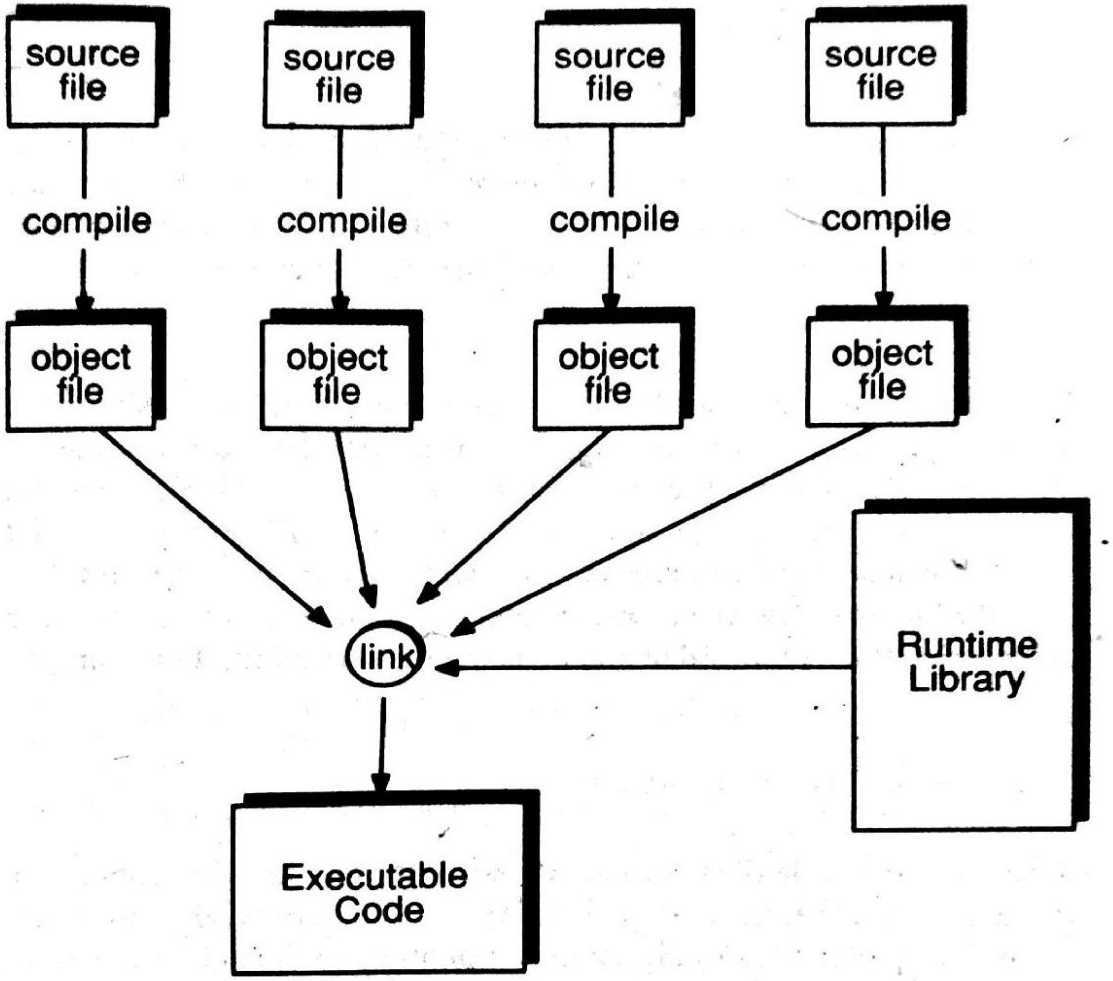
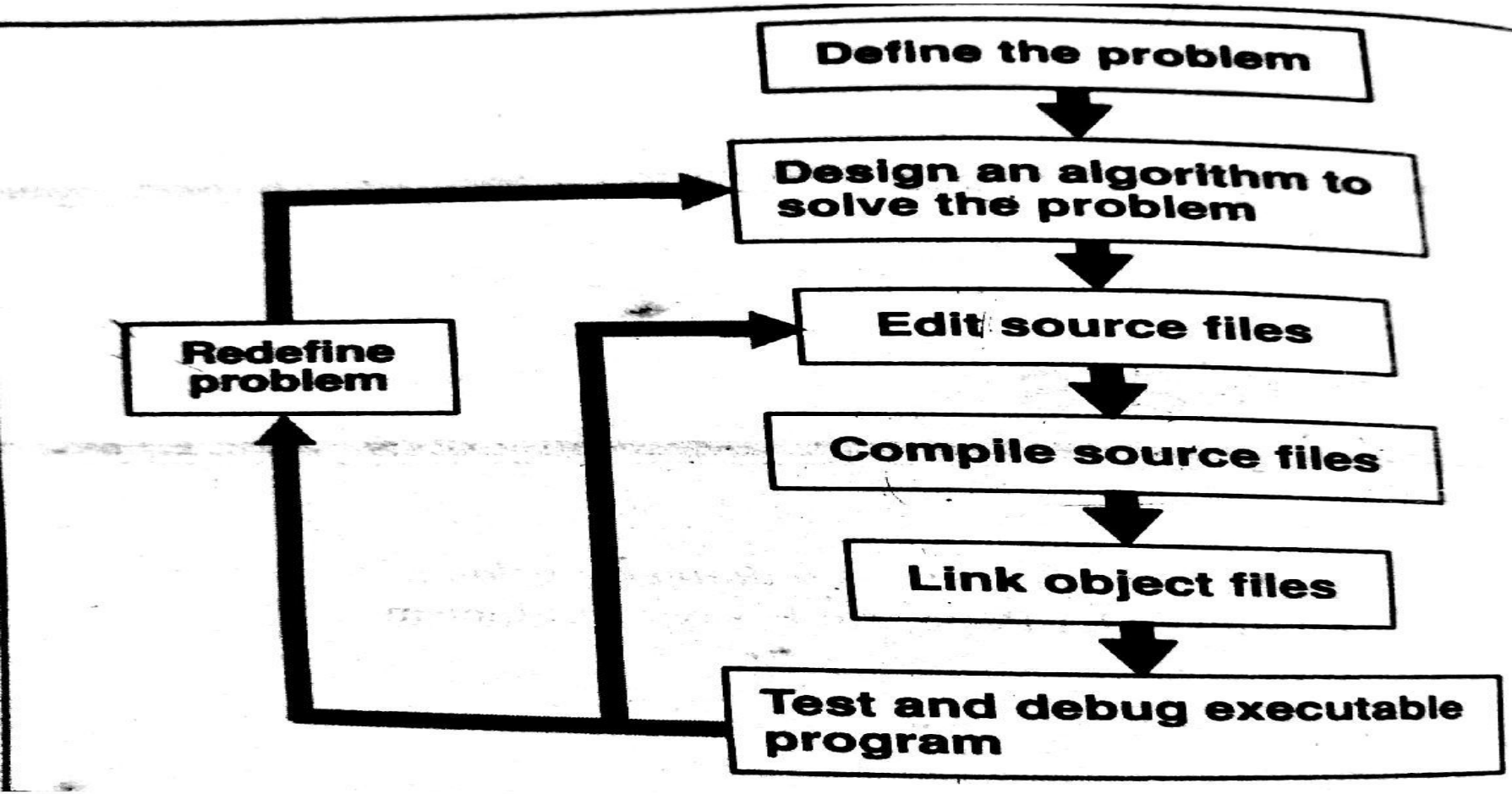
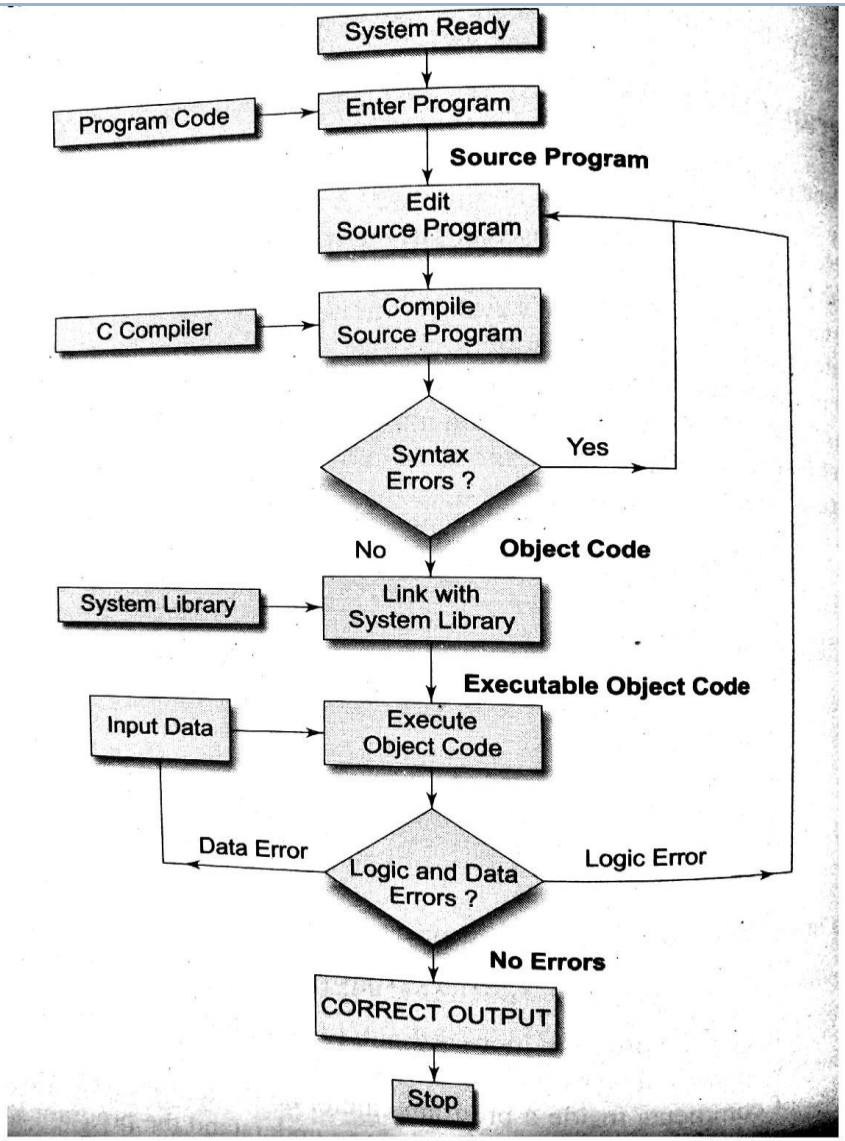
COMPUTER PROGRAMMING AND UTILISATION
UNIT 2 – FUNDAMENTALS OF C

Compile and Linking





Compiling and Running of C Program



Classification of Computer Languages

- The computer languages can be classified into following 3 different categories.

  Machine Language, Assembly Language and High Level language.

- Machine Language
  - The processor understands only the binary language (language of 0s and 1s).
  - The advantage in writing machine language is that no translator is required because it is written in the language the processor directly understands.
  - It is very cumbersome to write the machine language program.

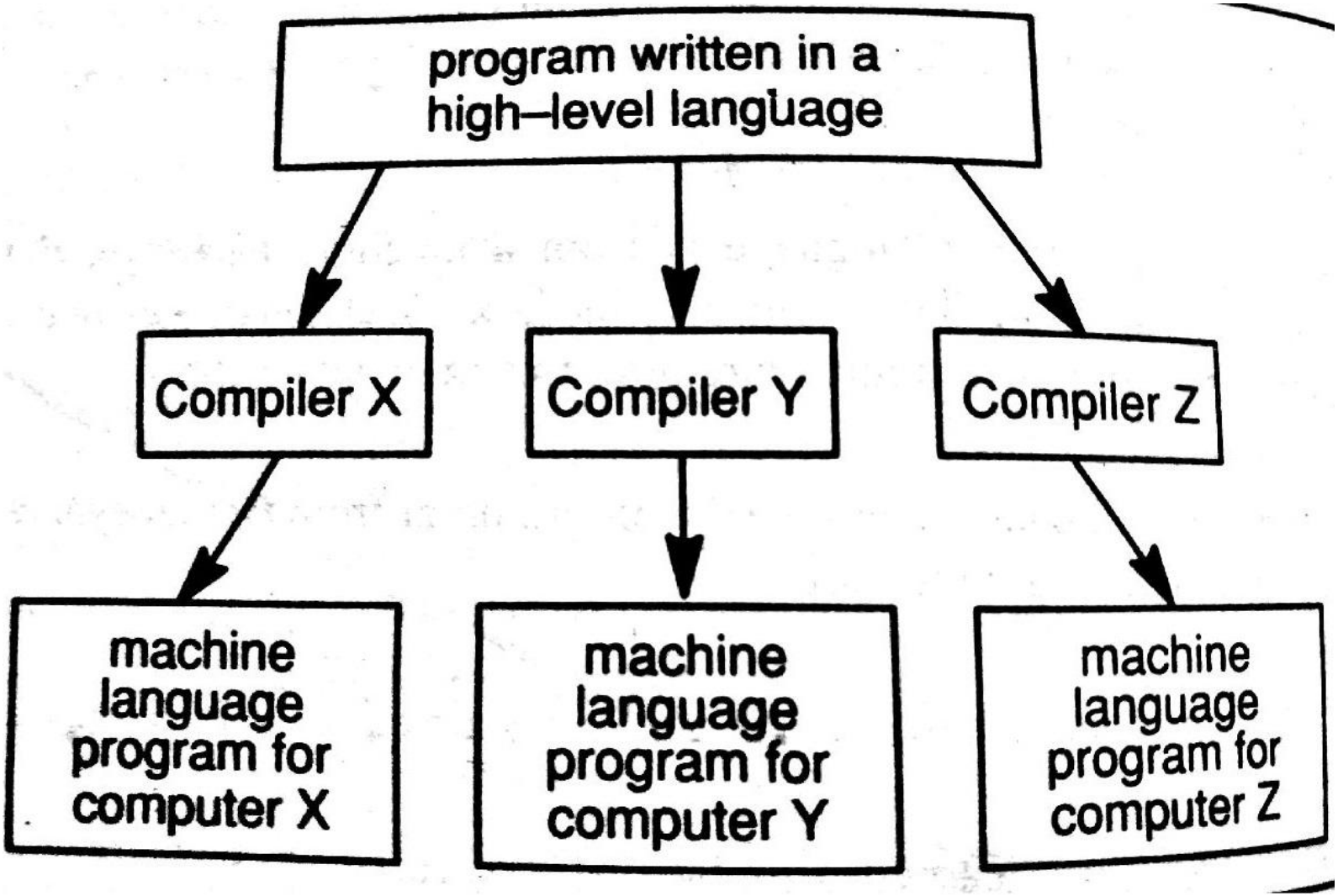
Classification of Computer Languages (Cont)

- Assembly Language
  - It makes use of **Mnemonics Operation Code** and uses symbolic addresses,
  - The advantage of using assembly language is it is easy for the programmer to remember Mnemonic Op-code,
  - the translator is required to covert assembly language into machine language.

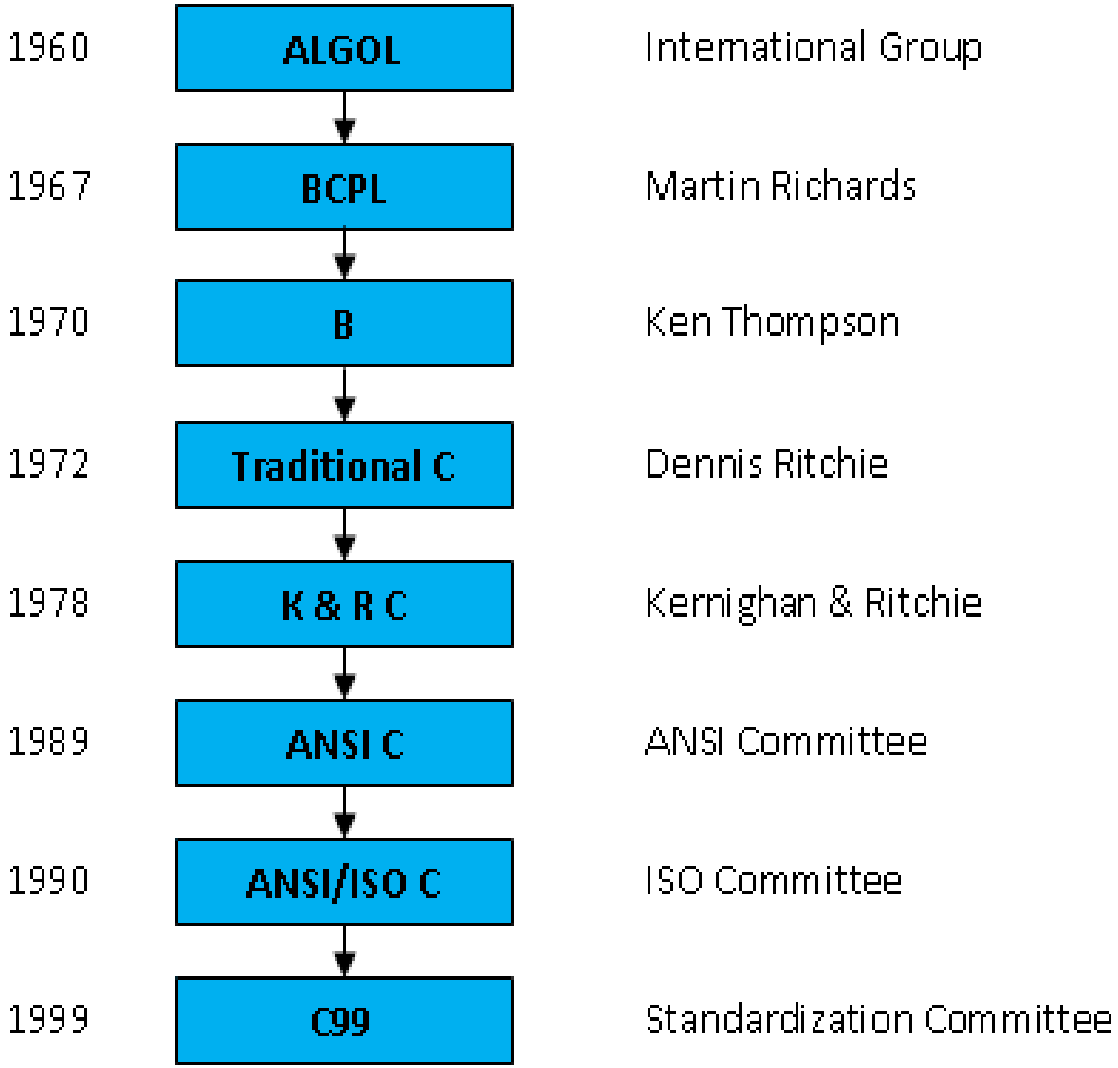
Classification of Computer Languages (Cont)

- High Level Language
  - It enable the programmer to concentrate more on the logic part and not on the architectural details of the processor.
  - The advantages are it is easy to learn and use, the program is machine independent, and efficiency of programmer is improved, easy maintenance of program.
  - Examples of high level languages are Basic, COBOL, Pascal, FORTRAN, C, C++, Java etc.
  - The disadvantage is that it requires a special converter (known as **compiler**) which will convert high level language into machine language.

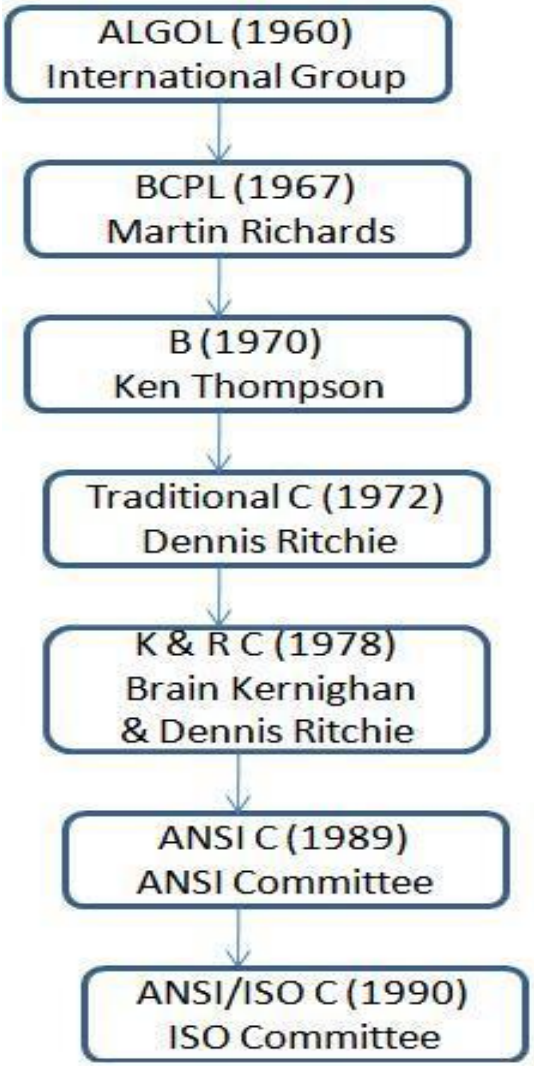
Different Compilers for Different Machines



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 & Ritchie
1989
ANSI C
ANSI Committee
1990
ANSI/ISO C
ISO Committee
1999
C99
Standardization Committee

History of C


ALGOL (1960)
International Group
BCPL (1967)
Martin Richards
B (1970)
Ken Thompson
Traditional C (1972)
Dennis Ritchie
K & R C (1978)
Brain Kernighan
& Dennis Ritchie
ANSI C (1989)
ANSI Committee
ANSI/ISO C (1990)
ISO Committee

Features of ‘C’

- Important feature of ‘C’ language is it is **portable,** by portability we mean that the program can be run on any hardware machine.
- It is a **robust** language, by robust we mean any complex program can be written with rich set of built-in functions and operators.
- It is **efficient** and **fast.**
- It supports **modular** programming so It is known as structured programming language.
- It supports bit-wise operations.

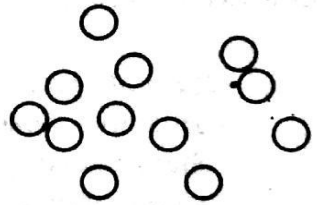
Basic Structure of C Program is as below,	Example of C Program is as below,
Documentation Section	// This program is to find area of circle
Link Section	#include<stdio.h>
Definition Section	#define PI 3.14
Global Declaration Section	int i; float areaofcirde(float);
main() {	void main() {
Declaration Part	//Dedaration Part float r,area;

Executable Part	//Executable Part
	scanf("%f",&r);
	area=areaofcircle(r);
	printf("Area of Circle is :- %f",area);

}	}
Subprogram Section (User Defined Section)	float areaofcirde(float r)
Function1()	{
Function2()	return PI * r * r;
	}

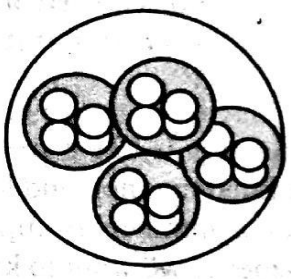
Software Hierarchy

Machine Instructions: At the lowest level, every program consists of primitive machine instructions.

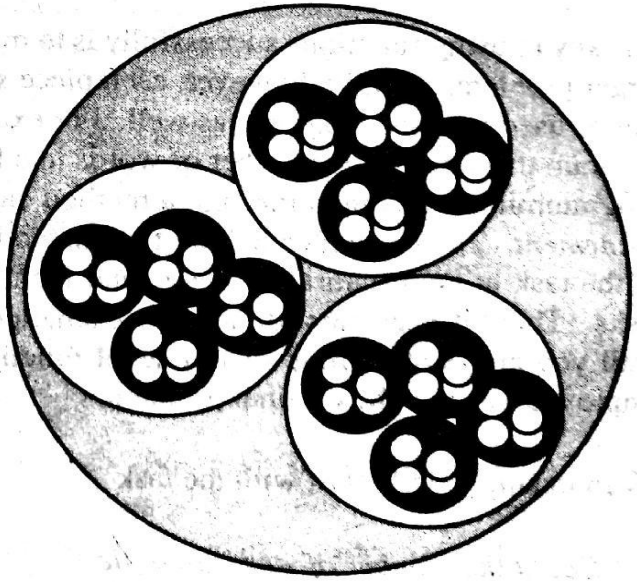


Language Statements: High–level languages consist of statements that perform one or more machine instructions.

Functions: Functions consist of groups of language statements.



Programs: Programs consist of groups of functions.



COMPUTER PROGRAMMING AND UTILISATION
UNIT 2 – INTRODUCTION TO C LANGUAGE

Character Set

- 'C' program basically consists of keywords, identifiers, constants, operators and some special symbols.
- The characters that can be used in a 'C' program are **Alphabets** (A-Z and a-z), **Digits** (0-9), **Special Characters** and **White Space Characters.**

Character Set

Letters

Uppercase A.....Z
Lowercase a.....z

Digits

All decimal digits 0.....9

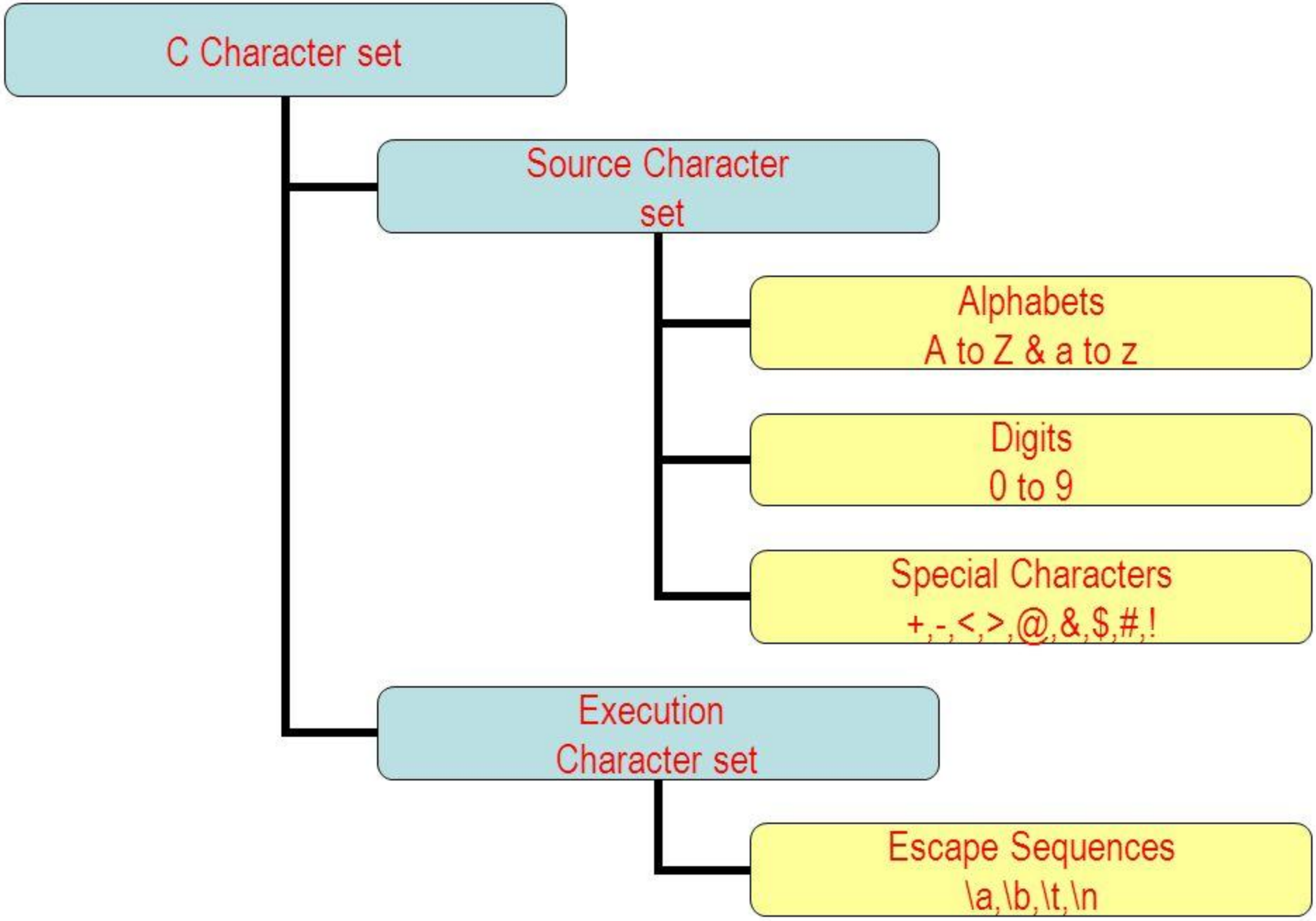
Special Characters

, comma	$ dollar sign	(left parenthesis
. period	% percent sign	(or less than sign)
:semicolon	& ampersand	)right parenthesis
: colon	^ caret	(or greater than sign)
? question mark	* asterisk	[left parenthesis
! exclamation mark	- minus sign	] right parenthesis
| vertical bar	+ plus sign	{ left brace
/ slash	< opening angle bracket	} right brace
\ backslash	> closing angle bracket	#number sign
~ tilde		
_ under score		

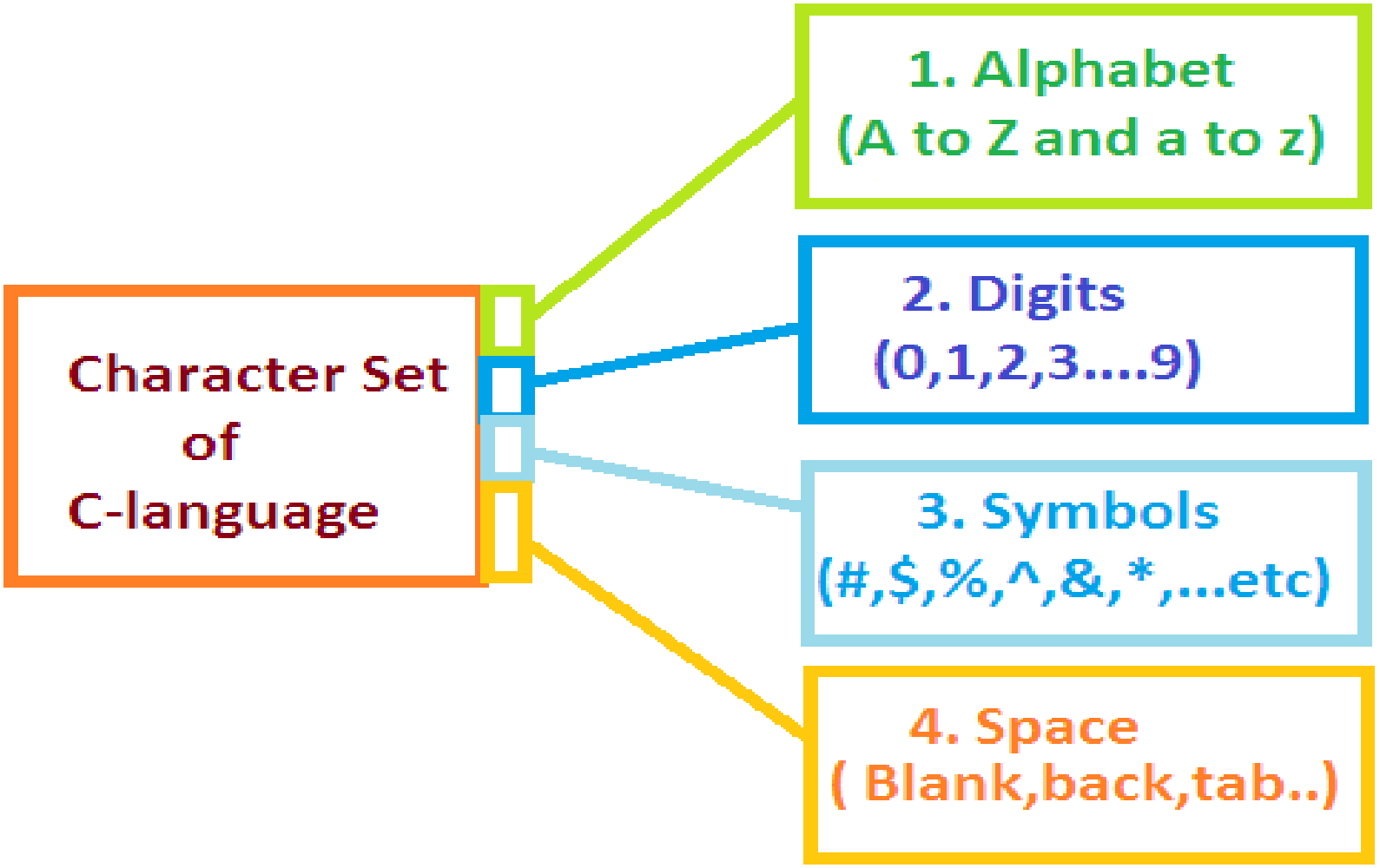
White Spaces

Blank space
Horizontal tab
Carriage return
Form feed

C's Character set



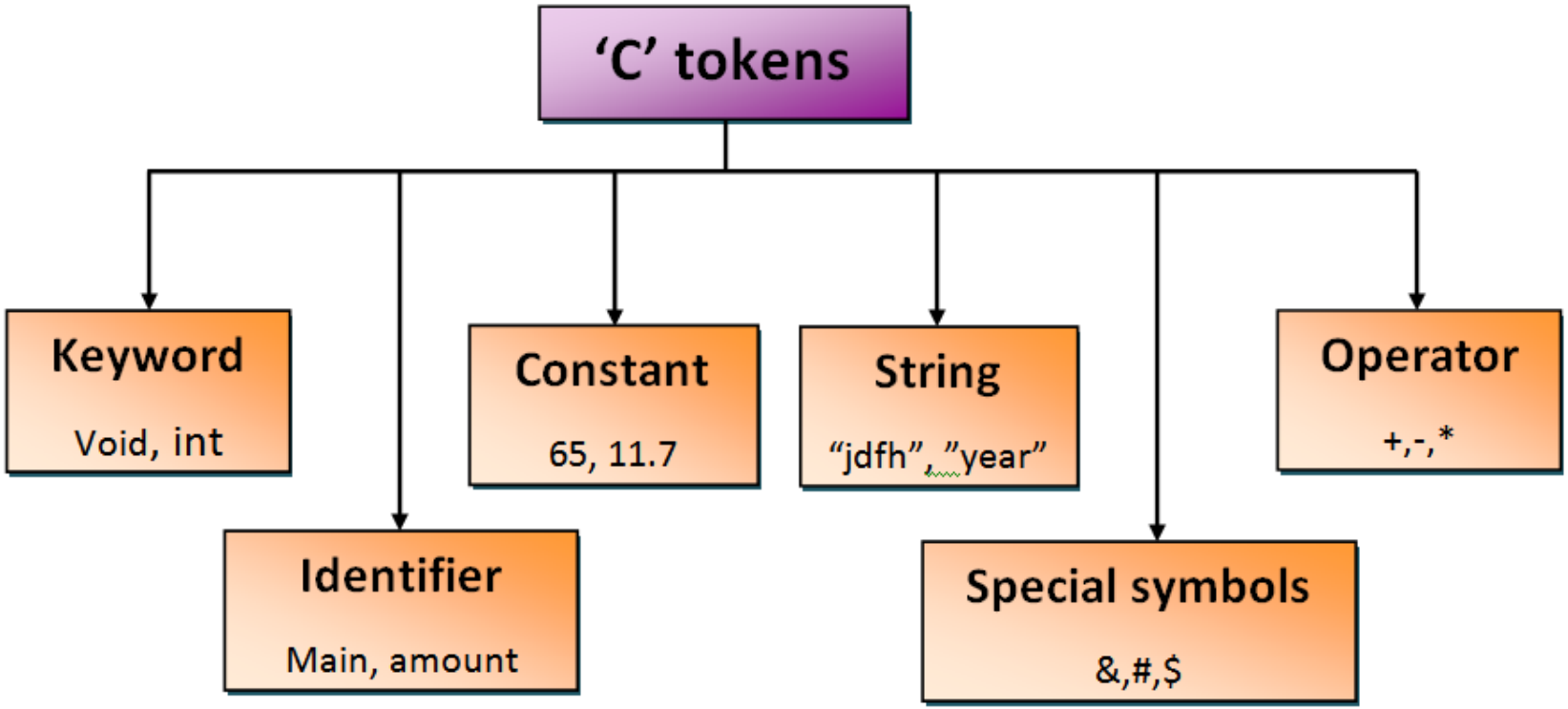
Character Set



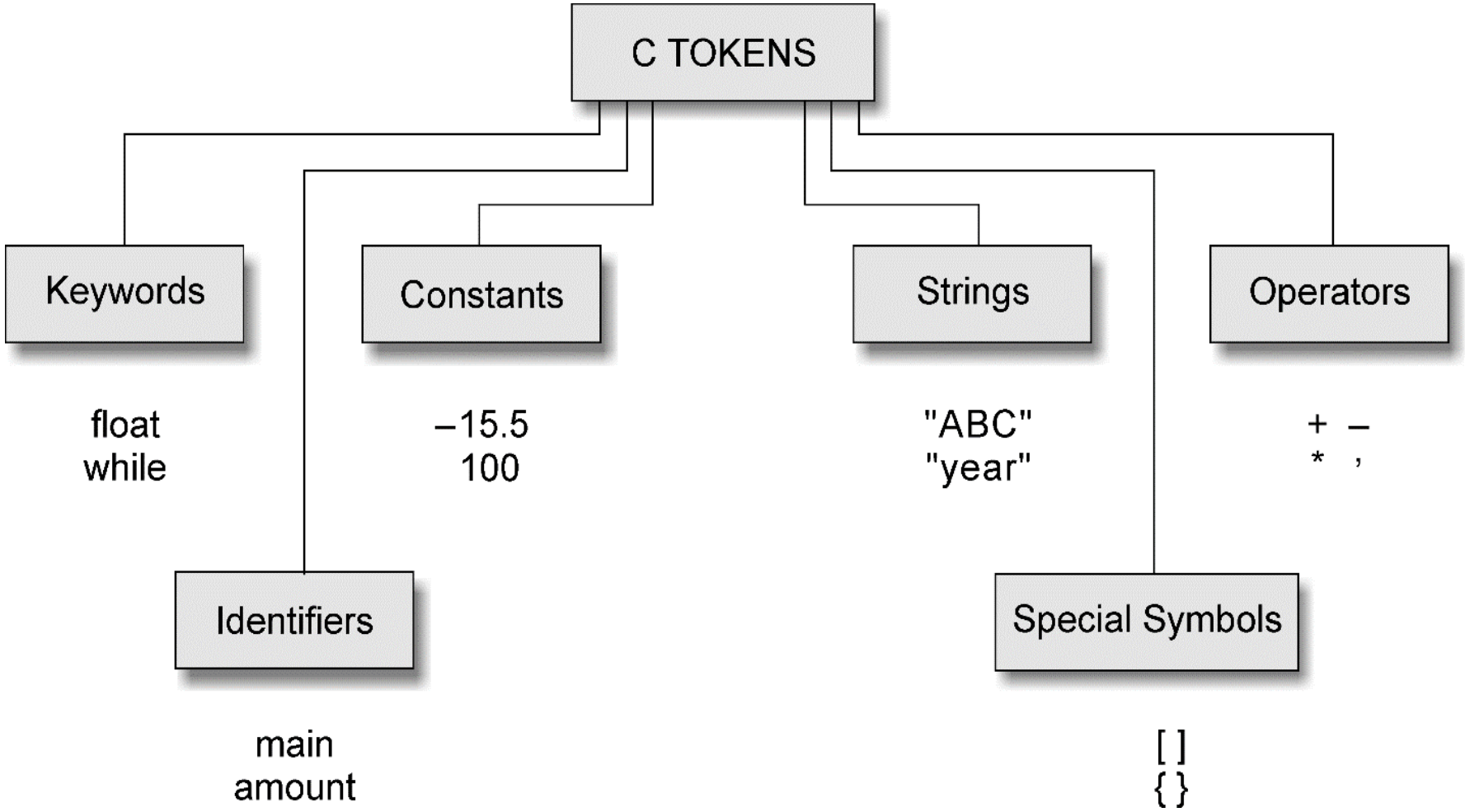
Character Set

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

C Tokens


'C' tokens
Keyword
Void, int
Identifier
Main, amount
Constant
65, 11.7
String
"jdfh", "year"
Special symbols
&,#,$
Operator
+,-,*

C Tokens


C TOKENS
Keywords
Constants
Strings
Operators
float
while
–15.5
100
"ABC"
"year"
+ –
* ,
Identifiers
Special Symbols
main
amount
[ ]
{ }

Keywords

- Keywords are words whose meaning is fixed, as a programmer we can not change its meaning.
- Keywords are also known as reserved words. The keywords must be in second alphabet.
- Some of the keywords are: int, void, double, for, while, switch, goto, if etc.

Keywords in C Language

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

Identifiers

- Identifiers are user defined.
- Identifiers can be the variables, symbolic constants, functions, procedures.
- The name of identifier should be meaningful so that the maintenance of the program becomes easy.
- The rules for naming an identifier are
  - The first character must be an alphabet.
  - Underscore ('_') is valid letter.
  - Reserved words i.e. keywords can not be used as identifiers.
  - Identifier names are case sensitive.
  - Must not contain white space

Write a program to print the message Hello !

```
#include <stdio.h>

void main()
{
    printf("Hello !");
}
```

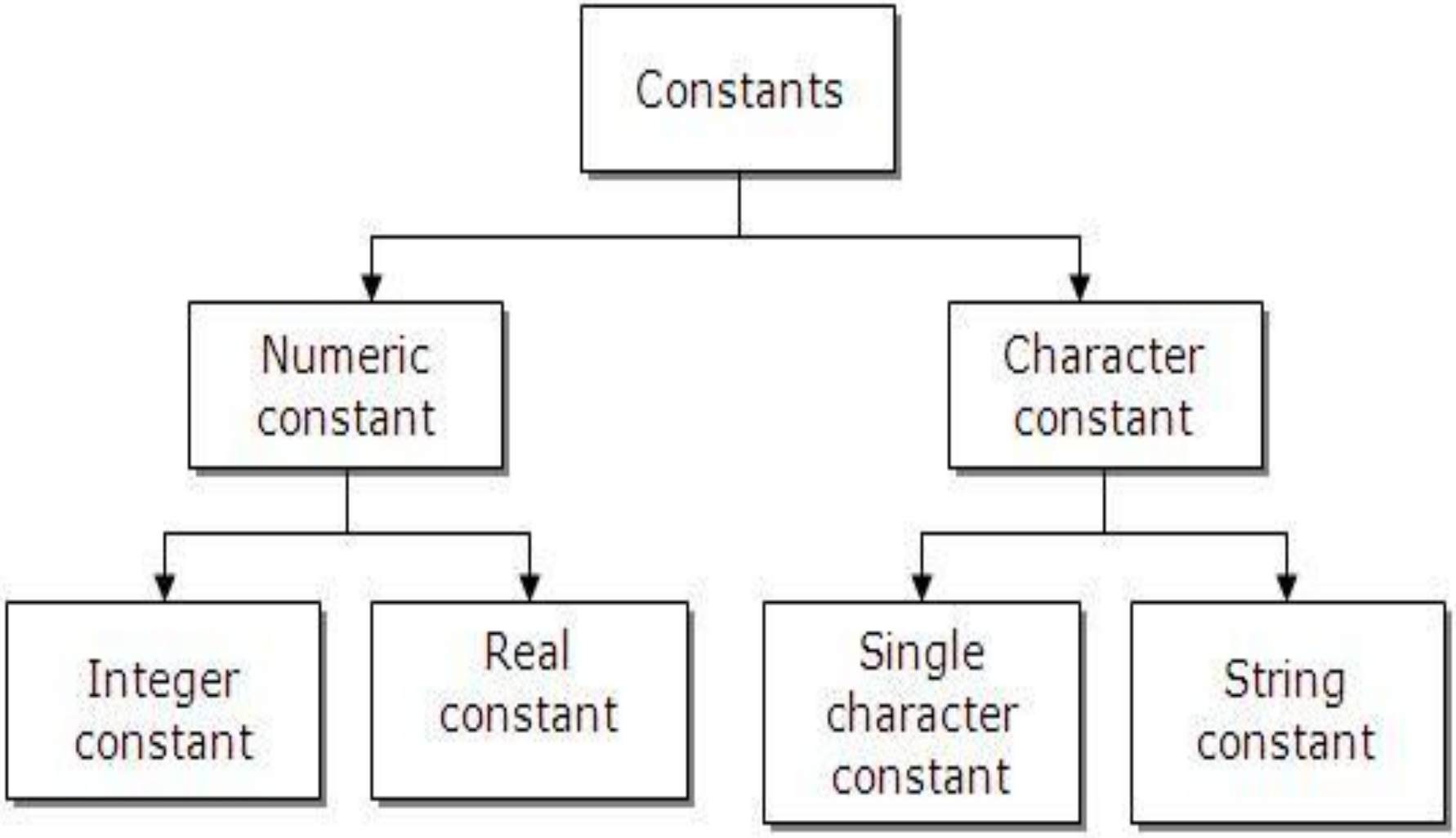
Write a program to print the message Hello World !

```
/* this program developed by XYZ */
#include <stdio.h>

void main()
{  /* start of main() */
    printf("Hello\n World !"); /* use of '\n' character */
}  /* end of main() */
```

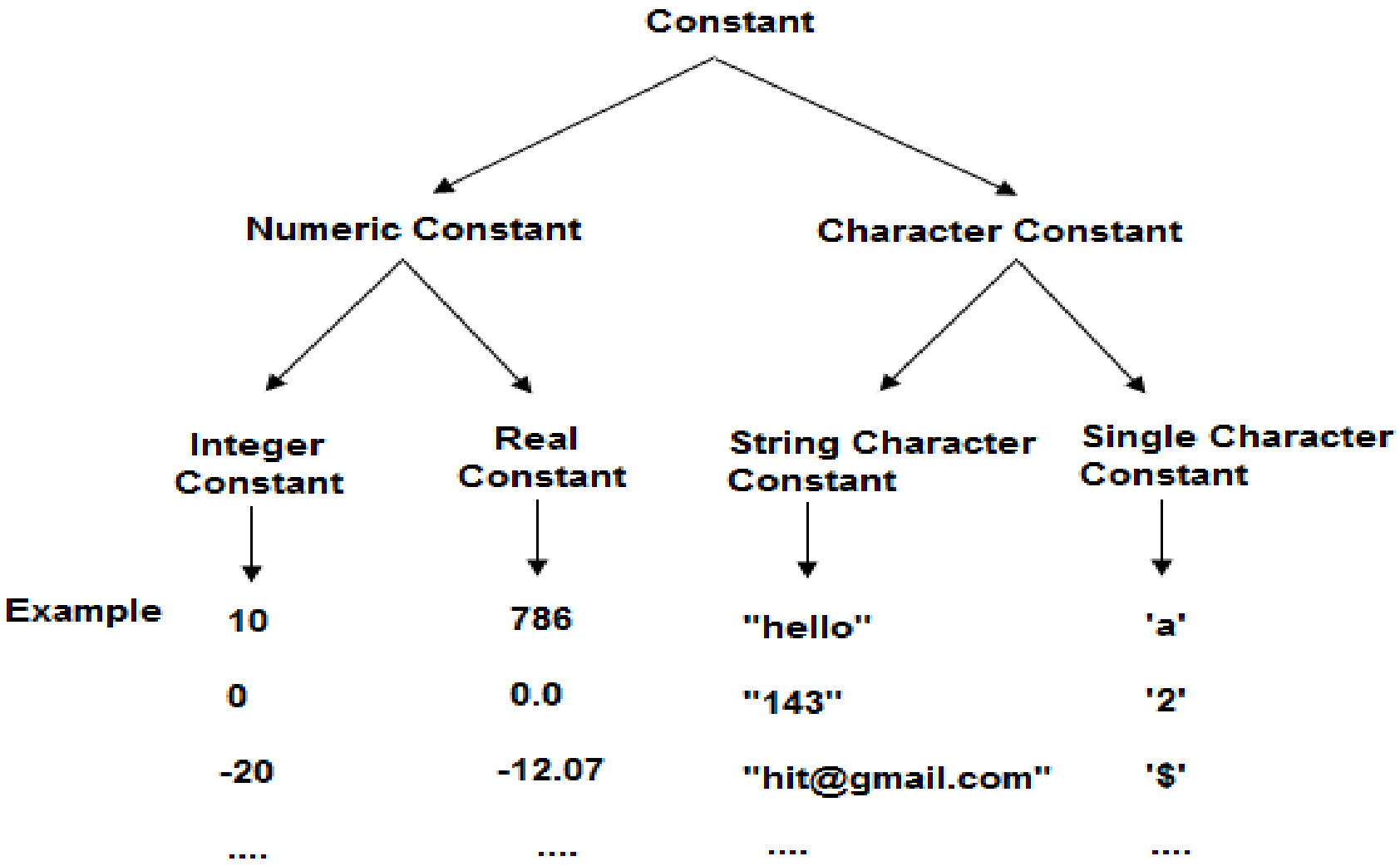
Constants

- The ‘C’ language supports following types of constants:
  - Numeric constants (5, 15, 3.6, -5.4 etc)
  - Non-numeric constants
    - Character constants (‘B’ , ‘a’ , ‘?’ , ‘5’ , ‘+’ etc)
    - String Constants (“Computer”, “XYZ”, “-5.4” etc)

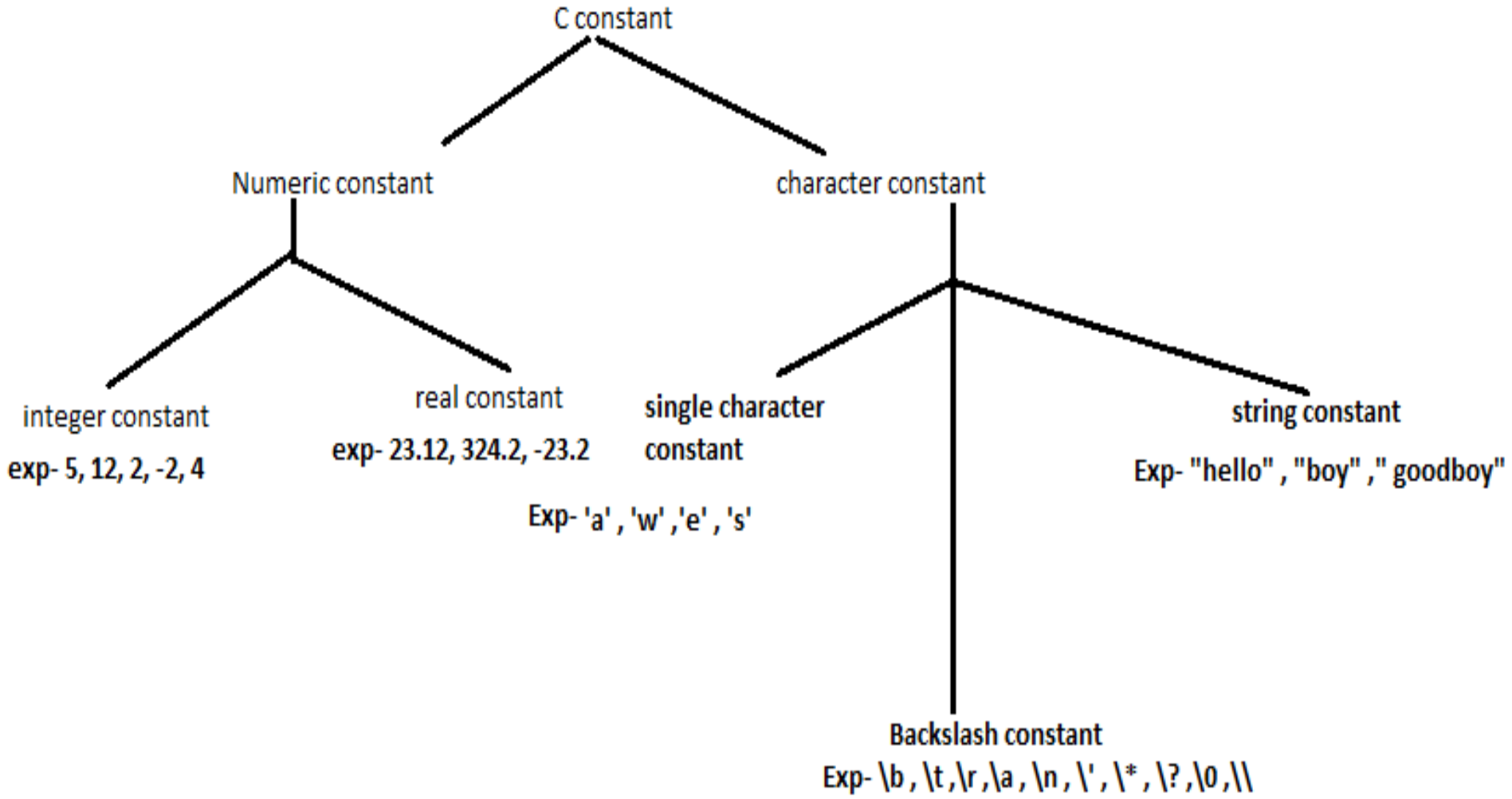
Constants


Constants
Numeric constant
Character constant
Integer constant
Real constant
Single character constant
String constant

Constants


Constant
Numeric Constant
Character Constant
Integer Constant
Real Constant
String Character Constant
Single Character Constant
Example
10
786
"hello"
'a'
0
0.0
"143"
'2'
-20
-12.07
"hit@gmail.com"
'$'
....
....
....
....

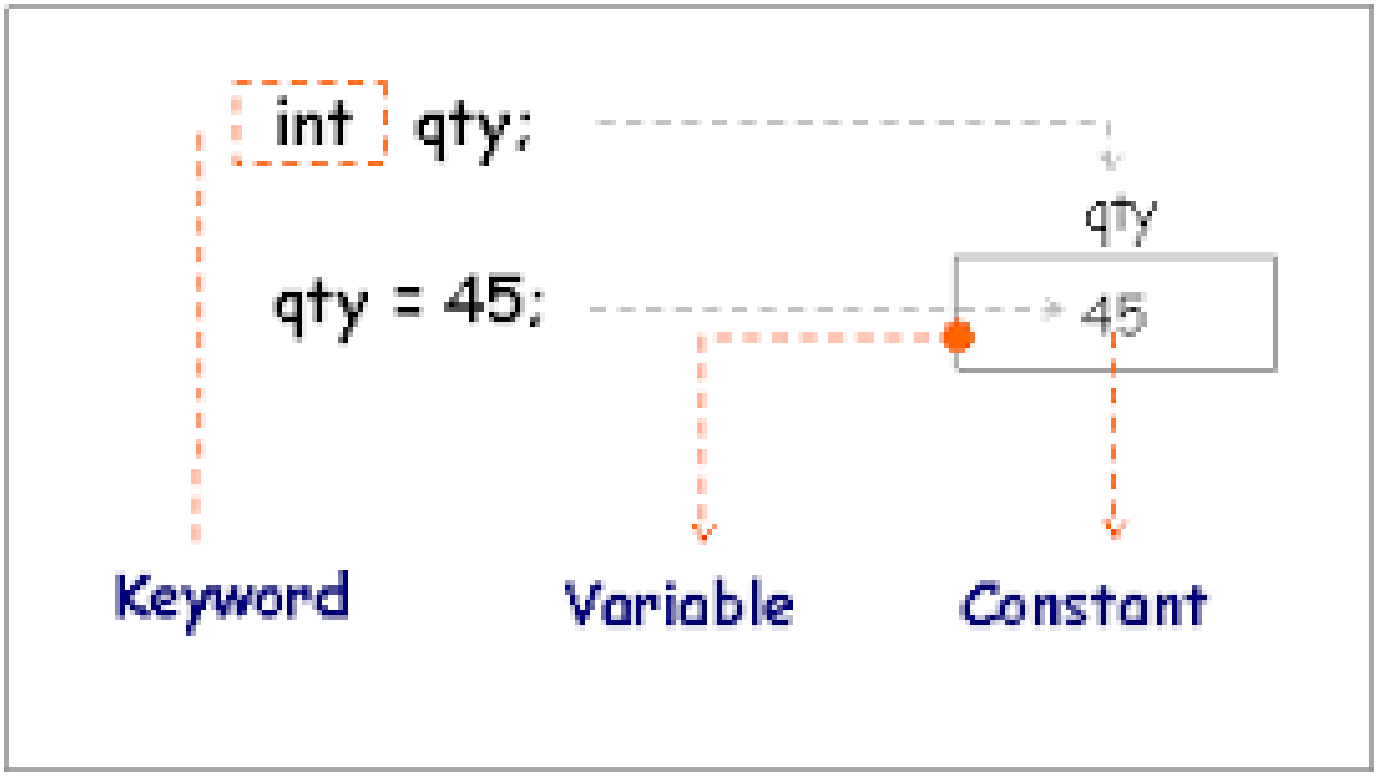
Constants


C constant
Numeric constant
character constant
integer constant
exp- 5, 12, 2, -2, 4
real constant
exp- 23.12, 324.2, -23.2
single character constant
Exp- 'a' , 'w' ,'e' , 's'
string constant
Exp- "hello" , "boy" ," goodboy"
Backslash constant
Exp- \b , \t ,\r ,\a , \n , \' , * , \? ,\0 ,\\

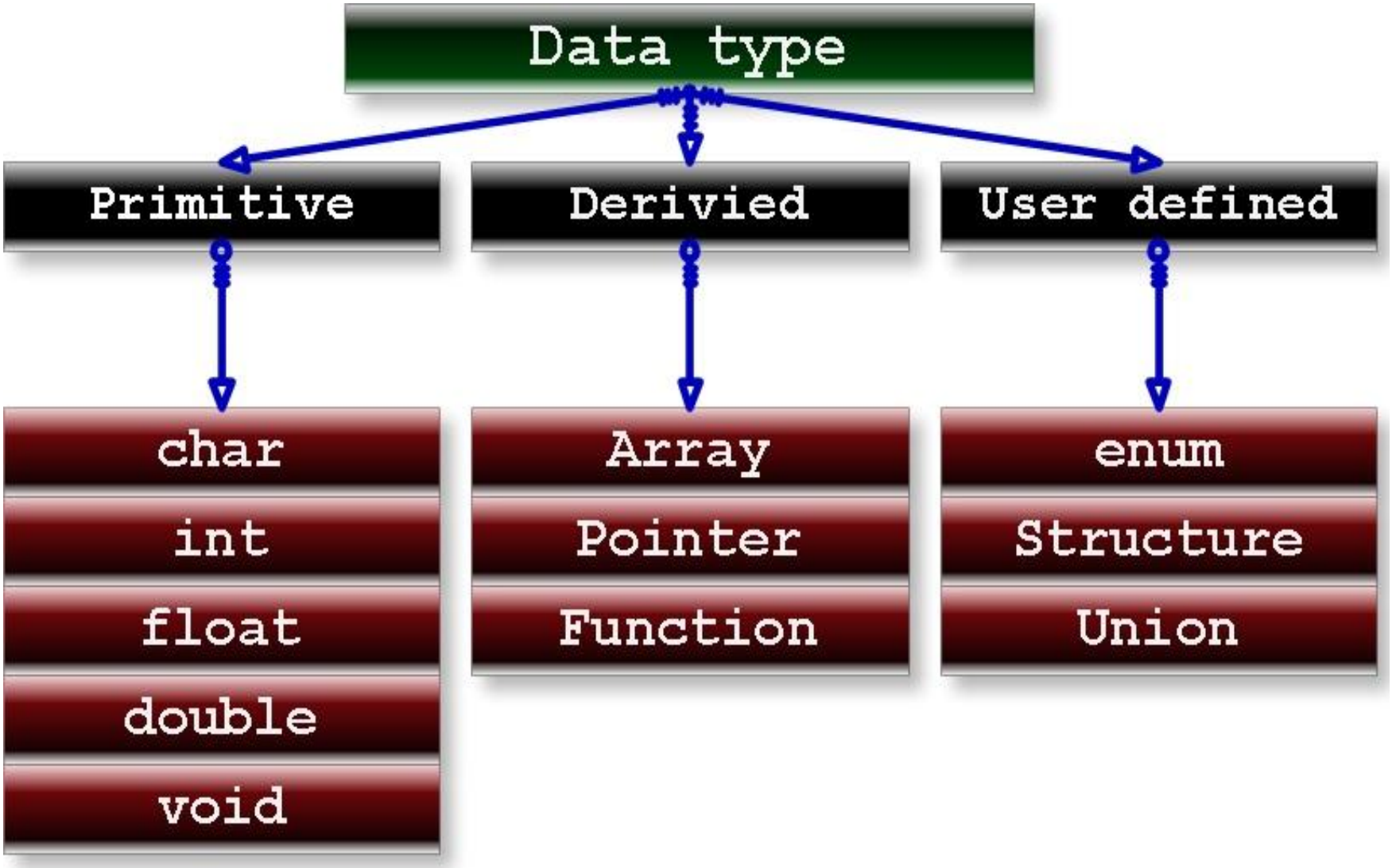
Variables

- A variable is data name that may be used to store a data value.
- Variables are the identifiers whose value changes as opposite to constants.
- As variable is an identifier, all the rules for naming an identifier applies to variables also.

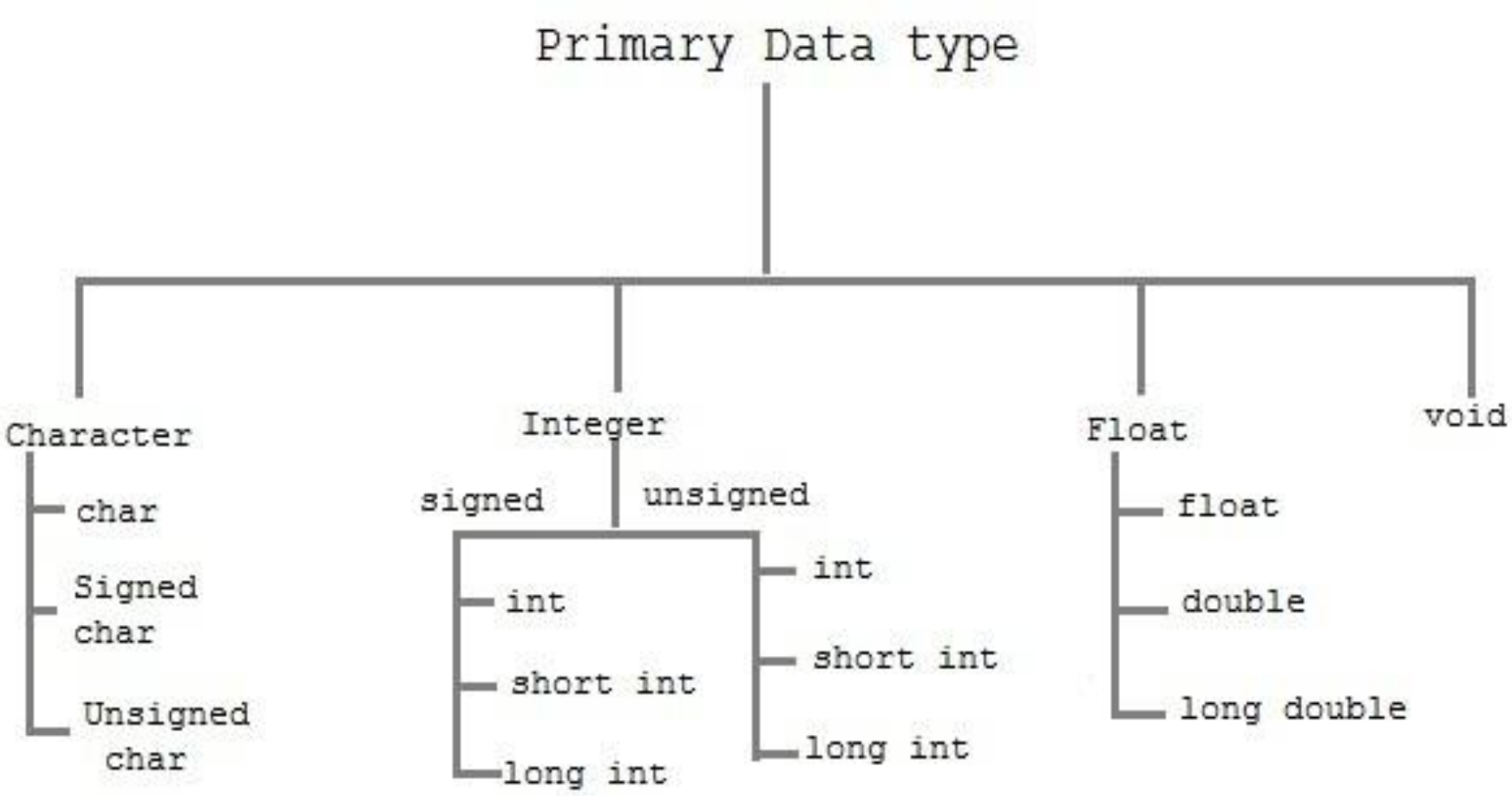
Keyword, Variable and Constant



Data Type in C



Primitive or Primary Data Type



Fundamental data types

Type	Purpose	Size (Bytes)	Range of values
char	for storing characters and strings	1 Byte	-128 to +127
int	For storing integers	2 Bytes	-32768 to +32767
float	For storing real numbers	4 Bytes	3.4e-38 to 3.4e+38
double	For storing double precision floating numbers	8 Bytes	1.7e-308 to 1.7e+308

Qualifiers

- In addition to fundamental 4 types of data, 'C' language supports different qualifiers such as **signed, unsigned, short** and **long.**
- These qualifiers can be put before the fundamental data types to change the range of values supported.

Qualifiers (Cont)

Type	Size (Bytes)	Range of values
unsigned char	1 Byte	0 to 255
unsigned int	2 Bytes	0 to 65535
short int	2 Bytes	-32768 to +32767
unsigned short int	2 Bytes	0 to 65535
long int	4 Bytes	-2147483948 to +2147483647
unsigned long int	8 Bytes	0 to 4294967295
long double	10 Bytes	3.4 e-4932 to 1.1e+4932

Variable declaration

- The syntax for variable declaration is :

  datatype var1,var2,….,varn;

- Following are valid declaration of variables.

```
int  sum, count;
float weight, average;
double rho;
char c;
```

Variable declaration (Cont)

- Assigning value to a variable

```
weight = 55.5;
sum =  0;
```

- Assigning value to a variable at the time of a declaration

```
int sum =0;
int length, count=0;
```

Symbolic Constants

- The constant is given a symbolic name and instead of constant value, symbolic name is used in the program.
- Symbolic constant is defined as

  #define symbolic_name value
- Example:

  #define FLAG 1

  #define PI 3.1415

Symbolic Constants Rules (Constant Identifiers)

- Symbolic names have the same form as variable.
- No blank space between the pound(#) sign and the word "define" is permitted.
- # must be the first character in the line.
- A blank space is required between #define and symbolic name and between symbolic name and the constant.
- There is **no semicolon** '**;**' at the end.
- Can not assign value to symbolic name within the program.
- #define can appear anywhere in the program but before it is referenced in the program.

Program with symbolic constant

```
/* Program illustarting use of declartion, assignment of value to variables. Also explains how to use symbolic
      constants.
Program to calculate area and circumference of a circle */
#include <stdio.h>
 #define PI 3.1415   /* no semicolon here */
 main()
{
      float rad = 5;  /* declaration and assignment */
      float area, circum;  /* declaration of variable */
      area = PI * rad *rad;
      circum = 2 * PI *rad;
      printf("Area of circle = %f\n",area);
      printf("Circumference of circle =%f\n",circum);
}
Output
--------------------------------------------------------------------------------------------------------
Area of circle = 78.537498
Circumference of circle =31.415001
```

We can override this default by specifying
as shown below:

Value
111
–222
45,678
–56,789
9,87,654

ype data. If we want the resulting data
to the number for **float** and letter l or

Value
0.0
0.0
12.0
1.234
–1.2
1.23456789

fine an identifier that would
be used to declare variables

" name given to the data
defined ones. Remember
reate a new type. Some

Another user-defined data type is enumerated data type provided by ANSI standard. It is defined as follows:

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 (known as *enumeration constants*). After this definition, we can declare variables to be of this 'new' type as below:

```
enum identifier v1, v2, ... vn;
```

The enumerated variables v1, v2, ... vn can only have one of the values *value1, value2, ... valuen*. The assignments of the following types are valid:

```
v1 = value3;
v5 = value1;
```

An example:

```
enum day {Monday, Tuesday, ... Sunday};
enum day week_st, week_end;
week_st = Monday;
week_end = Friday;
if (week_st == Tuesday)
week_end = Saturday;
```

The compiler automatically assigns integer digits beginning with 0 to all the enumeration constants. That is, the enumeration constant value1 is assigned 0, value2 is assigned 1, and so on. However, the automatic assignments can be overridden by assigning values explicitly to the enumeration constants. For example:

```
enum day {Monday = 1, Tuesday, ... Sunday};
```

Here, the constant Monday is assigned the value of 1. The remaining constants are assigned values that increase successively by 1.

The definition and declaration of enumerated variables can be combined in one statement. Example:

```
enum day {Monday, ... Sunday} week_st, week_end;
```

3.9 Assigning Values to Variables

integer constants, by default, represent int ... unsigned or long after the number (by appending U or L)

Literal	Type	Value
+111	int	111
-222	int	-222
45678U	unsigned int	45,678
-56789L	long int	-56,789
9876541UL	unsigned long int	9,87,654

Similarly, floating point constants, by default represent **double** type data. If we want the resulting data type to be **float** or **long double**, we must append the letter f or F to the number for **float** and letter l or L for **long double** as shown below:

Literal	Type	Value
0	double	0.0
.0	double	0.0
12.0	double	12.0
1.234	double	1.234
-1.2f	float	-1.2
1.23456789L	long double	1.23456789

3.8.2 User-Defined Type Declaration

C supports a feature known as "type definition" that allows users to define an identifier that would represent an existing data type. The user-defined data type identifier can later be used to declare variables. It takes the general form:

typedef *type* **identifier;**

Where *type* refers to an existing data type and "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 ones. Remember that the new type is 'new' only in name, but not the data type. **typedef** cannot create a new type. Some examples of type definition are:

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

Here, **units** symbolizes **int** and **marks** symbolizes **float.** They can be later used to declare variables as follows:

```
units batch1, batch2;
marks name[50], name2[50];
```

batch1 and batch2 are inclared as **int** variable and name1[50] and name 2[50] are declared as 50 element floating point array variables. The main advantage of **typedef** is that we can create meaningful data type names for increasing the readability of the program.

As example:

The compiler automatically as
That is, the enumeration constant
automatic assignments can be ov
For example:

Here, the constant Monda
that increase successively by
The definition and de
Example:

3.9 Assigning

Variables are created f

COMPUTER PROGRAMMING AND UTILISATION
UNIT 2 – OPERATORS AND THEIR HIERARCHY

Introduction

- The operators are used inside expressions.
- Expression is a formula having one or more operators and may have zero or more operands.
- An Instruction which is used to manipulate data using operators, is known as Arithmetic Instruction.
- The operand can be a variable, constant or a name of a function. For example, following is an expression involving two operands a and b with '+' operator.

a + b

Arithmetic Operators

- Arithmetic operators are used to perform arithmetic operations. 'C' language supports following arithmetic operators.

  +, -, *, / and %(modulo division)

- The precedence of arithmetic operators is

  * / % first priority

  + - second priority

- If in an expression, the operators having same priority occur, then the evaluation is from left to right. Sub expressions written in brackets are evaluated first.

Program explaining Arithmetic operators

```
/* Program illustrating use of arithmetic operators. The numbers x and y are initialized in the
    program itself with x = 25 and y =4 */
#include <stdio.h>
main()
{
    int x=25;
    int y=4;

    printf("%d + %d = %d\n",x,y,x+y);
    printf("%d - %d = %d\n",x,y,x-y);
    printf("%d * %d = %d\n",x,y,x*y);
    printf("%d / %d = %d\n",x,y,x/y);
    printf("%d %% %d = %d\n",x,y,x%y);
}
```

Program explaining Arithmetic operators

Output

```
--------------------------------------------------------------------------------
    ----
25 + 4 = 29
25 - 4 = 21
25 * 4 = 100
25 / 4 = 6
25 % 4 = 1
--------------------------------------------------------------------------------
    ----
```

Program – floating point arithmetic

/* Write a program to convert Fahrenheit temperature to centigrade. This program explains floating-point arithmetic. The formula is c = 5/9*(f-32). In the program, if we write the formula using 5/9 then, it will result into 0. Because in integer arithmetic 5/9 =0.

To get the correct answer we should write the formula as c= 5*(f-32)/9 or c= 5.0/9 * (f-32) or c =5.0/9.0*(f-32) so that integer gets upgraded to real. */

```
#include <stdio.h>
main()
{
    float fahr,cent;
    printf("Give the value of temperature in Fahrenheit\n");
    scanf("%f",&fahr);
    cent = 5*(fahr-32)/9;
    printf("Fahrenheit temperature = %f\n",fahr);
    printf("Centigrade temperature = %f\n",cent);
}
```

Program – floating point arithmetic

Output

```
--------------------------------------------------
--------------------------------------------------
Give the value of temperature in fahrenhit
100
Fahrenhit temperature = 100.000000
Centigrade temperature = 37.777779
--------------------------------------------------
--------------------------------------------------
```

Relational Operators

- In programming, relational operators are used to compare whether some variable value is higher, equal or lower with other variable. 'C' language provides following relational operators.
- == (Equals), != (Not Equal to), < (Less than)

  > (Greater than), <= (Less than or equal to),

  >= (Greater than or equal to)
- The relational expression will have value either True (1) or False (0).

Logical Operators

- We need to take certain action based on outcome of some conditions. Logical operators help us to combine more than one condition, and based on the outcome certain steps are taken.
- 'C' language supports following logical operators.

  && (Logical AND) , | | (Logical OR),

  ! (Logical NOT i.e Negation)
- If more than one operator is used, ! (NOT) is evaluated first, then && (AND) and then | | (OR). We can use parentheses to change the order of evaluation.

Logical Operators (Cont)

- For example, if we have a=2, b=3 and c=5 then

  a<b && c==5 is true

  a< b || c>10 is true

  (b<c || b>a) && (c ==5) is true

  ! (5>3) is false

Assignment Operator

- 'C' language supports = assignment operator. It is used to assign a value to a variable. The syntax is: variablename = expression;
- In an expression involving arithmetic operators, assignment operator has the least precedence
- 'C' language also supports the use of shorthand notation also. For example, the statement a=a+5 can be written using shorthand notation as a += 5. So, shorthand notation can be used for any statement whose form is

  varname = varname operator expression;

  into

  varname operator= expression;

Program using shorthand notations

```
/* Program explaining short hand notations   */
#include <stdio.h>
main()
{
      int a;
      printf("Give the value of a\n");
      scanf("%d",&a);
      a += 5;        /* a= a +5 */
      printf("a= %d\n",a);
      a -= 5;        /* a = a-5 */
      printf("a= %d\n",a);
      a *=5;         /* a = a* 5 */
      printf("a= %d\n",a);
      a /=5;         /* a= a /5 */
      printf("a= %d\n",a);
      a %= 5;         /* a= a %5 */
      printf("a= %d\n",a);
}
```

Program using shorthand notations (Cont)

Output

```
---------------------------------------------------------------------------
Give the value of a
4
a= 9
a= 4
a= 20
a= 4
a= 4
---------------------------------------------------------------------------
```

Increment/Decrement operators

- 'C' language provides special operators
- ++ (increment operator) and -- (decrement operator)
- If a is a variable, then

  ++a is called prefix increment

  a++ is called postfix increment.

  Similarly, --a is called prefix decrement

  a-- is called postfix decrement.

Program prefix and postfix

```
/* Program showing use of increment ++ and decrement -- operators */
#include <stdio.h>
main()
{
        int x=10;
        int y;
        int z=0;
        x++;        /* x incremented using postfix notation*/
        ++x;        /* x incremented using prefix notation*/
        y = ++x; /* x incremented first and then assigned to y.  */
        printf("Value of x=%d y=%d and z=%d\n",x,y,z);
        z= y--;      /* y assigned to z first and then decremented*/
        printf("Value of x=%d y=%d and z=%d\n",x,y,z);
}
```

Program prefix and postfix

Output:-

Value of x=13 y=13 and z=0

Value of x=13 y=12 and z=13

Program prefix and postfix (Stepwise)

```
/* Program showing use of increment ++ and decrement -- operators */
#include <stdio.h>
main()
{
        int x=10;
        int y;
        int z=0;
        x++;        /* x incremented using postfix notation x=11*/
        ++x;        /* x incremented using prefix notation x=12*/
        y = ++x; /* x incremented first and then assigned to y. y=13  x=13 */
        printf("Value of x=%d y=%d and z=%d\n",x,y,z);
        z= y--;      /* y assigned to z first and then decremented z=13 y =12*/
        printf("Value of x=%d y=%d and z=%d\n",x,y,z);
}
```

Conditional Operator (Ternary operator)

- 'C' language provides ? operator as a conditional operator. It is also known as a **ternary** operator. It is used as shown below.

  expr1 ? expr2 :expr3;

  where expr1 is a logical expression, logical expression can be TRUE or FALSE. expr2 and expr3 are expressions.

- expr1 is evaluated first, and depending on its value, either expr2 or expr3 is evaluated. If expr1 is TRUE, then expr2 is executed, otherwise expr3 is executed.

Program – Ternary operator

```
/* Write a program to find maximum and minimum of two numbers using
   ternary ? operator  */
#include <stdio.h>
main()
{
    int x,y,max,min;
    printf("Give two integer numbers\n");
    scanf("%d %d",&x,&y);

    max = (x>y) ? x: y; /* x>y is a logical expression */
    min = (x<y) ? x: y; /* x<y is a logical expression */

    printf("maximum of %d and %d is = %d\n",x,y,max);
    printf("minimum of %d and %d is = %d\n",x,y,min);
 }
```

Program – Ternary operator

Output

```
--------------------------------------------------------------------------------
    ------------------------
Give two integer numbers
3 6
maximum of 3 and 6 is = 6
minimum of 3 and 6 is = 3
--------------------------------------------------------------------------------
    ------------------------
```

Bit-wise operators

- 'C' language supports some operators which can perform at the bit level. These type of operations are normally done in assembly or machine level programming. But, 'C' language supports bit level manipulation also, that is why few people says 'C' language is also known as middle-level programming language.
- Bit-wise operators are:

  & (Bit-wise AND) , | (Bit-wise OR),

  ^ (Bit-wise Exclusive OR (XOR)), << (Left Shift) , >> (Right Shift), ~ (Bit-wise 1's complement)

Program –Bit wise operators

```
/* Write a program to multiply and divide the given number by 2 using bit-wise operators << and
    >> */

#include <stdio.h>
main()
{
    int x;
    int mul,div;
    printf("Give one integer number\n");
    scanf("%d",&x);
    mul = x << 1;   /* left shift */
    div = x >> 1;    /* right shift */

    printf("multiplication of %d by 2 = %d\n",x,mul);
    printf("division of %d by 2 = %d\n",x,div);

}
```

Program –Bit wise operators

Output

```
--------------------------------------------------
   --------------------------------------------
Give one integer number
8
multiplication of 8 by 2 = 16
division of 8 by 2 = 4
--------------------------------------------------
   --------------------------------------------
```

Other opeartors

- Comma (,) operator is used to combine multiple statements. It is used to separate multiple expressions. It has the lowest precedence.
- **sizeof** operator is used to find out the storage requirement of an operand in memory. It is an unary operator, which returns the size in bytes.
  - sizeof(datatype)
  - sizeof(variable)
  - sizeof(constant)

TABLE 2.9 Data types and their control strings

Data Type	*Size (bytes)*	*Range*	*Format String*
`char`	1	−128 to 127	`%c`
`unsigned char`	1	0 to 255	`%c`
`short or int`	2	−32768 to 32767	`%i` or `%d`
`unsigned int`	2	0 to 65535	`%u`
`long`	4	−2147483648 to 2147483647	`%ld`
`unsigned long`	4	0 to 4294967295	`%lu`
`float`	4	$3.4e^{-38}$ to $3.4e^{+38}$	`%f` or `%g`
`double`	8	$1.7e^{-308}$ to $1.7e^{+308}$	`%lf`
`long double`	10	$3.4e^{-4932}$ to $1.1e^{+4932}$	`%lf`
`enum`	2	-32768 to 32767	`%d`

Data Types and their size

Type	Size (bits)	Size (bytes)	Range
char	8	1	-128 to 127
unsigned char	8	1	0 to 255
int	16	2	-2^{15} to $2^{15}-1$
unsigned int	16	2	0 to $2^{16}-1$
short int	8	1	-128 to 127
unsigned short int	8	1	0 to 255
long int	32	4	-2^{31} to $2^{31}-1$
unsigned long int	32	4	0 to $2^{32}-1$
float	32	4	3.4E-38 to 3.4E+38
double	64	8	1.7E-308 to 1.7E+308
long double	80	10	3.4E-4932 to 1.1E+4932

Operator Groups

- Unary Operators (++, --, sizeof, !)
- Arithmetic Operators (+, -, *, /, %)
- Bitwise Operators (&, |, ^, ~, >>, <<)
- Relational Operators (<, >, <=, >=, ==, !=)
- Logical Operators (!, &&, ||)
- Conditional Operators (expr1 ? expr2 :expr3;)
- Assignment Operators (=)

Precedence and associativity of operators

- When arithmetic expressions are evaluated, the answer of that expression depends on the value of operands, the operators used and also on the precedence and associativity of the operators used.
- In 'C', the operators having the same precedence are evaluated from left to right or right to left depending upon the level at which these operators belong. So, associativity decides left to right evaluation or right to left evaluation.

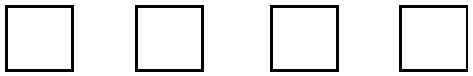
Rules for Evaluation of Expression

- Parenthesized sub-expression from left to right are evaluated.
- If parentheses are nested, the evaluation begins with innermost expression.
- The precedence rule is applied in determining the order of application of operators in evaluating sub-expression.
- The associativity 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 rule of precedence.
- When parentheses are used, the expression within parentheses assume highest priority.

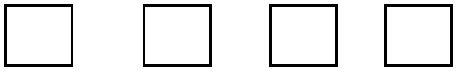
OPERATOR PRECEDENCE

Label the execution order for the following expressions

a + b + c + d + e



a + b * c - d / e



a / (b + c) + d % e

a / (b * (c + (d - e)))

a + b + c + d + e

a + b * c - d / e

a / (b + c) + d % e

a / (b * (c + (d - e)))

OPERATOR PRECEDENCE

Convert the following operations to C expression

a. $rate^2 + delta$

b. $2(salary + bonus)$

c. $\dfrac{1}{time + 3mass}$

d. $\dfrac{a - 7}{t + 9v}$

a. (rate*rate) + delta

b. 2*(salary + bonus)

c. 1/(time + (3*mass))

d. (a - 7) / (t + (9 * v))

Precedence and associativity of operators (Example)

- The expression, 5 -3 * 4 +6 /4 will be evaluated as 3 *4 first, then 6/4,then – operation and at last + operation as shown below.

5-3 * 4 + 6 /4

▼

5 - 12 + 6/4

▼

5 - 12 + 1

▼

-7 +1

▼

-6

Type conversion

- Whenever an expression involves two different types of operands, 'C' language applies the type conversion rules to evaluate an expression.
- If the operands are of different type, then the operand with a lower type is upgraded to the higher type and then the operation is performed.
- In the case of assignment,
  - If float is assigned to integer, then fractional part is truncated as shown in above program.
  - If double is assigned to float, then rounding takes place.
  - If long int is assigned to integer, then additional bits are dropped, remember that long int requires 4 bytes while int requires only 2 bytes.

Program – Type Conversion

```
/* Program demonstrating type conversion  */
#include <stdio.h>
main()
{
    int a,b;
    float c;
    double d;
    a=5;
    c= 5.5;
    d = 4.0;
    b = a *c +d /10;
    printf("value of b = %d\n",b);
}
```

Output

---

value of b = 27

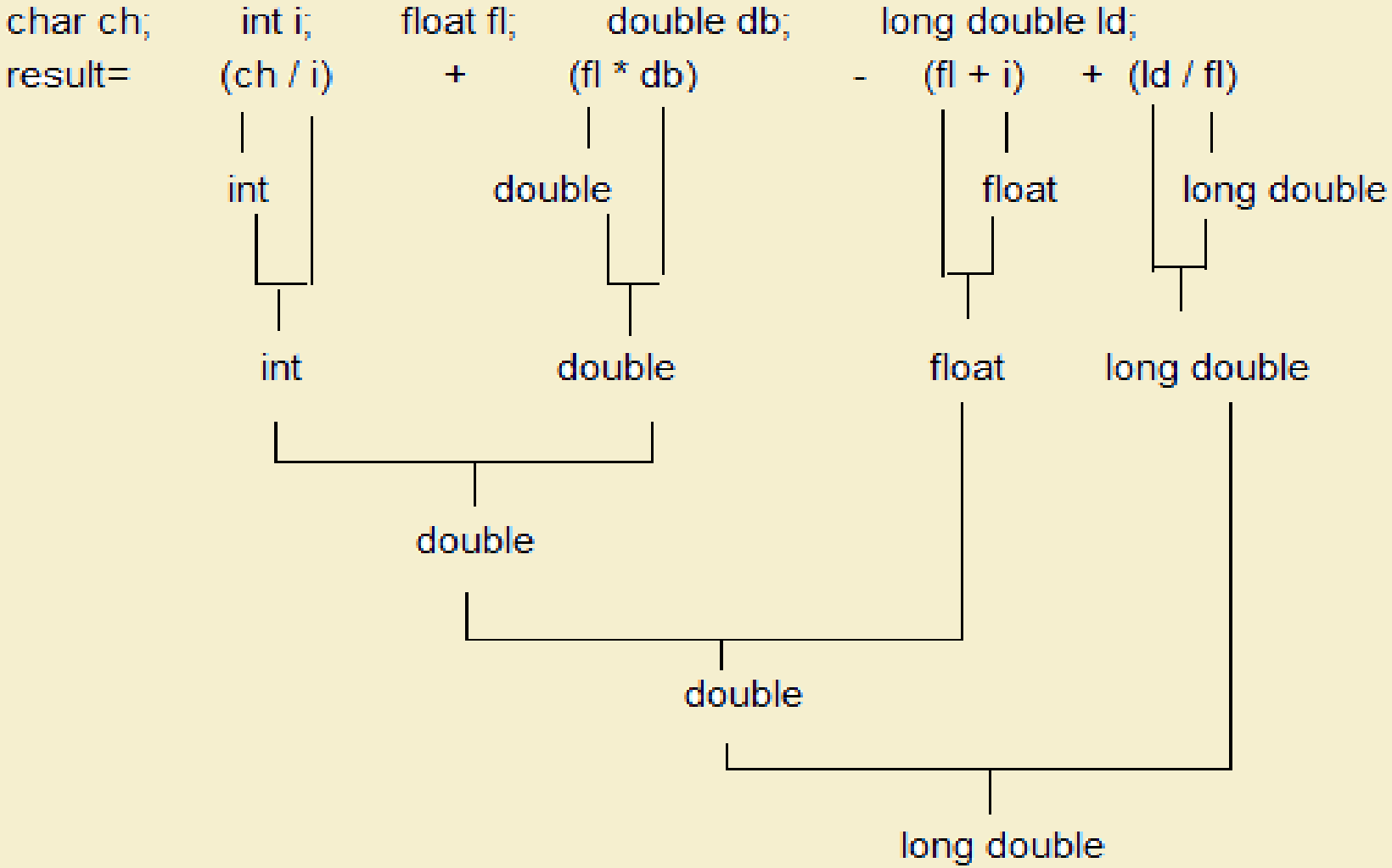
Explanation – Program Type conversion

In the statement b = a *c +d /10,

- a*c will be done first, here **a** will be upgraded to float because other operand **c** is float.

  So, a*c will evaluate to 27.5.
- Then, d/10 will be evaluated, 10 will be converted into 10.0 (double) because **d** is double. So, d/10 will evaluate to 0.4.
- Then 27.5 + 0.4 evaluates to 27.9. This value is assigned to variable **b**, which is integer, so truncated value of 27.9 will be the value of b i.e 27 will be assigned to **b**.

Type conversion - Implicit



Type casting

- When user needs the type conversion explicitly, then type casting is used.
- Type conversion is automatic while type casting is explicitly specified by the programmer in the program.
- The type casting can be specified in following form (typename) expression;

Program - Type casting

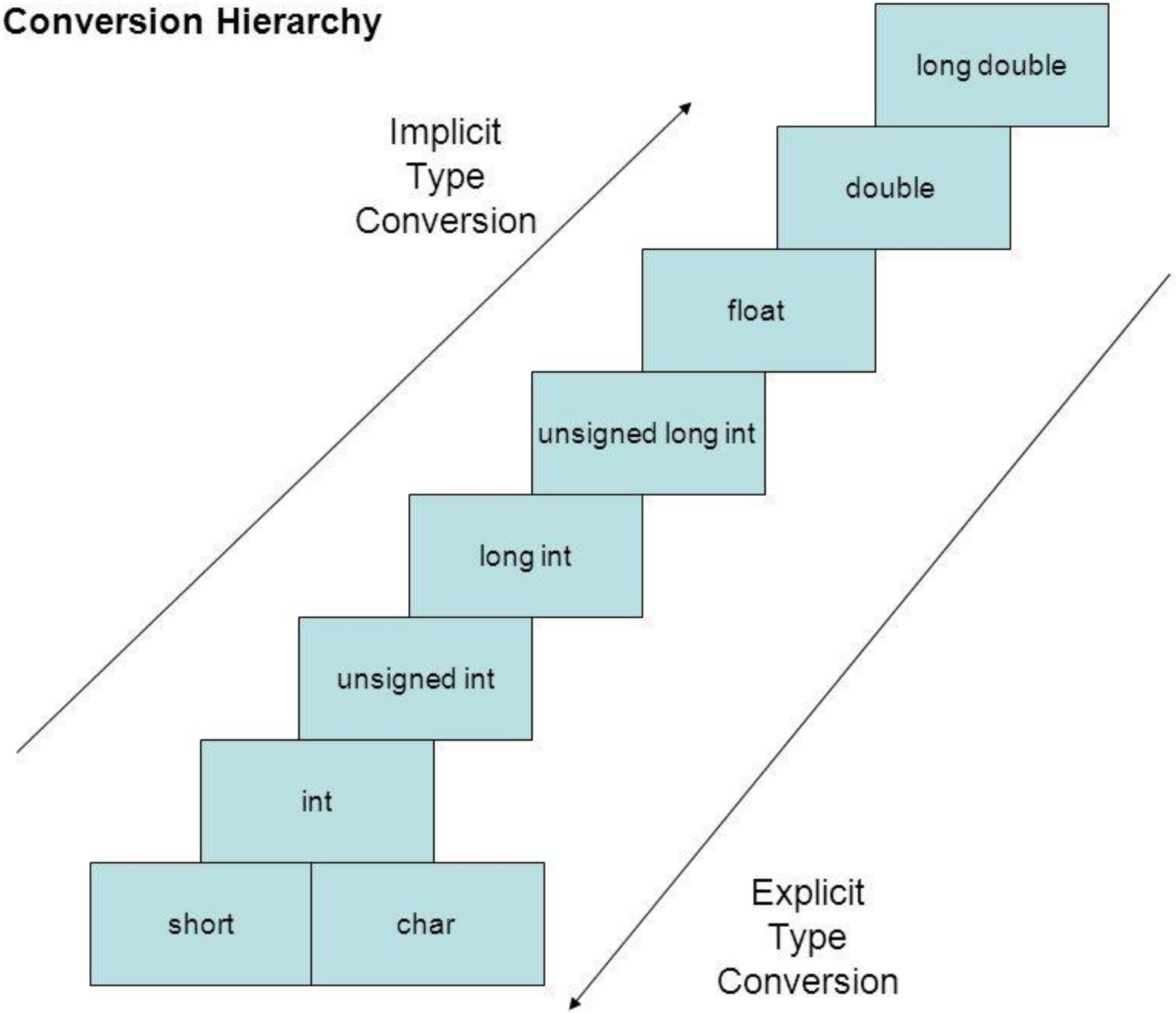
```
/* Program demonstrating type casting  */
#include <stdio.h>
main()
{
    int sum =47;
    int n = 10;
    float avg;

    avg = sum/n;                                    /*avg without type cast */
    printf("avg=sum/n          = %f\n",avg);
    avg = (float)sum/n;                             /*avg with type cast on sum */
    printf("avg=(float)sum/n   = %f\n", avg);
    avg = (float)(sum/n);                           /*avg without type cast on (sum/n) */
    printf("avg=(float)(sum/n) = %f\n", avg);
}
```

Program - Type casting

Output

```
--------------------------------------------------------------------------------
--------------------------------------
avg=sum/n                = 4.000000
avg=(float)sum/n         = 4.700000
avg=(float)(sum/n)       = 4.000000
--------------------------------------------------------------------------------
--------------------------------------
```

Type Conversion Hierarchy



Header files

- The files included at the beginning of 'C' program and having an extension .h are called as header files.
- The content of these files go at the head of the program, in which the header file is included.
- In 'C' language, the prototypes of all the library functions are categorized according to their function in header files.
- stdio.h header file contains all the library functions for standard Input- Output functionality, that is why the name stdio.h where, **std** stands for standard, **i** stands for Input and **o** stands for Output.

Header files (Cont)

- Following is the list of important header files

  stdio.h contains standard input-output functions.

  math.h contains mathematical functions.

  conio.h contains console input-output functions.

  string.h contains string processing functions.

  ctype.h contains character type checking functions.

- The advantage of using the header files is that we do not have to write the code again and again in each program.

Mathematical functions such as sqrt, cos, log etc., are the most frequently used ones. To use the mathematical functions in a C program, we should include the line

#include<math.h>

in the beginning of the program.

Function	Meaning
Trignometric	
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 x
Hyperbolic	
cosh(x)	Hyperbolic cosine of x
sinh(x)	Hyperbolic sine of x
tanh(x)	Hyperbolic tangent of x
Other functions	
ceil(x)	x rounded up to the nearest integer
exp(x)	e to the power x
fabs(x)	absolute value of x
floor(x)	x rounded down to the nearest integer
fmod(x,y)	remainder of x/y
log(x)	natural log of x, x>0
log10(x)	base 10 log of x.x>0
pow(x,y)	x to the power y
sqrt(x)	square root of x,x>=0

Program for pow()

```
#include <stdio.h>
#include <math.h>
int main()
{
   printf ("2 power 4 = %f\n", pow (2.0, 4.0) );
   printf ("5 power 3 = %f\n", pow (5, 3) );
   return 0;
}
```

OUTPUT:

2 power 4 = 16.000000
5 power 3 = 125.000000

Program for ceil()

```
#include <stdio.h>
#include <math.h>
int main ()
{
    double val1, val2, val3, val4;
    val1 = 1.6; val2 = 1.2; val3 = 2.8; val4 = 2.3;
    printf ("value1 = %.1lf\n", ceil(val1));
    printf ("value2 = %.1lf\n", ceil(val2));
    printf ("value3 = %.1lf\n", ceil(val3));
    printf ("value4 = %.1lf\n", ceil(val4));
    return(0);
    }
```

Preprocessor directives

- Preprocessor is a program that processes the source program before the control is given to the compiler. Preprocessor understands some commands known as preprocessor directives. The directive begins with symbol #. There are many directives which the preprocessor understands.
- For example,

  #include <stdio.h>

  # define PI 3.1415

  include directive includes the file mentioned after it, while define directive defines a symbolic constant.

Program - Preprocessor directives

```
/* Write a program that demonstrates symbolic constants */
#include <stdio.h>
#define cube(x) (x*x*x)
void main()
{
    int ans,a;
    a=3;
    ans = cube(a);
    printf("Answer = %d\n",ans);
}
```

Output

Answer = 27

-

Program

```
/* Write a program to exchange two variables  */
#include <stdio.h>
void main()
{
    int x, y, temp;
    x=5;
    y=3;
    printf("Before exchange x =%d and y =%d\n", x, y);
    temp = x;
    x = y;
    y = temp;
    printf("After exchange  x =%d and y =%d\n", x, y);
}
```

Program

```
/* Write a program to exchange two variables without use of third variable */
#include <stdio.h>
void main()
{
    int x, y;
    x=5;
    y=3;
    printf("Before exchange x =%d and y =%d\n", x, y);
    x = x + y;
    y = x - y;
    x = x - y;
    printf("After exchange  x =%d and y =%d\n", x, y);
}
```

Program

```
/* Write a program to calculate area, perimeter and diagonal length of rectangle of length l and
     width b. Area = l *b  Perimeter = 2*l + 2*b   Diagonal Length = sqrt(l*l +b*b)   */
#include <stdio.h>
#include <math.h>
void main()
{
    int l,b;
    int area, peri;
    float d;
    printf("Give length and width of rectangle\n");
    scanf("%d %d", &l, &b);
    area = l*b;
    peri = 2*l +2*b;
    d = sqrt(l*l + b*b);              /* find square root of number in brackets */
    printf("Area = %d Perimeter = %d Diagonal Length = %f\n", area, period);
}
```

Program

```
/* Write a program to evaluate the polynomial 3x^3 - 4x +9
 */
#include <stdio.h>
#include <math.h>

void main()
{
    float x, ans=0;
    printf("Give value of x\n");
    scanf("%f",&x);
    ans = ans + 3 * x*x*x;
    ans = ans - 4*x;
    ans = ans +9;
    printf("Value of polynomial is = %6.2f\n",ans);
}
```

COMPUTER PROGRAMMING AND UTILISATION
UNIT 2 – INPUT OUTPUT FUNCTIONS

Introduction

- 'C' language uses the references **stdin** for standard input device, **stdout** for standard output device, and **stderr** for standard error device.
- 'C' language does not provide any input/output operations as part of the language. The operations are available as library functions. The input/output functions are provided as library through the header file stdio.h.
- The header file stdio.h contains the definitions of stdin, stdout and stderr.
- standard input/output operations are buffered.

Input with scanf() function

- The scanf() function is used to get formatted input from standard input device which is keyboard. The syntax of scanf() function is:

  scanf("control string", list of addresses of variables);

- For example,

  char x;

  int y;

  float z;

  scanf("%c %d %f", &x, &y, &z);

  In above scanf() function, **"%c %d %f" is control string**, where format specifies are preceded by % symbol.

- The first input will be treated as character, second as integer, and third as float, and will be stored in x, y and z respectively.
- The variables x, y and z are all preceded by & symbol, because scanf() function requires the address of the variable in which the value is to be stored.

Output with printf() function

- The syntax of printf() function is:

  printf("Control string", list of variables);

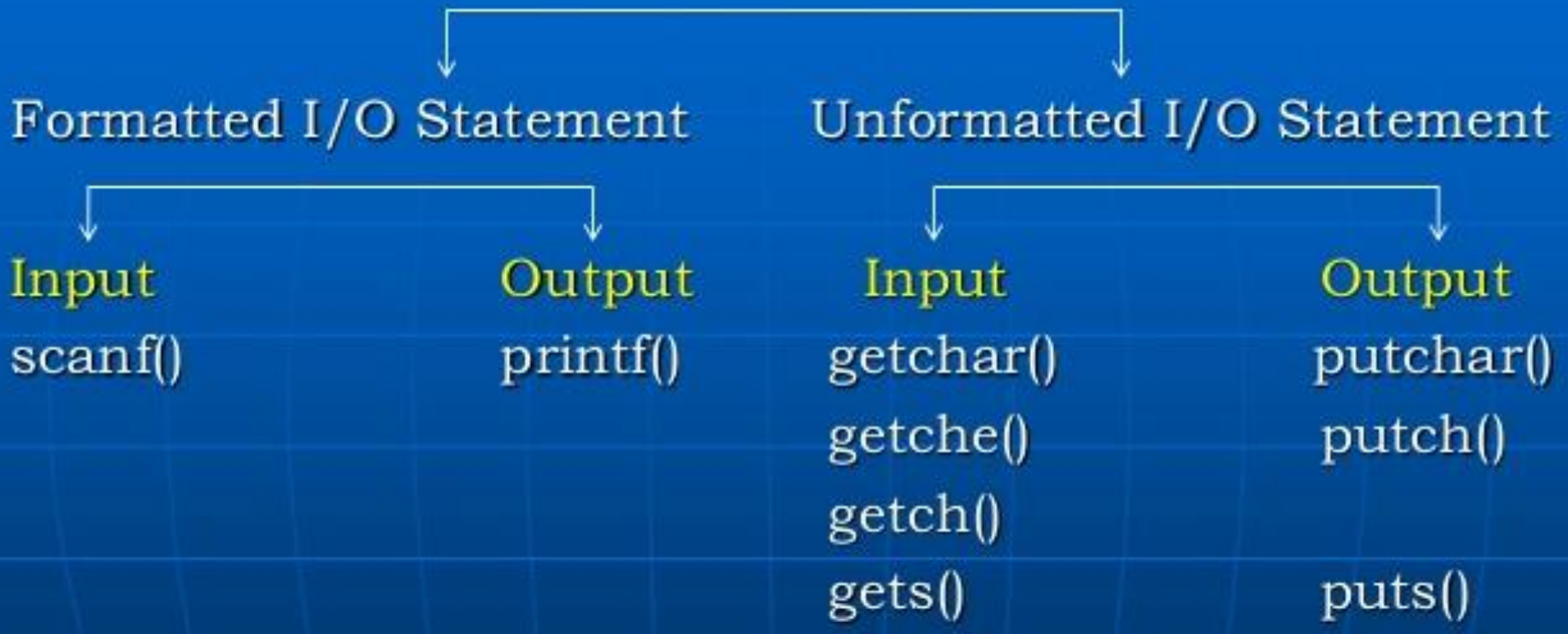
- For example,

```
float  avg=13.7;
printf(" Average = %f\n", avg);
printf("Over");
```

Program – Simple Interest

```
/* Write a program to calculate simple interest using the formula I = PRN/100
      where P is principal amount, R is rate of interest annually and N is duration in years */
#include <stdio.h>
main()
{
    int p,n;
    float r,i;
    clrscr();
    printf("Please give principal amount\n");
    scanf("%d", &p);
    printf("Please give rate of interest and number of years\n");
    scanf("%f %d", &r,&n);
    i=(p*r*n)/100;
    printf("Interest of amount Rs. %d at rate %f%% for %d years = %f\n",p,r,n,i);
}
```

Types of Standard I/O Statement



All these statements are stored in <stdio.h> header file.

Formatted Statements are those statements which facilitate the programmer to perform I/O operations in all type of data type available in C, whereas in Unformatted Statements, I/O operations can be performed in fixed format (date type).

Character Input

- C language provides getchar() function for getting character input from keyboard. As we know input/output is buffered, so getchar() function reads the next character from the input buffer and returns the ASCII value of the character. The syntax of getchar() is

  character_variable = getchar();

- The statements

  char c;

  c = getchar();

  will store the ASCII value of the character pressed from the keyboard.

Program- ASCII code

```
/* Write a program which accepts a character from keyboard
   and display it's ASCII code */
   #include <stdio.h>
   #include <conio.h>
   main()
   {
        char c;
        printf("Enter one character\n");
        c=getchar();
        printf("ASCII code of input character %c is = %d\n",c,c);
   }
```

getch(), getche() and getchar() functions

- getchar() function requires ENTER key to be pressed to terminate the input.
- Some times we want to simply read the characters (particularly) for reading the strings without terminating each character with ENTER key.
- In that case we can use **getch() or getche()** functions.
- **getch()** function does not echo the input character, while **getche()** function echoes the input character on the screen.
- The conio.h header file supports the above two functions. The prototype is

```
int getch();
int getche();
```

Character Output

- ‘C language provides putchar() function for displaying one character on the screen. It prints the character whose ASCII value is provided to it.
- The syntax of putchar() is

  putchar(character_variable);

  where, character_variable will be char type variable holding the ASCII value.
- The statement sequence,

  char c = ‘a’;

  putchar(c);

  will display the character ‘a’ on output screen.

Program – lowercase to uppercase

```
/* Write a program to convert given lowercase letter to uppercase
   character.
   uppercase ASCII value = lowercase ASCII value - 32 */
   #include <stdio.h>
   #include <conio.h>
   main()
   {
         char c;
         clrscr();
         printf("Enter any one lowercase character\n");
         c= getchar();
         printf("lowercase character = ");
         putchar(c);
         printf(" converted to uppercase character = ");
         putchar(c-32);
   }
```

Program – uppercase to lowercase

```
/* Write a program to convert given lowercase letter to uppercase
   character.
   lowercase ASCII value = uppercase ASCII value + 32 */
   #include <stdio.h>
   #include <conio.h>
   main()
   {
         char c;
         clrscr();
         printf("Enter any one uppercase character\n");
         c= getchar();
         printf("uppercase character = ");
         putchar(c);
         printf(" converted to lowercase character = ");
         putchar(c+32);
   }
```

Character Test Functions

Name	Test
`isalnum`	Is *c* an alphanumeric character?
`isalpha`	Is *c* an alphabetic character?
`isblank`	Is *c* a blank character (space or horizontal tab)?
`iscntrl`	Is *c* a control character?
`isdigit`	Is *c* a digit character?
`isgraph`	Is *c* a graphics character (any printable character except a space)?
`islower`	Is *c* a lowercase letter?
`isprint`	Is *c* a printable character (including a space)?
`ispunct`	Is *c* a punctuation character (any character except a space or alphanumeric)?
`isspace`	Is *c* a whitespace character (space, newline, carriage return, horizontal or vertical tab, or formfeed)?
`isupper`	Is *c* an uppercase letter?
`isxdigit`	Is *c* a hexadecimal digit character?

toupper() and tolower

```
/* Write a program to convert given lowercase letter to uppercase
   character and uppercase letter to lowercase letter. */
   #include <stdio.h>
   #include <ctype.h>
   main()
   {
        char c;
        printf("Enter any one character");
        putchar('\n');
        c= getchar();
        if(islower(c))
                putchar(toupper(c));
        else
                putchar(tolower(c));
   }
```

Program

```
#include <stdio.h>
#include <ctype.h>
main()
{
    char character;
    printf("Press any key\n");
    character = getchar();
    if (isalpha(character) > 0)/* Test for letter */
       printf("The character is a letter.");
    else
       if (isdigit (character) > 0)/* Test for digit */
          printf("The character is a digit.");
       else
          printf("The character is not alphanumeric.");
    }
```

Output

```
Press any key
h
The character is a letter.
Press any key
5
The character is a digit.
Press any key
*
The character is not alphanumeric.
```

Formatted Input

- scanf() function can be used to read various types of data in different formats.
- The format specification for integer data is %wd

  Where, w stands for width (number of digits), while d stands for integer number.
- For example,

  scanf(" %3d %4d", &num1,&num2);

  statement is used and if the input is given like

  234 4563

  then, 234 will be assigned to num1 and 4563 will be assigned to num2.

  But, if the input is 2344 563

  Then, num1=234 and num2=4
- For floating point numbers or double numbers, width is not necessary. To read **float** number, use **%f**, while for **double** number use **%lf.**

Program

```
main()
{
    int a,b,c,x,y,z;
    int p,q,r;
    printf("Enter three integer numbers\n");
    scanf("%d %*d %d",&a,&b,&c);
    printf("%d %d %d \n\n",a,b,c);
    printf("Enter two 4-digit numbers\n");
    scanf("%2d %4d",&x,&y);
    printf("%d %d\n\n", x,y);
    printf("Enter two integers\n");
    scanf("%d %d", &a,&x);
    printf("%d %d \n\n",a,x);
    printf("Enter a nine digit number\n");
    scanf("%3d %4d %3d",&p,&q,&r);
    printf("%d %d %d \n\n",p,q,r);
    printf("Enter two three digit numbers\n");
    scanf("%d %d",&x,&y);
    printf("%d %d",x,y);
}
```

Output

```
Enter three integer numbers
1 2 3
1 3 -3577
Enter two 4-digit numbers
6789 4321
67 89
Enter two integers
44 66
4321 44
Enter a nine-digit number
123456789
66 1234 567
Enter two three-digit numbers
123 456
89 123
```

%[characters] & %[^characters]

```
void main()
{
        char addr[80];
        printf("Enter address\n");
        scanf("% [a-z]", addr);
        printf("%s\n", addr);
}
```

```
Enter address
vadodara bits 391240
vadodara bits
```

```
void main()
{
        char addr[80];
        printf("Enter address\n");
        scanf("% [^\n]", addr);
        printf("%s\n", addr);
}
```

```
Enter address
Vadodara BITS 391240
Vadodara BITS 391240
```

Formatted Output

- We can format the output with printf() function. The format specification for integer data is %wd

  Where, w stands for width (number of digits) to be used for printing, while d stands for integer number.

- By default integer is printed right justified, we can change the justification by using hyphen (-) flag. Following table lists output flags :

Flag	Meaning
–	Left justified print, remaining spaces left blank
+	Print + or – before the number (signed number)
0	Leading zeros are printed
#o	0 before octal number
#x	0x before hex number
e	Scientific notation

Formatted Output (Cont)

- **Some examples**

Format	**Output**	
printf(“%d”,456);	456	
printf(“%5d”,456);	bb456	b stands for blank
printf(“%2d”,456);	456	
printf(“%05d”,456);	00456	
printf(“%-5d”,456);	456bb	b stands for blank
printf(“%+5d”,456);	b+456	b stands for blank
printf(“%#5o”,456);	b0710	b stands for blank
printf(“%#5x”,456);	0x1c8	

Formatted Output (Cont)

- The format specification for floating data is %w.pf

  Where, w denotes width, p denotes number of digits after decimal point, and f denotes float number. If **e** is used in place of **f**, then the number is displayed in scientific notation.

- If width is not specified, then the floating point number is displayed with six decimal point digits. Some examples,

	Format	**Output**	
	printf("%f",45.678);	45.678000	
	printf("%7.2f",45.678);	bb45.68	b stands for blank
	printf("%-7.2f",45.678);	45.68	
	printf("%7.2e",45.678);	4.57e+01	
	printf("%e",45.678);	4.567800e+01	

Program decimal to hex and octal

```
#include <stdio.h>
#include <conio.h>
    void main()
    {
            int x;
            clrscr();
            printf("Give the decimal number\n");
            scanf("%d",&x);
            printf("Decimal Number = %d\n",x);
            printf("Hexadecimal Number = %x\n",x);
            printf("Octal Number = %o\n",x);
    }
```

Program – Volume of Cylinder

```
/* Write a program to calculate volume of a cylinder. Formula V = pr2h */
#include <stdio.h>
#include <conio.h>
#define pi 3.1415
void main()
{
        int r,h;
        float v;
        clrscr();
        printf("Give radius and height of cylinder\n");
        scanf("%d%d", &r, &h);
        v = pi * r * r *h;
        printf("Volume of cylinder = %.2f\n", v);
}
```