Introduction to C++ Programming (PLC144)

UNIT – 1 & 2

Text Book: Object-Oriented Programming with C++ , E-Balaguruswamy.

Object-Oriented Programming with C++, Robert Lafore.

Programming with ANSI C++, Trivedi Bhushan

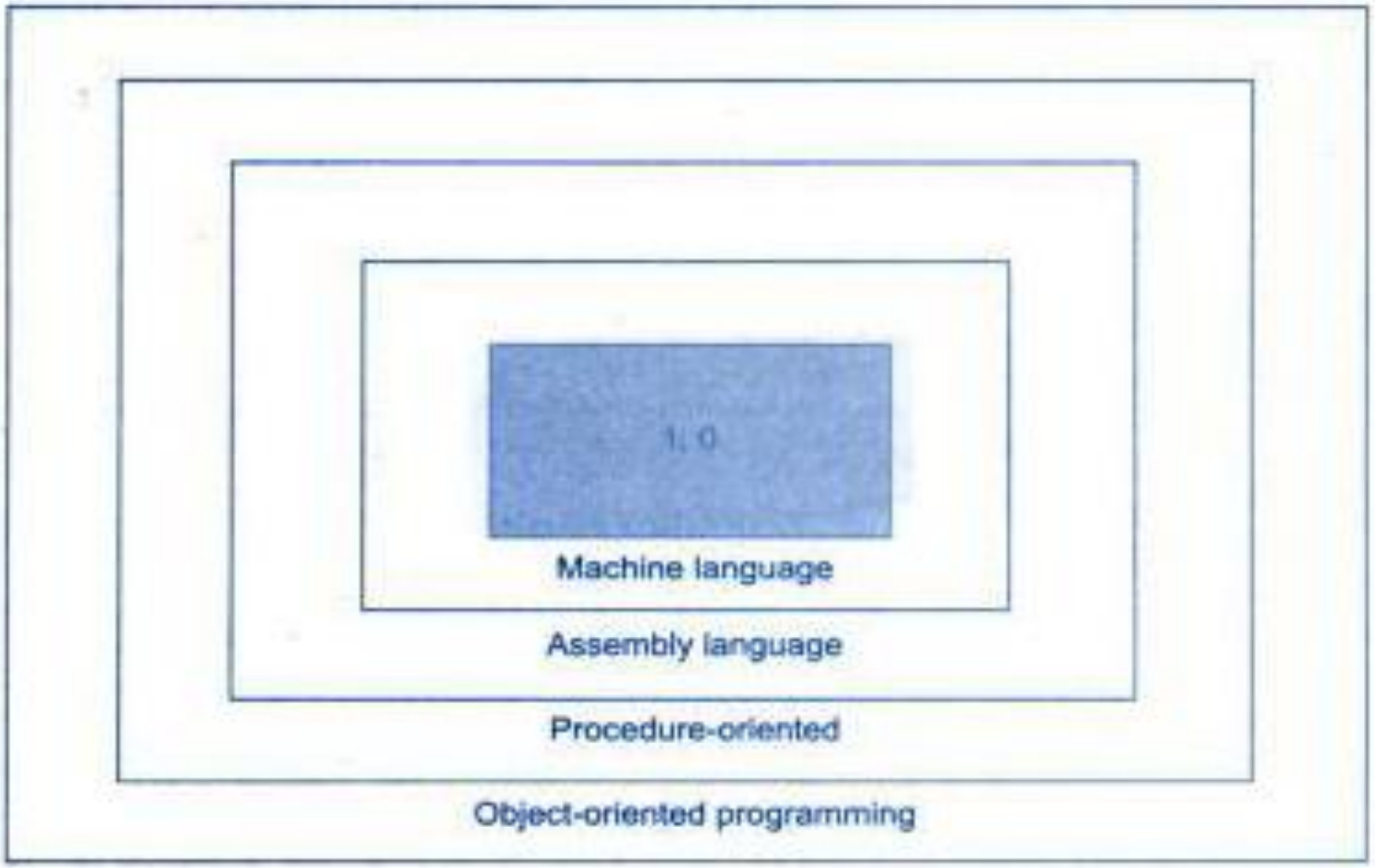


Fig. 1.3 Layers of computer software

- Structured Programming: Enabled Programmers to write moderately complex programs easily.

- Failed in terms of bug free, easy to maintain and reusable programs.

- Object oriented programming: approach to program organization and development.

- Procedure oriented programming:
- Viewed as sequence of things to be done (reading , calculating and printing)
- Number of functions are written to achieve these tasks
- Primary focus is on functions.
- Hierarchical decomposition for solving a problem.
- Writing a list of instructions and organizing these instructions into groups known as functions
- Concentrate on the development of functions
- Little attention given to data.
- Many important data items are placed as global.
- Each function may have its own local data.
- Global data are vulnerable to an inadvertent change by a function.
- Does not attend to real world problem.
- Functions are action oriented and does not correspond to elements of problem

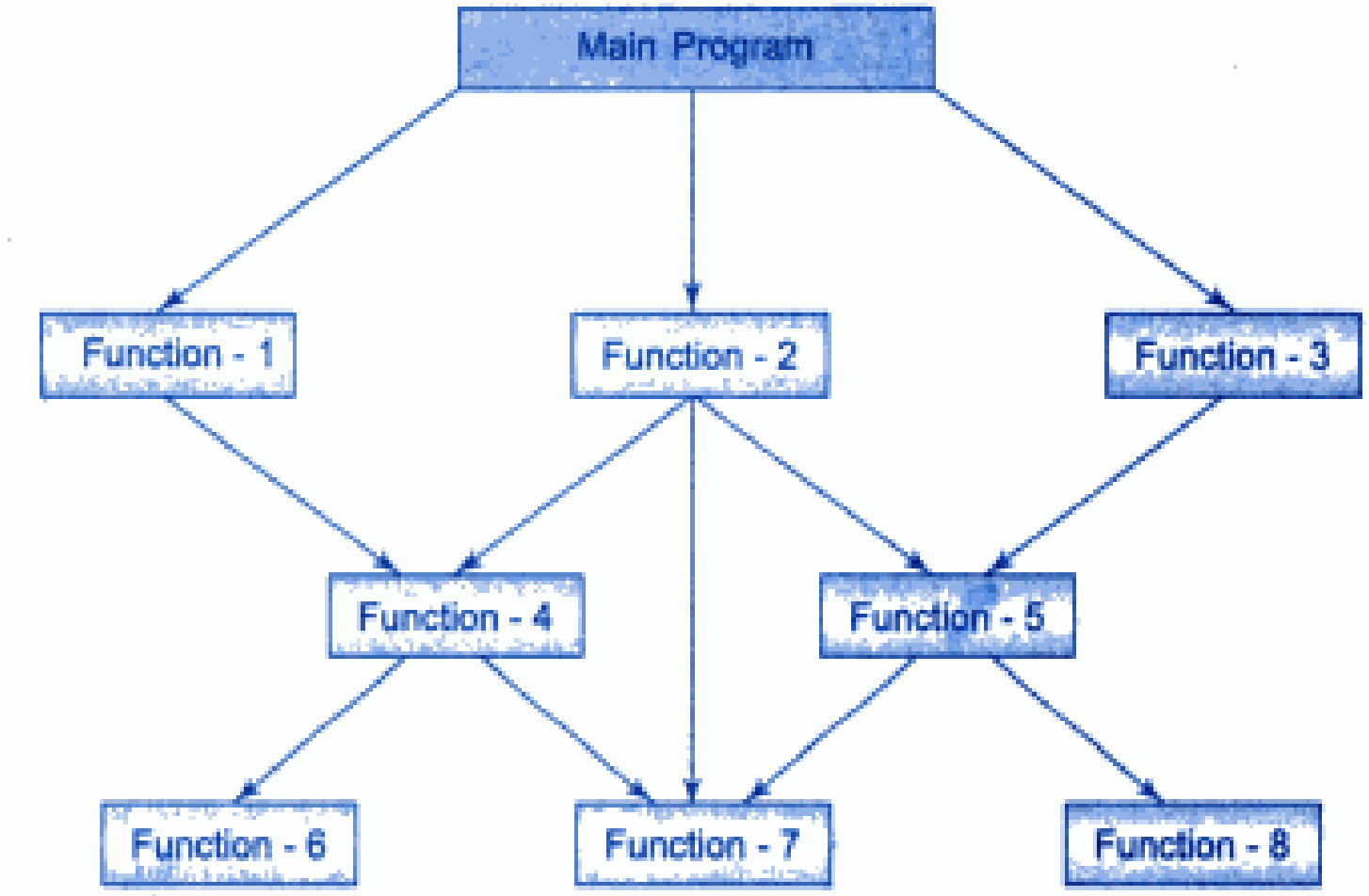


Fig. 1.4 ⇒ *Typical structure of procedure-oriented programs*

Characteristics of Procedure Oriented Programming:

- Emphasis is on doing things.
- Large Programs are divided into smaller programs known as functions.
- Most of the functions share global data.
- Data move openly around the system from function to function.
- Functions transform data from one form to other.
- Employs top down Approach.

Object Oriented Programming:

- Motivating factor of OOP to remove flaws encountered in procedural approach.
- OOP treats data as critical element and does not allow it to move freely.
- Ties data to function closely.
- Protects from accidental modification.
- Decomposition of problem into objects and builds data and functions around these objects.
- Data of an object can be accessed only by functions associated with that object.

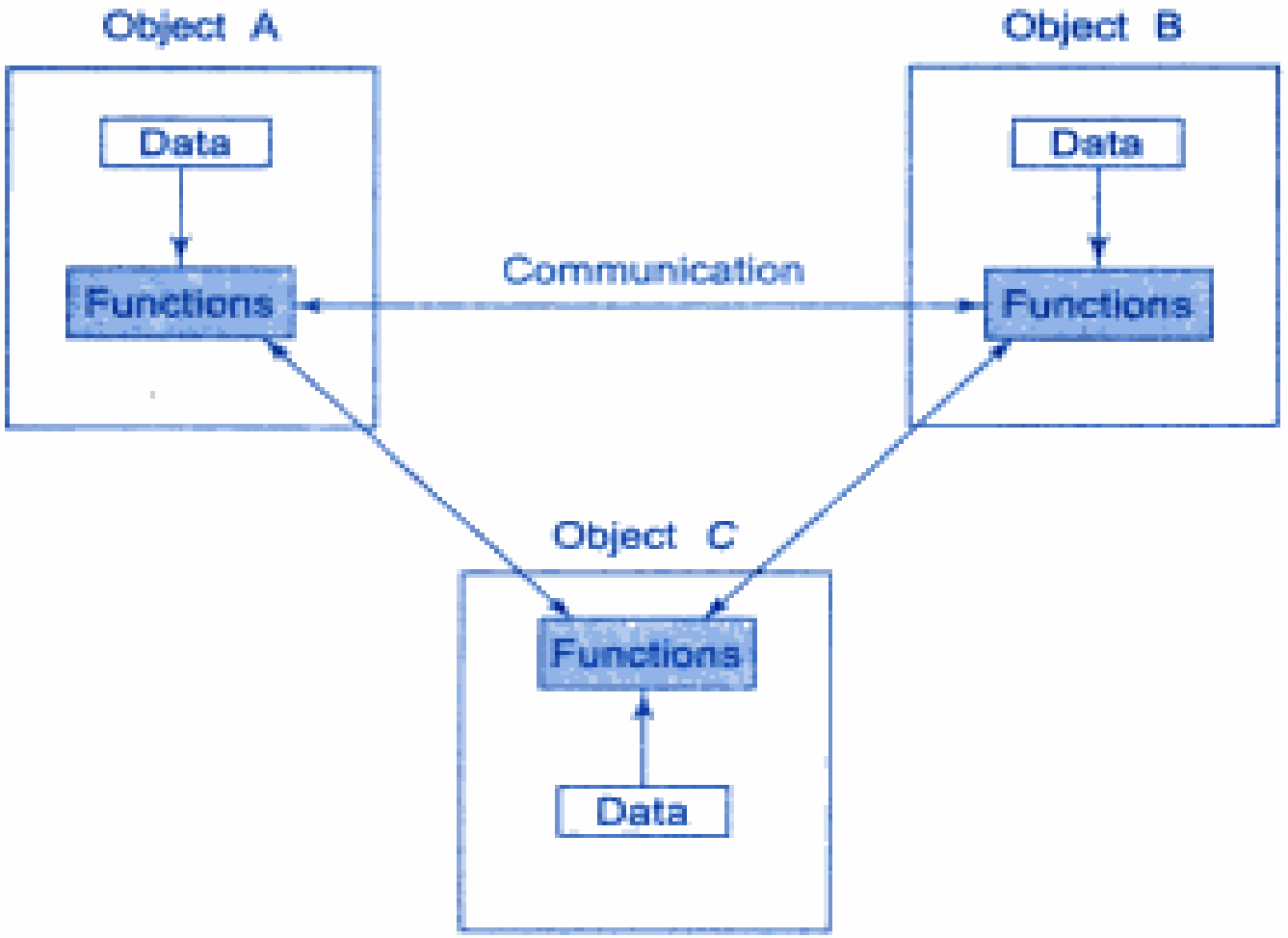


Fig. 1.6 ⇔ *Organization of data and functions in OOP*

- Striking features of object oriented programming:
- Emphasis is on data rather than Procedure.

- Programs are divided into objects.

- Data structures are designed such that they characterize the objects.

- Functions that operate on that data of an object are tied together in the data structure.

- Data is hidden and cannot be accessed by external functions.

- Objects may communicate with each other through functions.

- New data and functions can be easily added whenever necessary.

- Follows bottom up approach.

What is OOP?

- Approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

Basic Concepts of Object oriented Programming.

- Objects
- Classes
- Data abstraction and encapsulation.
- Inheritance.
- Polymorphism.
- Dynamic Binding
- Message Passing

Objects:

- Represent Run time Entities.
- Objects take up space in the memory.
- On execution objects interact by sending messages to one another.
- Objects interact without needing know about each others data or code.
- Sufficient to know type of message accepted and type of response returned by objects.

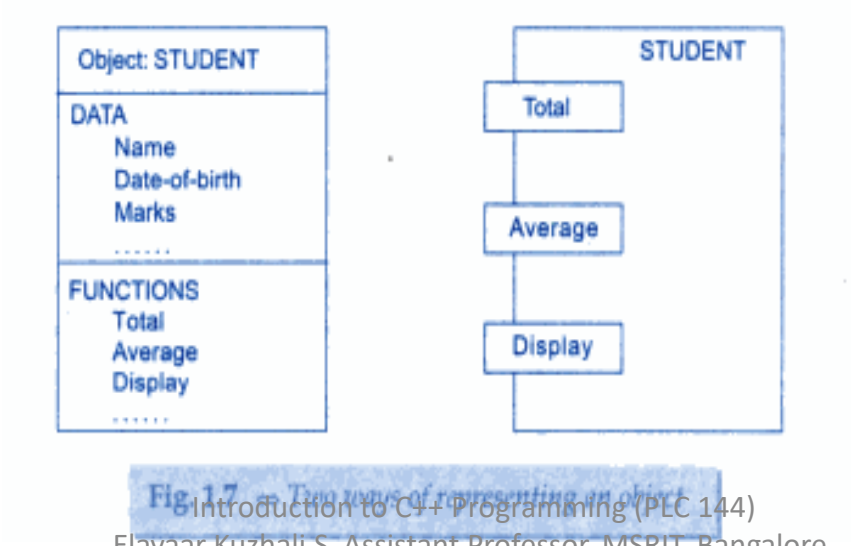


Fig. 1.7 — Two ways of representing an object

Classes:

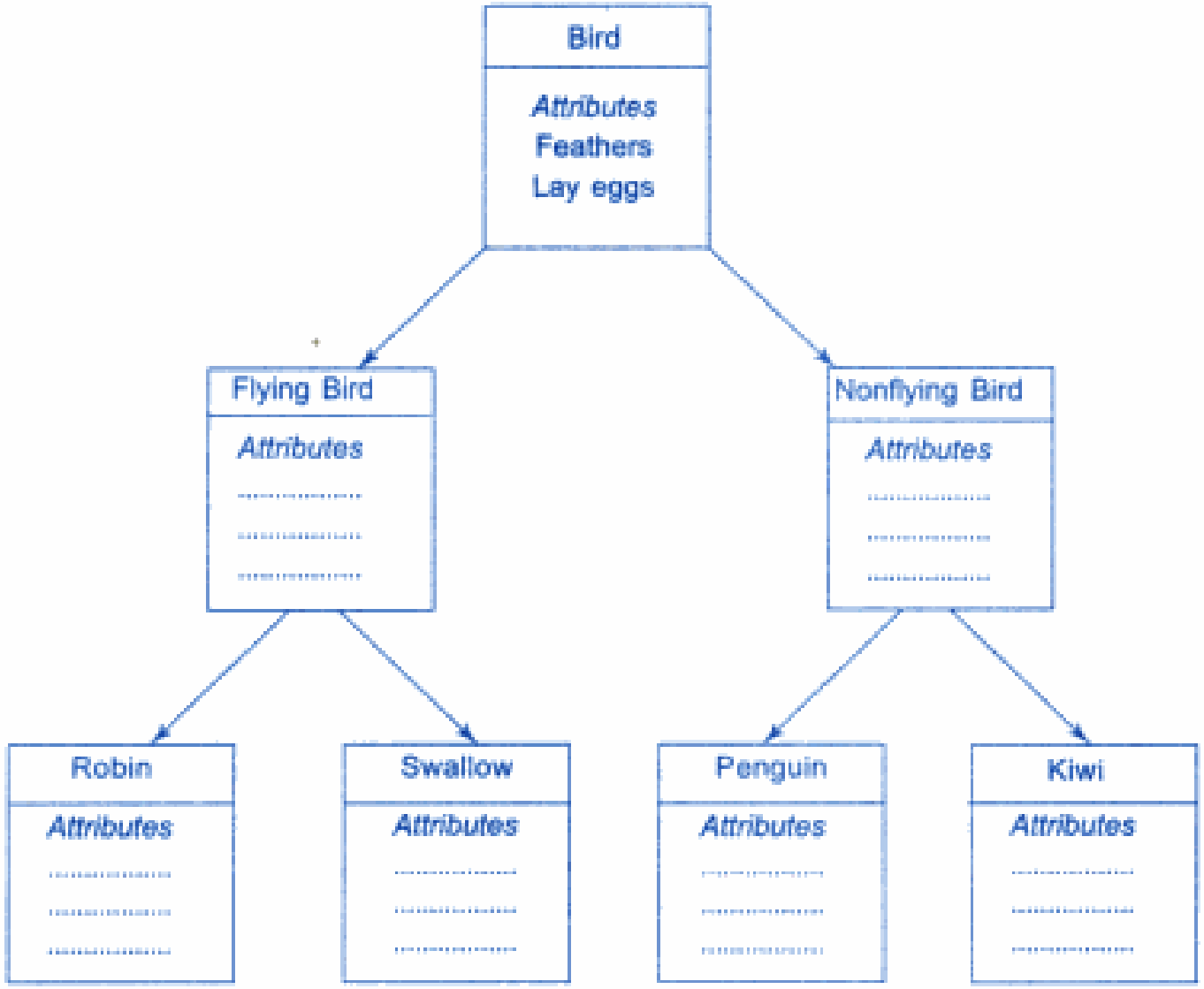
- Entire set of data and function of object made User defined data type.
- Objects are variables of type class.
- Once class is defined any number of objects can be created.
- Class is collection of objects.
- Mango, apple, Orange are members of class fruit.
  - Fruit mango

Data Abstraction and Encapsulation:

- Wrapping up of data and functions into a single unit is known as encapsulation.
- Data is not accessible outside the class. Only the function wrapped in the class can access it.
- These functions provide interface between objects data and the program.
- The insulation of data from direct access by the program is called data hiding.
- Abstraction refers to act of representing essential features.
- List of abstract attributes and functions to operate on these attributes.
- Attributes are called data members.
- Functions that operate on them are called member functions.

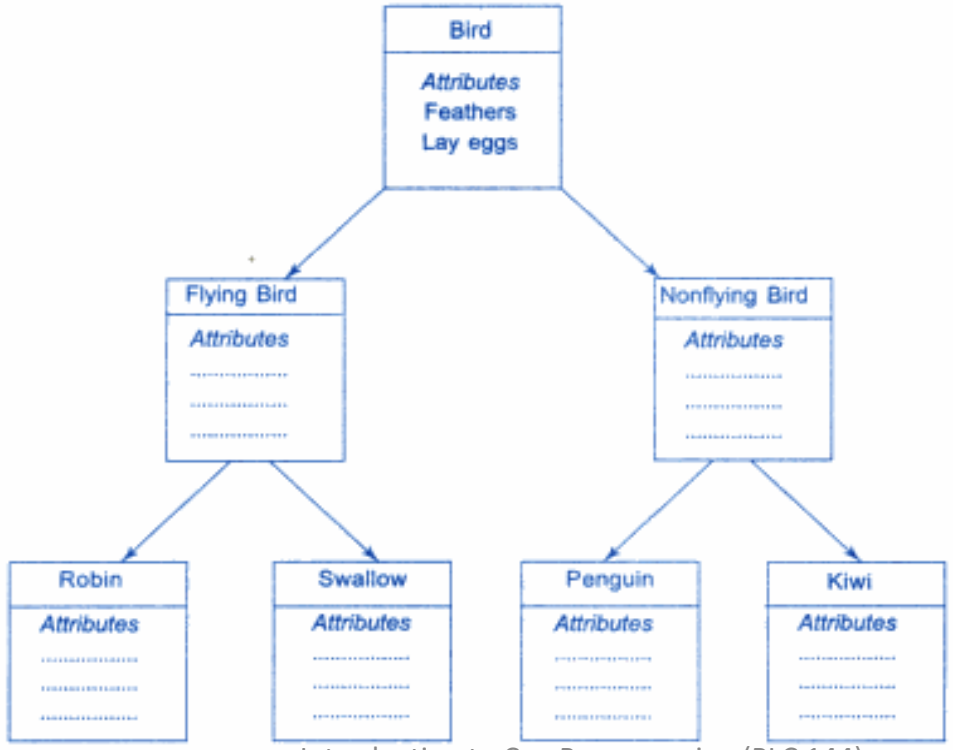
Inheritance:

- Objects of one class acquires the properties of objects of another class.
- Supports Hierarchical classification
- Provides concept of Reusability – can add addition features without modifying the existing class.
- New class is derived from existing class.
- The new class will have combined features.


Bird
Attributes
Feathers
Lay eggs
Flying Bird
Attributes
Nonflying Bird
Attributes
Robin
Attributes
Swallow
Attributes
Penguin
Attributes
Kiwi
Attributes

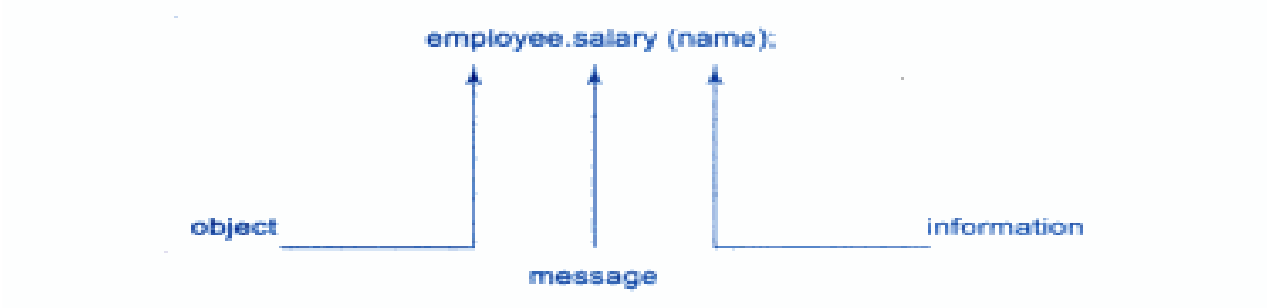
Polymorphism:

- Ability to take more than one form.
- Operator overloading, Function Overloading
- Allows to have different internal structure and share same external interface.



- Dynamic Binding:
- Binding – Linking of procedure call to code to be executed in response to the call.
- Dynamic Binding – code associated with the call is known in run time.

- Message Passing:
- 1) Create classes that define objects and their behaviour.
- 2) Creating objects from class definitions.
- 3) Establishing communication among objects.



Benefits of OOP:

- Through Inheritance can eliminate redundant code and extend the use of existing class.
- Can build programs from the standard working modules, saves development time and higher productivity.
- Data hiding helps the programmer to build secure programs.
- Easy to partition the project based on objects.
- Easy to upgrade from small to large systems.
- Software complexity can be easily managed.

Applications

- Real time Systems
- Simulations and Modelling
- Object Oriented Databases
- Hypertext, Hypermedia and Experttext
- AI & Expert Systems
- Neural Networks and Parallel Programming
- Decision Support And Office Automation Systems

Simple C++ Program

we will learn to create a simple program named "Hello World" in C++ programming.

```
// C++ program to display "Hello World"

// Header file for input output functions
#include <iostream>
using namespace std;

// Main() function: where the execution of program begins
int main()
{
        // prints hello world
        cout << "Hello World";

        return 0;
}
```

1) **// C++ program to display "Hello World":** This line is a comment line. A comment is used to display additional information about the program. A comment does not contain any programming logic. When a comment is encountered by a compiler, the compiler simply skips that line of code. Any line beginning with '//' without quotes OR in between /*…*/ in C++ is comment.
2) **#include**: In C++, all lines that start with pound (#) sign are called directives and are processed by a preprocessor which is a program invoked by the compiler. The **#include** directive tells the compiler to include a file and **#include<iostream>**. It tells the compiler to include the standard iostream file which contains declarations of all the standard input/output library functions.
3) **using namespace std**: This is used to import the entirety of the std namespace into the current namespace of the program.
4) **int main()**: This line is used to declare a function named "main" which returns data of integer type. A function is a group of statements that are designed to perform a specific task. Execution of every C++ program begins with the main() function, no matter where the function is located in the program. So, every C++ program must have a main() function.
5) **{ and }:** The opening braces '{' indicates the beginning of the main function and the closing braces '}' indicates the ending of the main function. Everything between these two comprises the body of the main function.
6) **cout<<"Hello World";:** This line tells the compiler to display the message "Hello World" on the screen. This line is called a statement in C++. Every statement is meant to perform some task. A semi-colon ';' is used to end a statement.
7) **return 0; :** This is also a statement. This statement is used to return a value from a function and indicates the finishing of a function.
8) **Indentation:** As you can see the cout and the return statement have been indented or moved to the right side. This is done to make the code more readable.

cout and output operator

- **cout** is an object of the class ostream.
- The operator << is called the insertion operator. It sends the contents of the variable on its right to the object on its left.
- Eg:

cout<<“C++”;

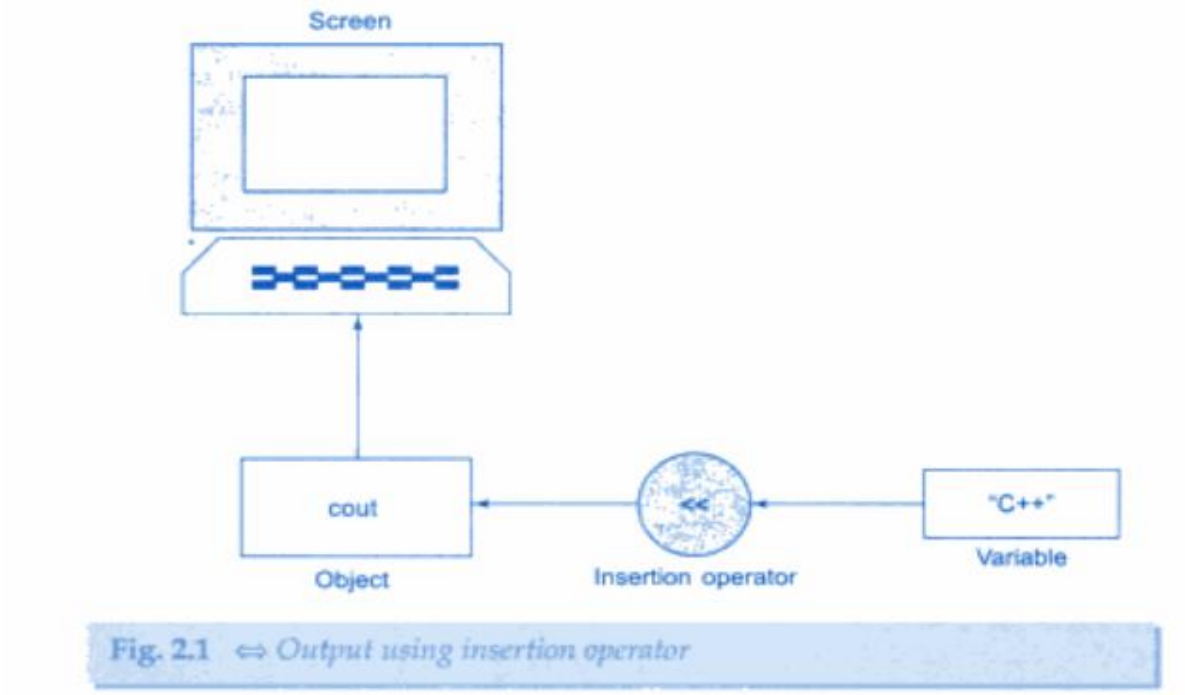


Fig. 2.1 ⇔ *Output using insertion operator*

Basic Input in C++

```
#include <iostream>
using namespace std;

int main()
{
        int age;

        cout << "Enter your age:";
        cin >> age;
        cout << "\nYour age is: " << age;

        return 0;
}
```

The program asks the user to input the age.

cin >> age ; is an input statement.

The statement causes the program to wait for the user to type in a number. The number keyed in is placed in the variable age.

The identifier cin is predefined object that corresponds to standard input stream.

The operator >> is known as Extraction operator

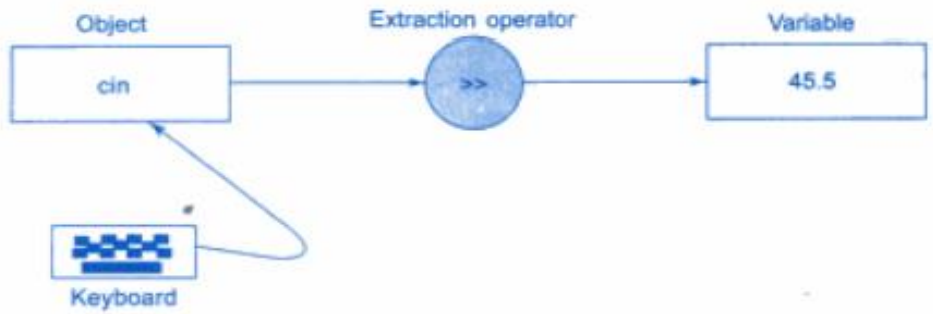


Fig. 2.2 ⇔ Input using extraction operator

Cascading of IO Operators

- The consecutive occurrence of input or output operators in a single statement. This is certainly an important part of the programming of C++
- Eg:

```
cout << "\nYour age is: " << age;


cin >> Var1 >> Var2;
```

Multiple cin and cout statements can be cascaded using >> and << operators.

Tokens:

- The smallest Individual Units in a program are known as tokens. C++ has following tokens
- Keywords,
- Identifiers
- Constants
- Strings
- Operators.

Keywords:

- They are reserved identifiers and cannot be used as names for the program variables.

Table 3.1 *C++ keywords*

asm	**double**	new	**switch**
auto	**else**	operator	template
break	**enum**	private	this
case	**extern**	protected	throw
catch	**float**	public	try
char	**for**	**register**	**typedef**
class	friend	**return**	**union**
const	**goto**	**short**	**unsigned**
continue	**if**	**signed**	virtual
default	inline	**sizeof**	**void**
delete	**int**	**static**	**volatile**
do	**long**	**struct**	**while**
Added by ANSI C++			
bool	export	reinterpret_cast	typename
const_cast	false	static_cast	using
dynamic_cast	mutable	true	wchar_t
explicit	namespace	typeid	

Identifiers and Constants

- Identifiers refer to the names of variables, functions, arrays, classes etc. created by the programmer.
- Rules for naming these Identifiers:
- Only Alphabetic Characters, digits and Underscores are permitted.
- The name cannot start with a digit.
- Uppercase and Lowercase letters are distinct
- A declared Keyword cannot be used as a Variable name.

Constants:

Eg:

10	// decimal Integer
2.5	// floating point integer
0x37	// Hexa decimal Integer
“Hello”	// String Constant
‘A’	// Character Constant
L’ab’	// Wide Character constant

Expressions and Their Types:

- Expression: is a combination of operators, constants and variables
- May consist of one or more operands
- May consist of zero or more operators.

Types:

Constant Expression: Consist only of constant values

E.G:

10

10 + 20/5

'A'

Integral Expression : Produce Integer Results

E.G:

X

X+Y*5

Float Expression: Produce Floating Point Results

E.G:

X*Y/5

5+float(5)

10.6

Pointer Expression: Produce address Values

E.G:

int * ptr = &m

Ptr = ptr + 1

Relational Expression: Produce results of type bool (true or false)

E.G:

X <=Y

a+b == c+d

M >10

Logical Expression : Combine two or more relational expressions and produces bool type results.

E.G:

A>B && C<D

A==10 || C>10

Bitwise Expression : Used to manipulate data at bit level.

E.G:

A &B , C | D, X>>2, Y<< 2, ~Y

Operators in C++

- All C operators are valid in C++
- C++ introduces new operators.
- >> Extraction Operator, << Insertion Operator
- :: Scope Resolution Operator.
- ::* Pointer to member declarator
- ->* Pointer to member operator
- .* Pointer to member operator
- new - memory allocation operator
- delete - memory release operator

Scope Resolution Operator

- C++ is a block structured language.
- Blocks and Scopes can be used in constructing programs
- Same variable name can be used to have different meanings in different blocks.
- The scope of the variable extends from the point of its declaration till the end of the block containing the declaration.
- A variable declared inside the block is said to be local to the block.

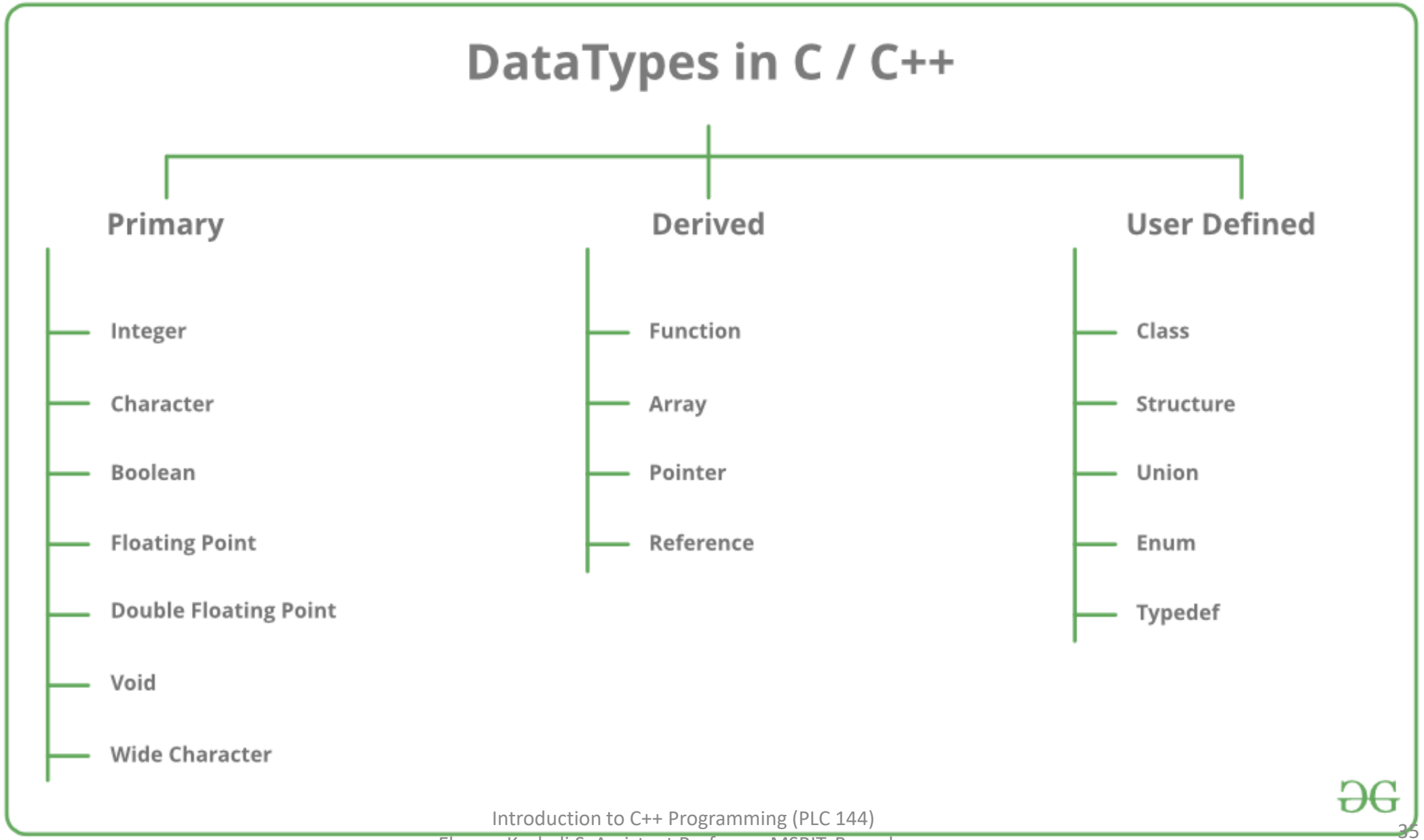
```
int main()
{
//BLOCK 1
   int x = 10;
   cout<< x << endl;
   // BLOCK 2
  {
      int x = 20;
      cout<< x << endl;
   }

   cout<< x << endl;
   return 0;
}
```

The Two declarations of x refers to different memory locations.
Statements in second block cannot refer to x in other block and vice versa

BLOCK 2 is contained in BLOCK1.
Declaration in inner block hides the declaration of outer block.

```
using namespace std;
int x = 5;
int main()
{
//BLOCK 1
   int x = 10;

   cout<< x << endl;
 // BLOCK 2
   {
      int x = 20;
      cout << x << endl;
      cout<<:: x << endl;
   }

   cout<< x << endl;
   return 0;
}
```

Global version of x cannot be accessed from inner block.
Soln: use scope resolution operator to access global variable ::x

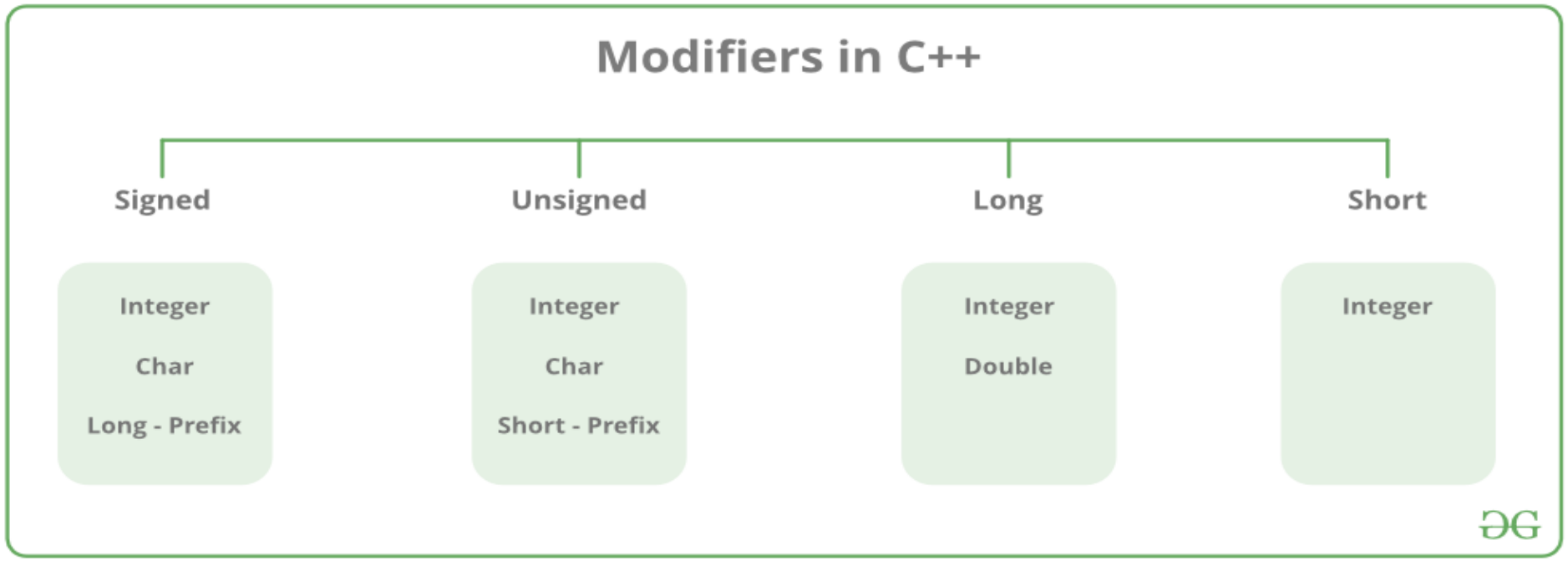

DataTypes in C / C++
Primary
Integer
Character
Boolean
Floating Point
Double Floating Point
Void
Wide Character
Derived
Function
Array
Pointer
Reference
User Defined
Class
Structure
Union
Enum
Typedef

- **Primitive Data Types**: These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char , float, bool etc. Primitive data types available in C++ are:
- Integer
- Character
- Boolean
- Floating Point
- Double Floating Point
- Valueless or Void
- Wide Character

- Derived Data Types: The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:
- Function
- Array
- Pointer
- Reference

- User-Defined Data Types: These data types are defined by user itself. Like, defining a class in C++ or a structure. C++ provides the following user-defined datatypes:
- Class
- Structure
- Union
- Enumeration
- Typedef defined DataType

- **Datatype Modifiers**
- As the name implies, datatype modifiers are used with the built-in data types to modify the length of data that a particular data type can hold.



short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	8	-2,147,483,648 to 2,147,483,647

unsigned long int	4	0 to 4,294,967,295
long long int	8	-(2^63) to (2^63)-1
unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	
double	8	
long double	12	
wchar_t	2 or 4	1 wide character

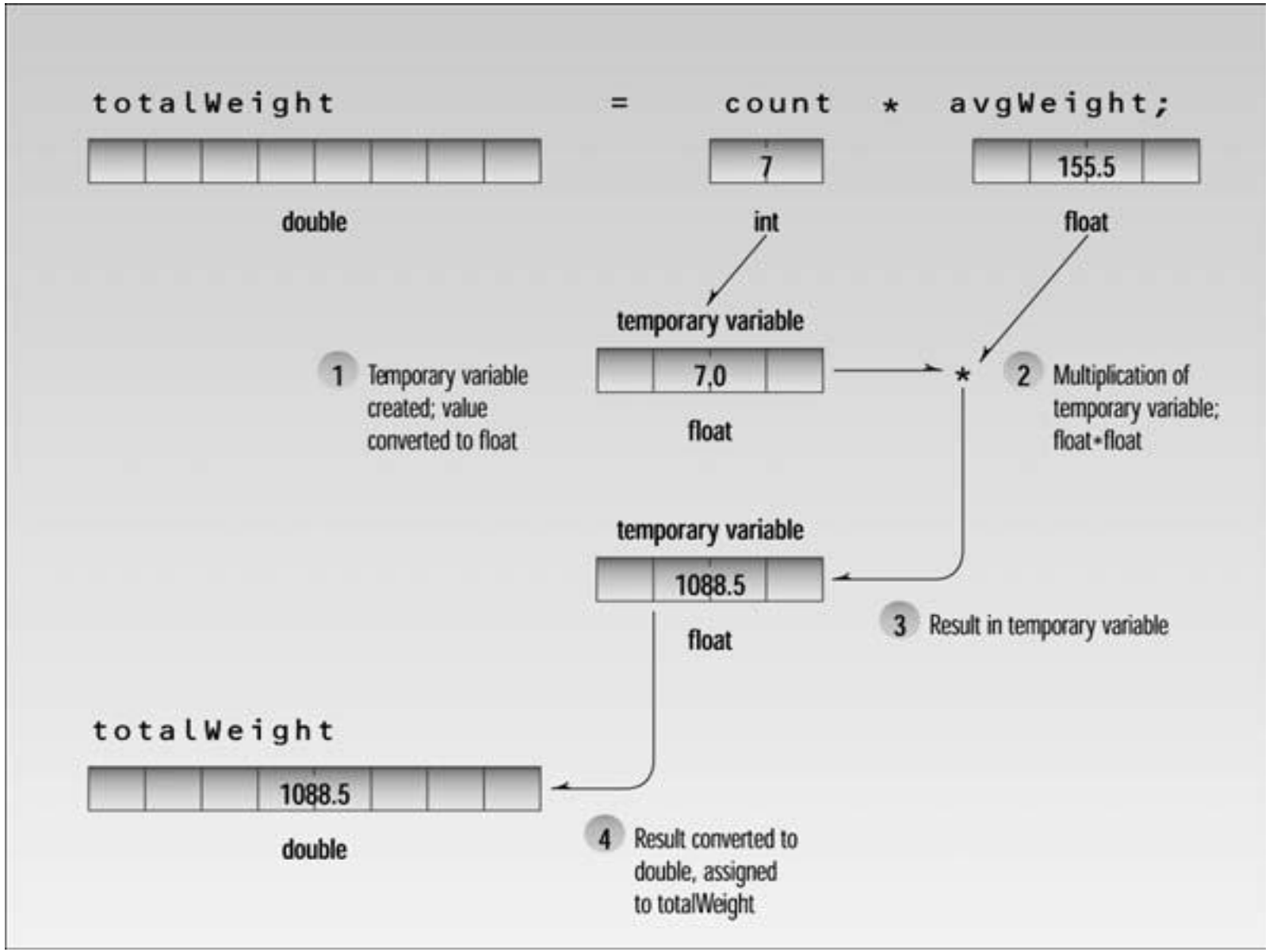
Keyword	Numerical Range		Digits of Precision	Bytes of Memory
	Low	High		
int	-2,147,483,648	2,147,483,647	n/a	4
long	-2,147,483,648	2,147,483,647	n/a	4
float	3.4×10^{-38}	3.4×10^{38}	7	4
double	1.7×10^{-308}	1.7×10^{308}	15	8

TABLE 2.4 Order of Data Types

Data Type	*Order*
`long double`	Highest
`double`	
`float`	
`long`	
`int`	
`short`	
`char`	Lowest

```
using namespace std;
int main()
{
int signedVar = 1500000000;                    //signed
unsigned int unsignVar = 1500000000;           //unsigned
signedVar = (signedVar * 2) / 3; //calculation exceeds range
unsignVar = (unsignVar * 2) / 3; //calculation within range
cout << “signedVar = “ << signedVar << endl;          //wrong
cout << “unsignVar = “ << unsignVar << endl;          //OK
return 0;
}
```

```
int main()
{
int count = 7;
float avgWeight = 155.5F;
double totalWeight = count * avgWeight;
cout << “totalWeight=” << totalWeight << endl;
return 0;
}
```

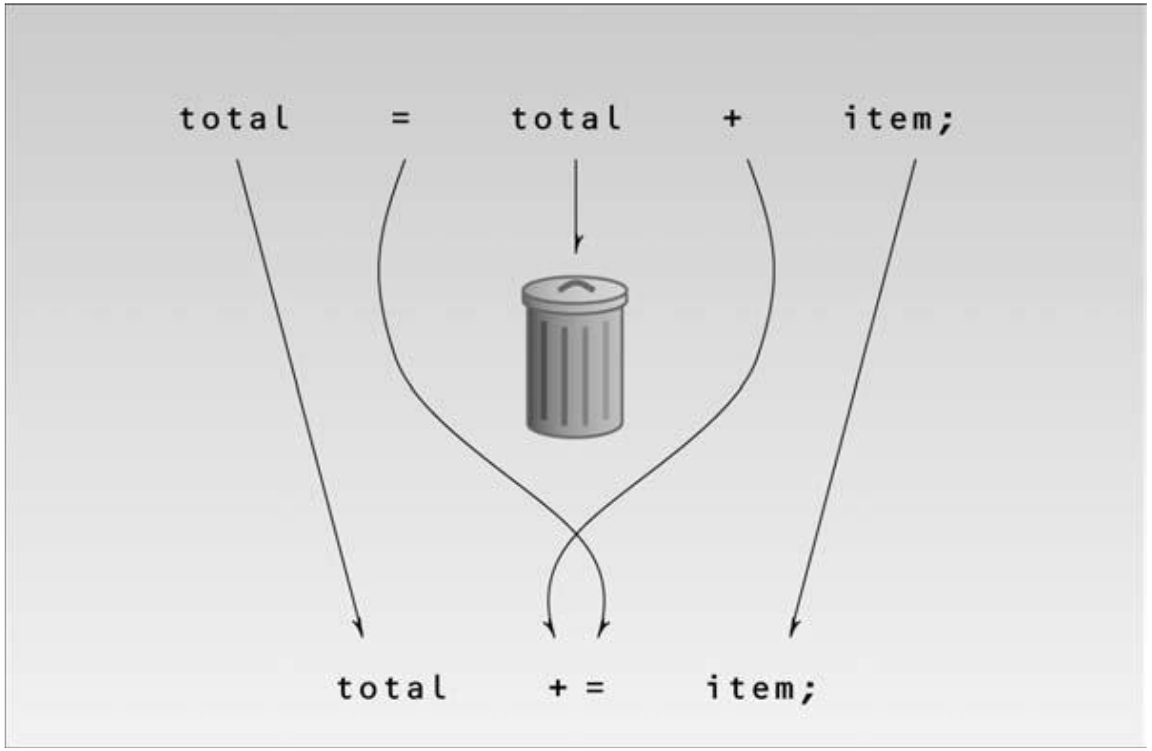

totalWeight = count * avgWeight;
7
155.5
double
int
float
temporary variable
7.0
float
1 Temporary variable created; value converted to float
*
2 Multiplication of temporary variable; float*float
temporary variable
1088.5
float
3 Result in temporary variable
totalWeight
1088.5
double
4 Result converted to double, assigned to totalWeight

```
#include <iostream>
using namespace std;
int main()
{
int intVar = 1500000000;                    //1,500,000,000
intVar = (intVar * 10) / 10;                //result too large
cout << “intVar = “ << intVar << endl;      //wrong answer
intVar = 1500000000;                         //cast to double
intVar = (double(intVar) * 10) / 10;
cout << “intVar = “ << intVar << endl;      //right answer
return 0;
}
```

- **Arithmetic Operators**
- arithmetic operators +, -, *, and /
- for addition, subtraction, multiplication, and division.

```
#include <iostream>
using namespace std;
int main()
{
cout << 6 % 8 << endl                  // 6
<< 7 % 8 << endl                        // 7
<< 8 % 8 << endl                        // 0
<< 9 % 8 << endl                        // 1
<< 10 % 8 << endl;                      // 2
return 0;
}
```

- **Arithmetic Assignment Operators**
- total = total + item; // adds “item” to “total”
- total += item; // adds “item” to “total”



```
#include <iostream>
using namespace std;
int main()
{
int ans = 27;
ans += 10;          //same as: ans = ans + 10;
cout << ans << “, “;
ans -= 7;           //same as: ans = ans - 7;
cout << ans << “, “;
ans *= 2;                   //same as: ans = ans * 2;
cout << ans << “, “;
ans /= 3;                   //same as: ans = ans / 3;
cout << ans << “, “;
ans %= 3;           //same as: ans = ans % 3;
cout << ans << endl;
return 0;
}
```

- **Increment Operators**
- count = count + 1; // adds 1 to “count”

Or you can use an arithmetic assignment operator:

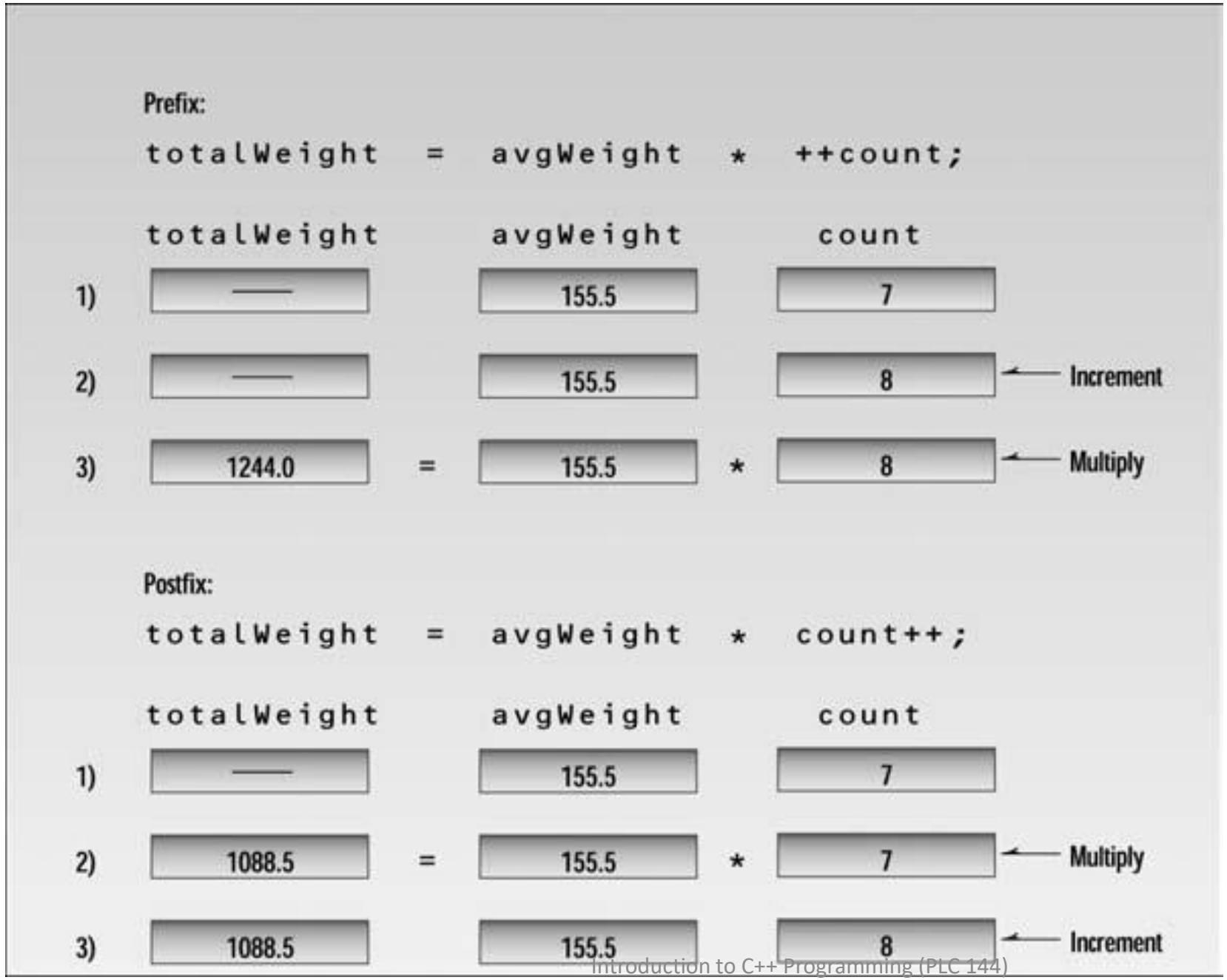
- count += 1; // adds 1 to “count”

But there’s an even more condensed approach:

- ++count; // adds 1 to “count”

The ++ operator increments (adds 1 to) its argument.

Prefix,Postfix


Prefix:
totalWeight = avgWeight * ++count;
totalWeight avgWeight count
1) —— 155.5 7
2) —— 155.5 8 ← Increment
3) 1244.0 = 155.5 * 8 ← Multiply
Postfix:
totalWeight = avgWeight * count++;
totalWeight avgWeight count
1) —— 155.5 7
2) 1088.5 = 155.5 * 7 ← Multiply
3) 1088.5 155.5 8 ← Increment

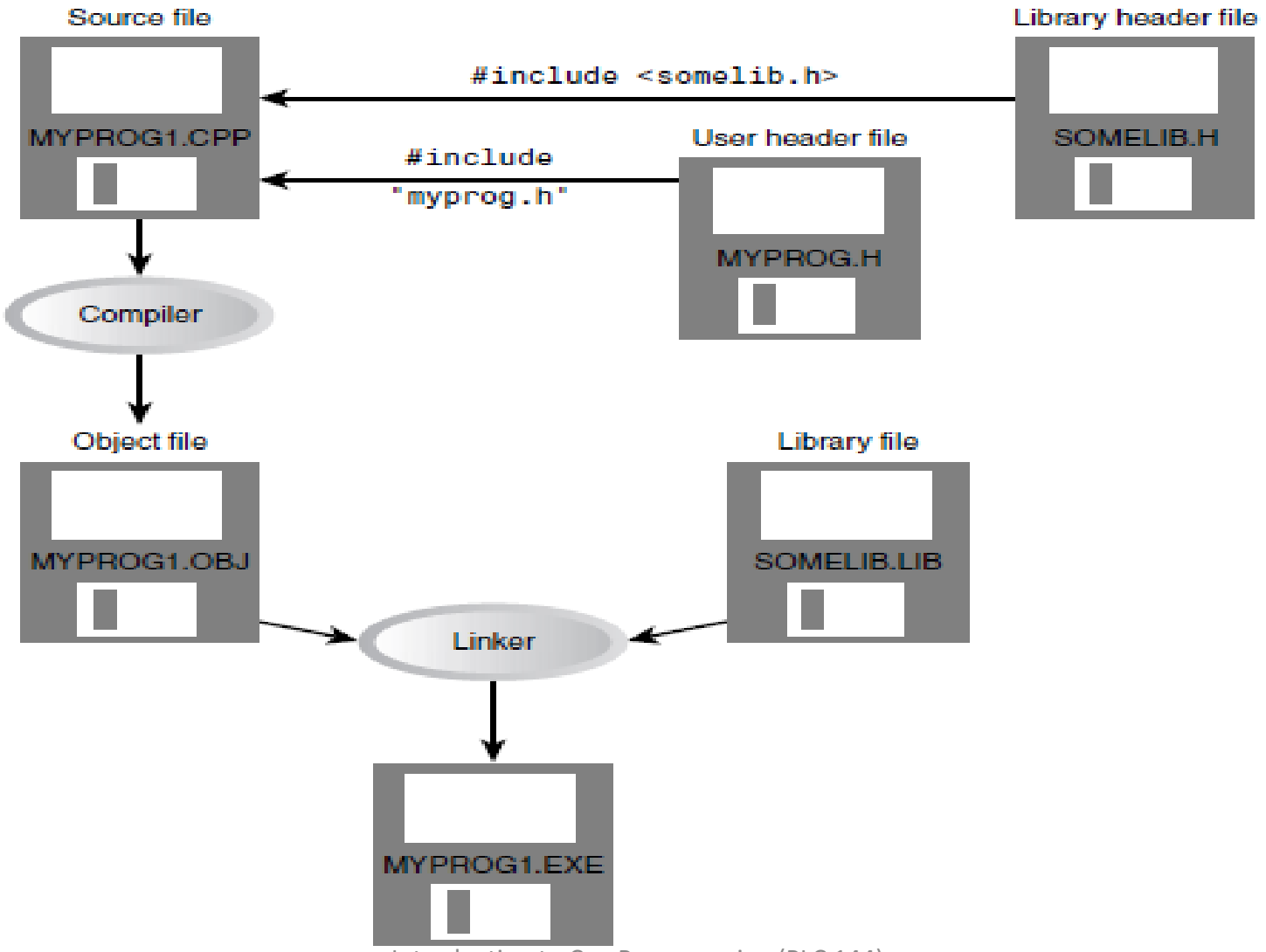
```
#include <iostream>
using namespace std;
int main()
{
int count = 10;
cout << “count=” << count << endl;          //displays 10
cout << “count=” << ++count << endl;        //displays 11 (prefix)
cout << “count=” << count << endl;          //displays 11
cout << “count=” << count++ << endl;        //displays 11 (postfix)
cout << “count=” << count << endl;          //displays 12
return 0;
}
```

- **The Decrement (--) Operator**
- The decrement operator, --, behaves very much like the increment operator, except that it subtracts 1 from its operand. It too can be used in both prefix and postfix forms.

Library Functions

These functions perform file access, mathematical computations, and data conversion, among other things.

```
#include <iostream>                    //for cout, etc.
#include <cmath>                       //for sqrt()
using namespace std;
int main()
{
double number, answer; //sqrt() requires type double
cout << “Enter a number: “;
cin >> number; //get the number
answer = sqrt(number); //find square root
cout << “Square root is “ << answer << endl; //display it
return 0;
}
```


Source file
Library header file
#include <somelib.h>
MYPROG1.CPP
SOMELIB.H
User header file
#include
"myprog.h"
MYPROG.H
Compiler
Object file
Library file
MYPROG1.OBJ
SOMELIB.LIB
Linker
MYPROG1.EXE

Questions to solve

- Assuming there are 7.481 gallons in a cubic foot, write a program that asks the user to enter a number of gallons, and then displays the equivalent in cubic feet.
- Write a program that generates the following table:

1990 135

1991 7290

1992 11300

1993 16200

Use a single cout statement for all output.

- cout<<“135”<<endl<<“1440”<<endl;

```
 135
1440
```

```
cout<<setw(4)<<“135”<<endl;          135
cout<<setw(4)<<“1440”<<endl;        1440
```

```
#include <iostream>
using namespace std;
int main()
{
float gallons, cufeet;
cout << “\nEnter quantity in gallons: “;
cin >> gallons;
cufeet = gallons / 7.481;
cout << “Equivalent in cublic feet is “ << cufeet << endl;
return 0;
}
```

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
cout << 1990 << setw(8) << 135 << endl
<< 1991 << setw(8) << 7290 << endl
<< 1992 << setw(8) << 11300 << endl
<< 1993 << setw(8) << 16200 << endl;
return 0;
}
```

- A library function, islower(), takes a single character (a letter) as an argument and returns a nonzero integer if the letter is lowercase, or zero if it is uppercase. This function requires the header file CTYPE.H. Write a program that allows the user to enter a letter, and then displays either zero or nonzero, depending on whether a lowercase or uppercase letter was entered. (See the SQRT program for clues.)

```
#include <iostream>
#include <ctype.h>
using namespace std;
int main()
{
        char ch;
        int j;
        cout<<“Enter  a char “;
        cin>>ch;
        j=islower(ch);
        if(j)
        cout<<“typed character is in lower case”<<endl;
        else
        cout<<“typed character is in upper case”<<endl;
}
```

- On a certain day the British pound was equivalent to $1.487 U.S., the French franc was $0.172, the German deutschemark was $0.584, and the Japanese yen was $0.00955. Write a program that allows the user to enter an amount in dollars, and then displays this value converted to these four other monetary units.

If you have two fractions, a/b and c/d, their sum can be obtained from the formula

$$\frac{a}{b} + \frac{c}{d} = \frac{a*d + b*c}{b*d}$$

For example, 1/4 plus 2/3 is

$$\frac{1}{4} + \frac{2}{3} = \frac{1*3 + 4*2}{4*3} = \frac{3 + 8}{12} = \frac{11}{12}$$

Write a program that encourages the user to enter two fractions, and then displays their sum in fractional form. (You don't need to reduce it to lowest terms.)

The interaction with the user might look like this:

Enter first fraction: 1/2

Enter second fraction: 2/5

Sum = 9/10

```
#include <iostream>
using namespace std; int main()
{
    int num1,den1,num2,den2,num3,den3;
    char slash;
    cout << “Happy Programming" << endl;
    cout << "Enter the first fraction: " ;
    cin >> num1 >> slash >> den1;
    cout << endl;
    cout << "Enter the second fraction: " ;
    cin >> num2 >> slash >> den2;
    cout << endl;
    num3 = (num1*den2) + (num2*den1);
    den3 = den1*den2;
    cout << "Addition of the two fractions is " << num3 << "/" << den3 ;
    return 0;
}
```

An electricity board charges the following rates to domestic users to discourage large consumption of energy.

For the first 100 units Rs. 1.50 per unit
For the next 200 units Rs. 3.00 per unit
Beyond 300 units Rs. 5.00 per unit

All users are charged a minimum of Rs.100. If the total cost exceeds Rs.250, then an additional surcharge of 15% is added. Write a program to read the names of users and number of units consumed and print out the charges with names.

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello world!" << endl;
   int i;
   char names[50]; int units; double cost;

   cout<<"Enter names of the users and the number of units consumed:"<<endl;

   cout<<"Name: " ;
   cin>>names;
   cout<<"Units:" ;
   cin>>units;

   cout<<endl;
   cout<<"Name\t\tCost"<<endl;

   if(units <= 100)
     {
     cost = (units*1.50) + 100;
     }
     else if(units > 100 && units <= 300)
     {
     cost=((100*1.50) + ((units-100) * 3)) + 100;
     }
     else
     {
     cost =((100*1.50) + (200*3)+((units - 300)*5)) + 100;
     }

     if(cost > 250)
     {
     cost += (cost * 15/100);
     }
     cout<<names <<"\t\tRs."<<cost<<endl;
     return 0;
}
```

Relational Operators

A relational operator compares two values.

- The comparison involves such relationships as equal to, less than, and greater than.

```
#include <iostream>
using namespace std;
int main()
{
int numb;
cout << “Enter a number: “;
cin >> numb;
cout << “numb<10 is “ << (numb < 10) << endl;
cout << “numb>10 is “ << (numb > 10) << endl;
cout << “numb==10 is “ << (numb == 10) << endl;
return 0;
}
```

Output:

```
Enter a number: 20
numb<10 is 0
numb>10 is 1
numb==10 is 0
```

Operator Meaning

> Greater than (greater than)

< Less than

== Equal to

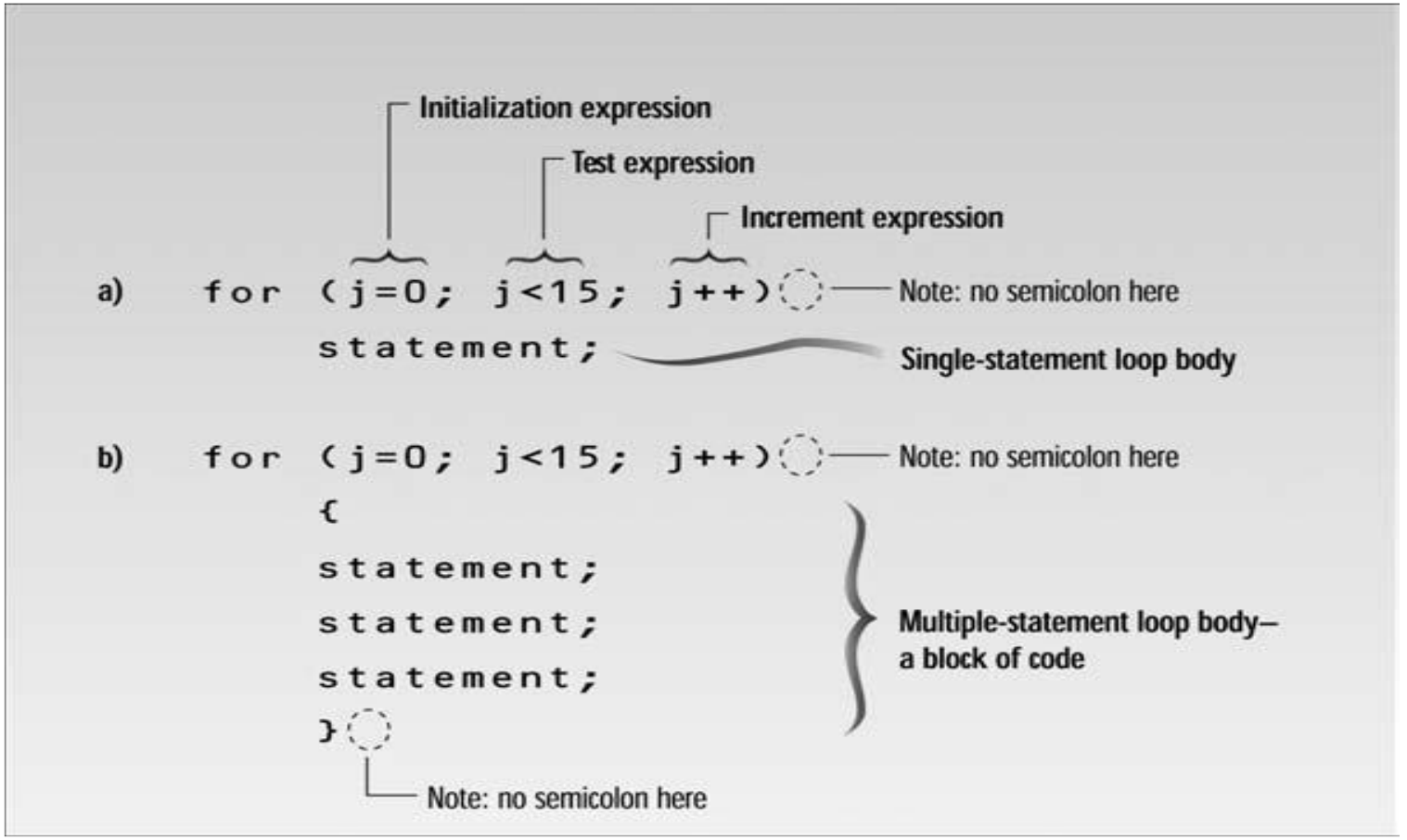
!= Not equal to

>= Greater than or equal to

<= Less than or equal to

- jane = 44; //assignment statement
- harry = 12; //assignment statement
- (jane == harry) //false
- (harry <= 12) //true
- (jane > harry) //true
- (jane >= 44) //true
- (harry != 12) // false
- (7 < harry) //true
- (0) //false (by definition)
- (44) //true (since it's not 0)

- **Loops**
- **The for Loop**



```
#include <iostream>
using namespace std;
int main()
{
int j; //define a loop variable
for(j=0; j<15; j++) //loop from 0 to 14,
cout << j * j << " "; //displaying the square of j
cout << endl;
return 0;
}
```

```
#include <iomanip> //for setw
using namespace std;
int main()
{
int numb; //define loop variable
for(numb=1; numb<=10; numb++) //loop from 1 to 10
{
cout << setw(4) << numb; //display 1st column
int cube = numb*numb*numb; //calculate cube
cout << setw(6) << cube << endl; //display 2nd column
}
return 0;
}
```

- Here’s the output from the program:

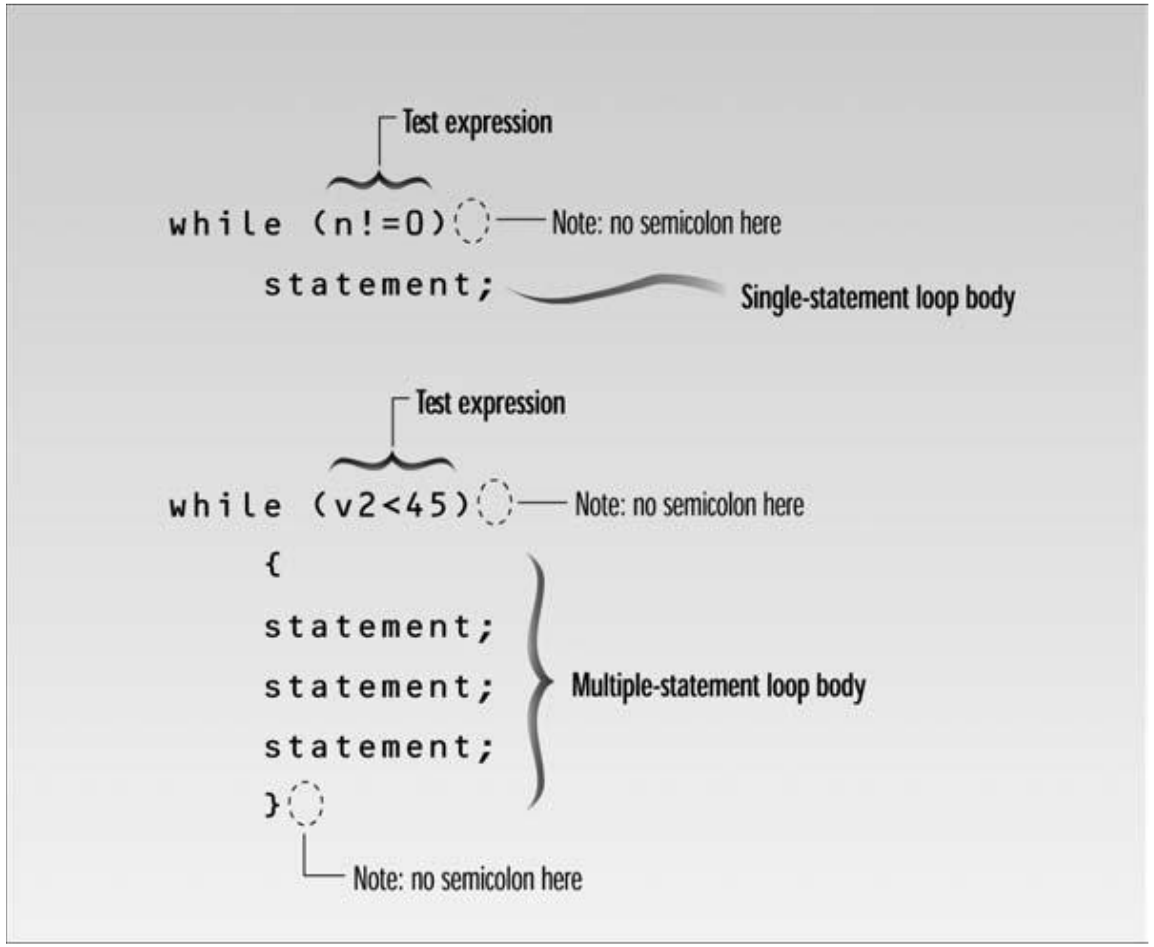
```
1        1
2        8
3       27
4       64
5      125
6      216
7      343
8      512
9      729
10    1000
```

```
#include <iostream>
using namespace std;
int main()
{
    unsigned int numb;
    unsigned long fact=1;       //long for larger numbers
    cout << “Enter a number: “;
    cin >> numb;
    for(int j=numb; j>0; j--)    //multiply 1 by
    fact *= j;                   //numb, numb-1, ..., 2, 1
    cout << “Factorial is “ << fact << endl;
    return 0;
}
```

- **Variables Defined in for Statements**
- for(int j=numb; j>0; j--)
- **Multiple Initialization and Test Expressions**

```
for( j=0, alpha=100; j<50; j++, beta-- )
{
// body of loop
}
```

- **The while Loop**

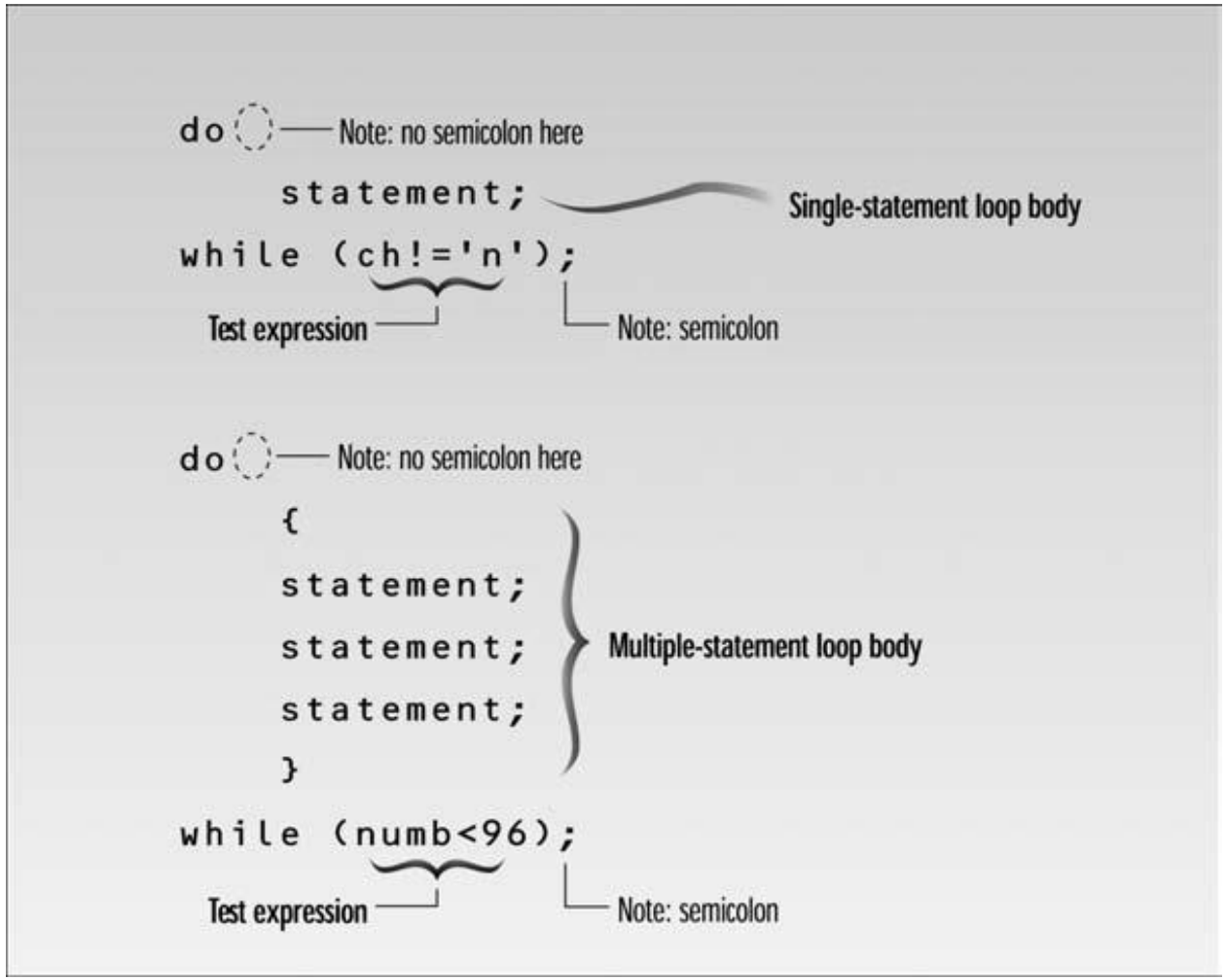


```
#include <iostream>
using namespace std;
int main()
{
int n = 99; // make sure n isn't initialized to 0
while( n != 0 ) // loop until n is 0
cin >> n; // read a number into n
cout << endl;
return 0;
}
```

```
#include <iostream>
#include <iomanip> //for setw
using namespace std;
int main()
{
int pow=1;                              //power initially 1
int numb=1;                             //numb goes from 1 to ???
while( pow<10000 )                      //loop while power <= 4 digits
{
cout << setw(2) << numb;                //display number
cout << setw(5) << pow << endl;         //display fourth power
++numb;                                 //get ready for next power
pow = numb*numb*numb*numb;              //calculate fourth power
}
cout << endl;
return 0;
}
```

```
 1    1
 2   16
 3   81
 4  256
 5  625
 6 1296
 7 2401
 8 4096
 9 6561
```

The do Loop


do
Note: no semicolon here
statement;
Single-statement loop body
while (ch!='n');
Test expression
Note: semicolon
do
Note: no semicolon here
{
statement;
statement;
statement;
}
Multiple-statement loop body
while (numb<96);
Test expression
Note: semicolon

```
#include <iostream>
using namespace std;
int main()
{
long dividend, divisor;
char ch;
do                                                    //start of do loop
{                                                      //do some processing
cout << “Enter dividend: “; cin >> dividend;
cout << “Enter divisor: “; cin >> divisor;
cout << “Quotient is “ << dividend / divisor;
cout << “, remainder is “ << dividend % divisor;
cout << “\nDo another? (y/n): “; //do it again?
cin >> ch;
}
while( ch != ‘n’ );                                   //loop condition
return 0;
}
```

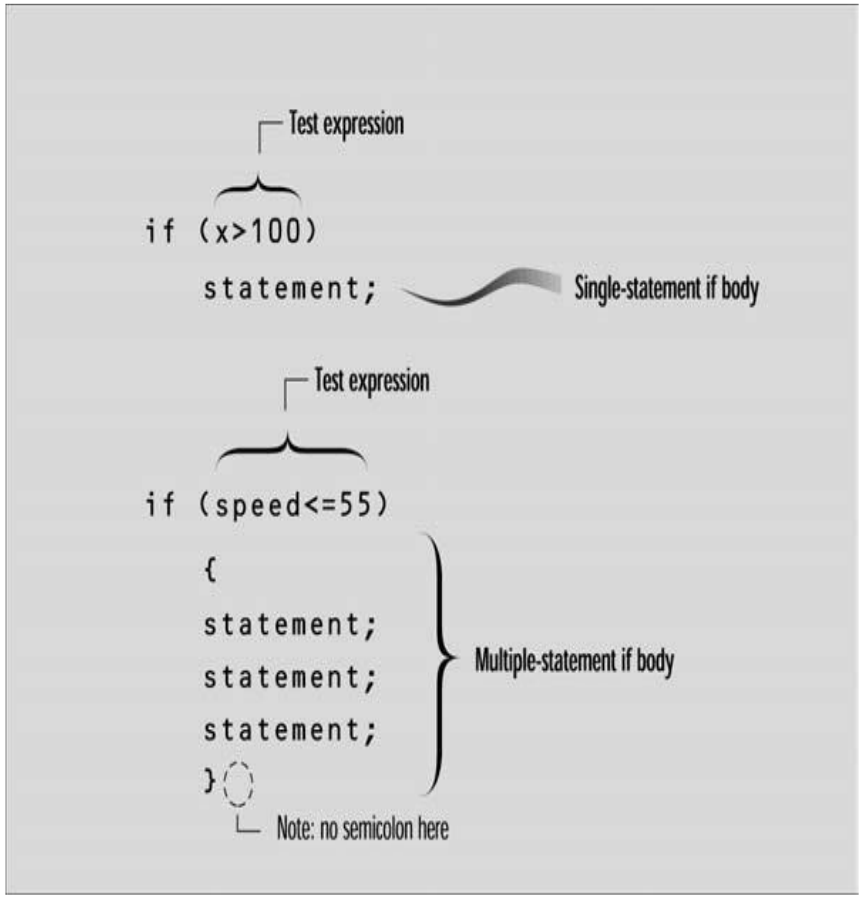
When to Use Which Loop

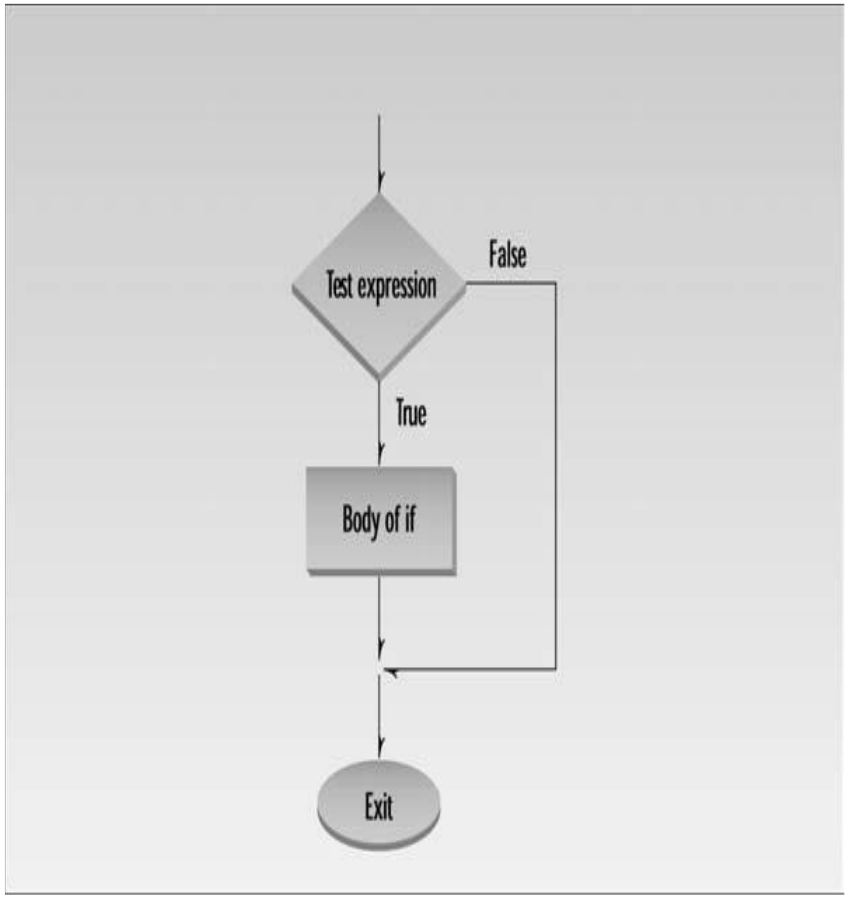
- The for loop is appropriate when you know in advance how many times the loop will be executed.

- The while and do loops are used when you don’t know in advance when the loop will terminate

- The while loop when you may not want to execute the loop body even once, and the do loop when you’re sure you want to execute the loop body at least once.

Decisions
The if Statement





Decisions

The if Statement

```
#include <iostream>
using namespace std;
int main()
{
int x;
cout << “Enter a number: “;
cin >> x;
if( x > 100 )
cout << “That number is greater than 100\n”;
return 0;
}
```

Multiple Statements in the if Body

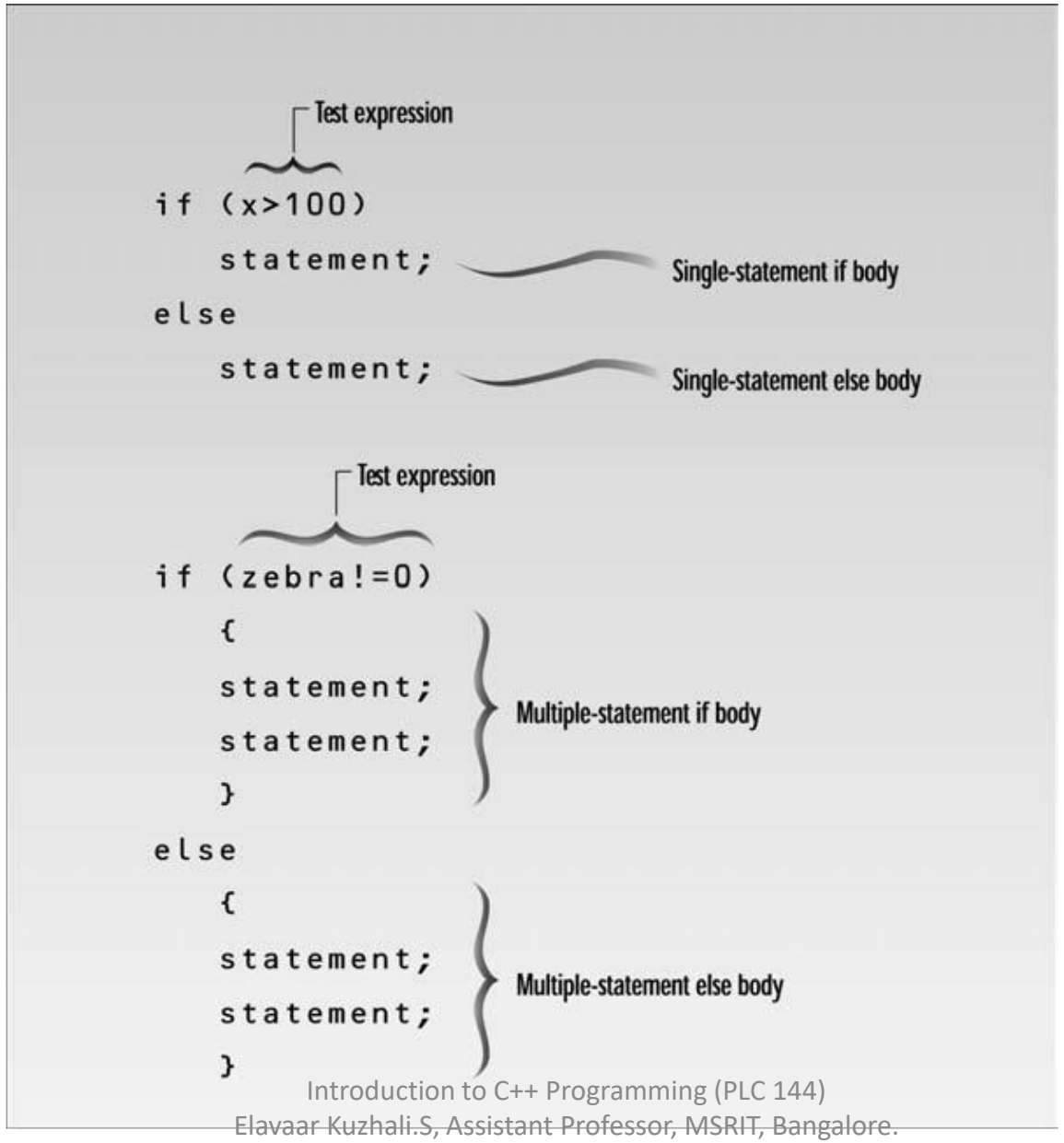
```
#include <iostream>
using namespace std;
int main()
{
int x;
cout << “Enter a number: “;
cin >> x;
if( x > 100 )
{
cout << “The number “ << x;
cout << “ is greater than 100\n”;
}
return 0;
}
```

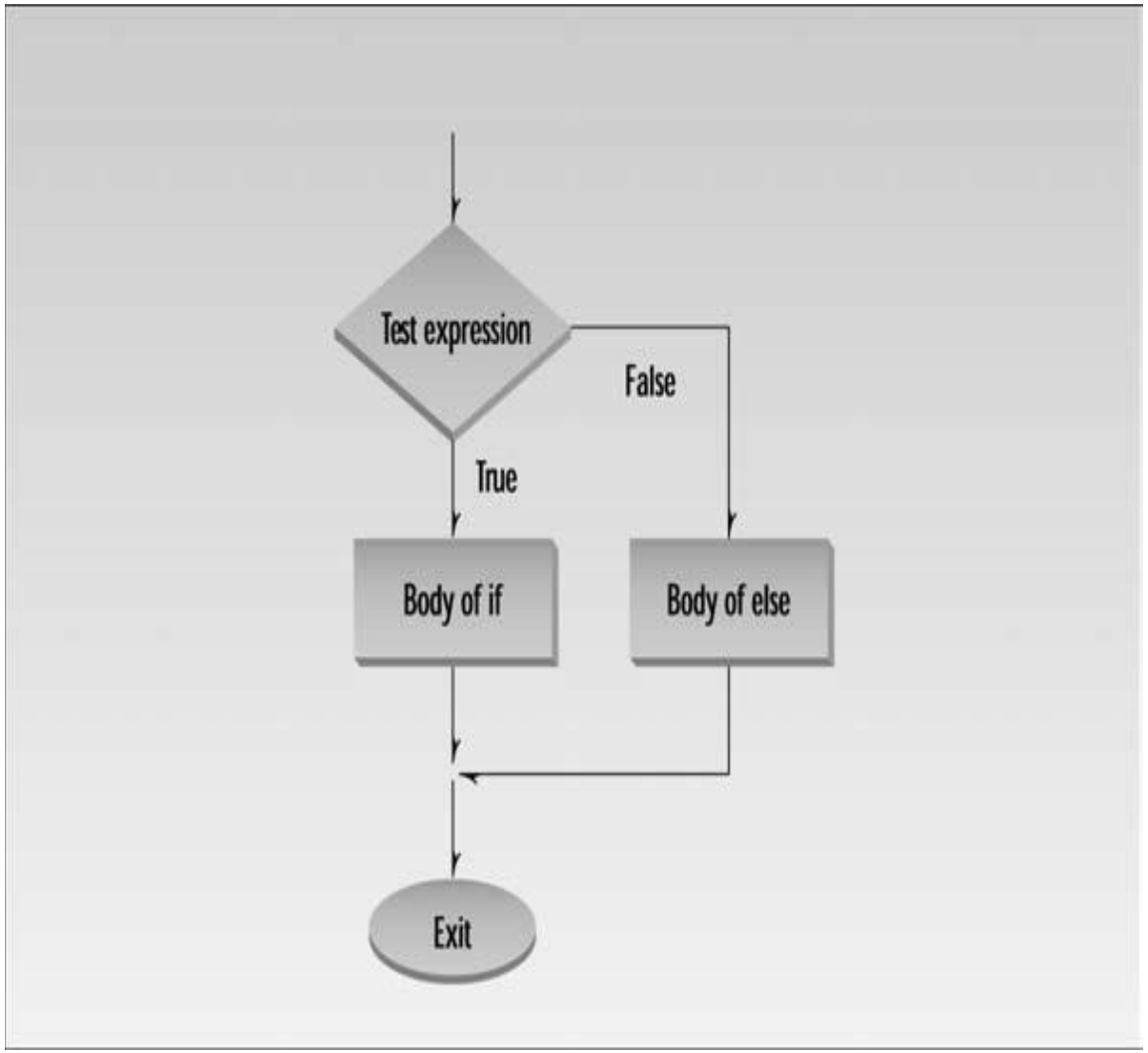
Nesting ifs Inside Loops

- example tells you whether a number you enter is a prime number.
- Prime numbers are integers divisible only by themselves and 1. The first few primes are 2, 3, 5, 7, 11, 13, 17

```
#include <iostream>
using namespace std;
#include <process.h>                          //for exit()
int main()
{
unsigned long n, j;
cout << "Enter a number: ";
cin >> n;                                     //get number to test
for(j=2; j <= n/2; j++)                       //divide by every integer from
if(n%j == 0)                                  //2 on up; if remainder is 0,
{                                             //it's divisible by j
cout << "It's not prime; divisible by " << j << endl;
exit(0);                                      //exit from the program
}
cout << "It's prime\n";
return 0;
}
```

The if...else Statement




Test expression
False
True
Body of if
Body of else
Exit

```
#include <iostream>
using namespace std;
int main()
{
int x;
cout << “\nEnter a number: “;
cin >> x;
if( x > 100 )
cout << “That number is greater than 100\n”;
else
cout << “That number is not greater than 100\n”;
return 0;
}
```

Program to counts the number of words and the number of characters in a phrase typed in by the user.

```
#include <iostream>
using namespace std;
#include<stdlib.h>
int main()
{
int chcount=0;
int wdcount=1;
char ch = 'a';
cout << "Enter a phrase: ";
while( ch != '\r' )
{
ch=getchar();
if( ch==' ' )                    //if it's a space
wdcount++;                       //count a word
else                             //otherwise,
chcount++;                       //count a character
}                                //display results
cout << "\nWords=" << wdcount << endl
<< "Letters=" << (chcount-1) << endl;
return 0;
}
```

Nested if...else Statements

```
int main()
{
char dir='a';
int x=10, y=10;
cout << "Type Enter to quit\n";
while( dir != '\r' )
 {
cout << "\nYour location is " << x << ", " << y;
cout << "\nPress direction key (n, s, e, w): ";
dir=getchar();
 if( dir=='n')
y--;
else if( dir=='s' )
y++;
else if( dir=='e' )
x++;
else if( dir=='w' )
x--;
} //end while
return 0;
} //end main
```

Matching the else

```
#include <iostream>
using namespace std;
int main()
{
int a, b, c;
cout << “Enter three numbers, a, b, and c:\n”;
cin >> a >> b >> c;
if( a==b )
        if( b==c )
        cout << “a, b, and c are the same\n”;
else
        cout << “a and b are different\n”;
return 0;
}
```

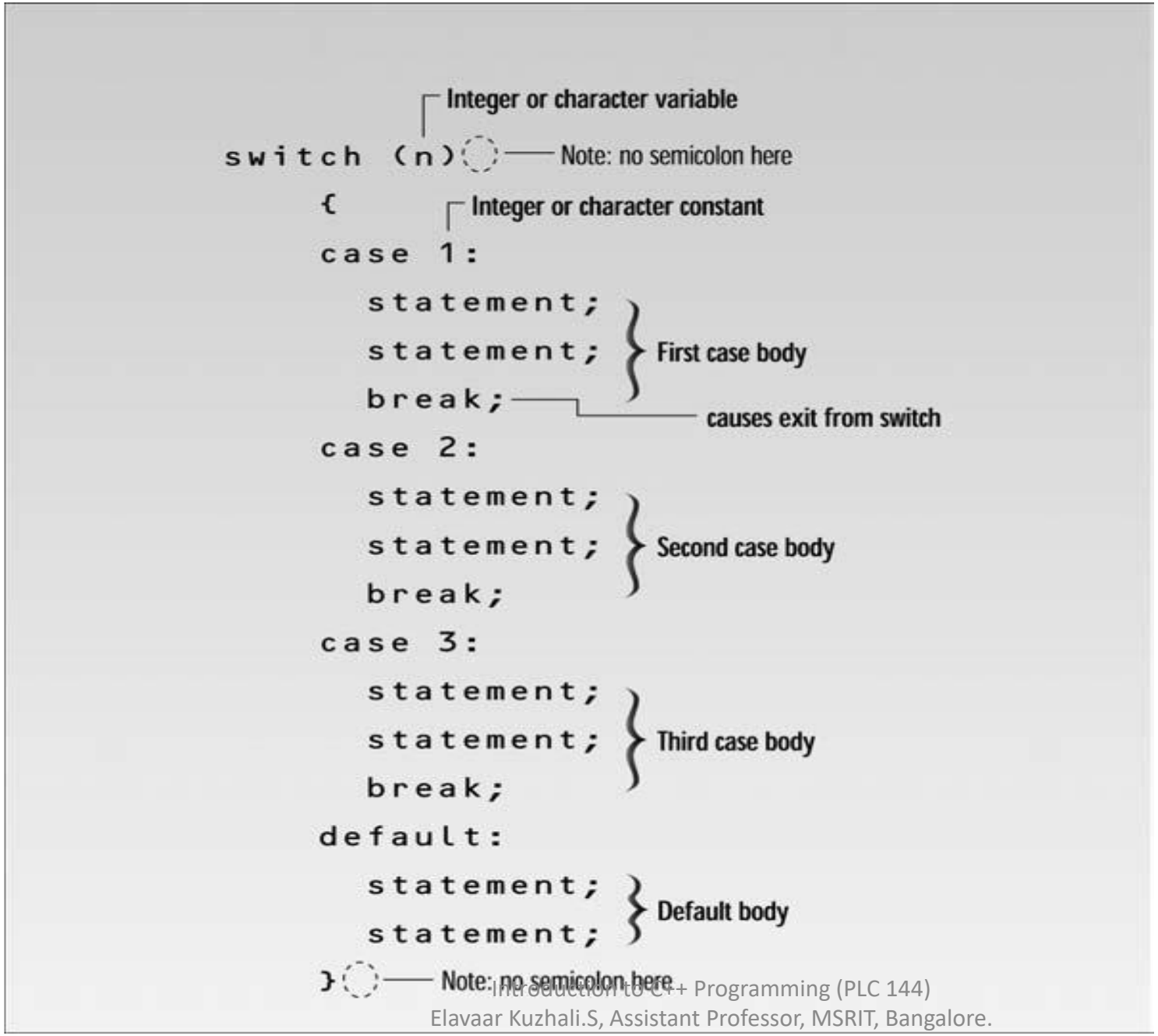
corrected version:

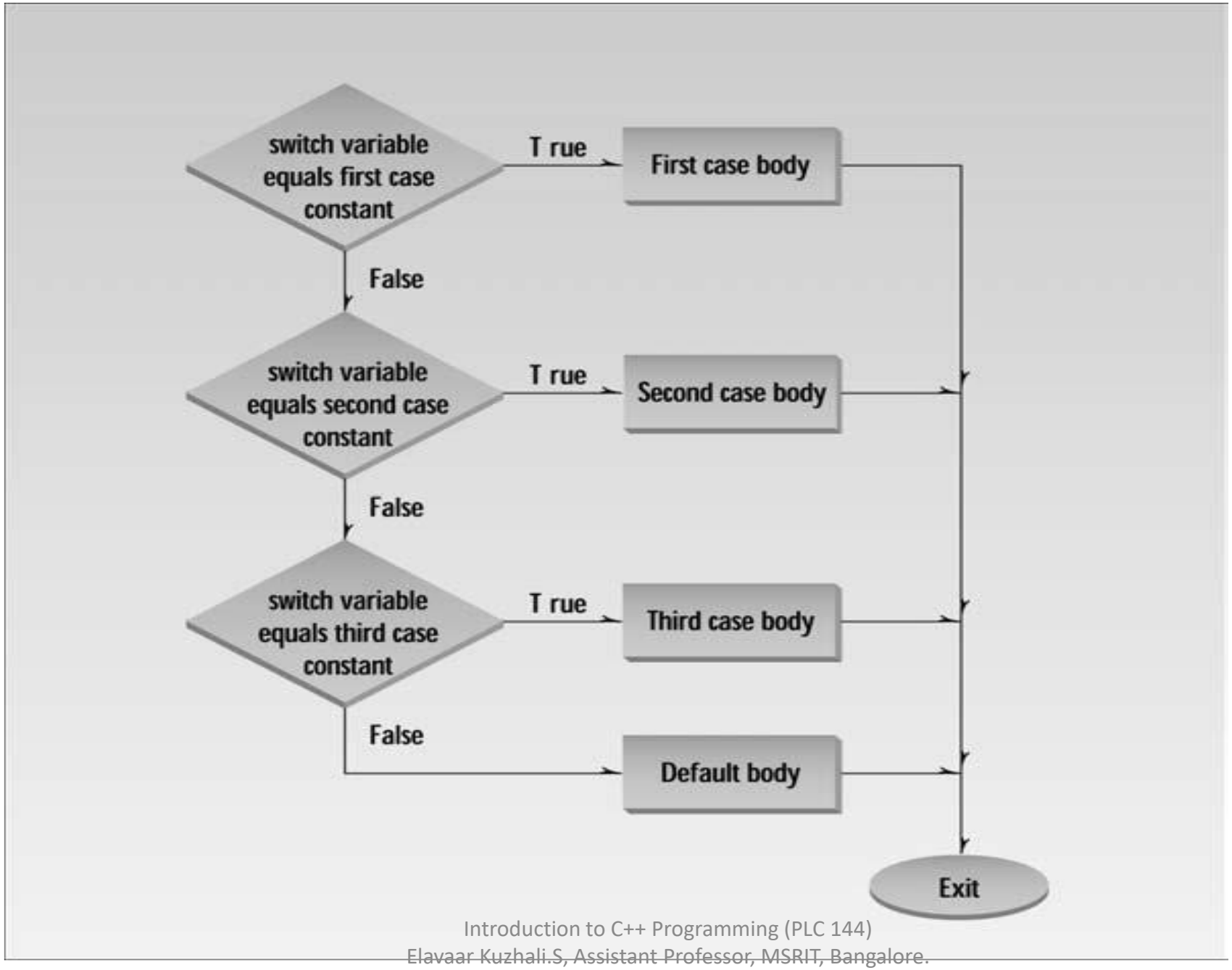
```
if(a==b)
        if(b==c)
        cout << “a, b, and c are
        the same\n”;
        else
        cout << “b and c are
        different\n”;
```

pair an else with an earlier if

```
if(a==b)
{
if(b==c)
cout << “a, b, and c are the
same”;
}
else
cout << “a and b are different”;
```

- SWITCH Statement




switch variable equals first case constant
T rue
First case body
False
switch variable equals second case constant
T rue
Second case body
False
switch variable equals third case constant
T rue
Third case body
False
Default body
Exit

The switch Statement

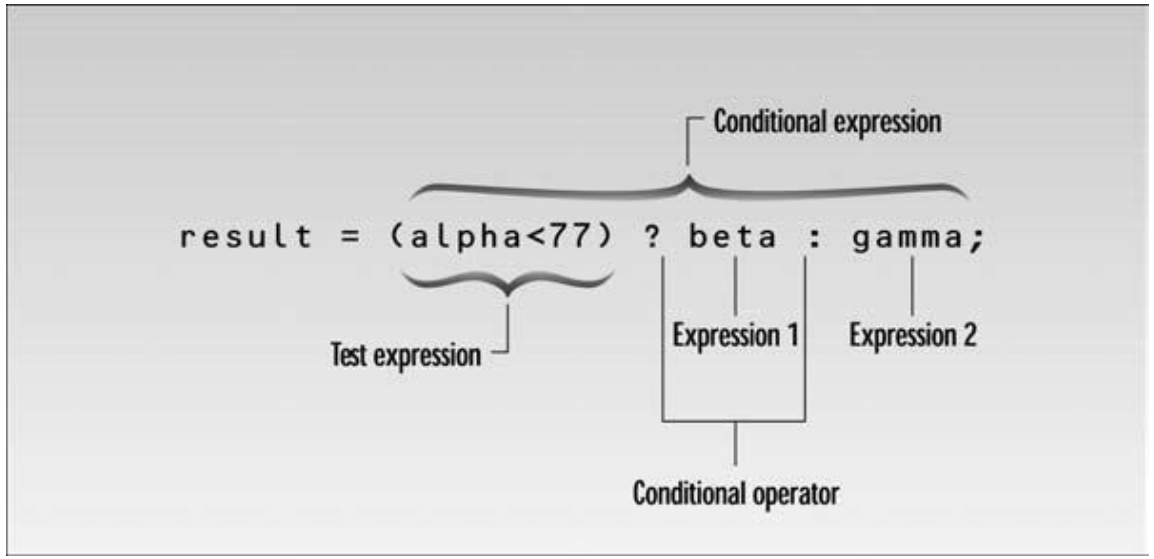
```
#include <iostream>
using namespace std;
int main()
{
	int speed; //turntable speed
	cout << "\nEnter 33, 45, or 78: ";
	cin >> speed; //user enters speed
	switch(speed) //selection based on speed
	{
		case 33:
			cout << "LP album\n";
			break;
		case 45:
			cout << "Single selection\n";
			break;
		case 78:
			cout << "Obsolete format\n";
			break;
	}
return 0;
}
```

```
int main()
{
char dir='a';
int x=10, y=10;
while( dir != '\r' )
{
cout << "\nYour location is " << x << ", " << y;
cout << "\nEnter direction (n, s, e, w): ";
cin>>dir;
switch(dir) //switch on it
{
      case 'n': y--; break;
      case 's': y++; break;
      case 'e': x++; break;
      case 'w': x--; break;
      case '\r': cout << "Exiting\n"; break;
      default: cout << "Try again\n";
}
      } //end while
return 0;
} //end main
```

The Conditional Operator

```
if( alpha < beta )
        min = alpha;
else
        min = beta;
```

```
min = (alpha<beta) ? alpha : beta;
```



Logical Operators

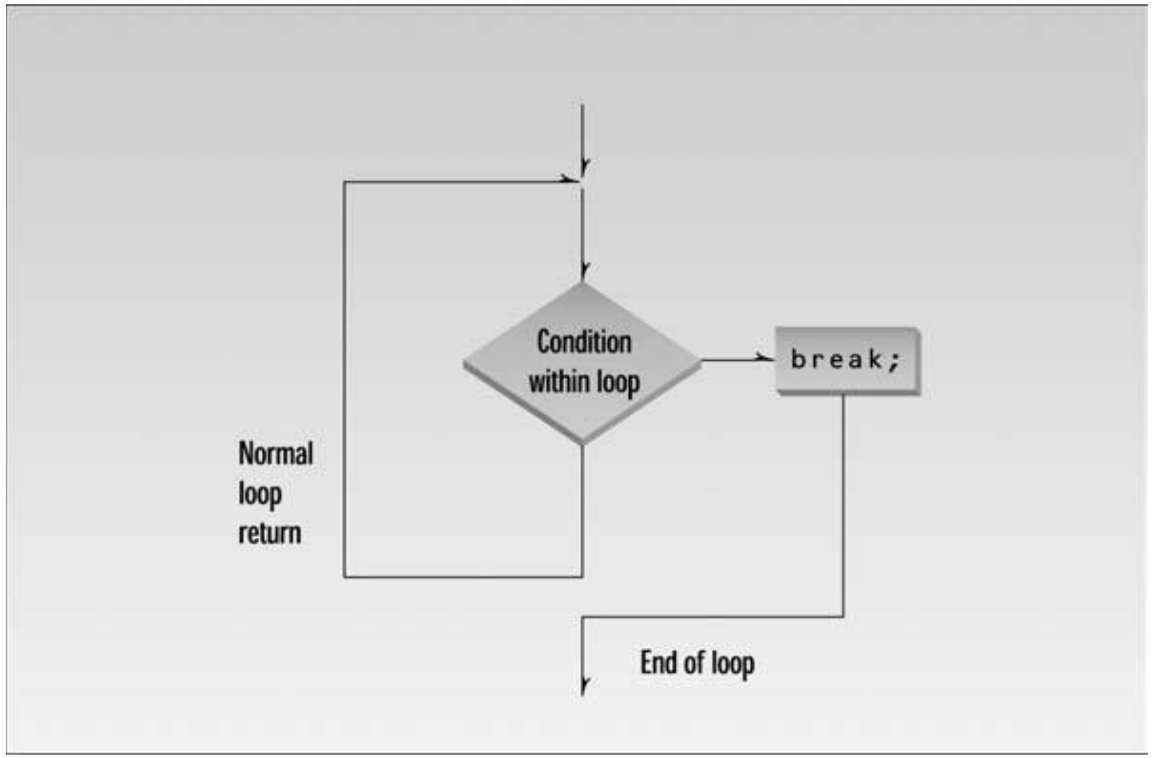
logically combine Boolean variables

Operator	*Effect*
&&	Logical AND
||	Logical OR
!	Logical NOT

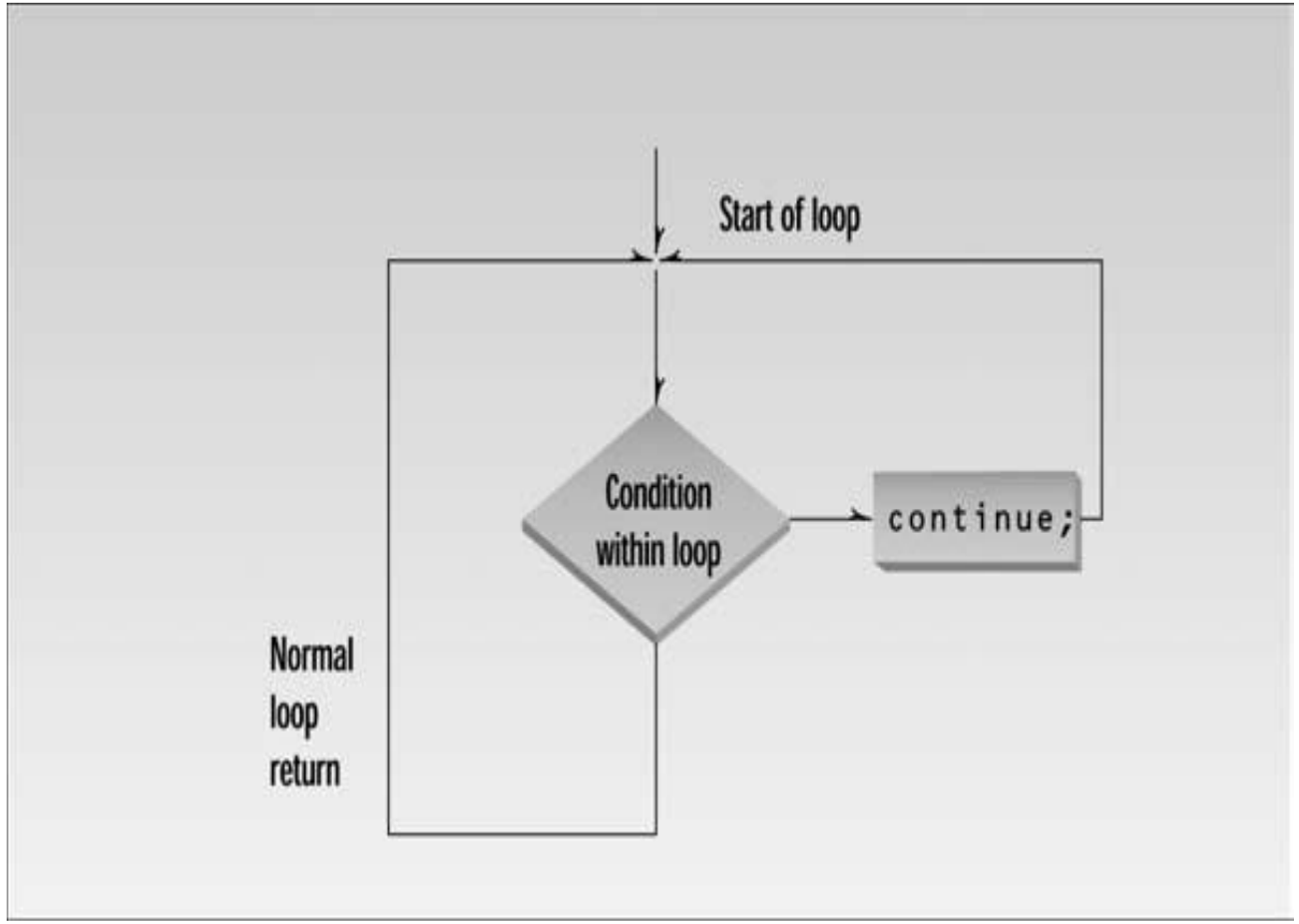
```
if( x<5 || x>15 ) //if x west of 5 OR east of 15
        cout << "\nBeware: dragons lurk here";

if( x==7 && y==11 ) //if x is 7 and y is 11
{
    cout << "\nYou found the treasure!\n";
    exit(0); //exit from program
}
```

!(x==7) is true if x is not equal to 7

The break Statement: The break statement causes an exit from a loop



The continue Statement: go back to the top of the loop



```
#include <iostream>
using namespace std;
int main()
{
        long dividend, divisor;
        char ch;
                do {
                cout << “Enter dividend: “; cin >> dividend;
                cout << “Enter divisor: “; cin >> divisor;
                if( divisor == 0 )
                {
                                cout << “Illegal divisor\n”;
                                break;
                }
                cout << “Quotient is “ << << dividend / divisor;
                cout << “, remainder is “ << dividend % divisor;
                cout << “\nDo another? (y/n): “;
                cin >> ch;
                } while( ch != ‘n’ );
        return 0;
}
```

QA

1) A relational operator

a. assigns one operand to another.

b. **yields a Boolean result.**

c. **compares two operands.**

d. logically combines two operands.

2) Write an expression that uses a relational operator to return true if the variable george is not equal to sally.

3) In a for loop with a multistatement loop body, semicolons should appear following

a. the for statement itself.

b. the closing brace in a multistatement loop body.

c. **each statement within the loop body.**

d. **the test expression.**

4) The && and || operators

a. compare two numeric values.

b. combine two numeric values.

c. compare two Boolean values.

d. **combine two Boolean values.**

5) Assume that you want to generate a table of multiples of any given number. Write a program that allows the user to enter the number and then generates the table, formatting it into 10 columns and 20 lines. Interaction with the program should look like this (only the first three lines are shown):

Enter a number: 7

7 14 21 28 35 42 49 56 63 70

77 84 91 98 105 112 119 126 133 140

147 154 161 168 175 182 189 196 203 210

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    unsigned long n; //number
    cout << "\nEnter a number: ";
    cin >> n;
    for(int j=1; j<=200; j++)
    {
        cout << setw(5) << j*n << " ";
        if( j%10 == 0 )
        cout << endl;
    }
    return 0;
}
```

6) Write a temperature-conversion program that gives the user the option of converting Fahrenheit to Celsius or Celsius to Fahrenheit. Then carry out the conversion. Use floating-point numbers. Interaction with the program might look like this:

Type 1 to convert Fahrenheit to Celsius,

2 to convert Celsius to Fahrenheit: 1

Enter temperature in Fahrenheit: 70

In Celsius that's 21.111111

7) Create the equivalent of a four-function calculator. The program should ask the user to enter a number, an operator, and another number. (Use floating point.) It should then carry out the specified arithmetical operation: adding, subtracting, multiplying, or dividing the two numbers. Use a switch statement to select the operation. Finally, display the result.

When it finishes the calculation, the program should ask whether the user wants to do another calculation. The response can be 'y' or 'n'. Some sample interaction with the program might look like this:

Enter first number, operator, second number: 10 / 3

Answer = 3.333333

Do another (y/n)? y

Enter first number, operator, second number: 12 + 100

Answer = 112

Do another (y/n)? n

```
#include <iostream>
using namespace std;
int main()
{
int response;
double temper;
cout << “\nType 1 to convert fahrenheit to celsius,”
<< “\n 2 to convert celsius to fahrenheit: “;
cin >> response;
if( response == 1 )
{
cout << “Enter temperature in fahrenheit: “;
cin >> temper;
cout << “In celsius that’s “ << 5.0/9.0*(temper-32.0);
}
else
{
cout << “Enter temperature in celsius: “;
cin >> temper;
cout << “In fahrenheit that’s “ << 9.0/5.0*temper + 32.0;
}
cout << endl;
return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
double n1, n2, ans;
char oper, ch;
do {
cout << "\nEnter first number, operator, second number: ";
cin >> n1 >> oper >> n2;
switch(oper)
{
case '+': ans = n1 + n2; break;
case '-': ans = n1 - n2; break;
case '*': ans = n1 * n2; break;
case '/': ans = n1 / n2; break;
default: ans = 0;
}
cout << "Answer = " << ans;
cout << "\nDo another (Enter 'y' or 'n')? ";
cin >> ch;
} while( ch != 'n' );
return 0;
}
```

8) WAP to display the following pyramid (use loops)

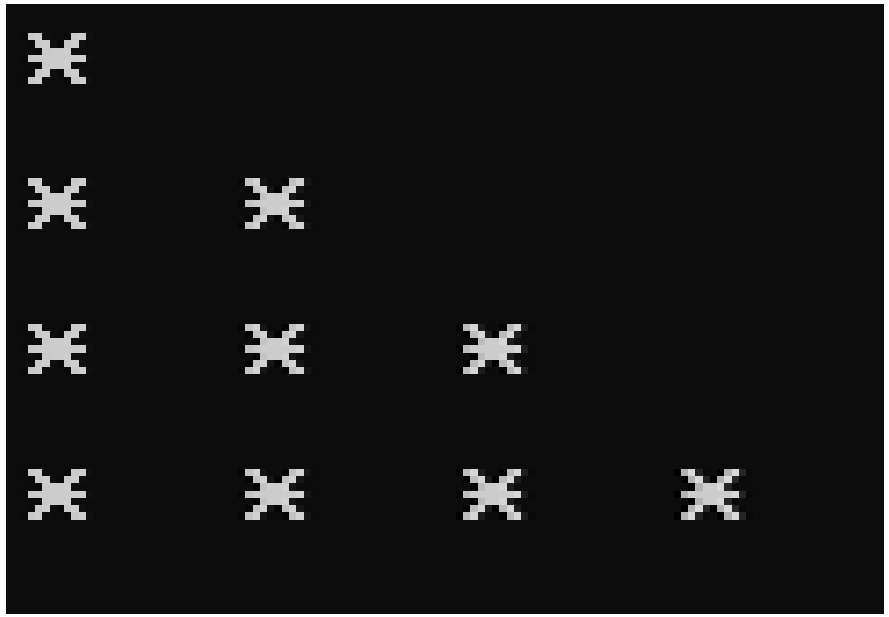
```
1
1   2
1   2   3
1   2   3   4
```

```
1
2       2
3       3       3
4       4       4       4
```

```
    1
  1 2 3
1 2 3 4 5
```

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

```
#include <iostream>
using namespace std;
int main()
{
    int i,j;

    for (i = 0;i < 4;i++)
    {
    for(j = 0;j <= i; j++)
    {
    cout << " * " ;
    }
    cout << endl;
    }
    return 0;
}
```

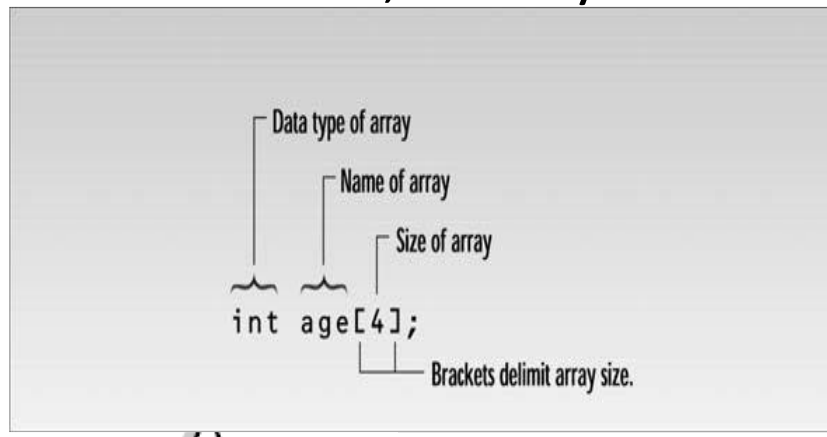


8) Create a four-function calculator for fractions. Here are the formulas for the four arithmetic operations applied to fractions:

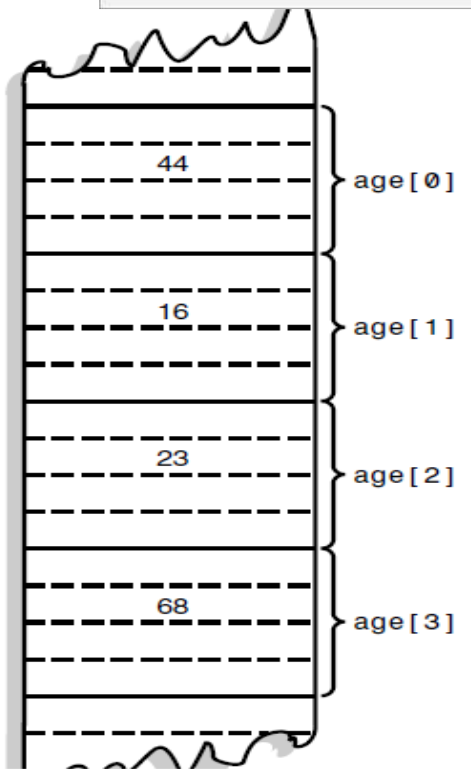
- Addition: a/b + c/d = (a*d + b*c) / (b*d)
- Subtraction: a/b - c/d = (a*d - b*c) / (b*d)
- Multiplication: a/b * c/d = (a*c) / (b*d)
- Division: a/b / c/d = (a*d) / (b*c)
- The user should type the first fraction, an operator, and a second fraction. The program should then display the result and ask whether the user wants to continue.

Arrays hold a few data items or tens of thousands.

The data items grouped in an array can be simple types such as int or float, or they can be user-defined types such as structures and objects.



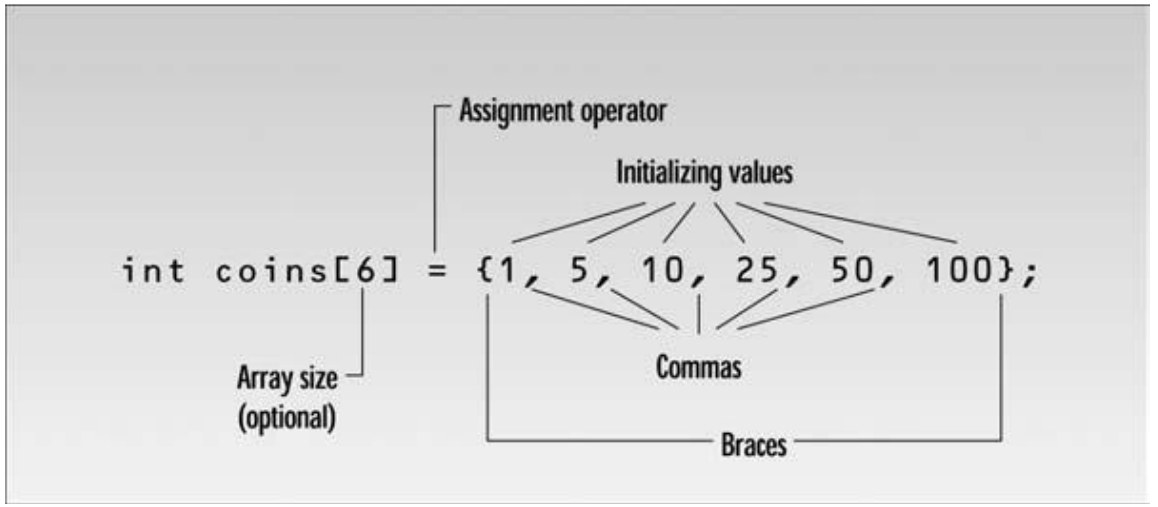

```
#include <iostream>
using namespace std;
int main()
{
int age[4];
for(int j=0; j<4; j++)     //get 4 ages
{
cout << “Enter an age: “;
cin >> age[j];              //access array element
}
for(j=0; j<4; j++)                    //display 4 ages
cout << “You entered “ << age[j] << endl;
return 0;
}
```



Averaging Array Elements

```
#include <iostream>
using namespace std;
int main()
{
const int SIZE = 6;                          //size of array
double sales[SIZE];                          //array of 6 variables
cout << “Enter widget sales for 6 days\n”;
for(int j=0; j<SIZE; j++)                    //put figures in array
cin >> sales[j];
double total = 0;
for(j=0; j<SIZE; j++)                        //read figures from array
total += sales[j];                           //to find total
double average = total / SIZE;               // find average
cout << “Average = “ << average << endl;
return 0;
}
```

- **Initializing Arrays**
- int days_per_month[12] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };



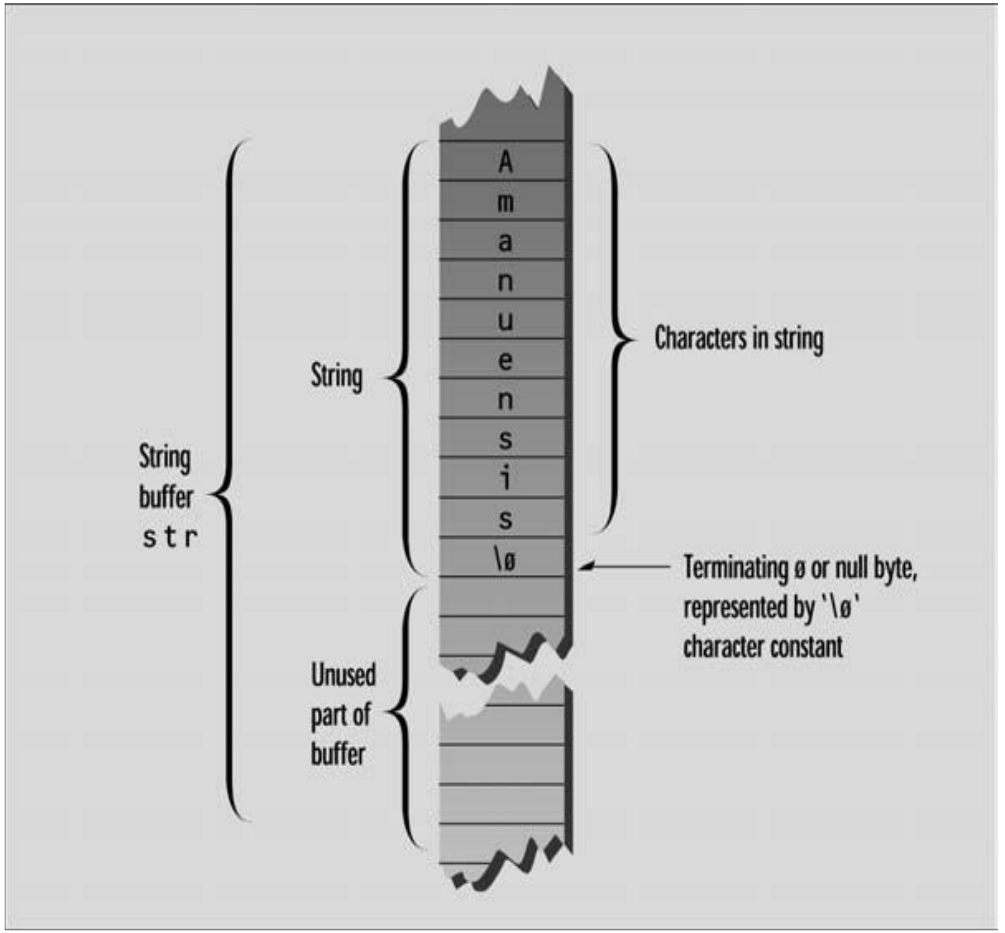
```
#include <iostream>

using namespace std;
#define MAX 4
int main()
{
   cout << "Happy Programming!" << endl;
   int arr[MAX],Key;
   cout << "Enter the elements of the array:"<<endl;
   for(int i=0;i<MAX;i++)
   {
      cin>>arr[i];
   }
   cout << "Enter the element to find:"<<endl;
   cin>>Key;
   for(int i=0;i<MAX;i++)
     {
        if(arr[i]==Key)
        {
           cout<<"Number is found at location "<<i+1<<end
           exit(0);
        }

     }
   cout<<"Number not found in array "<<endl;
   return 0;
}
```

C-Strings are arrays of type char.

They may also be called char* strings, because they can be represented as pointers to type char.

C-String Variables strings can be variables or constants

```
int main()
{
const int MAX = 80;
char str[MAX];
cout << “Enter a string: “;
cin >> str;
cout << “You entered: “ << str << endl;
return 0;
}
```



String Constants

```
#include <iostream>
using namespace std;
int main()
{
char str[] = “Farewell! thou art too dear for my possessing.”;
cout << str << endl;
return 0;
}
```

Copying a String the Hard Way

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
char str1[] = “Hello World”;
const int MAX = 80;
char str2[MAX];
for(int j=0; j<strlen(str1); j++)
str2[j] = str1[j];
str2[j] = ‘\0’;
cout << str2 << endl;
return 0;
}
```

Copying a String the Easy Way

```
int main()
{
char str1[] = “Hello World”;
const int MAX = 80;
char str2[MAX];
strcpy(str2, str1);
cout << str2 << endl;
return 0;
}
```

- Write a program that reverses a C-string (an array of char). Use a for loop that swaps the first and last characters, then the second and next-to-last characters, and so on. Print out the result.

- Start with a program that allows the user to input a number of integers, and then stores them in an int array. Write a program that goes through the array, element by element, looking for the largest one. display the largest element and its index number.

- **Console programs**
- Console programs are programs that use text to communicate with the user and the environment, such as printing text to the screen or reading input from a keyboard.

  Console programs are easy to interact with, and generally have a predictable behavior that is identical across all platforms. They are also simple to implement and thus are very useful to learn the basics of a programming language: The examples in these tutorials are all console programs.

- The easiest way for beginners to compile C++ programs is by using an Integrated Development Environment (IDE). An IDE generally integrates several development tools, including a text editor and tools to compile programs directly from it.

  Here you have instructions on how to compile and run console programs using different free Integrated Development Interfaces (IDEs):

-

IDE	Platform	Console programs
Code::blocks	Windows/Linux/MacOS	Compile console programs using Code::blocks
Visual Studio Express	Windows	Compile console programs using VS Express 2013
Dev-C++	Windows	Compile console programs using Dev-C++

Compiler	Platform	Command
GCC	Linux, among others...	g++ -std=c++0x example.cpp -o example_program
Clang	OS X, among others...	clang++ -std=c++11 -stdlib=libc++ example.cpp -o example_program

- **GCC** is portable and run in many operating platforms. **GCC** (and **GNU** Toolchain) is currently available on all Unix's. They are also ported to **Windows** (by Cygwin, MinGW and MinGW-W64). **GCC** is also a cross-**compiler**, for producing executables on different platform.