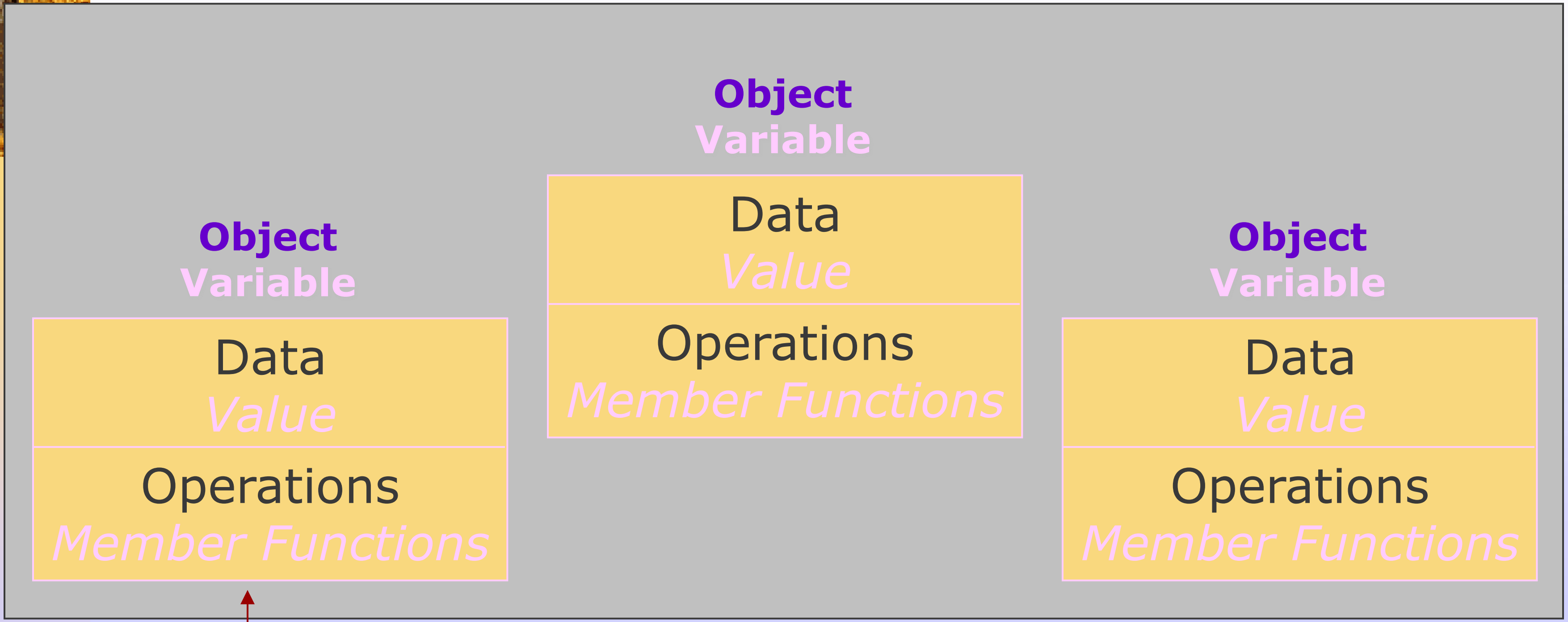
Classes and Data Abstraction

- Topic 5

Classes

Class

User Defined Data Type



Encapsulation: combining a number of items such as variables and functions into a single package (object).

Classes

Syntax:

```
class Class_Name
{
public:
        Member_Specification_1
        Member_Specification_2      }  public members
        ….
        Member_Specification_n
private:
        Member_Specification_n+1
        Member_Specification_n+2    }  private members
        ….
};
```

Classes

Example:

```
class Bicycle
{
public:
        char    get_color();
        int     number_of_speeds();
        void    set (int the_speeds, char the_color);
private:
        int speeds;
        char color;
};
```

public members

private members

```
Bicycle         my_bike, your_bike
```

Classes

Member Function Syntax:

```
Return_Type   Class_Name :: Function_Name (Parameter_List)
{
    Function_Body_Statements
};
```

```
Class DayOfYear
{
public:
        void    output();
        int     month;
        int     day;
};
```

Example:

```
void    DayOfYear::output()
{
    cout<< “month= “<<month<<“day= “<<day<<endl;
}
```

Classes

- *How do I call the member function output?*

```
DayOfYear  today;


today.month = 2;
today.day = 10;


today.output();
```

```
Class DayOfYear
{
public:
        void    output();
        int     month;
        int     day;
};
```

Classes

Dot Operator (.)

- Used with Objects – class variables
- Example:

```
new_student.output();
```

Scope Resolution Operator (::)

- Used with Class name
- Example:

```
void Student::output()
```

Classes & ADTs

Public Vs Private

- Separate the *rules for using* the class and the details of the class *implementation*
- Have enough member functions that you never need to access member variables directly, only through member functions
- → Code is easier to *understand* & *update*

Classes & ADTs

- Can we overload member functions?
  - void set(int the_id, char the_major[2]);
  - void set(int the_id);
  - void set(double score);

```
Student new_student;
new_student.set (16.0);
new_student.set (555);
New_student.set ((999,"CS");
```

Search for matching data types and/or number of parameters

Classes & ADTs

- Constructors: member functions automatically called when an object is declared
- Example:
  - Student (int the_id, char the_major);
  - Student (); ← Default constructor
    - When default constructor is called in main:
      - Student new_student;
      - Not: Student new_student();

```
Student   new_student(999,’C’)
```

Classes & ADTs

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

class Student
{
public:
      Student (int the_id, char the_major);
      Student (int the_id);
      Student ();
      int get_id();
      char get_major();
      void output();

private:
      int   id;
      char  major;
};
```

Classes & ADTs

```
int main()
{     Student new_student(55,'B');
      new_student.output();
      return 0;
}

Student::Student (int the_id, char the_major)
{     id = the_id;
      major = the_major;
}

Student::Student (int the_id)
{     id = the_id;
}

Student::Student ()
{     id = 0;
      major = 'X';
}

void Student::output()
{     cout<< id;
      cout<< major;
}
```

Classes & ADTs

- Constructors
  1. Default Constructor
  2. Constructor with all member variables
  3. Constructor with some member variables

```
class School
{
public:
          …
private:
      int       NumOfStudents;
      int       NumOf Classes;
      double Area;
}
```

```
School ();
School (int students, int classes, double area);
Student (int students);
Student (int students, int classes);
Student (int classes, double area);
```

Classes & ADTs

- Constructor Definitions:
  1. Must have the same name as the class
  2. Definition cannot return a value. No type – not even void- can be given at the start of the function prototype or header
- How can I call a constructor in main?

```
int main()
{  School myschool;
   School YourSchool(200,10,5000);
   School AnotherSchool();                    ✓
   AnotherSchool.School(300,15,4000);   ✗
   …                                          ✗
```

Classes & ADTs

- Accessor Functions
  - Functions that give you access to the values of the private member variables.

```
class School
{
public:
            …
private:
      int       NumOfStudents;
      int       NumOf Classes;
      double Area;
}
```

```
int get_Students();
//Return the number of students in a school
int get_Classes();
//Return the number of classes in a school
double get_Area();
//Return the area of a school
```

Classes & ADTs

- Private members → need for Accessor Functions

```
int get_id();
//returns the student id

char get_major();
//returns the student major

void set_id(int new_id);
//assigns a value to student id

void set_major(char new_major);
//assigns a value to student major
```

- Student
  - ID
  - Major

```
Student
{
public:
          int   id;
          char  major;

};
```

Classes & ADTs

```
Class DayOfYear
{
public:
        void    output();
private:
        int     month;
        int     day;
};
```

private member variables

Restriction:
Once you make a member variable private, the only way to access it or change its value is by using one of the member functions.

```
int main()
{  DayOfYear  Today;
   cin      >> Today.month;
   cout     << Today.month;
   If (Today.month ==1)
            cout << "January";
```

ILLEGAL!

6.2 Classes

Member Function Definition

```
void    DayOfYear::output()
{
        cout<< “month= “<< month;
        cout<<“day=“      << day;
        cout<<endl;
}
```

```
Class DayOfYear
{
public:
        void    output();
private:
        int     month;
        int     day;
};
```

Private members may be used in member function definitions (but not elsewhere).

6.2 Classes

```
Class Sample
{
public:
          int      variable;
          void     output();
          void     input();
private:
          int      month;
          int      day;
          void     doStuff();
};
```

public members

private members

Public members can be used in the main body of your program or in the definition of any function, even a non-member function.

```cpp
// Fig. 6.3: fig06_03.cpp
// Time class.
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setfill;
using std::setw;

// Time abstract data type (ADT) definition
class Time {

public:
   Time();                    // constructor
   void setTime( int, int, int ); // set hour, minute, second
   void printUniversal();        // print universal-time format
   void printStandard();         // print standard-time format

```

Define class **Time**.

fig06_03.cpp
(1 of 5)

```
private:
   int hour;     // 0 - 23 (24-hour clock format)
   int minute;   // 0 - 59
   int second;   // 0 - 59

}; // end class Time

// Time constructor initializes each data member to zero;
// ensures all Time objects start in a consistent state
Time::Time()
{
   hour = minute = second = 0;

} // end Time constructor

// set new Time value using universal time, perform validity
// checks on the data values and set invalid values to zero
void Time::setTime( int h, int m, int s )
{
   hour = ( h >= 0 && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   second = ( s >= 0 && s < 60 ) ? s : 0;

} // end function setTime

```

fig06_03.cpp
(2 of 5)

Constructor initializes **`private`** data members to **`0`**.

`public` member function checks parameter values for validity before setting **`private`** data members.

fig06_03.cpp
(3 of 5)

```
// print Time in universal format
void Time::printUniversal()
{
   cout << setfill( '0' ) << setw( 2 ) << hour << ":"
        << setw( 2 ) << minute << ":"
        << setw( 2 ) << second;

} // end function printUniversal

// print Time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << setfill( '0' ) << setw( 2 ) << minute
        << ":" << setw( 2 ) << second
        << ( hour < 12 ? "

} // end function printStandard

int main()
{
   Time t;  // instantiate object t of class Time

```

No arguments (implicitly "know" purpose is to print data members); member function calls more concise.

Declare variable **`t`** to be object of class **`Time`**.

```
    // output Time object t's initial values
    cout << "The initial universal time is ";
    t.printUniversal();   // 00:00:00

    cout << "\nThe initial standard time is ";
    t.printStandard();    // 12:00:00 AM

    t.setTime( 13, 27, 6 );   // change time

    // output Time object t's new values
    cout << "\n\nUniversal time after setTi
    t.printUniversal();   // 13:27:06

    cout << "\nStandard time after setTime is ";
    t.printStandard();    // 1:27:06 PM

    t.setTime( 99, 99, 99 );  // attempt invalid settings

    // output t's values after specifying invalid values
    cout << "\n\nAfter attempting invalid settings:"
         << "\nUniversal time: ";
    t.printUniversal();   // 00:00:00

```

Invoke **`public`** member functions to print time.

Set data members using **`public`** member function.

Attempt to set data members to invalid values using **`public`** member function.

fig06_03.cpp
(4 of 5)

```
cout << "\nStandard time: ";
t.printStandard();    // 12:00:00 AM
cout << endl;

return 0;

```

fig06_03.cpp
(5 of 5)

fig06_03.cpp
output (1 of 1)

```
The initial universal time is 00:00:00
The initial standard time is 12:00:00 AM

Universal time after setTime is 13:27:06
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Universal time: 00:00:00
Standard time: 12:00:00 AM
```

Data members set to **0** after attempting invalid settings.