Object-Oriented Programming

- Class Definition
- Class Examples
- Objects
- Constructors
- Destructors
### Class

- **The class is the cornerstone of C++**
  - It makes possible encapsulation, data hiding and inheritance
- **Type**
  - Concrete representation of a concept
    - Eg. `float` with operations like -, *, + (math real numbers)
- **Class**
  - A user defined type
  - Consists of both data and methods
  - Defines properties and behavior of that type
- **Advantages**
  - Types matching program concepts
    - Game Program (Explosion type)
  - Concise program
  - Code analysis easy
  - Compiler can detect illegal uses of types
- **Data Abstraction**
  - Separate the implementation details from its essential properties
Classes & Objects

```cpp
class Rectangle {
private:
    int width;
    int length;
public:
    void set(int w, int l);
    int area();
};
```

**Objects:** Instance of a class

```cpp
Rectangle r1;
Rectangle r2;
Rectangle r3;
...```

- Class `Rectangle` defines properties `width` and `length` as private members.
- Public methods `set` and `area` are provided for manipulating the rectangle.
- Objects `r1`, `r2`, and `r3` are instances of the `Rectangle` class.
Define a Class Type

```cpp
class class_name {
  permission_label: member;
  permission_label: member;
...
};

class Rectangle {
private:
  int width;
  int length;
public:
  void set(int w, int l);
  int area();
};
```
class Rectangle
{
    private:
        int width;
        int length;
        static int count;
    public:
        void set(int w, int l);
        int area();
};

Rectangle r1;
Rectangle r2;
Rectangle r3;
Define a Member Function

```cpp
class Rectangle
{
    private:
        int width, length;
    public:
        void set (int w, int l);
        int area() {return width*length; }
};

void Rectangle :: set (int w, int l)
{
    width = w;
    length = l;
}
```

```cpp
r1.set(5,8);
rp->set(8,10);
```
class Time
{
private:
    int hrs, mins, secs;

public:
    void Write() const;
};

void Time::Write() const
{
    cout << hrs << ":" << mins << ":" << secs << endl;
}
Member Functions

- **const member function**
  - declaration
    - `return_type func_name (para_list) const;`
  - definition
    - `return_type func_name (para_list) const { … }`
    - `return_type class_name :: func_name (para_list) const { … }`
  - Makes no modification about the data members (safe function)
  - It is illegal for a const member function to modify a class data member
Access Control

- **Information hiding**
  - To prevent the internal representation from direct access from outside the class

- **Access Specifiers**
  - public
    - may be accessible from anywhere within a program
  - private
    - may be accessed only by the member functions, and friends of this class
  - protected
    - acts as public for derived classes
    - behaves as private for the rest of the program
What is an Object?

**OBJECT**

- **Operations**
  - set of methods (member functions)

- **Data**
  - internal state (values of private data members)
Declaration of an Object

class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};

main()
{
    Rectangle r1;
    Rectangle r2;
    r1.set(5, 8);
    cout<<r1.area()<<endl;
    r2.set(8,10);
    cout<<r2.area()<<endl;
}
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};

main()
{
    Rectangle r1;
    r1.set(5, 8);
}

r1 is statically allocated

r1

width = 5
length = 8
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};

main()
{
    Rectangle r1;
    r1.set(5, 8);  // dot notation
    Rectangle *r2;
    r2 = &r1;
    r2->set(8, 10); // arrow notation

    r1
    width = 5
    length = 8
    5000

    r2
    width = 8
    length = 10
    6000

   // Declaration of an Object
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};

main()
{
    Rectangle *r3;
    r3 = new Rectangle();
    r3->set(80,100); //arrow notation
    delete r3;
    r3 = NULL;
}

r3 is dynamically allocated

r3
6000
NULL
# Objective Initialization

```cpp
#include <iostream.h>

class circle {
    public:
        double radius;
};

int main() {
    circle c1; // Declare an instance of the class circle
    c1.radius = 5; // Initialize by assignment
}
```

1. By Assignment
   - Only work for public data members
   - No control over the operations on data members
#include <iostream.h>

class circle
{
    private:
        double radius;

    public:
        void set (double r)
        {
            radius = r;
        }

        double get_r ()
        {
            return radius;
        }
};

int main(void) {
    circle c; // an object of circle class
    c.set(5.0); // initialize an object with a public member function
    cout << "The radius of circle c is " << c.get_r() << endl; // access a private data member with an accessor
}

2. By Public Member Functions
class Rectangle
{
    private:
        int width;
        int length;
    public:
        Rectangle();
        Rectangle(const Rectangle &r);
        Rectangle(int w, int l);
        void set(int w, int l);
        int area();
};

3. By Constructor

- Default constructor
- Copy constructor
- Constructor with parameters

They are publicly accessible
Have the same name as the class
There is no return type
Are used to initialize class data members
They have different signatures
Class Constructor

```cpp
class Rectangle
{
    private:
        int width;
        int length;
    public:
        void set(int w, int l);
        int area();
};
```

- **Default constructor**
  
  ```cpp
  Rectangle :: Rectangle() {};
  ```

- **Copy constructor**
  
  ```cpp
  Rectangle :: Rectangle (const Rectangle & r)
  {
      width = r.width;  length = r.length;
  };
  ```

When a class is declared with no constructors, the compiler automatically assumes `default` constructor and `copy` constructor for it.
class Account
{
    private:
        char *name;
        double balance;
        unsigned int id; //unique
    public:
        Account();
        Account(const Account &a);
        Account(const char *person);
        ~Account();
};

Destructor

Account :: ~Account()
{
    delete[] name;
}

• Its name is the class name preceded by a ~ (tilde)
• It has no argument
• It is used to release dynamically allocated memory and to perform other "cleanup" activities
• It is executed automatically when the object goes out of scope
Interacting Objects

Class A

- Private: data members
- Member methods
  - Public:
    - Constructor
    - Destructor
    - Other public methods
  - Private: methods

Class B

- Private: data members
- Member methods
  - Public:
    - Constructor
    - Destructor
    - Other public methods
  - Private: methods

Message passing
Working with Multiple Files

- To improve the readability, maintainability and reusability, codes are organized into modules.

- When working with complicated codes,
  - A set of .cpp and .h files for each class groups
    - .h file contains the prototype of the class
    - .cpp contains the definition/implementation of the class
  - A .cpp file containing main() function, should include all the corresponding .h files where the functions used in .cpp file are defined
Example: time.h

// SPECIFICATION FILE ( time.h )
// Specifies the data members and
// member functions prototypes.

#ifndef _TIME_H
#define _TIME_H

class Time
{
    public:
        ...
        
    private:
        ...
};

#endif
// IMPLEMENTATION FILE (time.cpp)
// Implements the member functions of class Time

#include <iostream.h>
#include “time.h”  // also must appear in client code

bool Time :: Equal ( Time otherTime ) const
{
    return ( (hrs == otherTime.hrs) && (mins == otherTime.mins)
              && (secs == otherTime.secs) ) ;
}

...
Example: main.cpp

// Client Code (main.cpp)
#include "time.h"

// other functions, if any

int main()
{
    ...
    ...
}

Compile and Run

g++ -o mainExec main.cpp time.cpp
Separate Compilation and Linking of Files

- `main.cpp`
- `time.h`
- `main.o`
- `time.o`
- `mainExec`

### Explanation
- The `main.cpp` file is the main program.
- The `time.h` file is the specification file.
- The `time.cpp` file is the implementation file.
- The `#include “time.h”` line indicates that `main.cpp` includes `time.h`.
- The `Compiler` tool processes `main.cpp` to produce `main.o`.
- The `Compiler` tool processes `time.cpp` to produce `time.o`.
- The `Linker` tool combines `main.o` and `time.o` to produce `mainExec`.

This process illustrates how separate compilation and linking work in a typical C++ program.
[Lab – Practice #1]

- Blackjack game
  - a change of your project