Year : I

Semester : II

Subject Title : C PROGRAMMING

Sub Code : 18BCS23C

UNIT – I :Over view of C-Importance of C-Sample C programs-Basic structure of C programs - Constants, Variables, and Data Types - C tokens - Keywords and Identifiers – Constants – Variables-Data types -Operators and Expressions – Arithmetic Operators-Relational operators-Logical operators- Assignment operators-Increment and decrement operators-Conditional operator-Bitwise operators - Special operators-Type conversion in expressions-Operator precedence and associativity .

UNIT – II :Managing Input and Output Operations -Reading a character-Writing a character-Formatted input -Formatted output - Decision Making and Branching-Decision making with IF statement-Simple IF statement-The IF ELSE statement--The switch statement-The?: Operator-The GOTO statement - Decision Making and Looping-The WHILE statement-The DO statement-The FOR statement

UNIT – III: Arrays – Introduction-One dimensional arrays-Two dimensional arrays-Multidimensional arrays.

Character Arrays and Strings-Declaring and initializing string variables-Reading strings from terminal-Writing strings to screen-Arithmetic operations on characters- Putting strings together -Comparison of two strings – String-Handling functions.

UNIT – IV User-Defined Functions – Introduction-Need for user-defined function-The form of C functions Return values and their types-Calling a function-Category of functions – Recursion-Functions with arrays-The scope and lifetime of variables in functions.

Structures and Unions-Structure definition Giving values to members-Structure initialization-Comparison of structure variables-Arrays of structure variables-Arrays within structures-Structures within structures-Structures and functions – Unions-Size of structures-Bit fields.

UNIT – V Pointers-Understanding pointers-Accessing the address of a variables-Declaring and initializing pointers-Accessing a variable through its pointer-Pointer expressions-Pointers and arrays-Pointers and character strings-Pointers and functions-Pointers and structures.

File Management in C-Defining and opening a file-Closing a file-Input/Output operations on files Error handling during I/O operations-Random access to files-Command line arguments.

TEXT BOOKS 1. E.Balagurusamy,"Programming in ANSI C", Seventh Edition McGraw Hill Education India Private Ltd, 2017

UNIT I

CHAPTER I

1.1 OVERVIEW OF C

C Language is a

- Structured
- High-level
- Machine independent language

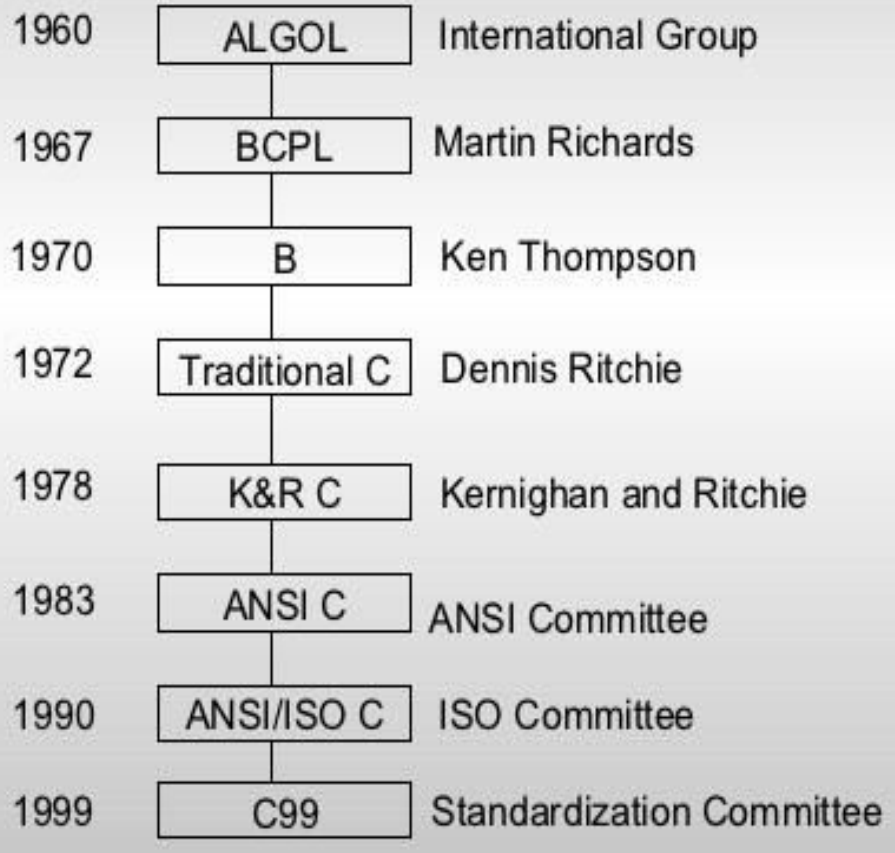
History of C

- The root of all modern languages is ALGOL (introduce in 1960s).
- ALGOL uses a structure programming.
- ALGOL is popular in Europe
- Computer scientists like Corrado Bohm, Guiseppe Jacopini and Edsger Dijkstra popularised the structured programming concepts.
- In 1967, Martin Richards developed a language called BCPL (Basic Combined Programming Language)

Contn…

- Primarily BCPL is developed for system software.
- In 1970, Ken Thompson created a new language called B.
- B is created for UNIX os at Bell Laboratories.
- Both BCPL and B were “typeless” languages
- C was developed by Dennis Ritchie at the Bell Laboratories in 1972.
- Added new features and concepts like “data types”.
- C was evolved from ALGOL, BCPL and B.
- It was developed along with the UNIX operating system.
- It was strongly integrated with the UNIX operating system.
- In 1983 American National Standards Institute (ANSI) appointed a technical committee to define a standard for C.
- The committee approved a version of C in December 1989 which is now known as ANSI C.
- In 1990 International Standards Organization (ISO) has approved C and this version of C is referred to as C89.

History of C


1960
ALGOL
International Group
1967
BCPL
Martin Richards
1970
B
Ken Thompson
1972
Traditional C
Dennis Ritchie
1978
K&R C
Kernighan and Ritchie
1983
ANSI C
ANSI Committee
1990
ANSI/ISO C
ISO Committee
1999
C99
Standardization Committee

1.2 IMPORTANCE OF C

It is a robust language whose rich set of built-in functions and operators can be used to write any complex program.

The C compiler combines the capabilities of an assembly language with the features of a high-level language and therefore it is well suited for writing both system software and business packages.

Programs written in C are efficient and fast.

C is highly portable.

C language is well suited for structured programming.

Ability to extend itself.

A C program is basically a collection of functions that are supported by the C library.

1.3 SAMPLE PROGRAM 1

```
main()   <----------------------------- Function Name
{<------------------------------------- Start of Program
    /* ……. Printing begins ……. */
    printf(" Hello Program "); <--------- Program Statements
    /* ……. Printing ends ……. */
} <------------------------------------- End of Program
```

Output

Hello Program

SAMPLE PROGRAM 2

```
main()
{
int number;
float amount;

number =100;
amount=30.75+75.35;
printf(“%d\n”,number);
printf(“%5.2f”,amount);
}
```

Output

100

106.10

1.4 BASIC STRUCTURE OF C PROGRAM

Documentation Section → **optional**
Link Section
Definition Section → optional
Global Declaration Section → optional

main() Function Definition
{
Declaration part
Executable part
}

Subprogram Section → optional

Function 1	**User-defined functions**
Function 2	

Function n	

Documentation section

A set of comment lines giving the name of the program, the author and other details.

Link section

Provides instructions to the compiler to link functions from the system library.

Definition section

defines all symbolic constants.

Global declaration section

There are some variables that are used in more than one function.

Such variables are called global variables.

Global variables are declared in the global declaration section that is outside of all the functions.

Also declares all the user-defined functions.

Main() function

Every c program must contain main() function section.

This section contains two parts

- Declaration part
- Executable part

These two parts must appear between the opening and closing braces ({ }).

Declaration part

Declares all the variables used in the executable part

Executable part

There is atleast one statement in the executable part.

The closing brace of main function sections is the logical end of the program.

All statements in the declaration and executable parts end with the semicolon(;).

Subprogram section

contains all user-defined functions that are called in the main function.

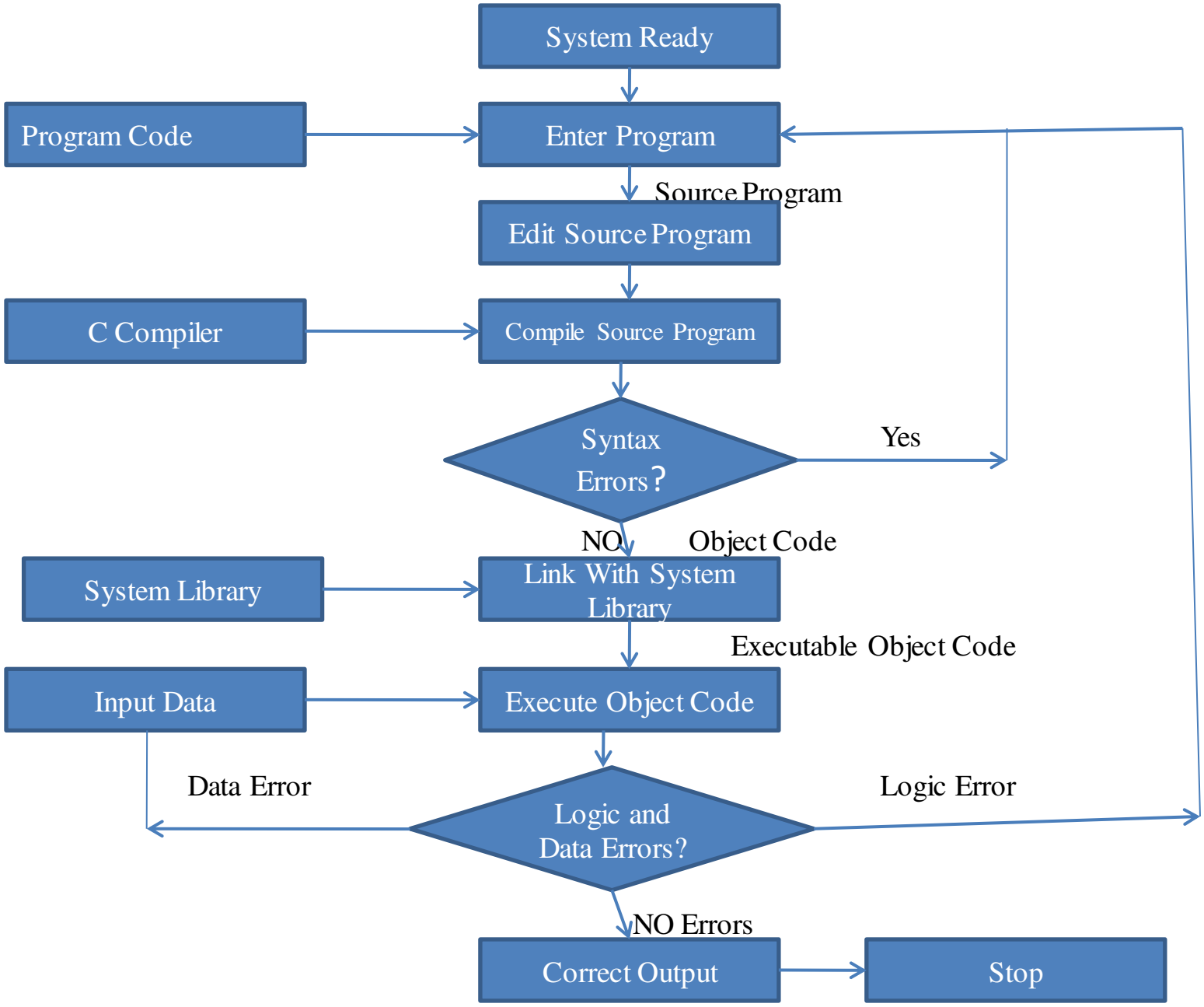
User-defined functions are generally placed immediately after the main function , although they may appear in any order.

EXECUTING A 'C' PROGRAM

Steps

1. Creating the program.
2. Compiling the program.
3. Linking the program with functions that are needed from the C library
4. Executing the program.

1.5 Process of Compiling and Running a C program



1.6 WRITING AND EXECUTING A C PROGRAM IN UNIX

Create a c program

- write the c-program using some text editors (eg. vi or ed)
  - **ed filename**
  - **vi filename**
- The program must be entered into a file.
- The file name can consists of letters, digits and special character, followed by dot(.) and a letter c.

Example

firstcpgm.c

- If the file existed before, it is loaded.
- Otherwise the file has to be created so that it is ready to receive the new program.
- The program that is entered into the file is called source program.

Compiling and linking

- use an existing compiler (gcc, cc etc) to compile your program an make an executable

Example

- **cc *filename***
- **cc *filename1, filename2, filename3 ……***

Common options:

- **–lm : library math**
- **-c : Compiles without linking. binaries are saved as "*filename.o"***
- **-o *exename: compiled binary with specified filename. (a.out default)***
- **cc –o *myprog firstprog.c***

Execute the program

- **a.out**

UNIT I

CHAPTER II

2.1. CHARACTER SET

The characters that can be used to form words, numbers and expressions.

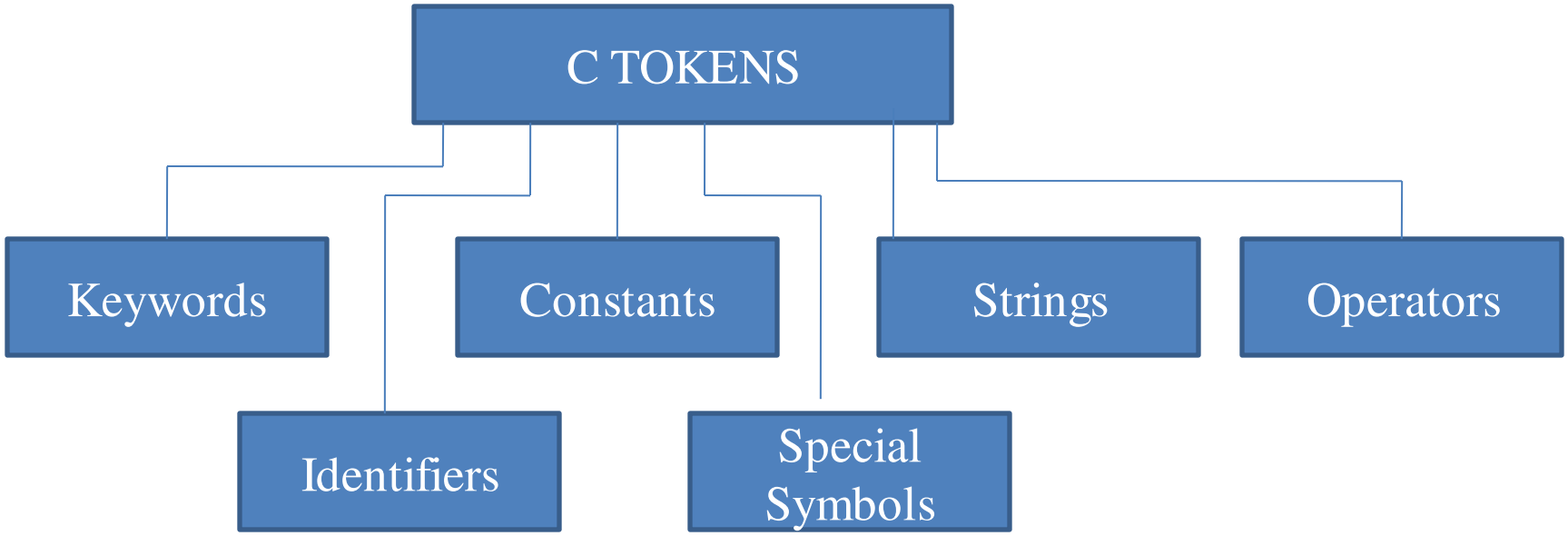
Letters	Digits
Upper case :A---Z, Lower case : a--z	0---9
Special Characters	
, (comma) .(period) ;(semicolon)	& (ampersand) ^ (caret)
: (colon) ? (question mark)	•(asterisk) - (minus sign)
‘ (apostrophe) “ (quotation mark)	+ (plus sign) < (less than sign)
! (exclamation mark) | (vertical bar)	> (greater than sign) ((left parenthesis)
/ (slash) \ (back slash)	) (right parenthesis) [(left bracket)
~ (tilde) _ (under score)	] (right bracket) # (number sign)
$ (dollar sign) % (percent sign)	{ (left brace) } (right brace)
White spaces	
Bank space, Horizontal tab	Carriage return, New line Form feed

2.1.1 TRIGRAPH CHARACTERS

Trigraph Sequence	Translation
??=	#
??(	[
??)	]
??<	{
??>	}
??!	|
??\	\
??/	^
??-	~

2.2. C TOKENS

- Individual words and punctuation marks are called tokens.
- Smallest individual units are known as C tokens.



2.2.1 KEYWORDS AND IDENTIFIERS

- All keywords have fixed meaning and these meanings cannot be changes.
- Keywords serve as basic building blocks for program statements.
- There are certain words reserved for doing specific task, these words
- are known as **reserved word or keywords.**
- **These words are predefined and always** written in lower case or small letter.
- These keywords cannot be used as a variable name as it assigned with fixed meaning.

ANSI C Keywords

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

IDENTIFIERS

Identifiers are user defined word used to name of entities like variables, arrays, functions, structures etc.

Rules for naming identifiers are:

1) **Must consists of alphabets (both upper and lower case), digits** and **underscore (_)** sign.
2) **First characters must be an alphabet or underscore.**
3) **Only first 31 characters are significant.**
4) **Cannot use a keyword.**
5) **Must not contain whitesoace.**

C is a case sensitive, the upper case and lower case considered

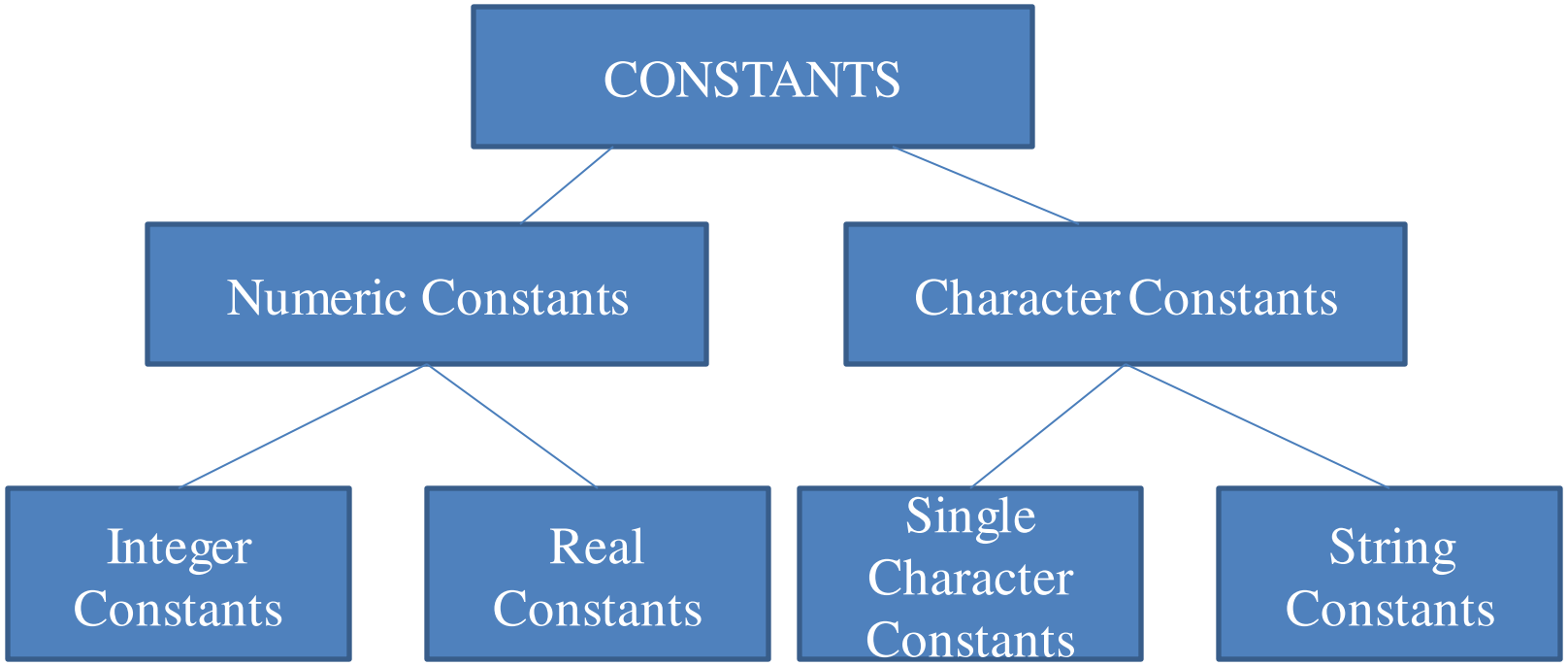
differently, for example code, Code, CODE etc. are different identifiers.

identifiers are generally given in some meaningful name such as value, net_salary, age, data etc.

Some invalid identifiers are 5cb, int, res#, avg no etc.

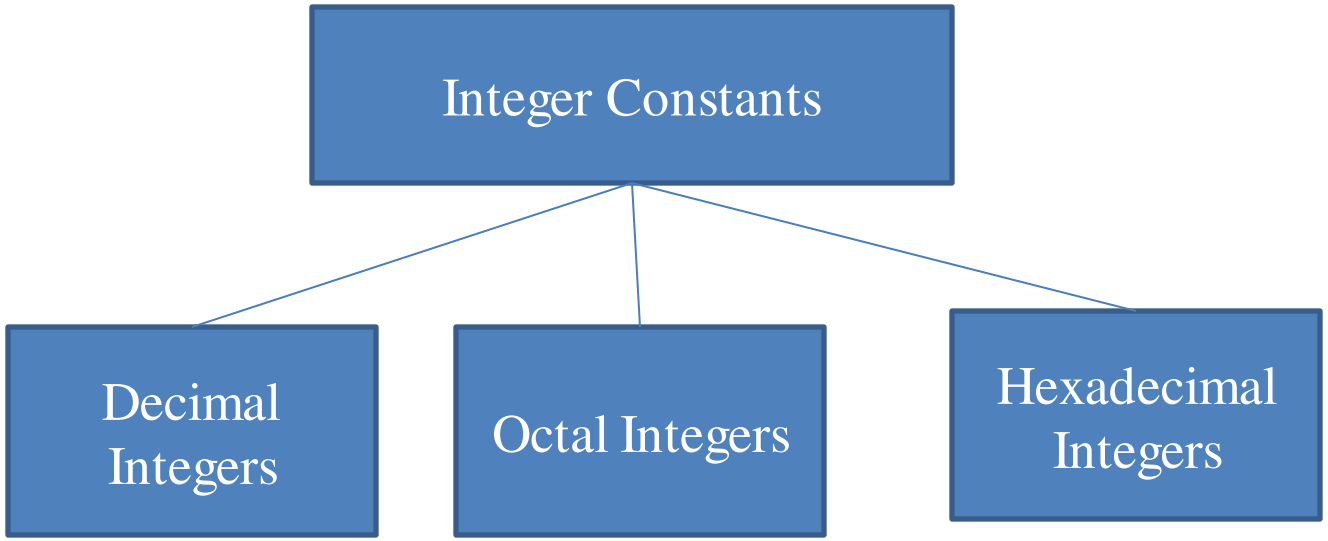
2.2.2 CONSTANTS

Fixed values that do not change during the execution of a program.



Integer Constants

An integer constant refers to a sequence of digits.



Consists of a set of digits, 0 through 9, preceded by an optional – or + sign.
Example: 123, -123,0, +73

Consists of any combination of digits from 0 to 7,while leading 0.
Example:
037, 0, 0437

A sequence of digits preceded by 0x or 0X. They may include Alphabet A through F. The letter A through F represents 0 through 15.
Example:
0X2,ox9F,oxABCD

Real Constants

These numbers are decimal notation,havinfg whole number followed by a decimal point and the fractional part.

It is possible to omit digits before the decimal part, or digits after the decimal part.

A real number may expressed in ***exponential*** notation.

mantissa e exponent

The mantissa is either a real number expressed in decimal notation or integer.

The exponent is an integer number with an optional + or – sign.

The letter e can be written in either uppercase or lowercase.

Embedded white space is not allowed.

The e notation is called floating-point form.

Floating –point constants are represented as double-precision quantities.

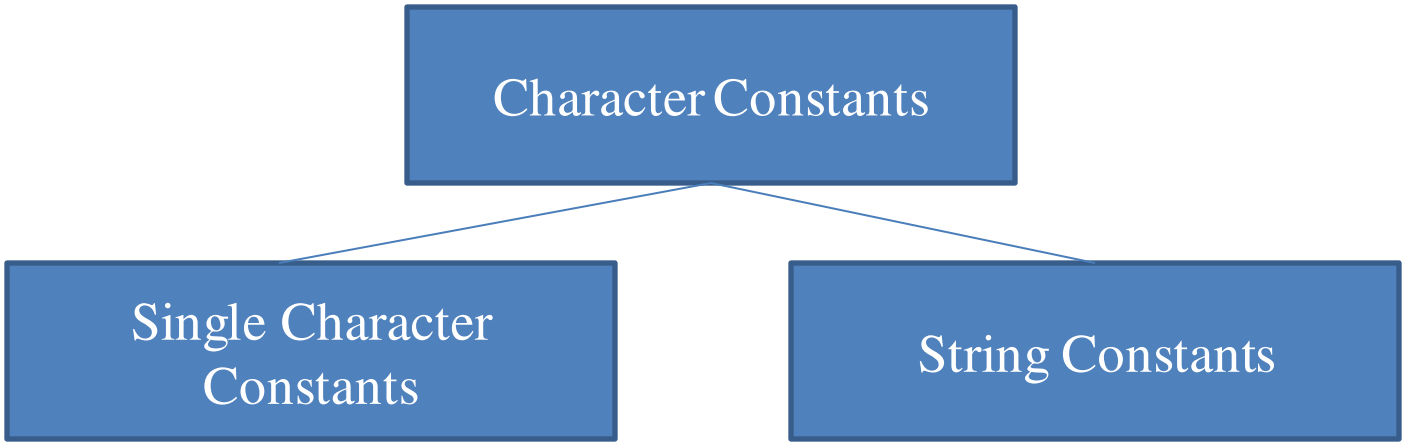
f or F --- Single-precision

l or L --- Double-precision

Valid real numbers: 78688L,25636l,444.643f,321.55F,2.5e+05,374e-04

Invalid real numbers: 1.5E+0.5,$25,ox7B,7.5 e -0.5,1,00,00.00

Character Constants



single character enclosed within a pair of single quote.
Example:
‘5’, ‘ a’, ‘ ’

A sequence of characters Enclosed in double quotes. The characters may be letters numbers, special characters and blank spaces.
Example:
“Hello World!”, “25”, “5”, “3+5”

Note: 5 ≠ ‘5’ ≠ “ 5” → 5 –Integer Constant ‘5’ – single Character Constant
“5”- String Contant

Backslash character constants

- used in output functions
- each one of them represents one character.
- these characters combinations are known as escape sequences

Character	Meaning
'\a'	alert (bell)
'\b'	backspace
'\f'	form feed
'\n'	newline
'\r'	carriage return
'\t'	horizontal tab
'\v'	vertical tab
'\\ '	backslash
'\''	single quote
'\"'	double quote
'\?'	question mark

SYMBOLIC CONSTANTS

Symbolic constant is defined as follows

#define symbolic_name value-of-constant

Example:

```
#define PI 3.14159
#define MAXMARKS 100
```

Rules

1) Symbolic names have the same form as variable names.
2) No blank space between the # sign define is permitted.
3) # must be the first character in the line.
4) A blank space is required between #define and symbolic name and between the symbolic name and constant.
5) #define statements must not end with a semicolon.
6) After definition, the symbolic name should not be assigned any other value within the program by using an assignment statement.
7) Symbolic names are not declared for data types. Its data type depends on the type of constant.
8) #define statements may appear anywhere in the program but before it is referenced in the program.

2.2 VARIABLES

Variable is a data name which is used to store data value .

The value of the variable can be change during the execution.

The rule for naming the variables is same as the naming identifier.

Rules :

1) They begin with a letter. Some systems permit underscore as the first character.
2) Upto 31 characters (ANSI)
3) Case Sensitive ie. Uppercase and lowercase are significant. A ≠ a
4) It should not be a keyword
5) White space is not allowed.

Examples:

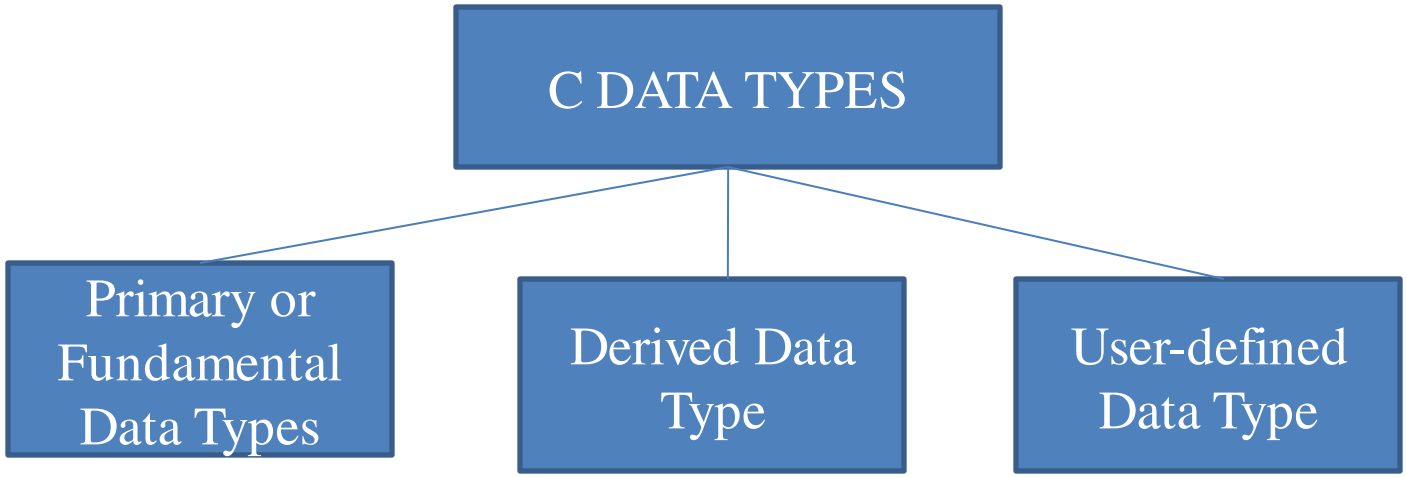
Valid variable names: sum, Sum, SUM, a1,odd_sum

Invalid Variable names: 1a,odd-sum,a%,1st,(area)

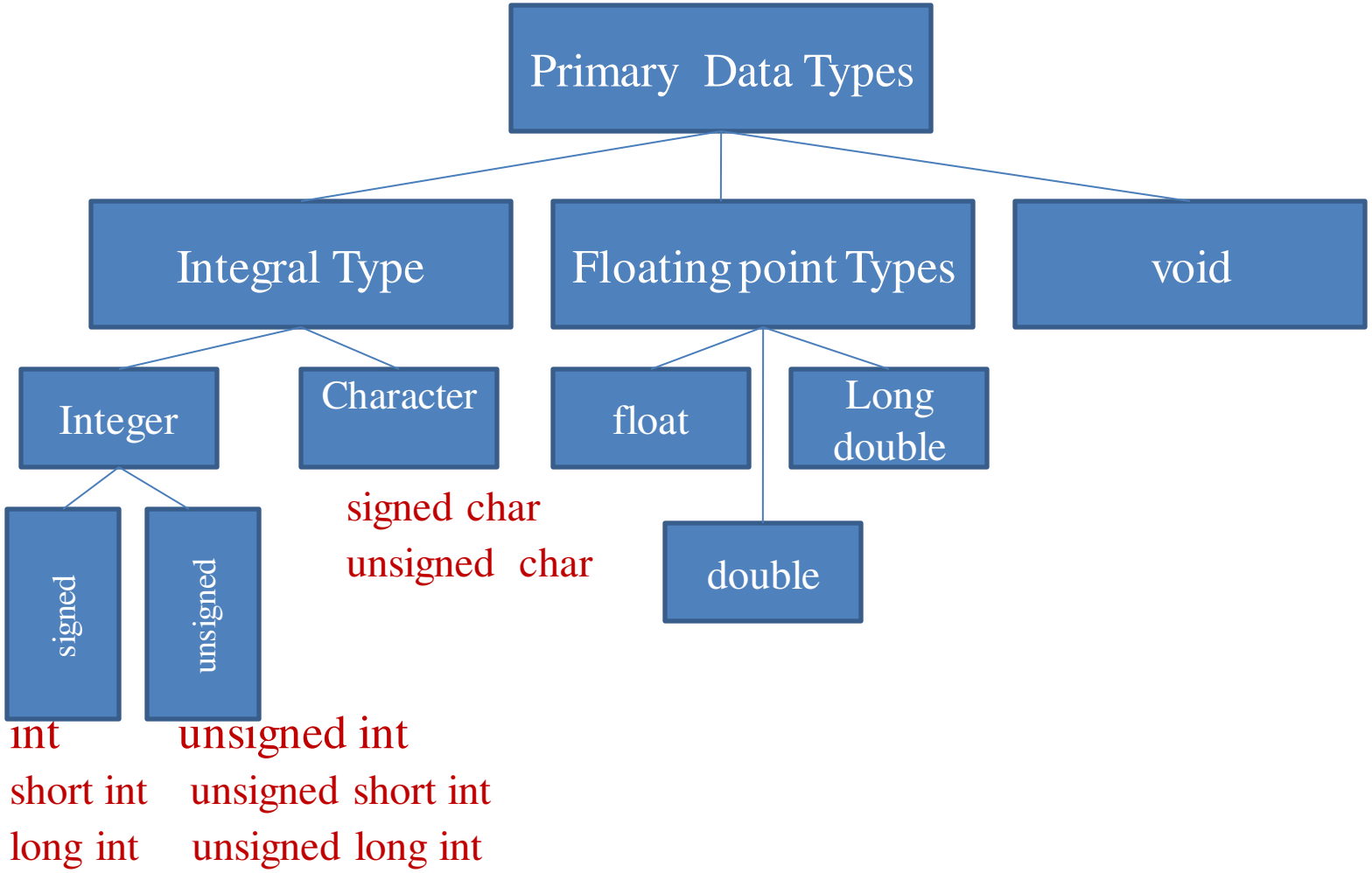
2.3 DATA TYPES

Data types refer to an extensive system used for declaring variables or functions of different types before its use.

The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.



PRIMARY OR FUNDAMENTAL DATA TYPES

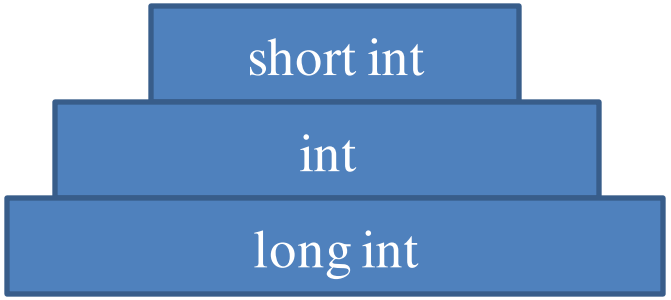


INTEGER TYPES

Integer occupy one word of storage, and word sizes of machines vary (16 or 32 bits).

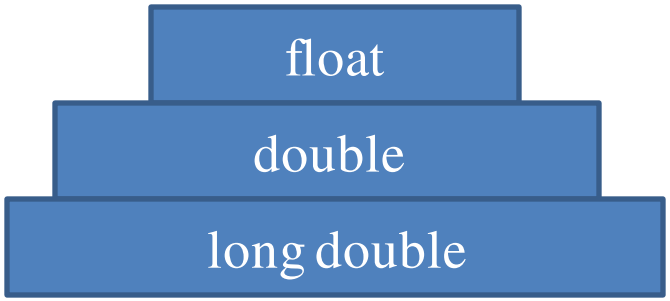
The signed integer uses one bit for sign and 15 bits for the magnitude of the number.

Smallest to largest.



FLOATING POINT TYPE

- Floating point numbers are stored in 32-bits, with 6 digits of precision.



CHARACTER TYPE

- A single character can be defined as a character (char) type data.
- Characters are stored in 8 bits of internal storage.
- The qualifier signed or unsigned may be explicitly applied to char.

VOID TYPE

- has no values.
- Used to specify the type of functions.
- The type of function is said to be void when it does not return any value to the calling function.
- Play the role of a generic type, ie it can represent any of the other stamdard type.

TYPE	KEYWORD	SIZE (BITS)	RANGE
Character	char or signed char	8	-128 to 127
	unsigned char	8	0 to 255
Integer	int or signed int	16	-32,768 to 32767
	unsigned int	16	0 to 65535
	short int or signed short int	8	-128 to 127
	unsigned short int	8	0 to 255
	long int or signed long int	32	-2,147,483,648 to 2,147,483,647
	unsigned long int	32	0 to 4,294,967,295
Real	float	32	3.4E-38 to 3.4E+38
	double	64	1.7E-308 to 1.7E+308
	long double	80	3.4E-4932 to 1.1E+4932

2.4 DECLARATION OF VARIABLES

Declaration does two things

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will hold.

Primary type declaration

A variable can be used to store a value of any data type.

data_type v1,v2,v3,….,vn;

v1,v2,v3,…vn are the name of the variables.

Variables are separated by commas.

Declaration statement end with a semicolon.

Example:

int count;

float total;

double ratio;

User-defined type declaration

1. "type definition" – allows users to define an identifier that would represent an existing data type.

The user-defined data type identifier can be used to declare variables later.

typedef type identifier;

Where **type** refers to an existing data type,

identifier refers to the new name given to the data type.

The existing data type may belong to any class of type, including the user-defined type.

Example:

```
typedef int units;
typedef float marks;
```

Here, units symbolizes int and marks symbolizes float.

```
units batch1,batch2;
marks name[10];
```

Advantage of typedef

We can create meaningful data type names for increasing the readability of the program.

2. “enumerated”

Enum identifier (value1,value2,…..,valuen);

The ‘identifier’ is a user-defined enumerated data type which can be used to declare variables

That can have one of the values enclosed within the braces – enumerated constants.

enum identifier v1,v2,…...,vn;

The enumerated variables v1,v2,…..,vn can have one of the values value1,value2,…..,valuen.

V1=value3;

V3=value5;

Example:

Enum day(Moday,Tuesday,…...,Sunday);

Enum day week_st,week_end;

Week_st=Monday;

Week_end=Friday;

2.5 DECLARATION OF STORAGE CLASS

Storage class provides the information about their location and visibility.

The storage class decides the portion of the program within which the variables are recognized.

i. Global variables:

Global variables are known throughout the program.

The variables hold their values throughout the programs execution.

Global variables are created by declaring them outside of any function.

It need not be declared in other function.

A global variable is also known as ***external variable***.

Global variables are defined above main () in the following way:

```
int n, sum;
int m,l;
char letter;
main()
{
}
```

it is also possible to pre-initialize global variables using the = operator for assignment.

Example:

```
float sum = 0.0;
int n= 0;
char c=`a';
main()
{
}
```

This is the same as:

```
float sum;
int n;
char letter;
main()
{
sum = 0.0;
n= 0;
c=`a';
}
```

is more efficient.

C also allows multiple assignment statements using =

Example:

a = b = 1;

This is same as

a = 1;

b = 1;

Note :

The assignment is valid if all the variable types are the same data type.

ii. **Local variables:**

Variable that are declared inside a function are called "local variables".

Local variables are referred to as ***"automatic variables***".

Local variables may be referenced only by statements that are inside the block in which the variables are declared.

Local variables exist only while the block of code in which they are declared is executing.

Local variables are not known to other function block.

Any changes are made in local variables does not affect its value in the other.

Example:

```
void func1 (void)
{
Int n;
n = 0;
}
void func2 (void)
{
int n;
n = 99;
}
```

The integer variable n is declared in func1 () & func2 ().

n is only known to the code with in the same block as variable declaration.

Storage class

There are four types of storage class specifiers

1. ***auto***
2. ***register***
3. ***static***
4. ***extern***

Storage class	meaning	Automatic initialization
auto	Local variable known to the function in which it is declared. Default is auto.	Garbage value (undefined value)
register	Local variable which is stored in the register	--
Static	Local variable which exists and retains its value even after the control is transferred to the calling function.	0
extern	Global variable known to all function in the file	0

Declaring a variable as constant

The value of the variable to remain constant during the program execution.

The variable can be declared with the qualifier ***const*** at the time of initialization.

const data_type variable_name=constant;

Example

const int n=50;

The value of n cannot be modified.

2.6 ASSIGNING VALUE TO VARIABLES

Variablename = constant;

Example:

```
n=5;
count=count+1;
```

datatype variable_name=constant;

Example:

```
int n=5;
float x=5.5;
```

CHAPTER III
OPERATORS AND EXPRESSIONS

3.1. INTRODUCTION

- A symbol use to perform some operation on variables, operands or with the constant is known as ***operator***.
- Some operator required 2 operand or Some required single operand to perform operation.
- C operators can be classified into a number of categories.

1) Arithmetic operators
2) Relational operators
3) Logical operators
4) Assignment operators
5) Increment and decrement operators
6) Conditional operators
7) Bitwise operators
8) Other operators

An ***expression*** is a sequence of operands and operators that reduces to a single value.

The value can be any type other than void.

3.1.1. ARITHMETIC OPERATORS

Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

Expression	values	Integer Arithmetic	Real Arithmetic
a+b	a=10, b=3	13	13.0
a-b	a=10, b=3	7	7.0
a*b	a=10, b=3	30	30.0
a/b	a=10, b=3	3	3.3333333
	a=15, b=3.0	5(lhs int)	5.0
a%b	a=10, b=3	1	Cannot be used with real operands
	a=-14, b=3	-2	
	a=-14, b=-3	-2	
	a=14, b=-3	2	

3.1.2 RELATION OPERATORS

Relation operators and meaning

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

Syntax

ae-1 relational operator ae-2

Examples

Relational expression	**Result**
4.5<10	true
4.5<=10	true
4.5 ==10	false
4.5 !=10	true
4.5 >10	false
4.5>=10	false

3.1.3 LOGICAL OPERATORS

Logical Operators and Meaning

Operators	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Syntax

Op-1 logicaloperator op-2

Truth Table

Op-1	Op-2	Value of the Expression	
		Op-1&&op-2	Op-1||op-2
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

3.1.4 ASSIGNMENT OPERATORS

Assignment Operators and Meaning

- Assignment operators are used to assign the result of an expression to a variables.

Syntax

v op=exp;

Where v is a variable, exp is an expression and op is a binary arithmetic operator.

The assignment statement

v op=exp;

is equivalent to

v=v op (exp);

Advantages

1. What appears on the left-hand side need not be repeated and therefore it becomes easier to write.
2. The statement is more concise and easier to read.
3. The statement is more efficient.

Operators and Examples

Operators	Example	Equivalent
+=	a+=1	a=a+1
-=	a-=1	a=a-1
*=	a*=n+1	a=a*(n+1)
/=	a*=n+1	a=a/(n+1)
%=	a%=b	a=a%b

3.1.5 INCREMENT & DECREMENT OPERATORS

operators

1. Pre-increment & decrement-prefix ++ and --
2. Post-increment & decrement-postfix ++ and --

- Prefix operator adds 1 to the operand and then the result is assigned to the variable on left.
- Postfix operator first assigns the value to the variable on left and then increments the operator.

Rules

1. Increment and decrement operators are unary operators.
2. When postfix ++ (or --) is used with a variable in an expression, the expression is evaluated first using the original value of the variable and then the variable is Increment (or decrement) by one.
3. When prefix ++ (or decrement) is used in an expression, the variable is incremented (or decremented) first and then the expression is evaluated using the new value of the variable.
4. The precedence and associatively of ++ and – operators are same as those of unary + and unary -.

Examples

m=5;

operator	Expression	Result	M value
Prefix ++	y=++m	y=6	m=6
Prefix --	y=--m	y=4	m=4
Postfix ++	y=m++	y=5	m=6
Postfix --	y=m--	y=5	m=4

3.1.6 CONDITIONAL OPERATOR

Operator and syntax

- A ternary operator pair "?:" is called conditional operator.

Syntax

exp1?exp2:exp3;

Where exp1, exp2 and exp3 are expression.

- Exp1 is evaluated first.
- If it is true(nonzero) then the expression exp2 is evaluated and becomes the value of the expression.
- Otherwise exp3 is evaluated and its value becomes the value of the expression.

Example

```
a=10;
b=15;
x=(a>b)?a:b;
```

output

x=10

The above code is equivalent to

```
if (a>b)
   x=a;
else
   y=b;
```

3.1.7 BITWISE OPERATORS

Bitwise operators

- Bitwise operators are used for manipulating data at bit level.
- These operators are used for testing the bits, or shifting them right or left.
- Bitwise operators may not be applied to float or double.

Operators	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

3.1.8 SPECIAL OPERATORS

Special operators are

1. comma operator (,)
2. sizeof operator
3. pointer operators (& and *)
4. member selection operators(.,->)

Comma operator

- Comma operator can be used to link the related expression together.
- A comma-linked list of expressions are evaluated left to right and the value of right-most expression is the value of the combined expression.

Example:

z=(x=10, y=15, x+y);

- First assigns the value 10 to x, then 15 to y and finally 25 to z.
- Comma operator has the lowest precedence of all operators.

Exchanging values:

t=x, x=y, y=t;

sizeof opeartor

- The sizeof is a compile time operator and when used with an operand, it returns the number of bytes the operand occupies.
- The operand may be a variable, a constant or a data type qualifier.

Examples:

m=sizeof(sum);
n=sizeof(long int);
k=sizeof(235l);

- The sizeof operator normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer.
- It is also allocate memory space dynamically to variables during program execution.

3.2 ARITHMETIC EXPRESSION

- An arithmetic expression is a combination of variables, constants and operators arranged as per the syntax of the language.
- C does not have the operator for exponentiation.

Algebraic expression	C expression
$\frac{ab}{c}$	a*b /c
a x b-c	a*b-c
(m+n)(x+y)	(m+n)*(x+y)
$3x^2+2x+1$	3*x*x+2*x+1
$\frac{a}{b}+c$	a/b+c

Evaluation of expressions

- Expressions are evaluated using an assignment statement of the form

Variable=expression;

- Variable is any valid c variable name.
- When the statement is encountered, the expression is evaluated first and the result then replaces the previous value of the variable on the left hand side.
- All the variables used in the expression must be assigned values before evaluation is attempted.

Example:

x = a * b – c;

Rules for evaluation of expressions.

- First, Parenthesized sub expression from left to right are evaluated.
- If parentheses are nested, the evaluations begins with the innermost sub expression.
- The precedence rule is applied in determining the order of application of operators in evaluating sub expression.
- The associative rule is applied when two or more operators of the same precedence level appear in a sub-expression.
- Arithmetic expressions are evaluated from left to right using the rules of precedence.
- When parentheses are used, the expressions within the parentheses assume highest priority.

Priority levels

High priority → * / %

Low priority → + -

3.3 OPERATOR PRECEDENCE AND ASSOCIATIVITY

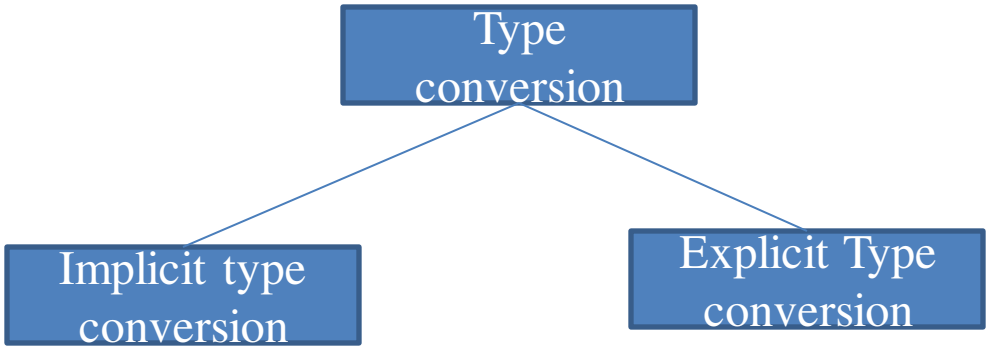
Operator	Description	Associativity	Rank
() []	function call and array element reference	left to right	1
+ - ++ -- ! ~ * & sizeof (type)	unary plus ,unary minus, increment, decrement, logical negation, ones complement, pointer reference, address, size of an object and type cast	right to left	2
* / %	multiplication, division and modulo division	left to right	3
+ -	addition and subtraction	left to right	4
<< >>	left shift and right shift	left to right	5
< <= > >=	less than, less than or equal to, greater than and greater than or equal to	left to right	6
== !=	equal to and not equal to	left to right	7
&	bitwise and	left to right	8
^	bitwise xor	left to right	9
|	bitwise or	left to right	10

Operator	Description	Associativity	rank
&&	Logical AND	Left to right	11
||	Logical OR	Left to right	12
?:	Conditional expressions	right to Left	13
= *= /= %= += -= &= ^= |= <<= >>=	Assignment operations	right to Left	14
,	Comma operator	Left to right	15

Rules of precedence and associativity

- Precedence rules decides the order in which different operators are applied
- Associative rule decides the order in which multiple occurrences of the same level operator are applied.

3.4. TYPE CONVERSION



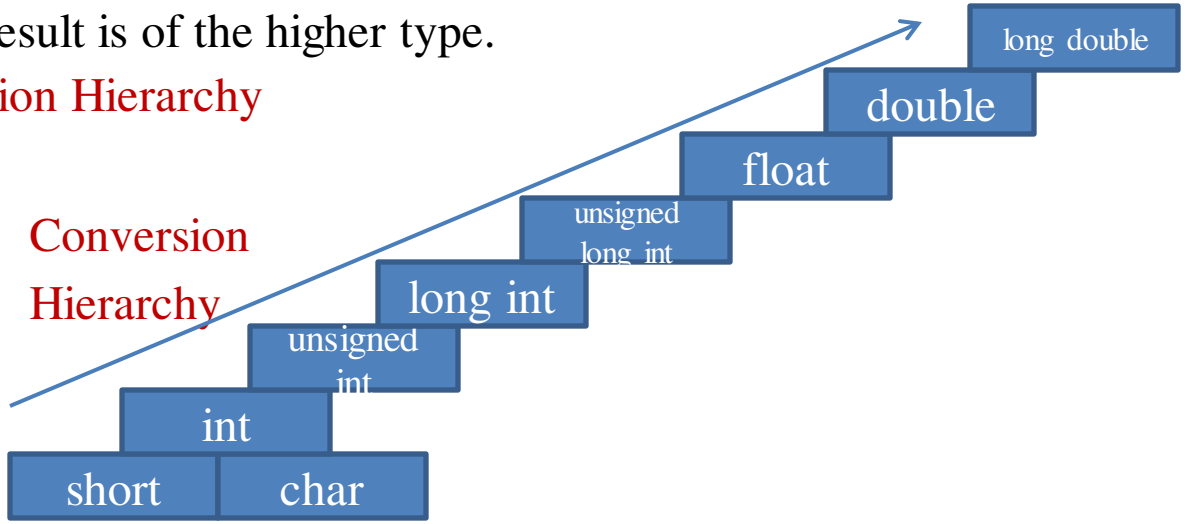
3.4.1 Implicit Type Conversion

Automatic type conversion is called implicit type conversion.

If the operands are of different types, the lower type is automatically converted to the higher type before the operations proceeds.

The result is of the higher type.

Conversion Hierarchy



3.4.2 Explicit Type Conversion

- The process of local conversion is called explicit type conversion or casting a value.

Syntax

(type-name)expression

- Where type-name is one of the standard c data types.
- The expression may be a constant , variable or an expression.

Example:

x=(int) 7.5; →7.5 converted to integer by truncation

a=(int)21.3/(int)4.5;→ evaluated as 21/4 and the result would be 5

b=(double) sum/n; → divison is done in floating point mode.

y=(int)(a+b); → the result of a+b is converted to integer.

z=(int)a+b; →a is converted to integer and then added to b

3.5 MATHEMATICAL FUNCTION

FUNCTION	MEANING
acos(x)	Arc cosine of x
asin(x)	Arc sine of x
atan(x)	Arc tangent of x
atan2(x,y)	Arc tangent of x/y
cos(x)	Cosine of x
sin(x)	Sine of x
tan(x)	Tangent of
cosh(x)	Hyperbolic cosine of x
sinh(x)	Hyperbolic sine of x
tanh(x)	Hyperbolic tangent of x
ceil(x)	X runded up to the nearest integer
floor(x)	X rounded down to the nearest integer
pow(x,y)	X to the power of y
sqrt(x)	Square root of x, x>=0

REFRENCES

- E. Balagurusamy, ”Programming in ANSI C”, Seventh Edition, McGraw Hill Education India Private Ltd, 2017.
- https://image.slidesharecdn.com/1-overviewandhistoryofc