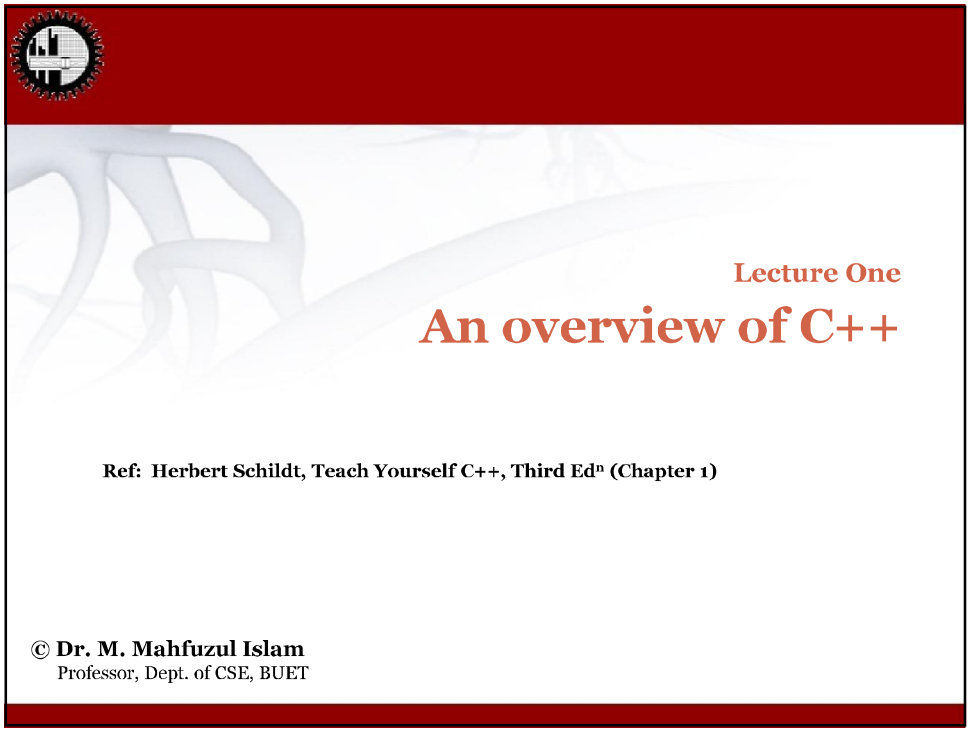


Object Oriented Programming (OOP)

- Emphasis is on data rather than procedure
- Programs are divided into classes. Instance of a class is called object.
- Data and functions are build around objects
- Data doesn't flow freely around the system
  - ✓ Data is hidden, no access from external functions
  - ✓ Data of an object can be accessed only by the functions associated with that object
- Objects communicate with each other through functions
- new data and functions can be easily added.

Example: C++, Java, Smalltalk

Features of OOP

Three Key Features of OOP

Encapsulation

- Wrap up data and functions/methods together
- Insulation of data from direct access by the program – data hiding

Polymorphism

One Interface, multiple methods

- Function overloading:
  (1) Use a single name to multiple methods;
  (2) different number and types of arguments.
- Operator overloading:
  Use of single operator for different types of operands

Inheritance

- One class inherits the properties of another class
- provide hierarchical classifications
- Permits reuse of common code and data

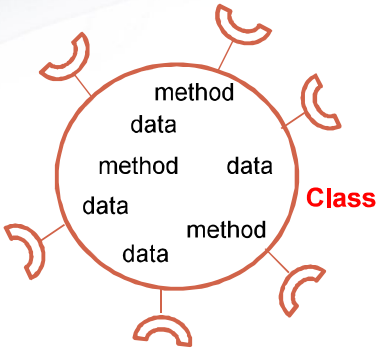
- Representing essential features without details
- Class defines a list of abstract attributes (data members) and methods to operate on these attributes

Abstraction

Encapsulation

- Wrap up data and functions/methods together
- Insulation of data from direct access by the program – data hiding

```
#include <iostream>
using namespace std;
class myclass {
   int  a;
public:
   myclass();  // constructor
   int geta() { return a;}
};
myclass::myclass(){
   cout << "In constructor\n";
   a = 10;
}
int main(){
   myclass ob;

    cout << a;              \\ wrong -Error
    cout << ob.geta();      \\ OK
    return 0;
}
```



Polymorphism

One Interface, multiple methods

- **Function overloading**:
  (1) Use a **single name** to **multiple methods**;
  (2) **different** number and types of **arguments**.
- **Operator overloading**:
  Use of **single operator** for **different** types of operands

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

class date {
      int month, day, year;
public:
      date(char *str);
      date( int d, int m, int y) ){
            day = d;
            month = m;
            year = y;
      }
      void show(){
            cout << day << '/' << month << '/' ;
            cout << year << '\n';
      }
};
```

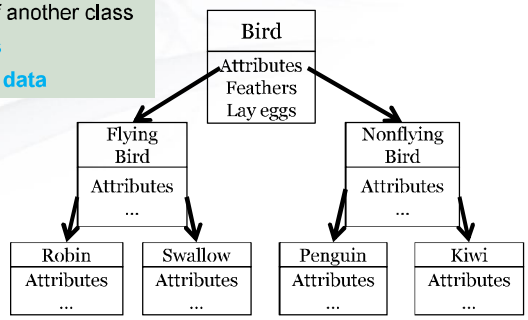
```
date::date(char *str){
      sscanf(str, "%d%*c%d%*c%d; &day,
                        &month, &year);
}

int main() {
     date sdate("31/12/99");
     date idate(31, 12, 99);

     sdate.show();
     idate.show();
     return 0;
}
```

Operator overloading:

```
a = 4 + 6;
ob1 = ob2 + ob3;
```

Inheritance

- One class **inherits** the **properties** of another class
- provide **hierarchical classifications**
- Permits **reuse** of common **code** and **data**




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

class base {
      int x;
public:
      void setx(int n) { x = n; }
      void showx() { cout << x << '\n'; }
      int getx(){ return x; }
};

class derived: public base {
      int y;
public:
      void sety(int n) { y = n; }
      void showy() {
            cout << y << '\n';
            cout << y+getx() <<'\n';
       }
};
```

```
int main(){
      derived ob;

      ob.setx(10);
      ob.sety(20);
      ob.showx();
      ob.showy();
      return 0;
}
```

Abstraction

Abstract Class and Method:

- **Abstract class** is a superclass without a complete implementation of every method.
  - There can be **no objects** of an abstract class.
  - Abstract can be used to create **object references**.
- **Abstract method** refers to **subclasser responsibility** to override it, otherwise, it will report a **warning message**.
  - **Constructor** and **static method** cannot be **Abstract.**

Java Code

```
Abstract class Figure {
    double dim1, dim2;
    Figure(double a, double b){ dim1 = a; dim2 = b;}
    abstract double area();
}
class Rectangle extends Figure {
    Rectangle(double a, double b) {supper(a, b);}
    double area(){ return dim1*dim2;}
}
class Traingle extends Figure {
    Triangle(double a, double b) {supper(a, b);}
    double area(){ return 0.5*dim1*dim2;}
}
```

```
class Dispatch {
    public static void main(String args[]){
        Rectangle r = new Rectangle(4,5);
        Triangle t = new Triangle(4, 3);
        Figure figref;

        figref = r;
        System.out.println("area: "+ figref.area());
    }
}
```

Programming with C++

Two versions of C++:

Old version of C++:

```
#include <iostream.h>

int  main(){
    /*  program code */
    return 0;
}
```

- Includes **filename**

New version of C++:

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

int main(){
    /* program code */
    return 0;
}
```

- Includes **stream** which is mapped to file by compiler

Filename used in Old Version	File stream used in New Version
iostream.h	iostream
string.h	cstring
math.h	cmath
graphics.h	cgraphics

Bjarne Stroustrup
(1979)

C/C++ I/O

General form of C++ Console I/O

- **Input Command:** **cin >> variable;**
- **Output Command:** **cout << expression;**

C I/O	C++ I/O
a) Input Statements	
scanf("%s", strName);	cin >> strName;
scanf("%d", &iCount);	cin >> iCount;
scanf("%f", &fValue);	cin >> fValue;
scanf("%d %d %d", &day, &month, &year);	cin >> day >> month >> year;
b) Output Statements	
printf("%s%c%s%c", "Hello", ' ', "World", '!');	cout << "Hello" << ' ' << "World" << '!';
printf("Value of iCount is: %d", iCount);	cout << "Value of iCount is: " << iCount;
printf("Enter day, month, year");	cout << "Enter day, month, year: ";

Namespaces

- A **namespace** is a **declarative region** that **localizes** the names of identifiers to **avoid name collisions.**
- The contents of new-style headers are placed in the **std namespace.**
- A newly created class, function or global variable can put in-
  - (1) an existing namespace;
  - (2) a new namespace; and
  - (3) unnamed namespace.

```
#include <iostream>

int main() {
    char str[16];

    std::cout << "Enter a string: ";
    std::cin >> str;
    std::cout << "String: " << str;

    return 0;
}
```

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

int main() {
    char str[16];

    cout << "Enter a string: ";
    cin >> str;
    cout << "String: " << str;

    return 0;
}
```

Scope Resolution Operator (::)

Scope Resolution Operator is used for **two purposes**:

To access a **hidden global variable**

```
#include <iostream>

int count = 0;

int main(void) {
    int count = 0;
    ::count = 1;    // set global count to 1
    count = 2;      // set local count to 2

    return 0;
}
```

To access a **hidden member class** or **member variable** with a **class**

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

class X {
public:
   static int count;
};

int X::count = 10;                  // define static data member

int main () {
    int X = 0;                      // hides class type X
    cout << X::count << endl;       // use static member of class X
}
```

C++ Comments

- Multi-line comments

  /* one or more lines of comments */

- Single line comments

  // ...

Some differences between C and C++

SL#	Area	C	C++
1.	Empty parameter list	**void** is mandatory. `char fl(void);`	**void** is optional. `char fl();`
2.	Function prototype	Function prototype is optional but recommended.	All functions must be prototyped.
3.	Returning a value	➢A **non-void** function in not required to actually return a value. If it doesn't, a **garbage value** is returned. ➢ "**Default-to-int**" rule: If a function does not explicitly specify the return type, an integer return type is assumed.	➢If a function is declared as returning a value, it **must** return a value. ➢ C++ has **dropped** the "default-to-int" rule.
4.	Local variable declaration	Local variables are declared at the **start of a block**, prior to any action statement.	Local variables can be declared **anywhere**.
5.	**bool** data type	-	C++ defines the **bool** data type and also **keywords true** and **false**.