CE143: COMPUTER CONCEPTS & PROGRAMMING

UNIT-2
Constants, Variables & Data Types in 'C'

N. A. Shaikh

nishatshaikh.it@charusat.ac.in

Topics to be covered

- Character set
- C tokens
- Keywords & Identifiers
- Constants
- Data types
- Variables
- Declaration of Variables
- Assigning Values to Variables
- Declaring a variable as Constant
- Defining Symbolic constants

Character set

- Like any other languages, C has it's own character set(collection of characters)
- It is used to construct C Program instructions

The characters in C are grouped into the following categories:

1. **Letters**
2. **Digits**
3. **Special characters**
4. **White spaces**

Character set

Letters	Digits
Uppercase A.....Z Lowercase a.....z	All decimal digits 0....9

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

White Spaces
Blank space
Horizontal tab
Carriage return
New line
Form feed

Trigraph Characters

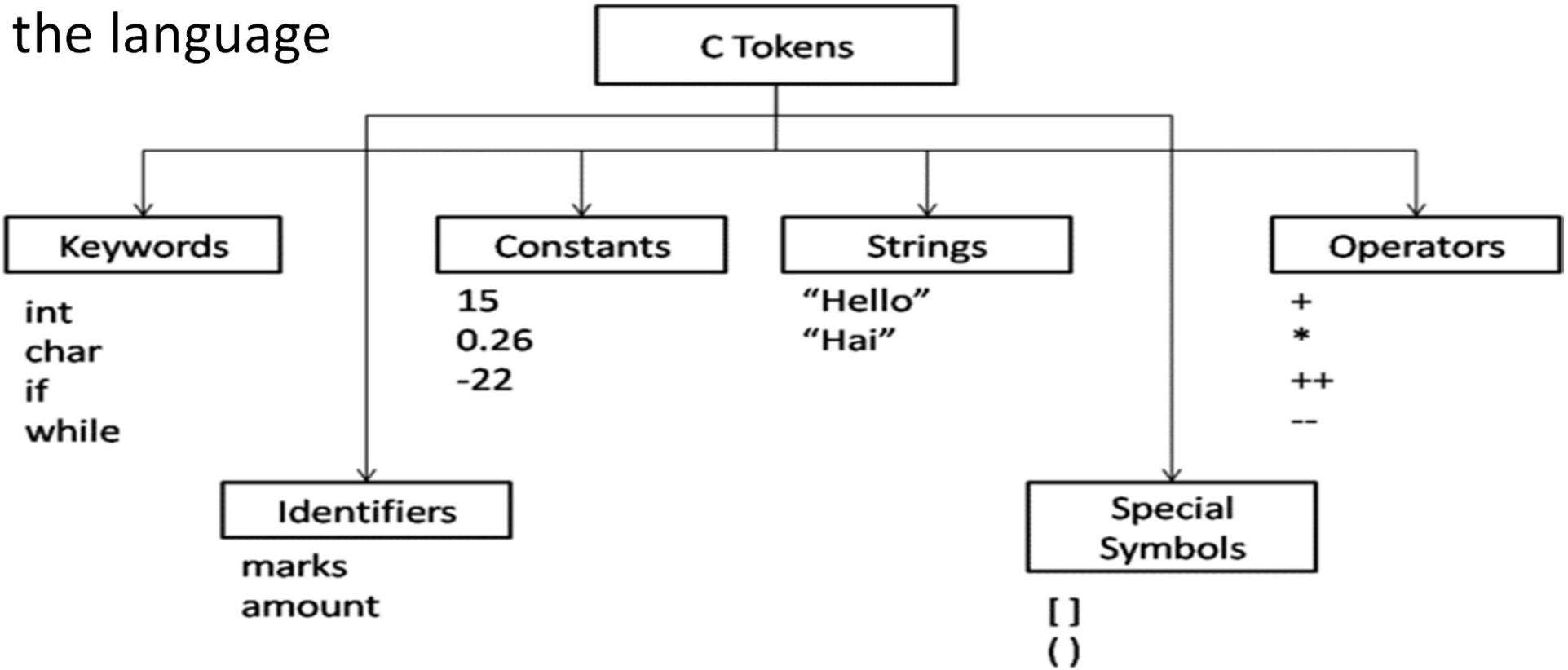
- Many non-English keyboards do not support all the characters
- ANCI C introduces the concept of “trigraph” sequences to provide a way to enter certain characters that are not available on some keyboards.

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

C Tokens

The smallest individual units in a C program are known as **C tokens.**

C programs are written using these tokens and the syntax of the language



Keywords

- Keywords are **predefined**, **reserved words** used in programming that have special meanings to the compiler.
- These **meanings can not be changed**
- It serve as basic building blocks for program statements
- Must be written in lowercase

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

Identifiers

- Identifiers refers to the **names of variables, functions and arrays.**
- These are user-defined names.
- An identifier can be composed of letters, underscore and digits.
- Both uppercase and lowercase letters are permitted.

Example:

int roll_no;

Identifiers

Rules for identifiers:

1. First character must be an alphabet or underscore
2. Must consist of only letters, digits or underscore
3. Only first 31 characters are significant
4. Cannot use a keyword
5. Must not contain white space

Identifiers

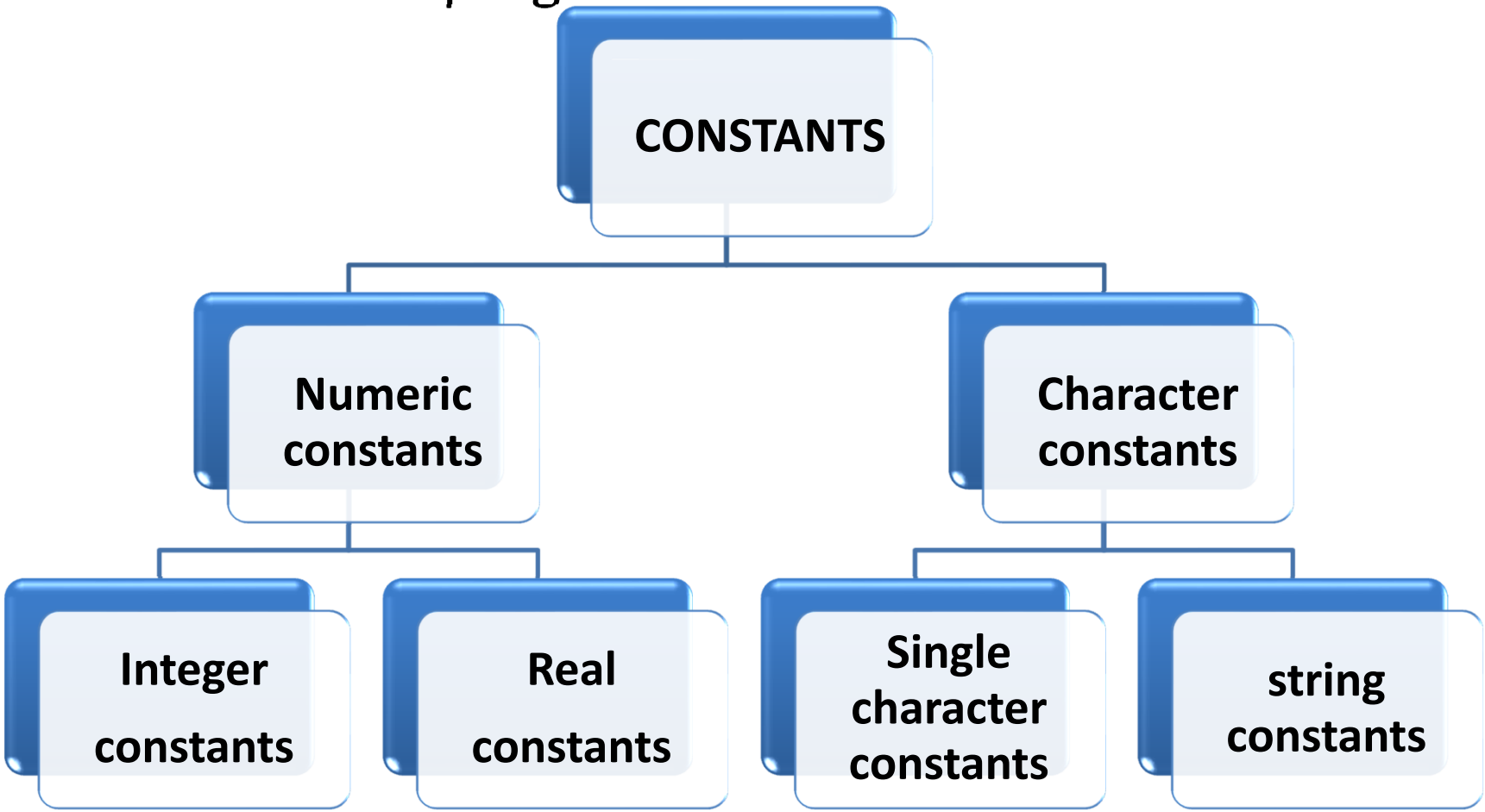
Example of valid identifiers	Example of invalid identifiers
total	**2sum** (starts with a numerical digit)
sum	**int** (reserved word)
average	**char** (reserved word)
_m_	**m+n** (special character, i.e., '+')
sum_1	**$sum** (cannot contain $)
student_name	**sum-salary**(cannot contain hyphen)
	student name (cannot contain space)
	total.sales (cannot contain .)

Keyword VS Identifier

Keywords	Identifier
Keyword is a pre-defined word.	The identifier is a user-defined word
It must be written in a lowercase letter.	It can be written in both lowercase and uppercase letters.
Its meaning is pre-defined in the c compiler.	Its meaning is not defined in the c compiler.
It is a combination of alphabetical characters.	It is a combination of alphanumeric characters.
It does not contain the underscore character.	It can contain the underscore character.

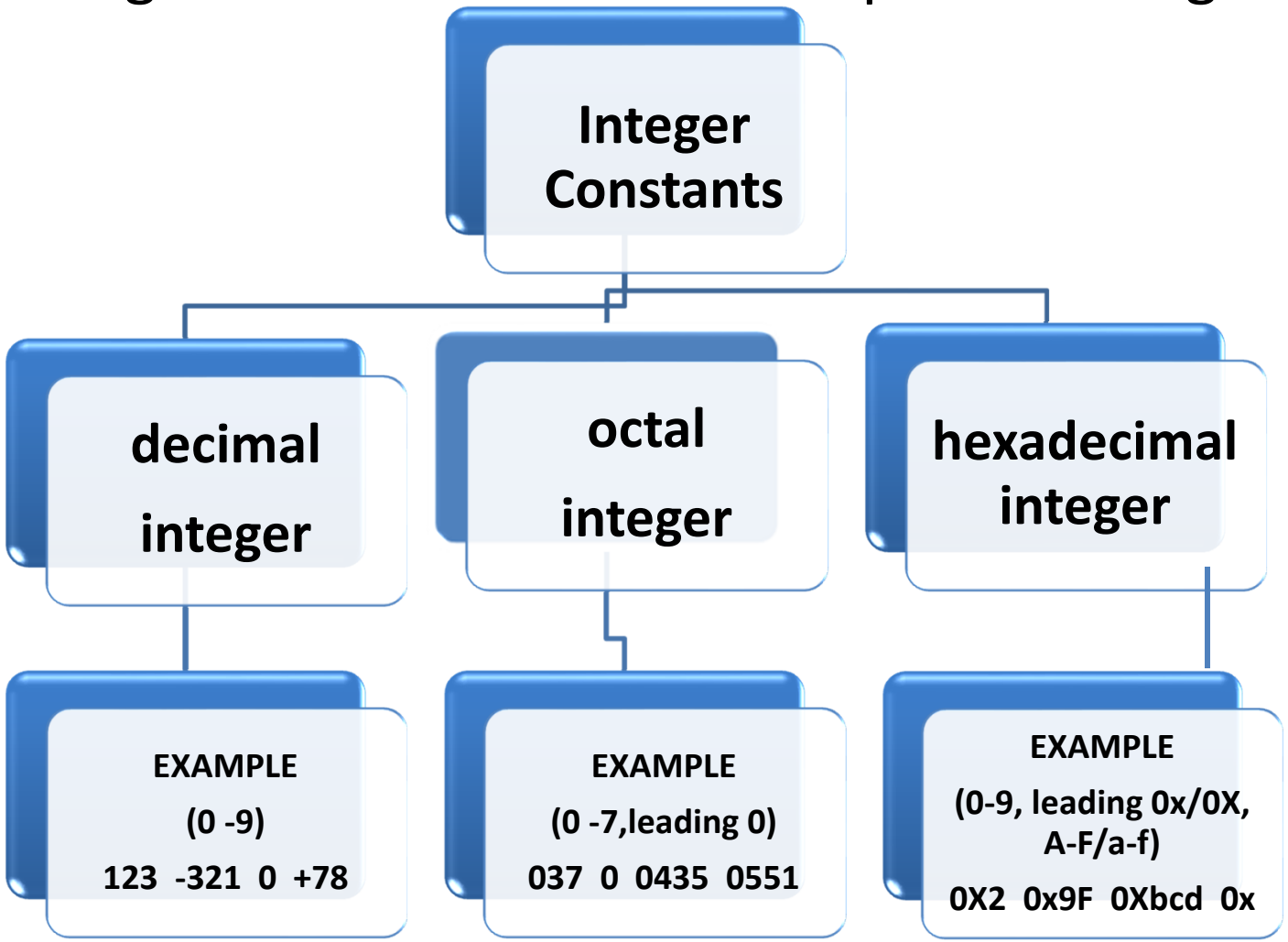
Constants

Constants refer to **fixed values that do not change** during the execution of a program



Integer Constants

An integer constant refers to a sequence of digits



Real Constants

- Real(or floating point) constants are represented by **numbers containing fractional parts** like 17.548

Example:

0.0083 -0.75 435.36 +247.0

- It may also be expressed in exponential(or scientific) notation like 2.1565e2(215.65)

Example:

0.65e4 12e-2 1.5e+5 3.18E3 -1.2E-1

- Embedded white space is not allowed

Single Character Constants

- A Single Character Constants(or simply character constant) contains a **single character enclosed within pair of single quotes [‘ ’].**

Example:

‘8’ ‘a’ ‘B’ ‘ ‘

String Constants

- A string constant is a **sequence of characters enclosed in double quotes [“ ”]**
- The characters may be letters, numbers, special characters and blank spaces

Example:

“0211” **“Well Done”** **“Hello!”** **“5+9”** **“X”**

Backslash Character Constants

- C supports some special Backslash Character Constants that are used in output functions.
- These character combinations are known as **escape sequences**

Constants	Meaning
'\a'	Audible alert (bell)
'\b'	Back space
'\f'	Form feed
'\n'	New line
'\r'	Carriage return
'\t'	Horizontal tab
'\v'	Vertical tab
'\''	Single quote
'\"'	Double quote
'\?'	Question mark
'\\'	Backslash
'\0'	Null character

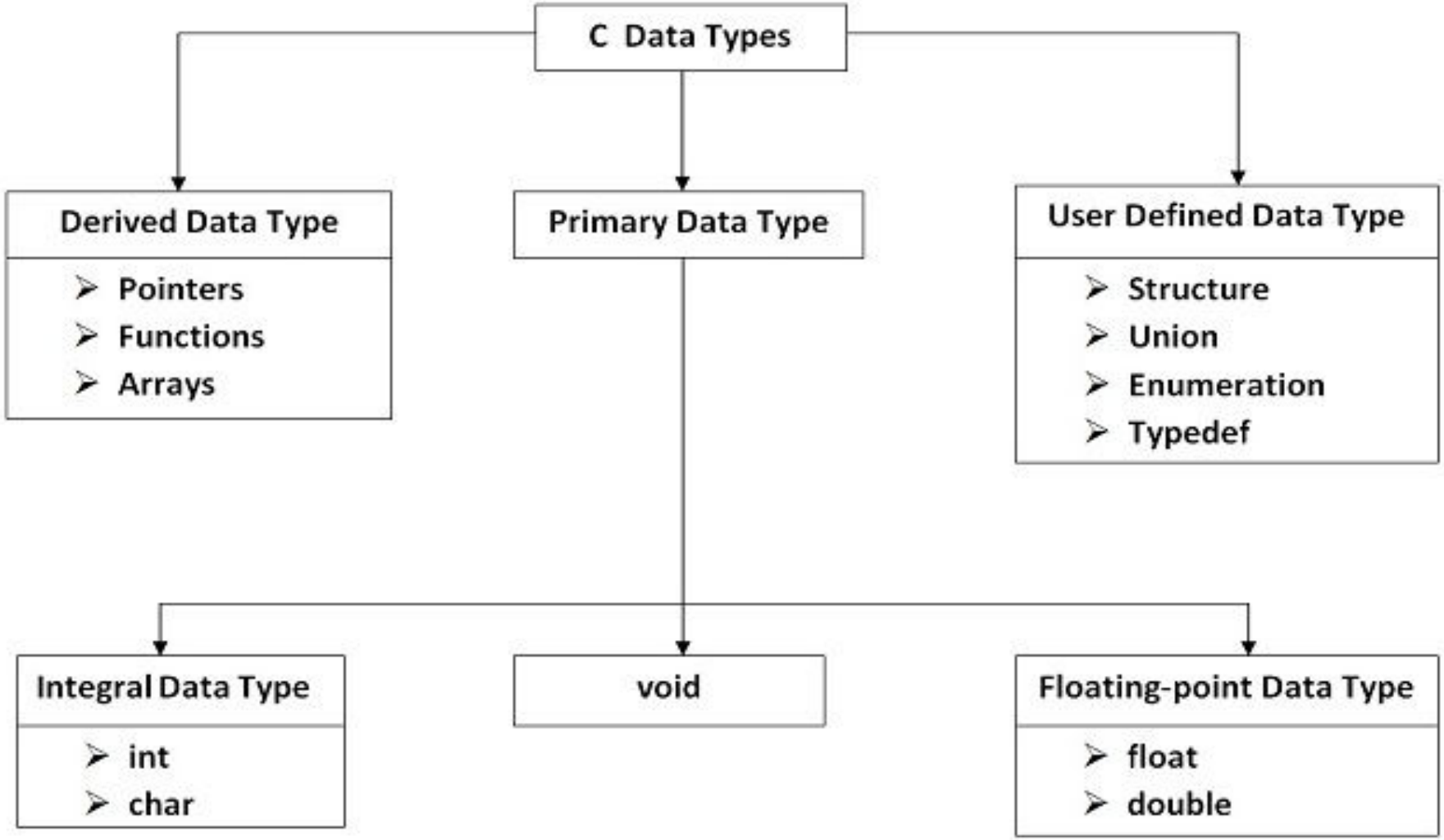
Data Types

- A data type is a classification that specifies **which type of value a variable has.**
- The type of a variable determines **how much space it occupies in storage**

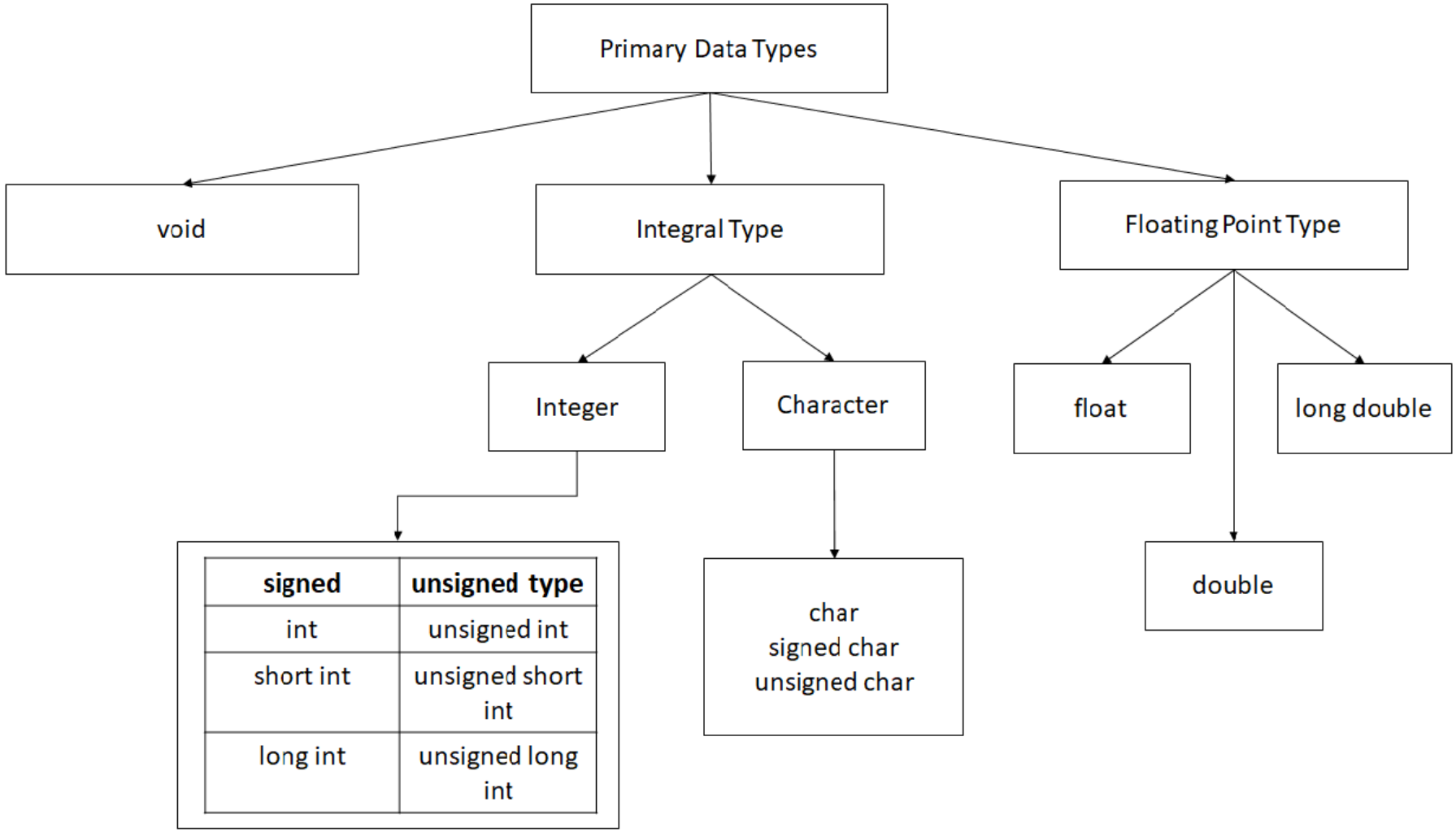
Three classes of data types:

1. **Primitive(Primary/Fundamental/ Basic/Built-in) data types**
2. **Derived data types**
3. **User-defined data types**

Data Types


C Data Types
Derived Data Type
Pointers
Functions
Arrays
Primary Data Type
User Defined Data Type
Structure
Union
Enumeration
Typedef
Integral Data Type
int
char
void
Floating-point Data Type
float
double

Primitive Data Type



signed	unsigned type
int	unsigned int
short int	unsigned short int
long int	unsigned long int

Integer Types

Integers are whole number(number without a fractional part)

Example:

234

- denoted by the keyword **int.**
- The size of an int is really compiler/processor dependent.
  - **2 bytes(16 bits)**
  - **4 bytes(32 bits)**
- using **sizeof(int)** is the best way to get the size of an integer for the specific system
- Size of **short int ≤ size of int ≤ size of long int**
- Can be signed or unsigned

Integer Types

Type	Size	Range	Control Specifier
int/signed int	16 bits(2 bytes)	-32,768 to 32,767	%d or %i
short int/ signed short int	16 bits(2 bytes)	-32,768 to 32,767	%hd or %hi
long int/ signed long int	32 bits(4 bytes)	-2,147,483,648 to 2,147,483,647	%ld or %li
unsigned int	16 bits(2 bytes)	0 to 65535	%u
unsigned short int	16 bits(2 bytes)	0 to 65535	%hu
unsigned long int	32 bits(4 bytes)	0 to 4,294,967,295	%lu

Size and Range of Data Types on a 16-bit Machine

Character Types

A single character/Any symbol enclosed in single quotation

Example:

'x' 'd'

- denoted by the keyword **char.**
- Represents a **single byte** (8 bits) of storage.
- Can be signed or unsigned

Type	Size	Range	Control Specifier
char/signed char	8 bits(1 byte)	-128 to +127	%c
unsigned char	8 bits(1 byte)	0 to 255	%c

Floating Point Types

It is used to store **fraction number(real numbers).**

Example:

12.45 -3.8

- denoted by the keyword **float.**
- All numbers are signed.
- Size of **float ≤ size of double ≤ size of long double**

Type	Size	Range	Control Specifier	Precision
float	4 byte(32 bits)	1.2E-38 to 3.4E+38	%f	6 decimal places
double	8 byte(64 bits)	2.3E-308 to 1.7E+308	%lf	15 decimal places
long double	10 byte(80 bits)	3.4E-4932 to 1.1E+4932	%Lf	19 decimal places

Void Types

- **Void type has no values**
- Generally, void is used to specify a **function which does not return any value**.
- We also use the void datatype to specify **empty parameters of a function**.

Data types and Their Keywords

Data type	Keyword equivalent
Character	char
Unsigned character	unsigned char
Signed character	signed char
Signed integer	signed int(or int)
Signed short integer	signed short int(or short int or short)
Signed long integer	signed long int(or long int or long)
Unsigned integer	unsigned int(or unsigned)
Unsigned short integer	unsigned short int(or unsigned short)
Unsigned long integer	unsigned long int(or unsigned long)
Floating point	float
Double-precision floating point	double
Extended double-precision floating point	long double

Variables

- A variable is a **data name** that is used to **store a data value**.
- Values of variables can be changed during execution
- It can be chosen by the programmer in a meaningful way

Example:

Average height Total Counter_1 class_strength

Variables

It may consist of letters, digits and the underscore(_)

Rules for Variables:

1. They must **begin with a letter or underscore**
2. ANSI standard recognizes a length of 31 characters(However, length should not be more than eight characters, since only first **eight characters are treated as significant** by many compilers)
3. Uppercase and lowercase are significant(**Total is not same as total or TOTAL**)
4. It should not be a **keyword**
5. **White space** is not allowed

Variables

Example of valid variable names	Example of invalid variable names
John	123
Delhi	%
X1	(area)
Sum1	25th
Ph_value	Char(char is keyword)
Distance	Prince$(Dollar sign is illegal)
	Group one(blank space is not permitted)

Declaration of Variables

- The declaration of variables must be done before they are used in the program
- **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

Syntax:

Data-type variable_name;

Example:

int count;

char gender;

int number,total;

float average;

double ratio;

Assigning values to Variables

Values can be assigned to variables using the assignment operator = as follows:

Syntax:

Variable_name = constant;

Example:

count=10;	**gender='f';**
number=5;	**total=50;**
average=35.548;	**ratio=4275.547**

Assigning values to Variables

we can assign a value to a variable at the time the variable is declared

Syntax:

Data-type variable_name = constant;

Example:

int count=10;

char gender='f';

float average=35.548;

double ratio=4275.547;

- The process of giving initial values to variables is called **initialization**

Program: Assigning values to Variables

```c
#include<stdio.h>
int main()
{
/*....DECLARATION....*/
float x,p;
double y,q;
unsigned k;

/*....DECLARATION AND ASSIGNMENT....*/
int m=54321;
long int n=1234567890;

/*....ASSIGNMENT....*/
x=1.234567890000;
y=9.87654321;
k=54321;
p=q=1.0;

/*....PRINTING....*/
printf("m=%d\n",m);
printf("n=%ld\n",n);
printf("x=%.12f\n",x);
printf("x=%f\n",x);
printf("y=%.12lf\n",y);
printf("y=%lf\n",y);
printf("k=%u p=%f q=%.12lf\n",k,p,q);

return 0;
}
```

C:\Users\NISH\Desktop\variable.exe

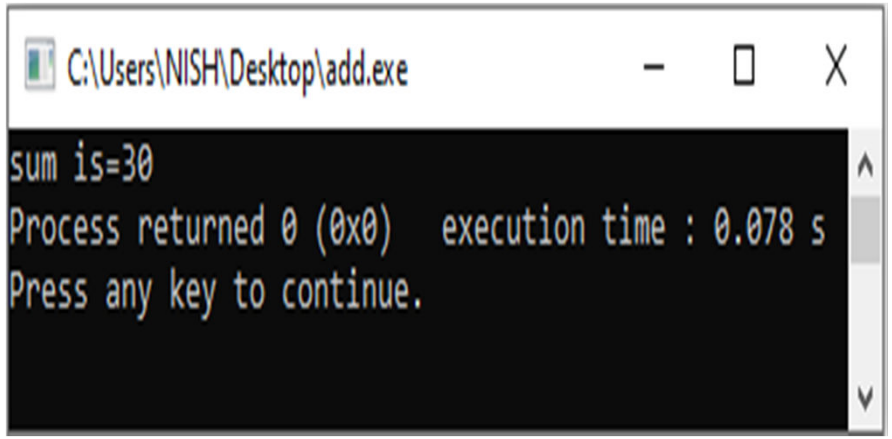
```
m=54321
n=1234567890
x=1.234567880630
x=1.234568
y=9.876543210000
y=9.876543
k=54321 p=1.000000 q=1.000000000000
```

Program: Addition of two numbers

```c
#include<stdio.h>

int main()
{
    int a,b,c;
    a=10; b=20;
    printf("sum is=%d",c=a+b);

return 0;
}
```



Local and Global Variables

Three places where variables can be declared,

- Inside a function or a block (**local variables**)
- Outside of all functions (**global variables**)
- In the definition of function parameters (formal parameters)

Local Variable	Global Variable
Declared inside the function	Declared outside the main function
Accessed only by the function they are declared in.	Accessed by all functions in the program.
They are alive within the function they are declared	They are alive throughout the program
If it is not initialized, a garbage value is stored	If it is not initialized zero is stored as default.

Program: Local and Global Variables

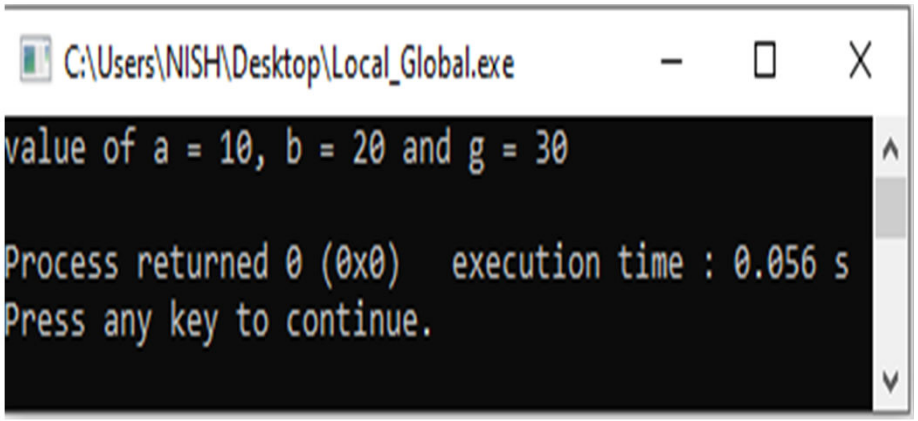
```c
#include <stdio.h>

/* global variable declaration */
int g;

int main ()
{
 /* local variable declaration */
int a, b;

 /* actual initialization */
a = 10; b = 20;
g = a + b;
printf ("value of a = %d, b = %d and g = %d\n", a, b, g);

return 0;
}
```



Program: Local and Global Variables

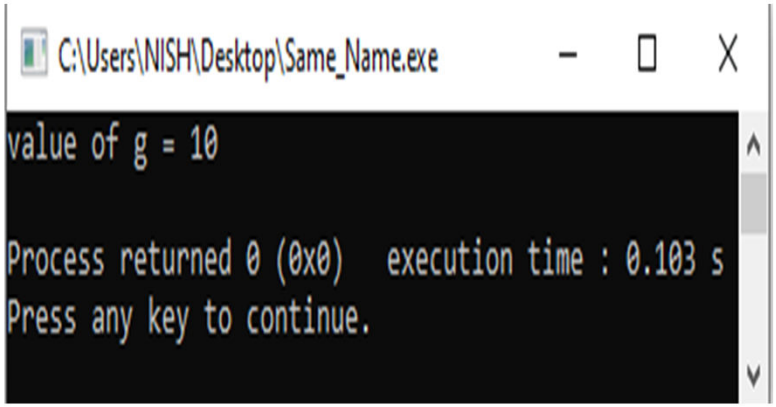
A program can have same name for local and global variables but the value of local variable inside a function will take preference.

```c
#include <stdio.h>

/* global variable declaration */
int g = 20;

int main ()
{
/* local variable declaration */
 int g = 10;
 printf ("value of g = %d\n", g);

 return 0;
}
```



Reading Data from Keyboard

Another way of giving values to variables is to input data through keyboard using **scanf function**

Syntax:

scanf("control string",&variable1,&variable2,..);

where,

Control string:format of data being received

&(ampersand symbol):specifies the variable name's address

Example:

scanf("%d",&number);

Program:Reading Data from Keyboard

```c
#include<stdio.h>

int main()
{
 int a,b,c;

 printf("Enter the value of a & b:\n");
 scanf("%d %d",&a,&b);

 c=a+b;

 printf("Sum is=%d",c);

 return 0;
}
```

C:\Users\NISH\Desktop\scanf.exe

```
Enter the value of a & b:
5
4
Sum is=9
Process returned 0 (0x0)   execution time : 2.607 s
Press any key to continue.
```

Reading Data from Keyboard

C Exercises: Perform addition, subtraction, multiplication and division of two numbers

Declaring a variable as constant

We can define constants in two ways as shown below:

1. **Using a const keyword**
2. **Using #define preprocessor directive**

Using a const keyword

We may like the value of certain variables to **remain constant** during the execution of a program

Syntax:

const datatype constantName=value;

Example:

const int a=50;

- This tells the compiler that the value of the int variable "a" must not be modified by the program
- The default value of constant variables are zero.

Using a const keyword

```c
#include <stdio.h>

int main()
{
int i = 9 ;
const int x = 10 ;

i = 15 ;
x = 100 ; // creates an error

printf("i = %d\n x = %d", i, x ) ;
return 0;
}
```

Line	Message
	=== Build file: "no target" in "no project" (compiler: unknown) ===
	In function 'main':
10	error: assignment of read-only variable 'x'
	=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

Using #define preprocessor directive

Declaring symbolic constants

Syntax:

#define symbolic-name value_of_constant

Example:

#define PI 3.14159

#define MAX 100

#define PASS_MARK 23

Using #define preprocessor directive

```c
#include<stdio.h>
#define PI 3.14

void  main()
{
float a;
a=100*PI;
printf("%f",a);
}
```

C:\Users\NISH\Desktop\const.exe

```
314.000000
Process returned 10 (0xA)   execution time : 0.218 s
Press any key to continue.
```

Declaring a variable as constant

#define	Const
Substitution of macro takes place at preprocessing stage.	Substitution of value takes place at runtime.
No memory required	Depending on the type of data, memory is required.
No semicolon is required.	Requires a semicolon.
Execution is fast and efficient.	Slower compared to #define
No need to mention data type	Data type must be specified

User-Defined Data Type

1. **typedef**
2. **Enumeration**
3. **Structure**
4. **Union**

User-Defined Data Type: Typedef

- typedef keyword is used to assign a **new name** to a existing data-type.

Syntax:

typedef type-name new-name;

Example:

typedef unsigned int uint;

uint i,j;

User-Defined Data Type: Typedef

```c
#include <stdio.h>

int main()
{
    typedef unsigned int ui;
    ui i = 5, j = 8;
    printf("i = %d\n", i);
    printf("j = %d\n", j);
    return 0;
}
```

C:\Users\NISH\Desktop\typedef.exe

```
i = 5
j = 8

Process returned 0 (0x0)   execution time : 0.136 s
Press any key to continue.
```

User-Defined Data Type: Enumeration

- User-defined data type with discrete set of possible values

A **weekdays** is something from:
{Monday, Tuesday, Wednesday, Thursday, Friday}

Syntax:

enum enum_name {const1, const2, …constn};

Example:

enum color {Red, Blue, Green, Black};

User-Defined Data Type: Enumeration

```c
#include<stdio.h>

enum week{Mon, Tue, Wed, Thur, Fri=8, Sat, Sun};

int main()
{
    enum week day;

    day = Wed;
    printf("Value of Wed=%d\n",day);

    day = Sat;
    printf("Value of Sat=%d",day);

    return 0;
}
```

C:\Users\NISH\Desktop\enum.exe

```
Value of Wed=2
Value of Sat=9
Process returned 0 (0x0)   execution time : 0.169 s
Press any key to continue.
```

End of Unit-02