Introduction to structured programming using c language

5.1 C concepts

C is a general purpose programming language, unlike other languages such as C and FORTRAN developed for some specific uses. C is designed to work with both software and hardware. C has in fact been used to develop a variety of software such as:

- ✓ Operating systems: Unix and Windows.
- ✓ Application packages: WordPerfect and Dbase.

It was originally developed by Dennis M. Ritchie to develop the UNIX operating system at Bell Labs. C was originally first implemented on the DEC PDP-11 computer in 1972. In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard. The UNIX operating system, the C compiler, and essentially all UNIX applications programs have been written in C. The C has now become a widely used professional language for various reasons.

- Easy to learn
- Structured language
- It produces efficient programs.
- It can handle low-level activities.
- It can be compiled on a variety of computer platforms.

Facts about C

- C was invented to write an operating system called UNIX.
- C is a successor of B language, which was introduced around 1970.
- The language was formalized in 1988 by the American National Standard Institute. (ANSI).
- The UNIX OS was totally written in C by 1973.
- Today, C is the most widely used and popular System Programming Language.
- Most of the state-of-the-art softwares have been implemented using C.
- Today's most popular Linux OS and RBDMS MySQL have been written in C.

Why to use C?

C was initially used for system development work, in particular the programs that make up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:

- Operating Systems
- Databases
- Language Interpreters
- Utilities
- Modern Programs
- Text Editors
- Language Compilers

MERITS OF C LANGUAGE

- **C Supports structured programming design features.**
  It allows programmers to break down their programs into functions. Further it supports the use of comments, making programs readable and easily maintainable.
- **Efficiency**
  - ✓ C is a concise language that allows you to say what you mean in a few words.
  - ✓ The final code tends to be more compact and runs quickly.
- **Portability**
  C programs written for one system can be run with little or no modification on other systems.
- **Power and flexibility**
  - ✓ C has been used to write operating systems such as Unix, Windows.
  - ✓ It has (and still is) been used to solve problems in areas such as physics and engineering.
- **Programmer orientation**
  - ✓ C is oriented towards the programmer's needs.
  - ✓ It gives access to the hardware. It lets you manipulate individual bits of memory.
  - ✓ It also has a rich selection of operators that allow you to expand programming capability.

5.2 C programming environment

Local Environment Setup

If you want to set up your environment for C programming language, you need the following two software tools available on your computer, (a) Text Editor and (b) The C Compiler.

Text Editor

This will be used to type your program. Examples of few a editors include Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, and vim or vi.

The name and version of text editors can vary on different operating systems. For example, Notepad will be used on Windows, and vim or vi can be used on windows as well as on Linux or UNIX.

The files you create with your editor are called the source files and they contain the program source codes. The source files for C programs are typically named with the extension ".c".

Before starting your programming, make sure you have one text editor in place and you have enough experience to write a computer program, save it in a file, compile it and finally execute it.

The C Compiler

The source code written in source file is the human readable source for your program. It needs to be "compiled", into machine language so that your CPU can actually execute the program as per the instructions given.

The compiler compiles the source codes into final executable programs. The most frequently used and free available compiler is the GNU C/C++ compiler; otherwise you can have compilers either from HP or Solaris if you have the respective operating systems.

Preprocessor directives

Preprocessor directives are lines included in a program that begin with the character #, which make them different from a typical source code text. They are invoked by the compiler to process some programs before compilation.

- #define- the #**define** directive allows the **definition** of macros within your source code.
- #include- Inserts a particular header from another file.
- #undef- Undefines a preprocessor macro.
- #ifdef- Returns true if this macro is defined.
- #ifndef- Returns true if this macro is not defined.
- #if- Tests if a compile time condition is true.
- #else- The alternative for #if.
- #elif- #else and #if in one statement.
- #endif- Ends preprocessor conditional.
- #error-Prints error message on stderr.
- #pragma- Issues special commands to the compiler, using a standardized method.

5.3 C program format

A C program basically consists of the following parts −

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

— **Preprocessor Commands:** These commands tells the compiler to do preprocessing before doing actual compilation. Like *#include <stdio.h>* is a preprocessor command which tells a C compiler to include stdio.h file before going to actual compilation.

— **Functions:** are main building blocks of any C Program. Every C Program will have one or more functions and there is one mandatory function which is called *main()* function. This function is prefixed with keyword *int* which means this function returns an integer value when it exits. This integer value is retured using *return* statement.

— The C Programming language provides a set of built-in functions. In the above example *printf()* is a C built-in function which is used to print anything on the screen. Check Builtin functionsection for more detail.

— You will learn how to write your own functions and use them in Using Function session.

— **Variables:** are used to hold numbers, strings and complex data for manipulation. You will learn in detail about variables in C Variable Types.

— **Statements & Expressions :** Expressions combine variables and constants to create new values. Statements are expressions, assignments, function calls, or control flow statements which make up C programs.

— **Comments:** are used to give additional useful information inside a C Program. All the comments will be put inside /*...*/ as given in the example above. A comment can span through multiple lines.

Note the followings

- C is a case sensitive programming language. It means in C *printf* and *Printf* will have different meanings.
- C has a free-form line structure. End of each C statement must be marked with a semicolon.
- Multiple statements can be one the same line.
- White Spaces (ie tab space and space bar) are ignored.
- Statements can continue over multiple lines.

```
#include <stdio.h>
int main() {
   /* my first program in C */
   printf("Hello, World! \n");
   return 0;
}
```

Explanation:

— The first line of the program #include <stdio.h> is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.
— The next line int main() is the main function where the program execution begins.
— The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
— The next line printf(...) is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
— The next line **return 0;** terminates the main() function and returns the value 0.

6.0 Fundamentals of c programming

6.1 C fundamentals

Operators and Operands

An **operator** is a component of any expression that joins individual constants, variables, array elements and function references.

An **operand** is a data item that is acted upon by an operator. Some operators act upon two operands (binary operators) while others act upon only one operand (unary operators).

An operand can be a constant value, a variable name or a symbolic constant.

Note: An expression is a combination of operators and operands.

Examples
x + y ; x, y are operands, + is an addition operator.
3 * 5; 3, 5 are constant operands, * is a multiplication operator.
x % 2.5; x, 5 are operands, % is a modulus (remainder) operator.
sizeof (int); sizeof is an operator (unary), int is an operand.

Arithmetic Operators

There are five arithmetic operators in C.

Operator	**Purpose**
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder after integer division

Note:
There exists no exponential operators in C.
The operands acted upon by arithmetic operators must represent numeric values, that is operands may be integers, floating point quantities or characters (since character constants represent integer values).
The % (remainder operator) requires that both operands be integers.
Thus;
5 % 3
int x = 8;
int y = 6 ; x % y are valid while;
8.5 % 2.0 and
float p = 6.3, int w = 7 ; 5 %p , p % w are invalid.
Division of one integer quantity by another is known as an integer division. If the quotient (result of division) has a decimal part, it is truncated.
Dividing a floating point number with another floating point number, or a floating point number with an integer results to a floating point quotient .

Exercise
Suppose a = 10, b = 3, v1 = 12.5, v2 = 2.0, c1 =’P’, c2 = ‘T’. Compute the result of the following expressions.

a + b	v1 * v2
a - b	v1 / v2
a * b	c1
a / b	c1 + c2 +5
a % b	c1 + c2 +’9’

Note:
c1 and c2 are character constants

ASCII codes for 9 is 57, P = 80,T = 84.

If one or both operands represent negative values, then the addition, subtraction, multiplication, and division operators will result in values whose signs are determined by their usual rules of algebra. Thus if a b, and c are 11, -3 and –11 respectively, then

a + b = 8
a – b = 14
a * b = -33
a / b = -3
a % b = -2
c % b = -2
c / b = 3

Examples of floating point arithmetic operators
r1 = -0.66, r2 = 4.50 (operands with different signs)
r1 + r2 = 3.84
r1 - r2 = -5.16
r1 * r2 = -2.97
r1 / r2 = -0.1466667

Note:
If both operands are floating point types whose precision differ (e.g. a float and a double) the lower precision operand will be converted to the precision of the other operand, and the result will be expressed in this higher precision. (Thus if an expression has a float and a double operand, the result will be a double).
If one operand is a floating-point type (e.g. float, double or long double) and the other is a character or integer (including short or long integer), the character or integer will be converted to the floating point type and the result will be expressed as such.
If neither operand is a floating-point type but one is long integer, the other will be converted to long integer and the result is expressed as such. (Thus between an int and a long int, the long int will be taken).
If neither operand is a floating type or long int, then both operands will be converted to int (if necessary) and the result will be int (compare short int and long int)

From the above, evaluate the following expressions given:
i =7, f = 5.5, c = 'w'. State the type of the result.

i + f
i + c
i + c-'w'
(i + c) - (2 * f / 5)

('w" has ASCII decimal value of 119)

Note: Whichever type of result an expression evaluates to, a programmer can convert the result to a different data type if desired. The general syntax of doing this is:

(data type) expression.
The data type must be enclosed in parenthesis (). For example the expression (i + f) above evaluates to 12.5. To convert this to an integer, it will be necessary to write
(int) (i + f).

Operator Precedence

The order of executing the various operations makes a significant difference in the result. C assigns each operator a precedence level. The rules are;

Multiplication and division have a higher precedence than addition and subtraction, so they are performed first.

If operators of equal precedence; (*, /), (+, -) share an operand, they are executed in the order in which they occur in the statement. For most operators, the order (associativity) is from left to right with the exception of the assignment (=) operator.

Consider the statement;

butter = 25.0 + 60.0 * n / SCALE;
Where n = 6.0 and SCALE = 2.0.

The order of operations is as follows;

First: 60.0 * n = 360.0
(Since * and / are first before + but * and / share the operand n with * first)

Second: 360.0 / SCALE = 180
(Division follows)

Third: 25.0 + 180 = 205.0 (Result)
(+ comes last)

Note that it is possible for the programmer to set his or her own order of evaluation by putting, say, parenthesis. Whatever is enclosed in parenthesis is evaluated first.

What is the result of the above expression written as:
+ 60.0 * n) / SCALE.

Example: Use of operators and their precedence

```
/* Program to demonstrate use of operators and their precedence */
include<stdio.h >
main()
{
        int score,top;
```

```
        score = 30;
        top = score - (2*5) + 6 * (4+3) + (2+3);
        printf ("top = %d \ n" , top);
        system("pause");
        return 0;
}
```

Try changing the order of evaluation by shifting the parenthesis and note the change in the top score.

Example: Converting seconds to minutes and seconds using the % operator

```
#include<stdio.h >
#define SEC_PER_MIN 60
main()
{
  int sec, min, sec_left;
  printf(" Converting  seconds to minute and seconds \n ") ;
  printf( "Enter number of seconds you wish to convert \n ") ;
  scanf("% d" , &sec ) ;        /* Read in number of seconds */
  min = sec / SEC_PER_MIN ;        / * Truncate number of seconds */
  sec_left = sec % SEC_PER_MIN ;
  printf("% d seconds is % d minutes,% seconds\n " ,sec,min,sec_left);
  system("pause");
  return 0;
}
```

The sizeof operator

sizeof returns the size in bytes, of its operand. The operand can be a data type e.g. sizeof (int), or a specific data object e.g. sizeof n.

If it is a name type such as int, float etc. The operand should be enclosed in parenthesis.

Example : Demonstrating 'sizeof' operator

```
#include <stdio.h>
main()
{
        int n;
        printf("n has % d bytes; all ints have % d bytes \n", sizeof n, sizeof(int)) ;
       system("pause");
              return 0;
}
```

The Assignment Operator

The Assignment operator (=) is a value assigning operator. There are several other assignment operators in C. All of them assign the value of an expression to an identifier.

Assignment expressions that make use of the assignment operator (=) take the form;

identifier = expression;

where identifier generally represents a variable, constant or a larger expression.

Examples of assignment;

```
a = 3 ;
x = y ;
pi = 3.14;
sum = a + b ;
area_circle = pi * radius * radius;
```

Note

You cannot assign a variable to a constant such as 3 = a ;

The assignment operator = and equality operator (= =) are distinctively different. The = operator assigns a value to an identifier. The equality operator (= =) tests whether two expressions have the same value.

Multiple assignments are possible e.g. a =b = 5 ; assigns the integer value 5 to both a and b.

Assignment can be combined with +, -, /, *, and %

The Conditional Operator

Conditional tests can be carried out with the conditional operator (**?**). A conditional expression takes the form:

expression1 ? expression2 : expression3 and implies;

evaluate **expression1**. If **expression1** evaluates to **true** (value is 1 or non zero) then evaluate **expression 2**, otherwise (i.e. if expression 1 is false or zero) , evaluate **expression3**.

Consider the statement **(i < 0) ? 0 :100**

Assuming **i** is an integer, the expression (i < 0) is evaluated and if it is true, then the result of the entire conditional expression is zero (0), otherwise, the result will be 100.

Unary Operators

These are operators that act on a single operand to produce a value. The operators may precede the operand or are after an operand.

Examples

Unary minus e.g. - 700 or –x

Incrementation operator e.g. c++

Decrementation operator e.g. f - -

sizeof operator e.g. sizeof(float)

- Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.

- Syntax:
  Increment operator: ++var_name; (or) var_name++;
  Decrement operator: – -var_name; (or) var_name – -;
- Example:
  Increment operator : ++ i ; i ++ ;
  Decrement operator : – – i ; i – – ;

Relational Operators

There are four relational operators in C.

<	Less than
<=	Less than or equal to
>	Greater than
> =	Greater than or equal to

Closely associated with the above are two equality operators;

= =	Equal to
! =	Not equal to

The above six operators form **logical expressions.**

A logical expression represents conditions that are either true (represented by integer 1) or false (represented by 0).

Example
Consider a, b, c to be integers with values 1, 2,3 respectively. Note their results with relational operators below.

Expression	**Result**
a < b	1 (true)
(a+ b) > = c	1 (true)
(b + c) > (a+5)	0 (false)
C != 3	0(false)
b = = 2	1 (true)

Logical operators

&&	Logical AND
||	Logical OR
!	NOT

The two operators act upon operands that are themselves logical expressions to produce more complex conditions that are either true or false.

Example
Suppose i is an integer whose value is 7, f is a floating point variable whose value is 5.5 and C is a character that represents the character 'w', then;

(i > = = 6) && (C = = 'w') is 1 (true)
(C' > = 6) || (C = 119) is 1 (true)
(f < 11) && (i > 100) is 0 (false)
(C! = ' p') || ((i + f) < = 10) is 1 (true)

Revision Exercises

Describe with examples, four relational operators.
What is 'operator precedence'? Give the relative precedence of arithmetic operators.
Suppose a, b, c are integer variables that have been assigned the values a =8, b = 3 and c = - 5, x, y, z are floating point variables with values x =8.8, y = 3.5, z = -5.2.

Further suppose that c1, c2, c3 are character-type variables assigned the values E, 5 and ? respectively.

Determine the value of each of the following expressions:

a / b	(v) x % y
2 * b + 3 * (a – c)	(vi) 2 * x / (3 * y)
(a * c) % b	(vii) c1 / c3
(x / y) + z	(viii) (c1 / c2) * c3

Data Types
In computer programming, a data type or simply type is a classification of data which tells the compiler or interpreter how the programmer intends to use the data. Most programming languages support various types of data, for example: real, integer or Boolean. A Data type provides a set of values from which an expression (i.e. variable, function ...) may take its values. The type defines the operations that can be done on the data, the meaning of the data, and the way values of that type can be stored.

Simple Data types - primitive data type is either of the following:
a basic type is a data type provided by a programming language as a basic building block. Most languages allow more complicated composite types to be recursively constructed starting from basic types.
a built-in type is a data type for which the programming language provides built-in support. They include

Type	Meaning	**Keyword**
Character	Character data	Char
Integer	Signed whole number	Int
Float	floating-point numbers	Float
Double	double precision floating-point numbers	Double

Void	Valueless	Void

The ‘int’ specifier
It is a type specifier used to declare integer variables. For example, to declare count as an integer you would write:
int count;

Integer variables may hold signed whole numbers (numbers with no fractional part). Typically, an integer variable may hold values in the range –32,768 to 32,767 and are 2 bytes long.

The ‘char’ specifier
A variable of type char is 1 byte long and is mostly used to hold a single character. For example to declare **ch** to be a character type, you would write:

```
char ch;
```

The ‘float’ specifier
It is a type specifier used to declare floating-point variables. These are numbers that have a whole number part and a fractional or decimal part for example 234.936. To declare **f to** be of type float, you would write:

```
float f;
```

Floating point variables typically occupy 4 bytes.
The ‘double’ specifier
It is a type specifier used to declare double-precision floating point variables. These are variables that store float point numbers with a precision twice the size of a normal float value. To declare **d** to be of type double you would write:

```
double d;
```

Double-type variables typically occupy 8 bytes.

An identifier- is a string of alphanumeric characters that begins with an alphabetic character or an underscore character that are used to represent various programming elements such as variables, functions, arrays, structures, unions and so on. Actually, an identifier is a user-defined word.

An expression - in a programming language is a combination of one or more explicit values, constants, variables, operators, and functions that the programming language interprets (according to its particular rules of precedence and of association) and computes to produce ("to return", in a stateful environment) another value. This process, as for mathematical expressions, is called evaluation.

I/O Instructions –

C programming has several in-built library functions to perform input and output tasks.

Two commonly used functions for I/O (Input/Output) are `printf()` and `scanf()`.

The `scanf()` function reads formatted input from standard input (keyboard) whereas the `printf()` function sends formatted output to the standard output (screen).

Example 1: C Output

```
#include <stdio.h>      // Including header file to run printf() function.
int main()
{
    printf("C Programming");  // Displays the content inside quotation
    return 0;
}
```

Output

```
C Programming
```

- All valid C program must contain the `main()` function. The code execution begins from the start of `main()` function.
- The `printf()` is a library function to send formatted output to the screen. The `printf()`function is declared in `"stdio.h"` header file.
- Here, `stdio.h` is a header file (standard input output header file) and `#include` is a preprocessor directive to paste the code from the header file when necessary. When the compiler encounters `printf()` function and doesn't find `stdio.h` header file, compiler shows error.
- The `return 0;` statement is the "Exit status" of the program.

Example 2: C Integer Output

```
#include <stdio.h>
int main()
{
```

```
    int testInteger = 5;
    printf("Number = %d", testInteger);
    return 0;
}
```

Output

```
Number = 5
```

Inside the quotation of `printf()` function, there is a format string `"%d"` (for integer). If the format string matches the argument (`testInteger` in this case), it is displayed on the screen.

Example 3: C Integer Input/Output

```
#include <stdio.h>
int main()
{
    int testInteger;
    printf("Enter an integer: ");
    scanf("%d",&testInteger);
    printf("Number = %d",testInteger);
    return 0;
}
```

Output

```
Enter an integer: 4

Number = 4
```

Keywords - These are reserved words that have special meaning in a language. The compiler recognizes a keyword as part of the language's built – in syntax and therefore it cannot be used for any other purpose such as a variable or a function name. C keywords must be used in lowercase otherwise they will not be recognized.

Examples of keywords

auto	break	case	else	int	void
default	do	double	if	sizeof	long
float	for	goto	signed	unsigned	**Error! Bookmark not defined.**
register	return	short	union	continue	
struct	switch	typedef	const	extern	
volatile	while	char	enum	static	

Handling Errors

Logic Errors

These occur from the incorrect use of control structures, incorrect calculation, or omission of a procedure. Examples include: An indefinite loop in a program, generation of negative values instead of positive values. The compiler will not detect such errors since it has no way of knowing your intentions. The programmer must try to run the program so that he/she can compare the program's results with already known results.

Run Time Error

This is an error that occurs during the execution of a program. In contrast, compile-time errors occur while a program is being compiled. Runtime errors indicate bugs in the program or problems that the designers had anticipated but could do nothing about. For example, running out of memory will often cause a runtime error.

Semantic Errors

They are caused by illegal expressions that the computer cannot make meaning of. Usually no results will come out of them and the programmer will find it difficult to debug such errors. Examples include a data overflow caused by an attempt to assign a value to a field or memory space smaller than the value requires division by zero.

Syntax errors

These are errors in the code you write so that the compiler cannot understand your code and the code will not compile. They include missing colon, semicolon and parenthesis.

Syntaxerrors occur when a program does not conform to the grammar of a programming language, and the compiler cannot compile the source file. **Logic** errors occur when a program does not do what the programmer expects it to do.

Coding - Computer programming (often shortened to programming) is a process that leads from an original formulation of a computing problem to executable computer programs. Programming involves activities such as analysis, developing understanding, generating algorithms, verification of requirements of algorithms including their correctness and resources consumption, and implementation (commonly referred to as coding) of algorithms in a target programming language. Source code is written in one or more programming languages. The purpose of programming is to find a sequence of instructions that will automate performing a specific task or solving a given problem. The process of programming thus often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms, and formal logic.

Compiling - A compiler is a computer program (or a set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language), with the latter often having a binary form known as object code. The most common reason for converting source code is to create an executable program. The name "compiler" is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language or machine code). If the compiled program can run on a computer whose CPU or operating system is different from the one on which the compiler runs, the compiler is known as a cross-compiler. More generally, compilers are a specific type of translator.

Debugging - Debugging is the routine process of locating and removing computer program bugs, errors or abnormalities, which is methodically handled by software programmers via debugging tools. Debugging checks, detects and corrects errors or bugs to allow proper program operation according to set specifications. Developing software programs undergo heavy testing, updating, troubleshooting and maintenance. Normally, software contains errors and bugs, which are routinely removed. In the debugging process, complete software programs are regularly compiled and executed to identify and rectify issues. Large software programs, which contain millions of source code lines, are divided into small components.

Testing - testing is finding out how well something works. In terms of human beings, testing tells what level of knowledge or skill has been acquired. In computer hardware and software development, testing is used at key checkpoints in the overall process to determine whether objectives are being met. For example, in software development, product objectives are sometimes tested by product user representatives. When the design is complete, coding follows and the finished code is then tested at the unit or module level by each programmer; at the component level by the group of programmers involved; and at the system level when all components are combined together. At early or late stages, a product or service may also be tested for usability.

Language Translators- These are programs that translate programs written in a language other than machine language into machine language programs for execution. Programs written in assembly or high level languages are called source programs or source code before they undergo translation. After the translation the machine language version of the same program is called object program or object code. Language translators can be classified into

- Assemblers

- Compilers
- Interpreters

Assembler- This is a program that translates a source program written in assembly language into its equivalent machine code (object code).

Compiler- During compilation both the high level source program and the compiler are loaded into the RAM. The compiler reads through the source program statement by statement, converting each correct statement into multiple machine code instructions. The process continues until the compiler has read through the entire source program or it encounters an error. An erroneous statement is not translated but is placed on the source program error listing, which will be displayed to the programmer at the end of the translation process. Where there are no errors the compiler will generate a machine code object program which is stored for subsequent execution.

Interpreter

It Is similar to the compiler in that it translates high level language programs into machine code for execution but no object code is generated. An interpreter translates the program one statement at a time and if it is error free it executes the statement. This continues till the last statement in the program has translated and executed. Thus an interpreter translates and executes a statement before it goes to the next statement. When it encounters an error it will immediately stop the execution of the program.

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

Examples of compiler

- gcc
- clang
- javac
- go (compiler)

Some compiler runs before the program first run, but there are some case that compiler run after program started that called JIT (just in time).
Interpreter is program that executes source code or byte code, for example:

- ruby (interpreter)
- python (interpreter)
- php (interpreter)

Source Code files

When you write a program in C language, your instructions from the source code (or simply source file), C filenames have an extension .c. The part of the name before the period is called the base name and the part after the period is called the extension.

Object code, Executable code and Libraries

An executable file is a file containing ready to run machine code. C accomplishes this in two steps.
Compiling – The compiler converts the source code to produce the intermediate object code.
An **object file** is a **file** containing **object** code, meaning relocatable format machine code that is usually not directly executable. There are various formats for **object files**, and the same **object** code can be packaged in different **object files**. An **object file** may also work like a shared library.
The linker combines the intermediate code with other code to produce the executable file. C does this in a modular manner.
Every C file must be saved with a .c file extension.
Linker: it is a system software which combines One or more object files and possible some library code into either some executable some library or a list of error.
Loader: A program which loads the executable file to the primary memory of the machine.

Comments

Comments are non – executable program statements meant to enhance program readability and allow easier program maintenance, i.e. they document the program.

Types

Multiline Comments

Multiline comments have one or more lines of narrative within a set of comment delimiters. These comment delimiters are the begin comment /* and end comment */ markers. Anything between these two markers is considered a comment. The compiler ignores comments when

reading the source code. Lines 1 through 4 of Listing 1 show a multiline comment. For convenience, Lines 1 through 4 of Listing 1 are repeated here.

```
1: /*
2: * FileName: HowdyParner.cs
3: * Author:  Joe Mayo
4: */
```

Some languages allow embedded multiline comments, but C# does not. Consider the following example:

```
1: /*
2:  Filename:  HowdyPartner.cs
3:  Author:    Joe Mayo
4:   /*
5:      Initial Implementation:  04/01/01
6:      Change 1:         05/15/01
7:      Change 2:         06/10/01
8:   */
9: */
```

The begin comment on Line 1 starts a multiline comment. The second begin comment on line 4 is ignored in C# as just a couple characters within the comment. The end comment on Line 8 matches with the begin comment on line 1. Finally, the end comment on Line 9 causes the compiler to report a syntax error because it doesn't match a begin comment.

Single-Line Comments

Single-line comments allow narrative on only one line at a time. They begin with the double forward slash marker, //. The single-line comment can begin in any column of a given line. It ends at a new line or carriage return. Lines 6, 9, and 12 of Listing 1 show single-line comments. These lines are repeated here for convenience.

```
 6: // Program start class
 7: public class HowdyPartner
 8: {
 9:   // Main begins program execution
10:   public static void Main()
```

```
11:  {
12:    // Write to console
13:    System.Console.WriteLine("Howdy, Partner!");
```

Declaration statements

In C, all variables must be declared before they are used. Variable declarations ensure that appropriate memory space is reserved for the variables, depending on the data types of the variables. Line 6 is a declaration for an integer variable called num.

Assignment and Expression statements

An assignment statement uses the assignment operator "=" to give a variable on the operator's left side the value to the operator's right or the result of the expression on the right. The statement num =1; (Line 6) is an assignment statement.

Escape sequences

Escape sequences (also called back slash codes) are character combinations that begin with a backslash symbol (\) used to format output and represent difficult-to-type characters.

One of the most important escape sequences is \n, which is often referred to as the new line character. When the C compiler encounters \n, it translates it into a carriage return.
For example, this program:

```
#include<stdio.h>
        main()
        {
                printf("This is line one  \n");
                printf("This is line two \n");
                printf("This is line three");
           system(pause);
                return 0;
        }
```

displays the following output on the screen.

This is line one
This is line two
This is line three

Below are other escape sequences:

Escape sequence	**Meaning**
\a	alert/bell
\b	backspace

\n	new line
\v	vertical tab
\t	horizontal tab
\\	back slash
\’	Single quote (‘)
\”	Double quote (“”)
\0	null

Special character in C

,	Comma	*	Asterisk	'	Apostrophe	(	Left Parenthesis
&	.Ampersand	:	Colon	<	Opening Angle(Less than sign)	| .	Vertical Bar
.	Period	-	.Minus Sign	"	Quotation Marks	)	Right Parenthesis
;	Semicolon	?	Question Mark	>	Closing Angle(Greater than sign)	/	Slash
^	Caret	+	.Plus Sign	!	Exclamation Mark	[	Left Bracket
\	Backslash	~	Tilde	%	Percentage Sign	$	Dollar Sign
]	Right Bracket	{	Left Brace	}	Right Bracket	#	Number Sign
-	Underscore						

Variables

A variable is a memory location whose value can change during program execution. In C, a variable must be declared before it can be used. Variables can be declared at the start of any block of code.

A declaration begins with the type, followed by the name of one or more variables. For example, int high, low, results[20];

Types of Variables
There are two places where variables are declared: inside a function or outside all functions.

Variables declared outside all functions are called **global variables** and they may be accessed by any function in your program. Global variables exist the entire time your program is executing.

Variables declared inside a function are called **local variables.** A local variable is known to and may be accessed by only the function in which it is declared. You need to be aware of two important points about local variables.

The local variables in one function have no relationship to the local variables in another function. That is, if a variable called **count** is declared in one function, another variable called **count** may

also be declared in a second function – the two variables are completely separate from and unrelated to one another.

Local variables are created when a function is called, and they are destroyed when the function is exited. Therefore local variables do not maintain their values between function calls.

Constants

A constant is a value that does not change during program execution. In other words, constants are fixed values that may not be altered by the program.

Integer constants are specified as numbers without fractional components. For example –10, 1000 are integer constants.

Floating - point constants require the use of the decimal point followed by the number's fractional component. For example, 11.123 is a floating point constant. C allows you to use scientific notation for floating point numbers. Constants using scientific notation must follow this general form:
number E sign exponent
The number is optional. Although the general form is shown with spaces between the component parts for clarity, there may be no spaces between parts in an actual number . For example, the following defines the value 1234.56 using scientific notation.

```
123.456E1
```

Character constants are usually just the character enclosed in single quotes; 'a', 'b', 'c'.

```
ch = ‘z’;
```

Note:
There is nothing in C that prevents you from assigning a character variable a value using a numeric constant. For example the ASCII Code for ‘A ‘ is 65. Therefore, these two assignments are equivalent.

```
char ch;
ch = “A’;
ch = 65;
```

Types of Constants
Constants can be used in C expressions in two ways:

Directly
Here the constant value is inserted in the expression, as it should typically be.
For example:

```
Area = 3.14 * Radius * Radius;
```

The value **3.14** is used directly to represent the value of **PI** which never requires changes in the computation of the area of a circle

Using a Symbolic Constant
This involves the use of another C preprocessor, #define.

For example, **#define SIZE 10**
A symbolic constant is an identifier that is replaced with replacement text by the C preprocessor before the program is compiled. For example, all occurrences of the symbolic constant **SIZE** are replaced with the replacement text 10.

This process is generally referred to as macro substitution. The general form of the #define statement is;

#define **macro-name string**

Notice that this line does not end in a semi colon. Each time the **macro - name** is encountered in the program, the associated **string** is substituted for it. For example, consider the following program.

Declarations can be spread out, allowing space for an explanatory comment. Variables can also be initialised when they are declared. This is done by adding an equals sign and the required value after the declaration.

```
int high = 250;          /* Maximum Temperature */
int low = -40;           /* Minimum Temperature */
int results[20];    /* Series of temperature readings */
```

Scope of variables

Scope of variable refers to the visibility of a variable in a program. Meaning which parts of your program can see or use it.

A scope is a region of the program and broadly speaking there are three places, where variables can be declared −

Inside a function or a block which is called local variables,

In the definition of function parameters which is called formal parameters.

Outside of all functions which is called global variables.

Local Variables

Are variables declared inside a given procedure and can be accessed or used only in a procedure in which they are declared. These types of variables are only meaningful inside the procedure. No statements outside the procedure may reference local variables .A local variable is a variable that is given local scope. Local variable references in the function or block in which it is declared override the same variable name in the larger scope. In programming languages with only two levels of visibility, local variables are contrasted with global variables.

2) Global Variables

Variables declared for the entire program. Global variables can be utilized anywhere within the program block, whether inside of or external to the procedure. a global variable is a variable with global scope, meaning that it is visible (hence accessible) throughout the program, unless shadowed.

Variable Names

Every variable has a name and a value. The name identifies the variable and the value stores data. There is a limitation on what these names can be. Every variable name in C must start with a letter; the rest of the name can consist of letters, numbers and underscore characters.

C recognizes upper and lower case characters as being different (C is case- sensitive). Finally, you cannot use any of C's keywords like main, while, switch etc as variable names.

Rules for constant and variables

1) Variable or constant name shouldn't be a keyword it should be a user defined

2) Every variable or constant name should start with either underscore or alphabet

3) A variable or constant name can be made up of either underscore or alphabet as well as numerals and combination of both.

4) No special character are allowed within either constant or variable name except underscore

5) In case a variable or a constant name is made up of more than one word, the words

can either be fully joined or joined by use of underscore

Examples of legal variable names

x	result	outfile	bestyet
x1	x2	out_file	best_yet

Using printf() To Output Values

You can use printf( ) to display values of characters, integers and floating - point values. To do so, however, requires that you know more about the **printf( )** function.

For example:

printf("This prints the number %d ", 99);

displays **This prints the number 99** on the screen. As you can see, this call to the printf( ) function contains two arguments. The first one is the quoted string and the other is the constant 99. Notice that the arguments are separated from each other by a comma.

In general, when there is more than one argument to a function, the arguments are separated from each other by commas. The first argument is a quoted string that may contain either normal characters or formal specifiers that begin with a percent (%) sign.

Normal characters are simply displayed as is on the screen in the order in which they are encountered in the string (reading left to right). A format specifier, on the other hand informs **printf( )** that a different type item is being displayed. In this case, the **%d**, means that an integer, is to be output in decimal format. The value to be displayed is to be found in the second argument. This value is then output at the position at which the format specifier is found on the string.

If you want to specify a character value, the format specifier is %c. To specify a floating-point value, use %f. The %f works for both **float** and **double.** Keep in mind that the values matched with the format specifier need not be constants, they may be variables too.

Code	Format
%c	Character
%d	Signed decimal integers
%i	Signed decimal integers
%e	Scientific notation (lowercase 'e')
%E	Scientific notation (lowercase 'E')
%f	Decimal floating point
%s	String of characters
%u	Unsigned decimal integers
%x	Unsigned hexadecimal (lowercase letters)
%X	Unsigned hexadecimal (Uppercase letters)

Examples

1. The program shown below illustrates the above concepts. First, it declares a variable called **num**. Second, it assigns this variable the value 100. Finally, it uses **printf( )** to display **the value is 100** on the screen. Examine it closely.

```
#include <stdio.h>
main()
{
        int num;
        num = 100;
        printf(" The value is %d ", num);
  system("pause");
        return 0;
}
```

This program creates variables of types **char, float**, and **double** assigns each a value and outputs these values to the screen.

```
#include<stdio.h>
main()
{
        char ch;
        float f;
        double d;
        ch = 'X';
        f = 100.123;
        d  = 123.009;

        printf(" ch is %c ", ch);
        printf(" f  is %f ", f);
        printf(" d  is %f ", d);
        system("pause");
        return 0;
}
```

Inputting Numbers From The Keyboard Using scanf( )

There are several ways to input values through the keyboard. One of the easiest is to use another of C's standard library functions called **scanf( ).**

To use **scanf( )** to read an integer value from the keyboard, call it using the general form:
scanf("%d", &int-var-name);

Where int-var-name is the name of the integer variable you wish to receive the value. The first argument to **scanf( )** is a string that determines how the second argument will be treated. In this case the %d specifies that the second argument will be receiving an integer value entered in decimal format. The fragment below, for example, reads an integer entered from the keyboard.

```
int num;
scanf("%d", &num);
```

The **&** preceding the variable name means 'address of'. The values you enter are put into variables using the variables' location in memory. It allows the function to place a value into one of its arguments.

When you enter a number at the keyboard, you are simply typing a string of digits. The **scanf( )** function waits until you have pressed **<ENTER>** before it converts the string into the internal format used by the computer.

The table below shows format specifiers or codes used in the scanf () function and their meaning.

Code	**Meaning**
%c	Read a single character
%d	Read a decimal integer
%i	Read a decimal integer
%e	Read a floating point number
%f	Read a floating point number
%lf	Read a double
%s	Read a string
%u	Reads an unsigned integer

Examples

1. This program asks you to input an integer and a floating-point number and displays the value.

```
#include<stdio.h>
main()
{
        int num;
        float f;
        printf(" \nEnter an integer: ");
        scanf( "%d ", &num);
        printf("\n Enter a floating point number: ");
        scanf( "%f ", &f);
        printf( "%d  ", num);
        printf( "\n %f ", f);
  system("pause");
        return 0;
}
```

2. This program computes the area of a rectangle, given its dimensions. It first prompts the user for the length and width of the rectangle and then displays the area.

```
#include<stdio.h>
main()
{
        int len, width;
        printf("\n Enter length:  ");
        scanf ("%d ", &len);
        printf("\n Enter width :  " );
        scanf( " %d ", &width);
        printf("\n The area is %d ", len  * width);
    system("pause");
    return 0;
}
```

Exercises

Enter, compile and run the example programs.
Write a program that inputs two floating-point numbers (use type **float)** and then displays their sum.
Write a program that computes the volume of a cube. Have the program prompt the user for each dimension.

Example: Area of a circle

```
#include <stdio.h>
#define PI  3.14
main()
{
  float radius, area;
  printf("Enter the radius of the circle \n");
  scanf("%f", &radius);
  area = PI * radius * radius;  /* PI is a symbolic constant */
  printf("Area is %.2f cm  squared ",area);
  system("pause");
  return 0;
}
```

Note:
At the time of the substitution, the text such as 3.14 is simply a string of characters composed of 3, ., 1 and 4. The preprocessor does not convert a number into any sort of internal format. This is left to the compiler.

Revision Exercises

Discuss four fundamental data types supported by C, stating how each type is stored in memory.
Distinguish between a variable and a constant.
Suggest, with examples two ways in which constant values can be used in C expression statements.
Give the meaning of the following declarations;

(i) **char name[20];**
(ii) **int num_emp;**
(iii) **double tax, basicpay;**
(iv) **char response;**

What is the difference between a local and a global variable?
Write a program that will prompt you to enter the radius a cylinder. The code then calculates the volume of the cylinder.

C PROGRAM EXAMPLES.

A C code that calculates the average of 3 integers.

```
#include<stdio.h>
main()
{
 int x,y,z;
 float average,sum=0.0;

        printf("Enter x ");
        scanf("%d", &x);
printf("Enter y ");
        scanf("%d", &y);
printf("Enter z ");
        scanf("%d", &z);
sum=x+y+z;
        average = sum/3;
        printf("\n The average is %f \n", average);
        system("pause");
        return 0;
}
```

C Program to Find Volume and Surface Area of Sphere

```
#include <stdio.h>
#include <math.h>
 int main()
{
    float radius;
```

```
    float surface_area, volume;
    printf("Enter radius of the sphere : \n");
    scanf("%f", &radius);
    surface_area =  4 * (22/7) * radius * radius;
    volume = (4.0/3) * (22/7) * radius * radius * radius;
    printf("Surface area of sphere is: %.3f", surface_area);
    printf("\n Volume of sphere is : %.3f", volume);
    system("pause");
    return 0;

}
```

Program to find the volume of the cylinder

```
#include <stdio.h>
   int main()
   {
      int height=38;
      int radius=35;
      double pie=3.14285714286;
      double volume=pie*(radius*radius)*height;
      printf("Volume of the cylinder=%f",volume);
   }
```

C Program to Find the Volume and Surface Area of Cylinder

```
/*
 * C Program to Find the Volume and Surface Area of cylinder
 */
#include <stdio.h>
#include <math.h>

int main()
{

   float radius, height;
   float surface_area, volume;

   printf("Enter  value for  radius and height of a cylinder : \n");
   scanf("%f%f", &radius, &height);
   surface_area = 2 * (22 / 7) * radius * (radius + height);
   volume = (22 / 7) * radius * radius * height;
   printf("Surface area of cylinder is: %.3f", surface_area);
```

```
    printf("\n Volume of cylinder is : %.3f", volume);
    return 0;
}
```

C Program to calculate the perimeter of a square

```
#include<stdio.h>
int main()
{
float side,perimeter;
printf("enter side of square: ");
scanf("%f",&side);

perimeter=4*side;
printf("POS: %f\n",perimeter);
return 0;
}
```

C Program to calculate the area of a square

```
#include <stdio.h>
#include <conio.h>
int main(){
    float side, area;
    printf("Enter length of side of square\n");
    scanf("%f", &side);
    /* Area of Square = Side X Side */
    area = side * side;
    printf("Area of square : %0.4f\n", area);

    getch();
    return 0;
}
```

Circumference Of Circle C Example Program

```
#include<stdio.h>
float PI = 3.14;
int main(){
   float radius, ci;
   printf("\nEnter the radius of Circle : ");
   scanf("%f", &radius);
   ci = 2 * PI * radius;

   printf("\nCircumference of Circle : %f", ci);
   return (0);
}
```