

2 Programming Logic

Problems solving

First and foremost step of solving a problem is knowing about the nature of problem in order to solve it. Almost all types of problems can be solved with computer. However, correct formulation of the problem in computer understandable terms is essential to formulate the problem and solve it using computer.

2.1 Understanding of problems, Feasibility and Requirement Analysis

In order to start to develop a solution to a problem using computer, we need to first understand the problem about its nature, complexities and others. Problems can be of various nature and we need to understand about the problems first to be able to solve it using computer. Once the problem has been understood and we can solve it using computer the feasibility of the problem solution needs to be determined. We can solve a problem using various logic but what needs to be worked out is the most feasible solution that can be derived with the least amount of complexities. Another major portion to solving problems using computer is the analysis of the requirements. The solution to the problem can be solved however the requirements required is beyond the possible technologies of today. Then the solution may not be feasible to the problem.

2.2 Design FlowChart and Algorithm

An algorithm is a solution to a computer programming problem. In other words a step by step procedure for developing a problem is called an algorithm. Algorithms can be written in two different ways.

1. **Pseudocode** English like steps that describe the solution. Pseudocode is an artificial and informal language that helps programmers develop algorithms. Pseudocode is a "text-based" detail (algorithmic) design tool. The rules of Pseudocode are reasonably straightforward. All statements showing "dependency" are to be indented. These include while, do, for, if, switch.

```
Set total to zero
Set grade counter to one
While grade counter is less than or equal to ten
    Input the next grade
    Add the grade into the total
Set the class average to the total divided by ten
Print the class average.
```

2. **FlowChart** Pictures Detailing with specific blocks detailing out the logical flow of the solution. For a better understanding of an algorithm, it is represented pictorially. The pictorial representation of an algorithm is called a Flow Chart. The algorithm and flowchart to add two numbers can be stated as:

- (a) Step1: Input numbers as a and b
- (b) Step2: $\text{Sum} = x + y$
- (c) Step3: Print the sum

Various symbols can be used to represent different actions like taking input, making decisions and connecting flowcharts.




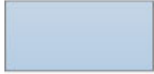

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

Figure 2: Flow chart Basic Symbols

The flow chart for the addition of two numbers whose algorithm has been stated above can be drawn as

2.3 Program Coding

Once algorithm and flowchart has been developed the task now remains is to write programs using some high level programming language. C, C++, Java, Python are the most popular programming language to develop programs. PHP and ASP remain a popular choice for developing web based applications.

2.4 Testing and Debugging

Once the program code has been written in a selected programming language of choice the next task to complete is the testing and debugging. Testing and debugging helps to find the problems associated with the program behavior under normal and abnormal circumstances. Extensive testing like white box testing and black box testing, integration testing needs to done before the program is deployed into the real world scenario.

2.5 Implementation

After a program has been written and tested it needs to be implemented to the target environment to solve the problem. Various needs of physical hardware and accessories required by the program to solve the intended problem needs to be present upon implementation. Once the program is implemented it starts to work.

2.6 Evaluation and Maintenance

The evaluation of the performance of the program needs to be done at frequent interval once the program or software is implemented. The advent of new technology, upscaling and downscaling of the business, request for the change from customers, finding of a new bug are the major reasons for the maintenance and change of the softwares. Once changes have been continuous monitoring of the software performance needs to done to discover the flaws in the software.

2.7 Documentation

The same workforce that developed the program may have left the project and gone in search of opportunities. So a program must be well documented in order for the new people to understand how the software was developed and how can it be modified. The documentation start from the very beginning of the problem formulation to the very end of the Evaluation and Maintenance.

3 Variables and Data Types

3.1 Constant and Variables

The smallest meaningful units in C programming is called C Token. Keywords, variables, various special characters fall under c tokens.

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called literals. Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are enumeration constants as well. Constants are treated just like regular variables except that their values cannot be modified after their definition.

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable. The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive.

Type	Size(Byte)	Range
char	1	-127 to 127
int	2	-32,767 to 32,767
float	4	$1*10^{-37}$ to $1*10^{37}$ six digit precision
double	8	$1*10^{-37}$ to $1*10^{37}$ ten digit precision
long int	4	-2,147,483,647 to 2,147,483,647

Table 1: Data types in C

3.2 Variable declaration in C

A variable declaration provides assurance to the compiler that there exists a variable with the given type and name so that the compiler can proceed for further compilation without requiring the complete detail about the variable. A variable definition has its meaning at the time of compilation only, the compiler needs actual variable definition at the time of linking the program.

```
#include <stdio.h>
extern int a, b;
extern int c;
extern float f;
int main () {
    int a, b;
    int c;
    float f;

    a = 10;
    b = 20;
```

BCANOTESNEPAL.com

```

    c = a + b;
    printf(" value of c : %d \n", c);
    f = 70.0/3.0;
    printf(" value of f : %f \n", f);
    return 0;
}

```

3.3 Rules for writing variable name in C

1. A variable name can have letters (both uppercase and lowercase letters), digits and underscore only.
2. The first letter of a variable should be either a letter or an underscore. However, it is discouraged to start variable name with an underscore. It is because variable name that starts with an underscore can conflict with a system name and may cause error.
3. There is no rule on how long a variable can be. However, the first 31 characters of a variable are discriminated by the compiler. So, the first 31 letters of two variables in a program should be different.

3.4 Input Output Function

C programming language provides many of the built-in functions to read given input and write data on screen, printer or in any file.

`scanf()` and `printf()` functions

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int i;
    printf(" Enter a value ");
    scanf("%d",&i);
    printf( " \nYou entered: %d",i);
    getch();
}

```

3.5 Operators in C

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators.

- Arithmetic Operator
- Relational Operator
- Logical Operator
- Bitwise Operator
- Assignment Operator
- Misc Operator

3.5.1 Arithmetic Operator

The basic operators for performing arithmetic are the same in many computer languages:

1. + Addition
2. - Subtraction
3. * Multiplication
4. / Division
5. % Modulus (Remainder)

For exponentiations we use the library function `pow`. The order of precedence of these operators is `% / * + -`. It can be overruled by parenthesis.

Division of an integer quantity by another is referred to as integer division. This operation results in truncation. i.e. When applied to integers, the division operator `/` discards any remainder, so `1 / 2` is 0 and `7 / 4` is 1. But when either operand is a floating-point quantity (type `float` or `double`), the division operator yields a floating-point result, with a potentially nonzero fractional part. So `1 / 2.0` is 0.5, and `7.0 / 4.0` is 1.75.

An operator that acts on a single operand to produce a new value is called a **unary operator**. The decrement and increment operators `- ++` and `-` are unary operators. They increase and decrease the value by 1.

`sizeof()` is another unary operator. The output given by the `sizeof()` operator depends on the computer architecture. The values stated by `sizeof()` operator down below are based on a 16-bit compiler.

```
1 int x, y;  
2 y=sizeof(x);
```

The value of `y` is 2. The `sizeof()` of an integer type data is 2, that of `float` is 4, that of `double` is 8, that of `char` is 1.

3.5.2 Relational Operator

`<` (less than), `<=` (less than or equal to), `>` (greater than), `>=` (greater than or equal to), `==` (equal to) and `!=` (not equal to) are relational operators. A logical expression is an expression connected with a relational operator. For example `'b'*b - 4*a*c < 0` is a logical expression. Its value is either true or false.

```
1 int i, j, k;  
2 i=1;  
3 j=2;  
4 K=i+j;
```

The expression `k > 5` evaluates to false and the expression `k < 5` evaluates to true.

3.5.3 Logical Operator

The relational operators work with arbitrary numbers and generate true/false values. You can also combine true/false values by using the Boolean operators, which take true/false values as operands and compute new true/false values. The three Boolean operators are:

1. `&&` AND

2. || OR

3. ! NOT

The && ('and') operator takes two true/false values and produces a true (1) result if both operands are true (that is, if the left-hand side is true and the right-hand side is true). The || ('or') operator takes two true/false values and produces a true (1) result if either operand is true. The ! ('not') operator takes a single true/false value and negates it, turning false to true and true to false (0 to 1 and nonzero to 0).

&& (and) and || (or) are logical operators which are used to connect logical expressions. Where as ! (not) is unary operator, acts on a single logical expression.

```
1 (a<5) && (a>2)
2 (a<=3) || (a>2)
```

In the example if a= 3 or a=6 the logical expression returns true.

3.5.4 Assignment Operator

These operators are used for assigning a value of expression to another identifier.

=, +=, -=, *=, /= and %= are assignment operators.

a = b+c results in storing the value of b+c in 'a'.

a += 5 results in increasing the value of a by 5.

a /= 3 results in storing the value a/3 in a and it is equivalent a=a/3

3.5.5 Conditional Operator

The operator ?: is the conditional operator. It is used as variable 1 = expression 1 ? expression 2 : expression 3. Here expression 1 is a logical expression and expression 2 and expression 3 are expressions having numerical values. If expression 1 is true, value of expression 2 is assigned to variable 1 and otherwise expression3 is assigned.

```
1 int a,b,c,d,e;
2 a=3; b=5; c=8;
3 d=(a<b) ? a : b;
4 e=(b>c) ? b : c;
```

The evaluation of the expression results the value of d = 3 and e = 8.

3.5.6 Bitwise Operator

C has a distinction of supporting special operator known as bitwise operator for manipulation of data at bit level. These operators are used for testing bits or shifting them right or left. Bitwise operator may not be applied to float or double. Following are the bitwise operators.

- & bitwise AND
- | bitwise OR
- ^ bitwise exclusive OR
- << shift left
- >> shift right

3.6 Operator Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, $x = 7 + 3 * 2$; here, x is assigned 13, not 20 because operator $*$ has a higher precedence than $+$, so it first gets multiplied with $3*2$ and then adds into 7. An arithmetic expression without parenthesis will be evaluated from *left to right* using the rules of precedence. There are two distinct priority levels of arithmetic operators in C.

- Higher Priority $*$ /
- Lower Priority $+$ -

The expression $x = a - b / 3 + c * 2 - 1$ where $a = 9$, $b = 12$ and $c = 3$ will be evaluated as
step1: $x = 9 - 12 / 3 + 3 * 2 - 1$ Higher priority operators left to right division first
step2: $x = 9 - 4 + 3 * 2 - 1$ Higher priority operators left to right multiply
step3: $x = 9 - 4 + 6 - 1$ Lower priority operators left to right subtraction
step4: $x = 5 + 6 - 1$ Lower priority operators left to right addition
step5: $x = 11 - 1$ Lower priority operators left to right subtraction
step6: $x = 10$ Final Result

Rules for Evaluation of Expression

- First, parenthesized sub expression from left to right are evaluated.
- If parentheses are nested, the evaluation begins with innermost sub expression.
- The precedence rule is applied in determining the order of operands in evaluating sub expressions.
- Arithmetic expressions are evaluated from left to right using the rules of precedence.
- When parentheses are used, the expressions within the parentheses assume highest priority.

3.7 Simple Input Output Function

For inputting and outputting data we use library function .the important of these functions are `getch()`, `putchar()`, `scanf()`, `printf()`, `gets()`, `puts()`.

3.7.1 `getchar()` function

It is used to read a single character (`char` type) from keyboard. The syntax is **`char variable name = getchar();`**

For reading an array of characters or a string we can use `getchar()` function.

```
1 #include <stdio.h>
2 #include <conio.h>
3 void main( )
4 {
5     char place [80];
6     int i;
7     for(i = 0; ( place [i] = getchar( )) != '\n', ++i);
8 }
```


3.7.2 putchar() function

It is used to display single character. The syntax is **putchar(char c);**

```
1 #include <stdio.h>
2 #include <conio.h>
3 void main()
4 {
5     char alphabet;
6     printf("Enter an alphabet");
7     putchar('\n');
8     alphabet=getchar();
9     putchar(alphabet);
10 }
```

3.7.3 scanf() function

This function is generally used to read any data type- int, char, double, float, string. The syntax is

scanf (control string, list of arguments)

The control string consists of group of characters, each group beginning % sign and a conversion character indicating the data type of the data item. The conversion characters are c,d,e,f,o,s,u,x indicating the type resp. char decimal integer, floating point value in exponent form, floating point value with decimal point, octal integer, string, unsigned integer, hexadecimal integer. ie, "%s", "%d" etc are such group of characters.

```
1 #include<stdio.h>
2 #include<conio.h>
3 void main( )
4 {
5     char name[30], line;
6     int x;
7     float y;
8     .....
9     .....
10 scanf("%s%d%f", name, &x, &y);
11 scanf("%c", line);
12 }
```

3.7.4 printf() function

This is the most commonly used function for outputting a data of any type. The syntax is

printf(control string, list of arguments)

Here also control string consists of group of characters, each group having % symbol and conversion characters like c, d, o, f, x etc.

```
1 #include<stdio.h>
2 #include<conio.h>
3 void main()
4 {
5     int x;
6     scanf("%d",&x);
7     x=x*x;
8     printf("The square of the number is %d", x);
9 }
```

Note that in this list of arguments the variable names are without & symbol unlike in the case of scanf() function. In the conversion string one can include the message to be displayed. In the above example "The square of the number is" is displayed and is followed by the value of x.

3.7.5 gets() and puts() function

The C library function gets() reads a line from stdin and stores it into the string pointed to by str. It stops when either the newline character is read or when the end-of-file is reached, whichever comes first.

```
1 #include<stdio.h>
2 #include<conio.h>
3 int main()
4 {
5     char str[50];
6     printf("Enter a string:");
7     gets(str);
8     printf("You entered: %s", str);
9     return(0);
10 }
```

The C library function int puts() writes a string to stdout up to but not including the null character. A newline character is appended to the output.

```
1 #include <stdio.h>
2 #include <string.h>
3 int main()
4 {
5     char str1[15];
6     char str2[15];
7     strcpy(str1, "Nepal");
8     strcpy(str2, "Online");
9     puts(str1);
10    puts(str2);
11    return(0);
12 }
```