RAMAIAH
Institute of Technology

M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Subject Name: Fundamental of Computing
Subject Code: CS16
Term: ODD SEMESTER 2020
UNIT-2

Faculty:
Darshana A Naik
Shilpa H
Sunitha R S

RAMAIAH
Institute of Technology

Unit 2

Operators and Expressions: Arithmetic Operators, Relational Operators, Logical Operators, Assignment Operators, Increment and Decrement Operators, Conditional Operators, Arithmetic Expressions, Evaluation of Expressions, Precedence of Arithmetic Operators, Type Conversions in Expressions, Operator Precedence and Associatively. **Control Structures in C**: Algorithm Development, **Decision Making and Branching:** Simple IF statement, IF..Else Statement, Nesting of IF...Else, The Else IF Ladder, The Switch Statements. The GOTO Statement. **Decision Making and Looping**: Introduction, The While Statement, The DO statement, The FOR statement, Jumps in Loops.

Operators and Expressions

- C supports a rich set of operators. Operators are used in programs to manipulate data and variables.
- They usually form a part of the mathematical of logical expressions.

Categories of Operators

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Increment and Decrement operators
- Conditional operators
- Bitwise operators
- Special operators

ARITHMETIC OPERATORS

- The operators are
- + (Addition)
  - - (Subtraction)
  - * (Multiplication)
  - **/ (Division)**
  - **% (Modulo division)**
- The Division operator produces the quotient.
- The modulo division produces the remainder of an integer division.
- The modulo division operator cannot be used on floating point data.
- **Note: C does not have any operator for *exponentiation***

Integer Arithmetic

- When both the operands in a single arithmetic expression are integers, the expression is called an *integer expression , and the operation is called integer arithmetic.*
- If a=14 and b=4, then
  - a-b=10
  - a+b=18
  - a*b=56
  - a/b=3
  - a%b=2

Integer Arithmetic

- What is the results of the following?

- 6/7=?
- 3/7=?
- 21/3=?

Integer Arithmetic

- During modulo division the sign of the result is always the sign of the first operand.
- -14 % 3 = -2
- -14 % -3 = -2
- 14 % -3 = 2

Real Arithmetic

- An arithmetic operation involving only real operands is called *real arithmetic*.
- *If* ***x and y*** are floats then we will have
  - 1) x = 6.0 / 7.0 = 0.857143
  - 2) y = 1.0 / 3.0 = 0.333333

- **The operator % cannot be used with real operands.**

Mixed-mode Arithmetic

- When one of the operands is real and the other is integer, the expression is called *a mixedmode arithmetic expression and its result is always a real number.*
- Eg: 1) 15 / 10.0 = 1.5

RAMAIAH Institute of Technology

C Program for the arithmetic operators

- What is the output of the following?

```c
#include <stdio.h>
int main()
{
      int a = 9,b = 4, c;

      c = a+b;
      printf("a+b = %d \n",c);

      c = a-b;
      printf("a-b = %d \n",c);

      c = a*b;
      printf("a*b = %d \n",c);

      c=a/b;
      printf("a/b = %d \n",c);

      c=a%b;
      printf("Remainder when a divided by b = %d \n",c);

      return 0;
}
```

RELATIONAL OPERATORS

- Comparisons can be done with the help of *relational operators*.
- *The expression* containing a relational operator is termed as a *relational expression*.
- *The value of a relational* expression is either *one or zero*.

RELATIONAL OPERATORS

- 1) < (is less than)
- 2) <= (is less than or equal to)
- 3) > (is greater than)
- 4) >= (is greater than or equal to)
- 5) = = (is equal to)
- 6) != (is not equal to)

Relational Operators example in C

```c
#include <stdio.h>
int main()
{
        int a = 5, b = 5, c = 10;

        printf("%d == %d = %d \n", a, b, a == b); // true
        printf("%d == %d = %d \n", a, c, a == c); // false

        printf("%d > %d = %d \n", a, b, a > b); //false
        printf("%d > %d = %d \n", a, c, a > c); //false


        printf("%d < %d = %d \n", a, b, a < b); //false
        printf("%d < %d = %d \n", a, c, a < c); //true


        printf("%d != %d = %d \n", a, b, a != b); //false
        printf("%d != %d = %d \n", a, c, a != c); //true


        printf("%d >= %d = %d \n", a, b, a >= b); //true
        printf("%d >= %d = %d \n", a, c, a >= c); //false


        printf("%d <= %d = %d \n", a, b, a <= b); //true
        printf("%d <= %d = %d \n", a, c, a <= c); //true

        return 0;

}
```

Relational Operators example in C

```c
#include <stdio.h>
int main()
{
	int a = 5, b = 5, c = 10;

	printf("%d == %d = %d \n", a, b, a == b); // true
	printf("%d == %d = %d \n", a, c, a == c); // false

	printf("%d > %d = %d \n", a, b, a > b); //false
	printf("%d > %d = %d \n", a, c, a > c); //false


	printf("%d < %d = %d \n", a, b, a < b); //false
	printf("%d < %d = %d \n", a, c, a < c); //true


	printf("%d != %d = %d \n", a, b, a != b); //false
	printf("%d != %d = %d \n", a, c, a != c); //true


	printf("%d >= %d = %d \n", a, b, a >= b); //true
	printf("%d >= %d = %d \n", a, c, a >= c); //false


	printf("%d <= %d = %d \n", a, b, a <= b); //true
	printf("%d <= %d = %d \n", a, c, a <= c); //true

	return 0;

}
```

```
5 == 5 = 1
5 == 10 = 0
5 > 5 = 0
5 > 10 = 0
5 < 5 = 0
5 < 10 = 1
5 != 5 = 0
5 != 10 = 1
5 >= 5 = 1
5 >= 10 = 0
5 <= 5 = 1
5 <= 10 = 1
```

LOGICAL OPERATORS

- Following are three *logical operators*.
  - **&& (logical AND)**
  - **|| (logical OR)**
  - **! (logical NOT)**
- Eg:
- 1) if(age>55 && sal<1000)
- 2) if(number<0 || number>100)

Truth Table

op-1	op-2	Value of the expression	
		op-1 && op-2	op-1 || op-2
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

LOGICAL OPERATORS

- If c = 5 and d = 2 then, expression
  - ((c == 5) && (d > 5)) equals to 0.
- If c = 5 and d = 2 then, expression
  - ((c == 5) || (d > 5)) equals to 1.
- If c = 5 then, expression
  - ! (c == 5) equals to 0.

RAMAIAH Institute of Technology

LOGICAL OPERATORS- Example

```c
#include <stdio.h>
int main()
{
      int a = 5, b = 5, c = 10, result;

      result = (a == b) && (c > b);
      printf("(a == b) && (c > b) equals to %d \n", result);

      result = (a == b) && (c < b);
      printf("(a == b) && (c < b) equals to %d \n", result);

      result = (a == b) || (c < b);
      printf("(a == b) || (c < b) equals to %d \n", result);

      result = (a != b) || (c < b);
      printf("(a != b) || (c < b) equals to %d \n", result);

      result = !(a != b);
      printf("!(a == b) equals to %d \n", result);

      result = !(a == b);
      printf("!(a == b) equals to %d \n", result);

      return 0;
}
```

RAMAIAH Institute of Technology

LOGICAL OPERATORS- Example- Output

```c
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) equals to %d \n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) equals to %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) equals to %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) equals to %d \n", result);

    result = !(a != b);
    printf("!(a == b) equals to %d \n", result);

    result = !(a == b);
    printf("!(a == b) equals to %d \n", result);

    return 0;
}
```

Output

```
(a == b) && (c > b) equals to 1
(a == b) && (c < b) equals to 0
(a == b) || (c < b) equals to 1
(a != b) || (c < b) equals to 0
!(a != b) equals to 1
!(a == b) equals to 0
```

ASSIGNMENT OPERATORS

- The usual assignment operator is '='.
- In addition, C has a set of 'shorthand' assignment operators.
- Eg: x += y+1;
- This is same as the statement x=x+(y+1);

ASSIGNMENT OPERATORS

- a + =1 → a = a + 1
- a - = 1 →a = a – 1
- a *= n + 1 → a = a * (n+1)
- a /= n + 1 → a = a / (n+1)
- a %= b → a = a % b

RAMAIAH Institute of Technology

INCREMENT AND DECREMENT OPERATORS

- These are the *increment and decrement operator:*
- **++ and --**
- The operator ++ adds 1 to the operands while -- subtracts 1.
- It takes the following form:
- ++m; or m++
  - --m; or m—
- x= 4++; // gives error, because 4 is constant

INCREMENT AND DECREMENT OPERATORS- Example

```
void main()
{
int x,i;  i=10;
x=i++;
printf("x: %d",x);
printf("i: %d",i);
}
```

INCREMENT AND DECREMENT OPERATORS- Example

```
void main()
{
int x,i;  i=10;


x=i++;
printf("x: %d",x);
printf("i: %d",i);
}
```

Output:

10

11

INCREMENT AND DECREMENT OPERATORS- Example

```
void main()
{
int x,i;  i=10;
x=++i;
printf("x: %d",x);
printf("i: %d",i);
}
```

INCREMENT AND DECREMENT OPERATORS- Example

```
void main()
{
int x,i;  i=10;


x=++i;
printf("x: %d",x);
printf("i: %d",i);
}
```

Output:

11

11

INCREMENT AND DECREMENT OPERATORS- Example

```
void main()
{
int x,i;  i=10;
x=i--;
printf("x: %d",x);
printf("i: %d",i);
}
```

RAMAIAH Institute of Technology

INCREMENT AND DECREMENT OPERATORS- Example

```
void main()
{
int x,i;  i=10;


x=i--;
printf("x: %d",x);
printf("i: %d",i);
}
```

Output:

10

9

RAMAIAH Institute of Technology

INCREMENT AND DECREMENT OPERATORS- Example

```
void main()
{
int x,i;  i=10;
x=--i;
printf("x: %d",x);
printf("i: %d",i);
}
```

INCREMENT AND DECREMENT OPERATORS- Example

```
void main()
{
int x,i;  i=10;


x=--i;
printf("x: %d",x);
printf("i: %d",i);
}
```

Output:

9

9

INCREMENT AND DECREMENT OPERATORS- Example

```
void main()
{
int x,a,b,c;  a = 2;
b = 4;
c = 5;
x = a-- + b++ -++c;
printf("x: %d",x);
}
```

RAMAIAH Institute of Technology

INCREMENT AND DECREMENT OPERATORS- Example

```
void main()
{
int x,a,b,c;  a = 2;
b = 4;
c = 5;
x = a-- + b++ -++c;
printf("x: %d",x);
}
```

Output: 0

RAMAIAH Institute of Technology

INCREMENT AND DECREMENT OPERATORS- Example

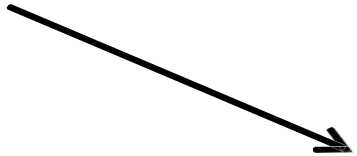
```
void main()
{
int x,a,b,c;  a = 2;
b = 4;
c = 5;
x = a-- + b++ -++c;
printf("x: %d",x);
}
```

Output: 0

Increment and decrement operators

```
void main()
{
int a=5;
printf("a: %d",++a);
printf("a: %d", a++);
}
```

a=a+1=5+1=6
a=a=6

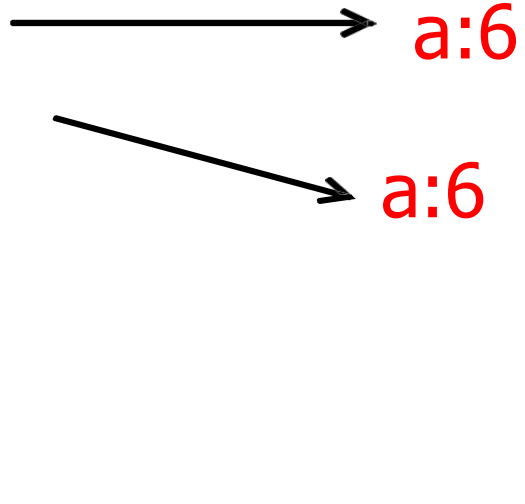


a=a=6 //Displayed
a=a+1=7

RAMAIAH Institute of Technology

Increment and decrement operators

```
void main()
{
int a=5;
printf("a: %d",++a);
printf("a: %d", a++);
printf("a: %d", a);
}
```

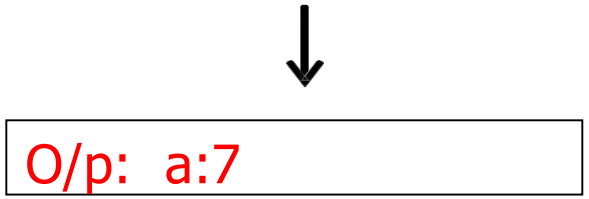


Exercise-What is the output?

```
#include <stdio.h>  int main()
{
int a=5;
printf("a: %d\n",++a, a++);  return 0;
}
```

Exercise-What is the output?

```
#include <stdio.h>  int main()
{
int a=5;
printf("a: %d\n",++a, a++);  return 0;
}
```



Exercise-What is the output?

```
#include <stdio.h>
int main()
{
int a=5;
printf("a: %d\n",a--);
printf("a: %d\n",--a);
return 0;
}
```

Exercise-What is the output?

```
#include <stdio.h>
int main()
{
int a=5;
    printf("a: %d\n",a--);
    printf("a: %d\n",--a);
    return 0;
}
```

```
a:5
a:3
```

Exercise-What is the output?

```
#include <stdio.h>
int main()
{
int a=5,x;
x=a--;
printf("a: %d\n",x);
printf("a: %d\n",a);
return 0;
}
```

Exercise-What is the output?

```
#include <stdio.h>
 int main()
{
int a=5,x;
x=a--;
printf("a: %d\n",x);
    printf("a: %d\n",a);
    return 0;
}
```

```
a:5
a:4
```

Exercise-What is the output?

```
#include <stdio.h>  int
main()
{
int a=5,x;
x=a--;
printf("a: %d\n",a);
printf("a: %d\n",x);
x=--a;
printf("a: %d\n",a);
printf("a: %d\n",x);
return 0;
}
```

Exercise-What is the output?

```
#include <stdio.h>
int main()
{
int a=5,x;
x=a--;
printf("a: %d\n",a);
printf("a: %d\n",x);
x=--a;
printf("a: %d\n",a);
printf("a: %d\n",x);
return 0;
}
```

```
a:4
a:5
a:3
a:3
```

Exercise-What is the output?

```
#include <stdio.h>
int main()
{
int a=5;
printf("%d",a--,a++,--a);
return 0;
}
```

Exercise-What is the output?

```
#include <stdio.h>
int main()
{
int a=5;
printf("%d",a--,a++,--a);  return
0;
}
```

a:5

Exercise-What is the output?

```
#include <stdio.h>
 int main()
{
int a=5;
printf("%d%d",a--,a++,--a);
return 0;
}
```

Exercise-What is the output?

```
#include <stdio.h>
int main()
{
int a=5;
printf("%d%d",a--,a++,--a);
   return 0;
}
```

a:5, 4

RAMAIAH Institute of Technology

Exercise-What are the values of n1, n2, n3 and n4 at the end?

```
int n1,n2,n3,n4;
n1 = 1;
n2 = ++n1;
n3 = ++n1;
n4 = n1++;
```

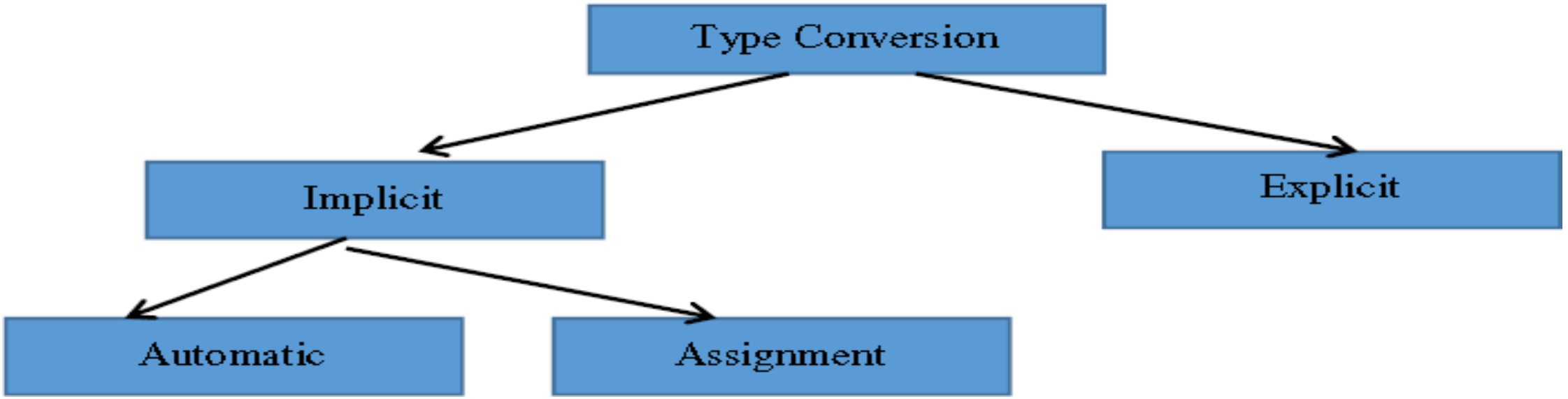
RAMAIAH Institute of Technology

Exercise-What are the values of n1, n2, n3 and n4 at the end?

```
int n1,n2,n3,n4;
n1 = 1;
n2 = ++n1;
n3 = ++n1;
n4 = n1++;
```

n1: 4	n2: 2	n3:3	n4: 3

TYPE CONVERSION IN EXPRESSIONS



Automatic Type Casting

1. char and short operands are converted to int
2. Lower data types are converted to the higher data types and result is of higher type.
3. The conversions between unsigned and signed types may not yield intuitive results.
4. Example

```
float f; double d; long l;
int i; short s;
```

`d + f` f will be converted to `double`
`i / s` s will be converted to `int`
`l / i` i is converted to `long`; `long` result

Hierarchy

Double float

long Int

Short and char

Assignment type casting

- If types of the operand in an assignment expression is different, then the operand on the right-hand side will be converted to the type of left-hand operand according to the following rules.
  - int i = 'z';
  - i=122;

Explicit Type Casting

- The general form of a type casting operator is
- (type-name) expression
- It is generally a good practice to use explicit casts than to rely on automatic type conversions.
- Example

```
C = (float)9 / 5 * ( f - 32 )
```

- `float` to `int` conversion causes truncation of fractional part
- `double` to `float` conversion causes rounding of digits
- `long int` to `int` causes dropping of the higher order bits.

Example on implicit type casting

```
#include<stdio.h>  main ( )
{
int i=17;
char c='c';  int sum;  sum=i+c;
printf("Value of sum:%d\n",sum);
}
```

Output: Value of sum: 116

Example on explicit type casting

With explicit

```
int sum=17,count=5;  float mean;
mean=(float)sum/count;  printf("Value of
 mean:%f\n",mean);
```

Output:

Value of mean: 3.400000

Without explicit

```
int sum=17,count=5;  float mean;
mean=sum/count;  printf("Value of
 mean:%f\n",mean);
```

Output:

Value of mean: 3.000000

Operator Precedence and Associativity

Operator	Description	Associativity
() [] . ->> ++ --	Parentheses or function call Brackets or array subscript Dot or Member selection operator Arrow operator Postfix increment/decrement	left to right
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus and minus not operator and bitwise complement type cast Indirection or dereference operator Address of operator Determine size in bytes	right to left
* / %	Multiplication, division and modulus	left to right
+ -	Addition and subtraction	left to right
<< >>	Bitwise left shift and right shift	left to right
< <= > >=	relational less than/less than equal to relational greater than/greater than or equal to	left to right
== !=	Relational equal to and not equal to	left to right
&	Bitwise AND	left to right
^	Bitwise exclusive OR	left to right
|	Bitwise inclusive OR	left to right
&&	Logical AND	left to right
||	Logical OR	left to right
? :	Ternary operator	right to left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment operator Addition/subtraction assignment Multiplication/division assignment Modulus and bitwise assignment Bitwise exclusive/inclusive OR assignment	right to left
,	Comma operator	left to right

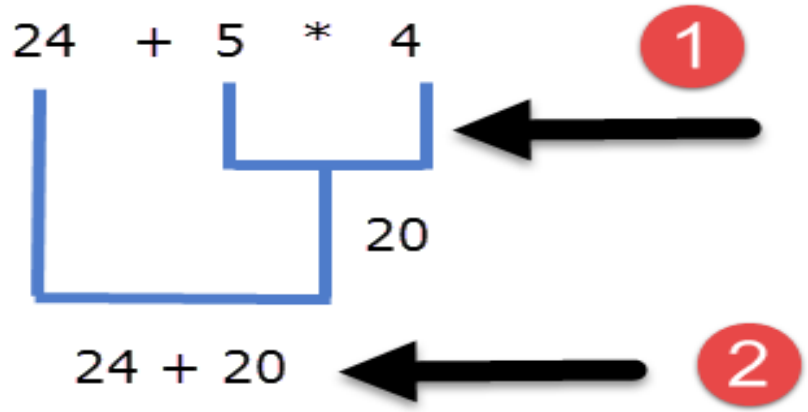
Operator precedence: It dictates the order of evaluation of operators in an expression.
Associativity: It defines the order in which operators of the same precedence are evaluated in an expression.
Associativity can be either from **left to right or right to left**.
Consider the following example:
24+5*4

Here we have two operators + and *, Which operation do you think will be evaluated first, addition or multiplication? If the addition is applied first then answer will be 116 and if the multiplication is applied first answer will be 44. To answer such question we need to consult the operator precedence table.

Examples:

Operators in the top have higher precedence and it decreases as we move towards the bottom.

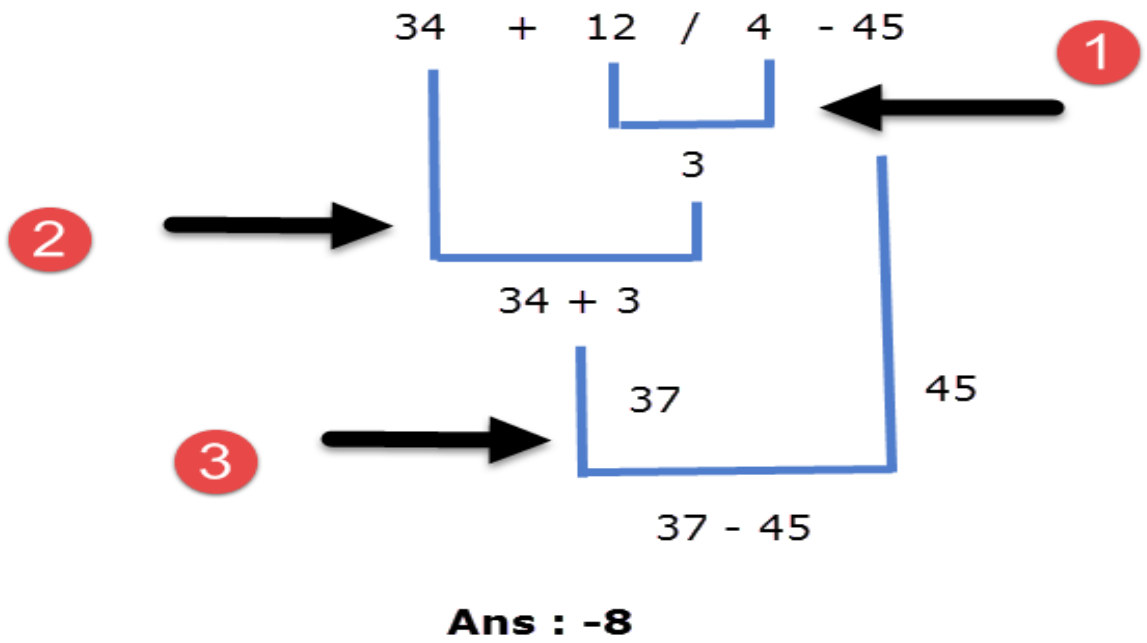
From the precedence table, we can conclude that the `*` operator is above the `+` operator, so the `*` operator has higher precedence than the `+` operator, therefore in the expression `24 + 5 * 4`, subexpression `5 * 4` will be evaluated first.



Ans : 44

Examples:

```
34 + 12/4 - 45
```



Ans : -8

Algorithm Development

Algorithm Development

the C programs that we developed were very simple. The steps were sequential and typically involved reading information from the keyboard, computing new information, and then printing the new information. In solving engineering problems, most of the solutions require more complicated steps, and thus we need to expand the algorithm development part of our problem-solving process.

Top-Down Design

Top-down design

Top-down design presents a "big picture" description of the problem solution in sequential steps. This overall description of the problem is then refined until the steps are detailed enough to translate to language statements.

Decomposition Outline

1. Read the new time value.
2. Compute the corresponding velocity and acceleration values.
3. Print the new velocity and acceleration.

Divide-and-conquer

Stepwise refinement

However, for most problem solutions, we need to refine the decomposition outline into a description with more detail. This process is often referred to as a **divide-and-conquer** strategy, because we keep breaking the problem solution into smaller and smaller portions. To describe this **stepwise refinement**, we use pseudocode or flowcharts.

RAMAIAH Institute of Technology

Structured Programming

Structured Programming

Structured program

A structured program is written using simple control structures to organize the solution to a problem. A simple structure is usually defined to be a sequence, a selection, or a repetition.

Sequence
Selection
Repetition

A sequence structure contains steps that are performed one after another; a **selection** structure contains one set of steps that is performed if a condition is true, and another set of steps that is performed if the condition is false; and a **repetition** structure contains a set of steps that is repeated as long as a condition is true. We now discuss each of these simple structures, and use pseudocode and flowcharts to give specific examples.

Sequence. A sequence contains steps that are performed one after another. All the programs developed in Chapter 2 have a sequence structure. For example, the pseudocode for the program that performed the linear interpolation is as follows:

Refinement in Pseudocode

main: *read a, f_a*
read c, f_c
read b
set f_b to $f\_a + \frac{b-a}{c-a} \cdot (f\_c - f\_a)$
print f_b

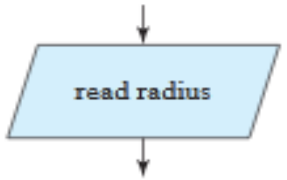
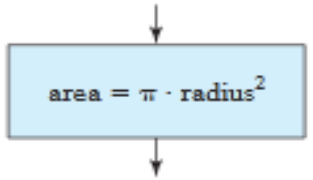
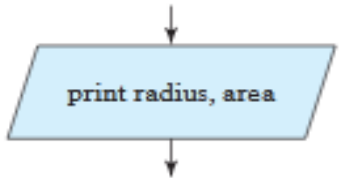
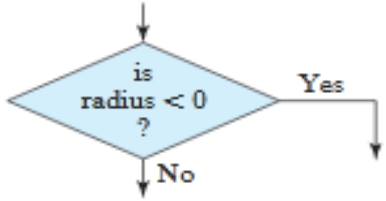
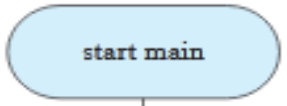
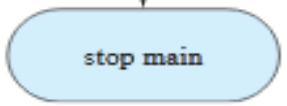
Sequence

Sequence. A sequence contains steps that are performed one after another. All the programs developed have a sequence structure. For example, the pseudocode for the program that performed the linear interpolation is as follows:

Refinement in Pseudocode

main: *read a, f_a*
read c, f_c
read b
set f_b to $f\_a + \frac{b-a}{c-a} \cdot (f\_c - f\_a)$
print f_b

RAMAIAH Institute of Technology

Basic Operation	Pseudocode Notation	Flowchart Symbol
Input	read radius	
Computation	set area to $\pi \cdot radius^2$	
Output	print radius, area	
Comparisons	if radius < 0 then ...	
Beginning of algorithm	main:	
End of algorithm		

Pseudocode notation and flowchart symbols.

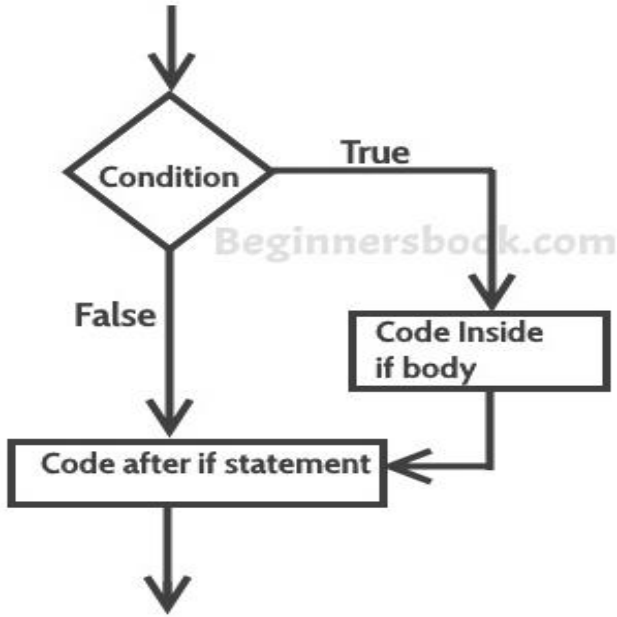
Decision Making and Branching

- Simple if statement
- IF ELSE Statement
- Nesting of IF-else statement
- The else-if statement

Simple if Statement

The statements inside the body of “if” only execute if the given condition returns true. If the condition returns false then the statements inside “if” are skipped.

Flow Diagram of if statement



Syntax of Simple if statement

The general form of a simple **if** statement is

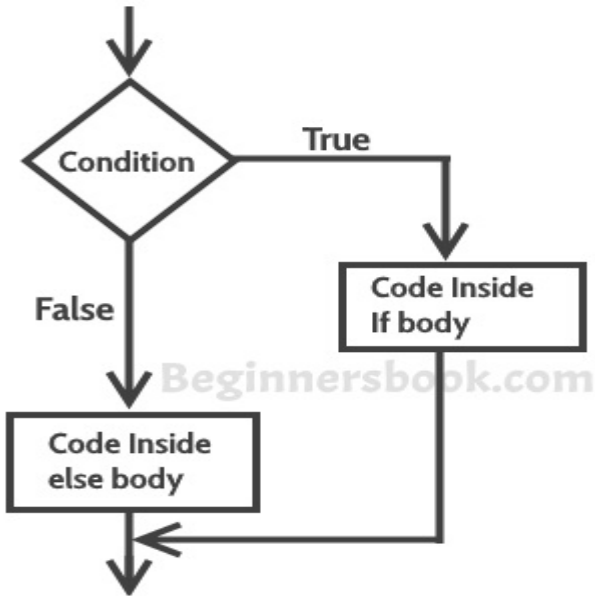
```
if (test expression)
{
    statement-block;
}
statement-x;
```

The 'statement-block' may be a single statement or a group of statements. If the *test expression* is true, the *statement-block* will be executed; otherwise the statement-block will be skipped and the execution will jump to the *statement-x*.

If else Statement

If condition returns true then the statements inside the body of “if” are executed and the statements inside body of “else” are skipped.
If condition returns false then the statements inside the body of “if” are skipped and the statements in “else” are executed.

Flow diagram of if else statement



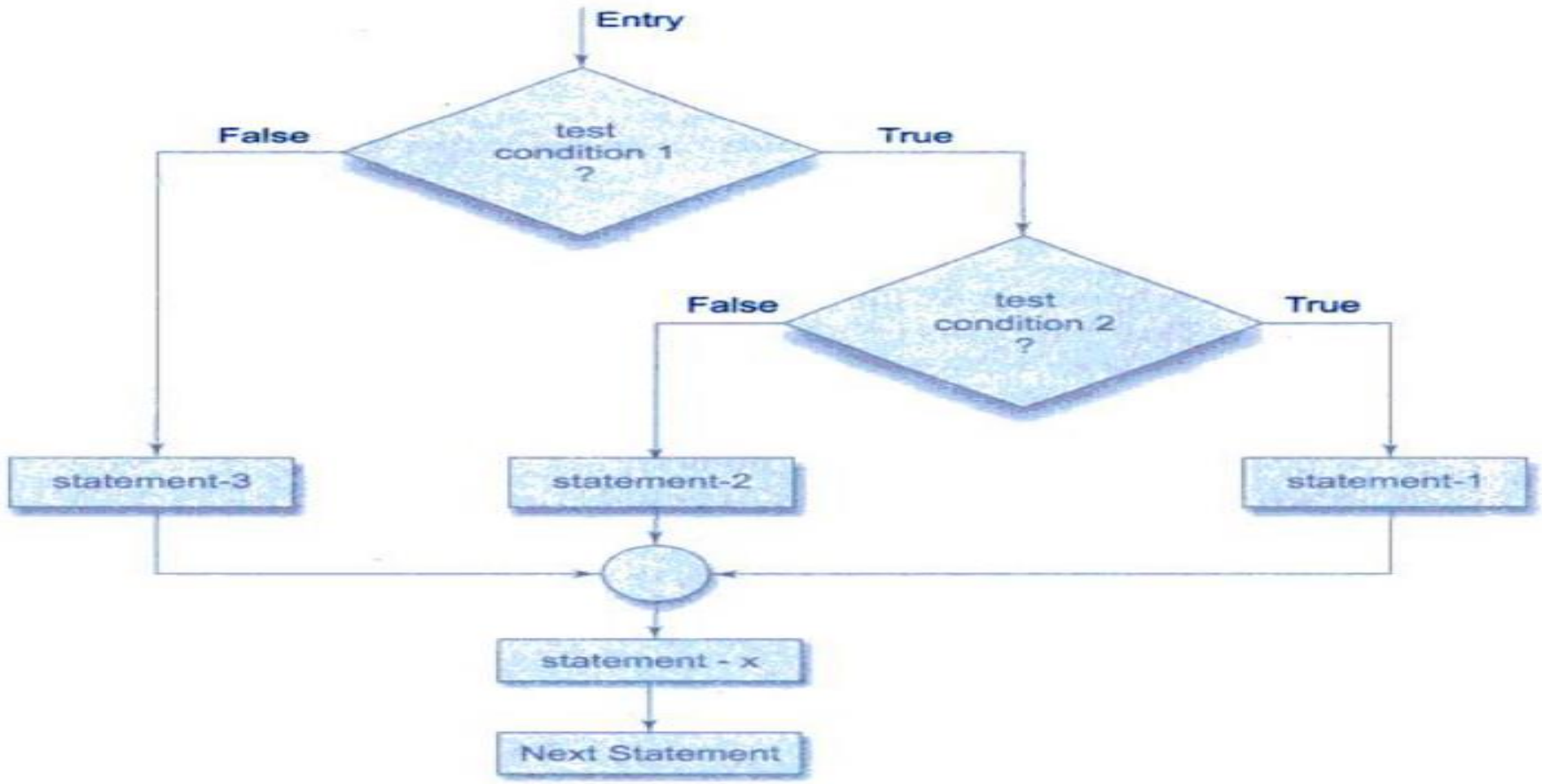
Syntax of if else statement

The if...else statement is an extension of the simple if statement. The general form is

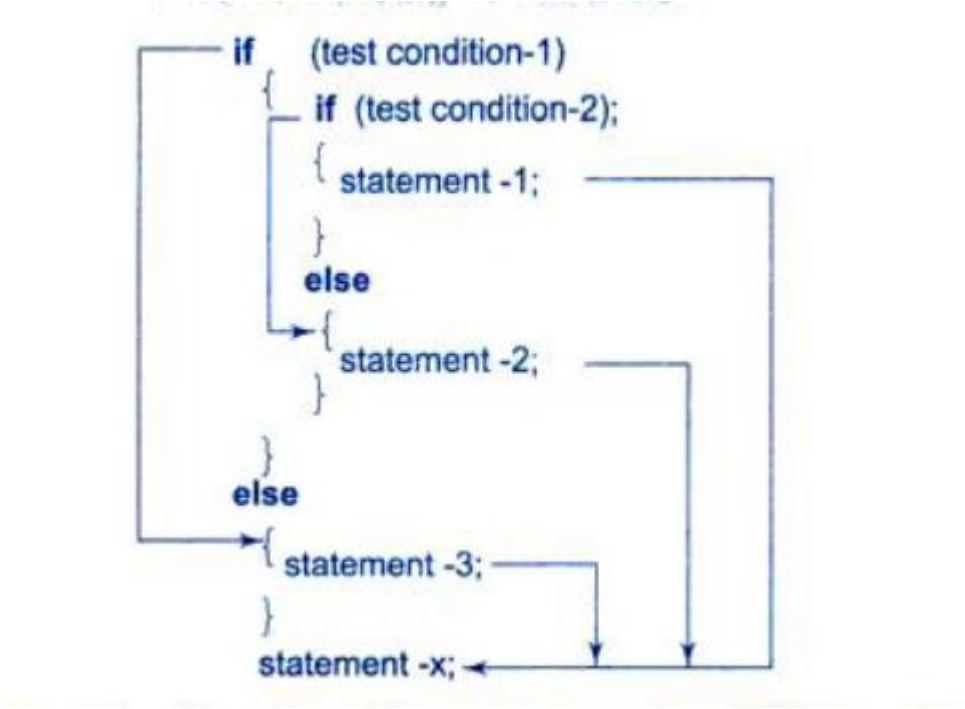
```
If (test expression)
     {
       True-block statement(s)
     }
else
     {
       False-block statement(s)
     }
statement-x
```

If the *test expression* is true, then the *true-block statement(s),* immediately following the if statements are executed; otherwise, the *false-block statement(s)* are executed. In either case, either *true-block* or *false-block* will be executed, not both. This is illustrated in Fig. In both the cases, the control is transferred subsequently to the statement-x.

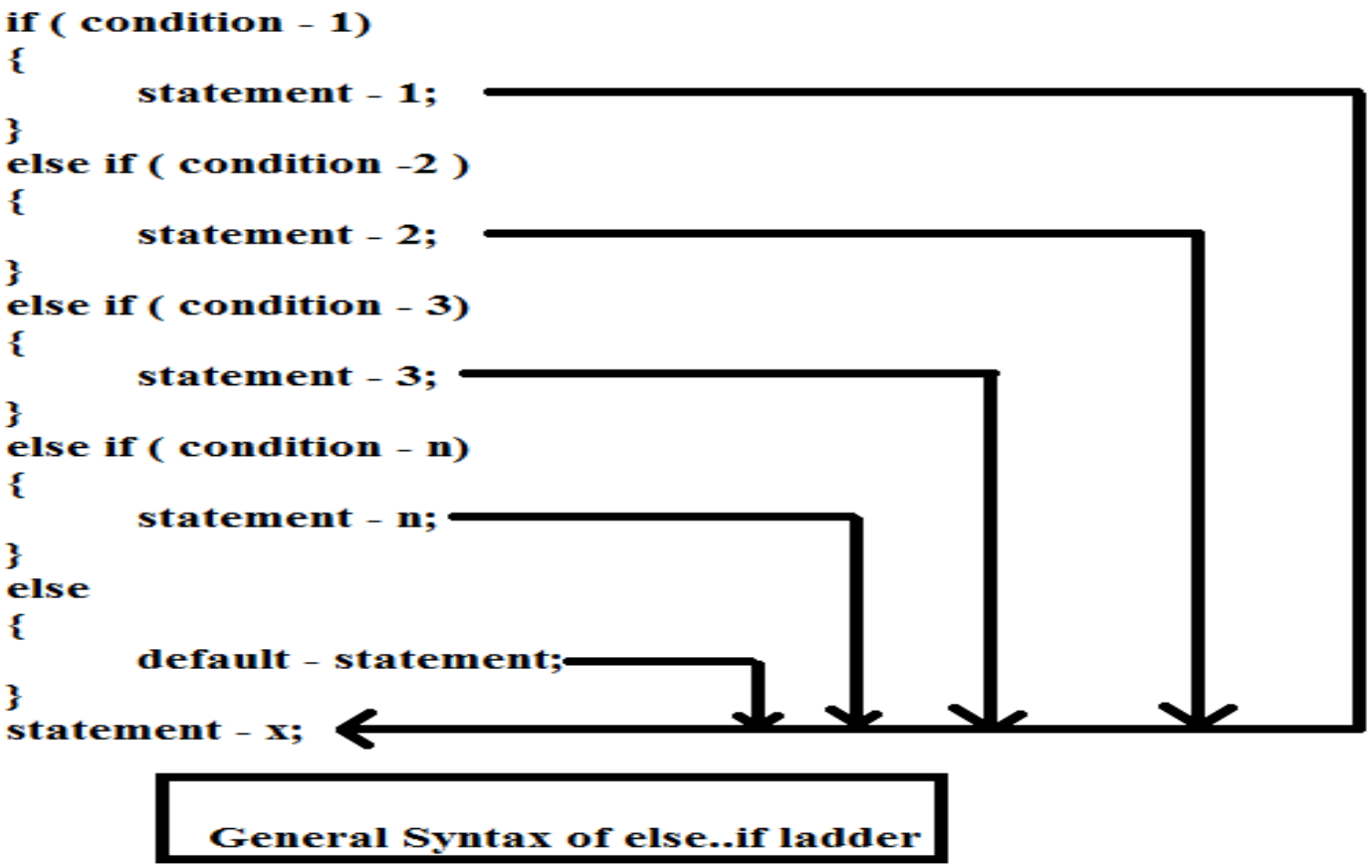
Nesting of if-else statement



Syntax of nesting of if-else statement

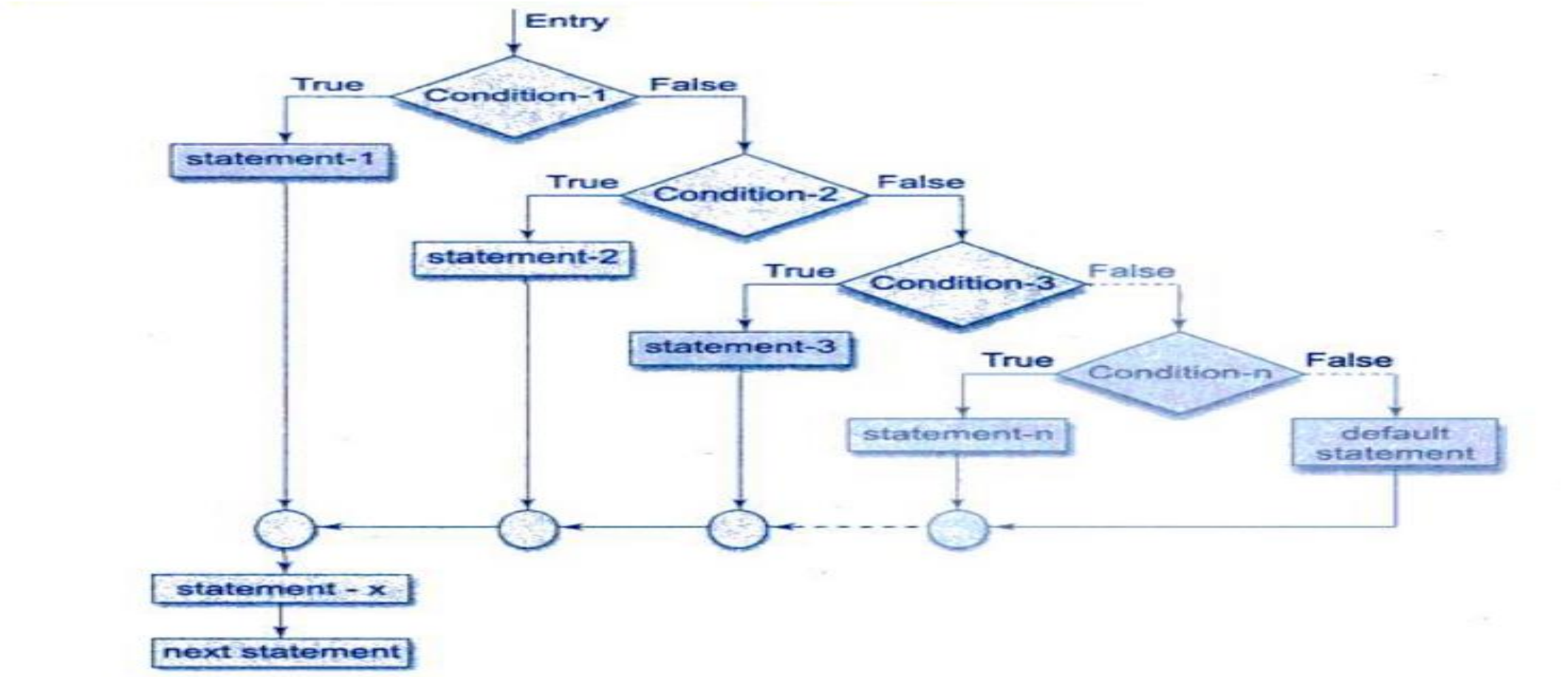


Syntax of else if ladder



General Syntax of else..if ladder

Nesting of else if ladder ladder



Switch statement in C

The **switch** Multiple-Selection Structure

```
switch ( integer expression )
{
    case constant1 :
        statement(s)
        break ;
    case constant2 :
        statement(s)
        break ;

        . . .
    default: :
        statement(s)
        break ;
}
```

RAMAIAH Institute of Technology

Case labels in switch

- **Which of the following are valid usages of case labels.**
  - case 1+1: **//valid**
  - case ‘A’:**//valid**
  - case 67:**//valid**
  - case var: **//valid**
  - case num1: **//valid**
  - case 10:**//valid**
  - case 20+20: **//valid**
  - case 10.12 **//invalid**
  - case 7.5**//invalid**

Programs

- Write and execute a C program to find whether a given alphabet is vowel or consonant using switch.

Programs

```c
#include<stdio.h>

int main() {
   char ch;
   printf(" Enter the character\n");
   scanf("%c",&ch);
   switch(ch)
   {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u': printf("Vowel\n");
                  break;
        default: printf("Consonant");
   }
}
```

RAMAIAH Institute of Technology

The goto Statement

- goto causes an unconditional jump to a labeled statement somewhere in the current function.
- Form of a labeled statement.

label: statement

Where label is an identifier.

RAMAIAH Institute of Technology

Goto Example

```
int main()
{
   char ch;
   int count=1;
    read:          printf(" Enter the character\n");
                    scanf("%c",&ch);
   count++;
   printf("Entered character is %c\n",ch);  if(count<=3)
   goto read;  printf("End");
   }
```

RAMAIAH Institute of Technology

The goto Statement

- Write and execute a C program to evaluate the square root for five numbers using goto statement.

RAMAIAH Institute of Technology

The goto Statement

```
#include<stdio.h>  #include<math.h>  int main() {
int num;
int count=1;
read: printf(" Enter the number\n");  scanf("%d",&num);
if(count<=5)
{
count++;
printf(" The square root of %d is %f\n",num,sqrt(num));
goto read;
}

printf("End");
}
```

RAMAIAH Institute of Technology

Loops – While, Do, For

- Repetition Statements
  - While
  - Do
  - For

RAMAIAH Institute of Technology

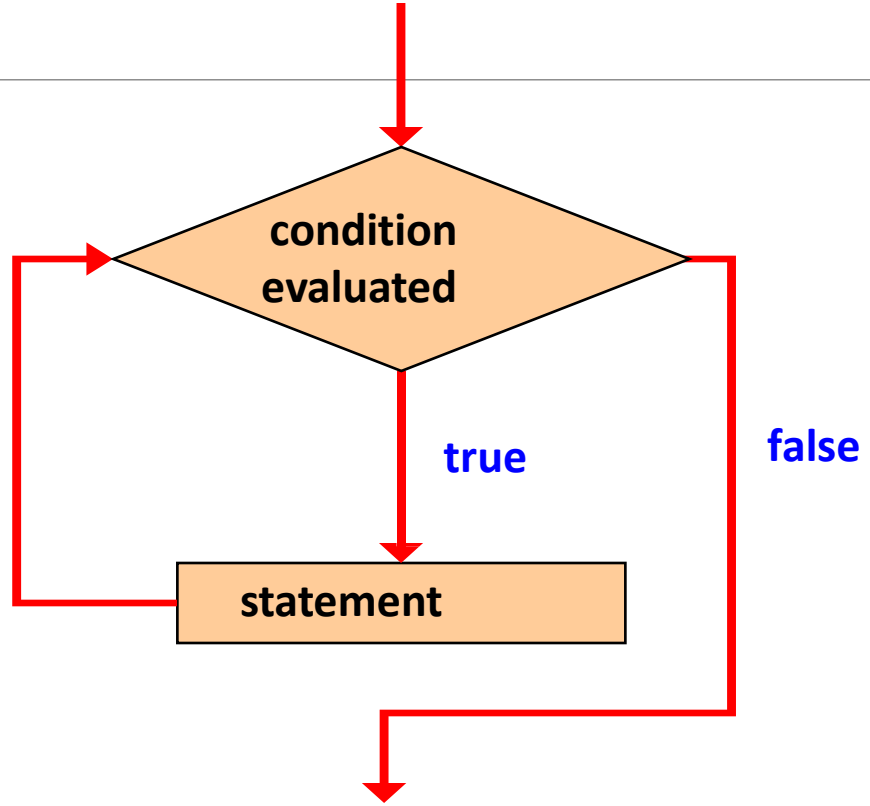
The while Statement

- A *while statement* has the following syntax:

```
while ( condition )
    statement;
```

- If the `condition` is true, the `statement` is executed
- Then the condition is evaluated again, and if it is still true, the statement is executed again
- The statement is executed repeatedly until the condition becomes false

Logic of a while Loop



The while Statement

- An example of a while statement:

```
int count = 0;  while (count < 2)
{
printf("Welcome to Java!");
count++;
}
```

- If the condition of a `while` loop is false initially, the statement is never executed
- Therefore, the body of a `while` loop will execute zero or more times

Trace while Loop

Initialize count

```
int count = 0;  while (count < 2)

{

printf("Welcome to C!");  count++;

}
```

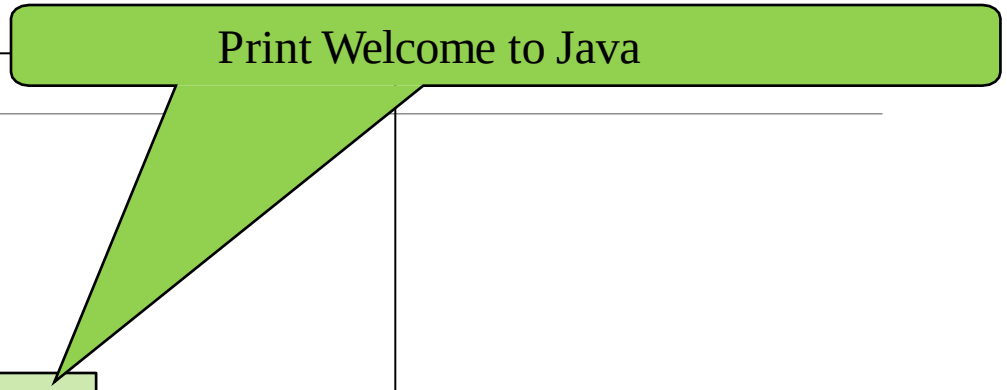
Trace while Loop, cont.

```
int count = 0;
while (count < 2)
{
printf("Welcome to C!");  count++;
}
```

(count < 2) is true

Trace while Loop, cont.

```
int count = 0;  while
(count < 2)
{
   printf("Welcome to C!");
    count++;
}
```



Trace while Loop, cont.

```
int count = 0;  while (count < 2)
{
printf("Welcome to C!");

    count++;
}
```

Increase count by 1
count is 1 now

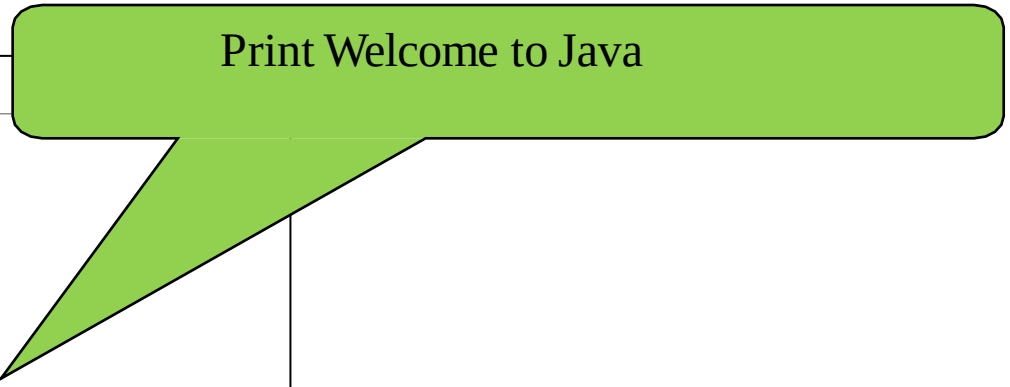
Trace while Loop, cont.

```
int count = 0;
while (count < 2)
{
printf("Welcome to C!");
count++;
}
```

(count < 2) is still true since count is 1

Trace while Loop, cont.

```
int count = 0;  while
(count < 2)
{
   printf("Welcome to C!");
    count++;
}
```



Trace while Loop, cont.

```
int count = 0;  while (count < 2)
{
printf("Welcome to C!");

   count++;
}
```

Increase count by 1
count is 2 now

RAMAIAH Institute of Technology

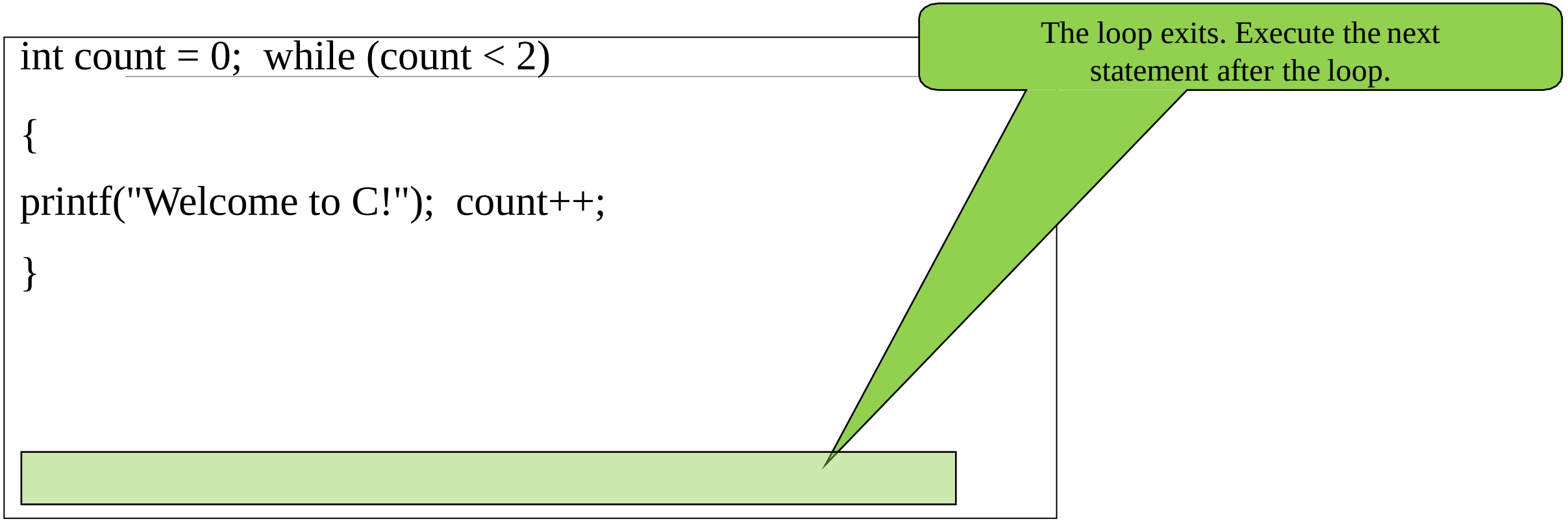
Trace while Loop, cont.

```
int count = 0;
while (count < 2)
{
printf("Welcome to C!");  count++;
}
```

(count < 2) is false since count is 2 now

Trace while Loop

```
int count = 0;  while (count < 2)
{
printf("Welcome to C!");  count++;
}
```



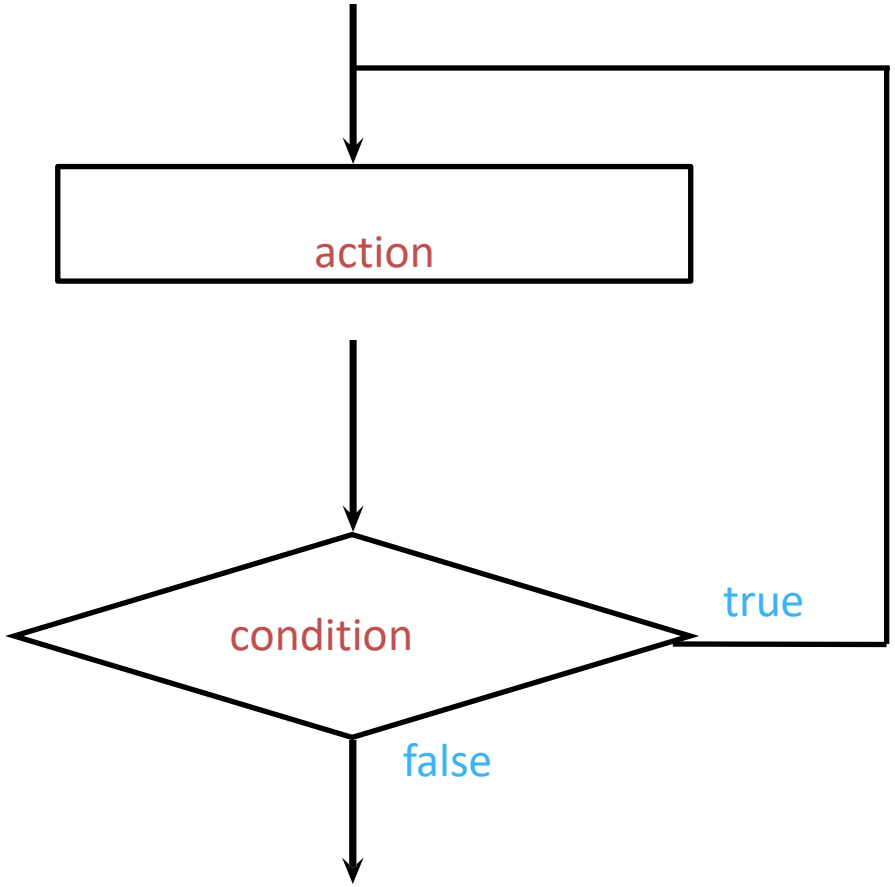
The `do-while` Statement

- Syntax

do action

while (**condition**)

- How it works:
  - execute action
  - if condition is true then execute action again
  - repeat this process until condition evaluates to false.
- action is either a single statement or a group of statements within braces.



Trace do while Loop

Initialize count

```
int count = 0;  do
{
printf("Welcome to C!");  count++;
} while (count < 2) ;
```

Trace do while Loop

```
int count = 0;  do
{
   printf("Welcome to C!");
   count++;
} while (count < 2) ;
```

Trace do while Loop

```
int count = 0;  do
{
printf("Welcome to C!");

   count++;
} while (count < 2) ;
```

count=1

Trace do while Loop

```
int count = 0;  do
{
printf("Welcome to C!");  count++;




} while (count < 2) ;
```

1<2

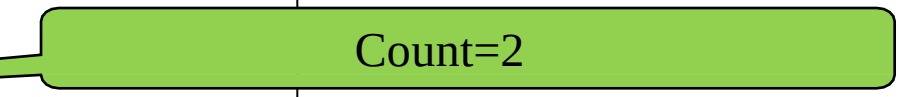
Trace do while Loop

```
int count = 0;  do
{
    printf("Welcome to C!");
    count++;
    } while (count < 2) ;
```

Trace do while Loop

```
int count = 0;  do
{
printf("Welcome to C!");

    count++;
} while (count < 2) ;
```



Trace do while Loop

```
int count = 0;  do
{
printf("Welcome to C!");  count++;



} while (count < 2) ;
```

2<2

Infinite Loops

- An example of an infinite loop:

```
int count = 1;
while (count <= 25)
{
printf(count);
count = count - 1;
}
```

- This loop will continue executing until the user externally interrupts the program.

Nested Loops

- How many times will the string "Here" be printed?

```
count1 = 1;
while (count1 <= 3)
{
   count2 = 1;
   while (count2 <= 3)
   {
      printf ("Here");
      count2++;
   }
   count1++;
}
```

What is the output of the following code?

```
#include<stdio.h>  void
main()
{
int val=1;  do{
val++;
++val;
}while(val++>25);

printf("%d\n",val);
}
```

What is the output of the following code?

```
#include<stdio.h>  void
main()
{
int val=1;  do{
val++;
++val;
}while(val++>25);

printf("%d\n",val);
}
```

Answer: 4

What is the output of the following code?

```
#include<stdio.h>  int
main()
{
int val=1;  while(val<=5)
printf("Hi");  val++;
}
```

What is the output of the following code?

```
#include<stdio.h>  int main()
{
int val=1;  while(val<=5)
printf("Hi");  val++;
}
```

HiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHiHi……..

What is the output of the following code?

```
#include<stdio.h>
int main()
{
int val=1;
while(val<=5)
{
printf("Hi");  val++;
}
}
```

Hi
Hi
Hi
Hi
Hi

What is the output of the following code?

```
#include<stdio.h>  int
main()
{
int val=1;
while(val<=5);
{
printf("Hi");  val++;
}
}
```

Programs

- **Write and execute a C program to read a number using keyboard, check whether a given number is palindrome or not using a while loop and display the result.**

Programs

```
#include <stdio.h>  int main()
{ int n, rev = 0, remainder, originalInteger; printf("Enter an integer: ");
scanf("%d", &n);
originalInteger = n; // reversed integer is stored in variable while( n!=0 )
{
remainder = n%10;
reversedInteger = reversedInteger*10 + remainder;  n /= 10;
}
// palindrome if orignalInteger and reversedInteger are equal
if (originalInteger == reversedInteger)
printf("%d is a palindrome.", originalInteger);  else
printf("%d is not a palindrome.", originalInteger);  return 0;
}
```

RAMAIAH
Institute of Technology

For Loop

- Syntax
- Example

Programs

- **Write and execute a C program to read numbers using keyboard, find the sum of odd numbers and even numbers in first n natural numbers using a for loop.**

Programs

- **Write and execute a C program to read a number using keyboard, find the factorial of a number using for loop and display the result.**

Programs

- **Write and execute a C program to print.**

1

12

123

1234

12345

Programs

- **Write and execute a C program to print.**

```
#include <stdio.h>

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

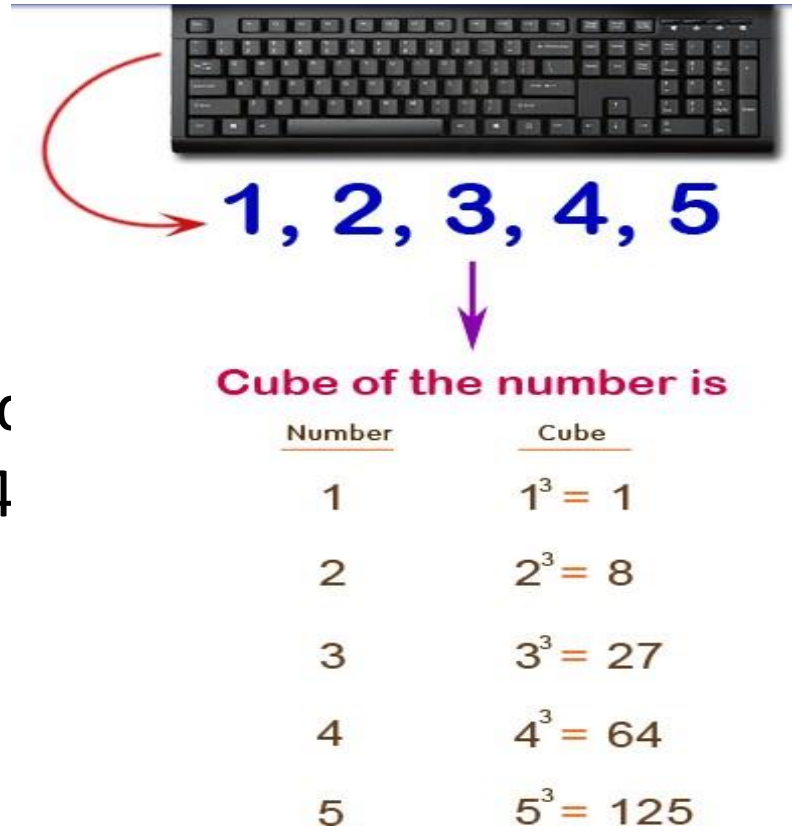
Programs

- Write a program in C to display the cube of the number upto given an integer.

Input number of terms : 5

Expected Output :

Number is : 1 and cube of the 1 is :1 Number is : 2 and cube of the 2 is :8 Number is : 3 and cube of the 3 is :27 Number is : 4 and cube of the 4 is :64 Number is : 5 and cube of the 5 is :125



Solution

```
#include    <stdio.h>
void   main()   {
int      i,ctr;
printf("Input     number   of terms    :   ");
scanf("%d",     &ctr);  for(i=1;i<=ctr;i++)   {
printf("Number is   :   %d   and   cube of  the
%d    is     :%d  \n",i,i,    (i*i*i));
}
}
```

Programs

- Write a program in C to display the multiplication table of a given integer.

```
#include    <stdio.h>
void        main()
{
int         j,n;
printf("Input the number (Table to be calculated) :    "); scanf("%d",&n);
printf("\n"); for(j=1;j<=10;j++)
{
printf("%d X    %d =    %d \n",n,j,n*j);
}
}
```

Programs

- **Write and execute a C program to read two numbers using keyboard, find GCD and LCM of two numbers using a do-while loop and display the result.**

Jumps in Loops

- The **continue statement** is used inside loops.
- When a continue statement is encountered inside a loop, *control jumps to the beginning of the loop for next iteration*,
  - skipping the execution of statements inside the body of loop for the current iteration.

RAMAIAH Institute of Technology

Continue statement

```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
} while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

RAMAIAH Institute of Technology

Continue Example

```
int main()
{
for (int j=0; j<=8; j++)
{
if (j==4)
continue;  printf("%d ", j);
} return 0;
}
```

Output

```
0 1 2 3 5 6 7 8
```

Continue Example

```
int main()
{
int counter=10;  while (counter >=0)
{
if (counter==7)
{
counter--;  continue;
}
printf("%d ", counter);
counter--;
}


        return 0;
}
```

Output

```
10 9 8 6 5 4 3 2 1 0
```

RAMAIAH Institute of Technology

Continue Example- Guess the output

```
int main()
{
int j=0;  do
{
if (j==7)
{
j++;
continue;
}
printf("%d ", j);
j++;
}while(j<10);

        return 0;
}
```

Continue Example

```
int main()
{
int j=0;  do
{
if (j==7)
{
j++;
continue;

                }
                printf("%d ", j);
                j++;
                }while(j<10);  return
                0;
}
```

0 1 2 3 4 5 6 8 9

RAMAIAH Institute of Technology

C – break statement

- It is used to come out of the loop instantly.
- When a break statement is encountered inside a loop, the control directly comes out of loop and the loop gets terminated.
- It is used with **if statement**, whenever used inside loop.
- This can also be used in switch case control structure. Whenever it is encountered in switch- case block, the control comes out of the switch- case.

RAMAIAH
Institute of Technology

C – break statement

```
while (testExpression) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
do {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

RAMAIAH Institute of Technology

Break Example- Guess the output

```
int main()
{
int num =0;  while(num<=100)
{
printf("value of variable num is: %d\n", num);
if (num==2)
{
break;
}
num++;
}
printf("Out of while-loop");

        return 0;
}
```

RAMAIAH Institute of Technology

Break Example

```
int main()
{
int num =0;  while(num<=100)
{
printf("value of variable num is: %d\n", num);
if (num==2)

                {
                                break;
                }
                num++;
        }
        printf("Out of while-loop");
        return 0;
}
```

Value of variable num is 0 Value of variable num is 1 Value of variable num is 2 Out of while loop

RAMAIAH Institute of Technology

Break Example- Guess the output

```
int main()
{
int var;
for (var =100; var>=10; var --)
{
printf("var: %d\n", var);
if (var==99)
{
break;
}
}
printf("Out of for-loop");  return 0;
}
```

Break Example- Guess the output

```
int main()
{
    int var;
    for (var =100; var>=10; var --)
    {
            printf("var: %d\n", var);
              if (var==99)
              {
                        break;
              }
    }
    printf("Out of for-loop");  return
    0;
    }
```

var: 100
var: 99 Out of for loop

Interesting examples in for loop

```
main()
{
int num=10;  for(;--num;)
printf("%d",num);
}
```

9 8 7 6 5 4 3 2 1

Interesting examples in for loop

```
main()
{
int i;  for(i=0;i<5;i++);
printf("%d ",i);
}
```

6

Interesting examples in for loop

```
#include<stdio.h>  main()
{
int x=1,y=1;
for(;y;printf("%d    %d\n",x,y))
y = x++ <= 5;
}
```

```
2  1
3  1
4  1
5  1
6  1
7  0
```

Thank you