CPT106

C++ Programming and Software Engineering II

Lecture 3 Classes and Objects

Dr. Xiaohui Zhu

Xiaohui.zhu@xjtlu.edu.cn

Room SD535

Outline

- From structure to object
- Objects and classes
- CRC method for Object-oriented Programme Design
- Class Definition
- Declare objects
- Initialisation of data members
- Destructor

START FROM ……

- **EXAMPLE**
- Write a programme that can be used to keep information of 5 members in a team and display required information on the screen at the user's request.
  - The information for each member includes:

    ID number,

    Name,

    DoB (Date of Birth),

    Office number,

    Total hours of work per day.

Holding information – by arrays

- Data

```
long    ID[5];
char    name[5,50];
long    dob[5];
int     office[5];
int     workhour[5];
```

Each array contains information of all five employees. We use arrays mainly to hold information.

The arrays are not related to each other, and the operations to them have to be defined in the program.

Holding information – by structure

- Define structure type

```
struct employee
{
    long        ID;
    char        name[50];
    long        dob;
    int         office;
    int         workhour;
};
```

- Declare structure variable

```
employee group1[5];
employee *ptrstr=&group1[0];
```

Defining the structure **`employee`** to hold the information of each person, and structure array **`group1`** carrying the information of the 5 persons.

The structure members are logically related to each other, but the operations on them have to be defined in the programme.

Declaration of the structure student		Using student to declare variables	
1_1	struct student	2_1	student henry;
1_2	{	2_2	student tom, jerry;
1_3	int age;	2_3	student *stPtr;
1_4	char name[20];	2_4	student Y1[10];
1_5	};		

```
1. struct typename          // definition
   {
       datatype member1;
       datatype member2;
       ......
   };

2. typename var1;           // declare a single variable
   typename arr1[5];        // declare an array
   typename *ptrstr;        // declare a structure pointer
```

Structure (II)

- Initialisation of structures:
  - values on the same line

```
student henry = {20, "henry"};
```

- Access the members of a structure
  - Use a dot (.) inserted between the structure variables name and the member name. For example:

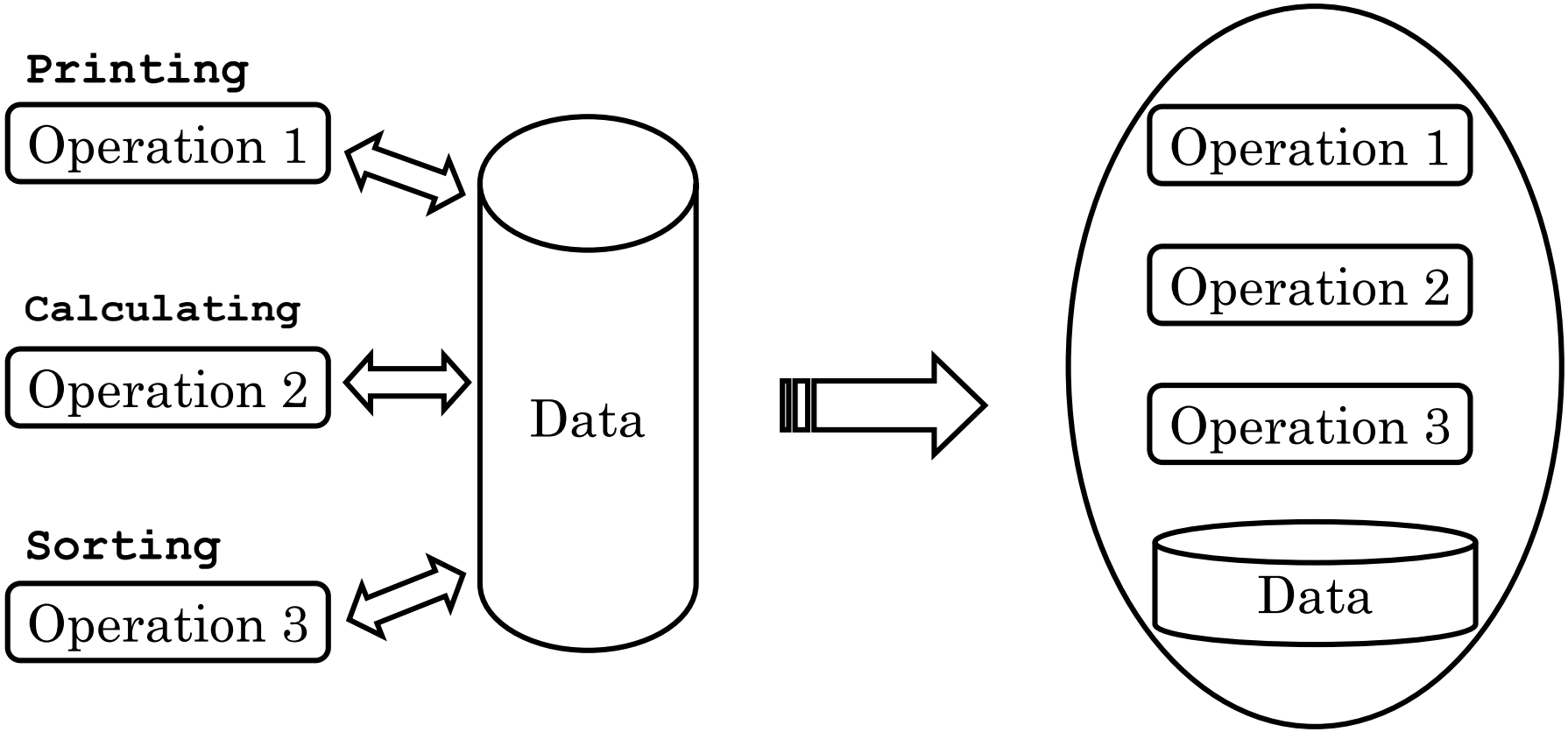
```
cout <<"Name: " <<henry.name <<"Age: " <<henry.age <<endl;
```

  - Access through a pointer which points to the structure variable. For example:

```
student *stPtr;
stPtr = &henry;
cout <<"Name: " <<stPtr->name <<"Age: " << stPtr->age <<endl;
```

CAN WE …

- Encapsulate operations with structure?



Data and operations as separated entities

Data and operations bound as single unit

The idea of OBJECT-ORIENTED Design

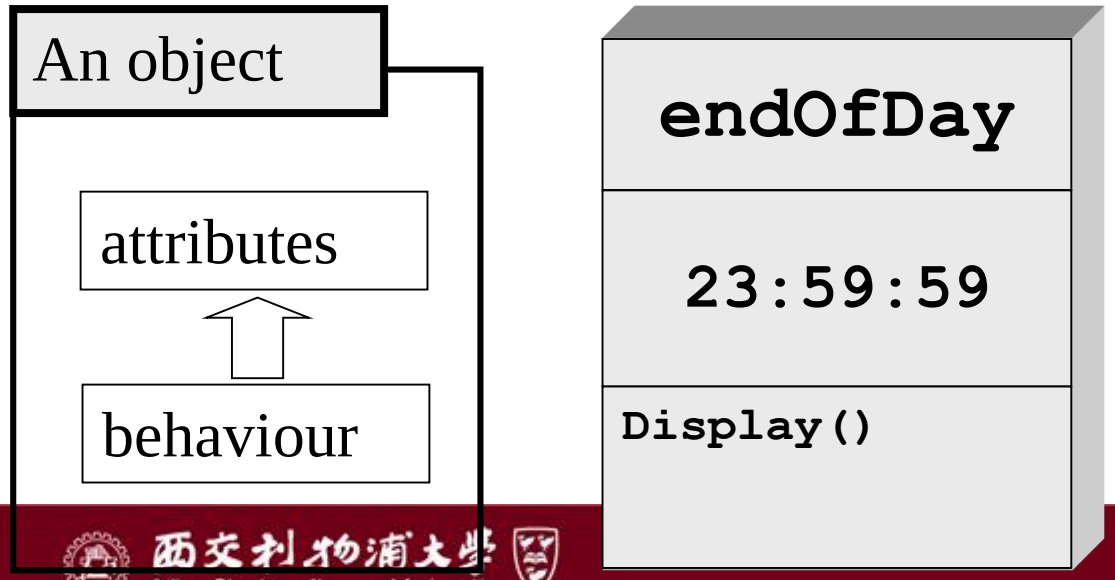
- In the example, each *employee* can be regarded as an entity or object.
  - Each object (employee) has several pieces of information.
  - Each object (employee) can perform some actions, such as tell or change its information.

- An object has:
  - Data ---- Information about an object;
  - Methods ---- Actions that an object can perform on its data.

- An object in OOD is a component in the real or conceptual world that is mapped into the programme domain.

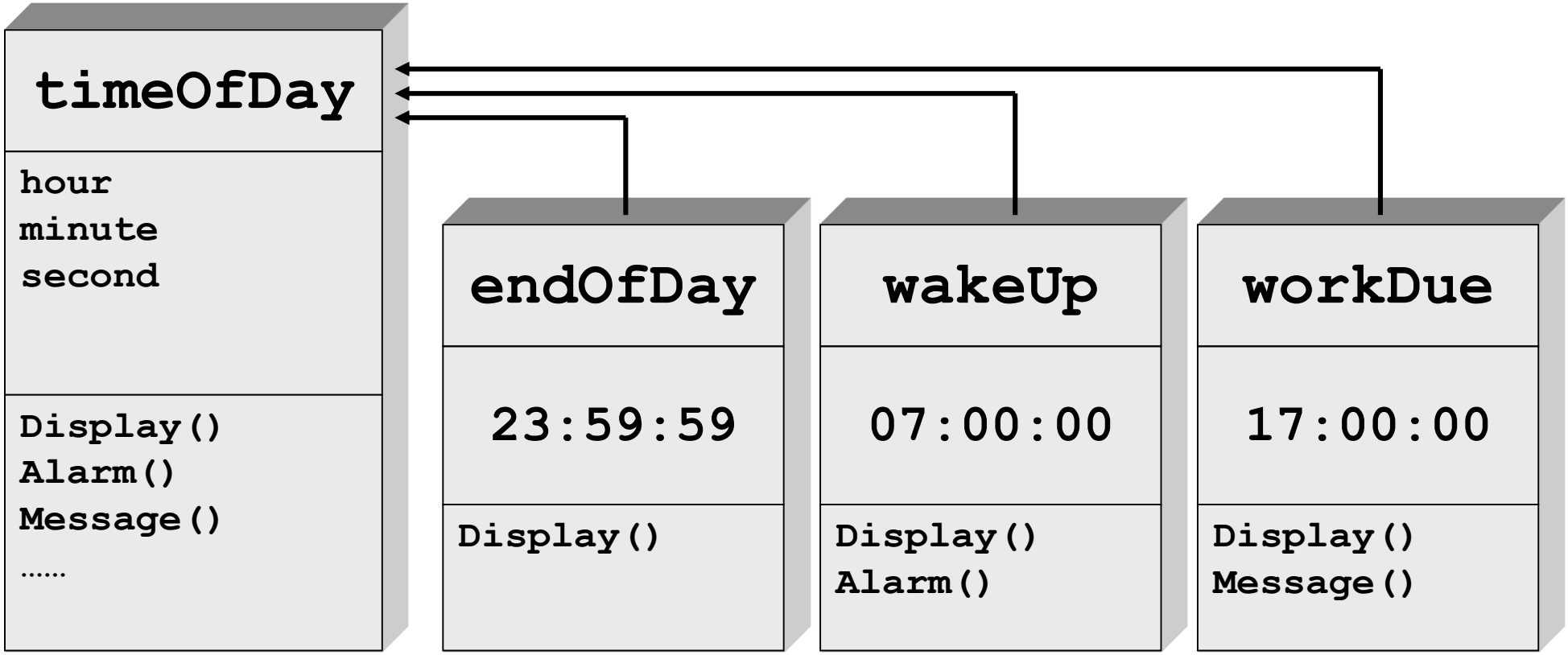
Objects

- An object in OOD is a component in the real or conceptual world that is mapped into the programme domain.
- **Properties** or **attributes** or **data members:**
  - Necessary information required to describe the state of an object.
- **Methods** or **behaviour** or **function members:**
  - All possible actions that an object should perform in order to describe its behaviour in the problem specification.



Object -> Class

- A class is an abstraction of all objects that have similar property structures and methods.



Class

- A class defines the structure of a group of similar objects (the properties and methods).
- An object is an instance of a class.
- Class is the basic structure in object-oriented programme design
- **Abstraction** can be easily achieved by using different classes.
- **Modularity** is automatically realised in programming based on classes since a class has data and actions encapsulated in it.
- **Information hiding** is realised by only allowing a class' method to access its data.
- Once defined, a class definition can be put into a header file and **repeatedly used** in any programme.

Class definition

- Syntax
  - Data member & function member
  - Private & public

```
class class_name
{
private:
  datatype variable_name;
  returntype function_name(parameter list);
public:
  datatype variable_name;
  returntype function_name(parameter list);
};
```

Declaration and definition of function members

- The ***declaration*** of member functions has to be placed in the class body;

- The ***definition*** of member functions can be placed outside the class body.

```
class time
{
private:
        //data members;
        int hour, minute, second;
public:
        //declaration of the method
        void setTime(int H, int M, int S);
};

//definition of the method
void time :: setTime(int H, int M, int S)
{
    hour=H;
    minute=M;
    second=S;
}
```

Declaration and definition of function members

- The member functions can also be defined inside the class body.
- In this case, it's called ***inline function***.
- During the compiling of program, the compiler replaces the function call with the corresponding function code. It's faster than regular functions, but coming with a memory penalty.

```
class time
{
private:
        //data members;
        int hour, minute, second;
public:
        //declaration and definition
        void setTime(int H, int M, int S);
        {
            hour=H;
            minute=M;
            second=S;
        }
};
```

Forgetting the semicolon at the end of a class definition is a syntax error.

Declare Class in Header File

- Class **`time`** is defined in the header (**`time.h`**)
- We tell the compiler what an **`time`** is by including its header, as in: **`#include "time.h"`**
- If we omit this, the compiler issues error messages wherever we use class **`time`** and any of its capabilities.
- In an **`#include`** directive, a header that you define in your program is placed in double quotes ("""), rather than the angle brackets (<>) used for C++ Standard Library headers like **`<iostream>`**.
- The double quotes in this example tell the compiler that header is in the same folder, rather than with the C++ Standard Library headers.
- You include headers into source-code files, though you also may include them in other headers.

```
class person
{
private:
    int     id;
    string name;
    int     office;
    int     nowhour
public:
    void show_nowhour(void);
    void set_hour(string Name, int workHour);
};

void person::show_nowhour()
{
    cout<<"Name: "<<name<<endl;
    cout<<"Worked "<<nowhour<< " hours.";
    cout<<endl;
}

void person::set_hour(string Name, int workHour)
{
        name = Name;
        nowhour = workHour;
}
```

```
void main()
{
    person John;
    John.set_hour("John", 20);
    John.show_nowhour();
}
```

Declaration and initialisation of an object

```
void main()
{
        person John;
        John.set_hour("John", 20);
        John.show_nowhour();
}
```

declaration

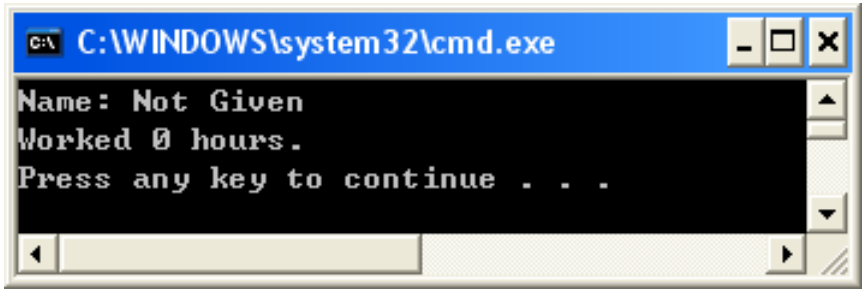
setup value

- Question: Can we initialise the data members when an object is declared? How?
  - A class constructor is needed to initialise all data members when an object of a class is declared.
  - A class constructor is a member function.

Default constructor

- Use the class name as the function name
- No return type
- Automatically called when an object of the class is declared
- **No parameter need to be specified**

  --- *default* constructor.



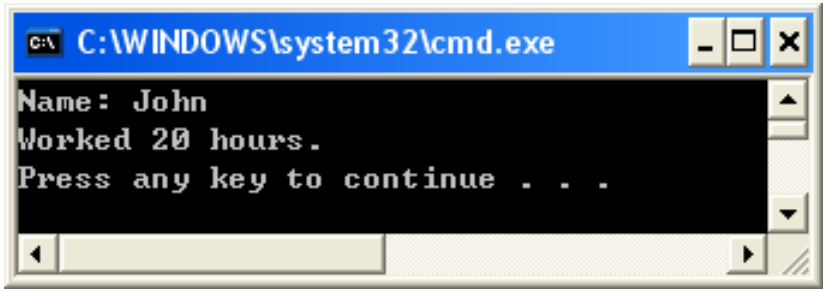

```
class person
{
    private:
        int     id;
        string  name;
        int     office;
        int     nowhour;

    public:
        person();
        void show_nowhour(void);
};
```

```
person::person()
{
        id=0;
        name="Not Given";
        office=0;
        nowhour=0;
}
```

```
void main()
{
        person John;
        John.show_nowhour();
}
```

Normal constructor

- Use the class name as the function name
- No return type
- Automatically called when an object of the class is declared
- **Parameter need to be specified**
  - It differs from the default constructor by have parameters to accept information.




```
class person
{
    private:
        int     id, office, nowhour;
        string name;

    public:
        person(int ID, string firstName,
int officeNumber, int workHour);
        void show_nowhour(void);
};
```

```
person::person(int ID, string firstName,
int officeNumber, int workHour)
{
        id=ID;
        name=firstName;
        office=officeNumber;
        nowhour=workHour;
}
```

```
void main()
{
    person John(1234, "John", 501, 20);
    John.show_nowhour();
}
```

Example of calling methods

```
#include <iostream>
#include <string>

/*CLASS DECLARATION GOES
BEFORE MAIN FUNCTION*/

void main(  )
{
    person p1;
    person p2(1, "James Bond", 107, 8);
    p1.show_nowhour( );
    p2.show_nowhour( );
}
```

```
class person
{
    private:
            int    id, office, nowhour;
            string name;
    public:
            person();
            person(int ID, string firstName, int
officeNumber, int workHour);
            void show_nowhour(void);
};
person::person()
{
            id=0; office=0; nowhour=0;
            name="Not Given";
}
person::person(int ID, string firstName, int
officeNumber, int workHour)
{
            id=ID;
            name=firstName;
            office=officeNumber;
            nowhour=workHour;
}
void person::show_nowhour()
{
    cout<<"Name: "<<name<<endl;
    cout<<"Worked "<<nowhour<< " hours.";
    cout<<endl;
}
```

Learn how to ask an object to do things --- calling its function by

`objectName.memberFunction();`

Example of ordinary function member

```
// Example of ordinary member functions
// A member function changes the working
hour
// Without any input argument


void person::change_nowhour(void)
{
    int value;
    cout<<"Please input new value
for nowhour for "<<name<<endl;
    cin>>value;
    nowhour=value;
    cout<<"Action finished with "
<<name<<", Thank you!"<<endl;
}
```

```
// Example of ordinary member functions
// A member function changes the person's
name
// With a string type argument


void person::change_name(string
newName)
{
    name=newName;
    cout<<"Name is changed to "
<<name<<", Thank you!"<<endl;
}
```

In class's member function, private data members of the class doesn't need to be declared in the function member. They can be directly used.

Building up Functionality gradually

```
// A member function to ask the user what to be done
// acting as the central control for all possible methods.

void person::action(void)
{
        int choice;
        string newName;
        cout<<"what do you want to do with "<<name<<endl;
        cout<<"1 for display, 2 for changing working hours, 3 for
changing name: ";
        cin>>choice;
        if(choice==1)
                show_nowhour( );
        else if(choice==2)
        {       change_nowhour( );
                show_nowhour( );
        }
        else if(choice==3)
        {       cout<<"Please input the new name: ";
                cin>>newName;
                change_name(newName);
        }
}
```

Destructor

- To free the memory allocated for an object when it goes out of its scope (the programme finishes using it) ---- especially important when using dynamic memory allocation.
  - A special function member of a class
  - With the name of a class
  - with the sign "~" (tilde) in front of its name
  - No type needed in front of the function name
  - No parameter
  - Called automatically when an object goes out of its scope

```
class person
{
    private:
        int     id, office, nowhour;
        string name;
    public:
        person();
        person(person &pp);
        ~person();
        void show_nowhour(void);
};

person::~person()
{
        cout<<"Destructor called!"<<endl;
}
```

Homework

- Write a programme with 3 different ways (array, structure and class) respectively that can let users input, save and print out information of 5 members in a team at the user's request.
  - The information for each member includes:

    ID number,

    Name,

    DoB (Date of Birth),

    Office number,

    Total hours of work per day.