1. Introduction

In this lecture we will learn basic classes in C++.

C and C++ allow you to define your own data types. These user-defined data types are created using the struct or the class keywords.

In C++, a class is like an array: it is a derived type. But unlike an array, the elements of a class may have different types. Furthermore, some elements of a class may be functions and operators.
The `struct` keyword is mostly used in the C programming language. In C, the elements of `struct` can be ordinary data types and/or other structures. To remain compatible with the C language, C++ maintains the `struct` keyword. However, in C++, a `struct` and a `class` have the same meaning and functionality.

Although any storage region in RAM is referred to as an `object`, the word is usually used to describe variables whose data type is a class. Thus `object-oriented programming` involves programs that use classes.

2. Structures in C/C++

A data structure (or derived data type) is a set of data elements grouped together under one name.

These data elements, known as members, can have different types and different lengths.

```c
struct name {
    type1 member_name1;
    type2 member_name2;
    .
    .
    .
} object_names;
```

```c
struct Student{
    string name;
    int mt1, mt2, fin;
    double avr;
} s1, s2;
```

Here, `Student` is a new valid type name like the fundamental ones `int` or `double`. `s1` and `s2` are objects (or variables) derived from this new type.
### 3. Basic Classes

- A **class** is an expanded concept of a data structure in C. Instead of holding only data, a **class can hold both data and functions**.

- An **object** is an instantiation of a class. In terms of variables, a class would be the **type**, and an object would be the **variable**.

- Classes are decelerated by using **class** keyword.

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

struct Fruit{
    double weight;
    double price;
};

int main(){
    Fruit orange, apricot;
    orange.price = 2.50; // TL/kg
    apricot.price = 3.25; // TL/kg
    cin >> orange.weight;
    cout << "Input the amount of orange in kg: ";
    cin >> orange.weight;
    cout << "Input the amount of orange in kg: ";
    cin >> apricot.weight;
    cout << "\nTotal prices (TL):\nOrange = " << orange.price * orange.weight << endl;
    cout << "Apricot = " << apricot.price * apricot.weight << endl;
}
```
An access specifier is one of the followings:

- **private**
  members of a class are accessible only from within other members of the same class

- **public**
  members are accessible from anywhere where the object is visible

- **protected**
  members are accessible from members of their same class but also from members of their derived classes

By default, all members of a class declared with the **class** keyword have **private** access for all its members.

The following class can be used to represent a planet whose mass is $M$ and radius is $R$.

```cpp
// Example Class
class Planet{
public:
    void SetMassRadius(double, double);
    double Density();
    double Gravity();
private:
    double M, R, G;
};
```

- declares a class (i.e. a type) called **Planet**
- The functions: 
  `SetMassRadius()` 
  `Density()` 
  `Gravity()`
- Member $M, R$ and $G$ have (default) **private** access and member functions have **public** access.
Implementation of the Planet Class

- Consider a planet of mass $M$ and equatorial radius $R$. The mean mass density \( d \) and equatorial gravity \( g \) of the planet are given respectively by

\[
g = \frac{GM}{R^2}
\]

\[
d = \frac{M}{4\pi R^3 / 3}
\]

- where \( G \) is the universal gravitational constant and has the value $6.67428 \times 10^{-11}$ m$^3$/kg/s.
A basic use of classes

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

class Planet{
public:
    void SetMassRadius(double, double);
    double Density();
    double Gravity();
private:
    double M, R, G;
};

int main(){
    Planet Mars;
    Mars.SetMassRadius(6.4e23, 3.4e6);
    cout << "Density = " << Mars.Density() << endl;
    cout << "Gravity = " << Mars.Gravity() << endl;
}
```

// continue...

Set the mass (kg) and equatorial radius (m) of the planet

```cpp
void Planet::SetMassRadius(double mass, double radius){
    M = mass;
    R = radius;
    G = 6.67428e-11;
}

// Mass density in g/cm3
double Planet::Density(){
    double d = M/(4.0*M_PI*R*R*R/3);
    return d * 1.0e-3;
}

// Surface gravity in m/s2
double Planet::Gravity(){
    double g = G*M/(R*R);
    return g;
}
```

Density = 3.88736
Gravity = 3.6951
Here Mars is declared to be an object of the Planet class.
Consequently, Mars has its own internal data members M, R, and G and has also ability call member functions.

The mass and radius of Mars are supplied via the SetMassRadius() method.

Its density and surface gravity are evaluated and output.

Notice one must use the specifier Planet:: before each member function to indicate that these functions are the members of the Planet class.

The output shows that the density of the Mars is about 3.9 g/cm³ and its surface gravity is 3.7 m/s².

---

- public members are accessible from outside the class but private members are not.

- Therefore, the following accesses are forbidden:

  ```cpp
  cout << Mars.M << endl; // forbidden
  cout << Mars.R << endl; // forbidden
  ```
// Self contained implementation in a class
#include <iostream>
#include <cmath>
using namespace std;

class Planet{
public:
    void SetMassRadius(double mass, double radius){
        M = mass; R = radius; G = 6.67428e-11;
    }
    double Density(){
        return 1.0e-3 * M/(4.0*_PI*R*R*R/3); }
    double Gravity(){ return G*M/(R*R); }
private:
    double M, R, G;
};

int main(){
    Planet Mars;
    Mars.SetMassRadius(6.4e23, 3.4e6);

    cout << "Density = " << Mars.Density() << endl;
    cout << "Gravity = " << Mars.Gravity() << endl;
}

Constructors and Destructors

- The Planet class uses the SetMassRadius() function to initialize its objects. However, you can initialize the values when the object is declared like ordinary variables
  
  ```
  int p = 35;
  string name = "Bjarne";
  ```

- This is done by means of a constructor function which is a member function called automatically when an object is declared.

- A constructor function must have the same name as the class name and have no return type.
// A basic use of class constructor
#include <iostream>
#include <cmath>
using namespace std;

class Planet{
  public:
    Planet(double, double);
    double Density();
    double Gravity();
  private:
    double M, R, G;
};

int main(){
  Planet Mars(6.4e23, 3.4e6), Jupiter(1.9e27, 7.0e7);

  cout << "Mars Density = " << Mars.Density() << endl;
  cout << "Mars Gravity = " << Mars.Gravity() << endl;

  cout << "Jupiter Density = " << Jupiter.Density() << endl;
  cout << "Jupiter Gravity = " << Jupiter.Gravity() << endl;
}

// continue ...

// Set the mass (kg) and
equatorial radius (m) of the planet
Planet::Planet(double mass, double radius){
  M = mass;
  R = radius;
  G = 6.67428e-11;
}

// Mass density in g/cm3
double Planet::Density(){
  double d = M/(4.0*PI*R*R*R/3);
  return d * 1.0e-3;
}

// Surface gravity in m/s2
double Planet::Gravity(){
  double g = G*M/(R*R);
  return g;
}

Mars Density = 3.88736
Mars Gravity = 3.6951
Jupiter Density = 1.32242
Jupiter Gravity = 25.8799
Pointers to Classes

It is perfectly valid to create pointers that point to classes.

For example:

```c++
Planet *p;
```

is a pointer to an object of class `Planet`.

In order to refer directly to a member of an object pointed by a pointer we can use the arrow operator (`->`) of indirection.

```c++
// Pointer to a class
#include <iostream>
#include <cmath>
using namespace std;

class Planet{
 public:
   Planet(double mass, double radius){
      M = mass; R = radius; G = 6.67428e-11;
   }
   double Density(){ return 1.0e-3 * M/(4.0*M_PI*R*R*R/3); }
   double Gravity(){ return G*M/(R*R); }
 private:
   double M, R, G;
};

int main(){
   Planet *gezegen = new Planet(6.4e23, 3.4e6);
   cout << "Density = " << gezegen->Density() << endl;
   cout << "Gravity = " << gezegen->Gravity() << endl;
}
```
Example: ‘A Cat class’

Each object of this class will represent a cat. The class includes

* a constructor function whose prototype is
  \[
  \text{Cat(int \text{ Age}=1, double \text{ Mass}=2.0);} \\
  \text{to set (initialize) the age and weight of the cat.}
  \]

* a member function named \texttt{void speak()} that outputs a "meow" message.

* a member function named \texttt{void kill()} that reduces the cat's lives by one
  (the cat has nine lives).

* a member function named \texttt{double getMass()} to get the mass of the cat.

* a member function named \texttt{int getAge()} to get the age of the cat.

* a member function named \texttt{int getLife()} to get the remaining life(s) of the cat.


Including a Class from a File

The contents of the main program, and of the class(es), can be
placed into separate files.

Then, using the \texttt{#include} directive you can use the class(es) required.

In general, the files containing classes (or functions) are called
\textit{header files}. Usually headers have the extension ".h" or ".hpp".
/ **Planet.h**

```cpp
#ifndef PLANET_H
#define PLANET_H

class Planet{
public:
    Planet(double, double);
    double Density();
    double Gravity();
private:
    double M, R, G;
};

// Constructor function to set the mass and radius of the planet
// By default the planet is assumed to be Earth
Planet::Planet(double mass = 6.0e24, double radius = 6.4e6)
    M = mass; R = radius;
    G = 6.67428e-11;

// Mass density in g/cm3
double Planet::Density(){
    return M/(4.0*M_PI*R*R*R/3) * 1.0e-3;
}

// Surface gravity in m/s2
double Planet::Gravity(){
    return G*M/(R*R);
}
#endif
```

// Including a class from a file
```cpp
#include <iostream>
#include <cmath>
using namespace std;

#include "Planet.h"

int main(){
    Planet Mars(6.4e23, 3.4e6), Jupiter(1.9e27, 7.0e7);

    cout << "Mars Density = " << Mars.Density() << endl;
    cout << "Mars Gravity = " << Mars.Gravity() << endl;
    cout << "Jupiter Density = " << Jupiter.Density() << endl;
    cout << "Jupiter Gravity = " << Jupiter.Gravity() << endl;
}
```
Homeworks

Solve the following problems. You have to prepare a pdf document and sent it to me until next lecture.

E-mail: bingul[at]gantep.edu.tr (replace [at] with @)

******************************************
Please put only
Circle.h and RCCircuit.h
into your pdf document
******************************************

In the xy plane, the general equation of a circle of radius r is given by: $(x - a)^2 + (y - b)^2 = r^2$.
Implement a Circle class. Each object of this class will represent a circle, storing its radius (r) and the a and b coordinates of its center as doubles.

The class must include:
- a default constructor function whose prototype is:
  
  ```
  Circle(double radius, double centerX, double centerY);
  ```
  
  to set (initialize) radius and center coordinates.
- a member function named double area() that returns the area of the circle.
- a member function named double circumference() that returns circumference.
- a member function named bool inside(double x, double y) that returns true if the given point (x, y) is inside the circle and returns false otherwise.

Assume that the class declaration and its members/methods are stored in the file Circle.h. An example usage of the Circle() is given below:

```c++
#include <iostream>

using namespace std;

#include "circle.h"

int main(){
    // a circle whose center is origin
    Circle guzelCember(10.0, 0.0, 0.0);
    cout << guzelCember.area() << endl;
    cout << guzelCember.circumference() << endl;
    cout << guzelCember.inside(1.5, 2.7) << endl;
    return 0;
}
```
Implement an Rcircuiat class. Each object of this class will represent a simple charging RC circuit.

The class must include

- A default constructor function whose prototype is
  `Rcircuiat(double R, double C, double V);
  ` to initialize the values of resistance (R) in Ohms, capacitor (C) in Farads and the potential difference across DC voltage source (V) in Volts.
- A member function named `double current(double t)` that returns the current the circuit at given time (in seconds) where t > 0.
- A member function named `double voltage(double t)` that returns potential across the capacitor at given time (in seconds) where t > 0.
- A member function named `double tau()` that returns the time constant of the circuit defined by $\tau = RC$.

Assume that the class declaration and its members/methods are stored in the file Rcircuiat.h.

Example usage of the Rcircuiat class is given below:

```
#include <iostream>
#include <Rcircuiat.h>

int main()
{
    Rcircuiat *device = new Rcircuiat(1.2e+6, 1.0e-6, 12);
    double time = 0.0;

    cout << "time constant: " << device->tau() << endl;

    do{
        cout << device->current(time) << " t" << endl;
        cout << device->voltage(time) << " V" << endl;
        time += 0.1;
    }while(time < 5*device->tau());

    return 0;
}
```