# Basic Concepts of Object Oriented Programming

- Objects
- Classes
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing

# Objects

- Objects are the basic run time entities in an object-oriented system.
- They may represent
  - a person, a place, a bank account, a table of data or any item that the program has to handle.
  - user-defined data such as vectors, time and lists.
- An object contains both data and methods that manipulate that data
- Objects take up space in the memory and have an associated address
- objects interact by sending messages to one another

# Classes

- Objects are variables of the type class.
- Once a class has been defined, we can create any number of objects belonging to that class.
- Each object is associated with the data of type class with which they are created.
- A class is thus a collection of objects similar types.
- Classes are user-defined that types and behave like the built-in types of a programming language.
- The syntax used to create an object is not different then the syntax used to create an integer object in C
- Every object belongs to (is an instance of) a class
- An object may have fields, or variables and methods
  - The class describes those fields and methods
- A class is like a template, or cookie cutter
- Classes are like Abstract Data Types

# Data Abstraction and Encapsulation

- The wrapping up of data and function into a single unit (called class) is known as encapsulation.
- Data and encapsulation is the most striking feature of a class.
- The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it.
- These functions provide the interface between the object's data and the program.
- This insulation of the data from direct access by the program is called data hiding or information hiding.
- Abstraction refers to the act of representing essential features without including the background details or explanation.
- Classes use the concept of abstraction
- They encapsulate all the essential properties of the object that are to be created.

# Inheritance

- Inheritance is the process by which objects of one class acquired the properties of objects of another classes.
- It supports the concept of hierarchical classification. For example, the bird, ‘robin’ is a part of class ‘flying bird’ which is again a part of the class ‘bird’. The principal behind this sort of division is that each derived class shares common characteristics with the class from which it is derived as illustrated in fig 1.6.
- the concept of inheritance provides the idea of reusability
- we can add additional features to an existing class without modifying it.
- The new class will have the combined feature of both the classes.
- The real appeal and power of the inheritance mechanism is that it allows us to reuse a class

# Polymorphism

- Polymorphism, a Greek term, means the ability to take more than on form.
- An operation may exhibit different behavior is different instances
- The behavior depends upon the types of data used in the operation
- For example, consider the operation of addition. For
- The process of making an operator to exhibit different behaviors in different instances is known as operator overloading.
  - two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation.
- This is something similar to a particular word having several different meanings depending upon the context.
- Using a single function name to perform different type of task is known as function overloading
- Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface
- This means that a general class of operations may be accessed in the same manner even though specific action associated with each operation may differ.

# Dynamic Binding

- Binding refers to the linking of a procedure call to the code to be executed in response to the call.
- Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time.
- It is associated with polymorphism and inheritance.
- A function call associated with a polymorphic reference depends on the dynamic type of that reference.
- At run-time, the code matching the object under current reference will be called

# Message Passing

- Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another.
- The concept of message passing makes it easier to talk about building systems that directly model or simulate their real-world counterparts.
- A Message for an object is a request for execution of a procedure, and therefore will invoke a function (procedure) in the receiving object that generates the desired results.
- *Message passing involves specifying the name of object, the name of the function (message) and the information to be sent*
- Communication with an object is feasible as long as it is alive

# Benefits of OOP

- Through inheritance, we can eliminate redundant code extend the use of existing Classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure program that can not be invaded by code in other parts of a programs.
- It is possible to have multiple instances of an object to co-exist without any interference.
- It is possible to map object in the problem domain to those in the program.
- It is easy to partition the work in a project based on objects.
- Object-oriented system can be easily upgraded from small to large system.
- Message passing techniques for communication between objects makes to interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

# Object Oriented Language

Depending upon the features they support, they can be classified into the following two categories:

- Object-based programming languages
  - Data encapsulation
  - Data hiding and access mechanisms
  - Automatic initialization and clear-up of objects
  - Operator overloading
- Object-oriented programming languages
  - Object-based features + inheritance + dynamic binding

# Characteristics of Object Oriented Languages

| *Characteristics* | *Simula* * | *Smalltalk* * | *Objective C* | *C++* | *Ada* ** | *Object Pascal* | *Turbo Pascal* | *Eiffel* * | *Java* * |
|---|---|---|---|---|---|---|---|---|---|
| Binding (early or late) | Both ✓ | Late ✓ | Both ✓ | Both ✓ | Early ✓ | Late ✓ | Early ✓ | Early ✓ | Both ✓ |
| Polymorphism | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Data hiding | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Concurrency | ✓ | Poor | Poor | Poor | Difficult | No | No | Promised | ✓ |
| Inheritance | ✓ | ✓ | ✓ | ✓ | No | ✓ | ✓ | ✓ | ✓ |
| Multiple Inheritance | No | ✓ | ✓ | ✓ | No | --- | --- | ✓ | No |
| Garbage Collection | ✓ | ✓ | ✓ | ✓ | No | ✓ | ✓ | ✓ | ✓ |
| Persistence | No | Promised | No | No | like 3GL | No | No | Some Support | ✓ |
| Genericity | No | No | No | ✓ | ✓ | No | No | ✓ | No |
| Object Libraries | ✓ | ✓ | ✓ | ✓ | Not much | ✓ | ✓ | ✓ | ✓ |

* *Pure object-oriented languages*
** *Object-based languages*
*Others are extended conventional languages*

# Let us start C++

# What Is C++?

- C++ is an object-oriented programming language.
- It was developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early 1980's.
- Therefore, C++ is an extension of C, initially called the new language 'C with classes'
- Later in 1983, the name was changed to C++
- C+ + is a superset of C
- Almost all c programs are also C++ programs. However, there are a few minor differences that will prevent a c program to run under C++ complier

# What Is C++?.. contd

- (Mostly) an extension of *C* to include:–
  - Classes
  - Templates
  - Inheritance and Multiple Inheritance
  - Function and Operator Overloading
  - New (and better) Standard Library
  - References and Reference Parameters
  - Default Arguments
  - Inline Functions
  - …
- A few small syntactic differences from *C*

# Comments

- A new comment symbol // (double slash).
- A comment may start anywhere in the line, and whatever follows till the end of the line is ignored.
- The double slash comment is basically a single line comment.
- // This is an example of

# Input/Output Operators

**Input Operator**

- cin >> number1;
- Identifier cin - a predefined object in C++ that corresponds to the standard input stream. (i.e. Keyboard)
- The operator >> is known as extraction or get from operator.

**Output Operator**

- Cout<<"C++ is better than C.";
- The identifier cout(pronounced as C out) is a predefined object that represents the standard output stream in C++. (i.e. screen)
- The operator << is called the insertion or put to operator.

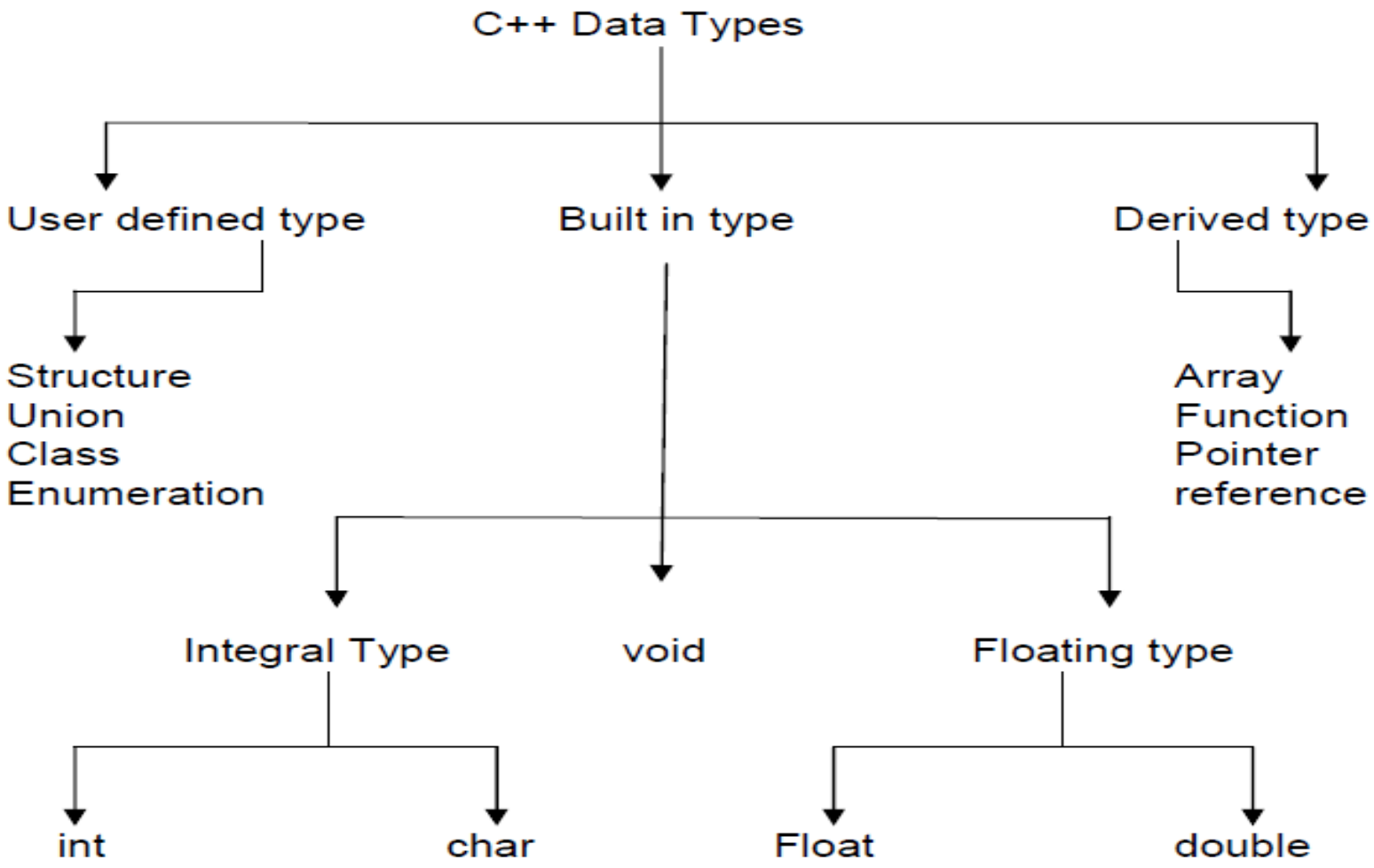
**The iostream File**

- **#include <iostream> or #include <iostream.h>**
- The header file iostream.h should be included at the beginning of all programs that use input/output statements.
- Use <iostream.h> if system doesn't support ANSI C++ features

# Return Type of main()

- In C++, main () returns an integer value to the operating system.
- Every main () in C++ should end with a return (0) statement; otherwise a warning or an error might occur.
- the default return type for all function in C++ is int. The main without type and return will run with a warning.

# BASIC DATA TYPES IN C++



# Data Types

| Name | Description | Size* | Range* |
|---|---|---|---|
| Char | Character or small integer | 1byte | signed: -128 t127<br>unsigned: 0 to 255 |
| short int (short) | Short Integer. | 2bytes | signed: -32768 to 32767<br>unsigned: 0 to65535 |
| Int | Integer. | 4bytes | signed: - 2147483648 to 2147483647<br>unsigned: 0 to 4294967295 |
| long int (long) | Long integer. | 4bytes | signed: -2147483648 to 2147483647<br>unsigned: 0 to 4294967295 |
| Bool | Boolean value. It can take one of two values: true or false. | 1byte | true or false |
| Float | Floating point number. | 4bytes | 3.4e +/- 38 (7 digits) |
| double | Double precision floating point number. | 8bytes | 1.7e +/- 308 (15 digits) |
| long double | Long double precision floating point number. | 8bytes | 1.7e +/- 308 (15 digits) |

# Identifier

- Alphabets, digits, underscore
- Name can’t start with digit
- Upper and lower cases are different
- Keywords can’t be used
- No limit on length

# 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 | |

# Reference variables

A reference variable provides an alias for a previously defined variable.

Ex int sum = 200;

int & total = sum;

- total is the alternative name declared to represent the variable sum.
- Both variable refer to the same data object in memory.
- A reference variable must be initialized at the time of declaration.
- C++ assigns additional meaning to the symbol .&..Here & is not an address operation. The notation int & means reference to int.
- A major application of reference variable is in passing arguments to functions.

# Contd.

**Ex**

```
void fun (int &x) // uses reference
{    x = x + 10;      // x increment, so n also incremented.
}

int main( )
{int n = 10;
fun(n);       // function call
}
```

- When the function call fun(n) is executed, it will assign x to n  i.e. int &x = n;
- Therefore x and n are aliases & when function increments x. n is also incremented. This type of function call is called call by reference.

```
int main()
{
  int x = 10;

  // ref is a reference to x.
  int & ref = x;

  // Value of x is now changed to 20
  ref = 20;
  cout << "x = " << x << endl ;    // 20

  // Value of x is now changed to 30
  x = 30;
  cout << "ref = " << ref << endl ;  // 30

  return 0;
}
```

# Some of the new operators in c++

1. :: Scope resolution operators.
2. ::* To access a pointer to a member of a class
3. → * To access a member using a pointer in the object & a pointer to the member
4. .* To access a member using object name & a pointer to that member
5. delete memory release operator.
6. endl Line feed operator (\n)
7. new Memory allocation operator. (malloc)
8. setw Field width operator.

# Scope resolution operator

- the global version of a variable cannot be accessed from within the inner block.
- C++ resolves this problem by using scope resolution operator (::), because this operator allows access to the global version of a variable

# Contd.

```
# include<iostream.h >
int m = 10;
int main ( )
{int m = 20;
cout << “m =“ << m;   // 20
cout << “Global M =“ <<::m;  //10
return 0;
}
```

# Setw : -

- With the stew, we can specify a common field width for all
- the numbers and force them to print with right alignment.

**EX**

- **cout<<setw (5) <<sum<<endl;**
- The manipulator setw(5) specifies a field width of 5 for
- printing the value of variable sum the value is right justified.
- _ _ 3 5 6

# new and delete

**new**

- The new operator allocated sufficient memory to hold a data object of type data-type & returns the address of the object.
- **EX p = new int.**
- Where p is a pointer of type int. Here p must have already been declared as pointer of appropriate types.
- New can be used to create a memory space for any data type
- including user defined type such all array, classes etc.
- **Ex int * p = new int [10]**
- Creates a memory space for an array of 10 integers.

**delete**

- When a data object is no longer needed, it is destroyed to release the memory space for reuse. The general form is delete variable. If we want to free a dynamically allocated array, we must use following form.
- delete [size] variable.
- **delete [ ] p** new versions
- The size specifies the no of elements in the array to be freed.

# Contd..

The new operator has following advantages over the function malloc() in c -.

- It automatically computes the size of the data object. No need to use sizeof()
- If automatically returns the correct pointer type, so that there is no need to use a type cast.
- new and delete operators can be overloaded.
- It is possible to initialize the object while creating the memory space.
- int * array = malloc(10 * **sizeof**(int))
- int * array = calloc(10, **sizeof** (int)) - initialized to zero

# Control Statements

- do while
- while
- for
- if
- if-else
- Nested if
- switch
- break
- continue

# Functions

- Definition
- Declaration
- Prototype – essential in C++
- Call

# Arguments Type

- Function Call ----
  - Actual
- Function Definition ----
  - Formal
- Function Prototype ------
  - dummy (optional)

# Call by reference

function definition to swap the values

**By value**

```
void swap(int x, int y)
 { int temp;
temp = x;          /* save the value of x */
x = y;             /* put y into x */
y = temp;          /* put x into y */
 return; }
```

**By Reference in C & C++ (using pointer)**

```
void swap(int *x, int *y)
{ int temp;
temp = *x;         /* value at address x */
*x = *y;           /* put y into x */
*y = temp;         /* put temp into y */
return; }
```

**By reference in C++ (reference variable)**

```
void swap(int &x, int &y)
{ int temp;
 temp = x;         /* value at address x*/
x = y;             /* put y into x */
y = temp;          /* put x into y */
return; }
```

# Inline Function

- These are the functions designed to speed up program execution.
- An inline function is expanded (i.e. the function code is replaced when a call to the inline function is made) in the line where it is invoked.
- In case of normal functions, the compiler have to jump to another location for the execution of the function and then the control is returned back to the instruction immediately after the function call statement.
- Execution time taken is more in case of normal functions.
- There is a memory penalty in the case of an inline function.

# Inline Fn Contd..

Example

```
//function definition min()
inline void min (int x, int y)
cout<< (x < Y? x : y);
}
Void main()
{
int num1, num2;
cout<<"\Enter the two intergers\n";
cin>>num1>>num2;
min (num1,num2) ;     //function code inserted here
-----------------
}
```

# Inline Fn Contd..

- An inline function definition must be defined before being invoked
- min ( ) being inline will not be called during execution, but its code would be inserted into main ( ) as shown and then it would be compiled.
- If the size of the inline function is large then heavy memory penalty makes it not so useful and in that case normal function use is more useful.

# Inline Fn Contd..

- **The inlining does not work for the following situations :**
  1. For functions returning values and having a *loop or a switch or a goto statement.*
  2. For functions that do not return value and having a return statement.
  3. For functions having static variable(s).
  4. If the inline functions are recursive (i.e. a function defined in terms of itself).

- **The benefits of inline functions are as follows :**
  1. Better than a macro.
  2. Function call overheads are eliminated.
  3. Program becomes more readable.
  4. Program executes more efficiently.

# OVERLOADED FUNCTIONS

- In C++ two different functions can have the same name if their parameter types or number are different.
- You can give the same name to more than one function if they have either a different number of parameters or different types in their parameters.

**EX**

```
int operate (int a, int b)
{
return (a*b);
}
float operate (float a, float b)
{
return (a/b);
}
```

- a function cannot be overloaded only by its return type.
- At least one of its parameters must have a different type.

# macro

- In the **C** Programming Language, the **#define** directive allows the definition of macros within your source code. These macro definitions allow constant values to be declared for use throughout your code. Macro definitions are not variables and cannot be changed by your program code like variables.

```
#include <stdio.h>
#define PI  3.1415
#define circleArea(r)      (PI*r*r)
int main() {
float radius, area;
printf("Enter the radius: ");
scanf("%f", &radius);
area = circleArea(radius);
 printf("Area = %.2f", area);
 return 0; }
```