

UNIT II

- First C program - Hello world,
- How to open a command prompt on Windows or Linux
- How to read and print on screen - printf(),scanf(),getchar(), putchar()
- Variables and Data types - Variables, Identifiers, data types and sizes, type conversions, difference between declaration and definition of a variable, Constants
- Life of a C program (Preprocessing, Compilation, Assembly, Linking, Loading, Execution), Compiling from the command line, Macros,
- Operators – equality and assignment, Compound assignment operators, Increment and decrement operators, Performance comparison between pre and post increment/decrement operators, bitwise operators (AND, OR, NOT and XOR), Logical Operators, comma operator, precedence and associativity, Logical operators (AND, OR)

Why C language is so important?

- Worth to know about C language
 - Oracle is written in C
 - Core libraries of android are written in C
 - MySQL is written in C
 - Almost every device driver is written in C
 - Major part of web browser is written in C
 - Unix operating system is developed in C
 - C is the world's most popular programming language

Why C language is so important?

- For students
 - C is important to build programming skills
 - C covers basic features of all programming language
 - Campus recruitment process
 - C is the most popular language for hardware dependent programming

History of C language

- Developer of **BCPL**
- **B**asic **C**ombined **P**rogramming Language
- 1966



Martin Richards

History of C language

- Developer of B language
- 1969
- Also developer of UNIX operating system



Ken Thompson

History of C language

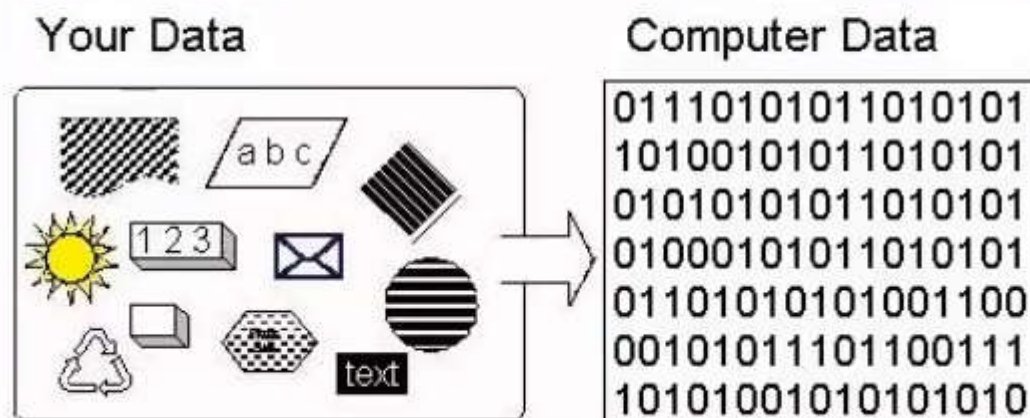
- Developer of **C language**.
- In **1972**
- At AT & T's Bell LABs, USA
- Co-developer of UNIX operating system.



Dennis Ritchie

What is 0 and 1?

- There is nothing like 0 and 1 in computer. There is no physical significance of 0 and 1
- Any information can be encoded as a sequence of 0 and 1.



How 0 and 1 get stored in memory?



What is Hardware?

- **Hardware** is a comprehensive term for all of the physical parts of a computer, as distinguished from the data it contains or operates on, and the software that provides instructions for the hardware to accomplish tasks.
- Hardware is anything which is tangible

What is a File?

- File is a data bundle.

File Name

File Extension

Track01.mp3

DSC1.jpg

list.txt

Sum.exe

010101
000110
101001
001110

010101
000110
101001
001110

010101
000110
101001
001110

010101
000110
101001
001110

What is software?

- Application software
- System Software

This file is a software 

Sum.exe

010101
000110
101001
001110

Program and Process

- Set of instructions is called **program**
- Active state of a program is called **process**

Operating System

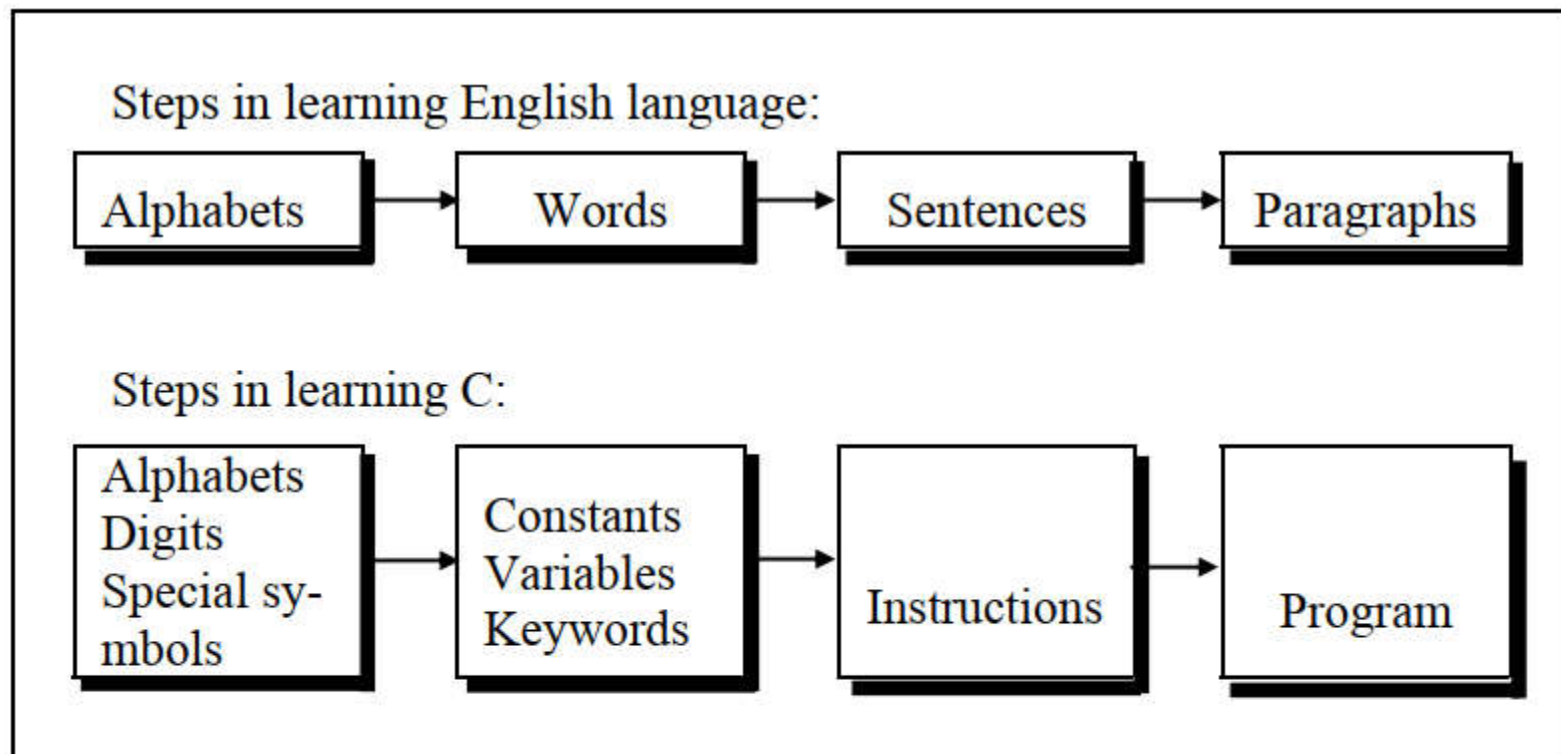
- It is a system software
- Examples are DOS, windows, Solaris, Macintosh, Linux, ubuntu etc
- It provides interface between user and machine
- Acts as a manager of the computer system
- It does process management, memory management, file management.

Objective

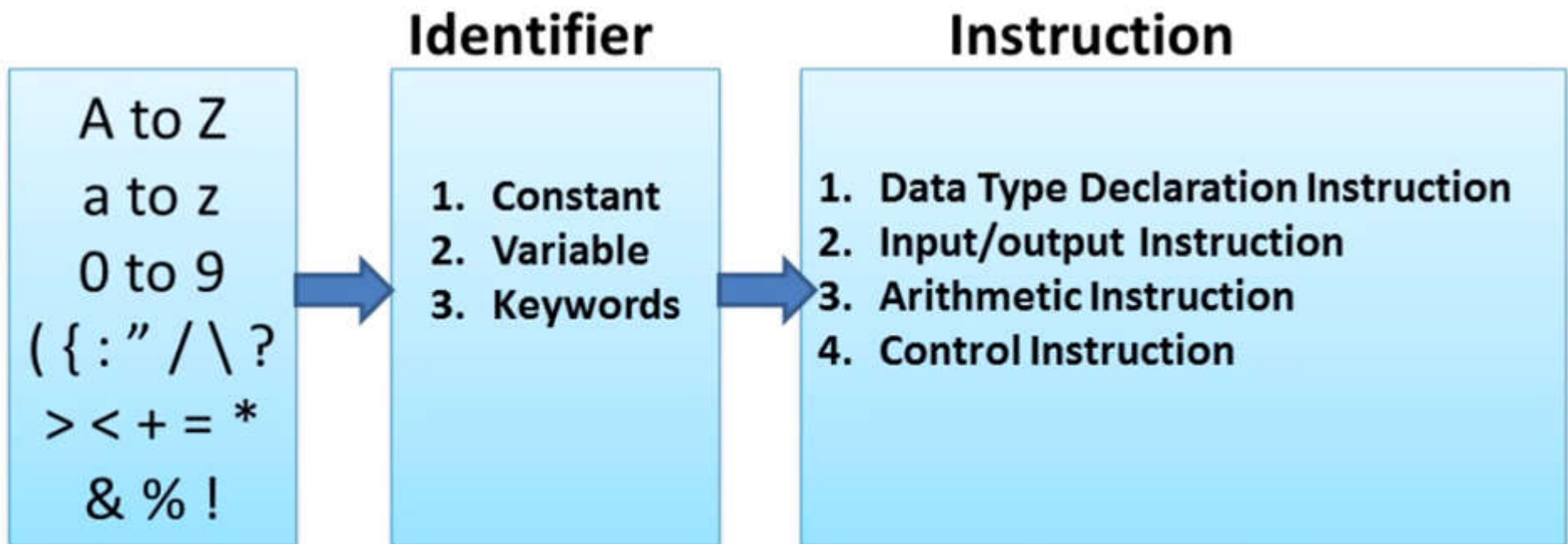
- Understanding Program's execution
- How to create a software using C language

- **What is C**

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie.



Begin Learning C...



- **The C Character Set**

A character denotes any alphabet, digit or special symbol used to represent information.

- Following are the valid alphabets, numbers and special symbols allowed in C.

Alphabets	A, B,, Y, Z a, b,, y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ' ! @ # % ^ & * () _ - + = \ { } [] : ; " ' < > , . ? /

-
- Constant
 - Variable
 - Keywords

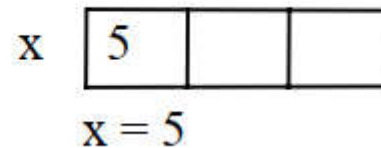
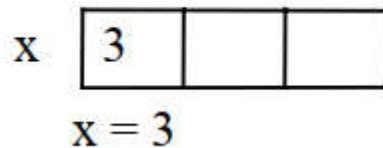
Constant

- Any information is constant
- Data = Information = constant

Constants and Variables

A constant is an entity that doesn't change whereas a variable is an entity that may change.

- *The results of calculations are stored in computers memory. Like human memory the computer memory also consists of millions of cells.*
- *These memory cells (also called memory locations) are given names. Since the value stored in each location may change the names given to these locations are called variable names.*
- Consider the following example.

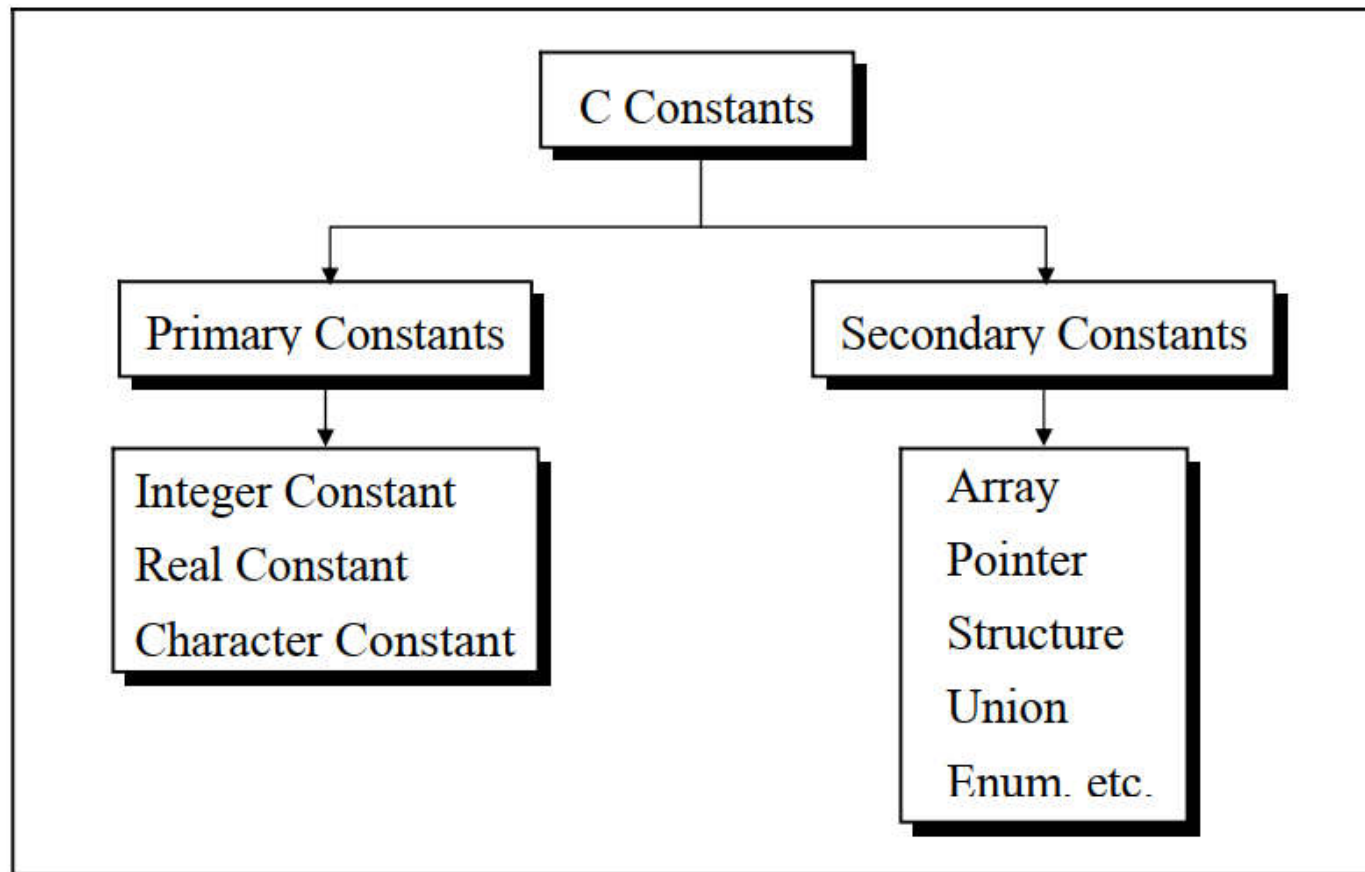


■ Types of C Constants

C constants can be divided into two major categories:

(a) Primary Constants

(b) Secondary Constants



■ Rules for Constructing Integer Constants

- (a) An integer constant must have at least one digit.
- (b) It must not have a decimal point.
- (c) It can be either positive or negative.
- (d) If no sign precedes an integer constant it is assumed to be positive.
- (e) No commas or blanks are allowed within an integer constant.

Ex. of Integer: 426
 +782
 -8000
 -7605

- (f) The range for integer constants is **-32768 to 32767**.

***Note:** The range of an Integer constant depends upon the compiler.*

For a 16-bit compiler range is -32768 to 32767.

For a 32-bit compiler the range would be even greater.

The range for long integer constant is **-2147483648 to 2147483647**

- **Rules for Constructing Real Constants (Floating Point constants).**

The real constants could be written in two forms

Fractional form and Exponential form.

- (a) A real constant must have at least one digit.
- (b) It must have a decimal point.
- (c) It could be either positive or negative.
- (d) Default sign is positive.

Ex. Of Floating Point constants :

	+325.34
	426.0
	-32.76
	-48.5792

- Range of real constants expressed in exponential form is **-3.4e38 to 3.4e38**

- Ex. Real constants expressed in exponential form :

+3.2e-5
4.1e8
-0.2e+3
-3.2e-5

Prepared by: Amit Kr Mishra, Department
of CSE, GEHU, DDUN

▪ Rules for Constructing Character Constants

- A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.
- The maximum length of a character constant can be 1 character.

Ex. of character constants : 'a'
 'I'
 '5'
 '=='

Data Type	Minimum Storage Allocated	Used for Variables that can contain
Int	2 bytes (16 bits)	integer constants in the range -32768 to 32767
short	2 bytes (16 bits)	integer constants in the range -32768 to 32767
long	4 bytes (32 bits)	integer constants in the range -2147483648 to 2147483647
float	4 bytes (32 bits)	real constants with minimum 6 decimal digits precision
double	8 bytes (64 bits)	real constants with minimum 10 decimal digits precision
char	1 byte (8 bits)	character constants
enum	2 bytes (16 bits)	Values in the range -32768 to 32767
void	No storage allocated	No value assigned

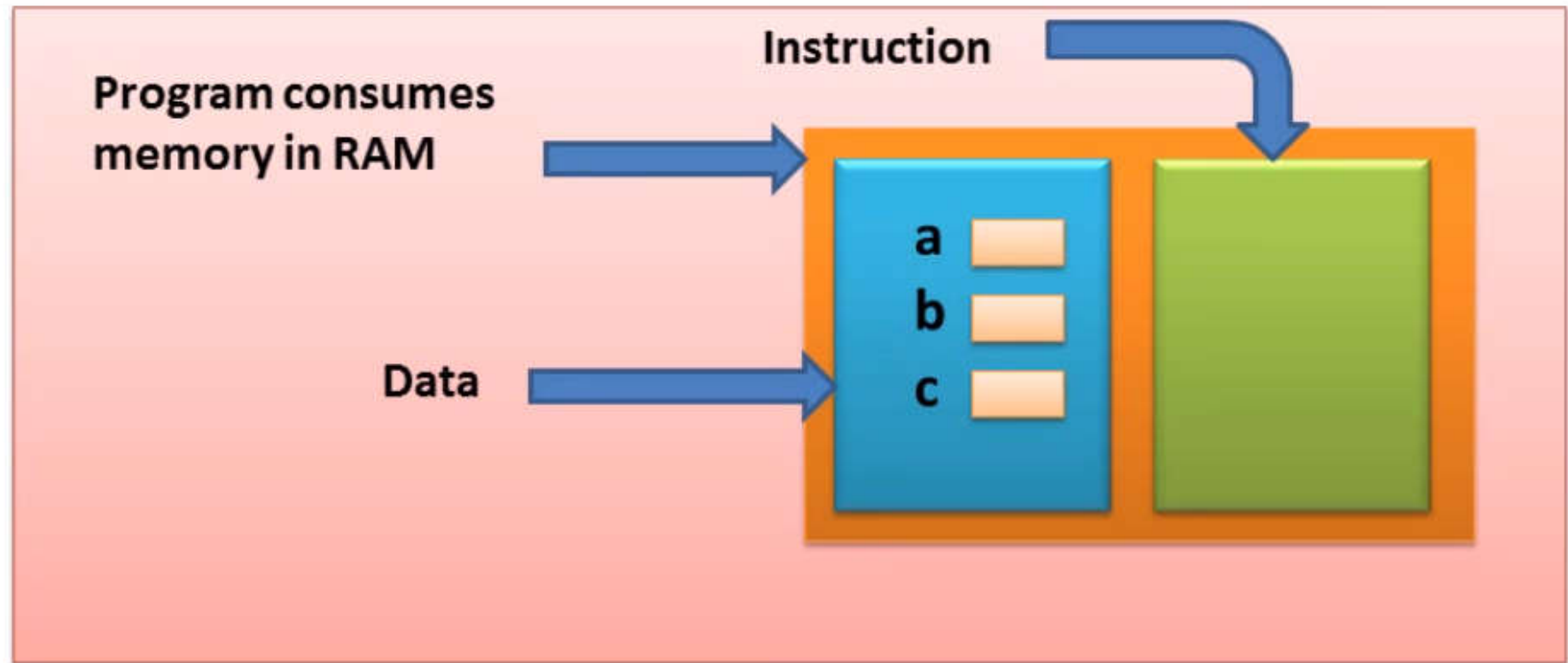
Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Floating-Point Types

The following table provides the details of standard floating-point types with storage sizes and value ranges and their precision:

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Variables



- Variables are the names of memory locations where we store data.

Rules

- Variable name is any combination of alphabet, digit and underscore.
- A valid variable name cannot start with digit

- **Types of C Variables**

Variable names are names given to locations in memory. These locations can contain integer, real or character constants.

- For example, an integer variable can hold only an integer constant, a real variable can hold only a real constant and a character variable can hold only a character constant.

- **Rules for Constructing Variable Names**

(a) A variable name is any combination of alphabets, digits or underscores. *Do not create unnecessarily long variable names as it adds to your typing effort.*

(b) The first character in the variable name must be an alphabet or underscore.

(c) No commas or blanks are allowed within a variable name.

(d) No special symbol other than an underscore (as in **student_id**)

can be used in a variable name. Ex.: a, b, sum, si_int , m_hra,

■ C Keywords

“Keywords are the words whose meaning has already been explained to the C compiler”.

- The keywords **cannot** be used as variable names.
- The keywords are also called ‘Reserved words’.
- There are only 32 keywords available in C.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

■ **The First C program -**

Before we begin with our first C program do remember the following rules that are applicable to all C programs:

- (a) Each instruction in a C program is written as a separate statement. Therefore a complete C program would comprise of a series of statements.
- (b) The statements in a program must appear in the same order in which we wish them to be executed.
- (c) Blank spaces may be inserted between two words to improve the readability of the statement. However, no blank spaces are allowed within a variable, constant or keyword.
- (d) All statements are entered in small case letters.
- (e) Every C statement must end with a ;. Thus ; acts as a statement terminator.

- Now a few useful tips about the program...
 - there are two types of comments in our 'C' programming:

// **single line comment**

Or

/* */ **multiple line comment**

Any number of comments can be written at any place in the program.

Though comments are not necessary, but it is a good practice to begin a program with a comment indicating the objective of the program, its author name and other details etc.

For example:

/* Write a 'C' program to.....*/

NAME:

SECTION:

ROLL NO:

BRANCH:

***/**

- ***First C program – to print “Hello world”***
 - *Let us now write down our first C program. It would simply print a message print “Hello world”.*
-

/* Write a ‘C’ program to print a message “Hello world”

NAME:

SECTION:

ROLL NO:

BRANCH:

*/

```
#include<stdio.h>           // header file
void main( )                 // main function, without return
{                             // start : body of the program
printf(“hello world”);       // print function which will print the message
}                             // end : body of the program
```

OUTPUT:

hello world

- Let us take a look at the various parts of the above program:
 1. The first line of the program *#include <stdio.h>* is a preprocessor command, which tells a C compiler to include *stdio.h* file before going to actual compilation.
 2. The next line *void main()* is the main function where the program execution begins.
 3. The next line *printf(...)* is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
 4. */*...*/* will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.

- Any variable used in the program must be declared before using it. For example,
int a, b, sum;
float r, si;
- The general form of **printf()** function is,
printf (“< format/message >”) ;
printf ("<format string>", <list of variables>) ;

<format string> can contain,

%f for printing real values

%d for printing integer values

%c for printing character values

In addition to format specifiers like **%f**, **%d** and **%c** the format string may also contain any other characters. These characters are printed as they are, when the **printf()** is executed.

- Following are some examples of usage of **printf()** function:

```
printf ( "%d", sum) ;
```

```
printf ( "%f", si ) ;
```

```
printf ( "%d %d %f %f", p, n, r, si ) ;
```

```
printf ( "Simple interest = Rs %f ", si ) ;
```

```
printf ( "amount = %d \n Rate = %f ", p, r ) ;
```

- **Receiving Input**

To make the program general, the program itself should ask the user to supply the values through the keyboard during execution.

- This can be achieved using a function called **scanf()**.
- This function is a counter-part of the **printf()** function.

printf() outputs the values to the screen whereas **scanf()** receives them from the keyboard.

This is illustrated in the program shown below.

```

/* Write a 'C' program to Calculation of simple interest.*/
void main( )
{
    int p, n ;
    float r, si ;
    printf ( "Enter values of p, n, r" ) ;
    scanf ( "%d %d %f", &p, &n, &r ) ;
    si = p * n * r / 100 ;
    printf ( "%f" , si ) ;
}

```

- The first **printf()** outputs the message.
- ampersand (**&**) before the variables in the **scanf()** is must.
- **&** is an 'Address of' operator. It gives the location number used by the variable in memory.
- For Ex: When we say **&p**, we are telling **scanf()** at which memory location should it store the value supplied of p by the user from the keyboard.

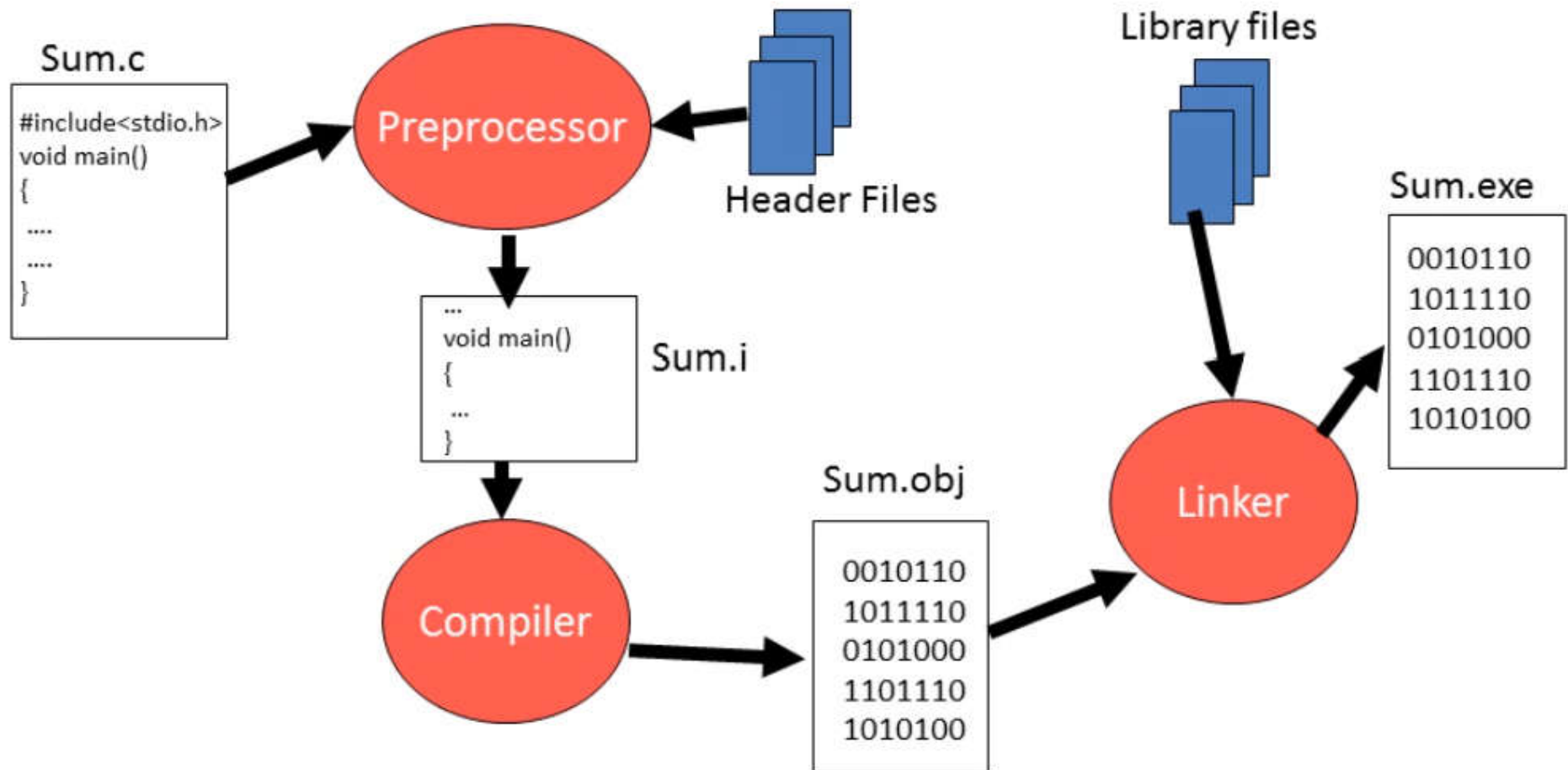
- **Compilation and Execution**

Once you have written the program you need to type it and instruct the machine to execute it.

- To type your C program you need another program called Editor.
- Once the program has been typed it needs to be converted to machine language (0s and 1s) before the machine can execute it.
- To carry out this conversion we need another program called Compiler.

Compiler vendors provide an Integrated Development Environment (IDE) which consists of an Editor as well as the Compiler.

Software development in C



-
- What is Instruction?
 - Data Type Declaration Instruction

Instruction

- Program statements are called Instructions.
- Instructions are commands
- Types of Instructions
 - Data Type Declaration Instruction
 - Input Output Instruction
 - Arithmetic Instruction
 - Control Instruction

■ C Instructions

There are basically three types of instructions in C:

- 1) **Type Declaration Instruction:** This instruction is used to declare the type of variables being used in the program.

Ex.: int a, b, sum;
float p, r, t;
char gender;

While declaring the type of variable we can also initialize it

```
int i = 10, j = 25 ;  
float a = 1.5, b=3.75;
```

Declaration statements

int a,b;

a

b



float k;

k



char ch,m;

ch

m



double d1;

d1



2) Arithmetic Instruction: To perform arithmetic operations between constants and variables.

In this 'operands' that are operated upon by the 'arithmetic operators' and the result is assigned, using the assignment operator, to the variable on left-hand side.

A 'C' arithmetic statement could be of three types:

- **Integer mode arithmetic statement –**

Ex: int a=10, b=15, sum;
 sum=a+b;

- **Real mode arithmetic statement –**

Ex: float p=370.46, r=8.88, t=1.8, si;
 si=(p*r*t)/100;

- **Mixed mode arithmetic statement -**

Ex: int p=9000, t=11;
 float r=8.8, si;
 si=(p*r*t)/100;

- **3) Control Instruction:** To control the sequence of execution of various statements in a C program.
- There are four types of control instructions in C. They are:
 - (a) Sequence Control Instruction
 - (b) Selection or Decision Control Instruction
 - (c) Repetition or Loop Control Instruction
 - (d) Case Control Instruction

■ Hierarchy of Operations

An arithmetic statement are performed is called the hierarchy of operations.

Priority	Operators	Description
1 st	* / %	multiplication, division, modular division
2 nd	+ -	addition, subtraction
3 rd	=	assignment

- C can handle any complex expression:

Algebraic Expression	C Expression
$a \times b - c \times d$	<code>a * b - c * d</code>
$(m + n) (a + b)$	<code>(m + n) * (a + b)</code>
$3x^2 + 2x + 5$	<code>3 * x * x + 2 * x + 5</code>
$\frac{a + b + c}{d + e}$	<code>(a + b + c) / (d + e)</code>
$\left[\frac{2BY}{d+1} - \frac{x}{3(z+y)} \right]$	<code>2 * b * y / (d + 1) - x / 3 * (z + y)</code>

OPERATORS IN C

C language supports a lot of operators to be used in expressions. These operators can be categorized into the following major groups:

- Arithmetic operators
- Relational Operators
- Equality Operators
- Logical Operators
- Unary Operators
- Conditional Operators
- Bitwise Operators
- Assignment operators
- Comma Operator
- Sizeof Operator

■ ARITHMETIC OPERATORS

OPERATION	OPERATOR	SYNTAX	COMMENT
Multiply	*	<code>a * b</code>	<code>result = a * b</code>
Divide	/	<code>a / b</code>	<code>result = a / b</code>
Addition	+	<code>a + b</code>	<code>result = a + b</code>
Subtraction	-	<code>a - b</code>	<code>result = a - b</code>
Modulus	%	<code>a % b</code>	<code>result = a % b</code>

- RELATIONAL OPERATORS

- Also known as a comparison operator, it is an operator that compares two values. Expressions that contain relational operators are called *relational expressions*. Relational operators return true or false value, depending on whether the conditional relationship between the two operands holds or not.

OPERATOR	MEANING	EXAMPLE
<	LESS THAN	3 < 5 GIVES 1
>	GREATER THAN	7 > 9 GIVES 0
>=	LESS THAN OR EQUAL TO	100 >= 100 GIVES 1
<=	GREATER THAN EQUAL TO	50 >=100 GIVES 0

EQUALITY OPERATORS

- C language supports two kinds of equality operators to compare their operands for strict equality or inequality. They are equal to (==) and not equal to (!=) operator.
- The equality operators have lower precedence than the relational operators.

OPERATOR	MEANING
==	RETURNS 1 IF BOTH OPERANDS ARE EQUAL, 0 OTHERWISE
!=	RETURNS 1 IF OPERANDS DO NOT HAVE THE SAME VALUE, 0 OTHERWISE

LOGICAL OPERATORS

- C language supports three logical operators. They are- Logical AND (&&), Logical OR (||) and Logical NOT (!).
- As in case of arithmetic expressions, the logical expressions are evaluated from left to right.

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A B
0	0	0
0	1	1
1	0	1
1	1	1

A	! A
0	1
1	0

UNARY OPERATORS

- Unary operators act on single operands. C language supports three unary operators. They are unary minus, increment and decrement operators.
- When an operand is preceded by a minus sign, the unary operator negates its value.
- The increment operator is a unary operator that increases the value of its operand by 1. Similarly, the decrement operator decreases the value of its operand by 1. For example,

`int x = 10, y;`

`y = x++;`

is equivalent to writing

`y = x;`

`x = x + 1;` whereas, `y = ++x;`

is equivalent to writing

`x = x + 1;`

`y = x;`

Prepared by: Amit Kr Mishra, Department
of CSE, GEHU, DDUN

CONDITIONAL OPERATOR

- The conditional operator operator (?:) is just like an if .. else statement that can be written within expressions.
- The syntax of the conditional operator is **exp1 ? exp2 : exp3**

Here, exp1 is evaluated first. If it is true then exp2 is evaluated and becomes the result of the expression, otherwise exp3 is evaluated and becomes the result of the expression. For example,

large = (a > b) ? a : b

- Conditional operators make the program code more compact, more readable, and safer to use as it is easier both to check and guarantee that the arguments that are used for evaluation.
- Conditional operator is also known as ternary operator as it is neither a unary nor a binary operator; it takes *three* operands.

BITWISE OPERATORS

- Bitwise operators perform operations at bit level. These operators include: bitwise AND, bitwise OR, bitwise XOR and shift operators.
- The bitwise AND operator (&) is a small version of the boolean AND (&&) as it performs operation on bits instead of bytes, chars, integers, etc.
- The bitwise OR operator (|) is a small version of the boolean OR (||) as it performs operation on bits instead of bytes, chars, integers, etc.
- The bitwise NOT (~), or complement, is a unary operation that performs logical negation on each bit of the operand. By performing negation of each bit, it actually produces the ones' complement of the given binary value.
- The bitwise XOR operator (^) performs operation on individual bits of the operands. The result of XOR operation is shown in the table

ASSIGNMENT OPERATORS

- The assignment operator is responsible for assigning values to the variables. While the equal sign (=) is the fundamental assignment operator, C also supports other assignment operators that provide shorthand ways to represent common variable assignments. They are shown in the table.

COMMA OPERATOR

- The comma operator in C takes two operands. It works by evaluating the first and discarding its value, and then evaluates the second and returns the value as the result of the expression.
- Comma separated operands when chained together are evaluated in left-to-right sequence with the right-most value yielding the result of the expression.
- Among all the operators, the comma operator has the lowest precedence. For example,

int a=2, b=3, x=0;

x = (++a, b+=a);

Now, the value of x = 6.

Prepared by: Amit Kr Mishra, Department
of CSE, GEHU, DDUN

sizeof OPERATOR

- sizeof is a unary operator used to calculate the sizes of data types.
- It can be applied to all data types.
- The operator returns the size of the variable, data type or expression in bytes.
- 'sizeof' operator is used to determine the amount of memory space that the variable/expression/data type will take. For example,
- sizeof(char) returns 1, that is the size of a character data type. If we have,

```
int a = 10;
```

```
int result;
```

```
result = sizeof(a);
```

```
then result = 2,
```

- To get the exact size of a type or a variable on a particular platform, we can use the **sizeof** operator. The expressions *sizeof(type)* the storage size of the object or type in bytes.

For Example:

```
#include <stdio.h>
void main()
{
printf("Storage size for int : %d \n", sizeof(int));
printf("Storage size for float : %d \n", sizeof(float));
}
```

TYPE CONVERSION AND TYPE CASTING

- Type conversion and type casting of variables refers to changing a variable of one data type into another.
- While type conversion is done implicitly, casting has to be done explicitly by the programmer. We will discuss both of them here.
- Type conversion is done when the expression has variables of different data types. So to evaluate the expression, the data type is promoted from lower to higher level where the hierarchy of data types can be given as: double, float, long, int, short and char.
- For example, type conversion is automatically done when we assign an integer value to a floating point variable. For ex,

float x;

int y = 3;

x = y;

Now, x = 3.0,

Prepared by: Amit Kr Mishra, Department
of CSE, GEHU, DDUN

Type casting is also known as forced conversion. It is done when the value of a higher data type has to be converted into the value of a lower data type. For example, we need to explicitly type cast an integer variable into a floating point variable.

```
float salary = 10000.00;
```

```
int sal;
```

```
sal = (int) salary;
```

Typecasting can be done by placing the destination data type in parentheses followed by the variable name that has to be converted.

▪ Basic Library Functions for I/O Operations

I/O Library Functions	
getchar()	Inputs a single character from console and echoes it, but requires <i>Enter</i> key to be typed after the character.
putchar() or putch()	Outputs a single character on console (screen).
scanf()	Enables input of formatted data from console (keyboard). Formatted input data means we can specify the data type expected as input. Format specifiers for different data types are given in Figure 21.6.
printf()	Enables obtaining an output in a form specified by programmer (formatted output). Format specifiers are given in Figure 21.6. Newline character “\n” is used in <i>printf()</i> to get the output split over separate lines.
gets()	Enables input of a string from keyboard. Spaces are accepted as part of the input string, and the input string is terminated when <i>Enter</i> key is hit. Note that although <i>scanf()</i> enables input of a string of characters, it does not accept multi-word strings (spaces in-between).
puts()	Enables output of a multi-word string

- **Basic Format Specifiers for *scanf()* and *printf()***

Format Specifiers	Data Types
%d	integer (short signed)
%u	integer (short unsigned)
%ld	integer (long signed)
%lu	integer (long unsigned)
%f	real (float)
%lf	real (double)
%c	character
%s	string

```

/* A portion of C program to illustrate formatted input and output */
#include <stdio.h>
void main()
int maths, science, english, total;
    float percent;

    printf ( "Maths marks = " );    /* Displays "Maths marks = " */
    scanf ( "%d", &maths);          /* Accepts entered value and stores in variable "maths" */
    printf ( "\n Science marks = " ); /* Displays "Science marks = " on next line because of \n */
    scanf ( "%d", &science);         /* Accepts entered value and stores in variable "science" */
    printf ( "\n English marks = " ); /* Displays "English marks = " on next line because of \n */
    scanf ( "%d", &english);         /* Accepts entered value and stores in variable "english" */

    total = maths + science + english;
    percent = total/3;               /* Calculates percentage and stores in variable "percent" */

    printf ( "\n Percentage marks obtained = %f", percent);
                                   /* Displays "Percentage marks obtained in next line because of \n */

```

C library function - getchar()

The C library function **getchar()** gets a character (an unsigned char) from stdin. This is equivalent to **getc** with stdin as its argument.

- Following is the declaration for **getchar()** function.

int getchar(void)

This function returns the character read as an unsigned char cast to an int or EOF on end of file or error.


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

```
int main () {
```

```
    char c;
```

```
    printf("Enter character: ");
```

```
    c = getchar();
```

```
    printf("Character entered: ");
```

```
    putchar(c);
```

```
    return(0);
```

```
}
```

C library function - putchar()

- The C library function **putchar()** writes a character (an unsigned char) specified by the argument char to stdo

Following is the declaration for putchar() function.

```
int putchar(int char)
```

Parameters: char – This is the character to be written. This is passed as its int promotion.

Return Value

- This function returns the character written as an unsigned char cast to an int or EOF on error.

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

```
int main () {
```

```
    char ch;
```

```
    for(ch = 'A' ; ch <= 'Z' ; ch++) {
```

```
        putchar(ch);
```

```
    }
```

```
    return(0);
```

```
}
```

- **Preprocessor Directives**
- *Preprocessor* is a program that prepares a program for the C compiler
- Three common preprocessor directives in C are:
 - **#include** – Used to look for a file and place its contents at the location where this preprocessor directives is used
 - **#define** – Used for macro expansion
 - For EX:
`#include <stdio.h>`
`#define PI 3.1415`
`#define AND &&`

- **Syntax:** The syntax for creating a **constant** using #define in the C language is:
- #define CNAME value OR
- #define CNAME (expression)

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

```
#define AGE 27
```

```
{
    printf("%s is over %d years old.\n", NAME, AGE);
    return 0;
}
```

- NOTE:
- The three primary constants and variable types in C are integer, float and character.
- A variable name can be of maximum 31 characters.
- Do not use a keyword as a variable name.
- An expression may contain any sequence of constants, variables and operators.
- Operators having equal precedence are evaluated using associativity.
- Input/output in C can be achieved using **scanf()** and **printf()** functions

//Write a program for addition of two numbers?

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

```
void main()
```

```
{
```

```
int a, b, sum;                // Variable declaration
```

```
    printf("Enter two numbers to add\n");
```

```
    scanf("%d%d",&a,&b);        // Take two numbers as
```

```
input from the user
```

```
    sum = a+b;
```

```
    printf("Sum of entered numbers = %d\n",sum);
```

```
}
```

//Write a program for swapping of two integer numbers?

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

```
void main()
```

```
{
```

```
int a, b, temp;                // Variable declaration
```

```
    printf("Enter the value of a and b \n");
```

```
    scanf("%d %d", &a, &b);      // Take two numbers as input
```

from the user

```
    printf("Before swapping the value of a=%d \nand the value of  
b=%d\n",a,b);
```

```
        temp=a;
```

```
        a=b;
```

```
        b=temp;
```

```
    printf("After swapping the value of a=%d \nand the value of  
b=%d\n",a,b);
```

```
}
```



```
/*Write a program for swapping of two integer numbers without using third
variable?
*/
#include<stdio.h>

void main()
{
int a, b;                // Variable declaration
printf("Enter the value of a and b \n");
scanf("%d %d", &a, &b); // Take two numbers as input from the user
printf("Before swapping the value of a=%d \nand the value of b=%d\n",a,b);
    a=a+b;
    b=a-b;
    a=a-b;
printf("After swapping the value of a=%d \nand the value of b=%d\n",a,b);
}
```

```
/*Write a program to calculate the area of a triangle using Hero's formula.*/
```

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

```
#include <math.h>
```

```
int main()
```

```
{
```

```
float a, b, c, area, S;
```

```
printf("\n Enter the lengths of the three sides of the triangle : ");
```

```
scanf("%f %f %f", &a, &b, &c);
```

```
S = ( a + b + c)/2; // sqrt is a mathematical function defined in math.h header file
```

```
area = sqrt(S*(S-a)*(S-b)*(S-c));
```

```
printf("\n Area = %f", area);
```

```
return 0;
```

```
}
```

/*Write a program to calculate the distance between two points.*/

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

```
#include <math.h>
```

```
int main()
```

```
{
```

```
int x1, x2, y1, y2;
```

```
float distance;
```

```
printf("\n Enter the x1 and y1 coordinates of the first point : ");
```

```
scanf("%d %d", &x1, &y1);
```

```
printf("\n Enter the x2 and y2 coordinates of the second point :");
```

```
scanf("%d %d", &x2, &y2); // sqrt and pow are mathematical functions  
    defined in math.h header file
```

```
distance = sqrt(pow((x2-x1), 2)+pow((y2-y1), 2));
```

```
printf("\n Distance = %f", distance);
```

```
return 0;
```

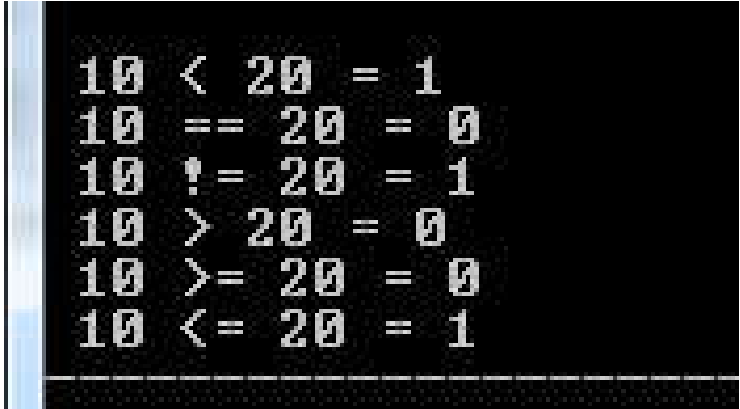
```
}
```

Distance between two points $P(x_1, y_1)$ and $Q(x_2, y_2)$ is given by:

$$d(P, Q) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad \{\text{Distance formula}\}$$

/* Write a program to show the use of relational operators.*/

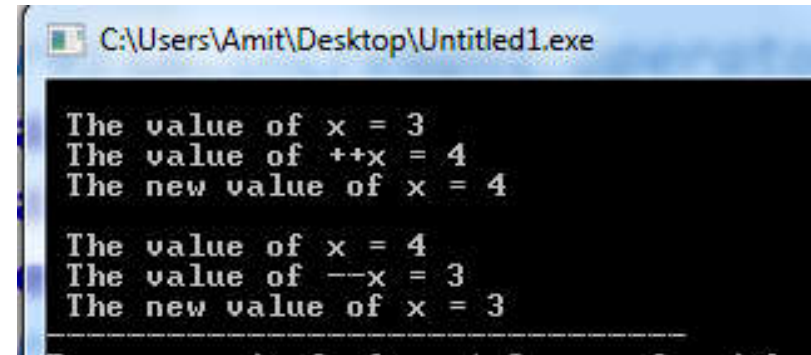
```
#include <stdio.h>
main()
{
int x=10, y=20;
printf("\n %d < %d = %d", x, y, x<y);
printf("\n %d == %d = %d", x, y, x==y);
printf("\n %d != %d = %d", x, y, x!=y);
printf("\n %d > %d = %d", x, y, x>y);
printf("\n %d >= %d = %d", x, y, x>=y);
printf("\n %d <= %d = %d", x, y, x<=y);
return 0;
}
```



```
10 < 20 = 1
10 == 20 = 0
10 != 20 = 1
10 > 20 = 0
10 >= 20 = 0
10 <= 20 = 1
```

`/*Write a program to illustrate the use of unary prefix increment and decrement operators.*/`

```
#include <stdio.h>
main()
{
int x = 3;
// Using unary prefix increment operator
printf("\n The value of x = %d", x);
printf("\n The value of ++x = %d", ++x);
printf("\n The new value of x = %d", x);
// Using unary prefix decrement operator
printf("\n\n The value of x = %d", x);
printf("\n The value of --x = %d", --x);
printf("\n The new value of x = %d", x);
return 0;
}
```

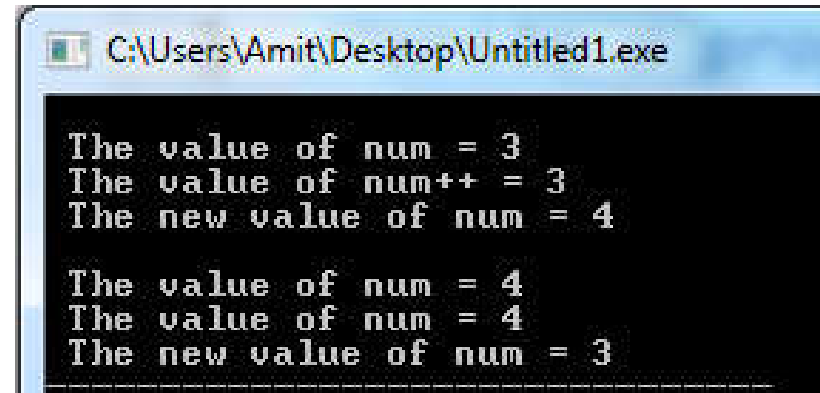


```
The value of x = 3
The value of ++x = 4
The new value of x = 4

The value of x = 4
The value of --x = 3
The new value of x = 3
```

`/*Write a program to illustrate the use of unary postfix increment and decrement operators.*/`

```
#include <stdio.h>
main()
{
int num = 3;
// Using unary postfix increment operator
printf("\n The value of num = %d", num);
printf("\n The value of num++ = %d", num++);
printf("\n The new value of num = %d", num);
// Using unary postfix decrement operator
printf("\n\n The value of num = %d", num);
printf("\n The value of num = %d", num--);
printf("\n The new value of num = %d", num);
return 0;
}
```

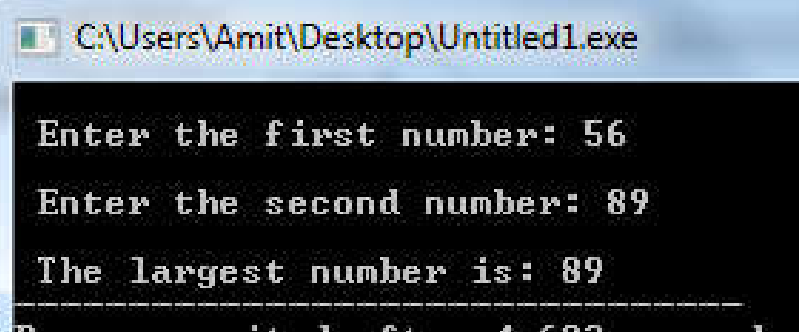


```
The value of num = 3
The value of num++ = 3
The new value of num = 4

The value of num = 4
The value of num = 4
The new value of num = 3
```

/*Write a program two find the largest of two numbers using ternary operator.*/

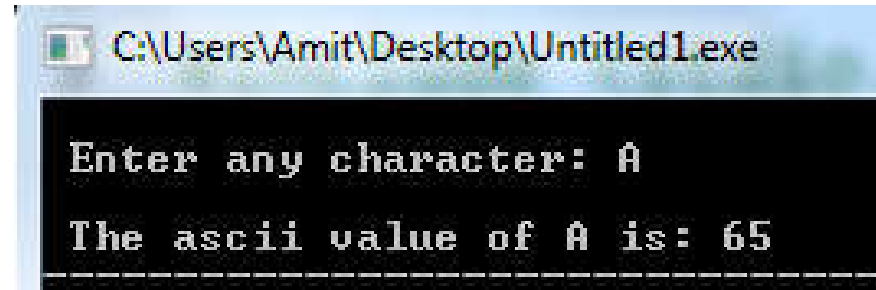
```
#include <stdio.h>
int main()
{
    int num1, num2, large;
    printf("\n Enter the first number: ");
    scanf("%d", &num1);
    printf("\n Enter the second number: ");
    scanf("%d", &num2);
    large = num1>num2?num1:num2;
    printf("\n The largest number is: %d", large);
    return 0;
}
```



```
C:\Users\Amit\Desktop\Untitled1.exe
Enter the first number: 56
Enter the second number: 89
The largest number is: 89
```

/*Write a program to print the ASCII value of a character.*/

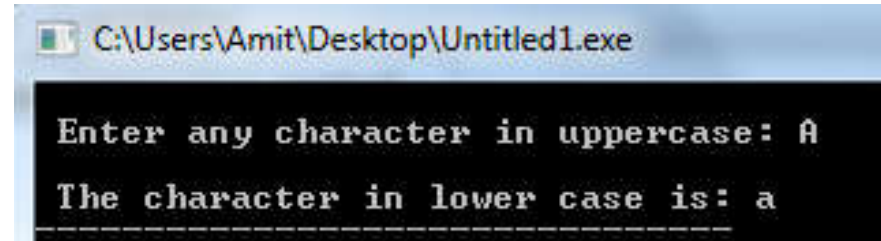
```
#include <stdio.h>
#include <conio.h>
int main()
{
    char ch;
    printf("\n Enter any character: ");
    scanf("%c", &ch);
    printf("\n The ascii value of %c is: %d",ch,ch);
    return 0;
}
```



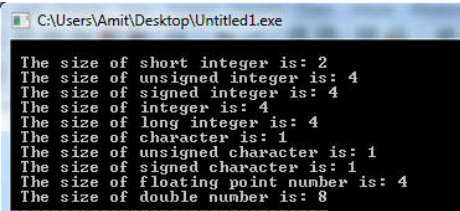
```
C:\Users\Amit\Desktop\Untitled1.exe
Enter any character: A
The ascii value of A is: 65
```


`/*Write a program to read a character in upper case and then print it in lower case.*/`

```
#include <stdio.h>
int main()
{
char ch;
printf("\n Enter any character in uppercase: ");
scanf("%c", &ch);
printf("\n The character in lower case is: %c", ch+32);
return 0;
}
```



```
/*Write a program that displays the size of every data type.*/  
#include <stdio.h>  
int main() {  
printf("\n The size of short integer is: %d", sizeof(short int));  
printf("\n The size of unsigned integer is: %d", sizeof(unsigned int));  
printf("\n The size of signed integer is: %d", sizeof(signed int));  
printf("\n The size of integer is: %d", sizeof(int));  
printf("\n The size of long integer is: %d", sizeof(long int));  
printf("\n The size of character is: %d", sizeof(char));  
printf("\n The size of unsigned character is: %d", sizeof(unsigned char));  
printf("\n The size of signed character is: %d", sizeof(signed char));  
printf("\n The size of floating point number is: %d", sizeof(float));  
printf("\n The size of double number is: %d", sizeof(double));  
return 0;  
}
```

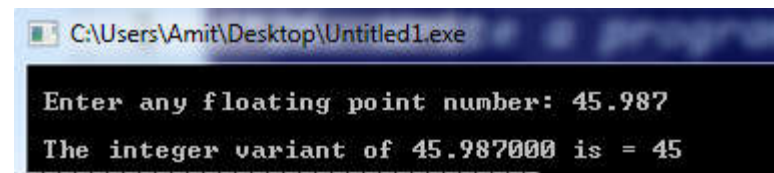


```
C:\Users\Amit\Desktop\Untitled1.exe  
The size of short integer is: 2  
The size of unsigned integer is: 4  
The size of signed integer is: 4  
The size of integer is: 4  
The size of long integer is: 4  
The size of character is: 1  
The size of unsigned character is: 1  
The size of signed character is: 1  
The size of floating point number is: 4  
The size of double number is: 8
```

```
/*Write a program to convert a floating point number into the  
corresponding integer.*/
```

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

```
int main()  
{  
float fnum;  
int inum;  
printf("\n Enter any floating point number: ");  
scanf("%f", &fnum);  
inum = (int)fnum;  
printf("\n The integer variant of %f is = %d", fnum, inum);  
return 0;  
}
```



```
C:\Users\Amit\Desktop\Untitled1.exe  
Enter any floating point number: 45.987  
The integer variant of 45.987000 is = 45
```

`/*Write a program to convert an integer into the corresponding floating point number.*/`

`#include <stdio.h>`

`int main()`

`{`

`float f_num;`

`int i_num;`

`printf("\n Enter any integer: ");`

`scanf("%d", &i_num);`

`f_num = (float)i_num;`

`printf("\n The floating point variant of %d is = %f", i_num, f_num);`

`return 0;`

`}`



```
C:\Users\Amit\Desktop\Untitled1.exe
Enter any integer: 399
The floating point variant of 399 is = 399.000000
```