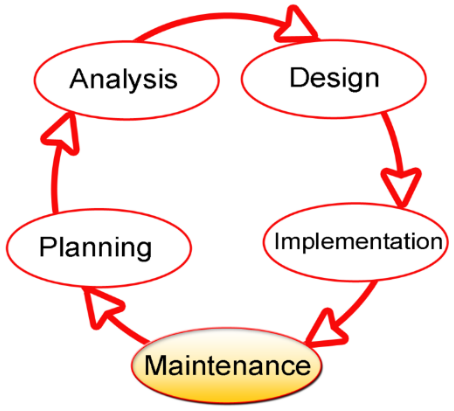
Chapter 7: System Development



The systems development life cycle (SDLC) is a term used in systems engineering, information systems and software engineering to describe a process for planning, creating, testing, and deploying an information system.

A systems development life cycle is composed of a number of clearly defined and distinct work phases which are used by systems engineers and systems developers to plan for, design, build, test, and deliver information systems.

Design: In systems design, the design functions and operations are described in detail, including screen layouts, business rules, process diagrams and other documentation. The output of this stage will describe the new system as a collection of modules or subsystems.

Design can be shown using algorithms.

Algorithm: an effective method expressed as a finite list of well-defined instructions for solving a problem.

An algorithm must possess the following properties:

finiteness: The algorithm must always terminate after a finite number of steps.

definiteness: Each step must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case.

input: An algorithm has zero or more inputs, taken from a specified set of objects.

output: An algorithm has one or more outputs, which have a specified relation to the inputs.

effectiveness: All operations to be performed must be sufficiently basic that they can be done exactly and in finite length.

An algorithm may be expressed in a number of ways, including:

natural language: usually verbose and ambiguous

flow charts:

- avoid most (if not all) issues of ambiguity
- Difficult to modify without specialized tools
- Largely standardized

pseudo-code:

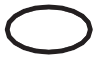
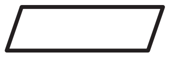
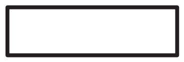
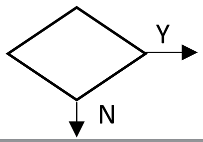
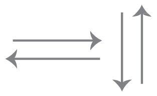
- also avoids most issues of ambiguity
- Vaguely resembles common elements of programming languages
- No particular agreement on syntax

programming language: tend to require expressing low-level details that are not necessary for a high-level understanding.

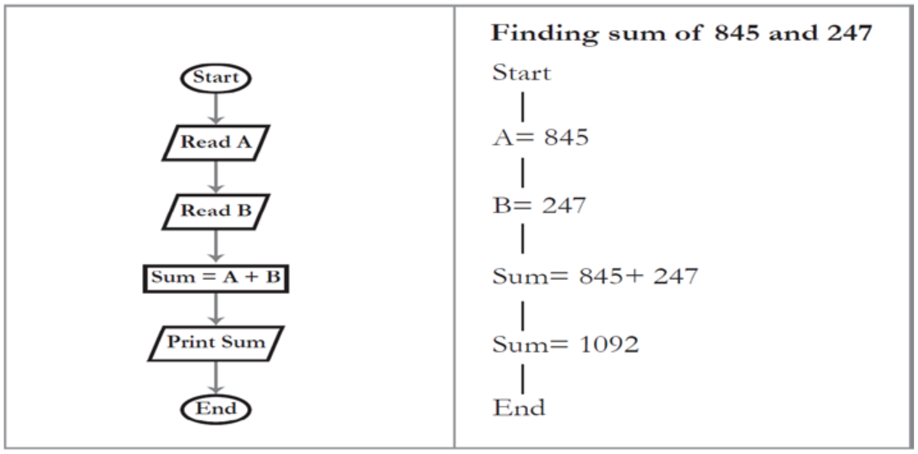
1. Flowchart

- is a picture (graphical representation) of the problem
- solving process.

A flowchart gives a step-by-step procedure for solution of a problem.

Flowchart symbols	Geometric shape	Purpose
Ellipse		Ellipse is used to indicate the start and end of a flowchart. Start written in the ellipse indicates the beginning of a flowchart. End or Stop or Exit written in the ellipse indicates the end of the flowchart.
Parallelogram		A parallelogram is used to read data (input) or to print data (output).
Rectangle		A rectangle is used to show the processing that takes place in the flowchart.
Diamond		A diamond with two branches is used to show the decision making step in a flowchart. A question is specified in the diamond. The next step in the sequence is based on the answer to the question which is “Yes” or “No”.
Arrows		Arrows are used to connect the steps in a flowchart, to show the flow or sequence of the problem solving process

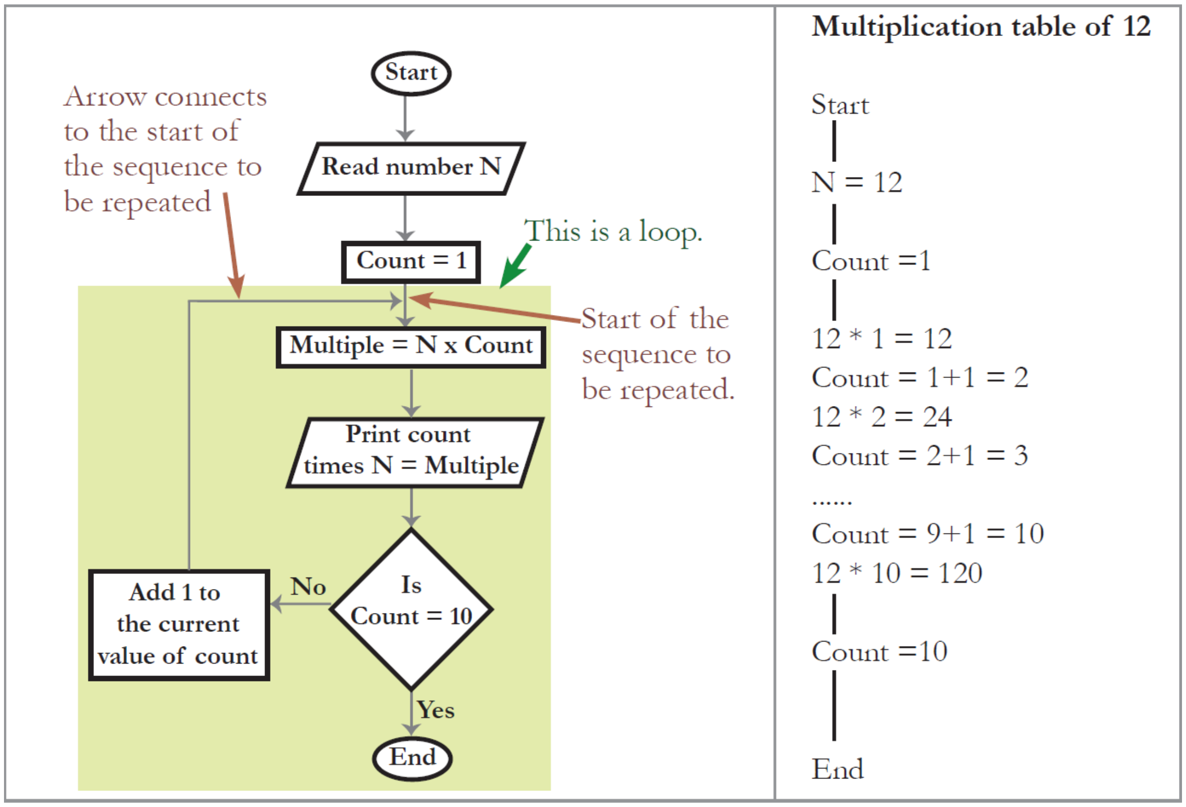
Example1 : Ana algorithm that takes two numbers and finds the sum.



Finding sum of 845 and 247

```
Start
  |
A= 845
  |
B= 247
  |
Sum= 845+ 247
  |
Sum= 1092
  |
End
```

Example 2: An algorithm that list the multiplication table for a given number.



2. Pseudocode

Artificial and Informal language that helps programmers to plan an algorithm. It is somewhat similar to everyday English with a feel of programming language.

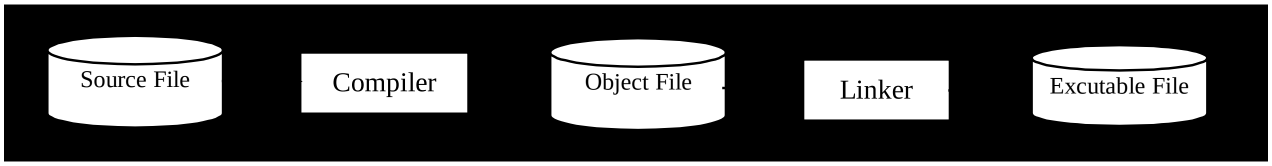
Example: a Pseudocode for a program that calculates the factorial for a given number.

```
algorithm Factorial takes number N

# Computes N! = 1 * 2 * . . . * N-1 * N, for N >= 1
# Pre:  input value must be a non-negative integer.
#
   number nFactorial

   nFactorial := 1

   while N > 1
            nFactorial := nFactorial * N
            N := N - 1
   endwhile

   display nFactorial
   halt
```

Chapter 8: Introduction to C Programming

Introduction

A program written in a high-level language is called **a source program (or source code).** Since a computer cannot understand a source program. Program called a compiler is used to translate the source program into a machine language program called an **object program**. The object program is often then **linked** with other supporting library codes before the object can be executed on the machine.



- **Source program**
  - The form in which a computer program, written in some formal programming language, is written by the programmer.
  - Can be compiled automatically into object code or machine code or executed by an interpreter.
  - C source programs have extension ‘. c’
- **Object program**
  - Output from the compiler
  - Equivalent machine language translation of the source program
  - Files usually have extension ‘.obj’ or (.o) in C
- **Executable program**
  - Output from linker/loader
  - Machine language program linked with necessary libraries & other files
  - Files usually have extension ‘.exe’
- **Linker**
  - A program that pulls other programs together so that they can run.
  - Most programs are very large and consist of several modules*.*
  - Even small programs use existing code provided by the programming environment called libraries.
  - The linker pulls everything together, makes sure that references to other parts of the program (code) are resolved.

Only about 100 instructions that the computer understands - Different programs will just use these instructions in different orders and combinations. The most valuable part of learning to program is learning how to think about arranging the sequence of instructions to solve the problem or carry out the task.

Putting the Instructions Together

- Sequential Processing
  - A List of Instructions
- Conditional Execution
  - Ifs
- Repetition
  - Looping / Repeating
- Stepwise Refinement / Top-Down Design
  - Breaking Things into Smaller Pieces
- Calling Methods / Functions / Procedures / Subroutines
  - Calling a segment of code located elsewhere
  - Reuse of previously coded code segment

C Programming

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It is a Procedural Oriented Programming Language (POP) i.e., it focuses on the procedure in which the program is written. It is one of the oldest programming languages almost 3 decades old.

C is one of the most popular programming languages. It is being used on several different software platforms. C is an procedural language. It was designed to be compiled using a relatively straightforward compiler, to provide low-level access to memory, to provide language constructs that map efficiently to machine instructions, and to require minimal run-time support.

Relations to other languages

Many later languages have borrowed directly or indirectly from C, including C++, Java, JavaScript, C#, Unix's C shell. These languages have drawn many of their control structures and other basic features from C. Most of them are also very syntactically similar to C in general.

Characteristics of C

1. A high level programming language enables the programmer to concentrate on the problem at hand.
2. C has only 32 keywords.
3. Makes extensive use of function calls.
4. C is well suited for structured programming.
5. Facilitates low level(bitwise) programming.
6. C is a core language as many other programming languages(like c++, java etc) are based on C.
7. C is a portable language i.e a C program written for one computer can be run on another computer with little modification.
8. C is an extensible language as it enables the user to add his own functions to C library.

Structure of a C program

Writing the first C program

```
#include
int main()
{
  printf(”\n Welcome to the world of C”);
  return 0;
}
```

Output

```
Welcome to the world of C
```

Program Explanation

#include

This is a preprocessor command that comes as the first statement in our code.The # include statement tells the compiler to include the standard input/output library i.e stdio.h in the program. So by simply including this file in our code we can use these functions directly.

int main ()

Execution starts from main() and int is the return value of the main function. The two curly braces {} are used to group all the related statements of the function main.

printf ()

This function is defined in stdio.h file and is used to print text on the screen. Like in this program the line written in printf in double quotes is automatically printed on the output screen i.e Welcome to the world of C. ‘\n’ is an escape sequence and represents a new line.

return 0;

This is a return command that is used to return the value 0 to the operatinf system to give an indication that there were no errors during the execution of the program.

Note: Every statement in the main function ends with a semicolon ;

Keywords: are the reserved words in programming and are part of the syntax.

Identifiers: are the names that are given to various program elements such as variables, symbolic constants and functions.
Some rules are to be followed:

1. Identifier name must be a sequence of letter and digits, and must begin with a letter.
2. The underscore character (‘_’) is considered as letter.
3. Names shouldn't be a keyword (such as int , float, if ,break, for etc).
4. Both upper-case letter and lower-case letter characters are allowed. However, they're not interchangeable.
5. No identifier may be keyword.
6. No special characters, such as semicolon,period,blank space, slash or comma are permitted.

Basic data types in C

C has a concept of 'data types' which are used to define a variable before its use. The definition of a variable will assign storage for the variable and define the type of data that will be held in the location.
The value of a variable can be changed any time.

C has the following basic built-in datatypes.

1. int
2. float
3. double
4. char
5. Void: - Void is a data type that holds no value. Its size in bytes is 0. The range is valueless.

Variables and Constants

- Variables

An entity that varies through the programming is called a variable. These are names given to locations in memory which can be integer, real or character constant. The type of variables depend upon the constants it can hold ie an integer variable will hold integer constants.

There are some rules which have to be followed while constructing variables. These rules hold true for every type of variable.

1. It can be a combination of 1 to 31 characters (alphabets, numbers or underscores). Some compilers allow a combination of up to 247 characters but it is recommended to stick to 31 characters.
2. First character has to be an alphabet or an underscore.
3. There can’t be any blanks or commas in variable name.
4. You can’t use any special character except underscore ex abc_xyz

The C compiler can identify each variable as it makes compulsory for the user to declare the type of variable it is. This is done in the beginning of the program.

Example, int si,add ; float ci ; etc

NOTE: There are some predefined keywords in C compiler. It is advised to not use variable names similar to these.

- Numeric Variables

Numeric Variable can be used to store either integer values or floating point values. Numeric Variables can also be associated with modifiers like short, long, signed, and unsigned.

When we do not specify the signed/ unsigned modifier, C language automatically takes it as a signed variable.

- Character Variables

Character variables can include any letter from any alphabet or from the ASCII chart and numbers 0-9 that are given within single quotes. For eg:- 2 is considered an Integral value whereas ‘2’ is considered a character variable.

Declaring Variables

Each variable to be used in the program must be declared.

To declare a variable, specify the data type of the variable followed by its name. This data type indicates the kind of data that the variable will store. Variable names should always be meaningful and must reflect the purpose of their usage in the program. In C, variable declaration always ends with a semicolon,

For example:-

```
int roll_no;

float marks;

char grade;

double amount;

unsigned short int add_no;
```

C allows multiple variables of the same type to be declared in one statement.

```
float temp_in_celcius , temp_in_farenheit;
```

is legal in c

In C variables are declared at three basic places:

1. When a variable is declared inside a function it is known as local variable.
2. When a variable is declared in the definition of function parameters it is known as formal parameter.
3. When the variable is declared outside all functions it is known as global variable.

Note: - A variable cannot be of type void

Initializing Variables

While declaring the variables, we can also initialize them with some value.

For example: -

```
int roll_no = 7;

float salary =1000;
```

Constants

Constants are identifiers whose value does not change.Variables can change their value at any time but constants can never change their value.Constants are used to define fixed values like Pi or charge on an electron so that their value does not get changed in the program even by mistake.

We can use alphabets (A,B,C,d,e…) , numbers (1,2,3…) and characters or special symbols (-,*,&…) while using C. These when combined properly form constants and variables. There are mainly two types of C Constants:

1. Primary Constants
2. Secondary Constants

Rules for constructing integer constants:

1. Must have at least one digit.
2. Can be either positive or negative.
3. **Should not** have a decimal point.
4. If there is no sign before a digit it is taken as positiv.
5. No commas or blanks within.
6. Allowable range is -32768 to 32767.

Well actually the range depends on the compiler ie if it is a 16-bit or 32-bit or 64-bit, the range keeps on increasing respectively.

Rules for constructing real constants:

1. Often called floating point constants.
2. Must have at least one digit.
3. Must have a decimal.
4. Could be either positive or negative.
5. Default sign is positive.
6. No commas or blanks within.

Real constants can also be expressed in exponential form. The part appearing before **e** is called **mantissa**and following it is called **exponent**. These rules must be taken care of before expressing in exponential form:

1. **Mantissa** and **exponential** part should be separated by **e**.
2. **Mantissa** part may be negative or positive.
3. Default sign is positive.
4. There must be a digit after exponent; it may be positive or negative.
5. Ranges from -3.4e38 to 3.4e38.

Rules for constructing character constants:

1. It can be a single digit, a single alphabet, a single special symbol. It should be enclosed within two inverted commas going towards left ie, ‘A’.
2. The maximum length can be of 1 character.

Declaring Constants

To declare a constant, precede the normal variable declaration with const keyword and assign its value. For example, const float pi= 3.14.
The constant keyword specifies the value of pi which cannot change.
However another way to designate a pre processor command defines with a preceded # symbol.
#define PI 3.141459
#define service tax 0.12
Ex:

```
#include
/* Calculating area and circumference of circle */


int main()
{
   int rad;          // variable
   float PI = 3.14;  // constant
   float area, ci;


   printf("\n Enter radius of circle: ");
   scanf("%d", &rad);


   area = PI * rad * rad;
   printf("\nArea of circle : %f ", area);
   ci = 2 * PI * rad;

   printf("\nCircumference : %f ", ci);

   return (0);

}
```

Explaining the program:

1. We have used /* */ these, which are used by the programmer to write comments. It is not necessary to use it. But it helps the other person to understand the program more easily.
2. **main( )** is a collective name given to a set of statements. This name has to be **main( )**, it cannot be anything else. All statements that belong to **main( )** are enclosed within a pair of braces { }. And main is a function about which will study later.
3. Any variable used must be declared before ie float pi=3.14.
4. ';' it is used to terminate the statement.
5. **printf** is used to display the calculated area and circumference. It is a library function. The general form for printf() is, printf("",); .
6. Format string can contain %d for containing integer values, %f for containing real values, %c for containing characters.
7. printf can also be used to print results of an expression ie a combination of constants, variables and operators. < br> Ex: 3, 5+y-c, a+b-c*d which would be, printf(“d,”3, 5+y-c, a+b-c*d);

The Decision Control Structure

In previous chapter we read about how statements are executed sequentially. In this chapter we will study about how one can use a set of instructions for one situation and the other set for a more general situation. This can be done by decision control structure in which can be implemented in C using:

1. The if statement
2. The if-else statement
3. The conditional operators

The if statement

General form of **if** statement:

```
If (this condition is true)

    execute this statement;
```

Whatever follows if statement tells the compiler that it is a decision control instruction. The condition in if is always enclosed in parentheses and if true the statement is executed and if not it is skipped.

The user can create decisions using relational operators using the relational operators we can compare two values, equate them, use greater or lesser than etc here is a list.

NOTE: = and == are two different operators. = is used as an assignment operator whereas == is a relational operator ie it is used for comparison.

```
/*Example of if statement*/
main()
 {
   int num;
   printf(“Enter any number less than 100”);
   scanf(“%d”, &num);
      if(num<100);
          printf(“You are my servant HAHAHA”);
  }
```

Let’s explain the program a little bit to you. In the above program the statement you are my servant HAHAHA gets executed only if you enter a number less than 100 if you don’t the execution the statement doesn’t take place and the program stops there and then only.

The if statement can also have arithmetic statements ie,

1. If (a=10)
   printf(“I told you that it will work”);
2. If (-5)
   printf(“even this works”);

NOTE: Only non-zero constants can be used in the if condition.

We can write multiple statements in if statement using the following method.

```
/*calculation of discount*/
main()
  {
    int qty, dis = 0 ;
    float rate, tot ;
    printf ( "Enter quantity and rate " ) ;
    scanf ( "%d %f", &qty, &rate) ;
    if ( qty > 1000 )
       {
         dis = 10 ;
         printf(“discount = %d”, dis);
       }
  }
```

The statements that are to be executed are placed in between the parentheses if this is not done then the exact statement after the **if** is executed on satisfaction this means that the default scope of **if** is the statement right after it.

The if-else statement

The **if** statement alone is not enough what if we want to execute a different set of statements that doesn’t satisfy the **if** condition for this we use the **if-else**statement.

Example: To calculate the raise in salary of an employee by looking at his/her year of joining, year of service.

```
/*calculation of raise in salary*/
main()
  {
    int yoj,cy,yos,raise,cs;
    float ns;
    printf(“Enter year of joining and current year”);
    scanf(“%d %d”, &yoj, &cy)
   yos = cy – yoj;
   printf(“Enter current salary”);
   scanf(“%d”, &cs);
   if(yos>5)
      {
        raise = 27%
        printf(“Raise=%d”, raise);
      }
   else
      {
        raise = 15%
        printf(“Raise=%d”, raise);
      }
   ns = cs + cs*raise/100;
   printf(“New salary =  %f”, ns);
}
```

As you can see in the above program we have used if-else statement to calculate the new salary of an employee. If the condition comes out to be false then the else condition is executed.

NOTE

1. The statements upto the if not including else form the ‘if block’ similarly , ‘else block’ is formed.
2. else is written exactly below if and the statements succeeding are intended towards right to make the program understandable.
3. If we want to execute only one statement then pair of parenthesis could have been dropped. To override the default scope one uses a pair of parentheses

The Nested if-elses

We can also write an if-else statement within an if block or an else block, this is called nesting of **if**s.

Example:

```
main()
   {
     int x;
     printf(“Enter any number less than 10”);
     scanf(“%d”, &x);
     if(x==5)
        printf(“you entered the highest factor of 10”);
     else
       {
         If(x==2)
         printf(“you entered the smallest factor of10);
         else
         printf(“you are an idiot”);
       }
   }
```

As you can see in this program, if the first statement in**if** is false then the compiler checks for the **if** statement in the **else block** if even this fails then the compiler goes to the **else** statement and executes it. Here, it is only an example you can build if statement within **if block** too.

This is how nesting of if-elses takes place.

Different Forms of if

```
if ( condition )
do this ;



if ( condition )
{
  do this ;
  and this ;
}



if ( condition )
do this ;
else
do this ;



if ( condition )
{
   do this ;
   and this ;
}
else
{
    do this ;
    and this ;
}



if ( condition )
do this ;
else
{
   if ( condition )
   do this ;
else
{
    do this ;
    and this ;
}
}



if ( condition )
{
  if ( condition )
      do this ;
  else
  {
    do this ;
    and this ;
  }
}
else
do this ;
```

Use of Logical Operators

C allows its users to use three logical operators namely &&, ||, ! these are AND, OR and NOT operators respectively.

Remember AND(&&) , OR(||) we can't use &,| they have different meaning, you have to always use them in pair.

AND(&&) and OR(||) operators help the user to combine two conditions within an if statement.

Let's understand this with the help of an example.
In this example we will give students cash prize by calculating their overall percentage in 5 subjects.

```
/*Giving cash prize according to their percentage*/


main()

   {

     int s1,s2,s3,s4,s5,cash_prize;

     float per;

     printf("enter marks in 5 subjects");

     scanf("%d %d %d %d %d %d", &s1, &s2, &s3, &s4, &s5);

     per= (s1+s2+s3+s4+s5)/5;

     if(per>=85)

       {

         cash_prize=3500;
```

```
        printf("Cash prize given =%d", &cash_prize);
      }
    if((per>=70) && (per<85))
      {
       cash_prize=3000;
       printf("Cash prize given =%d", &cash_prize);
      }
    if((per>=60) && (per<70))
      {
        cash_prize=2000;
        printf("Cash prize given =%d", &cash_prize);
      }
    else
      printf("You are not given any cash prize");
  }
```

As you can see in the above program we have used AND operator to combine to conditions and if they come out to be true then printf is executed accordingly.

There are some advantages of using this, which are:

1. The program doesn't go on intending towards right.
2. Mismatching of if-else doesn't occur.

We can write the same program using the else if block ie,

```
if(per>=85)
      {
        cash_prize=3500;
        printf("Cash prize given =%d", &cash_prize);
      }
    else if(per>=70)
      {
       cash_prize=3000;
       printf("Cash prize given =%d", &cash_prize);
      }
```

```
     else if(per>=60)
       {
         cash_prize=2000;
         printf("Cash prize given =%d", &cash_prize);
       }
     else
       printf("You are not given any cash prize");
```

As you can see in this program the indentation reduces and every else is associated with its previous of.

Now we shall discuss the NOT(!) operator. When we use the ! operator it reverses the meaning of the statement ie it will turn non-zero value to zero value and vice-versa. Ex: !(y>10), this means that we can use any value of y for which it is less than 10 or in other words not y greater than 10.

The Conditional Operators

'?' and ':' these are known as conditional operators or as ternary operators as they take three arguments. These can be used in place of if-else in some cases.
Their general form is:
Expression1? Expression2 : Expression3

Which means that if expression1 is true then the value returned will be expression2 and if false it will return expression3, by true or false it means that the value should be non-zero or zero respectively.

Example:

```
    int f,g;
    printf("Enter value of f");
    scanf("%d", &f);
    g = (f>6 ? 5 : 0);
```

If the entered value is more than 6 then 5 will be stored in g else 0 will be stored in it.

The GOTO Statement

The goto statement is used to transfer control to a specified label . However, the label must reside in the same function and can appear only before one statement in the same function. Syntax as follows-