

Unit 1

Overview of Procedural Programming

History of C

- C is a general-purpose computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system.
- Although C was designed for implementing system software, it is also widely used for developing portable application software.
- C is a procedural systems implementation language.
- It was designed to be compiled using a relatively straightforward compiler, to
 - ✓ provide low-level access to memory
 - ✓ provide language constructs that map efficiently to machine instructions
 - ✓ require minimal run-time support.

Character Set

- C does not use, nor requires the use of, every character found on a modern computer keyboard.
- The character set in C Language can be grouped into the following categories.
 - ✓ Letters
 - ✓ Digits
 - ✓ Special Characters
 - ✓ White Spaces

Character Set – Letters

- a b c d e f g h i j k l m n o p q r s t u v w x y z
- A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Character Set – Digits

- 0 1 2 3 4 5 6 7 8 9

Special characters

- () { } [] < > = ! \$ % ? , . : ; ' " & | ^ ~ ` # \ blank - _ / * % @

Keywords

- Every word in C language is a keyword or an identifier.
- Keywords in C language cannot be used as a variable name.
- They are specifically used by the compiler for its own purpose and they serve as building blocks of a c program.

- They are also called as **reserved words**.
- C makes use of only 32 keywords or reserved words which combine with the formal syntax to form the C programming 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

Structure of C Program

- Structured language means that the program is compartmentalized.
- The program is divided into a number of modules or parts.
- Each part is used for a specific task.
- The parts are called as **functions**.
- Every C program must begin with the function **main**.
- main()
 - ✓ It is an identifier for the start of the main part of the program.
 - ✓ It does not have any arguments as it has empty parenthesis.
 - ✓ The function will contain instructions.
 - ✓ An opening brace { must appear before the first instruction.
 - ✓ A closing brace } must follow the last instruction.
 - ✓ main() – compiler starts execution from here
 - { – opening brace
 - ; – instructions closed with semicolon
 -; – instructions closed with semicolon
 - } – closing brace

Executing a C Program

- The steps involved are
 - ✓ Creating the program
 - 🔗 It results in a source file
 - ✓ Compiling the program
 - 🔗 Compiler consists of preprocessor and translator
 - ↳ Preprocessor reads the source code and prepares it for the translator.
 - ↳ Translator converts the program into machine language.
 - 🔗 It results in a object file.
 - ✓ Linking the program with functions that are needed from the C library
 - 🔗 The linker assembles all the functions into a final executable program.
 - ✓ Executing the program

- ✚ The OS program known as loader loads the executable program into the memory.

Constants

- A constant value is the one which does not change during the execution of a program.
- C supports several types of constants.
 1. Integer Constants
 2. Real Constants
 3. Single Character Constants
 4. String Constants

Integer constants

- It refers to a sequence of digits.
- Three types of integers
 - ✓ Decimal
 - ✚ Set of digits from 0 to 9 preceded by an optional + or – sign
 - ✚ Example - 10, -90, 98
 - ✓ Octal
 - ✚ Set of digits from 0 to 7 with a leading 0
 - ✚ Example - 037, 07765, 0
 - ✓ Hexadecimal
 - ✚ Sequence of digits preceded by 0x or 0X.
 - ✚ It may also include alphabets from A to F representing numbers from 10 thru 15.
 - ✚ Example - 0X2, 0Xcda

Real constants

- Numbers that contain fractional parts.
- They can be written in
 - ✓ Decimal notation
 - ✚ A whole number followed by a decimal point and the fractional part.
 - ✚ Examples 0.0087, 90.98987, -90.78786, +456.90
 - Exponential notation or scientific notation
 - ✓ The general form is
 - ✚ ***mantissa e exponent***
 - ↳ Mantissa
 - † A real number expressed in decimal notation or an integer
 - ↳ Exponent
 - † An integer number with an optional plus or minus sign.

↳ The letter e separating the mantissa and the exponent can be written in small case or upper case.

✓ Examples

✚ 0.65e4, 18e-2, 8.5e+5

Single character constants

- It contains a single character enclosed within a pair of single quotation marks.
- Examples

✚ 'A', '10', '.', ''

✚ Here character 10 is not same as number 10.

String constants

- It is a sequence of characters enclosed in double quotes.
- Examples

✚ "Hello!!!", "1998", "8*6"

Variables

- A variable is a value that can change any time.
- It is a memory location used to store a data value.
- A variable name should be carefully chosen by the programmer so that its use is reflected in a useful way in the entire program.
- Variable names are case sensitive.
- Example num, salary, Emp_name

Data types

- C language data types can be broadly classified as

✚ Primary data type

✚ Derived data type

✚ User-defined data type

Primary Data type

- All C Compilers accept the following fundamental data types

✚ Integer - int

✚ Character - char

✚ Floating Point - float

✚ Double precision floating point – double

✚ void - void

- The size and range of each data type is given in the table below

DATA TYPE	RANGE OF VALUES
char	-128 to 127
int	-32768 to +32767
float	3.4 e-38 to 3.4 e+38
double	1.7 e-308 to 1.7 e+308

Integer type

- Integers are whole numbers with a machine dependent range of values.
- C has 3 classes of integer storage
 - short int
 - int
 - long int
- All of these data types have signed and unsigned forms.
- A short int requires half the space than normal integer values.
- Unsigned numbers are always positive and consume all the bits for the magnitude of the number.
- The long and unsigned integers are used to declare a longer range of values.
- Examples
 - 12, -90, 8987656, -898989

Floating point types

- Floating point number represents a real number with 6 digits precision.
- C allows us
 - float
 - double
 - long double
- When the accuracy of the floating-point number is insufficient, double can be used to define the number.
- The double is same as float but with longer precision.
- To extend the precision further we can use long double which consumes 80 bits of memory space.

Void type

- It has no values and only one operation – assignment.
- Using void data type, we can specify the type of a function.
- It is a type that can represent any other standard type.

- It is a good practice to avoid functions that does not return any values to the calling function.

Character type

- A single character can be defined as a character type of data.
- Characters are usually stored in 8 bits of internal storage.
- The qualifier signed or unsigned can be explicitly applied to char.
- While unsigned characters have values between 0 and 255, signed characters have values from -128 to 127.

Size and Range of Data Types on 32 bit machine

Type	SIZE (Bytes)	Range
Char / Signed char	1	-128 to 127
Unsigned char	1	0 to 255
short int / signed short int	2	-32768 to 32767
unsigned short int	2	0 to 65535
int / signed int	4	-2147483648 to 2147483647
unsigned int	4	0 to 4294967295
long int / signed long int	4	-2147483648 to 2147483647
Unsigned long int	4	0 to 4294967295
Float	4	$3.4e^{-38}$ to $3.4e^{+38}$
Double	8	$1.7e^{-308}$ to $1.7e^{+308}$
Long double	16	

Declaration of variables

- Every variable used in the program should be declared to the compiler.
- The declaration does two things.
 1. Tells the compiler the variables name.
 2. Specifies what type of data the variable will hold.
- The general format of any declaration


```
datatype v1, v2, v3, ..... Vn;
```

Where v_1, v_2, v_3 are variable names.
- Variables are separated by commas.

- A declaration statement must end with a semicolon.
- Examples

```
int sum;
int number, salary;
double average, mean;
```

Datatype	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

User defined type declaration

- In C language a user can define an identifier that represents an existing data type.
- The user defined datatype identifier can later be used to declare variables.
- The general syntax is

```
typedef type identifier;
```

```
here
```

↳ 'type' represents existing data type

↳ 'identifier' refers to the 'row' name given to the data type.

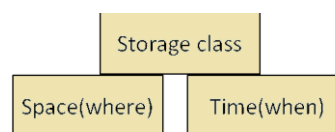
- Example

```
typedef int salary;
typedef float average;
```

- ↳ Here salary symbolizes int and average symbolizes float.
- ↳ They can be later used to declare variables as follows:
 - † Units dept1, dept2;
 - † Average section1, section2;
 - † Therefore dept1 and dept2 are indirectly declared as integer datatype and section1 and section2 are indirectly float data type.
- The second type of user defined datatype is enumerated data type.
- An enumeration consists of a set of named integer constants.
- An enumeration type declaration gives the name of the (optional) enumeration tag and defines the set of named integer identifiers (called the "enumeration set," "enumerator constants," "enumerators," or "members")
- It is defined as follows.
 - ✚ enum identifier {value₁, value₂ Value_n};
 - ↳ The identifier is a user defined enumerated datatype which can be used to declare variables that have one of the values enclosed within the braces.
- After the definition we can declare variables to be of this 'new' type as below.
 - ✚ enum identifier V₁, V₂, V₃, V_n
- The enumerated variables V₁, V₂, V_n can have only one of the values value₁, value₂ value_n.
- Enumerated data type variables can only assume values which have been previously declared.
- Example
 - ✚ enum day {Monday, Tuesday, Sunday};
 - ✚ enum day week_st, week_end;
 - ✚ week_st = Monday;
 - ✚ week_end = Friday;
 - ✚ if(week_st == Tuesday)
 - ✚ week_en = Saturday;

Declaration of storage class

- Variables in C have not only the data type but also storage class that provides information about their location and visibility.
- The storage class divides the portion of the program within which the variables are recognized.










- auto

- ✚ It is a local variable known only to the function in which it is declared.
 - ✚ Auto is the default storage class.
 - ✚ They contain undefined values known as garbage unless they are initialized explicitly.
- static
 - ✚ Local variable which exists and retains its value even after the control is transferred to the calling function.
 - ✚ They are automatically initialized to 0.
- extern
 - ✚ Global variable known to all functions in the file.
 - ✚ They are automatically initialized to 0.
- register
 - ✚ Local variables which are stored in the register.




Scopes of variables

- Scopes determine the visibility of an identifier.
- File scope
 - ✚ Starts at the beginning of the file. (translation unit)
 - ✚ Ends with the end of the file.
 - ✚ Refers to identifiers declared outside all the functions.
 - ✚ Visible throughout the entire file.
 - ✚ They are global variables.
- Block scope
 - ✚ Starts at the beginning of the { of a block.
 - ✚ Ends with the end of the block }.
 - ✚ Variables are local to their block.
 - ✚ Extends to function parameters in a function definition.
- Function Prototype scope
 - ✚ Identifiers declared in a function prototype.
 - ✚ Visible within the prototype.
- Function scope
 - ✚ Starts at the opening of a '{' of a function.
 - ✚ Ends with the ending of a function'}'.
 - ✚ Applies only to labels.
 - ↳ Used as a target of goto statement.
 - ↳ Must be within the same function as the goto.


Assigning values to variables

- Values can be assigned to variables using the assignment operator.
 `variable_name = constant;`
- C permits multiple assignments in one line.
 `int_val = 0; fin_val = 1000;`
- The assignment statement implies that the value of the variable on the left side of the equal sign is equal to the value of the quantity or expression on the right side.
- Example
 `y = y + 4;`
- The value to a variable can be assigned at the time it is being declared.
 `data type variable = constant;`
- This is known as initialization.
- Example
 `int i = 10;`
 `char c = 'y';`
- C allows initialization of more than one variable in one statement using multiple assignment operators.
- Example
 `int i=0,j=1,h = 10;`

Type Qualifiers

- Control how variables may be accessed or modified.
 `const`
 `volatile` – both C89 standards
 `restrict` – C99 standards
- The qualifiers precede the type name.

Type Qualifiers – const

- The values of some variable may be required to remain constant through-out the program.
- This can be done by using the qualifier `const` at the time of initialization.
- Such variables are normally kept in ROM by the compiler.
- Example
 `const int class_size = 40;`
- The `const` data type qualifier tells the compiler that the value of the `int` variable `class_size` may not be modified in the program.
- They have to be initialized.
- They cannot be assigned.

Type Qualifiers – volatile

- A volatile variable is the one whose values may be changed at any time by some external sources.
- The syntax is
 - ✚ ***volatile data type variable_name;***
- Example
 - ✚ `volatile int num;`
- The value of data may be altered by some external factor, even if it does not appear on the left hand side of the assignment statement.
- When a variable is declared as volatile the compiler will examine the value of the variable each time it is encountered to see if an external factor has changed the value.
- Example
 - ✚ A pointer referencing a device's hardware register.
 - ✚ Threaded code
- The reason to use volatile is to ensure that the compiler generates code to reload the data each time it is referenced in the program.

Type Qualifiers – restrict

- Applies only to pointers
- A pointer qualified by restrict is initially only the means to access the object that it points to.
- Access to the object by another pointer can occur only if second pointer is based on first.

Constants

- Refer to fixed values that the program cannot alter.
- Can be of any of the basic data types.
- Also called literals.
- Character constants
 - ✚ Enclosed within single quotes.
 - ✚ Can be
 - ↳ multibyte characters
 - † Enclose within single quotes
 - ↳ wide characters
 - † Enclose within single quotes and precede with an L.
 - ✚ Example
 - † `X='a'; lx=L'x';`
- Numeric constants
 - ✚ Fits it into the smallest compatible data type that will hold it.

- ✚ Can be of
 - ↳ Integer constants
 - † Specified as numbers without fraction components.
 - † Suffix with U for unsigned.
 - † Suffix with L for long
 - ↳ Float constants
 - † Specified as numbers with fraction components
 - † Can use scientific notation.
 - † Suffix the number with F.
 - † Suffix the number with L for long double.
- Hexadecimal Constants
 - ✚ Must consist of 0x followed by the constant in hexadecimal format
 - ↳ 0-9, a-f
- Octal Constants
 - ✚ Must consist of 0 followed by octal format
 - ↳ 0-7
- String constants
 - ✚ Enclosed within " ".
- Backslash character constants
 - ✚ Also called as escape sequences.
 - † \b – backspace
 - † \t – horizontal tab
 - † \f – form feed
 - † \" – double quotes
 - † \n – new line
 - † \' – single quote
 - † \r – carriage return
 - † \\ – back slash
 - † \v – vertical tab
 - † \a – alert
 - † \? – question mark
 - † \N – octal constant (N is an octal constant)
 - † \xN – hexadecimal constant (N is a hexadecimal constant)

Defining symbolic constants

- A symbolic constant value can be defined as a preprocessor statement and used in the program as any other constant value.

- The general form of a symbolic constant is
 - ✚ # define symbolic_name value of constant
- Valid examples of constant definitions are
 - ✚ # define marks 100
 - ✚ # define total 50
 - ✚ # define pi 3.14159
- These values may appear anywhere in the program, but must come before it is referenced in the program.
- It is a standard practice to place them at the beginning of the program.

Overflow and underflow of data

- Overflow of data
 - ✚ It occurs when the value of the data is too big for the data type to hold.
- Underflow of data
 - ✚ It occurs when the value of the data is too small for the data type to hold.
- Floating type
 - ✚ Overflow results in largest possible real value.
 - ✚ Underflow results in zero.
- Integers
 - ✚ They are always exact within the limits of the range of the integral data type used.
 - ✚ Overflow is a serious problem in integers since it occurs if the data type does not match the value of the constant.
 - ✚ C does not give any warning or error of integer overflow.
 - ✚ C gives incorrect result.
 - ✚ Overflow usually produces a negative number.

Compilers vs. Interpreters

- Two general methods in which a program is executed.
 - ✚ Compiled
 - ✚ Interpreted
- Interpreter
 - ✚ Reads the source code of the program one line at a time performing the specific instructions contained in that line.
 - ✚ Run-time interpreter is required to be present to execute the program.
- Compiler
 - ✚ Reads the entire source code of the program and converts into object code.

- ✚ It is a translation of the program's code into a form that the computer can execute directly.
- ✚ Object code is referred as binary code or machine code.
- ✚ Compilation is one time cost.

Questions

- Write a typical structure of a C program.
- What are variables? How are they different from key words? Give examples in each case.
- Explain basic data types in C? Give their memory requirement.
- What is a variable? List the rules for naming variables. Give atleast two examples for each valid and invalid variables names.
- With reference to C language, explain
 - ✚ Identifiers, constants, keywords
- What are the simple data types supported by C programming language? Give and explain the syntax of declaring variables of these types with examples.