VOL-I

TELNET TECHNOLOGIES

C PROGRAMMING

B-2 Mahaveer colony, Near Shivam Hospital ,Bhaskar Circle, Ratanada, Jodhpur

Cell: 9461839460 | [redacted]

C Programming

C is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis M. Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in U.S.A to develop the UNIX operating system.

The C Language is developed for creating system applications that directly interact with the hardware devices such as drivers, kernels, etc.

Today's most popular Linux OS and RDBMS MySQL have been written in C.

It was developed to overcome the problems of previous languages such as ALGOL60, CPL, BCPL and B.

C programming is considered as the base for other programming languages, that is why it is known as mother language.

It can be defined by the following ways:

1. Mother language
2. System programming language
3. Procedure-oriented programming language
4. Structured programming language
5. Mid-level programming language

Discuss about Generation of Computer Programming Language

TRANSLATORS

A program written in high-level language is called as source code. To convert the source code into machine code, translators are needed.

A translator takes a program written in source language as input and converts it into a program in target language as output. It also detects and reports the error during translation.

The different types of translator are as follows:

Assembler

Assembler is a translator which is used to translate the assembly language code into machine language code. It was the first translator.

Interpreter

Interpreter translates line by line and reports the error if encountered during the translation process. It directly executes the operations specified in the source program when the input is given by the user. It gives better error diagnostics than a compiler.

Compiler

Compiler is a translator which is used to convert programs in high-level language to low-level language. It translates the entire program and also reports the errors in source program encountered during the translation.

Differences between compiler and interpreter

Sl. No	Compiler	Interpreter
Working	Compiler first scans the whole program (Source Code) for error. If no error, the compiler will convert source code to machine code called Object Code.	Interpreter reads each line of Source Code statements line by line into machine Code and also gets it executed.
Speed	Programs run fast, because the translated version already available and can run directly	Program run slow, translation takes place first then the execution follows.
Debugging	Debugging is hard as the error messages are generated after scanning the entire program only	Debugging is easy, because t It stops translation when the first error is met.
Memory	Compiler based programs occupy less space in RAM during execution, because the translated version of the source code is already available and can run directly without the aid of any software	Interpreter base programs occupy more space in RAM during execution, because the source code as well as the interpreter program must reside in the memory
Safety of source code	Source code is safe because it is not required once the translated version is ready.	Source code is not safe because it required every time the program is run.
Step wise development	It takes an entire program	It takes a single line of code.
Error detection	No error can escape the eye of the compiler.	Certain erroneous statement may escape detection, because the interpreter checks only those statements for error that it executes.
Space	Compiler based programs occupy more space on disk	Interpreter based program occupy less space on disk.
Language	Programming languages like PASCAL, FORTRAN, COBOL C, C++, C#, SCALA, JAVA etc uses compilers.	Programming languages like B, BASIC, FOXPRO, Python, Ruby etc. uses interpreters.

The Library

The 'C' Library is a collection of Resources that act as building blocks in the development of 'C' programs.

There are 2 types of files available in the 'C' Library.

1.) **Header File** :- The prototype (declaration) of the functions are present in their respective header files, and must be included in your program to access them. C Standard library functions or simply C Library functions are inbuilt functions in C programming.

For example: If you want to use printf() function, the header file <stdio.h> should be included.

Some name of header files which is available in 'C' programming.

STDIO.H	CONIO.H	STRING.H	STDLIB.H	GRAPHICS.H	MATH.H
CTYPE.H	DIR.H	BIOS.H	DOS.H	ALLOC.H	TIME.H

2.) **Library Files:** These files contains the definition of Resources.

Library Files are available as pre compiled (object) code. With the extension of .obj /.lib

Variable in C Language

When we want to use some data value in our program, we can store it in a memory space and name the memory space so that it becomes easier to access it. Variable is the name of memory location.

C is a strongly typed language. What this means it that, the type of a variable cannot be changed.

RULES FOR NAMING C VARIABLE:

1 Length of variable name

ANSI C allows to have any number of character in the variable name, but considers only first 31 of them.

Other C compiler allow a maximum of 31 character in the variable name.

2 Variable name must begin with alphabet or underscore.

3 Variable name must not start with a digit.

4 Variable name can consist of alphabets, digits and special symbols like underscore _.

5 Blank or spaces are not allowed in variable name.

6 Keywords (reserved words) are not allowed as variable name.

7 Upper and lower case names are treated as different, as C is case-sensitive, so it is suggested to keep the variable names in lower case.

Types of Variables in C

There are many types of variables in c:

1. **local variable**:- A variable that is declared inside the function or block is called local variable.
2. **global variable**:- A variable that is declared outside the function or block is called global variable. Any function can change the value of the global variable. It is available to all the functions.
3. **static variable**:- A variable that is declared with static keyword is called static variable. It retains its value between multiple function calls.
4. **automatic variable**:- All variables in C that is declared inside the block, are automatic variables by default. By we can explicitly declare automatic variable using auto keyword.
5. **external variable**:- We can share a variable in multiple C source files by using external variable. To declare a external variable, you need to use extern keyword.

Keywords in C Programming Language:

1. Keywords are those words whose meaning is already defined by Compiler
2. Cannot be used as **Variable Name** , function name.
3. There are **32 Keywords** in C
4. C Keywords are also called as **Reserved words** .

32 Keywords in C Programming Language:

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

Data types

Fundamental Data Types

char
Int
float

Derived Data Types

Array
Pointers
Structure
Union
Enumeration

char :- use for storing single character values.

Type	Storage size	Value range
Char	1 byte	-128 to 127
signed char	1 byte	-128 to 127
unsigned char	1 byte	0-255

E.g.,

```
char  ans;

ans='Y';
```

single character constants must be enclosed within single quotes.

int:- used for storing integer values.

Type	Storage size	Value range
int	2 byte (short) 4 byte (long)	-32,768 to 32,767 -2,147,483,648 to 2,147,483,647
Unsigned int	2 byte 4 byte	0 to 65,535 0 to 4,294,967,295

Floating- point type

The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision –

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Constants/Literals

A constant is a value or an identifier whose value cannot be altered in a program. For example: 1, 2.5, "C programming is easy", etc.

As mentioned, an identifier also can be defined as a constant.

```
const double PI = 3.14
```

Here, PI is a constant. Basically what it means is that, PI and 3.14 is same for this program.

Operators

An operator is a symbol which operates on a value or a variable. For example: + is an operator to perform addition.

C programming has various operators to perform tasks including arithmetic, relational, logical, assignment, conditional and bitwise operations.

Operators in C programming

Arithmetic Operators:- Arithmetic Operators are used for performing arithmetic operations. There are two types of Arithmetic Operators.:

Unary Operators:-

Unary (they only operate on a single operand.)

Unary +

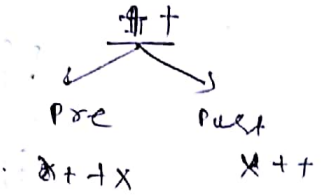
Unary –

& (Address extraction operator):-	Returns the address of a variable.
* (Dereferencing operator):	Pointer to a variable.
Sizeof ()	Returns the size of given data type
(<type cast>) type cast operator):-	use for temporary conversion of data type.
++ Increment	Increases the value by 1
-- Decrement	Decreases the value by 1.

Increment and Decrement operator must be either pre or post.

Binary operator: operate upon two operands.

+	(Addition Operator)
-	(Subtraction Operator
*	(Multiplication Operator)
/	(Division Operator)
%	(Modulus Operator)



x=10

Printf("%d", x++);

11, 12.

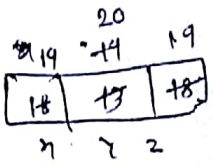
Printf("%d", x++);

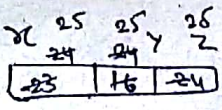
10, 11

x=10
y=19
z=x++;
y=++z;
x=y++;

z=19
y=11

x=17





x=18
y=13
x=x++
y=++z
z=y++;

Relational Operators

Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0

Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning of Operator	Example
&&	Logical AND. True only if all operands are true	If ((c == 5) && (d > 5))
||	Logical OR. True only if either one operand is true	If ((c == 5) || (d > 5))
!	Logical NOT. True only if the operand is 0	If !(c == 5)

C Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b

/=	a /= b	a = a/b
%=	a %= b	a = a%b

Bitwise Operators

During computation, mathematical operations like: addition, subtraction, addition and division are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Bitwise Shift left
>>	Bitwise Shift right

C Ternary Operator (?:)

A conditional operator is a ternary operator, that is, it works on 3 operands.

Conditional Operator Syntax

conditionalExpression ? expression1 : expression2

The conditional operator works as follows:

- The first expression conditionalExpression is evaluated first. This expression evaluates to 1 if it's true and evaluates to 0 if it's false.

Max = (a > b) ? a : b ;

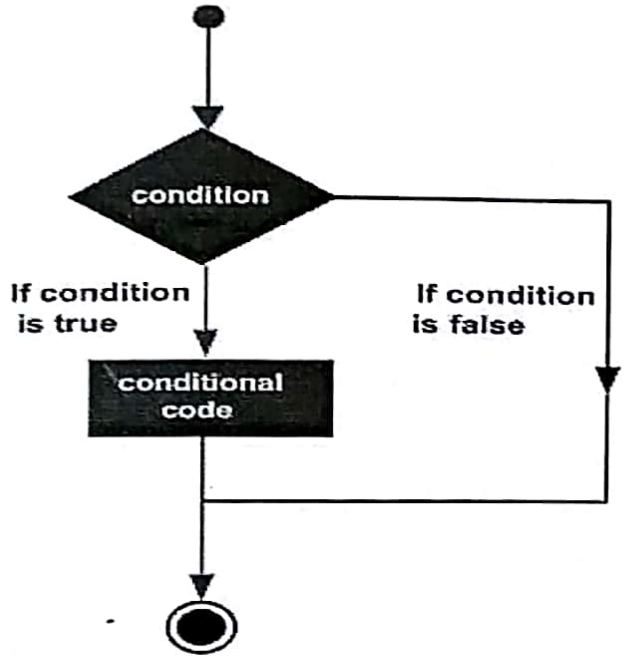
max = (x > y) ? x : y ;
True False

x = 7

C - Decision Making

Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Show below is the general form of a typical decision making structure found in most of the programming languages –



C programming language assumes any **1** and **non-null**values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.

C programming language provides the following types of decision making statements.

4 **switch statement**

A **switch** statement allows a variable to be tested for equality against a list of values.

```
If ( condition )

{
Statement 1;
Statement 2;
// lie within the scope of the condition, hence will be executed only if the condition is satisfied
}
Statement 3;
Statement 4;
// lie outside the scope of the condition, hence will be executed whether or not condition is satisfied.
```

If there is only one statement to execute depending on the condition, it may or may not be enclosed within curly brackets.

Switch statement

The switch statement is often faster than nested `if...else` (not always). Also, the syntax of switch statement is cleaner and easy to understand.

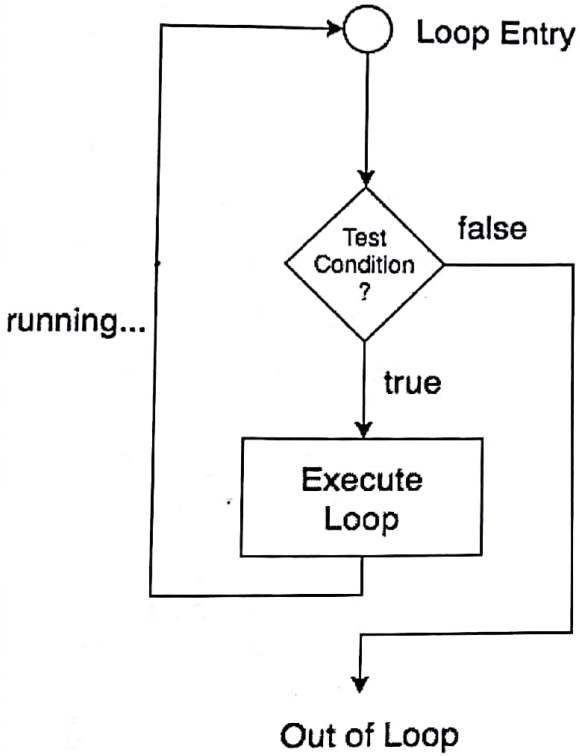
Syntax of switch...case

```
switch (n)
{
    case 1:
       // code to be executed if n is equal to 1;
        break;
   case constant2:
 // code to be executed if n is equal to constant2;
        break;
        .
        .
        .
    default:
        // code to be executed if n doesn't match any constant
}
```

Iteration (Loop)

In any programming language including C, loops are used to execute a set of statements repeatedly until a particular condition is satisfied.

The below diagram shows a loop execution,



As per the above diagram, if the Test Condition is true, then the loop is executed, and if it is false then the execution breaks out of the loop. After the loop is successfully executed the execution again starts from the Loop entry and again checks for the Test condition, and this keeps on repeating.

The sequence of statements to be executed is kept inside the curly braces { } known as the **Loop body**. After every execution of the loop body, **condition** is verified, and if it is found to be **true** the loop body is executed again. When the condition check returns **false**, the loop body is not executed, and execution breaks out of the loop.

Types of Loop

There are 3 types of Loop in C language, namely:

1. while loop
2. do while loop
3. for loop

while loop

while loop can be addressed as an entry control loop. It is completed in 3 steps.

- Variable initialization.(e.g. int x = 0;)
- condition(e.g. while(x <= 10))
- Variable increment or decrement (x++ or x-- or x = x + 2)

do while loop

In some situations it is necessary to execute body of the loop before testing the condition. Such situations can be handled with the help of **do-while** loop. **do** statement evaluates the body of the loop first and at the end, the condition is checked using while statement. It means that the body of the loop will be executed at least once, even though the starting condition inside **while** is initialized to be **false**.

```
do
{
    .....
    .....
}
while(condition);
```

Jumping Out of Loops

Sometimes, while executing a loop, it becomes necessary to skip a part of the loop or to leave the loop as soon as certain condition becomes **true**. This is known as jumping out of loop.

1) break statement

When **break** statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop.

```
while( condition check )
{
   statement-1;
   statement-2;
   if( some condition)
   {
      break;
   }
   statement-3;
   statement-4;
}
```

Jumps out of the loop, no matter how many cycles are left, loop is exited.

2) continue statement

It causes the control to go directly to the test-condition and then continue the loop process. On encountering **continue**, cursor leave the current cycle of loop, and starts with the next cycle.

```
while( condition check )
{
   statement-1;
   statement-2;
   if( some condition)
   {
      continue;
   }
   statement-3;
   statement-4;
}
```

Jumps to the next cycle directly.

Not executed for the cycle of loop in which continue is executed.

for loop

for loop is used to execute a set of statements repeatedly until a particular condition is satisfied. We can say it is an **open ended loop.**. General format is,

```
for(initialization; condition; increment/decrement)
{
    statement-block;
}
```

In **for** loop we have exactly two semicolons, one after initialization and second after the condition. In this loop we can have more than one initialization or increment/decrement, separated using comma operator. But it can have only one **condition**.

The **for** loop is executed as follows:

1. It first evaluates the initialization code.
2. Then it checks the condition expression.
3. If it is **true**, it executes the for-loop body.
4. Then it evaluate the increment/decrement condition and again follows from step 2.
5. When the condition expression becomes **false**, it exits the loop.

Nested for loop

We can also have nested **for** loops, i.e one **for** loop inside another **for** loop. Basic syntax is,

```
for(initialization; condition; increment/decrement)
{
for(initialization; condition; increment/decrement)
```