

# UNIT – II

## INTRODUCTION TO C

# Contents

Role of programming languages, Need to study programming languages, Characteristics of Programming Languages, Fundamentals of C- Programming - Character Set, Identifiers and keywords, Data types, Constants, Variables, Operators, Expression, statements, Library Functions, Pre-processor directives. Data Input and Output, Control Structures- Decision making,- if, if else, switch, Control Structures- Iterative- while ,do-while, for, break and continue statements, Structure of C program, Coding conventions.

# Role of Programming Languages

3

- A programming language is formal language that specifies a set of instructions that can be used to produce various kinds of output .
- Programming languages generally consist of instructions for a computer .
- Programming languages can be used to create programs that implement specific algorithms .

# What is Programming Language

4

- A programming language is a notational system for describing computation in machine-readable and human-readable form.
- A programming language is a tool for developing executable models for a class of problem domains.

# What is Programming Language

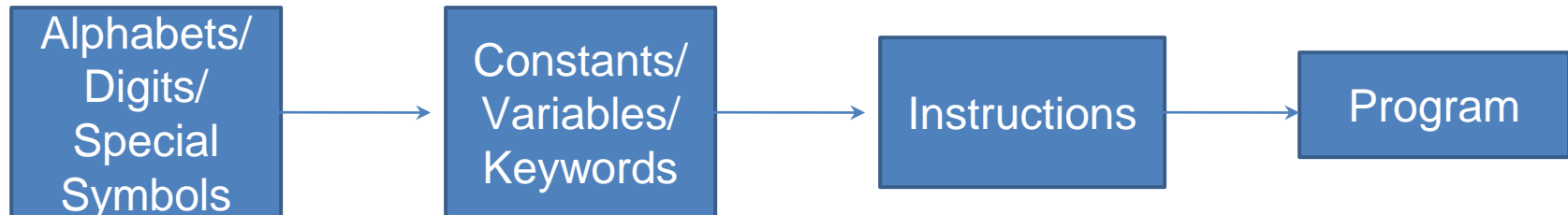
5

- English is a **natural language**. It has words, symbols and grammatical rules.
- A programming language also has words, symbols and rules of grammar.
- The grammatical rules are called **syntax**.
- Each programming language has a different set of syntax rules.

## Steps in learning English Language-



## Steps in learning C-Language-



# Need to Study Programming Language

- To improve your ability to develop effective algorithms.
- To improve your use of existing programming languages.
- To increase your vocabulary of useful programming constructs.
- To allow a better choice of programming language.
- To make it easier to learn a new language.
- To make it easier to design a new language.

# Characteristics of Programming Languages

- ❑ The language must allow the programmer to write simple, clear and concise programs.
- ❑ It must be simple to use so that a programmer can learn it without any explicit training.
- ❑ It must be platform independent. That is, the program developed using the programming language can run on any computer system.
- ❑ The Graphical User Interface (GUI) of the language must be attractive, user-friendly, and self-explanatory.
- ❑ The function library used in the language should be well documented so that the necessary information about a function can be obtained while developing application.



# Characteristics of Programming Languages

## (Cont....)

- ❑ Several programming constructs supported by the language must match well with the application area it is being used for.
- ❑ The programs developed in the language must make efficient use of memory as well as other computer resources.
- ❑ The language must provide necessary tools for development, testing, debugging, and maintenance of a program. All these tools must be incorporated into a single environment known as Integrated Development Environment (IDE), which enables the programmer to use them easily.
- ❑ The language must be consistent in terms of both syntax and semantics.

# History and Features of C

10

- ❑ Dennis Ritchie is the creator of C
  - ❑ Created at Bell Laboratories
- ❑ Portable Language
  - ❑ C language is machine independent. Source Code written using C can be compiled on any machine(i.e. platform independent)
- ❑ Structured Language
  - ❑ problem is solved using a divide and conquer approach

# Constants

11

- ❑ A Constant is a value that never changes during program execution.
- ❑ Constants are given name and are referred by the given name.

Ex.  $\pi = 3.142$

# Variables

12

## Hold the data in your program

- ❑ A variable is an entity that can change during program execution

## Rules for Variable Names

- ❑ The first character in variable name must be an alphabet
- ❑ No other special character except underscore is allowed
- ❑ No blanks or commas are allowed
- ❑ Variable names are case sensitive
- ❑ Keywords cannot be used as variable names

# Variables (cont'd)

13

Ex : int a; -----variable declaration  
int a=10----var definition

amount\_in\$  
2many  
**are**  
if  
iTotal

Which of these  
variable names  
are  
valid ones?

# Program Structure

14

## A sample C Program

```
#include<stdio.h>
int main()
{
    int a;
    --other statements
}
```

# Header Files

15

- ❑ The files that are specified in the include section is called as header file
- ❑ These are precompiled files that has some functions defined in them
- ❑ The functions can be called in the program by supplying parameters
- ❑ Header file is given an extension .h
- ❑ C Source file is given an extension .c

# What is the purpose of header file in C?

16

- A **header file** is a **file** with extension **.h** which contains **C** function declarations and macro definitions to be shared between several source **files**.
- Header file is used in the program by including it, with the C preprocessing directive ' `#include` ' .



# Main Function

17

- This is the entry point of a program
- When a file is executed, the start point is the main function
- From main function the flow goes as per the programmers choice.
- There may or may not be other functions written by user in a program
- Main function is compulsory for any c program

# Writing the first program

18

```
#include<stdio.h> — header file
int main() ← main function
{
    printf("Hello");
    return 0;
}
```

- This program prints Hello on the screen when we execute it

# Running a C Program

19

- ❑ Type a program
- ❑ Save it
- ❑ Compile the program – This will generate an exe file (executable)
- ❑ Run the program (Actually the exe created out of compilation will run and not the .c file)
- ❑ In different compiler we have different option for compiling and running. We give only the concepts.

# Comments

20

- ❑ Comments are used to document programs and improve readability
- ❑ In C a comment will start with `/*` and ends with `*/`

**Syntax:** `/* Comments */`

```
/*This is a single line comment */
```

```
/* This is a multi line  
comment in C */
```

```
/******  
*
```

```
* This style of commenting is used for functions
```

```
*****  
/
```

- ❑ Only C style comments should be used

# File Header Block

21

- All *source* and *header files* must contain at the beginning of file, a section providing information about the source or the header file

- **Format:**

```
/******
```

```
* File : <filename>
```

```
* Description : <description>
```

```
* Author : <author>
```

```
* Date : <Date>
```

```
*****/
```

## File Footer Block

- All files should have this footer at the end of the file

```
/******
```

```
* End of <filename>
```

```
*****/
```

# Indentation of Code

22

- **Indentation** is the practice by Software Engineers to use spaces or tabs consistently in every line of code to group lines together based on their scope for easy readability
- An indented code looks better and can be understood easily
- The code in any line should not exceed 80 columns

## Compiler and Linker Errors

- If a program does not follow the syntax of a language then the compiler raises errors

**Example:** Missing semicolon

- When the linker is not able to find a piece of code the linker errors are generated

**Example:** Variable or function referenced, but not defined anywhere in code

# Data Types

23

- Data types determine the following:
  - Type of data stored
  - Number of bytes it occupies in memory
  - Range of data
  - Operations that can be performed on the data
- C supports the following data types:
  - `int` - for storing whole numbers
  - `char` - for storing character values, represents a single character
  - `float` - for storing fractional values
  - `double` - for storing fractional values

**Note:** `float` can store up to 6 digits of precision

`double` can store up to 12 digits of precision

- C supports the following modifiers along with data types:
  - **short**
  - **long**
  - **signed**
  - **unsigned**



# Range of Data Types

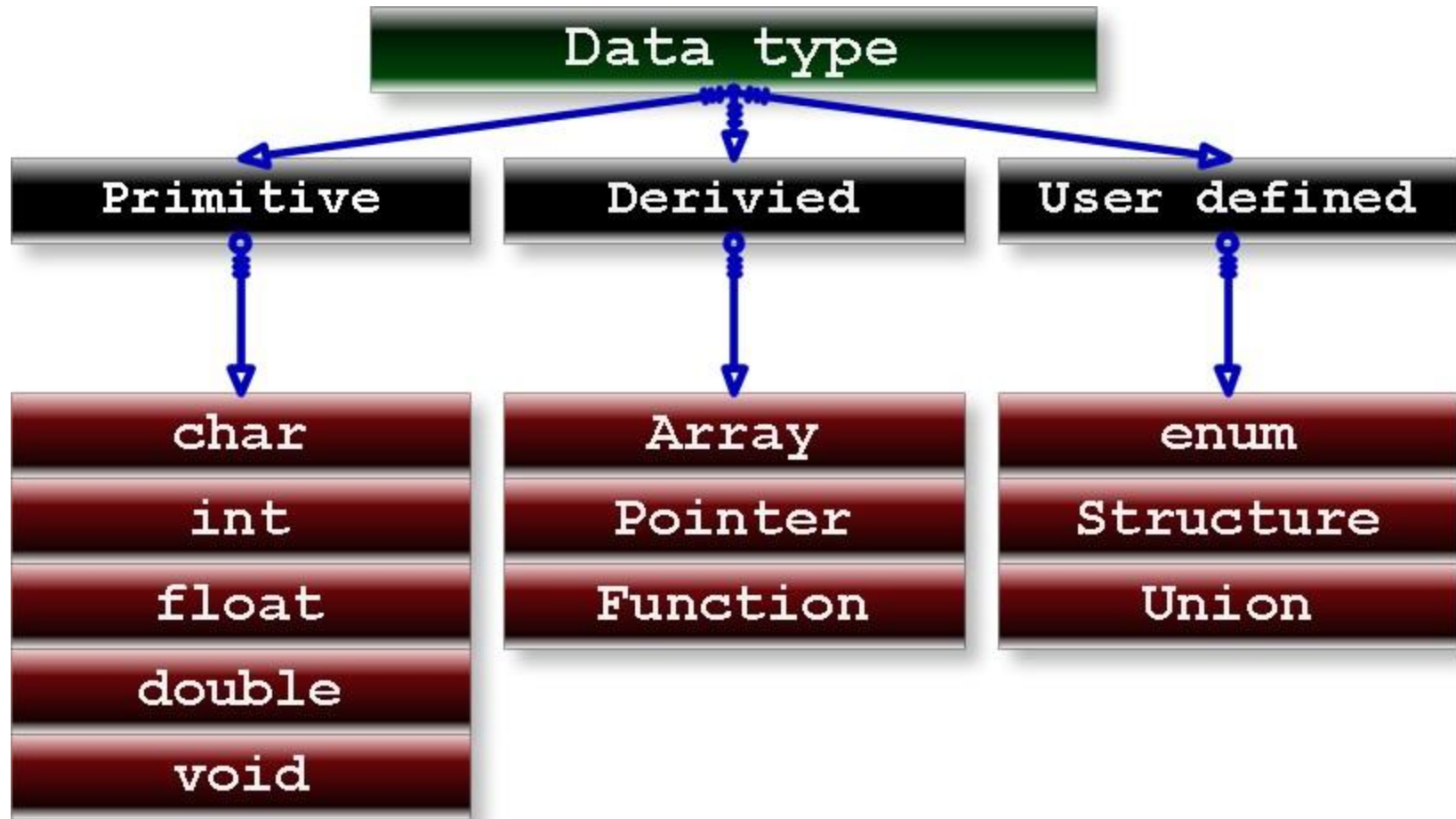
25

Data Types	Sizes in byte	Sizes in bits	Range formula $2^n-1$	Ranges
int	4 bytes	32bits	$2^{32}-1$	-2,147,483,648 to 2,147,483,647
unsigned int	4 bytes	32 bits	$2^{32}-1$	0 to 4294967295
float	4 bytes	32 bits	$2^{32}-1$ (5 points)	$3.4 \times 10^{-38}$ to $3.4 \times 10^{+38}$
double	8 bytes	64 bits	$2^{64}-1$ (15 points)	$1.7 \times 10^{-308}$ to $1.7 \times 10^{+308}$
long double	10 bytes	80 bits	$2^{80}-1$ (19 points)	$1.7 \times 10^{-4932}$ to $1.7 \times 10^{+4932}$
char	1 byte	8 bits	$2^8-1$	0 to 255

Note: Number of bytes and range given to each data type is platform dependent

# Data Types in C

26



# Variables

27

- Variables are data that will keep on changing
- Declaration  
    <<Data type>> <<variable name>>;  
    int a;
- Definition  
    <<varname>>=<<value>>;  
    a=10;
- Usage  
    <<varname>>  
    a=a+1;      //increments the value of a by 1

# Declaration of Variables

28

## Syntax:

**data-type [variable-name list]**

**data-type variable-name1 , variable-name2, ... ;**

## Example:

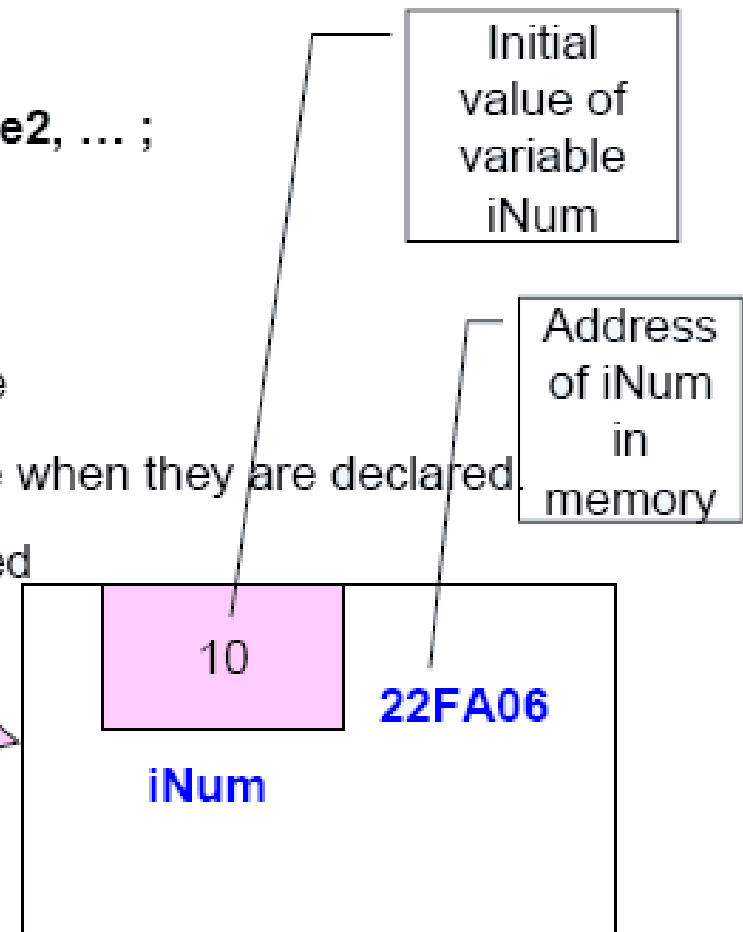
```
int iNum;
```

declares a variable of `int` data type

The variables will contain some garbage value when they are declared.

A variable can be **initialized** when it is declared

```
int iNum = 10;  
float fData = 2.3F ;  
char cChoice = 'Y';
```



# Keywords

29

- **Keywords** are predefined, reserved words used in programming that have special meanings to the compiler.
- **Keywords** are part of the syntax and they cannot be used as an identifier.

# Standard ANSI C Keywords

30

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

# Identifiers

31

- Identifiers are names for entities in a C program, such as variables, arrays, functions, structures, unions and labels.
- An identifier can be composed only of uppercase, lowercase letters, underscore and digits, but should start only with an alphabet or an underscore.

# Rules for constructing identifiers

32

- The first character in an identifier must be an alphabet or an underscore and can be followed only by any number alphabets, or digits or underscores.
- They must not begin with a digit.
- Uppercase and lowercase letters are distinct. That is, identifiers are case sensitive.
- Commas or blank spaces are not allowed within an identifier.
- Keywords cannot be used as an identifier.
- Identifiers should not be of length more than 31 characters.
- Identifiers must be meaningful, short, quickly and easily typed and easily read.

**Valid** identifiers: total sum average \_x y\_ mark\_1 x1

**Invalid** identifiers: 1x -> It begins with a digit  
char -> It is reserved word  
x+y -> It is a special character



# Operators In C

33

- Assignment operator ( = )
- Arithmetic operators ( + , - , \* , / , % )
- Relational operators ( > , >= , < , <= , == , != )
- Logical operators ( ! , && , || )
- Address operator ( & )
- Increment and Decrement operators ( ++ , -- )
- Compound Assignment Operators ( = , += , -= , /= , \*= , %= , • )
- `sizeof` operator

# Use of Modulus (%) Operator

34

- Used to find the remainder after integer division
- The operands that are supplied to this operator should always be integers
- The operator returns an integer value
- Using 'float' for any of the operands will result in a compiler error

## Example:

Remainder = Number % 4 ;

If the variable 'Number' has a value 21, then the resultant value in 'Remainder' will be 1

# Type Casting

35

- Temporary conversion of one data type into another.
- In some situations, the compiler will automatically convert one data type into another.

## Example:

```
float Result;
```

```
Result = 7 / 2 ;
```

The variable Result will store 3.0 instead of 3.5

To get the 'float' value, the expression should be:

```
Result = 7.0 / 2; or
```

```
Result = 7 / 2.0; or
```

```
Result = 7.0 / 2.0; or
```

```
Result = (float) 7 / 2; or
```

```
Result = 7 / (float) 2;
```

# Precedence of Arithmetic Operators

36

## Operator Priority

\* , / and % Highest

+ and - Lowest

The expression that is written within parenthesis is given highest priority

## Evaluate the following expression:

Using  $a = 5$ ,  $b = 3$ ,  $c = 8$  and  $d = 7$

$$b + c / 2 - (d * 4) \% a$$

$$b + c / 2 - 28 \% a$$

$$b + 4 - 28 \% a$$

$$b + 4 - 3$$

$$7 - 3$$

4

B	O	D	M	A	S
Brackets	Orders	Divide	Multiply	Add	Subtract
( ) { } [ ]	$x^2$ $\sqrt{x}$	$\div$ or	$\times$	$+$ or	$-$

# Relational Operators

37

- Used to compare two values and also called as **comparison operators**
- Expressions that contain relational operators are called as **relational expressions**
- A relational operator returns either zero or a non-zero value
- If the expression is true then it returns a non-zero value (>0)
- If the expression is false then it returns zero

## •Example:

**1500 > 700** returns 1 (true)

**1500 < 700** returns 0 (false)

Operator	Use	Example
<	Less than	if (a<b)
<=	Less than or equal to	if (a<=b)
>	Greater than	if (a>b)
>=	Greater than or equal to	if (a>=b)
==	Equal	if (a==b)
!=	Not equal	if (a!=b)

# Logical Operators

38

- Used to combine two or more relational expressions
- An expression involving logical operators is called as a **logical expression**

Operator	Description	Example
&&	Logical AND	(iNumber1 > 10) && (iValue1 <= 100)
	Logical OR	(iJobCode == 1)    (dSalary > 10500)
!	Logical NOT	!(iJobCode == 5)

Expression-1	Operator	Expression-2	Result
true	&&	true	true
true	&&	false	false
false	&&	true	false
false	&&	false	false
true		true	true
true		false	true
false		true	true
false		false	false
true	!		false
false	!		true

# Increment and Decrement Operators

- Operators ++ and -- are called as increment and decrement operators
- These operators increment or decrement the variable's value by 1
- They are also called as *unary operators* because they have only one operand to operate
- If the operator is used before the operand, it is *prefix* and if the operator is used after the operand, it is *postfix*

## Example:

++ Value and -- Value is called as *prefix*

Value++ and Value-- is called as *postfix*

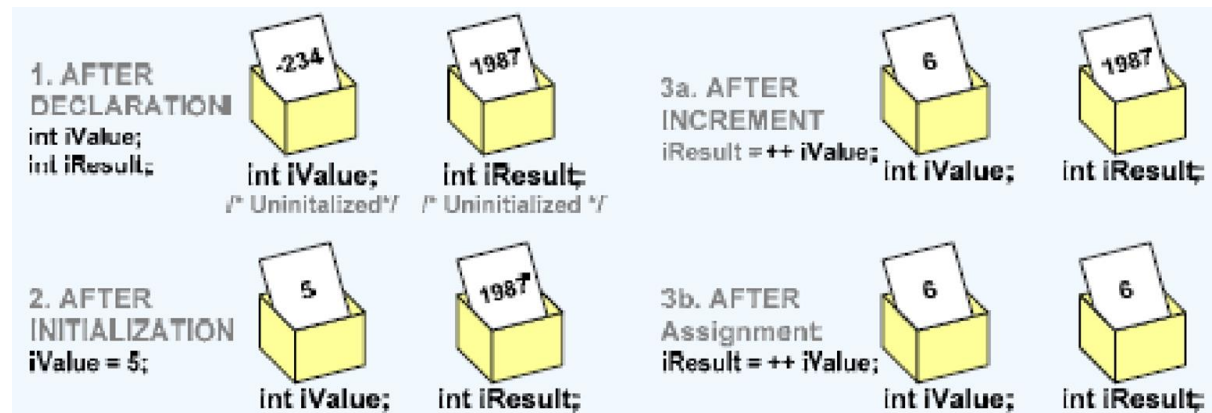
# Difference Between Prefix and Postfix

40

- Prefix operator first increments / decrements and then makes the assignment

## Example:

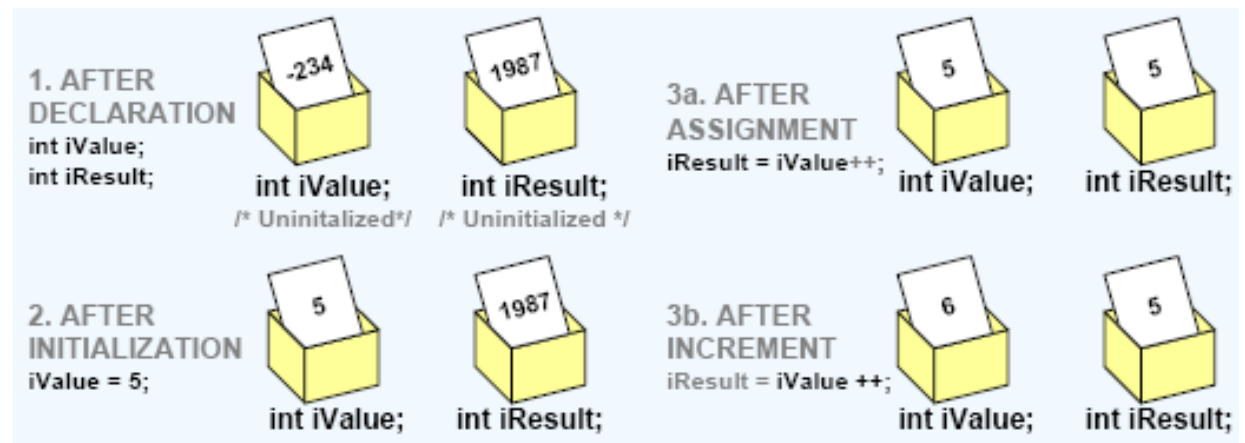
```
int iValue;  
int iResult;  
  
iValue = 5;  
iResult = ++iValue;
```



- Post fix operator makes the assignment and then increments/decrements the Value

## Example:

```
int iValue;  
int iResult;  
  
iValue = 5;  
iResult = iValue++;
```

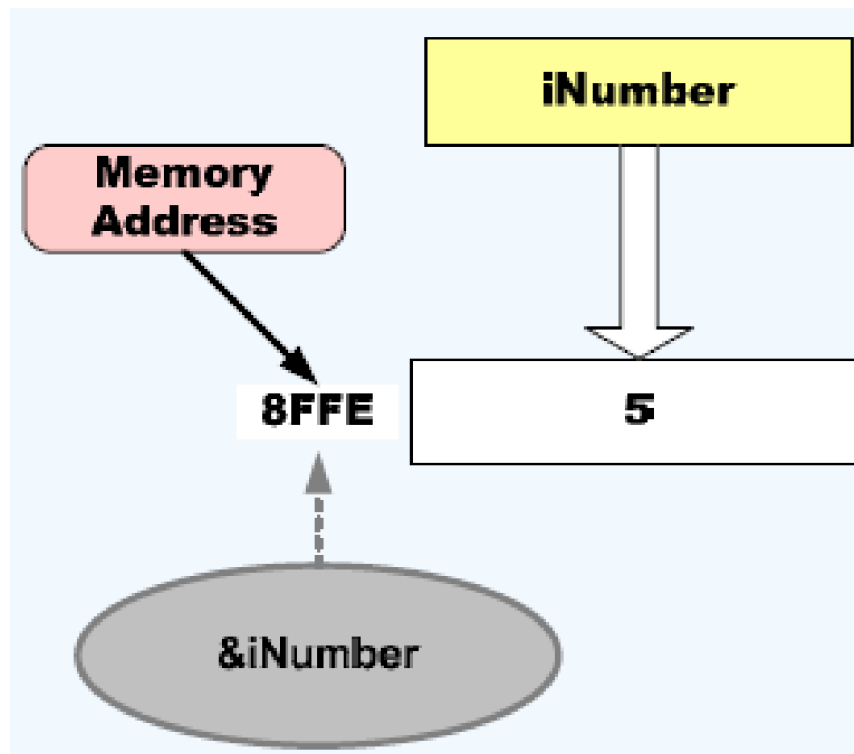




# Address of Operator

41

- Ampersand (&) is the “address of” operator
- It is used to fetch the memory address of a variable



# Formatted Output Using printf

42

- Writes onto screen (standard output)

## Syntax

`printf("Conversion Specifier-list", variable-1, variable-2,.....);`

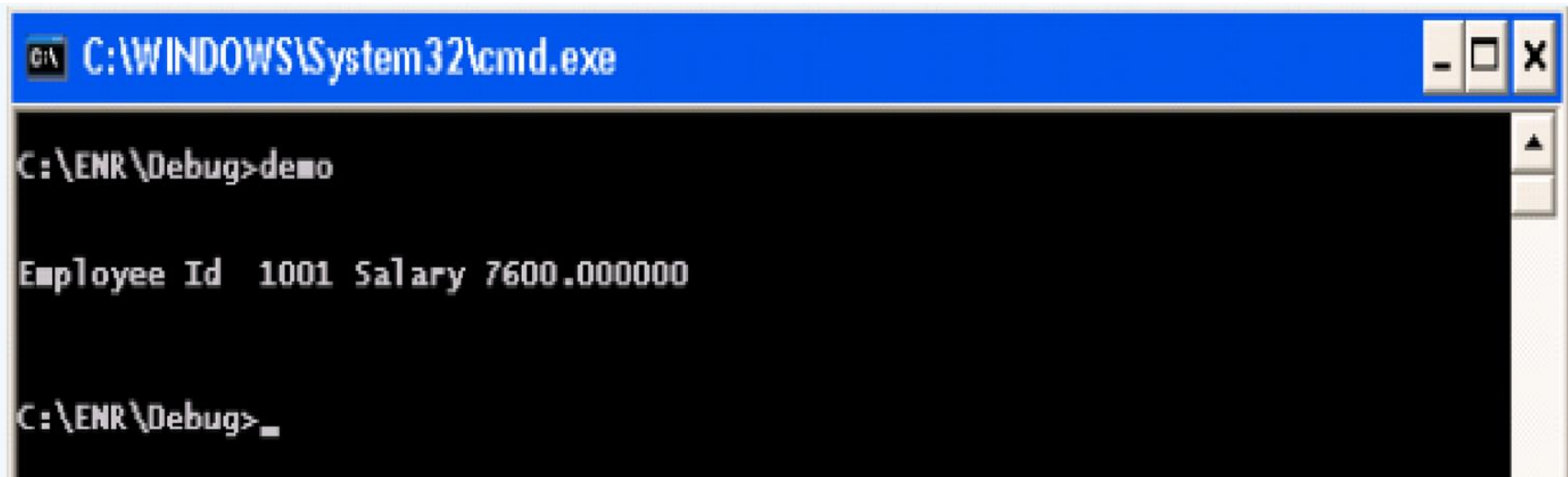
Conversion Specifier	Purpose	Example
%d	To print a signed integer	<code>printf("%d", iValue);</code>
%x	To print an integer as in Hex format	<code>printf("%x", iValue);</code>
%f	To print a float value	<code>printf("%f", fValue);</code>
%c	To print a character (both signed and unsigned)	<code>printf("%c", cChoice);</code>
%u	To print an unsigned integer	<code>printf("%u", iResult);</code>
%ld	To print a signed long integer	<code>printf("%ld", lNumber);</code>
%lu	To print an unsigned long integer	<code>printf("%lu", lFactorial);</code>
%lf	To print a double value	<code>printf("%lf", dAverage);</code>
%s	To print a string (Strings are discussed later)	<code>printf("%s", acEmpName);</code>
%x	To print a hex value	<code>printf("%x", iNumber);</code>
%%	To print % sign	<code>printf ("Percentage %d%%", iScore);</code>

# Formatted Output Using printf (cont'd)

43

## Example:

```
int EmployeeId = 1001;  
double Salary = 7600.00;  
printf("Employee Id %d" , EmployeeId);  
printf("Salary %lf", Salary);
```



```
C:\WINDOWS\System32\cmd.exe  
C:\ENR\Debug>demo  
  
Employee Id  1001 Salary 7600.000000  
  
C:\ENR\Debug>_
```

# Scanf()

44

- Read input from screen (standard input)

## Syntax

`scanf("format Specifier-list",&variable-1, &variable-2,.....);`

## Example

```
void main()
{
    int a;
    printf("Enter a number");
    scanf("%d",a);
}
```

# Formatted Output Using printf (cont'd)

45

- An escape sequence is interpreted to have a special meaning in the screen output
- All the escape sequences must be preceded by a back slash (\)
- Escape sequences are non printable characters
- Escape sequences are generally used with 'printf' function

Escape Sequence	Purpose
\n	New line character. This moves the cursor to the next line
\t	Prints a sequence of blank spaces.
\\	Prints back slash (\).
\"	Prints the double quote (")
\'	Prints a single quote (').
\a	Causes an audible sound on the computer

# Formatted Output Using printf (cont'd)

46

## Example:

```
int EmployeeId = 1001;  
double Salary = 7600.00;  
printf("Employee Id %d\n", EmployeeId);  
printf("Salary %lf\n", Salary);
```



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\System32\cmd.exe". The prompt is at "C:\ENR\Debug>demo". The output of the program is displayed as follows:

```
C:\ENR\Debug>demo  
  
Employee Id 1001  
Salary 7600.000000  
  
C:\ENR\Debug>_
```

# Declaring and Using Character Variables

47

## Syntax:

```
char variablename1, variablename2,...;
```

## Example:

```
char Alphabet;
```

```
char Status, Number ;
```

- The value can be assigned by enclosing it in a single quote ( ' ' )

## Example:

```
Alphabet = 'W';
```

```
Status = 'y';
```

- A character variable can be assigned with a numeric value by directly assigning a number

## Example:

```
Number = 77;
```

- The above statement can also be written as Number = 'M';

# The ASCII Character Set

48

- ❑ Character data is represented in a computer by using standardized numeric codes which have been developed.
- ❑ The most widely accepted code is called the **American Standard Code for Information Interchange (ASCII)**.
- ❑ The ASCII code associates an integer value for each symbol in the character set, such as letters, digits, punctuation marks, special characters, and control characters.
- ❑ ASCII value for capital A is 65, B is 66,....., Z is 90.
- ❑ ASCII value for small a is 97, b is 98,....., z is 122.



# Printing a character on screen

49

- For printing characters, '%c' conversion specifier is used in 'printf'

## Example:

```
char Alphabet = 'N';
```

```
printf ("%c", Alphabet);
```

The above code is same as:

```
char Alphabet = 78;
```

```
printf ("%c", Alphabet);
```

# Control Structures

50

## Selectional Control Structures

- There are two selectional control structures
  - If statement
  - Switch statement

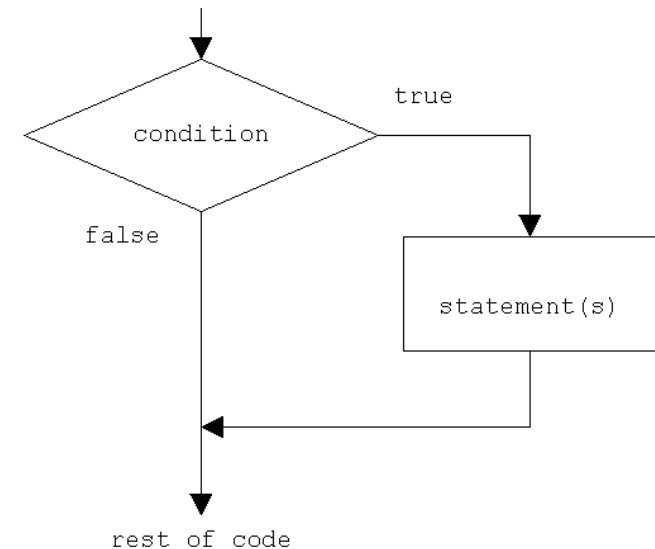
## Simple if Statement

- In a simple 'if' statement, a condition is tested
- If the condition is true, a set of statements are executed
- If the condition is false, the statements are not executed and the program
- control goes to the next statement that immediately follows if block

Example

```
if (Duration >= 3)
```

```
{ RateOfInterest = 6.0; }
```



# else Statement

51

## else Statement

- In simple 'if' statement, when the condition is true, a set of statements are executed. But when it is false, there is no alternate set of statements
- The statement 'else' provides the sar

### Syntax:

```
if (testExpression)
{
    // codes inside the body of if
}
else
{
    // codes inside the body of else
}
```

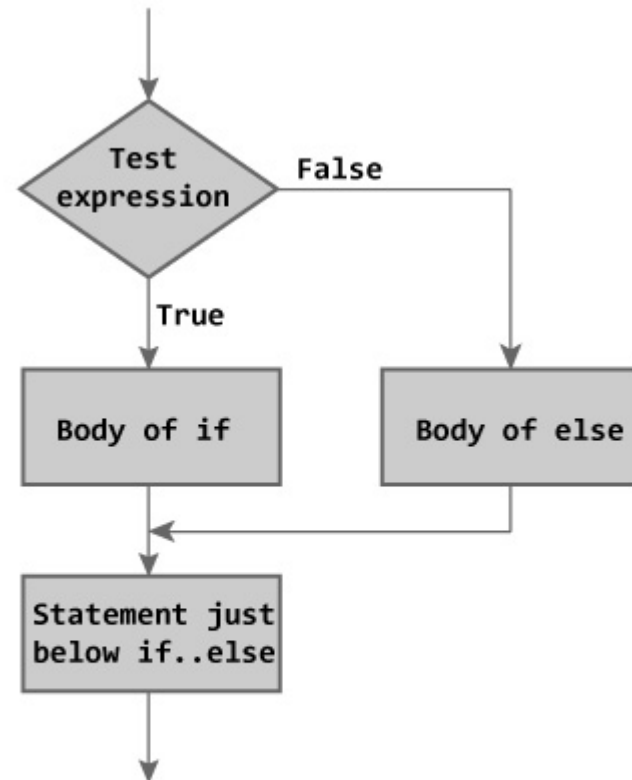


Figure: Flowchart of if...else Statement

# else Statement (Cont'd)

52

## else Statement

### Example:

```
if (Duration >= 3) {  
    RateOfInterest = 6.0;  
}  
else {  
    RateOfInterest = 5.5;  
}
```

## else if Statement

- The 'else if' statement is to check for a sequence of conditions
- When one condition is false, it checks for the next condition and so on
- When all the conditions are false the 'else' block is executed
- The statements in that conditional block are executed and the other 'if' statements are skipped

# Nested if Statement

53

## Syntax:

```
if (condition-1)
{
    Statement 1;
    if (condition-2)
    {
        Statement 2;
    }
    else
    {
        Statement n;
    }
}
else {
    Statement
}
x;
Next Statement:
```

## Nested if Statement

- An 'if' statement embedded within another 'if' statement is called as nested 'if'

- Example:

```
if (iDuration > 6 )
{
    if (dPrincipalAmount > 25000)
    {
        printf("Your percentage of incentive is 4%");
    }
    else
    {
        printf("Your percentage of incentive is 2%");
    }
}
else {
    printf("No incentive");
}
```

# Example

54

*/\* Program to check whether an integer entered by the user is odd or even \*/*

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int number;
```

```
    printf("Enter an integer: ");
```

```
    scanf("%d",&number);
```

*/\* True if remainder is 0 \*/*

```
    if( number%2 == 0 )
```

```
        printf("%d is an even integer.",number);
```

```
    else
```

```
        printf("%d is an odd integer.",number);
```

```
    return 0;
```

```
}
```

# switch case Statement

55

- The 'switch' statement is a selectional control structure that selects a choice from the set of available choices.
- It is very similar to 'if' statement.
- But 'switch' statement cannot replace 'if' statement in all situations.

Syntax:

```
switch (n)
```

```
{
```

```
case 1: // code to be executed if n = 1;
```

```
    break;
```

```
case 2: // code to be executed if n = 2;
```

```
    break;
```

```
default: // code to be executed if n doesn't match any cases
```

```
}
```

# switch case Example

56

*/\*Following is a simple program to demonstrate syntax of switch.*

```
#include <stdio.h>

void main()
{
    int x = 2;
    switch (x)
    {
        case 1: printf("Choice is 1");
                break;
        case 2: printf("Choice is 2");
                break;
        case 3: printf("Choice is 3");
                break;
        default: printf("Choice other than 1, 2 and 3");
                break;
    }
}
```

Output: Choice is 2



# Iterational Control Structures

57

- Iterational (repetitive) control structures are used to repeat certain statements for a specified number of times
- The statements are executed as long as the condition is true
- These kind of control structures are also called as **loop control structures**
- Three kinds of loop control structures are:
  - while
  - do while
  - for

# while Loop Control Structure

58

- A 'while' loop is used to repeat certain statements as long as the condition is true
- When the condition becomes false, the 'while' loop is quitted
- This loop control structure is called as an **entry-controlled** loop because, only when the condition is true, are the statements executed

## Syntax:

```
while (condition)
{
    Set of statements;
}
Next Statement;
```

## Example:

```
unsigned int Count = 1;
while (Count <= 3) {
    printf("%d\n",Count);
}
```

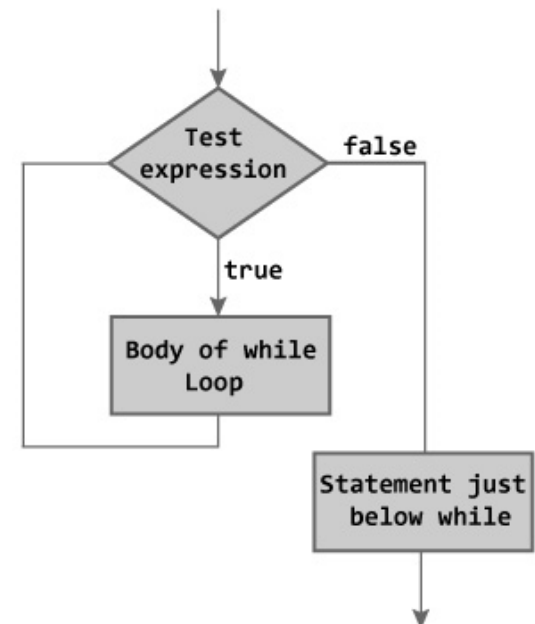


Figure: Flowchart of while Loop

# What is the output of the following code?

## Example 1

```
unsigned int Count=3;
while (Count<=5)
{
    printf(“%u\n”,Count);
    Count++;
}
```

## Example 2

```
int Count = 1;
while(Count<=10)
{
    printf(“\n MITCOE”);
    Count++;
}
```

# do while Loop Control Structure

60

- The 'do while' loop is very similar to 'while' loop. In 'do while' loop, the condition is tested at the end of the loop.
- Because of this, even when the condition is false, the body of the loop is executed at least once.
- This is an **exit-controlled** loop.

## Syntax:

do

{

Set of statement(s);

} while (condition);

Next Statement;

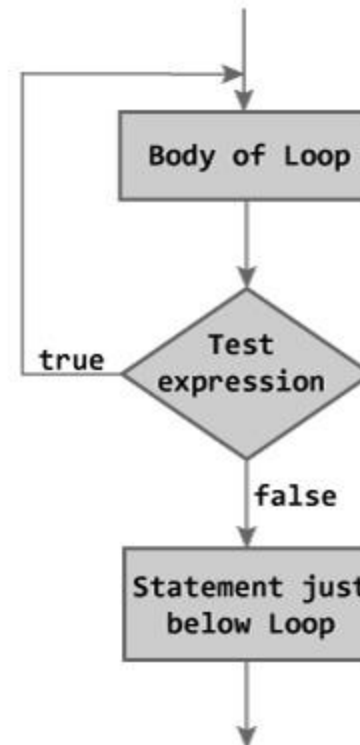


Figure: Flowchart of do...while Loop

# do while Loop Control Structure Example

```
int Number, Sum = 0;

do {

    printf("Enter a number. Type 0(zero) to end the
    input ");

    scanf("%d",&Number);

    Sum = Sum + Number;

} while (Number != 0);
```

# Difference between while and do while loops

62

While loop	Do-while loop
<b>Syntax:</b> <pre>while ( condition ) { statements; //body of loop }</pre>	<b>Syntax:</b> <pre>do{ . statements; // body of loop. . } while( Condition );</pre>
In 'while' loop the controlling condition appears at the start of the loop.	In 'do-while' loop the controlling condition appears at the end of the loop.
The iterations do not occur if, the condition at the first iteration, appears false.	The iteration occurs at least once even if the condition is false at the first iteration.

# for Loop Control Structure

63

- The 'for' loops are similar to the other loop control structures
- The 'for' loops are generally used when certain statements have to be executed a specific number of times
- Advantage of for loops:
  - All the three parts of a loop (initialization, condition , increment) can be given in a single statement
  - Because of this, there is no chance of user missing out initialization or increment steps which is the common programming error in 'while' and 'do while' loops

## Syntax:

for (Initialization; Termination-Condition; Increment-Step)

{

Set of statement(s);

}

Next Statement;

# for Loop Control Structure (cont'd)

64

## Example:

```
int Count;  
for (Count = 1; Count <= 5; Count++)  
{  
    printf("%d\n",Count);  
}
```

Output: 1

2

3

4

5

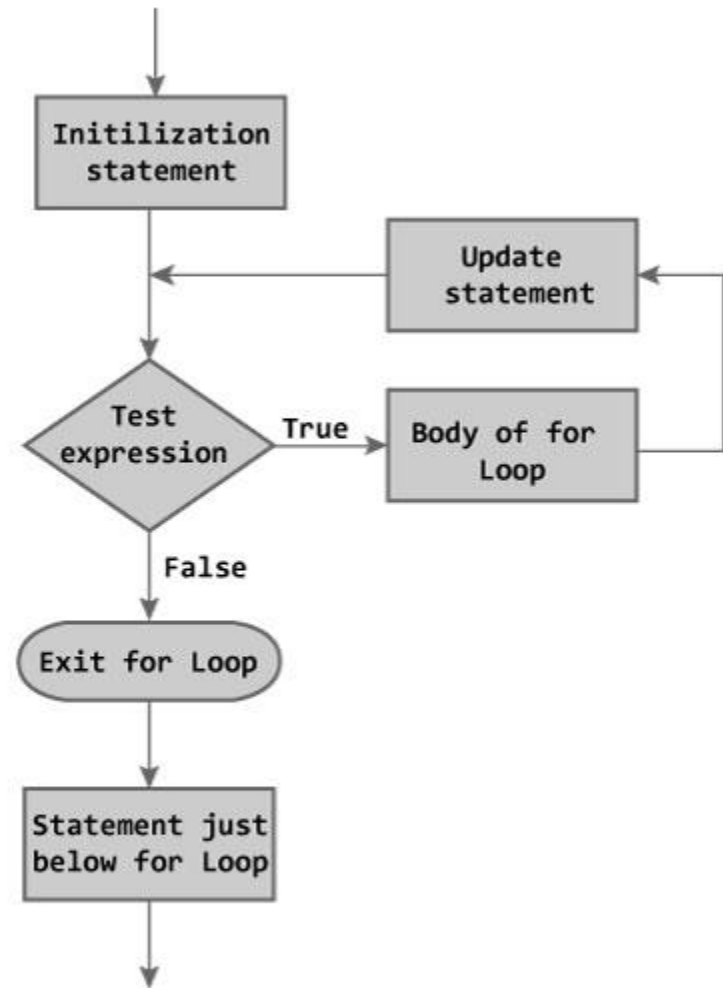


Figure: Flowchart of for Loop



# What is the output of the following code?

65

```
int Num;  
int Counter;  
int Product;  
for(Counter=1; Counter<= 3; Counter++)  
{  
    Product = Product * Counter;  
}  
printf("%d", Product);
```

The output is a junk value -- WHY???

# What is the output of the following code?

66

```
for (Count=0; Count<10; Count++) ;  
{  
    printf ("%d\n", Count);  
}
```

Have U observed this?

The output is 10

# for and while loops

67

## Given

```
for (Sum=0, Ctr=0; Ctr<10; Ctr=Ctr+1)
{
    scanf ("%d", &Num);
    Sum=Sum+Num;
}
printf ("%d", Sum);
```

## Rewrite it using **while** statement

```
Sum=0, Ctr=0;
while (Ctr<10)
{
    scanf ("%d", &Num);
    Sum=Sum+Num;
    Ctr=Ctr+1;
}
printf ("%d", Sum);
```

# Nested Loops

68

- A loop within another loop is called a nested loop.

## Example:

```
while (flag==1)
{
    for (Count=1;Count<=10;Count++)
    {
        statements;
    }
}
```

# Quitting the Loops – break Statement

The break statement is used to:

- Force the termination of a loop.
- When a break statement is encountered in a loop, the loop terminates immediately and the execution resumes the next statement following the loop.

## **Note:**

- Break statement can be used in an if statement only when the if statement is written in a loop
- Just an if statement with break leads to compilation error in C

# What is the output of the following code?

70

```
int Counter1=0;
int Counter2;
while(Counter1 < 3) {
    for (Counter2 = 0; Counter2 < 5; Counter2++) {
        printf("%d\t",Counter2);
        if (Counter2 == 2){
            break;
        }
    }
    printf("\n");
    Counter1 += 1;
}
```

0 1 2 is printed 3 times

# Continuing the Loops - continue Statement

71

- 'continue' statement forces the next iteration of the loop to take place and skips the code between continue statement and the end of the loop
- In case of for loop, continue makes the execution of the increment portion of the statement and then evaluates the conditional part.
- In case of while and do-while loops, continue makes the conditional statement to be executed.

## Example:

```
for(Count = 0 ; Count < 10; Count++) {  
    if (Count == 4) {  
        continue;  
    }  
    printf("%d", Count);  
}
```

The above code displays numbers from 1 to 9 except 4.

# Terminating the program using `exit()` function

- ❑ The function 'exit' is used to quit the program.
- ❑ Terminates the program and returns the status code to the operating system.
- ❑ This function is defined in 'stdlib.h' header file.

## **Syntax:**

`exit(int status);`

- ❑ The status code zero indicates that the program completed successfully.
- ❑ If there is a failure any other code has to be returned.



# Comparison of break, continue and exit

break	continue	exit()
Used to quit an innermost loop or switch	Used to continue the innermost loop	Used to terminate the program
Can be used only within loops or switch	Can be used only within the loops	Can be used anywhere in the program

# Conditional Expression Operator



(?:)

74

Syntax

`exp1 ? exp2 : exp3`

Arguments

`exp1`, `exp2`, and `exp3` Any expression.

goto

**Syntax**

`goto label;`

Arguments

`label` This is a name or tag associated with an executable statement.

# Conditional Expression Operator (cont'd)

return

## Syntax

return; /\* first form \*/

return exp; /\* second form \*/

## Arguments

exp Any valid C expression.

# Practice Programs

76

if - else statement

1. Create a program that prints a Student is passed(if marks $\geq$ 40) or failed.
2. Check the entered character whether it is vowel or consonant.

switch statement

1. Check the entered character whether it is vowel or consonant.
2. Write a program that calculates the area of circle, area of triangle, area of square using switch statement.

**while loop control structure**

1. Write a program that calculates the sum of the digits of a entered number and display the sum.
2. Write a program to print the output like:

```
*  
* *  
* * *  
* * * *  
* * * * *
```

# Source code of printing "\*" "

77

```
void main()
{
    int i,j;
    int space=4;
    /*run loop (parent
loop) till number of
rows*/
    for(i=0;i< 5;i++)
    {
        /*loop for
initially space, before
star printing*/
```

```
for(j=0;j< space;j++)
    {
        printf(" ");
    }

    for(j=0;j<=i;j++)
    {
        printf("* ");
    }

    printf("\n");
    space--; /*
decrement one space
after one row*/
    }
}
```

# Increment or Decrement Operator

What is the output of the following code:

1. `int j = 5, k = 5, l = 5, m = 5;`

`printf("j: %d\t k: %d\n", j++, k--);`

`printf("j: %d\t k: %d\n", j, k);`

`printf("l: %d\t m: %d\n", ++l, --m);`

`printf("l: %d\t m: %d\n", l, m);`

2. `int i=5;`

`printf("i++ = %d\n i--i=%d\n", i++, i--);`