Object Oriented Programming

1 INTRODUCTION

Why Object Technology?

Expectations are:

- ► Reducing the effort, complexity, and cost of development and maintenance of software systems.
- ► Reducing the time to adapt an existing system (quicker reaction to changes in the business environment).
- ► Flexibility, reusability.
- ► Increasing the reliability of the system.

Why C++

- C++ supports writing high quality programs (supports OO)
- C++ is used by hundreds of thousands of programmers in every application domain.
- Application domain:
  - Systems programming: Operating systems, device drivers. Here, direct manipulation of hardware under real-time constraints are important.
  - Banking, trading, insurance: Maintainability, ease of extension, ease of testing and reliability is important.
  - Graphics and user interface programs
  - Computer Communication Programs

Software Quality Metrics

- A program must do its job correctly. It must be useful and usable.
- A program must perform as fast as necessary (Real-time constraints).
- A program must not waste system resources (processor time, memory, disk capacity, network capacity) too much.
- It must be reliable.
- It must be easy to update the program.
- A good software must have sufficient documentation (users manual).

user

- Source code must be readable and understandable.
- It must be easy to maintain and update (change) the program.
- A program must consist of independent modules, with limited interaction.
- An error may not affect other parts of a program (Locality of errors).
- Modules of the program must be reusable in further projects.
- A good software must have sufficient documentation (about development).

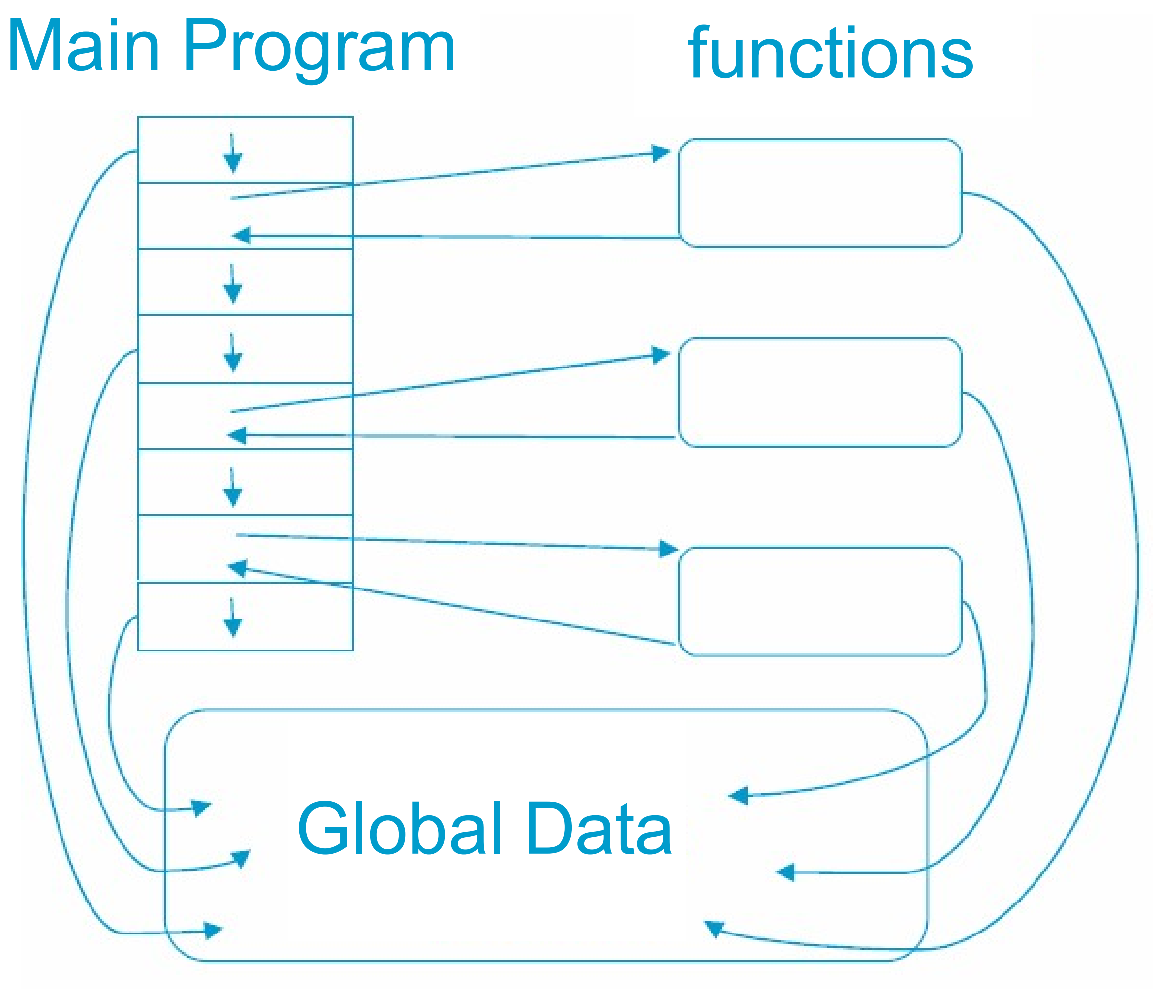
Software developer

Object-oriented programming technique enables programmers to build high-quality programs. While designing and coding a program, these quality metrics must be kept always in mind.

Procedural Programming

- Pascal, C, BASIC, Fortran, and similar traditional programming languages are procedural languages. That is, each statement in the language tells the computer to do something.
- In a procedural language, the emphasis is on doing things (functions).
- A program is divided into functions and—ideally, at least—each function has a clearly defined purpose and a clearly defined interface to the other functions in the program.

Procedural Programming



Problems with Procedural Programming

- Data is Undervalued
- Data is, after all, the reason for a program's existence. The important parts of a program about a school for example, are not functions that display the data or functions that checks for correct input; they are student, teacher data.
- Procedural programs (functions and data structures) don't model the real world very well. The real world does not consist of functions.
- Global data can be corrupted by functions that have no business changing it.
- To add new data items, all the functions that access the data must be modified so that they can also access these new items.
- Creating new data types is difficult.

Besides...

- It is also possible to write good programs by using procedural programming (C programs).
- But object-oriented programming offers programmers many advantages, to enable them to write high-quality programs.

Object Oriented Programming

The fundamental idea behind object-oriented programming is:

• The real world consists of objects. Computer programs may contain computer world representations of the things (objects) that constitute the solutions of real world problems.

• Real world objects have two parts:

- •*Properties* (or *state* :characteristics that can change),
- •*Behavior* (or *abilities* :things they can do).

•To solve a programming problem in an object-oriented language, the programmer no longer asks *how the problem will be divided into functions*, but **how it will be divided into objects**.

•The emphasis is on **data**

Object Oriented Programming

- What kinds of things become objects in object-oriented programs?
  - Human entities: Employees, customers, salespeople, worker, manager
  - Graphics program: Point, line, square, circle, ...
  - Mathematics: Complex numbers, matrix
  - Computer user environment: Windows, menus, buttons
  - Data-storage constructs: Customized arrays, stacks, linked lists

OOP : Encapsulation and Data Hiding

Thinking in terms of objects rather than functions has a helpful effect on design process of programs. This results from the close match between objects in the programming sense and objects in the real world.

To create software models of real world objects both *data* and the *functions* that operate on that data are combined into a single program entity. Data represent the properties (state), and functions represent the behavior of an object. Data and its functions are said to be *encapsulated* into a single entity.

An object's functions, called *member functions* in C++ typically provide the only way to access its data. The data is *hidden*, so it is safe from accidental alteration.

OOP : Encapsulation and Data Hiding

- ***Encapsulation*** and ***data hiding*** are key terms in the description of object-oriented languages.

- If we want to modify the data in an object, we know exactly what functions interact with it: the member functions in the object. No other functions can access the data. This simplifies writing, debugging, and maintaining the program.

Example: A Point on the plane

A Point on a plane has two properties; x-y coordinates.

Abilities (behavior) of a Point are, moving on the plane, appearing on the screen and disappearing.

A model for 2 dimensional points with the following parts:

Two integer variables (**x** , **y**) to represent x and y coordinates

A function to move the point: **move** ,

A function to print the point on the screen: **print** ,

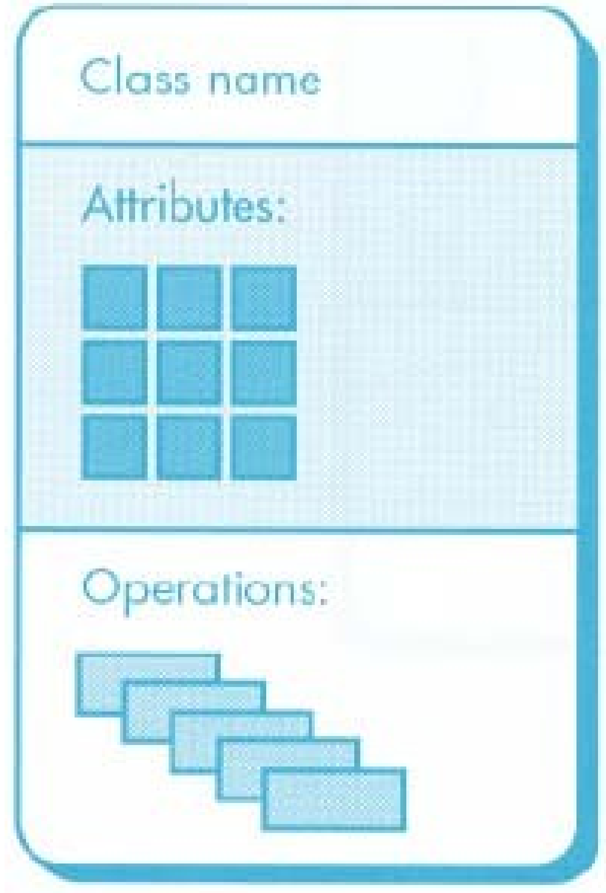
A function to hide the point: **hide** .

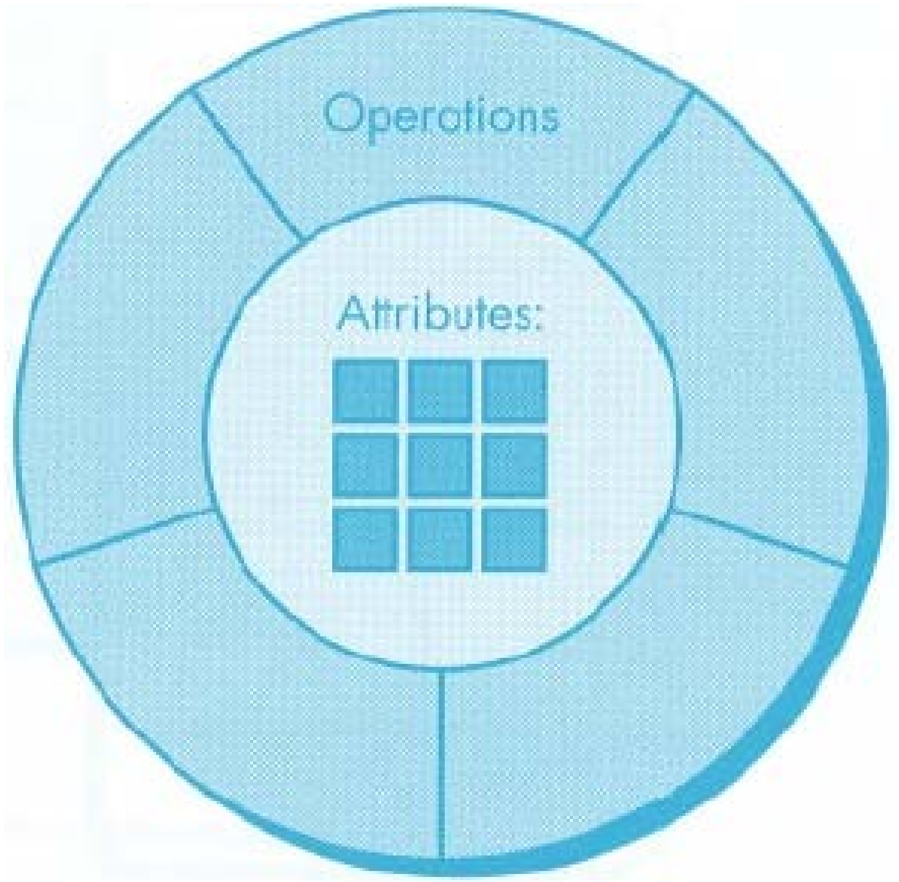
Example: A Point on the plane

Once the model has been built and tested, it is possible to create many objects of this model , in main program.

```
Point point1, point2, point3;
    :
point1.move(50,30);
point1.print();
```

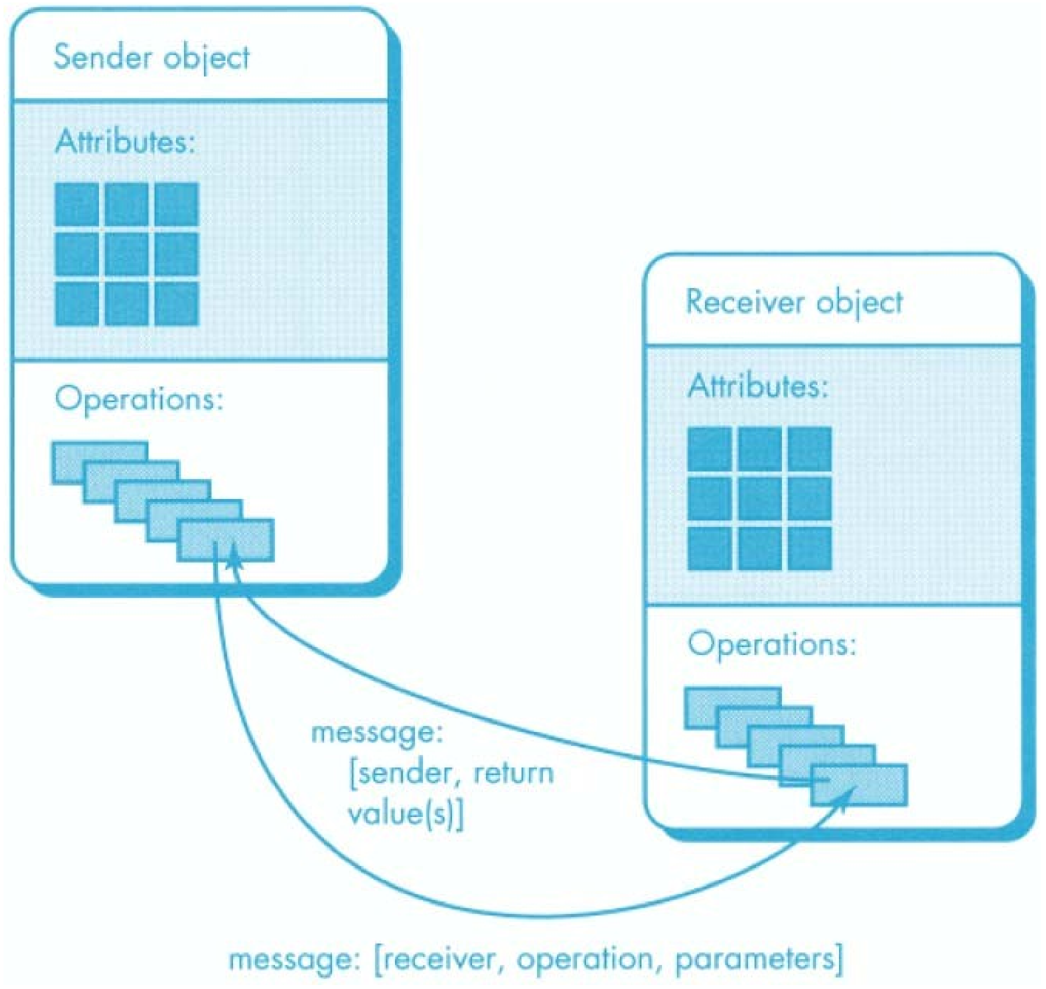
The Object Model





A C++ program typically consists of a number of objects that communicate with each other by calling one another's member functions.

The Object Model


Sender object
Attributes:
Operations:
Receiver object
Attributes:
Operations:
message: [sender, return value(s)]
message: [receiver, operation, parameters]

OOP *vs*. Procedural Programming

Procedural Programming:

•Procedural languages still requires us to think in terms of the structure of the computer rather than the structure of the problem we are trying to solve.

•The programmer must establish the association between the machine model and the model of the problem that is actually being solved.

•The effort required to perform this mapping produces programs that are difficult to write and expensive to maintain. Because the real world thing and their models on the computer are quite different.

Example: Procedural Programming

- Real world thing: student
- Computer model: char *, int, float ...
- It is said that the C language is closer to the computer than the problem.

OOP *vs*. Procedural Programming

Object Oriented Programming

► The object-oriented approach provides tools for the programmer to represent elements in the problem space.

► We refer to the elements in the problem space and their representations in the solution space as "objects."

► The idea is that the program is allowed to adapt itself to the problem by adding new types of objects, so when we read the code describing the solution, we're reading words that also express the problem.

► OOP allows us to describe the problem in terms of the problem, rather than in terms of the computer where the solution will run.

OOP *vs*. Procedural Programming

- Benefits of the object-oriented programming:
  - Readability
  - Understandability
  - Low probability of errors
  - Maintenance
  - Reusability
  - Teamwork

Function Overloading

- Function Overloading

```
double average(const double a[],int size) ;
double average(const int a[],int size) ;
double average(const int a[], const double b[],int size) ;
```

❶
```
double average(const int a[],int size)
        { double sum = 0.0 ;
        for(int i=0;i<size;i++)    sum += a[i] ;
        return ((double)sum/size) ;
}
```

```
❷   double average(const double a[],int size)
            { double sum = 0.0 ;
            for(int i=0;i<size;i++)   sum += a[i] ;
            return (sum/size) ;
    }
❸   double average(const int a[],const double b[],int size)
            { double sum = 0.0 ;
            for(int i=0;i<size;i++)   sum += a[i] + b[i] ;
            return (sum/size) ;
    }
```

```
int main() {
        int       w[5]={1,2,3,4,5} ;
        double  x[5]={1.1,2.2,3.3,4.4,5.5} ;
        cout << average(w,5) ;
        cout << average(x,5) ;
        cout << average(w,x,5) ;
        return 0 ;
}
```

Operator Overloading

- In C++ it is also possible to overload the built-in C++ operators such as +, -, = and ++ so that they too invoke different functions, depending on their operands.

- That is, the + in **a**+**b** will add the variables if **a** and **b** are integers, but will call a different function if **a** and **b** are variables of a user defined type.

Operator Overloading: Rules

- We can’t overload operators that don’t already exist in C++.
- We can not change numbers of operands. A binary operator (for example +) must always take two operands.
- We can not change the precedence of the operators.

  * comes always before +
- Everything we can do with an overloaded operator we can also do with a function. However, by making our listing more intuitive, overloaded operators make our programs easier to write, read, and maintain.
- Operator overloading is mostly used with objects. We will discuss this topic later more in detail.