

# The CLASS Construct

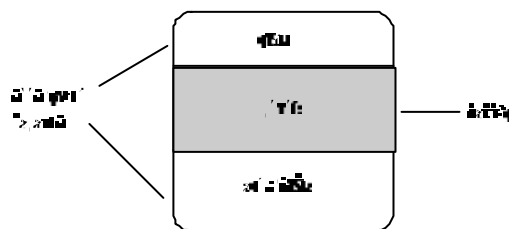
## 1. Introduction

In this handout classes, a concept which characterises object-oriented languages will be introduced. A class can be view as a user defined type which provides a set of rules about how an object of that type 'looks' and behaves. The most important feature of the class facility in an object-oriented language is that is allows the programmer to design types which are tailored to the particular job in hand.

When a program is written, a model of the real world problem is constructed, using variables to represent entities in the problem domain. In most languages, there are some basic built in types, such as an integer type, or a floating point type: a student ID number, for example, could be represented by an integer type variable. If something more complex is to be represented, such as a student, the a combination of the basic types would have to be used, one for the student ID number, another for the student name, another for the student course, and so on. whilst this may pose no problem, it may be found that the program quickly becomes obscured with the details of manipulating the complex entity, when what is really wanted is something which models the behaviour of a student without having to worry about the details of how the student type is implemented. For instance, if an integer type is used, the fact that it is implemented using two's complement binary numbers does not matter provided it behaves as an integer is expected to behave. This last point means that an integer is an abstract data type, i.e. abstracted from its implementation. Classes allows the building and use of user defined abstract types.

The relationship between a class and an object of that class is analogous to the relationship between an **int** variable and the built in type **int**. We can have many object of a given class in a program, in the same way that we can have many **int** variables, all of type **int**. We say that an object is an instance or an instantiation of a particular class. Each object in a program requires a unique name, just as each variable requires a unique variable name. Moreover, just as different variables of the same type can have different values, different objects of the same class can be in different states.

Classes allow us to effect data hiding in programs: the state of an object of a particular class is private to that object, and can not (easily) be changed in unexpected ways by rouge code, which makes programs much sager and easier to maintain. To achieve data



hiding, the definition of a class makes a distinction between the private state and the public interface.

The state of an object can only be changed in a controlled way by addressing the object interface.

## 2. C++ Class Construct

### Class Declarations

Before using objects in a program, we have to declare the classes from which those objects are to be instantiated. To declare a class in C++, we use the class keyword:

```
class class_name {
    private:
        // member data declarations .....

    public:
        // member function prototype declarations ...

};
```

where `class_name` is the name of the class, which can be used like any other type name in a program. Note the semicolon after the closing curly bracket of the class declaration body: the class construct is based on the struct construct, not the function.

Inside the body of the class declaration, there are two access control keywords: `public` and `private`. Anything declared in the `public` section is visible outside the object, and so this is the place where you would declare prototypes for the member functions which form the interface. Anything declared in the `private` section is hidden inside the class, the place where you would declare data members to store the state.

Member function prototypes declarations are just like ordinary function prototype declarations, i.e. `return_type member_function_name(parameter_type, ...);`, and member data declarations are similar to variable declarations, i.e. `type_name data_member_name;`. Member functions are, of course, part of the class, and so the data members declared in the `private` section of the class are visible to the member functions of that class.

The class declaration contains the most important information about the class, and so care should be taken over comments and the layout. If we are really going to use the class as an abstract type, then we should be able to find all the information we need in the class declaration,

and not worry about the details of the member function implementations. In fact, taking abstract types to the extreme, we should only be worried about the interface. Always put the interface as the first part of your class declaration.

## Member Function Definitions

Obviously, the programmer who actually writes the class does have to worry about the implementation. Not included in the class declaration are the implementations or definitions of the member functions (in the same way that a function prototype declaration still needs a function definition somewhere in the program). However, because a member function belongs to a particular class, we need extra syntax to tell the compiler the name of the class to which the member function belongs. This leads to a member function definition of the form:

```
return_type class_name::member_function_name(parameter_type parameter_name, ..)
{
    // statements in function body
}
```

Here, the scope operator `::` is added to tell the compiler that the function is in the scope of `class_name`.

## Constructors and Destructor

There are two special member functions, which are exceptional in that they are automatically called by the compiler, and do not have a return type in their declaration or definition.

The first is known as a constructor, a member function which is automatically called whenever an object of that class is instantiated. A constructor has the same name as the class. Typical actions by a constructor would be to provide initial values for data members (data members cannot be initialised when they are declared, as in the case with variables), or to dynamically allocate storage for data members, if required.

The second is known as a destructor, and is automatically called when an instance of the class is destroyed. A destructor has the same name as the class, prefixed by `~`, as in `~class_name`. Typical applications for a destructor would be to deallocate any dynamic data members, or to report a statistical summary of the object's activities.

You do not have to declare and define a constructor and destructor for your class. If you do not, the compiler will provide default versions.

## Using Classes

Once you have declared your class and defined the member functions, you can use the class in a similar way to any built in type. Say we have a class called country, then we could declare or instantiate objects of type country in our program using a variable declaration:

```
country england;  
country scotland;
```

or we could declare a pointer to an object of our class with a pointer variable declaration:

```
country* current_location;
```

Now, suppose that class country has a data member long population which stores the number of people living in a country, and a member function long give\_population(void) which returns the value of population to the caller. We could cause the population of Scotland to be printed on the screen by using the member selection operator . with the give\_population() function in the following way:

```
cout << scotland.give_population();
```

Note, however, that the data member population is in the private section of the class, so

```
cout << scotland.population();
```

would be illegal.

We could initialise a pointer to a country in the usual way:

```
current_location = &england;
```

because we now have a pointer to an object, we would need to use the arrow -> form of the member selection operator to see the population of current\_location:

```
cout << current_location -> give_population();
```

## Class Example

Here, we use an example of a customer in a hotel and declare a class for customer. Having declared our customer class, then define the member function, and finally provide a main

function using the class:

```
#include <iostream.h>
#include <string.h>
#include <assert.h>

// define maximum size for customer name
#define NAME_SIZE 21

// declare the customer class
class customer {

private:
    // state
    char firstname[NAME_SIZE];    // customers first name
    char surname[NAME_SIZE];      // customers surname
    int number_of_nights;        // number of nights staying

public:
    // interface
    customer(void);              // constructor
    void enter_customer(void);   // dialogue to set customer state
    void print_customer(void);   // print customer details on screen
    void set_number_of_nights(int); // set value of number of nights

};

// member function definitions

// constructor
customer::customer(void){
    strcpy(firstname,"nobody");
    strcpy(surname,"noone");
    number_of_nights = 0;
}

// enter customer details via dialogue
void customer::enter_customer(void){
    int n;

    cout << "enter customer surname (less than "<<
        NAME_SIZE-1 << " characters): ";
    cin >> surname;

    cout << "enter customer firstname (less than " <<
        NAME_SIZE-1 << " characters): ";
    cin >> firstname;

    do{
        // check that the number of nights is valid
        cout << "enter number of nights: ";
```

```

        cin >> n;
    }while(n < 0);
    number_of_nights = n;
}

// print customer details on screen
void customer::print_customer(void){

    cout << surname << ", " << firstname << "\t" <<
        number_of_nights << "\n";
}

// set number of nights
void customer::set_number_of_nights(int n){

    // precondition: number of nights > 0
    assert(n >= 0);
    number_of_nights = n;
}

// main function

void main(void){
    customer c;                // declare customer object called c

    c.enter_customer();        // start enter dialogue

    c.print_customer();        // print c to screen

    c.set_number_of_nights(99); // change number of nights

    c.print_customer();
}

```