variable.

The condition tells the program that while the condition is true the loop should continue to repeat itself.

The variable update section is the easiest way for a 'for' loop to handle changing of the variable.

It is possible to do things like x++, x = x + 10, or even you could call other functions that do nothing to the variable but still have a useful effect on the code.

A semicolon separates each of these sections. Also, every single one of the sections may be empty, though the semicolons still have to be there. If the condition is empty, it is evaluated as true and the loop will repeat until something else stops it.

For e.g.:

```
for (i=0;i<10;i++)
{
   printf("%d\n",i); //prints 0 to 9
}
```

Functions

A function is a segment that groups code to perform a specific task.

Types of C functions
Basically, there are two types of functions in C on basis of whether it is defined by user or not.

- Library function
- User defined function

Library function:

Library functions are the in-built function in C programming system. For example:

`main()` //The execution of every C program starts from this main() function.

main() is a special function. Your program begins executing at the beginning of main(). This means every program must have a main() somewhere. main()will usually call other functions to help perform its job, some that you wrote, and other from libraries that are provided for you.

`prinf()`//is used for displaying output in C

`scanf()`//is used for taking input in C.

User defined function:

C allows programmer to define their own function according to their requirement. These types of functions are known as user-defined functions.
Functions in C:

Declaration:

```
return_typefunction_name(data type of parameter-list );
```

Calling:

```
function_name( parameter-list );
```

Definition:

```
return_type function-name( parameter-list with data type )
{
      declarations and statements;
      return (value);
}
```

Let's write a program to demonstrate the working of user defined function in our example.c file:

```
#include<stdio.h>
#include<conio.h>
#define PI 3.14159

floatcirarea (int);                    //function prototype(declaration)

void main()
{
   float area;
   int radius;
   printf("Enter the radius=");
   scanf("%d",&radius);
   area = cirarea(radius);             //function call
   printf("area = %f", area);          //prints area of circle
   getch();
}

floatcirarea (int radius)              //function declaration
{
   float area;
   area = PI * radius * radius;        // calculates area of circle
   return area;                        //return statement of function
}
```

Arrays

An array is a collection of same type of elements which are grouped under a common name. In an array we can store multiple values and we have to remember just single variable name.

Arrays are of two types:

- One dimensional Array
- Multidimensional Array

Declaring single dimensional array:

Syntax:

```
arrayTypearrayName[ numberOfElements ];
```

Examples:

```
int data[ 10 ];
```

Here, the name of array is data. The size of array is 10, i.e., there are 10 items (elements) of array *age*.

data(0)	data(1)	data(2)	data(3)	data(4)	data(5)	data(6)	data(7)	data(8)	data(9)

Array Elements

Initializing an array:
An array can be initialized in many ways as shown in the code-snippets below.

- Initializing each element separately:
  For e.g.:

```
#define size 10
int main()
{
      intarr[size];
      inti = 0;
      for(i=0;i<size;i++)
      {
            arr[i] = i; // Initializing each element separately

      }
}
```

- Initializing array at the time of declaration:
  For example:

```
intarr[] = {'0','1','2','3','4','5','6','7','8','9'};
```

- Accessing Values in an Array:
  An array element is accessed as:

```
#define size 10
int main()
{
  intarr[size];
  inti = 0;
  for(i=0;i<size;i++)
  {
        arr[i] = i;// Initializing each element separately
  }
//Accessing the 6th element of integer array arr and assigning its value to integer 'j'.
  int j = arr[5];
```

Declaring a Multidimensional Array:
The general form of a multidimensional array declaration:

```
type name[size1][size2]...[sizeN];
```

The simplest form of the multidimensional array is the two-dimensional array.

```
typearrayName [ x ][ y ];
```

Initializing Two-Dimensional Arrays:
Multidimensional arrays may be initialized by specifying bracketed values for each row.
Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
 {0, 1, 2, 3},    // initializes for row indexed by 0
 {4, 5, 6, 7},    // initializes for row indexed by 1
 {8, 9, 10, 11}   // initializes for row indexed by 2
};
```

Accessing Two-Dimensional Array Elements:

An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example:

```
intval = a[2][3];
```

Let's write a program to find sum of two matrix of order 2*2 using multidimensional array:

```
#include <stdio.h>
int main(){
int a[2][2], b[2][2], c[2][2];
inti,j;
printf("Enter the elements of 1st matrix\n");
for(i=0;i<2;++i)
for(j=0;j<2;++j){
printf("Enter a[%d][%d]: ",i+1,j+1);
scanf("%d",&a[i][j]);
        }
printf("Enter the elements of 2nd matrix\n");
for(i=0;i<2;++i)
for(j=0;j<2;++j){
printf("Enter b[%d][%d]: ",i+1,j+1);
scanf("%d",&b[i][j]);
        }
for(i=0;i<2;++i)
for(j=0;j<2;++j){
c[i][j]=a[i][j]+b[i][j];   // Sum of corresponding elements of two arrays.
        }
printf("\nSum Of Matrix:");
for(i=0;i<2;++i)
for(j=0;j<2;++j){
printf("%d\t",c[i][j]);
        if(j==1)                // To display sum of matrix
        printf("\n");
              }
return 0;
}
```

Output will be:

Enter the elements of 1st matrix:
Enter a[1][1]:1
Enter a[1][2]:2
Enter a[1][3]:3
Enter a[1][4]:4
Enter the elements of 2nd matrix:
Enter b[2][1]: 1
Enter b[2][2]: 2
Enter b[2][3]: 3
Enter b[2][4]: 4

Sum of Matrix: 2 4

6 8

Number System

The digit of a number system is a symbol, which represents an integral quantity.
A number system of base (also called radix) R is a system, which hasR distinct symbols for R digits. Therefore, if the base is R, then R digits are used [0,1,2,3,...,R-1].
The base or radix of a number system is defined as the number of different digits, which can occur in each position in the number system.

For e.g.:
Consider a decimal number (base 10) - 1415, then we can represent these number as

$1415 = 1*10^3 + 4*10^2 + 1*10^1 + 5*10^0$

There are many ways to represent a number. The four commonly associated number systems with modern computers and digital electronics are:

- Decimal
- Binary
- Octal
- Hexadecimal

Decimal Number System:

- Decimalis the way most human beings represent numbers.
- The decimal system has a base or radix of 10.
- Decimal numbers uses digits from 0…9.

Decimal counting is as:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, etc.

For e.g.:

$1415_{10} = 1*10^3 + 4*10^2 + 1*10^1 + 5*10^0$

Binary Number System:

- Binary is the natural way most digital circuits represent and manipulate numbers.
- The binary system has a base or radix of 2
- Binary numbers uses only 0 and 1 digits.
- The binary digit is also referred to as Bit.
- A string of 4 bits is called a nibble and a string of 8 bits is called a byte.

Binary counting is as:

0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, etc.

For e.g.:

The decimal number 125 in binary system is represented as 1111101. Therefore,
$(1111101)_2 = 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$

Octal Number System:

- Octal was previously a popular choice for representing digital circuit numbers in a form that is more compact than binary.
- The binary system has a base or radix of 2
- Octal number system uses digits from 0…7.

Octal counting is as:

0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, etc.

For e.g.:

The decimal number 21 in octal system is represented as 25. Therefore,
$(25)_8 = 2*8^1 + 5*8^0$

Hexadecimal Number System:

- Hexadecimal is currently the most popular choice for representing digital circuit numbers in a form that is more compact than binary.
- The Hexadecimal system has a base or radix of 16
- 16 possible different numbers used represented in the form of digits from 0…9 and letters A to F.

Hexadecimal counting is as:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, etc.

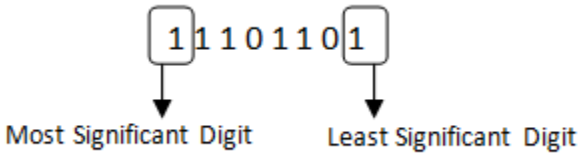
For e.g.:

The decimal number 32 in hexadecimal system is represented as 20. Therefore,
$20_{16} = 20H = 2\times16^1+0\times16^0$

E.g. 2: The decimal number 27 in hexadecimal system is represented as 1B. Therefore,
$1B_{16} = 16H = 1*16^1 + 11*16^0$

Significant Digits:

In Binary:



The Most Significant Digit/Bit is the bit position in a binary number having greatest value.
The Least Significant Digit/Bit is the bit position in a binary number having least value.

Some Number System Conversion used in embedded programming:

Binary Number to Decimal Number:

To convert a binary number to a decimal number, follow the steps:

- Multiply each bit by 2^n, where *n* is the "position" of the bit
- The position of the bit, starting from 0 on the right.
- Then add the results as shown below:
  101011_2 :

$1 \times 2^0 = 1$
$1 \times 2^1 = 2$
$0 \times 2^2 = 0$
$1 \times 2^3 = 8$
$0 \times 2^4 = 0$
$1 \times 2^5 = 32$

Decimal value:- 43_{10}

Decimal Number to Binary Number:

To convert a decimal number to Binary number, follow the steps:

- Divide the number by two, and keep track of the remainder
- First remainder is bit 0 (least-significant bit)
- Second remainder is bit 1, and so on as shown below:
  125_{10}

2	125	
2	62	1
2	31	0
2	15	1
2	7	1
2	3	1
2	1	1
	0	1

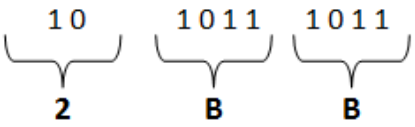
Binary Number:- $(1\ 1\ 1\ 1\ 1\ 0\ 1)_2$

Binary Number to hexadecimal Number:

To convert a binary number to hexadecimal, follow the steps:

- Group bits in set of four, starting on right.
- Convert each to hexadecimal digits as shown below:

1010111011_2



Hexadecimal Number: - $(2BB)_{16}$

Hexadecimal Number to Binary Number:
Converting from hexadecimal to binary is as easy as converting from binary to hexadecimal. Take each hexadecimal digit to obtain the equivalent group of four binary digits, as shown below:
$(A2DE)_{16}$

A	2	D	E
1010	0010	1101	1110

Binary Number:- $(1010001011011110)_2$

We hope this tutorial was helpful for you in learning basic C programming concepts.

Thank You!