哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

高级语言程序设计
High-level Language Programming

Lecture 9a Objects and Classes

Yitian Shao (shaoyitian@hit.edu.cn)
School of Computer Science and Technology

Objects and Classes

Course Overview

- Basic of object and class
- Construct a class
- Constructors
- Class properties

Try it yourself on https://www.onlinegdb.com

9.1 Basics of Object and Class

- An object is a component of a program that knows
  - how to perform certain actions
  - how to interact with other parts of the program.

- An object consists of
  - one or more data values, which define the state or properties of the object
  - functions that can be applied to the object
    The functions associated with an object represent what can be done to the object and how the object behaves.

9.1 Basics of Object and Class

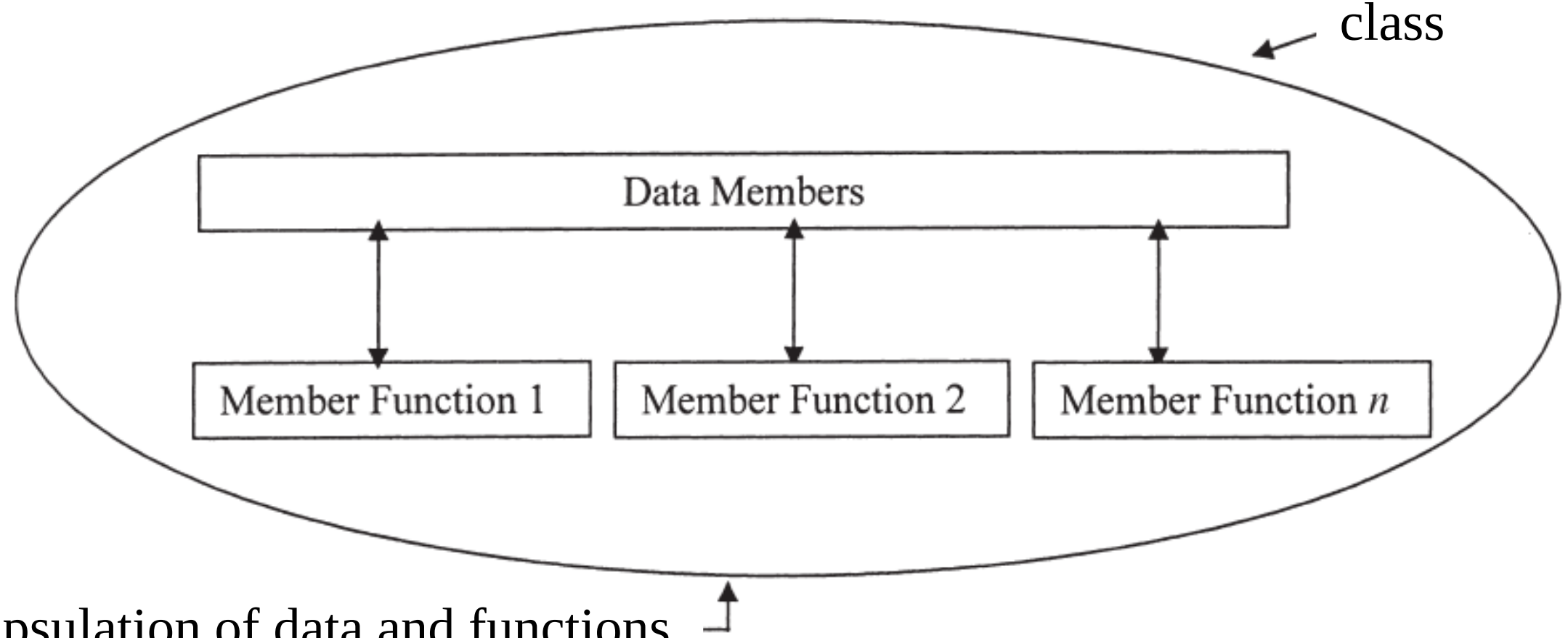
- Example: a computer game
  - A player will have data values to represent certain attributes, e.g. the state of their health or the weapons they possess.
  - A player must be able to perform functions such as walk, run, attack an enemy, and rescue the fair maiden to win.

9.1 Basics of Object and Class

- A **class** is a general category that defines:
  - The characteristics that an object of that category contains. The characteristics are called properties or class data members.
  - The functions that can be applied to objects of that category. The functions are also called class member functions or methods.

- **encapsulation**
  - A class defines both the type of data and the operations that can be applied to that data.
  - Including both the data and functions into one unit, the class, is called encapsulation.

9.1 Basics of Object and Class



9.1 Basics of Object and Class

- In our computer adventure game a player class may be defined as:

```
class player
{
// functions:
      walk()
      run()
      jump()
      attack()
      rescue()
      ...
//data:
      state_of_health
      type_of_weapon
      ...
} ;
```

Note:
•This is only a pseudo code representation of the class used for explanatory purposes only.

```
class dragon
{
//functions:
      walk()
      spit_fire()
      use_claws()
      use_tail()
      die()
      ...

//data:
      size
      number_of_claws
      state_of_health
      ...
}
```

9.1 Basics of Object and Class

- A class and instance of the class
  - A class is a **blueprint/template** that can be used to create many instances of the class.

  - An instance of a class is the actual object, created from the template, that can be manipulated by the member functions of the class.

  - The difference between a class and an **instance of a class** (an object) is like the difference between a noun and a proper noun.
    - person is a noun, John Smith is a proper noun
    - Person is the class and John Smith is the object.

9.1 Basics of Object and Class

- Example: the adventure game
  - The following statement creates an instance of a dragon:

```
class dragon george( 10, 4 ) ;
```

  - A class member function is called using the member selection operator (a dot):

```
george.spit_fire() ;
```

•Here george is an instance or an object of the dragon class.
•The data in the parentheses represent initial values for some of the data members of george, e.g. size and number_of_claws.

9.1 Basics of Object and Class

- Example: the adventure game (continued)
  - Any number of dragon objects can be created.

```
class dragon ivan( 6, 2 ), baby( 1, 0 ) ;
```

  - Each dragon can perform different functions independent of each other.

```
ivan.use_claws() ;    // ivan attacks with claws.
baby.die() ;          // baby dragon dies. sorry!
george.spit_fire() ; // george attacks with fire.
```

9.1 Basics of Object and Class

- To summarize:
  - An object is an instance of a class
  - An object has:
    - an identity, i.e. its name
    - a state, i.e. its data members
    - a behavior, i.e. its member functions.

9.1 Basics of Object and Class

- **An example: A student class**
  - Some of the properties (class data members) that define a student are the name, the address, the student number, the date of birth, the course and the course mark.
  - Some of the functions (class member functions or methods) that can be performed on a student are to update the student's course mark, display a progress report and so on.

```
class student
{
// functions:
      update_mark()
      display_report()

// data:
      name
      address
      student_number
      date_of_birth
      course_code
      course_marks
```

•A student object is a specific instance of the student class.

•Students in a course such as *John Smith or Liu Wei are instances of the student* Class.

9.1 Basics of Object and Class

- **An example: A bank account class**
  - A bank account has an account number and a balance.
  - Operations on a bank account class are to open the account with an amount of money and to deposit and withdraw money from the account.
    - These operations will be represented by class member functions such as open (amount), withdraw(amount) and deposit(amount).

```
class bank_account
{
// functions:
      open( amount )
      deposit( amount )
      withdraw( amount )
// data:
      account_number
      balance
}
```

```
class bank_account my_bank_account ;

my_bank_account.withdraw( 20.00 ) ;
```

9.2 Constructing a class in C++

- *Abstraction*
  - Each software object is a simplification of its real world counterpart.
  - In the **computer game**, each software object behaves only in some respects as its real-world counterpart.
    - there may be no need for a player to eat, drink or sleep in the game.
  - The **bank account** class describes only the characteristics and operations that are relevant for the purposes of the program.

9.2 Constructing a class in C++

- Program Example

```
2  // Demonstration of a C++ class.
3  #include <iostream> // Required later for input-output.
4  #include <iomanip>  // Required later for manipulators.
5  using namespace std ;
6
7  class bank_account
8  {
9  public:
10   void open( int acc_no, double initial_balance ) ;
11   void deposit( double amount ) ;
12   void withdraw( double amount ) ;
13   void display_balance() ;
14 private:
15   int account_number ;
16   double balance ;
17 } ;
```

The class member function prototypes.

The class data members.

← Don't forget the semicolon.

- On line 7, following the keyword class, the class is given the name bank_account
- Lines 10 ~13 declare the member functions of the class
- Lines 15 and 16 declare the data members of the class.
- The members of the class are divided into private and public members.
  - The keywords *private* and *public* specify the access control level for the data and function members of the class.

9.2 Constructing a class in C++

- **Information hiding**
  - The private data members are accessible only to member functions of the class and unavailable to any functions that are not members of the class.

- Members declared with **public access** are accessible in any part of a program.
  - A **public member** function can be called from any part of a program, while a **private member** function can only be called from within member functions of the same class.
  - The **public member** functions are known as the ***public interface*** in class.

9.2 Constructing a class in C++

Class

Members of a class are **private** by default.

Declared using the **class** keyword.

Normally used for **data abstraction** and **inheritance**.

Syntax:

```
class class_name {
    data_member;
    member_function;
};
```

C++ Structure (Lecture 6)

Members of a structure are **public** by default.

Declared using the **struct** keyword.

Normally used for the **grouping of different datatypes**.

Syntax:

```
struct structure_name {
    structure_data_member;
    structure_member_function;
};
```

```
class student_rec
{
    public:
        int number;
        float scores[5];
};

int main()
{
    class student_rec student1, student2;
}
```

```
struct student_rec
{
  int number ;    // Student number.
  float scores[5] ;  // Scores on five tests.
} ;
```

```
struct student_rec student1, student2 ;
```

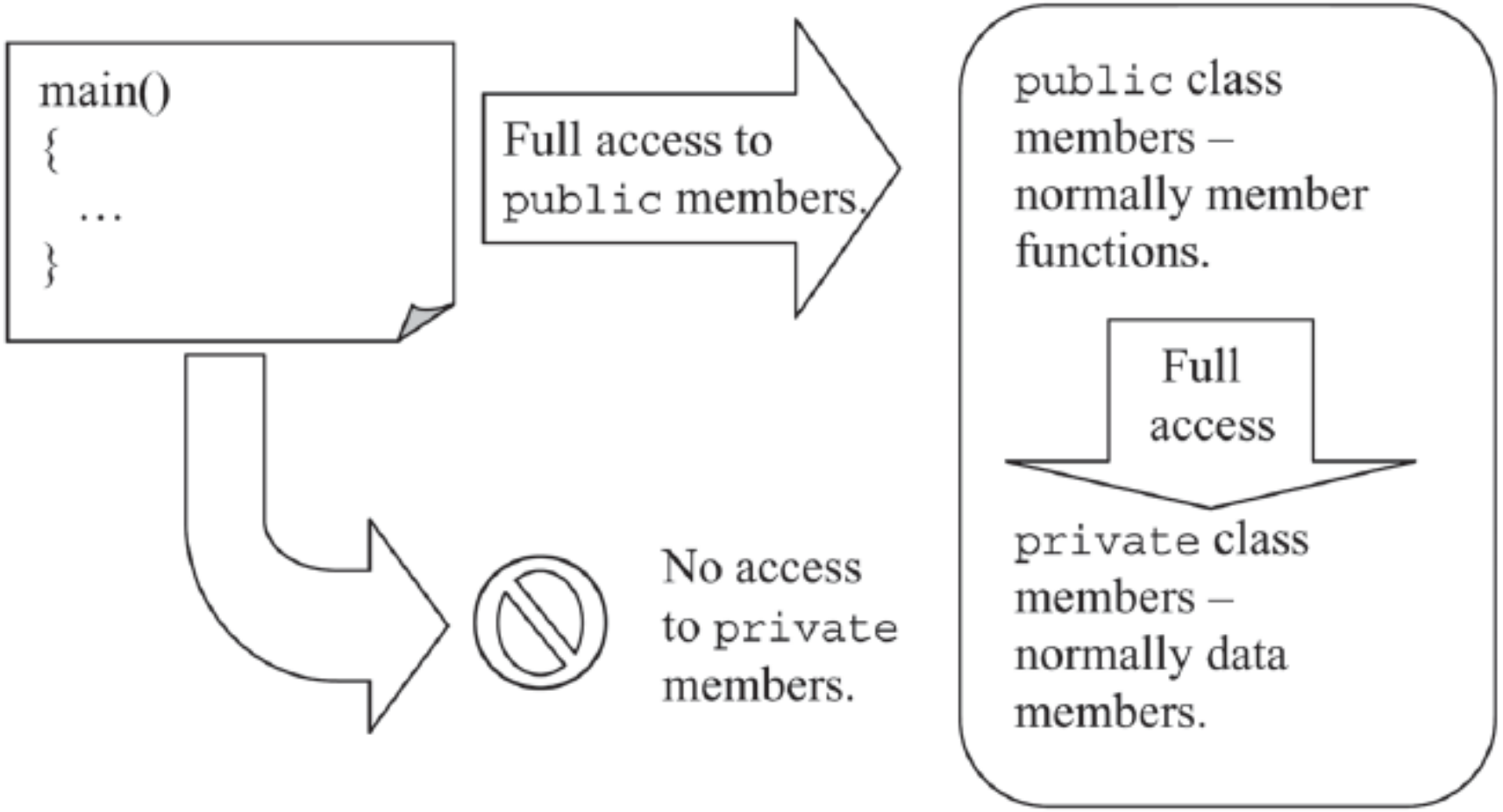
9.2 Constructing a class in C++

- The general format of a class

```
class class_name
{
public:
// Details of the public interface of the class.
private:
// Private member functions and data members.
} ;
```

  - Generally, **data members of a class are mostly private** and the **member functions of the class are mostly public**.
  - The public section is usually placed at the start of the class before the private section.

9.2 Constructing a class in C++



9.2 Constructing a class in C++

- The class member functions must be defined

```
19 void bank_account::open( int acc_no, double initial_balance )
20 {
21   account_number = acc_no ;
22   balance = initial_balance ;
23 }
24
25 void bank_account::deposit( double amount )
26 {
27   balance += amount ;
28 }
29
30 void bank_account::withdraw( double amount )
31 {
32    balance -= amount ;
33 }
34
35 void bank_account::display_balance()
36 {
37    cout << "Balance in Account " << account_number << " is "
38         << fixed << setprecision( 2 )
39         << balance << endl ;
40 }
```

The class function definitions.

9.2 Constructing a class in C++

- A class member function has the general format:

```
return_type class_name::function_name( parameter list )
{
// function statements.
}
```

  - The scope resolution operator :: is used here to specify that a function is a member of a class.

9.2 Constructing a class in C++

```
class bank_account
{
public:
  void open( int acc_no, double initial_balance ) ;
  void deposit( double amount ) ;
  void withdraw( double amount ) ;
  void display_balance() ;
private:
  int account_number ;
  double balance ;
} ;
```

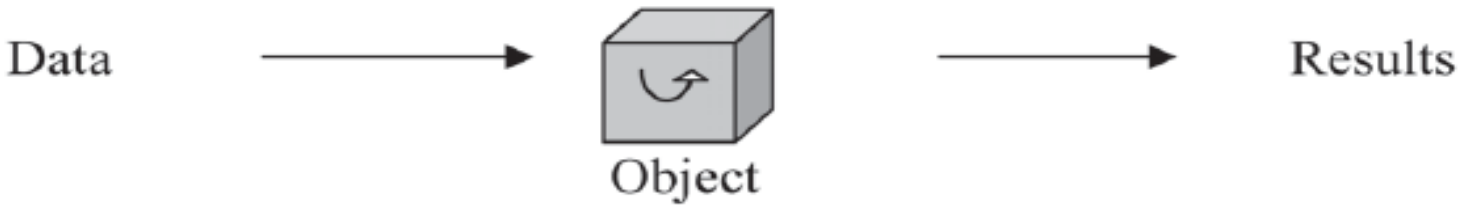
```
42 main()
43 {
44   class bank_account my_account ; // my_account is an object
45                                   // of class bank_account.
46
47   my_account.open( 123, 10.54 ) ; // Open account 123 with 10.54
48
49   my_account.display_balance() ;  // Display account details.
50
51   my_account.deposit( 10.50 ) ;   // Deposit 10.50
52
53   my_account.display_balance() ;
54
55   my_account.withdraw( 20.04 ) ;  // Withdraw 20.04
56
57   my_account.display_balance() ;
58 }
```

- To use the *bank_account* class, place the class and member function definitions before *main*().
- The objects of the class are defined and used in *main*().
- Line 44 creates a *bank_account* object called *my_account*
- Line 47 initializes the data members of the class by calling the member function *open*() with values for the *account_no* and *balance*

```
Balance in Account 123 is 10.54
Balance in Account 123 is 21.04
Balance in Account 123 is 1.00
```

9.2 Constructing a class in C++

- *Abstract Data Type (ADT)* The implementation details of the **abstract data types** are **hidden** from the programmer
  - The bank account class defines a new data type that is not built into the C++ language.
    - Once written, it is **not necessary to know the details of how each of the class member functions work**.
    - know how to call the public member functions of the class such as open(), deposit()and withdraw().



9.3 Constructors

- **Constructor**
  - **Constructor is a class member function that has the same name as the class**
  - Automatically called when an object of the class is created
  - Can be used to provide the private data members of the object with initial values.

  - Example:
    - The private *account_number* and *balance* are assigned their initial values in *open*().
    - a constructor is added to the *bank_account* class in place of the *open*() member function.

9.3 Constructors

- Program Example

```
7  class bank_account
8  {
9  public:
10    bank_account( int acc_no, double initial_balance ) ;
11    void deposit( double amount ) ;
12    void withdraw( double amount ) ;
13    void display_balance() ;
14 private:
15    int account_number ;
16    double balance ;
17 } ;
18
19 bank_account::bank_account( int acc_no, double initial_balance )
20 {
21   account_number = acc_no ;
22   balance = initial_balance ;
23 }
```

The class constructor has the same name as the class and has no return type.

The constructor is never explicitly called and therefore **cannot** return a value.
That's why a class constructor has **no return type**, not even void.

9.3 Constructors

- Rewriting main() to make use of the class constructor

```
42 main()
43 {
44   bank_account my_account( 123, 10.54 ) ;
45
46   my_account.display_balance() ;
47
48   my_account.deposit( 10.50 ) ;
49
50   my_account.display_balance() ;
51
52   my_account.withdraw( 20.04 ) ;
53
54   my_account.display_balance() ;
55 }
```

The class constructor bank_account() is automatically called, assigning the account member a value of 123 and the balance a value of 10.54.

•Line 44 of this program creates an object called *my_account* with these values:

my_account	
account_no	balance
123	10.54

```
Balance in Account 123 is 10.54
Balance in Account 123 is 21.04
Balance in Account 123 is 1.00
```

9.3 Constructors

- A *default class constructor* is a constructor that has no parameters.

```
7  class bank_account
8  {
9  public:
10    bank_account() ;
11    bank_account( int acc_no, double initial_balance ) ;
12    void deposit( double amount ) ;
13    void withdraw( double amount ) ;
14    void display_balance() ;
15 private:
16    int account_number ;
17    double balance ;
18 } ;
```

The default class constructor has no parameters.

9.3 Constructors

- Program Example

```
20 bank_account::bank_account()
21 {
22   account_number = 0 ;
23   balance = 0.0 ;
24 }
25
26 bank_account::bank_account( int acc_no, double initial_balance )
27 {
28   account_number = acc_no ;
29   balance = initial_balance ;
30 }
31
32 void bank_account::deposit( double amount )
33 {
34   balance += amount ;
35 }
```

9.3 Constructors

- Program Example···continued

```
49 main()
50 {
51   bank_account my_account ;
52
53   my_account.display_balance() ;
54
55   my_account.deposit( 10.50 ) ;
56
57   my_account.display_balance() ;
58
59   my_account.withdraw( 20.04 ) ;
60
61   my_account.display_balance() ;
62 }
```

The default class constructor bank_account() is automatically called assigning the account number a value of 0 and the balance a value of 0.0

my_account	
account_no	balance
0	0.0

```
Balance in Account 0 is 0.00
Balance in Account 0 is 10.50
Balance in Account 0 is -9.54
```

9.3 Constructors

- Constructors can be overloaded (Lecture 8 function overloading)
  - There can be several constructors within a class, each with a different number of parameters

```
7  class bank_account
8  {
9  public:
10   bank_account() ;
11   bank_account( int acc_no ) ;
12   bank_account( int acc_no, double initial_balance ) ;
13   void deposit( double amount ) ;
14   void withdraw( double amount ) ;
15   void display_balance() ;
16 private:
17   int account_number ;
18   double balance ;
19 } ;
```

9.3 Constructors

```
21 bank_account::bank_account()
22 {
23   account_number = 0 ;
24   balance = 0.0 ;
25 }
26
27 bank_account::bank_account( int acc_no )
28 {
29   account_number = acc_no ;
30   balance = 0.0 ;
31 }
32
33 bank_account::bank_account( int acc_no, double initial_balance )
34 {
35   account_number = acc_no ;
36   balance = initial_balance ;
37 }
```

```
56 main()
57 {
58   bank_account my_account( 123 ) ;
59
60   my_account.display_balance() ;
61 }
```

- Only one argument given when the object *my_account* is being created
- the class constructor on lines 27 to 31 is called, assigning 123 to the account number and 0.0 to the balance.

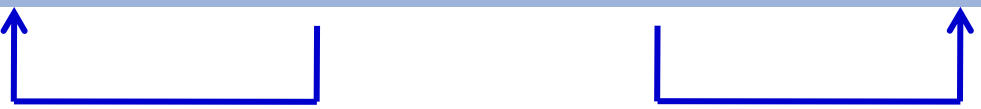
my_account	
account_no	balance
123	0.0

9.3 Constructors

- A data member initialization list is frequently used in constructors in place of assignment statements.

```
33 bank_account::bank_account( int acc_no, double initial_balance )
34 {
35   account_number = acc_no ;
36   balance = initial_balance ;
37 }
```

```
bank_account::
bank_account( int acc_no, float initial_balance ) :
  account_number( acc_no ), balance( initial_balance )
{
}
```



- No statements left in this constructor
- The chain brackets {} are still required

9.3 Constructors

- Constructors are functions, default arguments can also be used in constructors

```
7  class bank_account
8  {
9  public:
10   bank_account() ;
11   bank_account( int acc_no, double initial_balance = 0.0 ) ;
12   void deposit( double amount ) ;
13   void withdraw( double amount ) ;
14   void display_balance() ;
15 private:
16   int account_number ;
17   double balance ;
18 } ;
```

Default argument in a constructor.

9.3 Constructors

```
20 bank_account::bank_account()
21 {
22   account_number = 0 ;
23   balance = 0.0 ;
24 }
25
26 bank_account::
27 bank_account( int acc_no, double initial_balance ) :
28       account_number( acc_no ), balance( initial_balance )
29 {}
30
31 void bank_account::deposit( double amount )
32 {
33   balance += amount ;
34 }
```

Can use an initialization list and assignment statements in a constructor.

9.3 Constructors

```
50   bank_account account1 ( 1 ) ;
51   // The constructor starting on line 26 is called,
52   // assigning the account number a value of 1 and the balance
53   // the default value of 0.0
54
55   bank_account account2 ( 2, 10.55 ) ;
56   // The constructor starting on line 26 is again called,
57   // assigning the account number a value of 2 and the balance
58   // a value of 10.55
59
60   bank_account account3 ;
61   // The default constructor on lines 20 to 24 is called,
62   // assigning the account number a value of 0 and the balance
63   // a value of 0.00
```

```
Balance in Account 1 is 0.00
Balance in Account 2 is 10.55
Balance in Account 0 is 0.00
```

9.4 Class properties

- static class data members
  - A static class data member is independent of all the objects that are created from that class.
  - Only one copy of a static data member exists and is shared by all objects of a particular class.
  - The value of the static data member is therefore the same for all the class objects.
  - If even one of the objects of a class modifies the value of a static data member, then the value of the static data member changes for every object of that class.

9.4 Class properties

```
7   class bank_account
8   {
9   public:
10    bank_account() ;
11    bank_account( int acc_no ) ;
12    bank_account( int acc_no, double initial_balance ) ;
13    void deposit( double amount ) ;
14    void withdraw( double amount ) ;
15    void display_balance() ;
16  private:
17    static int next_account_number ;
18    int account_number ;
19    double balance ;
20  } ;
```

static class data member.

9.4 Class properties

```
22 bank_account::bank_account()
23 {
24   account_number = next_account_number++ ;
25   balance = 0.0 ;
26 }
27
28 bank_account::bank_account( int acc_no )
29 {
30   account_number = acc_no ;
31   balance = 0.0 ;
32 }
33
34 bank_account::bank_account( int acc_no, double initial_balance )
35 {
36   account_number = acc_no ;
37   balance = initial_balance ;
38 }
```

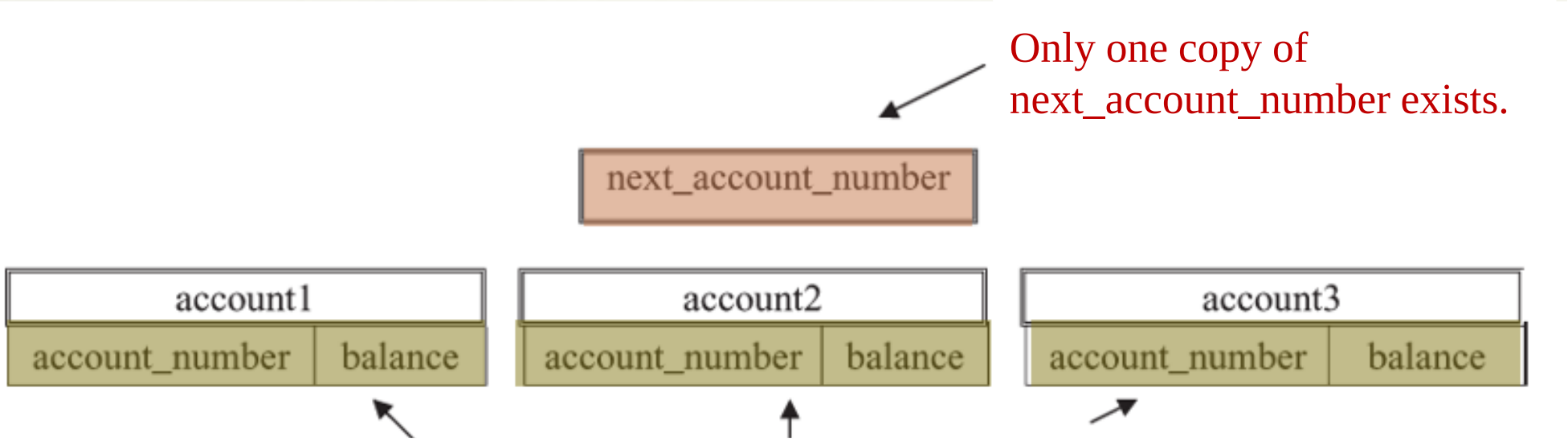
9.4 Class properties

```
57 int bank_account::next_account_number = 1 ;
58
59 main()
60 {
61   bank_account account1, account2, account3 ;
62
63   account1.deposit( 25.50 ) ;
64   account2.deposit( 30.50 ) ;
65   account3.deposit( 10.00 ) ;
66   account1.withdraw( 20.04 ) ;
67
68   account1.display_balance() ;
69   account2.display_balance() ;
70   account3.display_balance() ;
71 }
```

- A static data member is initialized outside the class and outside main()
- *next_account_number* occurs only once and is shared by all objects of the class and bank account objects.

```
Balance in Account 1 is 5.46
Balance in Account 2 is 30.50
Balance in Account 3 is 10.00
```

9.4 Class properties



- *class variable:*
  - Associated with a class rather than an object
- *instance variables*
  - Associated with instances of the class and are called.

9.4 Class properties

- A class member function can return a value using a return statement

```
7  class bank_account
8  {
9  public:
10   bank_account() ;
11   bank_account( int acc_no ) ;
12   bank_account( int acc_no, double i
13   void deposit( double amount ) ;
14   void withdraw( double amount ) ;
15   void display_balance() ;
16   double get_balance() ;
17 private:
18   static int next_account_number ;
19   int account_number ;
20   double balance ;
21 } ;
```

```
46 void bank_account::withdraw( double amount )
47 {
48    balance -= amount ;
49 }
50
51 void bank_account::display_balance()
52 {
53    cout << "Balance in Account " << account_number << " is "
54         << fixed << setprecision( 2 )
55         << balance << endl ;
56 }
57
58 double bank_account::get_balance()
59 {
60   return balance ;
61 }
```

9.4 Class properties

```
65 main()
66 {
67   // Create four accounts.
68   bank_account account1, account2, account3, account4 ;
69
70   // Put some money into each account.
71   account1.deposit( 125.55 ) ;
72   account2.deposit( 130.75 ) ;
73   account3.deposit( 100.25 ) ;
74   account4.deposit( 300.45 ) ;
76   // Calculate the total amount on deposit.
77   float total_balances = account1.get_balance() +
78                          account2.get_balance() +
79                          account3.get_balance() +
80                          account4.get_balance() ;
81
82   cout << "Total Balances = " << fixed << setprecision( 2 )
83        << total_balances << endl ;
84 }
```

- *inspector or accessor function*

The member function get_balance(), *Inspector* functions allow private data members of a class to be inspected from outside the class.

- *mutator function*

The class member functions deposit() and withdraw(). *Mutator functions change the values* of the private data members of a class.

9.4 Class properties

- class public interface: a list of the member functions of the class and how they can be used.
  - The public interface starts at public and ends at the keyword private.
  - A programmer using a class should only have to read the public interface in order to use the class;
  - To make the public interface easier to read, comments should be included explaining the purpose of the functions, their parameters and their return values.
- *class implementation:* details in the private section and in the member functions

9.4 Class properties

```
1   class bank_account
2   {
3   public:
4   // Constructors.
5     bank_account() ;
6     // Purpose: Default class constructor.
7     //          First instance of the class is assigned account number
8     //          1, the second instance account number 2, and so on.
9     //          The account balance is set to 0 for each instance.
10
11    bank_account( int acc_no ) ;
12    // Purpose  : Class constructor to set the account number
13    //            of an instance of the class to the parameter value.
14    // Parameter: An account number.
15
16    bank_account( int acc_no, double initial_balance ) ;
17    // Purpose   : Constructor to set the account number and balance of a
18    //             class instance to specified values.
19    // Parameters: An account number and a balance.
```

Public interface of the class.

9.4 Class properties

- **Separation of class interface and class implementation**
  - It is not possible in C++ to completely separate the public interface of a class from its implementation.
  - In C++, the class declaration is normally placed in a *header file (e.g.* bank_ac.h) and the member functions are normally placed in a separate file (e.g. bank_ac.cpp).
  - file *bank_ac.h*:

```
// Declaration of bank_account class.

#if !defined BANK_AC_H
#define BANK_AC_H
```

The lines beginning with # are standard preprocessor directives used to prevent multiple inclusions of the header file into a program.

9.4 Class properties

- *bank_ac.h*:

```
// Declaration of bank_account class.

#if !defined BANK_AC_H
#define BANK_AC_H
class bank_account
{
public:
// Constructors.
  bank_account() ;
  // Purpose: Default class constructor.
  //          First instance of the class is assigned account number
  //          1, the second instance account number 2, and so on.
  //          The account balance is set to 0 for each instance.
...
private:
  static int next_account_number ;
  int account_number ;
  double balance ;
} ;

#endif
```

9.4 Class properties

- Placing the member functions into *bank_ac.cpp*:

```
// Member function definitions and static data member initialisation
// code for bank account class.
#include <iostream>
#include <iomanip>
#include "bank_ac.h"
using namespace std ;

bank_account::bank_account()
{
  account_number = next_account_number ++ ;
  balance = 0.0 ;
}
...
int bank_account::next_account_number = 1 ;
```

9.4 Class properties

- Using these two files, we can write:

```
2  // Demonstration of using class header and source code files.
3  #include <iostream>
4  #include <iomanip>
5  #include "bank_ac.h"
6  #include "bank_ac.cpp"
7  using namespace std ;
8
9  main()
10 {
11   bank_account my_account ;
12
13   my_account.deposit( 12.34 ) ;
14   my_account.display_balance() ;
15 }
```

- Lines 5 and 6 are preprocessor directives that incorporate the bank account header and source files into the program.
- The quotes around the file names on lines 5 and 6 tell the compiler that the files to be included are in the same directory as the program.

高级语言程序设计
High-level Language Programming

Lecture 9b Class Inheritance

Class Inheritance

Course Overview

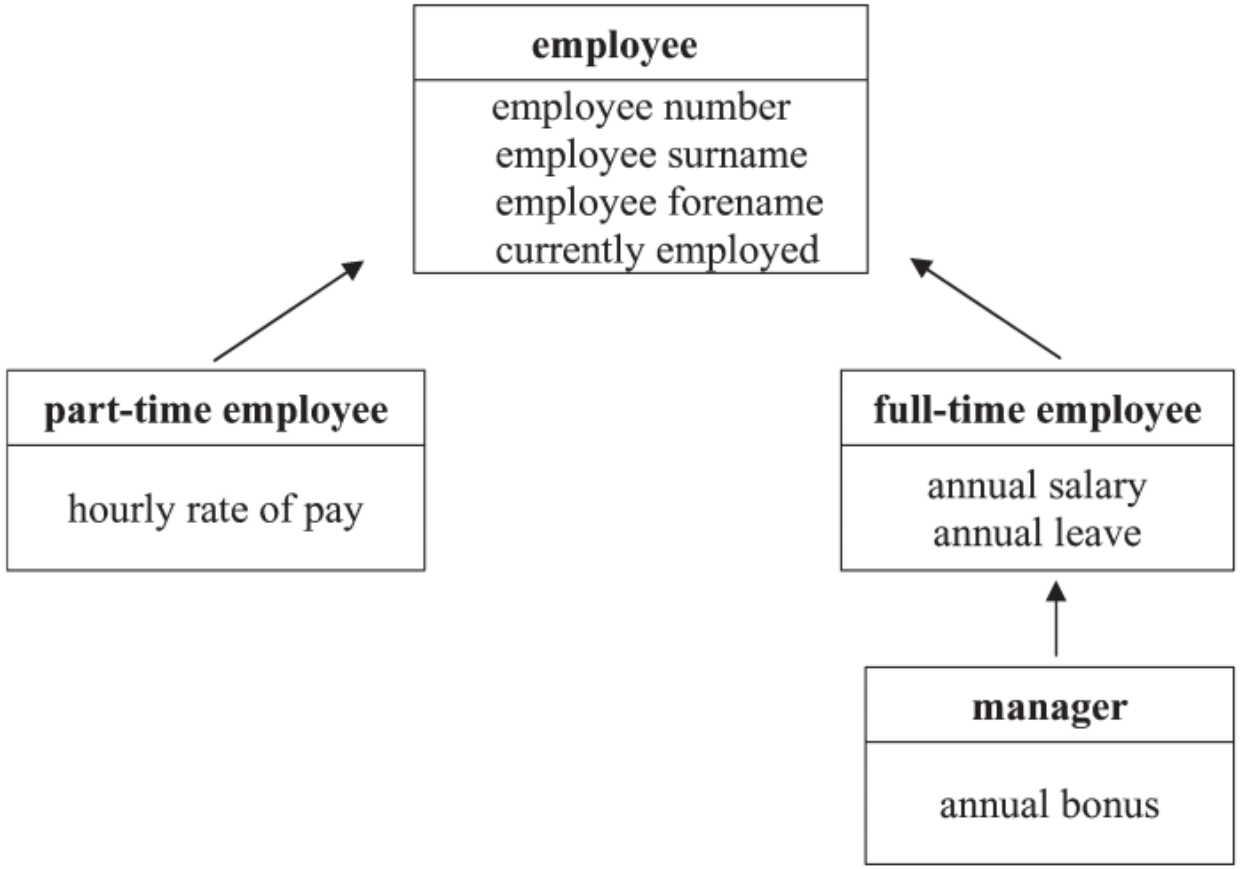
- Inheritance and its syntax
- Inheritance Constructor
- Protected class members
- Types of inheritance

Try it yourself on https://www.onlinegdb.com

9.1 Inheritance and its syntax

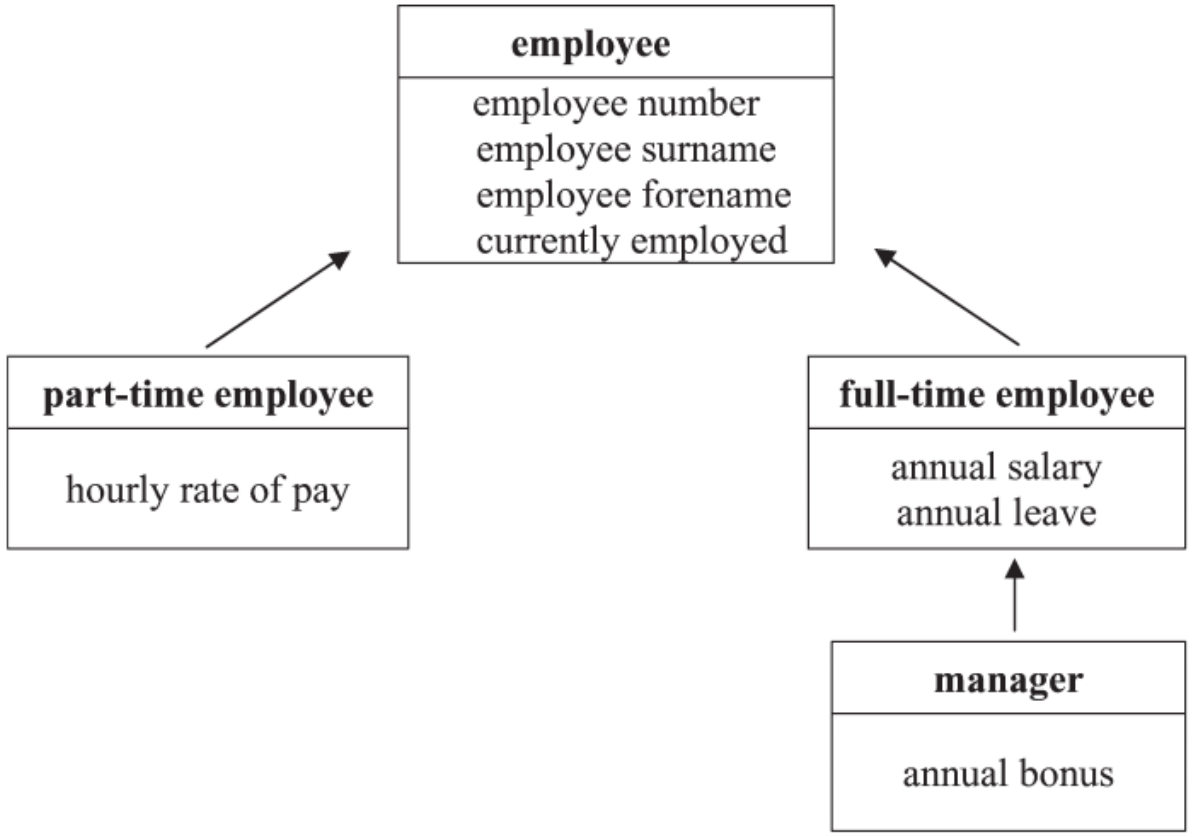
- *Inheritance* is one of the fundamental concepts of object-oriented programming.
- Inheritance allows a new class to be constructed **using an existing class as a basis**.
- The new class *incorporates* or *inherits* the data members and member functions of the existing class.
- Additional data members and member functions can be added to the new class, there by extending the existing class.
- The existing class is known as the *base* class.
- The new class is known as the *derived* or *inherited* class.

9.1 Inheritance and its syntax



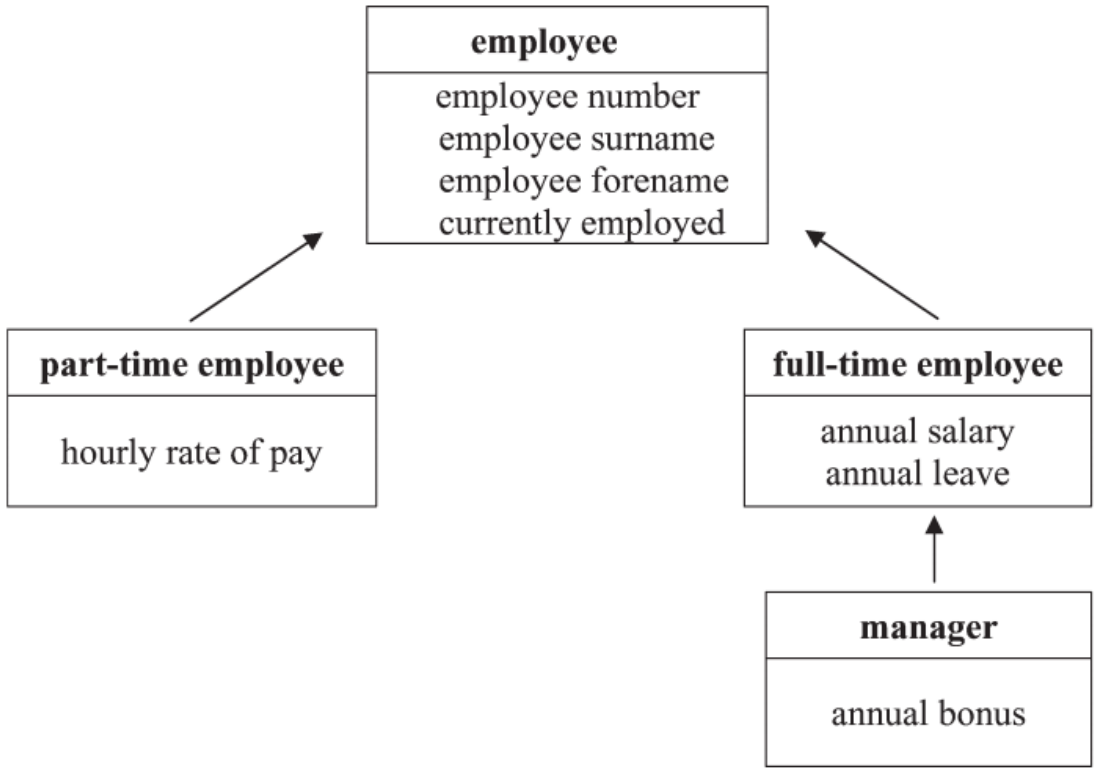
- How to implement this inheritance Hierarchy?
- Do we need to define employee number, surname, forename, employed for each class?

9.1 Inheritance and its syntax



- Employees in the company are either part-time or full-time. The **employee number and name are common** to both types of employee.
- Whether the person is **currently employed or not is also common** to both types of employees.
- Part-time employees are paid an hourly rate.
- full-time employees are on a salary and are allowed an annual leave.
- Managers are also full-time employees, who in addition to their annual salary and paid leave are also paid an annual bonus.

9.1 Inheritance and its syntax



- Employees class is known as the base.
  - the base class is not modified in any way and is simply used as a basis for writing the derived class.
  - This is called **reusability** (the base class is being reused) and is a major goal of object-oriented programming.
- The derived class can, in turn, be used as a base class from which other classes may be derived, thus creating a **class hierarchy**.

9.1 Inheritance and its syntax

- In its simplest form, inheritance is used by first defining the base class:

```
class base
{
// class data members and
// class member functions.
} ;
```

- Then the derived classes are defined (this is the same as defining any class):

Note the keyword **public**

```
class inherited : public base
{
// additional class data members for this class and
// additional class member functions for this class.
} ;
```

9.1 Inheritance and its syntax

```
3   #include<iostream>
4   #include<string>
5   using namespace std ;
6
7   class employee   // Base class.
8   {
9   public:
10    employee() ;
11    void display_data() ;
12    void left() ;
13  private:
14    unsigned int employee_number ;
15    string surname ;
16    string forename ;
17    bool currently_employed ;
18  } ;
19
```

```
20  // employee member functions.
21  employee::employee()
22  {
23    cout << endl << "Enter Employee Number: " ;
24    cin >> employee_number ;
25    cout << "Enter Employee Name: " ;
26    cin >> surname >> forename ;
27    currently_employed = true ;
28  }
29
```

9.1 Inheritance and its syntax

```
30  void employee::left()
31  {
32    currently_employed = false ;
33  }
34
35  void employee::display_data()
36  {
37    if ( currently_employed )
38       cout << "Currently Employed" ;
39    else
40       cout << "Not Currently Employed" ;
41    cout << endl << "Employee Number: " << employee_number << endl
42         << "Name: " << surname << ' ' << forename << endl ;
43  }
44
```

9.1 Inheritance and its syntax

```
45  class part_time : public employee
46  {  // part_time is a kind of employee.
47  public:
48    part_time() ;
49    void display_data() ;
50  private:
51    double hourly_rate ;
52  } ;
53
```

- derived class.
- Inheritance creates a hierarchy of related classes (types) which share code and interface.
- code reusability

class part_time : public employee

Syntax:

class *DerivedClassName* : access-level *BaseClassName*

access-level specifies the type of derivation (private by default)

public

protected

9.1 Inheritance and its syntax

```
45  class part_time : public employee
46  {  // part_time is a kind of employee.
47  public:
48    part_time() ;

54  // part_time member functions.
55  part_time::part_time()
56  {
57    cout << "Enter Hourly Rate: " ;
58    cin >> hourly_rate ;
59  }
60
61  void part_time::display_data()
62  {
63    employee::display_data() ;
64    cout << "Hourly Rate: " << hourly_rate << endl ;
65  }
66
```

A derived class can override methods defined in its parent class. With overriding,

- the method in the subclass has the identical signature to the method in the base class.
- a subclass implements its own version of a base class method.

9.1 Inheritance and its syntax

```
67  class full_time : public employee
68  {  // full_time is a kind of employee.
69  public:
70    full_time() ;
71    void display_data() ;
72  private:
73    double annual_salary ;
74    int annual_leave ;
75  } ;
76
77  // full_time member functions.
78  full_time::full_time()
79  {
80    cout << "Enter Salary: " ;
81    cin >> annual_salary ;
82    cout << "Enter Annual Leave (in days): " ;
83    cin >> annual_leave ;
84  }
85
```

```
86  void full_time::display_data()
87  {
88    employee::display_data() ;
89    cout << "Salary: " << annual_salary << endl ;
90    cout << "Annual Leave: " << annual_leave << endl ;
91  }
```

9.1 Inheritance and its syntax

```
92
93  class manager : public full_time
94  {  // manager is a kind of full_time employee.
95  public:
96    manager() ;
97    void display_data() ;
98  private:
99    double bonus ;
100 } ;
101
102 // manager member functions.
103 manager::manager()
104 {
105   cout << "Enter Bonus: " ;
106   cin >> bonus ;
107 }
```

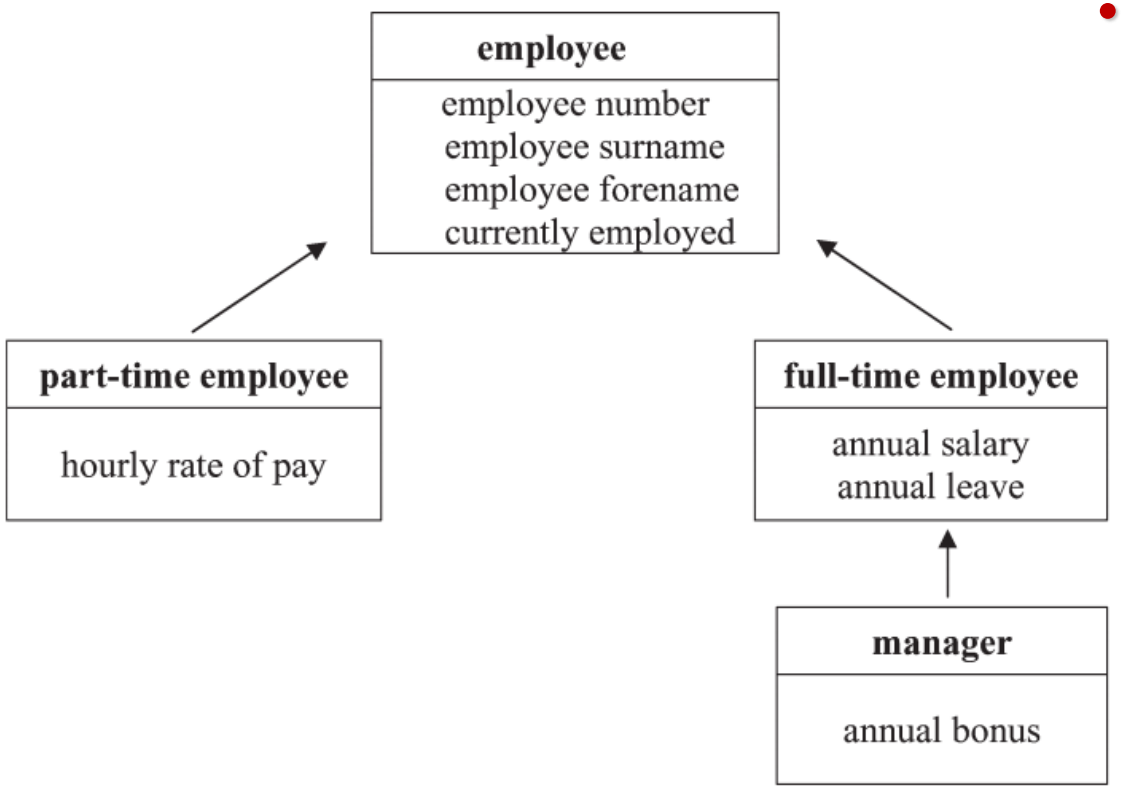
```
30  void employee::left()
31  {
32    currently_employed = false ;
33  }
```

```
108
109 void manager::display_data()
110 {
111   full_time::display_data() ;
112   cout << "Bonus: " << bonus << endl ;
113 }
114
115 main()
116 {
117   part_time pt ;
118   full_time ft ;
119   manager man ;
120
121   // Display employee data.
122   cout << endl << "Part-time Employee Data:" << endl ;
123   pt.display_data() ;
124   cout << endl<< "Full-time Employee Data:" << endl ;
125   ft.display_data() ;
126   man.left() ;
127   cout << endl << "Manager Employee Data:" << endl ;
128   man.display_data() ;
129 }
```

Note that the base class's default constructor is automatically called from the derived class.

Which display_data() is called?

9.2 Inheritance Constructor



- How to initialize the data members of objects ?
  - part_time pt(123, "Smith", "John", 5.12) ;
  - full_time ft(124, "Jones", "Mary", 21500, 21) ;
  - manager man(125, "Other", "A.N.", 32000, 30, 9500) ;

9.2 Inheritance Constructor

- To allow for initialization of an object when it is created, a constructor with a parameter list is required.
- When a derived object is created, **it should take care of the construction of the base object by calling the constructor for the base class.**
- **The base class default constructor is automatically called from the derived class**.
- When an object has initial values, the initialization of the base class object is done by **using an initialization list in the derived class constructor**.

9.2 Inheritance Constructor

```
22  // employee member functions.
23  employee::employee()
24  {
25    cout << endl << "Enter Employee Number: " ;
26    cin >> employee_number ;
27    cout << "Enter Employee Name: " ;
28    cin >> surname >> forename ;
29    currently_employed = true ;
30  }
```

```
31
32  employee::employee( unsigned int number,
33                      string sname, string fname )
34  {
35    employee_number = number ;
36    currently_employed = true ;
37    surname = sname ;
38    forename = fname ;
39  }
```

```
69  part_time::part_time()
70  {
71    cout << "Enter Hourly Rate: " ;
72    cin >> hourly_rate ;
73  }
74
75  part_time::part_time( unsigned int number,
76                        string sname, string fname,
77                        double rate )
78            : employee( number, sname, fname ), hourly_rate( rate )
79  {}
```

9.2 Inheritance Constructor

```
22  // employee member functions.
23  employee::employee()
24  {
25    cout << endl << "Enter Employee Number: " ;
26    cin >> employee_number ;
27    cout << "Enter Employee Name: " ;
28    cin >> surname >> forename ;
29    currently_employed = true ;
30  }
```

```
31
32  employee::employee( unsigned int number,
33                      string sname, string fname )
34  {
35    employee_number = number ;
36    currently_employed = true ;
37    surname = sname ;
38    forename = fname ;
```

```
101 full_time::full_time()
102 {
103   cout << "Enter Salary: " ;
104   cin >> annual_salary ;
105   cout << "Enter Annual Leave (in days): " ;
106   cin >> annual_leave ;
107 }
108
109 full_time::full_time( unsigned int number,
110                       string sname, string fname,
111                       double salary, int leave )
112             : employee( number, sname, fname ),
113               annual_salary( salary ), annual_leave( leave )
114 {}
```

9.2 Inheritance Constructor

```
109 full_time::full_time( unsigned int number,
110                       string sname, string fname,
111                       double salary, int leave )
112             : employee( number, sname, fname ),
113               annual_salary( salary ), annual_leave( leave )
114 {}
```

```
136 manager::manager()
137 {
138   cout << "Enter Bonus: " ;
139   cin >> bonus ;
140 }
141
142 manager::manager( unsigned int number,
143                   string sname, string fname,
144                   double salary, int leave, double annual_bonus )
145           : full_time( number, sname, fname, salary, leave ),
146             bonus( annual_bonus )
147 {}
```

9.2 Inheritance Constructor

```
157  part_time pt( 123, "Smith", "John", 5.12 ) ;
158  full_time ft( 124, "Jones", "Mary", 21500, 21 ) ;
159  manager man( 125, "Other", "A.N.", 32000, 30, 9500 ) ;
```

```
75  part_time::part_time( unsigned int number,
76                        string sname, string fname,
77                        double rate )
78            : employee( number, sname, fname ), hourly_rate( rate )
79  {}
```

```
32  employee::employee( unsigned int number,
33                      string sname, string fname )
34  {
35    employee_number = number ;
36    currently_employed = true ;
37    surname = sname ;
38    forename = fname ;
39  }
```

- Line 157 now initializes the object pt with the values in the parentheses.
- The constructor on lines 75 to 79 is called.
- The expression employee (number, sname, fname) in the initialization list on line 78 is an explicit call to the base class constructor on lines 32 to 39.

9.3 Protected class members

- The keywords **private** and **public** are used to control access to both the data members and the member functions of a class.
- A **private** data member or member function of a base class **cannot be accessed from a member function of a derived class**, thus ensuring that the principle of data hiding is upheld.
- In practice it can be convenient to share data between a base class and a derived class.

9.3 Protected class members

- There are two ways of doing this. One possibility is to change the access level of the relevant private members to **public**, thus allowing the derived class the required access to these members.
- this violates the principle of data hiding by making the data members available for uncontrolled modification from any part of the program.
- To get over this problem, C++ has a third level of access known as **protected** access.
- This level of access **allows derived classes (and only the derived classes) to have access to specified base class members**.

9.3 Protected class members

```
61    rectangle r( 1, 2 ) ;
62    square s( 3 ) ;
63
64    r.display_dimensions() ;
65    s.display_dimension() ;
66    r.display_area() ;
67    s.display_area() ;
```

```
class rectangle
{
public:
   rectangle( int w, int h ) ;
   int calc_area() ;
   void display_dimensions() ;
   void display_area() ;
protected:      // Available to derived classes.
   int width, height ;
};
```

```
49 void square::display_dimension()
50 {
51   cout << "Dimension of square: " << width << endl ;
52 }
53
```

- In order to display the dimension of the square, line 51 requires **access to the base class data member width**.
- This is achieved by making width a **protected data member** by placing it in the protected section of the class.

Line 61 defines a rectangle object r with sides of width 1 and height 2.

Line 62 defines a square object s with sides of size 3.

Line 64 displays the dimensions of the rectangle and line 65 displays the dimension of the square.

9.3 Protected class members

- If a base class member is declared as **private**, then this member is **not accessible outside the base class**, not even from a derived class.

- **Protected** members are accessible from within a **base class** and from within any of its **derived classes**.

- **Public** members (usually member functions) are accessible from **any part** of the program.

9.4 Types of inheritance

- The access rights of a base class member within a derived class can also be specified when the derived class is declared.
- **Public inheritance** specifies that the access rights of the inherited members will be the same as they were in the base class.

```
class square : public rectangle
```

- **Protected inheritance** makes all public members in the base class become **protected members in the derived (inherited) classes**.

- With **private inheritance**, all the public and protected members of the base class become **private in the derived (inherited) classes**.

  (By default)

9.4 Types of inheritance

- The three types of inheritance are summarized as follows:

Type of inheritance	Base class member access	Derived class member access
public	public	public
	protected	protected
	private	inaccessible
protected	public	protected
	protected	protected
	private	inaccessible
private	public	private
	protected	private
	private	inaccessible

* Protected members in the base class remain protected members in the derived class.
* Private inheritance is specified by using the keyword private.
* In all cases, whether it is public, protected or private inheritance, private members in a base class are inaccessible in a derived class.

HOMEWORK

Homework 9

- 1. Write a class declaration for each of the following classes. Include the member functions assign_data() to assign values to the data members and display_data() to display the values of the data members.

  (a) A **class** current_date with integer data members day, month, and year.

  (b) A **class** current_time with integer data members hours, minutes, and floating point data member seconds.

  Create an object of each of the above classes in main().

  Use assign_data() to assign values to the data members of each object.

  Display these values on the screen using display_data().

Homework 9

- 2. Add a default constructor to each class in exercise 1.

- 3. Modify the classes developed in exercise 1 as follows:

  (a) Add a member function increment_date() to the current_date class that adds one day to the current date.

  (b) Add a member function increment_time(s) to the current_time class that adds s seconds to the current time.

Homework 9

- 4. The following is a class for recording the position of a motorized robot.

```
class robot
{
    public:
    // Constructor with default arguments and constructor list.
    robot(float x = 0, float y = 0) : x_coord( x ), y_coord(y)
    {}
    // Inspector function to display the robot's position.
    void display_position()
    {
        cout <<  " ( "  << x_coord << " , "
             << y_coord <<  " ) "  << endl ;
    }
    private:
        float x_coord, y_coord ;
}
```

Homework 9

The following is a demonstration of the class robot (Default unit: cm)

Declare and **implement** the member functions left(), right(), forward(), back(), goto(), and return_to_base(), based on the comments (execution results) shown on the right

```
main()
{
  robot r2d2( 10.0, 8.1 ) ; // Constructor sets the initial
                             // position.

  r2d2.left( 1.3 ) ; // Move robot left 1.3 cms.
                     // New position is (8.7,8.1)
  r2d2.display_position() ;

  r2d2.back(4.21) ; // Move robot back 4.21 cms.
                    // New position is (8.7,12.31)
  r2d2.display_position() ;

  r2d2.right( 3.1 ) ; // Move robot right 3.1 cms.
                      // New position is (11.8,12.31)
  r2d2.display_position() ;

  r2d2.return_to_base() ; // Sets the position to (0,0).

  r2d2.forward( 0.3 ) // Move robot forward 3.1 cms.
                      // New position is (0,0.3).
  r2d2.display_position() ;

  r2d2.goto( 1.5, 4.5 ) ; // New position is (1.5,4.5).

  r2d2.return_to_base() ; // Move to position (0,0).
}
```

Homework 9

- 5. In the code segment below, how many data members has the object d_obj?

```
class b
{
    protected:
        int x ;
        int y ;
    private:
        int z ;
} ;

class d : public b
{
    protected:
        float a ;
    private:
        float b ;
} ;

main()
{
    d d_obj ;
}
```

Homework 9

- 6. What is the **output** from the following program segment?

```
class b
{
    public:
        b(){
            cout <<  " default constructor for b called "
                 << endl ;
        }
} ;
```

```
class d1 : public b
{
    public:
        d1(){
        cout <<  " default constructor for d1 called "
             << endl ;
        }

        d1( int v ){
            cout <<  " int parameter constructor for d1 called "
                 << endl ;
        }
} ;
```

```
class d2 : public d1
{
    public:
        d2(){
            cout <<  " default constructor for d2 called "
                 << endl ;
        }
        d2( int v ){
            cout <<  " int parameter constructor for d2 called "
                 << endl ;
        }
} ;
```

```
main()
{
    b b_obj ;
    d1 d1_obj( 2 ) ;
    d1 d1_objs[ 2 ] ;
    d2 d2_obj( 3 ) ;
}
```

Homework 9

- 7. (i) How many copies of the data member m, n and o does an object of class d inherit?

```
class a{
    private:
        int m ;
    protected:
        int n ;
    public:
        int o ;
        a(){
            cout <<  " default constructor for a called "
                 << endl ;
        }
} ;
```

```
class b : public a{
    private:
        int p ;
} ;

class c : public a{
    private:
    int q ;
} ;

class d : public b, public c{
    private:
        int r ;
} ;

main(){
    class d d_obj ;
}
```

(ii) How many times is the constructor

in the base class a called for the object d_obj?

Homework 9

- 8. Place the following two classes into a class hierarchy so that they inherit from a common base class:

```
class book
{
public:
  book() ;
private:
  string ISBN ;
  string title ;
  string author ;
  string publisher ;
  double price ;
} ;
```

```
class magazine
{
public:
  magazine() ;
private:
  char frequency ;
  string title ;
  string publisher ;
  string editor ;
  double price ;
} ;
```

Declare and implement the base class with its constructor.

Homework 9

- 9. From the “book” class of exercise 8, use inheritance to derive a “library_book” class. This class will contain details of when a book was borrowed and by whom. For identification purposes, each library user has a six-digit library number.