UNIT - II

TYPES, OPERATORS AND EXPRESSIONS

2.1 INTRODUCTION

- The computer programming languages are designed to support certain kind of data like numbers, characters, strings, etc., to get useful output known as result or information. Data consists of digits, alphabets and symbols.
- A program should support these data types for getting the required output known as information.
- A program is a set of statements (instructions) which performs a specific task, executed by users in a sequential form.
- These statements or instructions are formed using certain words and symbols according to the rules known as syntax rules or grammar of the language. Every program must accurately follow the syntax rules supported by the language.
- C language fundamentals introduce the fundamental elements of C. In this chapter, C character set, the type of variables, types of tokens, type casting, delimiters, data types etc. are described.
- The C programming language provides a rich set of built-in operators to perform certain mathematical or logical manipulations.
- Operators are the special symbol use to perform special operation (like, arithmetic, logical etc.) on operands. Operands are the variables, constants or expressions on which operators perform operation.
- For example, 25 – 10, here 25 and 10 are the operands and – is the arithmetic operator which will simply subtract 10 from 25.
- Operators are used in C programs to manipulate data and variables.
- An expression is a sequence of operands and operators that reduces to a single value. For example, 25-10 is an expression whose single value is 15.
- As per the number of operands required by the operator, it is classified in three categories:

1. **Unary Operators:** These operators operates only on one operand.
2. **Binary Operators:** These operators operates on two operand.
3. **Ternary Operators:** These operators operates on three operand.

- There are different types of operators which are used as per user need. The different types of operators available in C are shown in Fig. 2.1.

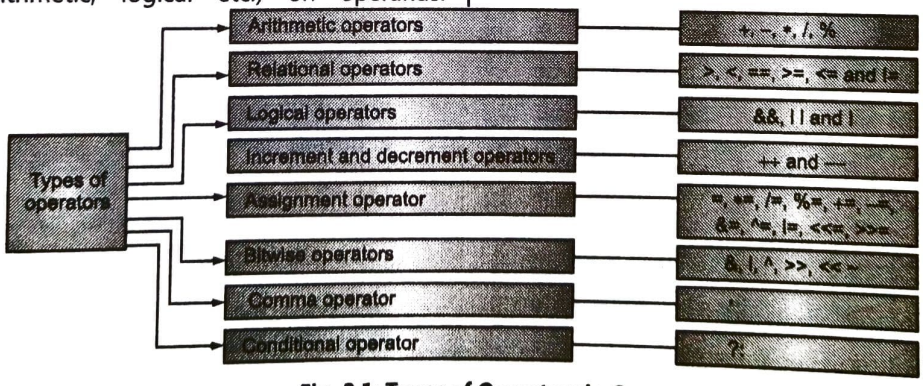


Fig. 2.1: Types of Operators in C

Properties of Operators:

- Fig. 2.2 shows properties of operators.

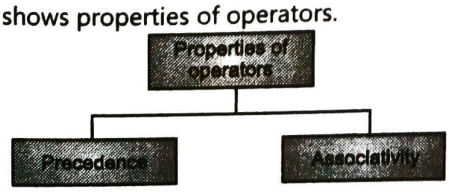


Fig. 2.2: Properties of Operators

1. **Precedence:** Precedence of operators refers to the priority given to the operator for a process.

When an expression contains many operators, the operations are carried out according to the priority of the operators. The higher priority operations are solved first.

For example, in arithmetic operators, the operators *, / and % are highest priority and of similar precedence. The operators + and – are having lowest precedence.

2. Associativity: When an expression has operators with equal precedence, the associativity property decides which operation to be carried out first. In simple words, associativity means the direction of execution. Associativity is of two types as explained below:

(i) Left to Right: In this type of associativity expression evaluation starts from the left to right direction.

Example: 12*4 / 8 % 2

In the above expression, all operators have the same precedence so the associativity rule is as follows. Multiplication is performed first, division, and finally modulous operation.

= 48 / 8 % 2
= 6 % 2
= 0

(ii) Right to Left: In this type of associativity expression evaluation starts from right to left direction.

Example: m = 8 + 5 % 2

In the above expression, assignment operator has right to left associativity, hence the right-hand side of assignment operator will be executed first (8+5%2) then its result is assigned to m (i.e. 8 + 1 = 9).

2.2 CHARACTER SET

- A character set can be defined as "a set of characters that either individually or in combination represents information".
- A character is indicated by any alphabets in lower case or uppercase, digits or special symbols/characters. These characters/symbols are used to write any kind of C language program.
- Character set is set of alphabets, letters and some special characters that are valid in C language and can be used in source program.

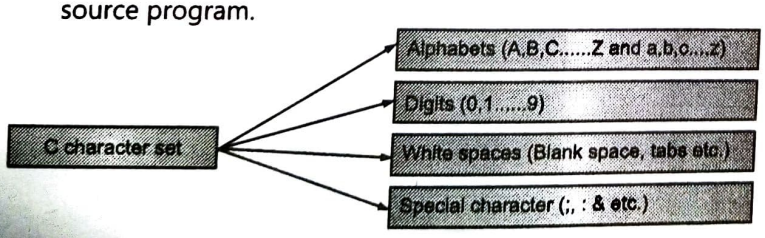


Fig. 2.3: Character Set in 'C' Language

- Table 2.1 shows character set in 'C' language while Table 2.2 shows special characters in C language.

Table 2.1: Character Set

[illegible]	Digits	White Spaces
[illegible] letters	All decimal digits 0 to 9	Black space Horizontal tab (\t) Vertical tab (\v) New line (\n) Form feed

Table 2.2: Set of Special Characters

Symbol	Name	Symbol	Name
,	Comma	&	Ampersand
.	Period or dot	^	Caret
;	Semicolon	*	Asterisk
"	Double Quote	–	Minus
'	Apostrophe	+	Plus
"	Quotation mark	<	Less than
!	Exclamation mark	>	Greater than
|	Vertical bar	()	Parenthesis left/right
/	Slash	[]	Bracket left/right
\	Back slash	{}	Braces left/right
~	Tilde	%	Percent sign
$	Dollar	=	Equal to
?	Question mark	@	At the rate

- Language pattern of C language uses special kind of sym which are called as delimiters.
- C uses following delimiters:

1. : (Colon) : Useful for label.
2. ; (Semicolon) : Terminate statements.
3. () (Parenthesis) : Used in expression and functio
4. [] (Square bracket) : Used for array declaration.
5. {} (Curly brace) : Scope of statement.
6. # (Hash) : Pre-processor directive.
7. , (Comma) : Variable separator.

2.3 TOKEN

- Tokens are the basic buildings blocks in C language whic are constructed together to write a C language program.
- In a C language, the smallest individual unit in a C program known as token.
- Fig. 2.4 shows various tokens in C language.

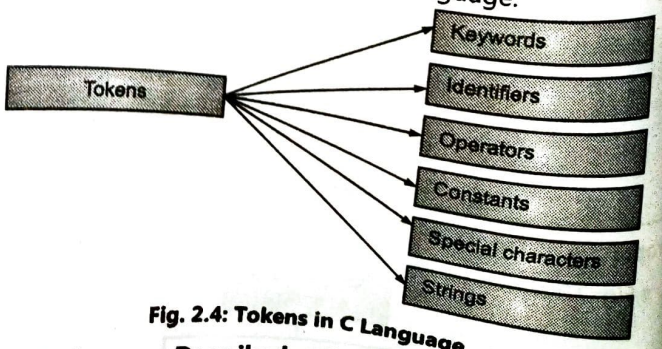


Fig. 2.4: Tokens in C Language

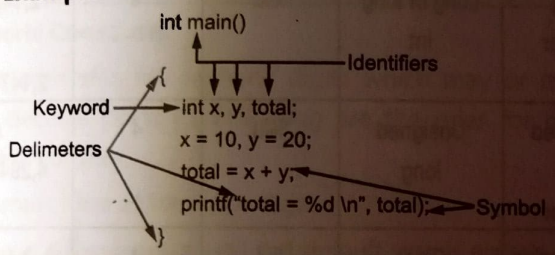
- **The Tokens are Described as Follows:**

1. **Keywords:** Reserved by the compiler and there are 32 keywords (ANSI standard) like auto, int, while etc.

variables, functions, and labels in C program. Emp_No, Sum, Student_ID are the examples of identifiers.

3. **Constants:** Constants are assigned to variables and they cannot be changed during the execution of program. For example: 1, 2.5, "C programming is easy."
4. **Operators:** Operators are the symbol which operates on value or a variable. For example: + is a operator to perform addition.
5. **Special Characters:** These characters are used in different declarations in C. like ; (semicolon) indicates end of an instruction.
6. **Strings:** Sequence of characters enclosed in double quotation marks like "C Programming".

For Example:



2.4 VARIABLES

- A variable is a named location in memory that is used to hold a value that can be modified by the program.
- In C language, a quantity which may vary (change) during program execution is called a variable. In short, variable is the name given to the location of the memory.

Rules for Variable Naming:

1. Variable name is the combination of alphabets, digit and underscore.
2. First character in the variable name must be an alphabet.
3. A keyword cannot be a variable.
4. No commas, blanks and special symbols are allowed in the variable name.
5. Variables are case sensitive.

2.4.1 Keywords

- The C language keywords are reserved words by the compiler, and they have been assigned fixed (special) meaning.
- The keywords cannot be used as variable names or function names because they have been assigned fixed meanings.
- The 32 C keywords provided in ANSI C are listed in Table 2.2.

Keyword Type	Keyword	Keyword Type	Keyword
Storage classes	auto	Storage type qualifier	const
	static		volatile
	register	User data types	enum
	extern		struct
Data types and Type modifiers	typedef		union
	void**	Control	if
	char		else
	short		while
	int		do
	long		for
	float		switch
	double		case
	signed		default
	unsigned		continue
	struct or union specifier*		break
	enum specifier*		goto
	typedef name*	Functions	return
			sizeof

Notes: * indicates that it is the name of a data type the user has created.

** indicates absence of a type specifier.

2.4.2 Identifiers

- In C, the names of variables, functions, labels, and various other user defined items are called identifiers.
- The rules which should be followed while naming an identifier are:
  - An identifier can only have alphanumeric characters (a-z, A-Z, 0-9) and underscore.
  - Keywords are not allowed to be used as identifiers.
  - Identifiers are also case sensitive in C. For example name and NAME are two different identifiers.
  - The first character of an identifier can only contain alphabet (a-z, A-Z) or underscore.
  - No sort of characters, such as punctuation marks, arithmetic signs or blanks are permitted to be used as part of an identifier.

Examples of Valid Identifiers:

sum, name, area, of_sum, Rate of_interest, area1

Examples of Invalid Identifiers:

2.5 DATA TYPES AND SIZES

- All high-level programming languages support the concept of data types. A data type defines a set of values that a variable can store along with a set of operations that can be performed on that variable.

Definition:

- The type of data that variables may hold in a programming language are known as data type.

OR

- C data types are defined as "the data storage format that a variable can store a data to perform a specific operation".
- C is rich in its data types. In C language, data types are broadly classified as Basic data types (primitive data types), Derived data types and User-defined data types as shown in Fig. 2.5.

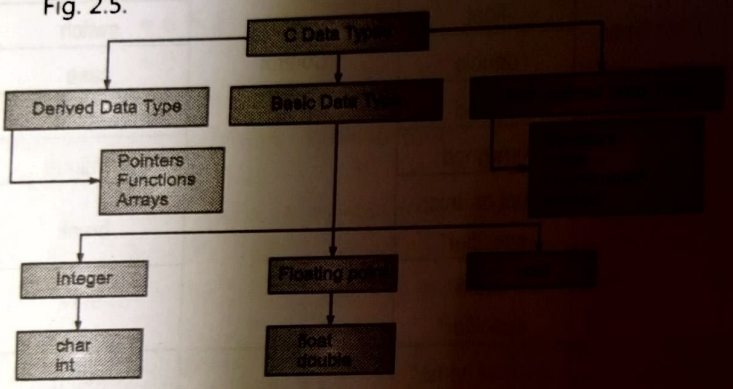


Fig. 2.5: Data Types in C

2.5.1 Basic Data Types

- Basic data types of C language also called as built-in data types or primitive data types.
- In C compilers support five basic fundamental data types, namely integer (int), character (char), floating point (float), double-precision floating points (double) and void. Many of them also offer extended data types such as long int and long double.
- Various data types and the terminology used to describe then are given in Fig. 2.6.

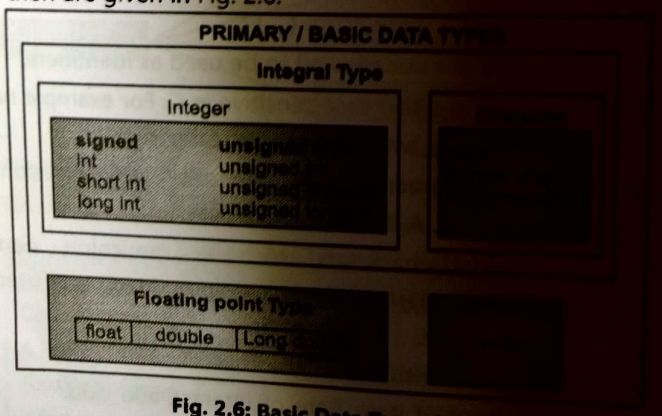


Fig. 2.6: Basic Data Types in C

- Table 2.4 shows ranges of primary data types in C language.

For Example :

```
enum week
{
Sunday, Monday, Tuesday, Wednesday,
Thursday, Friday, Saturday
} day1, day2;
```

- Above example creates a new type week in which the enumeration-list (i.e., Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday). The values for the enumeration list are set automatically to integers 0 to 6. Internally, the compiler treats the enumerators as integers. Each value o the list of permissible values corresponds to an intege starting with 0. This way of assigning numbers can b overridden by the programmer by initializing th enumerators to different integer values.

2.6 CONSTANTS

- Constants are a fixed value which does not change during program execution.
- Constants are also refers as literals.
- Fig. 2.7 shows classification of constants in C language.

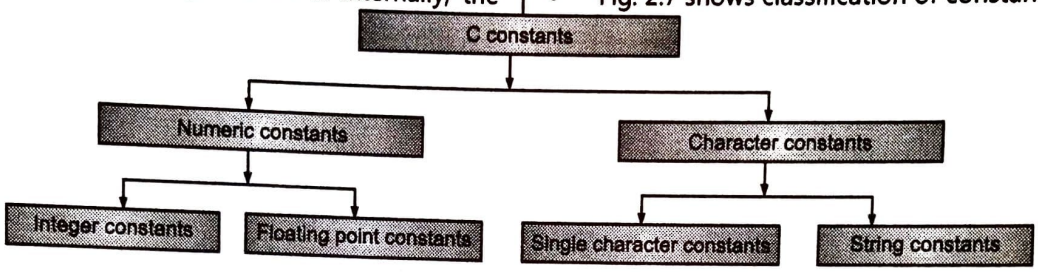


Fig. 2.7: Constants in C Language

1. Numeric Constants:

- These constants are numeric digits which may or may not have decimal point (.). Following are the rules for defining numeric constant:

➢ They must have at least one digit.

➢ They can either (+ve or -ve) but default sign is always +ve.

➢ No comma and space is allowed within the numeric constant.

(i) Integer Constant:

- An integer constant is whole number without any fractional value.
- Simply, an integer constant refers to a sequence of digits.

Valid examples: 10, 20, +30, –15

Invalid integer constants: 2.3, .235, $76. 3*^6

Integer constant can be written in three different number system i.e.,

➢ Decimal (base 10) constant,

➢ Octal (base 8) constant and

➢ Hexadecimal (base 16) constant.

- **A Decimal Constant** can be any combination of digits from 0 through 9. If the constant contains more than one digit, the first digit must be other than 0.

Examples:

424, 1248

- **An Octal Integer Constant** can consists of any combination of digits taken from the set 0 through 7. However, the first digit must be 0 in order to identify the constant as an octal number.

Examples:

0, 06

- **A Hexadecimal Integer Constant** must begin with either ox or OX. It can be followed by any combination of digits taken from the sets 0 through 9 and a through f (or A through F) represents the decimal quantity 10 through 15, respectively.

Examples:

ox, OX, ox8fffox

(ii) Floating-Point Constant:

A floating point constant consists of a whole number followed by the decimal point and also one or more decimal numbers following the decimal points which makes the fractional part.

Examples:

0.6, 12.00114

0.009e-2, .23e-10

2. Character Constant:

These constant are used to represent characters in the execution character set. C provides following characters constants:

(i) Single Character Constant: A character constant is a single character enclosed in single quotation marks (' ').

Examples: 'A', 'a', '9'.

(ii) String Constant: A string constant is a sequence of alphanumeric characters enclosed in double quotation marks (" ").

Examples: "Amar", "4567".

A symbolic constant is a constant represented by a name (symbol) in our program. Like a literal constant, a symbolic constant can't change.

- We can define a constant in any of the following ways:

1. #define Statement: You can use this preprocessor directive at the beginning of a program to define the name and value of a constant The directive replaces the constant by its value at each

of its occurrence in the program. Such constant is also referred as symbolic constant.

Syntax: #define <Constant_name> <Constant_value>

Example: # define g 10.9

2. **Const Keyword:** You can use the const keyword to define a constant in a program and assign it a value.

Syntax: const datatype <Constant_name> <Constant_value>

Example: const float g 10.9

- You can use the enum keyword to define a constant as an enumerated data type and assign it a set of constant values.

  Syntax: enum <Constant_name> <set of values>

  Example: enum option (yes, no);

2.6.1 Declaration and Initialization of Variables

1. **Declaration of Variables:**

- Before you can use a variable in a C program, it must be declared. A variable declaration tells the compiler the name and type of a variable.
- The declaration may also initialize the variable to as specific value.

  Syntax: data_type variable_name;

  Example: int x, y, z;

2. **Initialization of Variables:**

- Variable initialization means assigning a value to the variable.
- Values may be assigned to variables using the '=' operator.

  Syntax: data_type variable_name = value;

  Example: int x = 50, y = 40;

- When the values to the variable are assign at time of declaration are called as declaration and initialization.

  int number=120;

- In the above statement, number is declared of type integer and at the same time value 120 is assign to it. This is called as declaration and initialization.

Program 2.1: Program for variable declaration and initialization.

```
#include<stdio.h>
/* Variable declaration */
extern int a, b;
extern int c;
extern float f;
int main ()
{
    /* variable definition... */
    int a, b;
    int c;
    float f;
    clrscr();
    /* actual initialization... */
    a = 10;
    b = 20;
    c = a + b;
    printf("value of c : %d \n", c);
    f = 70.0/3.0;
    printf("value of f : %f \n", f);
    return 0;
}
```

Output:

```
TC
value of c : 30
value of f : 23.333334
```

Difference between Variable Declaration and Definition:

Sr. No	Variable Declaration	Variable Definition
1.	Declaration tells the compiler about data type and size of the variable.	Definition allocates memory for the variable.
2.	Variable can be declared many times in a program.	It can happen only one time for variable in a program.
3.	The assignment of properties and identification to a variable.	Assignments of storage space a variable.

2.6.2 Variable Scope

- A scope in any programming is a region of the prog where a defined variable can have its existence (visible) beyond that variable it cannot be accessed.
- There are three places where variables can be declared programming language:
  - ➢ Inside a function or a block which is called variables.
  - ➢ Outside of all functions which is called global variab
  - ➢ In the definition of function parameters which are formal parameters.

Program 2.2: Program for global and local variables.

```
#include<stdio.h>
int n=20;     /* n is global variable... */
int main()
{
    int m=40;/* m is local variable... */
    clrscr();
    if (m == n)
    {
        printf("m and n are equal");
    }
    else
```

```
{
    printf("m and n are not equal");
}
return 0;
```

tput:

```
TC
and n are not equal
```

7 DECLARATIONS

A declaration is a statement that sets aside some space in memory. Declaration consists of a type followed by a list of variables.

In C language there are various declaration are used like variable declaration, function declaration, constant declaration and so on.

In C language all variables must be declared before use, although certain declarations can be made implicitly by context.

A declaration specifies a type, and contains a list of one or more variables of that type, as in,

```
int lower, upper, step;
char c, line[100];
```

Variables can be distributed among declarations in any fashion, this lists above could equally well be written as

```
int lower;
int upper;
int step;
char c;
char line[100];
```

This later form takes more space, but is convenient for adding a comment to each declarations or for subsequent modifications.

A variable may also be initialized in its declaration. If the name is followed by an equals signs and an expression, the expression serves as an initializer, as in by an equals sign and an expression, the expression serves as an initializer, as in

```
char esc = '\\';
int i = 0;
int limit = MAXLINE+1;
float eps = 1.0e-5;
```

If the variable in question is not automatic, the initialization is done once only, conceptually before the program starts executing, and the initializer must be a constant expression.

- An explicitly initialized automatic variable is initialized each time the function or block it is in is entered; the initializer may be any expression.
- External and static variables are initialized to zero by default. Automatic variables for which there is no explicit initializer have undefined (i.e. garbage) values.
- The qualifier constant can be applied to the declaration of any variable to specify that its value will not be changed. For an array, the const qualifier says that the elements will be not be altered.

```
const double e = 2.71828182845905;
const char msg[] = "warning:";
```

- The const (constant) declaration can also be used with array arguments, to indicate that the function does not change that array:

```
int strlen (const char[]);
```

The result is implementation - defined if an attempt is made to change a const.

2.8 EXPRESSIONS

- An expression is a statement in C language.
- An expression is a combination of operands (i.e., constants, variables etc.) connected by operators and parenthesis. For example, in the expression given below X and Y are operands and '+' is an operator.
  Z = X + Y;
- An expression is defined as "a combination of operands and operator(s) to obtain some computation".
- C language supports may types of operators such as arithmetic, relational, logical etc.
- An expression that involves arithmetic operators is known as an arithmetic expression. The computed result of an arithmetic expression is always a numerical value.
- The expression which involves relational and/or logical operators is called as a Boolean expression or logical expression. The computed result of such an expression is an logical value, i.e., either 1 (True) or 0 (False).
- The rules of formation of an expression are as given below:
  - ➢ A signed or unsigned constant or variable is an expression.
  - ➢ Two operators should not occur in continuation, (may occur in some cases).
    For example: C = a++;

    or

    C = ++a;
  - ➢ An expression connected by an operator to a variable or a constant is an expression.
  - ➢ Two expressions connected by an operator is also an expression.

2.9 ARITHMETIC OPERATORS

- There are two types of arithmetic operators i.e., binary operator and unary operator.

2.9.1 Binary Operators

- Table 2.5 shows different arithmetic operators that are used in C language. These operators are commonly used in most of the computer languages. These arithmetic operators are used for numerical calculations between the two constant values.
- They are also called binary arithmetic operators. Binary operators are those operators which require two operands.
- Assume variable A holds 10 and variable B holds 20 then:

Table 2.5: Arithmetic Operators in C

Operator	Description	Example
+	Adds two operands.	A + B will give 30
-	Subtracts second operand from the first.	A - B will give -10
*	Multiply both operands.	A * B will give 200
/	Divide numerator by enumerators.	B / A will give 2
%	Modulus Operator and remainder of after an integer division.	B % A will give 0

2.9.2 Unary Operators

1. Unary Minus (–) and Plus (+) Operators

- We know that the operator that needs one operand is called unary operator.
- Unary minus operator is used for indicating or changing the algebraic sign of a value.
- Unary operators and their descriptions are given in the Table 2.6.

Table 2.6: Unary Arithmetic Operators

Operator	Description
+	Plus
–	Minus
+ +	Increment
– –	Decrement
&	Address operator
sizeof()	Sizeof operator

For example:

```
int x = - 40;
int y = -x;
```

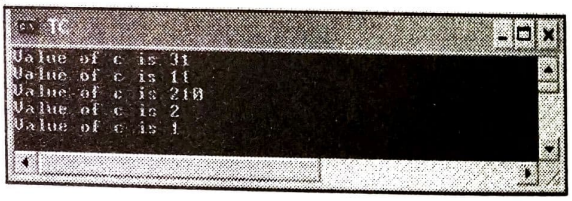
This assigns the value of – 40 to 'x' and the value of 40 to 'y' through 'x'. The – sign used in this way is called the unary operator because it takes just one operand.

- There is no unary plus (+) used in C language. A value assigned with plus sign is also valid. For example, int x = + 40; here, + is valid but in practice this sign should not be used.

Program 2.3: Program for arithmetic operators:

```
#include<stdio.h>
void main()
{
    int a = 21;
    int b = 10;
    int c ;
    clrscr();
    c = a + b;
    printf("Value of c is %d\n", c );
    c = a - b;
    printf("Value of c is %d\n", c );
    c = a * b;
    printf("Value of c is %d\n", c );
    c = a / b;
    printf("Value of c is %d\n", c );
    c = a % b;
    printf("Value of c is %d\n", c );
    getch();
}
```

Output:



2. Increment and Decrement Operators

- Increment (++) and decrement (– –) operators are also u operators.
- The operator ++ adds one to its operand, whereas operator – – subtracts one from its operand.
- For justification, a = a + 1; can be written as a++; and a= can be written as a– –;. Both of these operators may e follow or precede the operand. i.e. a = a + 1; ca represented as a++; or ++a;
- If '++' or '– –' are used as suffix to the variable name, post-increment/decrement operation take place. Conside example for understanding '++' operator as a suffix to variable.

  a = 20;

  b = 10;

  c = a * b ++;
- In the above equation, the current value of 'b' is used fo product. The result is 200, which is assigned to 'c'. A multiplication, the value of 'b' is incremented by one.
- If '++' or '– –' are used as prefix to the variable name, pre increment/decrement operation take place. Conside

example of understanding '++' operators as a prefix to the variable.

a = 20;

b = 10;

c = a * ++ b;

- In the above equation, the value of 'b' is incremented and then multiplication is carried. The result is 220, which is assigned to 'c'.

Program 2.4: Program of increment and decrement operators.

```
#include<stdio.h>
int main()

    int a = 21;
    int c;
    clrscr();
    c = a++;
    printf("Value of c is %d\n", c);
    c = a--;
    printf("Value of c is %d\n", c);
    return 0;
```

Output:

2.10 RELATIONAL OPERATORS

- Relational operators used in expressions and result of expression are either true or false.
- There are following relational operators supported by C language. Assume variable A holds 10 and variable B holds 20 then:

Table 2.7

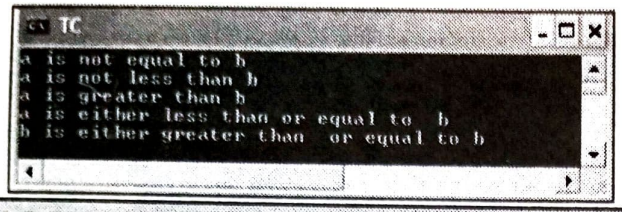
Operator	Description	Example
== (Equal to operator)	Checks if the value of two operands is equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (Not Equal to operator)	Checks if the value of two operands is equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (Greater than operator)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (Less than operator)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (Greater than or equal to operator)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (Less than or equal to operator)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Program 2.5: Program for relational operators.

```
#include<stdio.h>
int main()
{
    int a = 20;
    int b = 13;
    int c;
    clrscr();
    if( a == b )
    {
        printf("a is equal to b\n" );
    }
    else
    {
        printf("a is not equal to b\n" );
    }
    if ( a < b )
    {
        printf("a is less than b\n" );
    }
    else
    {
        printf("a is not less than b\n" );
    }
    if ( a > b )
    {
        printf("a is greater than b\n" );
    }
    else
    {
        printf("a is not greater than b\n" );
    }
    /* Lets change value of a and b */
    a = 15;
    b = 30;
    if ( a <= b )
    {
```

```
        printf("a is either less than or equal to  b\n" );
    }
    if ( b >= a )
    {
        printf("b is either greater than  or equal to b\n" );
    }
    return 0;
}
```

Output:



2.11 LOGICAL OPERATORS

Logical operators are used to perform logical operations on the given two variables.

There are following logical operators supported by C language. Assume variable A holds 10 and variable B holds 20 then :

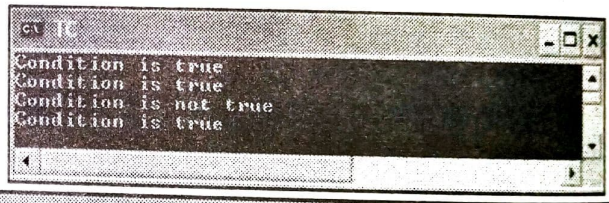
Table 2.8

Operate	Description	Example
&& (Logical AND operator)	If both the operands are non zero then then condition becomes true.	(A && B) is true.
|| (Logical OR Operator)	If any of the two operands is non zero then condition becomes true.	(A || B) is true.
! (Logical NOT Operator)	Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

Program 2.6: Program for logical operators.

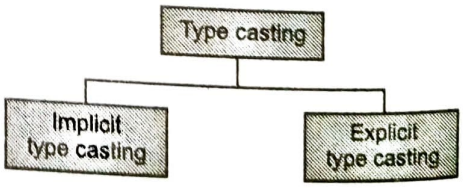
```
#include<stdio.h>
int main()
{
    int a = 5;
    int b = 20;
    int c ;
    clrscr();
    if ( a && b )
    {
        printf("Condition is true\n" );
    }
    if ( a || b )
    {
        printf("Condition is true\n" );
    }
    /* lets change the value of  a and b */
    a = 0;
    b = 10;
    if ( a && b )
    {
        printf("Condition is true\n" );
    }
    else
    {
        printf("Condition is not true\n" );
    }
    if ( !(a && b) )
    {
        printf("Condition is true\n" );
    }
    return 0;
}
```

Output:



2.12 TYPE CASTING

- 'C' language supports type casting feature. This type cast feature provides a mechanism to change data type o variable.
- Usually, type casting performs conversion of data type another data type.
- Type casting in 'C' programming language is categorized i two parts as shown below:



1. Implicit Type Casting:

- In this type casting, the type conversion is perform automatically by the compiler without human (programm interaction.
- Implicit type casting also known as automatic ty conversion in which the type conversion automatic performed by the C compiler.
- In implicit type casting, data type of operands is convert

implicit type casting provides to put value into large container or data storage.

It helps to perform calculation among different data types and obviously, result will not exceed the range of largest data type among operands. It will not truncate value since it got enough space after implicit type casting. It performs lowest data type conversion into highest data type.

A typical type conversion process is illustrated in Fig. 2.8.

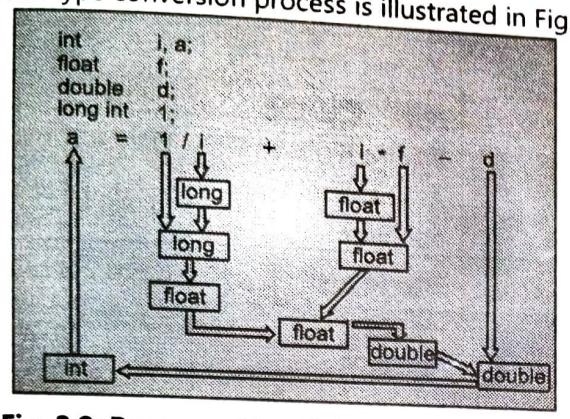


Fig. 2.8: Process of Implicit Type Conversion

Given below is the sequence of rules that are applied while evaluating expressions.

All short and char are automatically converted to int, then

(i) If one of the operands is long double, the other will be converted to long double and result will be long double.

(ii) Else, if one of the operands is double, the other will be converted to double and the result will be double;

(iii) Else, if one of the operands is float, the other will be converted to float and the result will be float;

(iv) Else, if one of the operands is unsigned long int, the other will be converted to unsigned long int and the result will be unsigned long int;

(v) Else, if one of the operands is long int, and the other is unsigned int, then

(a) if unsigned int can be converted to long int, the unsigned int operand will be converted as such as the result will be long int;

(b) else, both operands will be converted to unsigned long int and the result will be unsigned long int;

(vi) Else, if one of the operands is long int, the other will be converted to long int and the result will be long int;

(vii) Else, if one of the operands is unsigned int, the other will be converted to unsigned int and the result will be unsigned int;

Note that, 'C' language uses the rule that, in all expressions except assignments, any implicit type conversions are made from a lower size type to a higher size type as shown in Fig. 2.9.

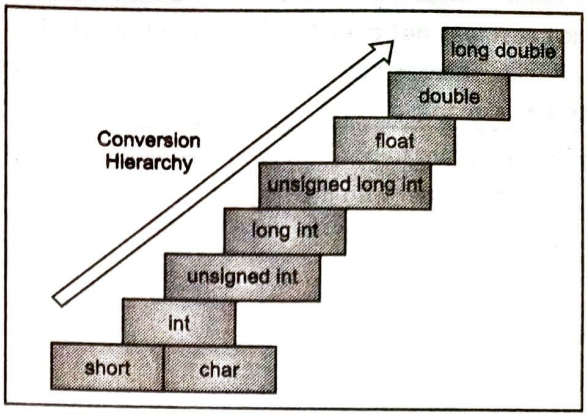


Fig. 2.9

- Note that some versions of C automatically convert all floating-point operands to double precision.
- The final result of an expression is converted to the type of the variable on the left of the assignment sign before assigning the value to it. However, the following changes are introduced during the final assignment.
  - ➢ Float to int causes truncation of the fractional part.
  - ➢ Double to float causes rounding of digits.
  - ➢ Long int to int causes dropping of the excess higher order bits.

2. Explicit Type Casting:

- Explicit type casting refers to the type conversion, i.e. performed explicitly using type cast operator.
- Explicit type casting is performed explicitly by the human or programmer. In this type casting, data type of operand is converted into specific data type mentioned by the programmer. It is done automatically by the compiler.
- Explicit type casting generally performs conversion of higher data type to lower data type. Therefore, explicit type casting may cause of truncate of value during type casting. It tries to put large value into smaller container.
- Obviously, it will overflow and the result will be truncated. For example, convert float data type into integer data type. It follows rules that are all integer types would be converted into float type, all float types would be converted to double, and all character types would be converted into integer type.
- The syntax of explicit type casting (type cast operator) is given below:

```
(data_type) expression;
```

- The above syntax is illustrated by the following example:

```
in a = 8;
float p = 5;
...
p = (float)(a) / p;
```

- **Explanation:** Since a is converted to a float before the division operation, the division will be between two floats and the result will be returned as a float with value 1.6. If the

explicit cast was not performed, the result would have been an int with value 1 and p would have been assigned the incorrect value 1.

Program 2.7: Program to perform type conversion from integer to floating point number with the help of typecasting operators.

```
#include<stdio.h>
#include<conio.h>
void main()
    {
    int a=5,b=2;
    float c;
    clrscr();
    c=(float)a/b;
    printf("\n Explicit type casting: %f",c );
    getch();
}
```

Output:

```
TC
Explicit type casting: 2.500000_
```

In above program, explicit type casting is performed. In this program, first a variable is converted into float type and then performed division operation. The result is in float and assigned to variable c.

2.13 ASSIGNMENT OPERATORS

- It is used to assign a value to variable.
- There are following assignment operators supported by C language :

Table 2.9

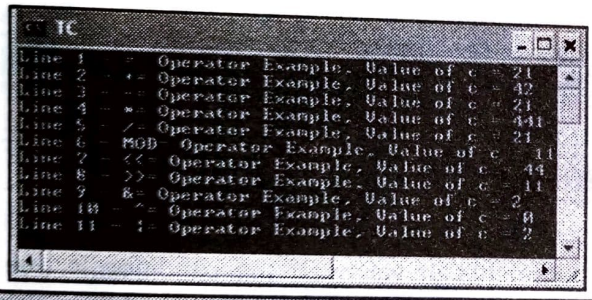
Operator	Description	Example
= (Simple assignment operator)	Assigns values from right side operands to left side operand.	C = A + B will assigns value of A + B into C
+= (Add AND assignment operator)	It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
- = (Subtract AND assignment operator)	It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C - A
*= (Multiply AND assignment operator)	It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/= (Divide AND assignment operator)	It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%= (Modulus AND assignment operator)	It takes modulus using two operands and assign the result to left operand.	C%= A is equivalent to C = C % A
<<=	Bitwise Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Bitwise Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Program 2.8: Program for assignment operators.

```
#include<stdio.h>
int main()
{
    int a = 21;
    int c ;
    clrscr();
    c = a;
    printf("Line 1 - = Operator Example, Value of c =
c );
    c += a;
    printf("Line 2 - += Operator Example, Value of c =
c );
    c -= a;
    printf("Line 3 - -= Operator Example, Value of c =
c );
    c *= a;
    printf("Line 4 - *= Operator Example, Value of c =
c );
    c /= a;
    printf("Line 5 - /= Operator Example, Value of c =
c );
    c = 200;
```

```
    printf("Line 6 - MOD= Operator Example, Value of c =
%d\n", c );
    c <<= 2;
    printf("Line 7 - <<= Operator Example, Value of c = %d\n",
c );
    c >>= 2;
    printf("Line 8 - >>= Operator Example, Value of c = %d\n",
c );
    c &= 2;
    printf("Line 9 - &= Operator Example, Value of c = %d\n",
c );
    c ^= 2;
    printf("Line 10 - ^= Operator Example, Value of c =
%d\n", c );
    c |= 2;
    printf("Line 11 - |= Operator Example, Value of c = %d\n",
c );
    return 0;
}
```

Output:



2.14 BITWISE OPERATORS

- Bitwise operators are used to manipulate the data at bit level.
- Basically computer does not understand any language. It knows only machine code which is in the form of 0's and 1's. These 0's and 1's are called bits. Using these bitwise operators multiple operations on these bits are done.
- Assume if A = 60; and B = 13; Now in binary format they will be as follows :

  A = 0011 1100

  B = 0000 1101

  A&B = 0000 1100

  A|B = 0011 1101

  A^B = 0011 0001

  ~A = 1100 0011

- There are following Bitwise operators supported by C language.

Table 2.10

Operator	Description	Example
& (Binary AND Operator)	Copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (Binary OR Operator)	Copies a bit if it exists in eather operand.	(A | B) will give 61 which is 0011 1101
^ (Binary XOR Operator)	Copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (Binary Ones Complement Operator)	Is unary and has the efect of 'flipping' bits.	(~A) will give -60 which is 1100 0011
<< (Binary Left Shift Operator)	The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (Binary Right Shift Operator)	The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Program 2.9: Program for bitwise operators.

```
#include<stdio.h>
int main()
{
    unsigned int a = 60;/* 60 = 0011 1100 */
    unsigned int b = 13;/* 13 = 0000 1101 */
    int c = 0;
    clrscr();
    c = a & b;       /* 12 = 0000 1100 */
    printf("Value of c is %d\n", c );
    c = a | b;       /* 61 = 0011 1101 */
    printf("Value of c is %d\n", c );
    c = a ^ b;       /* 49 = 0011 0001 */
    printf("Value of c is %d\n", c );
    c = ~a;          /*-61 = 1100 0011 */
    printf("Value of c is %d\n", c );
    c = a << 2;      /* 240 = 1111 0000 */
    printf("Value of c is %d\n", c );
    c = a >> 2;      /* 15 = 0000 1111 */
    printf("Value of c is %d\n", c );
    return 0;
}
```

Output:

```
TC
Value of c is 12
Value of c is 61
Value of c is 49
Value of c is 61
Value of c is 240
Value of c is 15
```

2.15 COMMA AND SIZEOF OPERATORS

2.15.1 Comma Operator

- The comma (,) operator is used to separate two or more expressions.
- The comma operator has the lowest priority among all the operators.
- It is not essential to enclose the expressions with comma operators within the parenthesis. For example the statements given below are valid.

  For example:

  a = 2, b = 4, c = a + b;

  (a = 2, b = 3, c = a + b;)

Program 2.10: Program for comma (,) operator.

```
#include<stdio.h>
int main()
{
    int a, b;
    clrscr();
    a=1,2,3,4,5;
    b=(1,2,3,4,5);
    printf("Resultant values of a and b are:\n");
    printf("%d %d", a, b);
    return 0;
}
```

Output:

```
TC
Resultant values of a and b are:
1 5
```

2.15.2 sizeof Operator

- It is an unary operator which provides the size, in bytes, of the given operand.
- The syntax of sizeof operator is,

  Syntax: `sizeof (operand)`

Here, the operand is a built-in or user-defined data type or variable.

- The sizeof operator always precedes its operand.

  For example: sizeof(float)

  returns the value 4.
- This information is quite useful when we execute our program on a different computer or a new version of C is used. The sizeof operator helps in case of dynamic memory allocation for calculating the number of bytes used by some user-defined data type.

Program 2.8: Program for sizeof operator.

```
#include<stdio.h>
int main()
{
    int a;
    float b;
    double c;
    char d;
    clrscr();
    printf("Size of int=%d bytes\n",sizeof(a));
    printf("Size of float=%d bytes\n",sizeof(b));
    printf("Size of double=%d bytes\n",sizeof(c));
    printf("Size of char=%d byte\n",sizeof(d));
    return 0;
}
```

Output:

```
TC
Size of int=2 bytes
Size of float=4 bytes
Size of double=8 bytes
Size of char=1 byte
```

2.16 TERNARY CONDITIONAL OPERATOR

- The conditional operator contains a condition followed two statements or values. If the condition is true the statement is executed, otherwise the second statement.
- The conditional operator (?) and (:) are sometimes ca ternary operators because they take three arguments.
- The syntax of conditional operator is as given below:

  Condition ? (expression 1): (expression2);
- Two expressions are separated by a colon. If the conditio true expression1 gets evaluated, otherwise expression 2. condition is always written before ? mark.
- The program given below illustrates the use of a conditi operator.

Program 2.12: Program for ternary operator.

```
#include<stdio.h>
void main()
{
    clrscr();
    printf ("\n Result = %d", 2==3? 4:5);
}
```

Output:

2.17 OPERATOR PRECEDENCE AND ASSOCIATIVITY

- When the expression involves more than one operation, then which operation is to be performed first is decided by the precedence of that operator. Highest precedence operation is solved first. Each operator of 'C' language has precedence associated with it.
- The operators with the same precedence are evaluated according to there associativity. That is, the expression is evaluated either from left to right or right to left.
- In short operator precedence determines a rank for the operators in an expression while associativity determines the grouping of operations among operators of the same precedence.
- The precedence and associativity of all operators in 'C' language is given in the Table 2.11.

Table 2.11: Precedence and Associatively of Operators in C

Category	Operator	What it does?	Associativity
1. Highest precedence	( ) [ ] – > : : .	Function call Array subscript C indirect component selector C scope access/resolution C direct component selector	Left to Right
2. Unary	! ~ + – + + – – & * sizeof	Logical negation (NOT) Bitwise (1's) component Unary plus Unary minus Pre-increment or post-increment Pre-decrement or post-decrement Address Indirection (returns sizeof operand, in bytes)	Right to Left
3. Member access	.* – > *	Dereference Dereference	Left to Right
4. Multiplication	* / %	Multiply Divide Remainder (modulus)	Left to Right
5. Additive	+ –	Binary plus Binary minus	Left to Right
6. Shift	<< >>	Shift left Shift right	Left to Right
7. Relational	< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left to Right
8. Equality	= = ! =	Equal to Not equal to	Left to Right
9. Bitwise AND	&	Bitwise AND	Left to Right
10. Bitwise XOR	^	Bitwise XOR	Left to Right
11. Bitwise OR	:	Bitwise OR	Left to Right
12. Logical AND	&&	Logical AND	Left to Right
13. Logical OR	|| ||	Logical OR	Left to Right
14. Conditional	?:	(exp ? x : y means "if exp then x, else y")	Right to Left
15. Assignment	= *= /= %= += –= &= ^= := <<= >>=	Simple assignment Assign product Assign quotient Assign remainder (modulus) Assign sum Assign difference Assign bitwise AND Assign bitwise XOR Assign bitwise OR Assign left shift Assign right shift	Right to Left
16. Comma	,	Separator of expression	Left to Right

EXERCISE

1. What is meant by token in C language?
2. Explain character set of C language in detail.
3. What is meant by datatype?
4. Describe primitive and enumerated data types in 'C' language detail.
5. What is expression? Enlist its types.
6. Define the following terms:
   (i) variable
   (ii) constant
   (iii) identifier.
7. Explain type casting in detail.
8. Write short note on: Identifier.
9. Explain the term constant in detail.
10. Differentiate between declaration and initialization.
11. What is an operator and expression?
12. Enlist arithmetic operators in C language.
13. Write short note: Bitwise operators.
14. Explain comma operator with example.
15. What are the types of assignment operator? Explain them in short.
16. Describe logical operators in detail.
17. Explain operator precedence and associativity of ope in detail.
18. Explain ternary conditional operator with example.
19. Explain increment and decrement operators in detail.

✠ ✠ ✠