

EEE102

C++ Programming and Software Engineering II

# Lecture 3 Classes and Objects

**Dr. Rui Lin/Dr. Fei Xue**

**Rui.Lin/Fei.Xue@xjtlu.edu.cn**

**Room EE512/EE222**



# Outline

- From structure to object
- Objects and classes
- CRC method for Object-orient 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	<code>struct student</code>	2_1	<code>student henry;</code>
1_2	<code>{</code>	2_2	<code>student tom, jerry;</code>
1_3	<code>    int age;</code>	2_3	<code>student *stPtr;</code>
1_4	<code>    char name[20];</code>	2_4	<code>student Y1[10];</code>
1_5	<code>};</code>		

```
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;
```

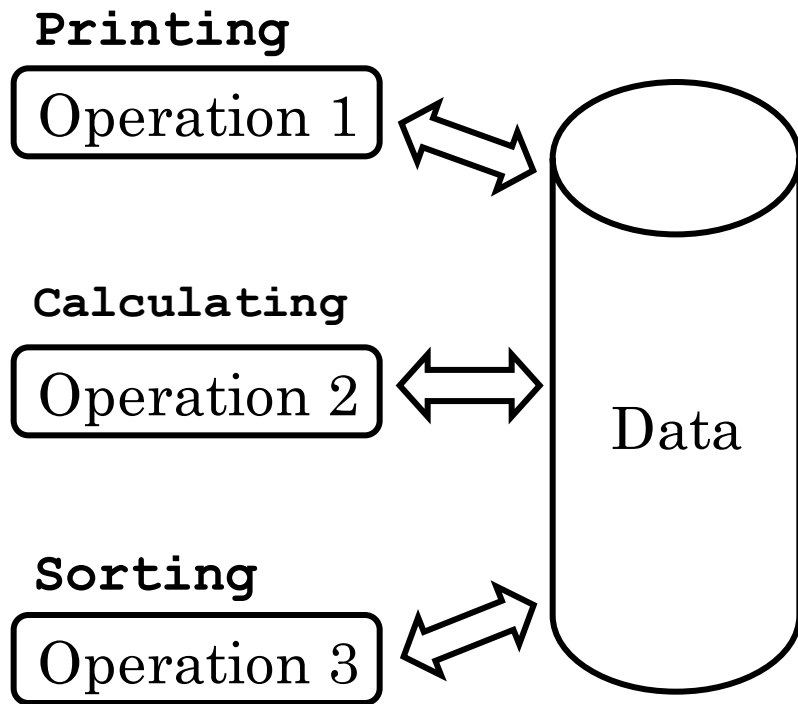
```
stPtr = &henry;
```

```
cout <<"Name: " <<stPtr->name <<"Age: " << stPtr->age <<endl;
```

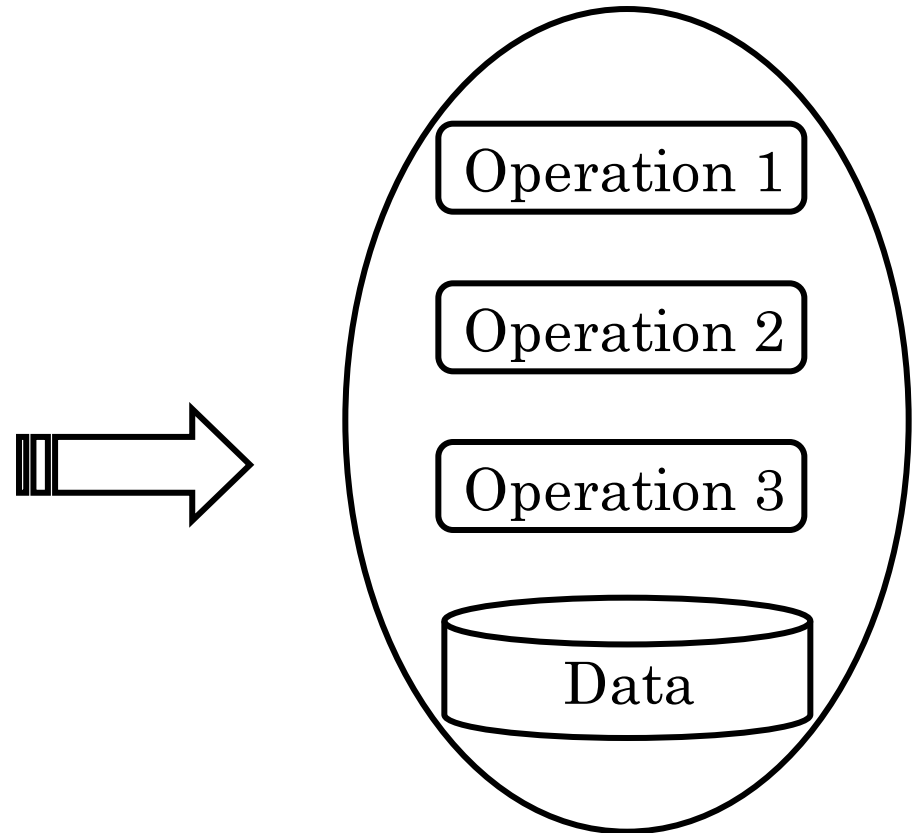


# CAN WE ...

- Encapsulate operations with structure?



Data and operations as separated entities



Data and operations bound as single unit





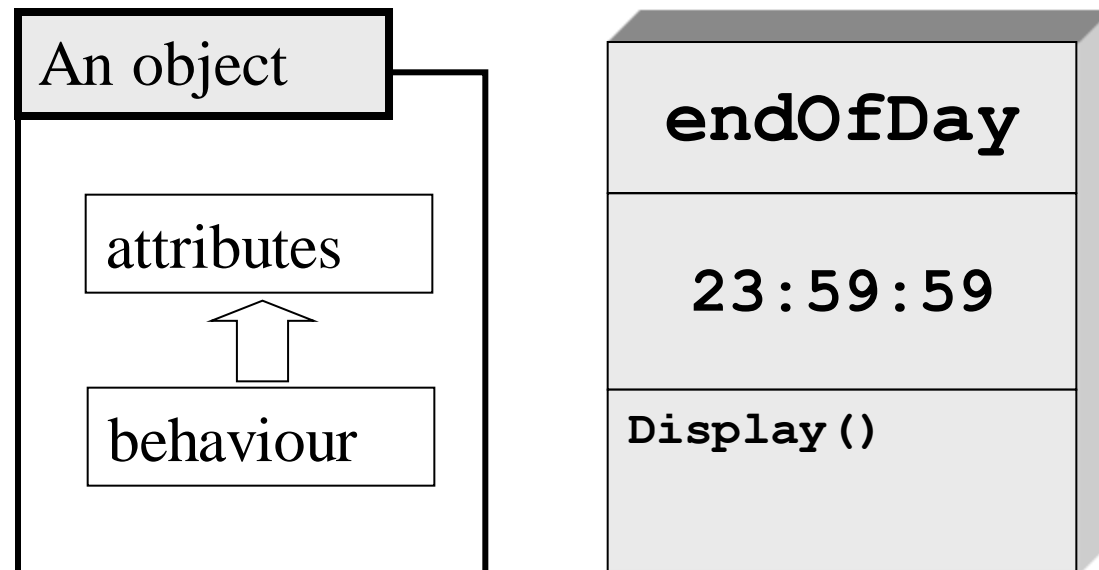
# The idea of OBJECT-ORIENTED Design

- 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.



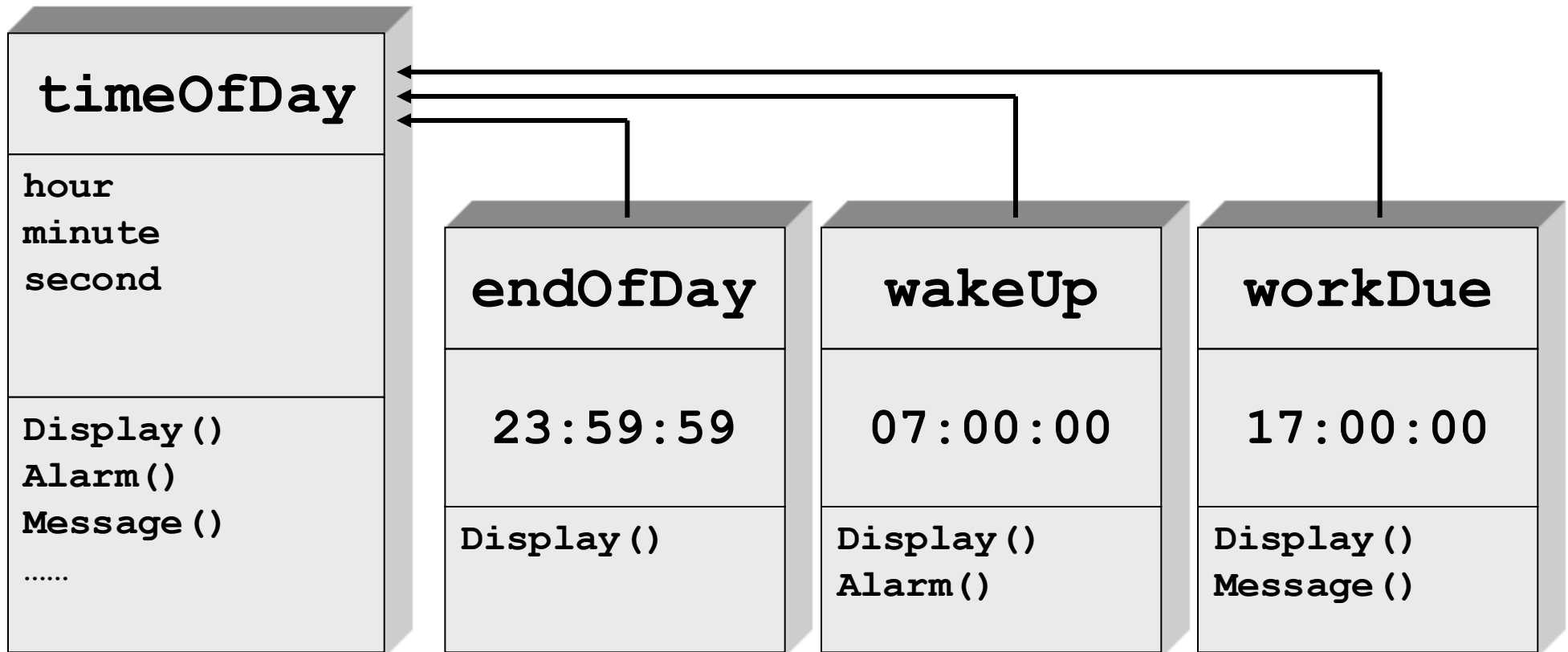
# Objects

- 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.



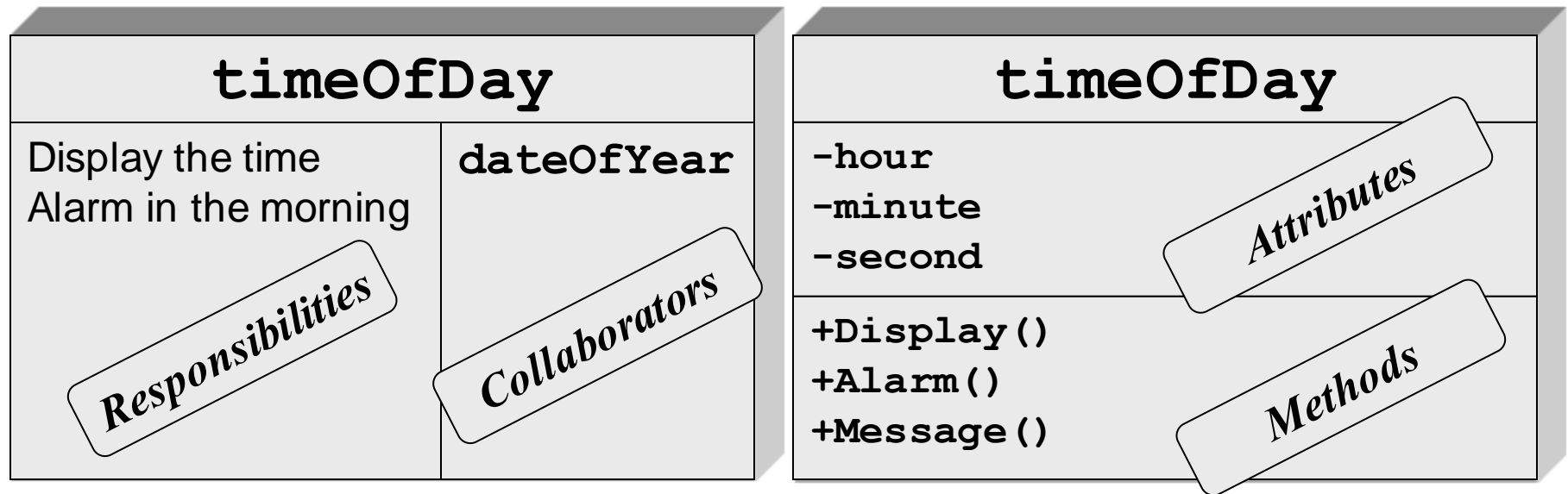
# Class

- 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.



# CRC Method

- 1) From specifications, identify typical objects and possible classes (**C - Class**).
- 2) Identify the responsibilities (actions) for each class (**R - Responsibilities**).
- 3) For each of the classes, identify the relevant classes that are involved in the actions of that class (**C – Collaborators**).
- 4) Refining the design by using use-cases.



# Example

- **Problem:**
- To design a calculator that can perform addition, subtraction, multiplication and division of two numbers. Such as:
  - $1+11=$
  - $8\times 12=$

Class: button  
Class: display  
Class: control unit



### Class: button

#### Responsibilities:

Inform the control unit when a key is pressed  
and tell control WHAT it represents

#### Collaborators

Control unit



### Class: display

#### Responsibilities:

Display the graphic interface of the calculator  
Display the input and result

#### Collaborators

Control unit



### Class: control

#### Responsibilities:

Get the inputs  
Display the inputs and calculated results  
Keep information and perform calculation

#### Collaborators

Button



Display



# Walk through using USE-CASES

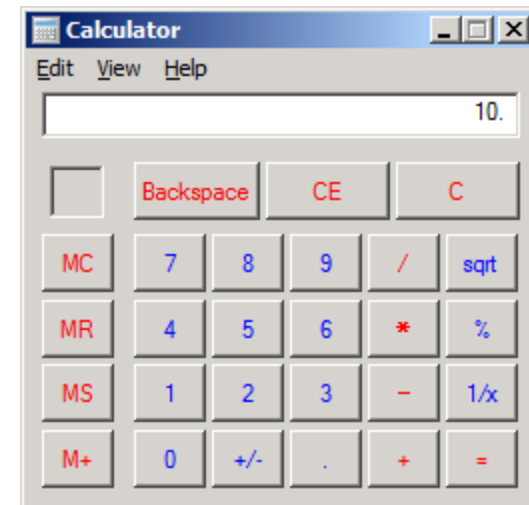
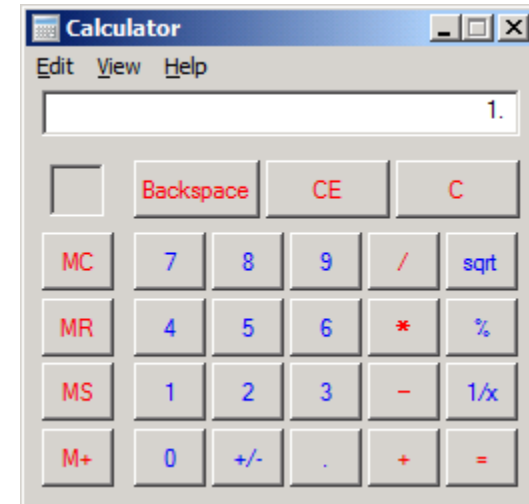
- Prepare a set of typical cases
- Go through each case.
- **Use-cases attempt to identify**
  - All objects involved in a typical case
  - The information passed between objects
  - The flow of control





## TRACE THE USE-CASE “10 + 1 = ”

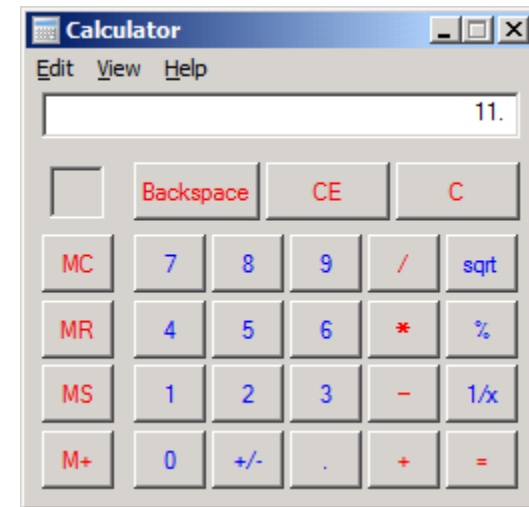
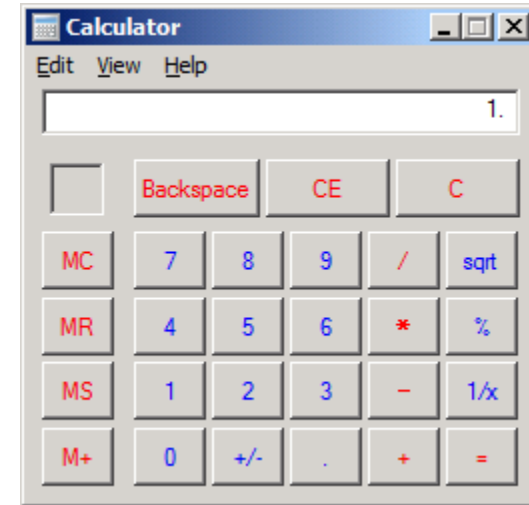
- First press 1. The **1\_button** tells the **control unit** it has been pressed. The control unit **stores** 1 and tells the **display** to show 1.
- Next press 0. The **0\_button** tells the control unit it has been pressed. The control unit stores 0, **appending it to the previous digit**, and tells the display to **append 0 to display**.
- Next press +. The **+** button tells the control unit it has been pressed. The control unit **stores the operator but do not display it**.



- Next press 1 again. The 1\_button tells the control unit it has been pressed. The control unit **stores 1 in an area separate from the first number**, and tells the display to show 1.

Implications --- The calculator clearly has to be able to store *operand1*, *operator* and *operand2*.

- Finally press =. The =\_button tells the control unit it has been pressed. The control unit then performs the **calculation** “operand 1 operator operand2” and tells the display to show the answer.



# Highlighted important aspects

- The button must be able to get the input from user and inform the control unit of what the input is;
- The control unit class must keep track of two operands and an operator, and do the calculation;
- The display must be able to display the input (as a string), and append a string to the current display;
- The interface protocol for various classes has been established



# 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 function members 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;
    }
};
```



```

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.

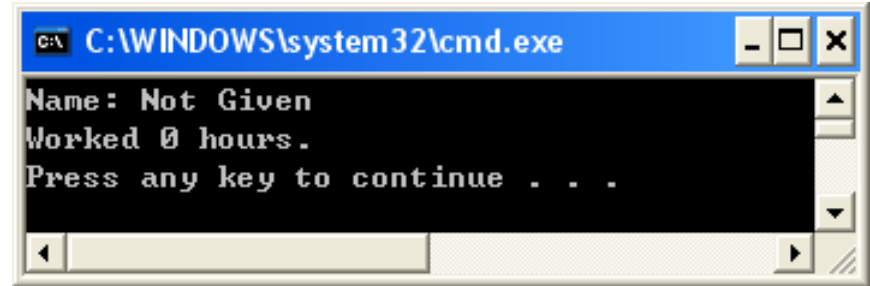




# Default constructor

- 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.



```
C:\WINDOWS\system32\cmd.exe
Name: Not Given
Worked 0 hours.
Press any key to continue . . .
```

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

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

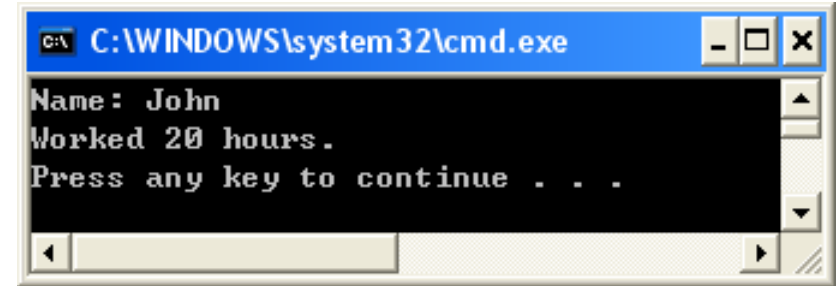
```
person::person()
{
    id=0;
    name="Not Given";
    office=0;
    nowhour=0;
}
```

```
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( );
}
```

Learn how to ask an object to do things --- calling its function by  
`objectName.memberFunction();`

```
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;
}
```



# 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 “~” 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;
}
```

