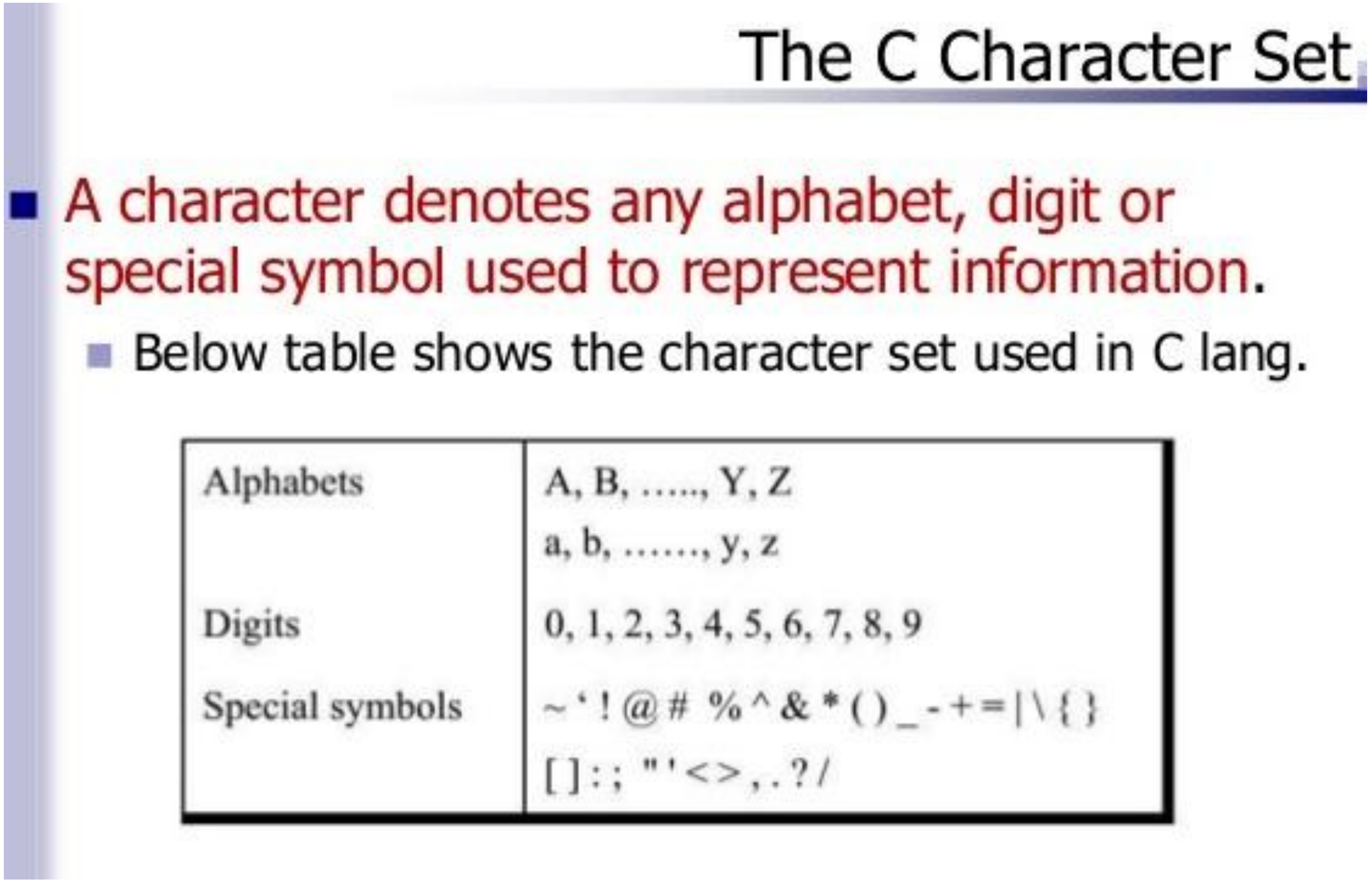
UNIT – 2
Elements of C

C Programming (CSC115)
BSc CSIT First Semester (TU)

Prepared by:
Dabbal Singh Mahara
ASCOL (2025)

CHARACTER SET

- The set of characters that are used to form words, numbers and expressions in C is called C character set.
- The combination of these characters form words, numbers and expressions.
- The C character set is grouped into the following four categories:
  1) Letters or alphabets
  2) Digits
  3) Special Characters
  4) White Spaces



The C Character Set

- A character denotes any alphabet, digit or special symbol used to represent information.
  - Below table shows the character set used in C lang.

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

- Letters
  - Uppercase: A……Z
  - Lowercase: a……z
- Digits
  - All decimal digits: 0……9
- Special Characters
  - , comma
  - . period
  - ; semicolon
  - : colon
  - ? question mark
  - ‘ apostrophe
  - “ quotation mark
  - ! exclamation mark
  - | vertical bar

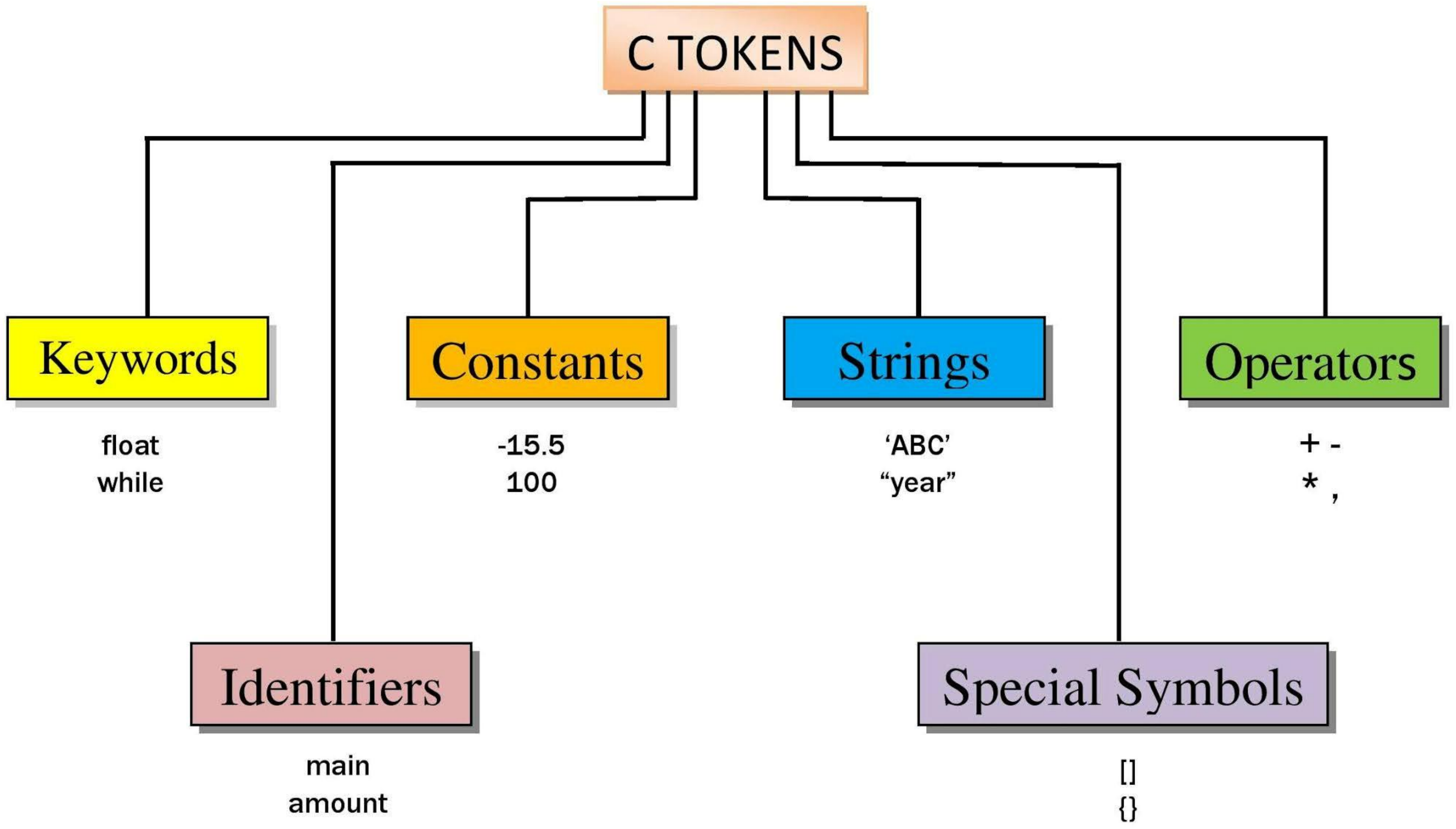
- / slash
- \ backslash
- ~ tilde
- _ underscore
- $ dollar sign
- % percent sign
- & ampersand
- ^ caret
- * asterisk
- - minus sign
- + plus sign
- < opening angle bracket (or less than sign)
- > closing angle bracket (or greater than sign)
- (left parenthesis
-) right parenthesis

- [left bracket
-] right bracket
- { left brace
- } right brace
- # number sign

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

C TOKENS

- In a C program the smallest individual units are known as C tokens.
- C tokens are the basic buildings blocks in C language which are constructed together to write a C program.
- These are the basic elements recognized by the C compiler.


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

KEYWORDS

- Every C word is classified as either a **keyword** or an **identifier**.
- Keywords are predefined words in C programming language.
- All keywords have fixed meaning and these meanings cannot be changed.
- Keywords are also called reserved words because they are used for pre-defined purposes and cannot be used as identifiers.
- There are total of 32 keywords in C.

ANSI C KEYWORDS

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

- Every word used in C program to refer to the names of variables, functions, arrays, pointers and symbolic constants are called identifiers.
- These are user-defined names and consist of a sequence of letters and digits, with a letter as the first character.
- Both uppercase and lowercase letters can be used, although lowercase letters are commonly preferred.
- The underscore character can also be used to link between two words in long identifiers.

RULES FOR IDENTIFIERS

1) First character must be an alphabet (or underscore).
2) Must consist of only letters, digits or underscore.
3) Must not contain white space. Only underscore is permitted.
4) Keywords cannot be used.
5) Only first 31 characters are significant.
6) It is case-sensitive, i.e. uppercase and lowercase letters are not interchangeable.

TEST

Determine which of the following are valid identifiers? If invalid, explain why?

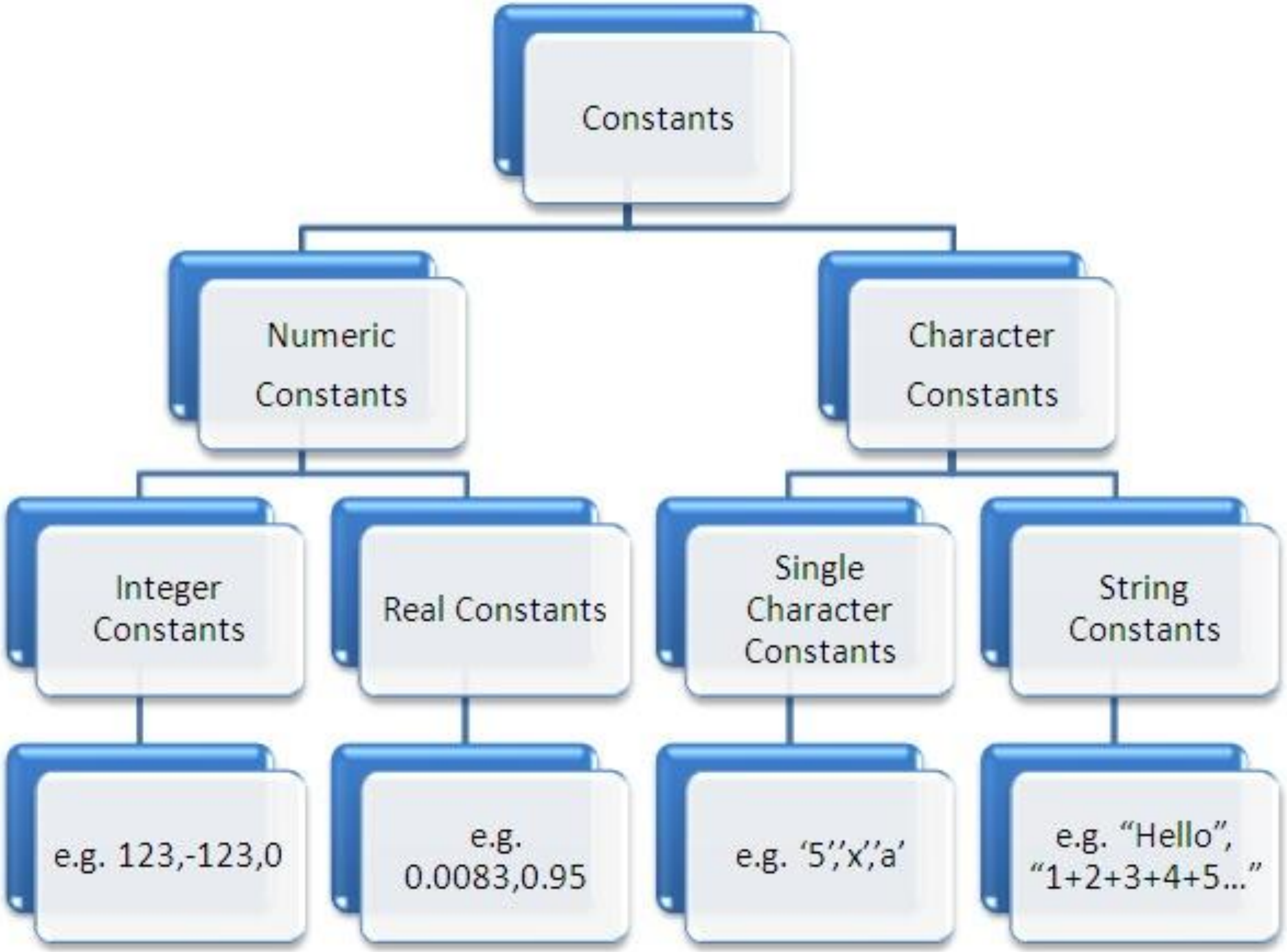
(a) keyword	(b) n1+n2
(c) file_3	(d) #ph_no
(e) double	(f) Rs2000
(g) doubles	(h) first_name
(i) 2var	(j) first-name
(k) $1000	(g) return
(h) _1	(j) inta
(k) _	(l) __

EXAMPLES: IDENTIFIERS

```
#include<stdio.h>
int main()
{
    int length, breadth;
    int area;
    return 0;
}
```

CONSTANTS

- Constants in C refer to fixed values that do not change during the execution of a program.
- C constants can be divided into different categories:
  - Numeric Constants
    - Integer Constants
    - Real Constants
  - Character Constants
    - Single Character Constants
    - String Constants


Constants
Numeric Constants
Character Constants
Integer Constants
Real Constants
Single Character Constants
String Constants
e.g. 123,-123,0
e.g. 0.0083,0.95
e.g. '5','x','a'
e.g. "Hello", "1+2+3+4+5..."

INTEGER CONSTANTS

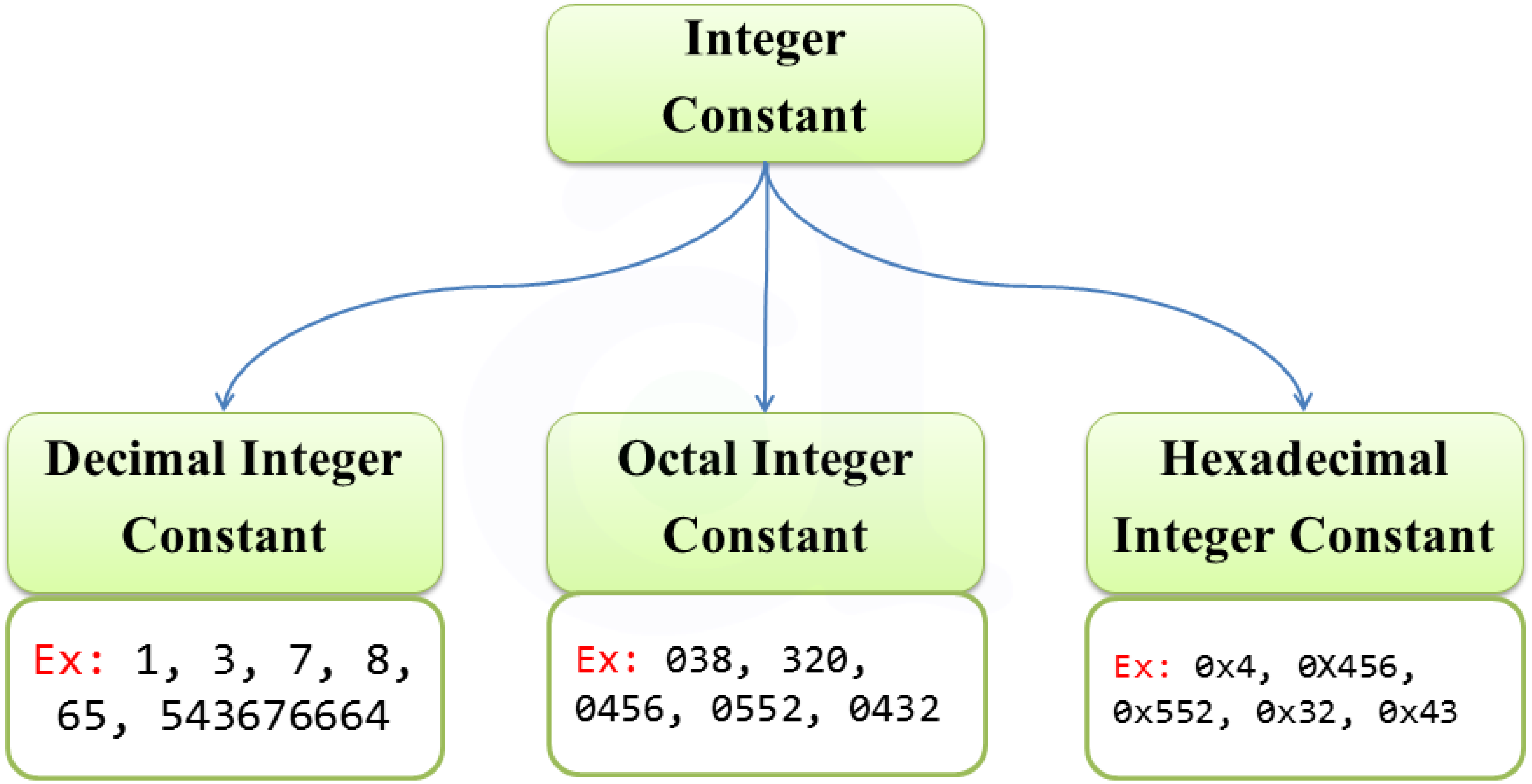
- Integer Constant refers to a sequence of digits (at least one digit) with no decimal point and either positive or negative.
- If no sign precedes an integer constant, it is assumed to be positive.
- No commas or blank spaces are allowed within the integer constant.
- E.g. 0, 123, +365, -555, etc.
- $1000, 20,000, 22 329, are not allowed.

INTEGER CONSTANTS…

- Three types:
  - Decimal Integer Constants
  - Octal Integer Constants
  - Hexadecimal Integer Constants

Decimal Integer Constants: Decimal integers consist of a set of digits, 0 through 9, preceded by an optional – or + sign. E.g. +78, 0, 123, etc.

Octal Integer Constants: An octal integer constant consists of any combination of digits from the set 0 through 7, with a leading 0. E.g. 056, 0, 0123, etc



Ex: 1, 3, 7, 8, 65, 543676664

Ex: 038, 320, 0456, 0552, 0432

Ex: 0x4, 0X456, 0x552, 0x32, 0x43

INTEGER CONSTANTS…

Hexadecimal Integer Constants:

- A hexadecimal integer constant consists of any combination of digits from the set 0 to 9, and alphabets A through F or a through f with a leading 0X or 0x.
- The letters A through F represent the numbers 10 through 15.
- E.g. 0X2, 0x9F, 0Xabc, 0x0, etc.

- NOTE: Octal and Hexadecimal numbers are rarely used in programming.

```
#include <stdio.h>
#include <conio.h>

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

   a=10;
   b=010;
   c=0xFF;
   printf("Decimal integer is %d",a);
   printf("\nOctal integer is %d",b);
   printf("\nHexadecimal integer is %d",c);
   getch();
}
```

REAL CONSTANTS

- Integer numbers are inadequate to represent quantities that vary continuously, such as distances, heights, temperatures, prices and so on.
- These quantities are represented by numbers having fractional parts like 3.14.
- Such numbers are called real (or floating point) constants.
- E.g. 0.00123, -0.25, +125.0, 23.35, etc
- Real constants can be written into two forms: fractional form and exponential form.

FRACTIONAL FORM CONSTANTS

- Fractional form constants must have at least one digit and a decimal point.
- It can either be positive or negative but the default sign is positive.
- Commas or blank spaces are not allowed within a real constant.
- E.g. +23.45, 456.0, -23.35, -5544.312, etc are in fractional form.

EXPONENTIAL FORM CONSTANTS

- In exponential form of representation, the real constant is represented in two parts as, mantissa e exponent
- The digits before ‘e’ is called mantissa and after is called exponent.
- The mantissa part may have a positive or negative sign, but the default is positive.
- The exponent must have at least one digit (must be integer), which can be either positive or negative.
- E.g. -3.2e-4 implies [$-3.2*10^{-4}$]

  -0.2e+3 implies [$-0.2*10^{3}$]

```
#include <stdio.h>
#include <conio.h>

int main()
{
float x;

x=-3.2e-4;
printf("\nFloating point constant is %f",x);
getch();
}
```

SINGLE CHARACTER CONSTANTS

- A single character constant (or simply character constant) contains a single character alphabet, a digit or a special symbol enclosed within a pair of single quote marks.
- E.g. ‘5’, ‘X’, ‘;’ ‘ ’, etc.
- Character constants have integer values known as ASCII values.
- NOTE: The character constant ‘5’ is not the same as the number 5.
- NOTE: ‘A’ is a valid character constant but ‘AA’ is not.

```
#include <stdio.h>
#include <conio.h>

Int main()
{
int n = 4;
char c = ‘4’;

n = n + 5;                          //n=4+5=9
c = c + 5;                          //c=52+5=57
printf(“Integer = %d”,n);
printf(“\nCharacter = %d”,c);
getch();
}
```

```
#include <stdio.h>
#include <conio.h>
int main()
{
char x;
x='A';
printf("The character %c's ASCII value is:%d", x, x);
getch();
}
```

Note: To find the ASCII value of \, we have to write a='\\';. Writing a='\'; gives error. Similarly to find out ASCII values of enter and backspace keys we have to write '\n' and '\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 space. However, it does not have an equivalent ASCII value.
- E.g. “Hi!”, “2011”, “WELL DONE”, “?...!”, “5+3”, “X”, etc.

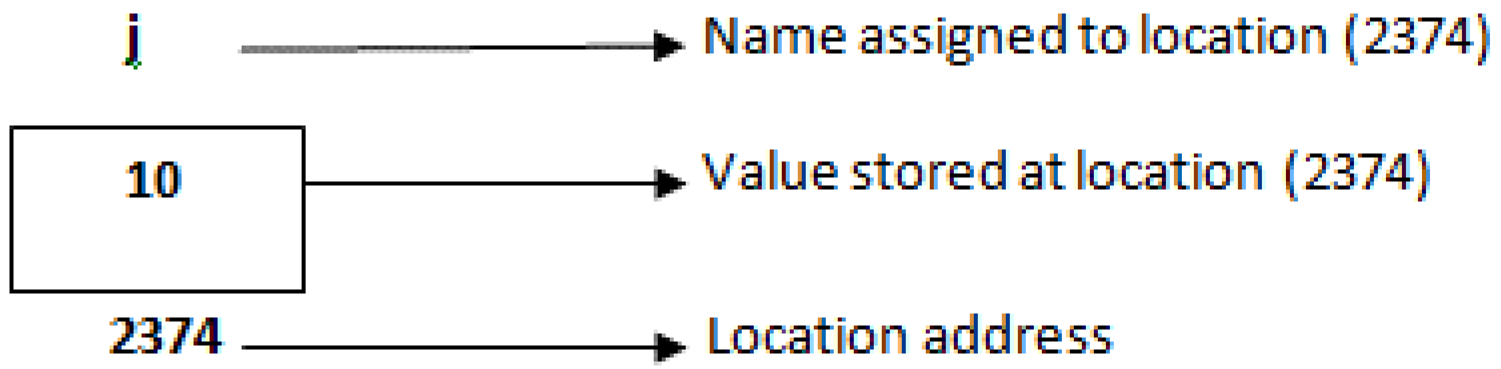
- NOTE: A character constant (e.g. ‘X’) is not equivalent to the single character string constant (e.g. “X”).
  Also, “5+3” is a string rather than an arithmetic operation

VARIABLES

- A variable is a data name that is used to store a data value.
- Its value can be changed, and it can be reused many times.
- A **variable** represents name of the memory location.
- It is a way to represent memory location through symbol so that it can be easily identified.
- Since a variable is an identifier, the rules for naming variables are similar to those of identifiers.
- A variable name can be chosen by the programmer in a meaningful way so as to reflect its function or nature in the program.
- E.g. Average, sum, counter, first_name, etc.
- 123, (area), %, 25th, Price$, blood group, etc. are not allowed.

VARIABLE: EXAMPLE

```
int j = 10;
```


j
Name assigned to location (2374)
10
Value stored at location (2374)
2374
Location address

VARIABLE DECLARATION

- Any variable should be defined before using it in a program.
- The variables are defined or declared using following syntax:

```
data_type variable_name;
```

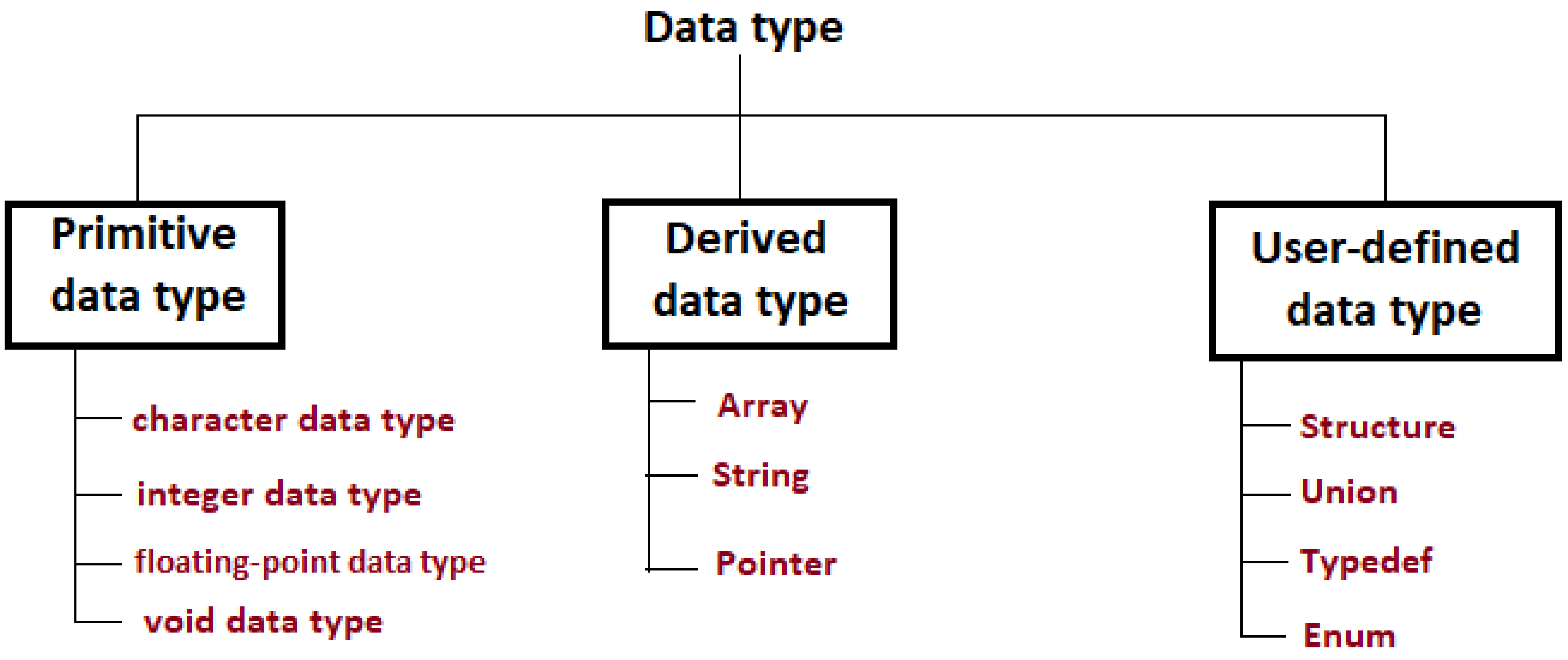
  where variable_name is the name of the variable.

  - E.g.

```
int a;
float radius;
char gender;
int  x1,x2,x3;
```

DATA TYPES

- A data type specifies the type of data that a variable can store such as integer, floating, character, etc.
- C language is rich in its data types. There are other varieties of data types available in C, each of which may be represented differently within computer's memory.
- ANSI C supports 3 classes of data types:
  - Primary (or fundamental) data types
  - Derived data types
  - User-defined data types


Data type
Primitive data type
character data type
integer data type
floating-point data type
void data type
Derived data type
Array
String
Pointer
User-defined data type
Structure
Union
Typedef
Enum

INTEGER TYPES

- Integers are whole numbers (positive, negative and 0), i.e. non-fractional numbers.
- If integer data is small then we can use keyword **short** or **short int** and to store long integer number we can use **long** or **long int** keyword and to store very long number we can use **long long** or **long long int** keyword.

FLOATING POINT TYPES

- Floating point types represent fractional numbers (i.e. real numbers).
- The data type qualifier is float.
- E.g. Variables are defined as- float a;
- Floating numbers reserve 32 bits (i.e. 4 bytes) of storage, with 6 digits of precision.
- When the accuracy provided by a float number is not sufficient, the type double can be used to define the number.
- A double data type number uses 64 bits (8 bytes) giving a precision of 14 digits. These are known as *double precision numbers*.
- To extend the precision further, long double can be used which uses 80 bits (10 bytes) giving 18 digits of precision.

CHARACTER TYPE

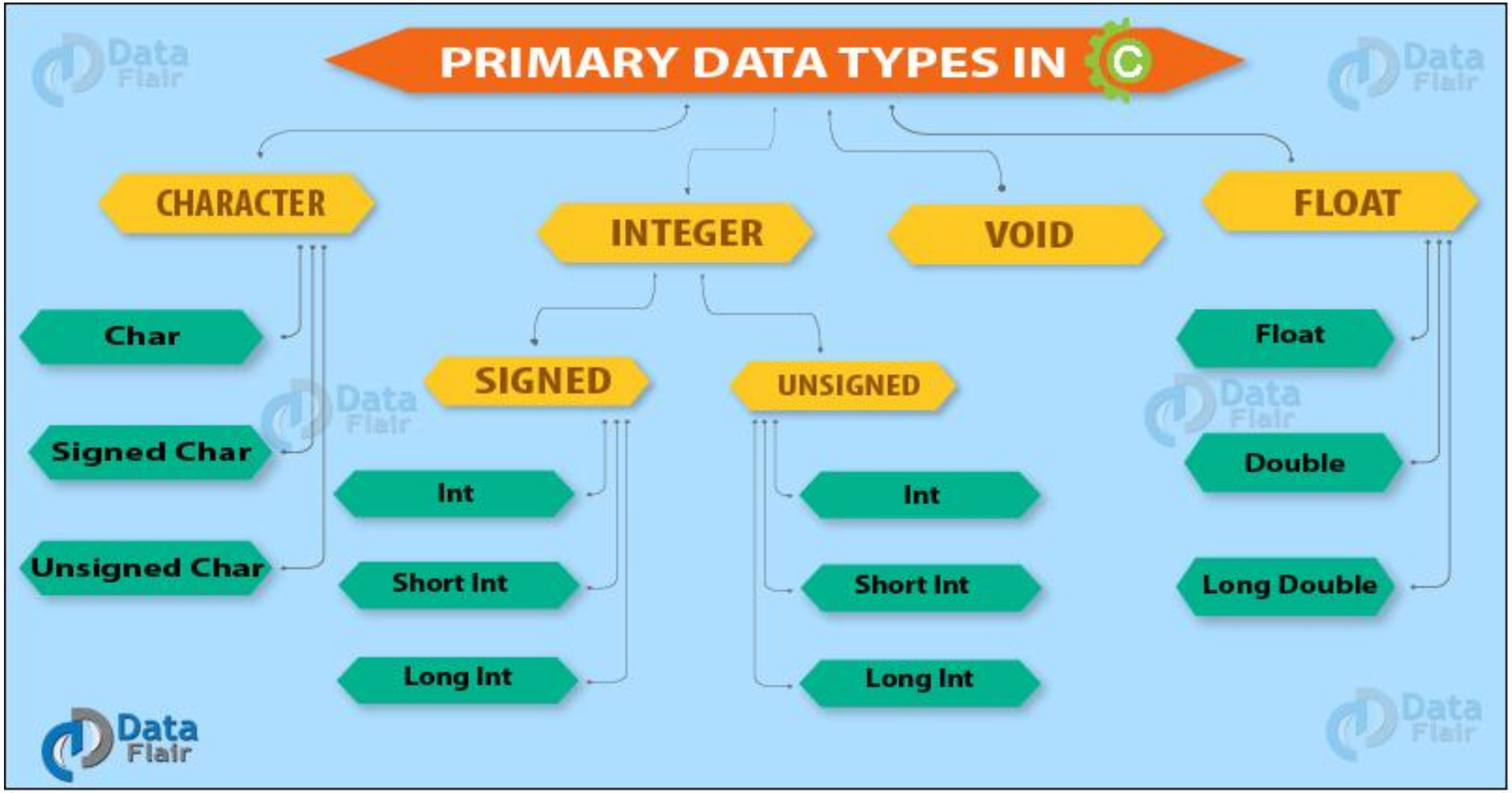
- A single character can be defined as a character type data.
- Characters are stored in 8 bits (1 byte).
- The data type qualifier is char.
- The qualifier signed or unsigned may be used with char.
- The unsigned char has values between 0 and 255 while signed char has values from -128 to 127.
- The conversion character is c.

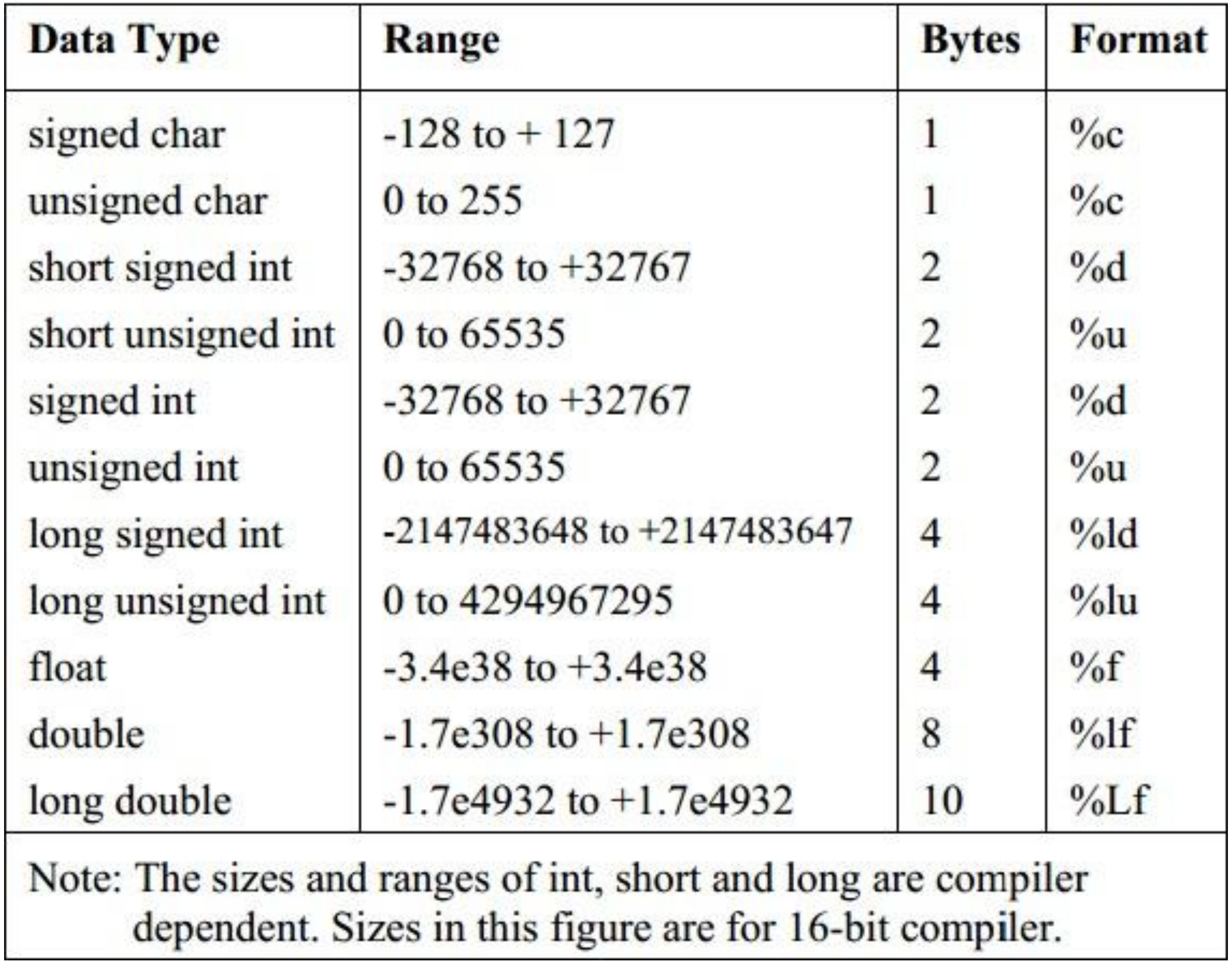
- In character data type, each character is represented by an ASCII (American Standard Code for Information Interchange) value internally.
- For e.g. the character ‘A’ is represented by 65, ‘B’ by 66 and so on for ‘Z’ by 90 and similarly for others.
- When a character is displayed using the conversion character d, it will display the ASCII value; and when it is displayed using the conversion character c, it will display the character.

```
# include <stdio.h>
# include <conio.h>
int main()
{
char c, d;
c=‘A’;
d=‘B’;
printf(“The character is %c”, c);
printf(“\nThe ASCII value of character is %d”, c);
printf(“\n\nThe ASCII value of %c is %d”, d,d);
getch();
}
```

VOID TYPE

- The void type has no values.
- It is usually used to specify a type of function when it does not return any value to the calling function.


PRIMARY DATA TYPES IN C
CHARACTER
INTEGER
VOID
FLOAT
Char
Signed Char
Unsigned Char
SIGNED
UNSIGNED
Int
Short Int
Long Int
Int
Short Int
Long Int
Float
Double
Long Double
Data Flair



Data Type	Range	Bytes	Format
signed char	-128 to + 127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32768 to +32767	2	%d
short unsigned int	0 to 65535	2	%u
signed int	-32768 to +32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to +2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to +1.7e4932	10	%Lf

Note: The sizes and ranges of int, short and long are compiler dependent. Sizes in this figure are for 16-bit compiler.

Exercise Problems

- Write a C program to add two integers and display the result.
- Write a program to calculate simple interest using the formula SI=(P*t*r)/100.

ESCAPE SEQUENCES

- An escape sequence is a non-printing character used in C.
- It is a character combination consisting of a backslash (\) followed by a letter or a digit.
- Escape sequences always represent single characters, even though they are written in terms of two or more characters.
- Escape sequences has a single ASCII value.
- Escape sequences are useful for formatting input and output.

Escape Sequence	Use
\a	Audible Alert or beep sound
\b	Backspace delete on character to the left
\n	Move cursor to the Next or New line of the screen
\v	Vertical tab
\t	Horizontal tab
\’	Single quote
\”	Double quote
\\	Backslash
\0	Null character

```
#include<stdio.h>
#include<conio.h>

 int main()
{
printf("\aHello \t World \n");
printf("He said \"Hello\" ");
getch();
}
```

printf() AND scanf() FUNCTIONS

- printf() and scanf() functions are inbuilt library functions in C programming language which are available in C library by default.
- These functions are declared and related macros are defined in “stdio.h” which is a header file in C language.
- We have to include “stdio.h” file in program to make use of these printf() and scanf() library functions in C language.

printf()

- In C programming language, printf() function is used to print the ("character, string, float, integer, octal and hexadecimal values") onto the output screen.
- We use printf() function with %d format specifier to display the value of an integer variable.
- To generate a newline,we use "\n" in C printf() statement.
- The general form of printf() is,

```
printf("control string", arg1, arg2, …, argN);
```

scanf()

- In C programming language, scanf() function is used to read character, string, numeric data from keyboard
- The general form of scanf() is,

  scanf("control string", arg1, arg2, …, argN);

```
#include<stdio.h>
 int main(){
     int radius;
  printf("Enter the radius of circle");
  scanf("%d",&radius);
    ………………………………….
}
```

Exercise Problems

- Write a program that takes radius of circle as input and displays the area and circumference of circle as output.
- Write a program that reads height and base of a triangle and finds its area.

PREPROCESSOR DIRECTIVES

- These are placed in the source program before the main function.
- When our source code is compiled, it is examined by the preprocessor for any preprocessor directives. If there are any, appropriate actions are taken and then the source program is handed over to the compiler.
- They follow special syntax rules: They all begin with the symbol # (hash) and do not require a ; (semicolon) at the end.

PREPROCESSOR DIRECTIVES…

- E.g.
```
#include <stdio.h>
#define PI 3.14
#define TRUE 1
#define FALSE 0
```
- These statements are called preprocessor directives as they are processed before compilation of any source code in the program.
- NOTE: The other codes in the program are compiled sequentially line by line.

SYMBOLIC CONSTANTS

- A symbolic constant is a name that is used in place of a sequence of characters. The character may represent numeric constant, a character constant or a string constant.
- When a program is compiled, each occurrence of a symbolic constant is replaced by its corresponding character sequence.
- The symbolic constants are defined at the beginning of the program.
- SYNTAX: `#define name value`
- E.g. `#define PI 3.1416`

SYMBOLIC CONSTANTS…

RULES

- Symbolic constant names are same as variable names. Convention: Use capital letters while defining symbolic constants.
- No blank space permitted between # and name.
- A blank space is required between #define and symbolic name and between symbolic name and its value (i.e. constant).

COMMENTS

- Used for program documentation.
- Comments are not compiled.
- The C syntax for writing comment is

  /*

  Anything written in between slash and asterisk and asterisk and slash is comment

  */
- Another way to write comment in C

  // Using double slash (This line only)

TYPE CONVERSION

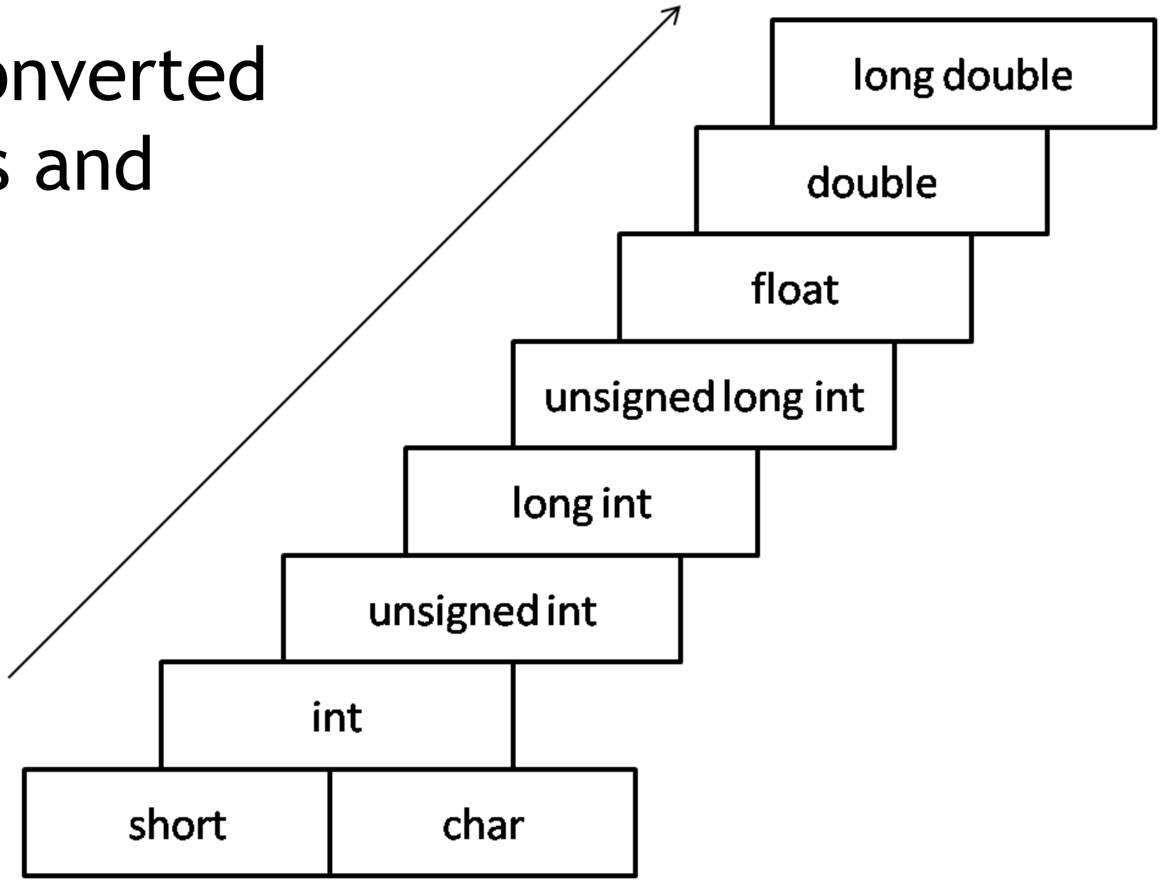
- The process of converting entity of one data type to another data type is called as type casting or type conversion.
- There are two types of type conversion:

1. **Implicit type conversion**: In implicit type conversion compiler automatically converts one data type into another and it i.e. generally obtained through arithmetic expression.
2. **Explicit type conversion**: When a user explicitly convert the one data type into another then it is called as the type casting or *explicit type conversion* in which programmer specifies special programming instruction what data type to convert.

IMPLICIT TYPE CONVERSION IN C

- Lower data types are converted to the higher data types and result is of higher type.

Conversion Hierarchy



// An Example of Implicit Conversion

```
#include<stdio.h>
int main()
{
    int x = 10;    // integer x
    char y = 'a';  // character c

    x = x + y;        // y implicitly converted to int. ASCII  value of 'a' is 97
   float z = x + 1.0; // x is implicitly converted to float

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

EXPLICIT TYPE CASTING

- The general form of a type casting operator is

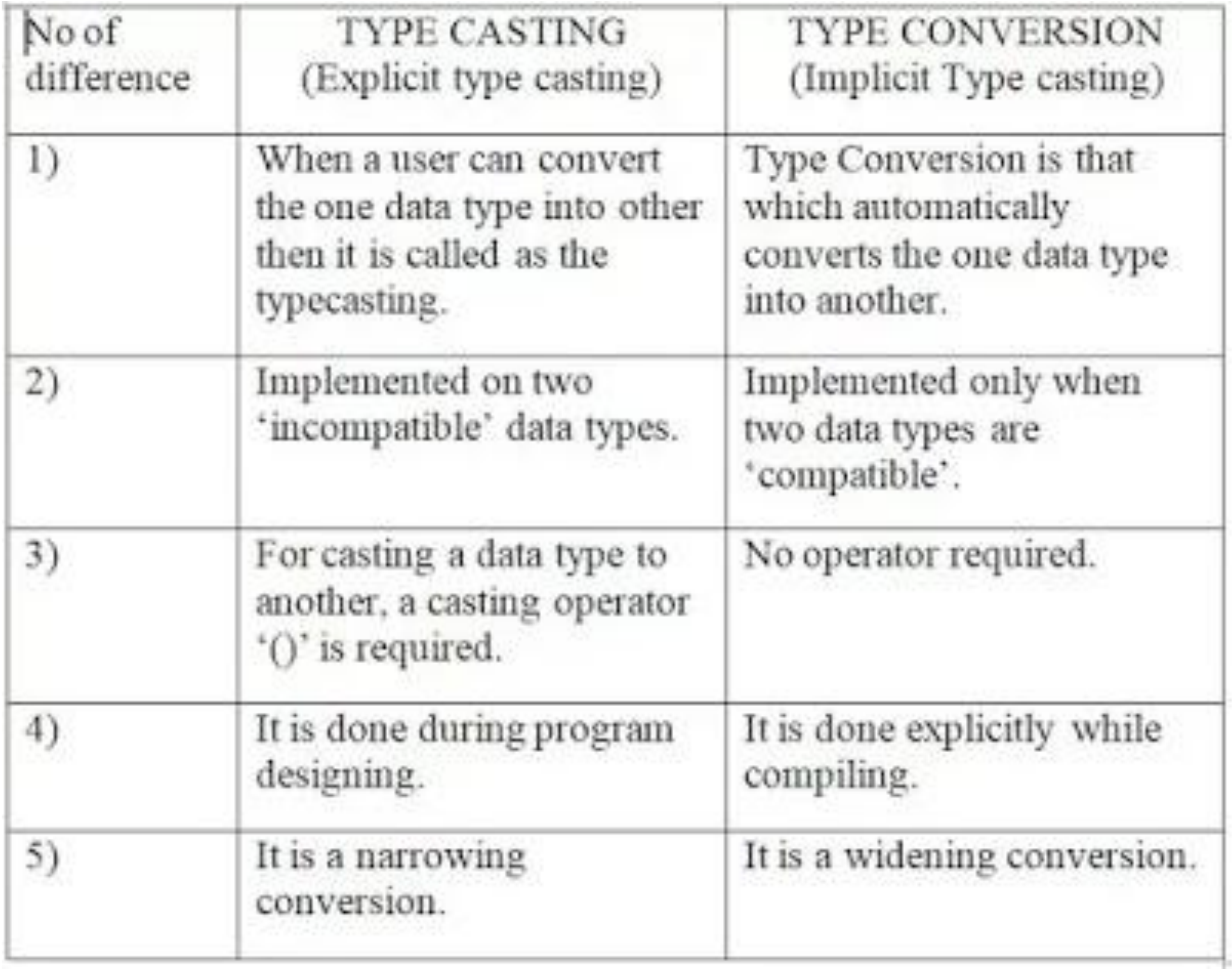
  (type-name) expression
- It is generally a good practice to use explicit casts than to rely on automatic type conversions.
- Example

  `C = (float)9 / 5 * ( f – 32 )`
- `float` to `int` conversion causes truncation of fractional part
- `double` to `float` conversion causes rounding of digits
- `long int` to `int` causes dropping of the higher order bits.

EXPLICIT TYPE CONVERSION

```
#include<stdio.h>
int main()
{
        float a = 1.2;
         int b = (int) a + 1;
        printf("Value of a is %f\n", a);
        printf("Value of b is %d\n",b);
        return 0;
}
```

IMPLICIT VS EXPLICIT TYPE CASTING



No of difference	TYPE CASTING (Explicit type casting)	TYPE CONVERSION (Implicit Type casting)
1)	When a user can convert the one data type into other then it is called as the typecasting.	Type Conversion is that which automatically converts the one data type into another.
2)	Implemented on two 'incompatible' data types.	Implemented only when two data types are 'compatible'.
3)	For casting a data type to another, a casting operator '()' is required.	No operator required.
4)	It is done during program designing.	It is done explicitly while compiling.
5)	It is a narrowing conversion.	It is a widening conversion.

SIZEOF OPERATOR

- The **sizeof** operator is used with an operand to return the number of bytes the operand occupies.
- It is a compile time operator.
- The operand may be a *constant*, *variable* or a *data type qualifier*.

SIZEOF : EXAMPLE

```
#include <stdio.h>
#include <conio.h>
int main()
{
int num;
printf("integer Occupies=> %d bytes\n", sizeof(num));
printf("double Constant Occupies=> %d bytes\n", sizeof(16.18));
printf("long int Data Type Qualifier Occupies=> %d bytes\n", sizeof(15L));
printf("float Data Type Occupies=> %d bytes", sizeof(float));
getch();
Return 0;
}
```

DERIVED DATA TYPES

- Those data types which are derived from the fundamental data types are called **derived data types**.
- Function, arrays, and pointers are derived data types in C programming language.
- For example, an array is derived data type because it contains the similar types of fundamental data types and acts as a new data type for C.
- Eg: int a[10];

USER DEFINDED DATA TYPES

- Those data types which are defined by the user as per his/her will are called **user-defined data types**.
- Examples of such data types are structure, union and enumeration.
- For example, let’s define a structure

```
struct student
 {
  char name[100];
  int roll;
  float marks;
 };
```

ENUMERATION (OR ENUM) IN C

- Enumeration (or enum) is a user defined data type in C.
- It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

ENUM EXAMPLE

```
// An example program to demonstrate working
// of enum in C
#include<stdio.h>

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

int main()
{
    enum week day;
    day = Wed;
    printf("%d",day);
    return 0;
}
```

ENUM EXAMPLE: 2

```
// Another example program to demonstrate working of enum in C
#include<stdio.h>
 enum year  {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};
int main()
{
   int i;
   for (i=Jan; i<=Dec; i++)
      printf("%d ", i);

   return 0;
}
```

INTERESTING FACTS ABOUT INITIALIZATION OF ENUM.

- If we do not explicitly assign values to enum names, the compiler by default assigns values starting from 0. For example, in the following C program, sunday gets value 0, monday gets 1, and so on.

```
#include <stdio.h>
enum day {sunday, monday, tuesday, wednesday,
thursday, friday, saturday};

int main()
{
    enum day d = thursday;
    printf("The day number stored in d is %d", d);
    return 0;
}
```

ENUM EXAMPLE

- We can assign values to some name in any order. All unassigned names get value as value of previous name plus one.

```
#include <stdio.h>
enum day {sunday = 1, monday, tuesday = 5,
          wednesday, thursday = 10, friday, saturday};

int main()
{
    printf("%d %d %d %d %d %d %d", sunday, monday, tuesday,
            wednesday, thursday, friday, saturday);
    return 0;
}
```

Output:

```
1 2 5 6 10 11 12
```

WHAT WILL BE THE OUTPUT?

```
#include <stdio.h>
enum week {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};

int main()
{
    // creating today variable of enum week type
    enum week today;
    today = Wednesday;
    printf("Day %d",today+1);
    return 0;
}
```

TYPE DEFINITIONS

- Type definition allows to create an alias or new name for an existing type or user defined type.
- The syntax of typedef is as follows:

  Syntax: typedef data_type new_name;

  - **typedef**: It is a keyword.
  - **data_type**: It is the name of any existing type or user defined type created using structure/union.
  - **new_name**: alias or new name you want to give to any existing type or user defined type.

TYPEDEF EXAMPLE

```
typedef int myint;
```

- From now on we can declare new int variables using myint instead of int keyword.

```
myint i = 0; // this statement is equivalent to int i = 0;
```

- This statement declares and initializes a variable i of type int.

EXERCISE

1. What are C tokens?
2. Define keyword. Write any four keywords in C.
3. What are rules to define identifiers?
4. What is a variable?
5. What are symbolic constants? Write its advantages.
6. What is data type? Write any four built in data types.
7. What are the different types of integer data types?
8. What are the different types of floating point data types?
9. What is the purpose of typedef?
10. What is the use sizeof operator?
11. Define enumerated data type.

Thank You