MODULE I

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.

The C Character Set

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

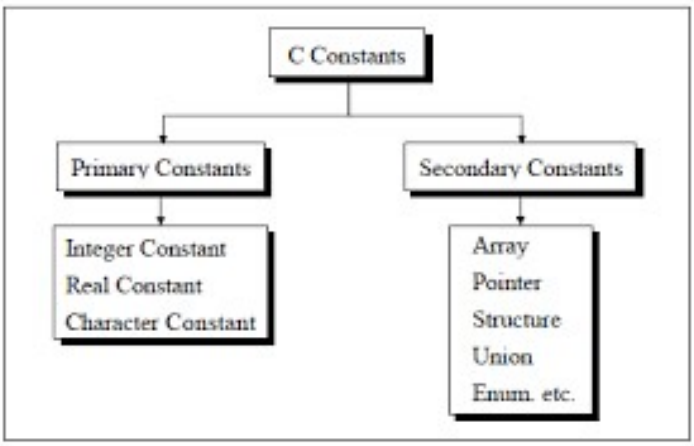
Alphabets	A, B, ..., Y, Z ,a, b, ..., y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /

Constants, Variables and Keywords

The alphabets, numbers and special symbols when properly combined form constants, variables and keywords.The calculated values are stored in the memory cells of a computer. To make the retrieval and usage of these values easy 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. Eg: 3 is stored in a memory location and a name x is given to it.Since the location whose name is x can hold different values at different times x is known as a variable. As against this, 3 or 5 do not change, hence are known as constants.

Types of C Constants

C constants can be divided into two major categories:
(a)Primary Constants
(b)Secondary Constants



Types of C Variables

An entity that may vary during program execution is called a variable. Variable names are names given to locations in memory. These locations can contain integer, real or character constants.

Rules for Constructing Variable Names

(a)A variable name is any combination of 1 to 31 alphabets, digits or underscores. Some compilers allow variable names whose length could be up to 247 characters.

(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 gross_sal) can be used in a variable name.

C Keywords

Keywords are the words whose meaning has already been explained to the C compiler (or in a broad sense to the computer). The keywords cannot be used as variable names because if we do so we are trying to assign a new meaning to the keyword, which is not allowed by the computer.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

C Instrcutions

There are basically three types of instructions in C:
(a) Type Declaration Instruction
(b) Arithmetic Instruction
(c) Control Instruction

The purpose of each of these instructions is given below:

(a) Type declaration instruction − To declare the type of variables used in a C program.
(b) Arithmetic instruction − To perform arithmetic operations between constants and variables.
(c) Control instruction − To control the sequence of execution of various statements in a C program.

Type Declaration Instruction
This instruction is used to declare the type of variables being used in the program. Any variable used in the program must be declared before using it in any statement. The type declaration statement is written at the beginning of main( ) function.
Ex.: int bas ;
float rs, grosssal ;
char name, code ;

Arithmetic Instruction
A C arithmetic instruction consists of a variable name on the left hand side of = and variable names & constants on the right hand side of =. The variables and constants appearing on the right hand
side of = are connected by arithmetic operators like +, -, *, and /.
Ex.: int ad ;
float kot, deta, alpha, beta, gamma ;
ad = 3200 ;
kot = 0.0056 ;
deta = alpha * beta / gamma + 3.2 * 2 / 5 ;

Here,

*, /, -, + are the arithmetic operators.
= is the assignment operator.
2, 5 and 3200 are integer constants.
3.2 and 0.0056 are real constants.
ad is an integer variable.
kot, deta, alpha, beta, gamma are real variables.

Integer and Float Conversions

(a) An arithmetic operation between an integer and integer always yields an integer result.
(b) An operation between a real and real always yields a real result.
(c) An operation between an integer and real always yields a real result. In this operation the integer is first promoted to a real and then the operation is performed. Hence the result is real.

Type Conversion in Assignments

It may so happen that the type of the expression and the type of the variable on the left-hand side of the assignment operator may not be same. In such a case the value of the expression is promoted or demoted depending on the type of the variable on left-hand side of =.

For example, consider the following assignment statements.
int i ;
float b ;
i = 3.5 ;
b = 30 ;
Here in the first assignment statement though the expression's value is a float (3.5) it cannot be stored in i since it is an int. In such a case the float is demoted to an int and then its value is stored. Hence what gets stored in i is 3. Exactly opposite happens in the next statement. Here, 30 is promoted to 30.000000 and then stored in b, since b being a float variable cannot hold anything except a float value.

Heirarchy Of Operations

While executing an arithmetic statement, which has two or more operators, we may have some problems as to how exactly does it get executed. For example, does the expression 2 * x - 3 * y correspond to (2x)-(3y) or to 2(x-3y)? This is done by heirarchy. The priority or precedence in which the operations in an arithmetic statement are performed is called the hierarchy of operations.

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

Associativity of Operators

When an expression contains two operators of equal priority the tie between them is settled using the associativity of the operators.Associativity can be of two types—Left to Right or Right to Left. Left to Right associativity means that the left operand must be unambiguous.

Operator	Left	Right	Remark
/	3	2 or 2 * 5	Left operand is unambiguous, Right is not
*	3 / 2 or 2	5	Right operand is unambiguous, Left is not

Control Instructions in C

The 'Control Instructions' enable us to specify the order in which the various instructions in a program are to be executed by the computer. In other words the control instructions determine the 'flow of control' in a 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

The Sequence control instruction ensures that the instructions are executed in the same order in which they appear in the program. Decision and Case control instructions allow the computer to take a decision as to which instruction is to be executed next. The Loop control instruction helps computer to execute a group of statements repeatedly.

A decision control instruction can be implemented in C using:
(a) The if statement
(b) The if-else statement
(c) The conditional operators

The if Statement

C uses the keyword if to implement the decision control instruction. The general form of if statement looks like this:

if (this condition is true)
execute this statement ;

The keyword if tells the compiler that what follows is a decision control instruction. The condition following the keyword if is always enclosed within a pair of parentheses. If the condition, whatever it is, is true, then the statement is executed. If the condition is not true then the statement is not executed; instead the program skips past it.

this expression	is true if
x == y	x is equal to y
x != y	x is not equal to y
x < y	x is less than y
x > y	x is greater than y
x <= y	x is less than or equal to y
x >= y	x is greater than or equal to y

Multiple Statements within if

It may so happen that in a program we want more than one statement to be executed if the expression following if is satisfied. If such multiple statements are to be executed then they must be placed within a pair of braces.

The if-else Statement

The if statement by itself will execute a single statement, or a group of statements, when the expression following if evaluates to true. It does nothing when the expression evaluates to false. We can execute one group of statements if the expression evaluates to true and another group of statements if the expression evaluates to false.This is the purpose of the else statement

Nested if-else

It is perfectly all right if we write an entire if-else construct within either the body of the if statement or the body of an else statement. This is called 'nesting'of ifs. This is shown in the following program.

```
/* A quick demo of nested if-else */
main( )
{
int i ;
printf ( "Enter either 1 or 2 " ) ;
scanf ( "%d", &i ) ;
if ( i == 1 )
printf ( "You would go to heaven !" ) ;
else
{
if ( i == 2 )
printf ( "Hell was created with you in mind" ) ;
else
printf ( "How about mother earth !" ) ;
}
}
```

The second if-else construct is nested in the first else statement. If the condition in the first if statement is false, then the condition in the second if statement is checked. If it is false as well, then the final else statement is executed.

Forms of if

The if statement can take any of the following forms:

```
(a) if ( condition )
    do this ;
(b) if ( condition )
    {
       do this ;
       and this ;
    }
(c) if ( condition )
        do this ;
    else
        do this ;
```

```
(d) if ( condition )
    {
       do this ;
       and this;
     }
    else
    {
       do this ;
       and this ;
    }
(e) if ( condition )
       do this ;
    else
    {
       if ( condition )
            do this ;
       else
       {
            do this ;
            and this ;
        }
    }
(f) if ( condition )
    {
       if ( condition )
            do this ;
       else
       {
            do this ;
            and this ;
       }
    }
    else
       do this ;
```

Use of Logical Operators

C allows usage of three logical operators, namely, &&, || and !. These are to be read as 'AND' 'OR' and 'NOT' respectively. There are several things to note about these logical operators. Most obviously, two of them are composed of double symbols: || and &&. Don't use the single symbol | and &. These single symbols also have a meaning. They are bitwise operators. The first two operators, && and ||, allow two or more conditions to be combined in an if statement.

The third logical operator is the NOT operator, written as !. This operator reverses the result of the expression it operates on. For eg, if the expression evaluates to a non-zero value, then applying ! operator to it results into a 0. Vice versa, if the expression evaluates to zero then on applying ! operator to it makes it 1, a non-zero value. The final result (after applying !) 0 or 1 is considered to be false or true respectively.

Hierarchy of Operators

Operators	Type
!	Logical NOT
* / %	Arithmetic and modulus
+ -	Arithmetic
< > <= >=	Relational
== !=	Relational
&&	Logical AND
||	Logical OR
=	Assignment

The Conditional Operators

The conditional operators ? and : are sometimes called ternary operators since they take three arguments. In fact, they form a kind of foreshortened if-then-else. Their general form is,
expression 1 ? expression 2 : expression 3
This means "if expression 1 is true (that is, if its value is non-zero), then the value returned will be expression 2, otherwise the value returned will be expression 3".
Let us understand this with the help of an example:

```
int x, y ;
scanf ( "%d", &x ) ;
y = ( x > 5 ? 3 : 4 ) ;
```

This statement will store 3 in y if x is greater than 5, otherwise it will store 4 in y.
The equivalent if statement will be,

```
if ( x > 5 )
y = 3 ;
else
y = 4 ;
```