C++ Classes
How to Create and Use Them (Constructor, Destructor)

By Kouros

Overview

- Functions in Classes (methods)
  - Constructor
  - Accessors/Modifiers
  - Miscellaneous
- Terminology
- File Topology
- Designing Classes
- The Driver and Object instantiation

Class Constructors

- **A class constructor is a member function whose purpose is to initialize the private data members of a class object**
- **The name of a constructor is always the name of the class, and there is no return type for the constructor**
- **A class may have several constructors with different parameter lists. A constructor with no parameters is the default constructor**
- **A constructor is implicitly and automaticly invoked when a class object is declared--if there are parameters, their values are listed in parentheses in the declaration**

Specification of `TimeType` Class Constructors

```
class  TimeType                    // timetype.h
{
public :                           //  7 function members
      void       Set ( int  hours ,  int  minutes ,  int   seconds ) ;
      void       Increment ( ) ;
      void       Write ( )  const ;
      bool       Equal ( TimeType   otherTime )  const ;
      bool       LessThan ( TimeType  otherTime )  const ;

      TimeType ( int  initHrs ,  int  initMins ,  int  initSecs ) ;  // constructor

      TimeType ( ) ;                          // default constructor

private :                          //  3 data members
      int          hrs ;
      int          mins ;
      int          secs ;
} ;
```

Implementation of `TimeType` Default Constructor

```
TimeType :: TimeType ( )
// Default Constructor
// Postcondition:
//          hrs == 0  &&  mins == 0  &&  secs == 0
{
     hrs  =  0 ;
     mins = 0 ;
     secs = 0 ;
}
```

Implementation of Another `TimeType` Class Constructor

```
TimeType :: TimeType (int   initHrs, int   initMins, int   initSecs )
// Constructor
// Precondition:  0 <= initHrs <= 23    &&    0 <= initMins <= 59
//                    0 <= initSecs <= 59
// Postcondition:
//         hrs == initHrs  &&  mins == initMins  && secs == initSecs
{
        hrs  =  initHrs ;
        mins =  initMins ;
        secs =  initSecs ;
}
```

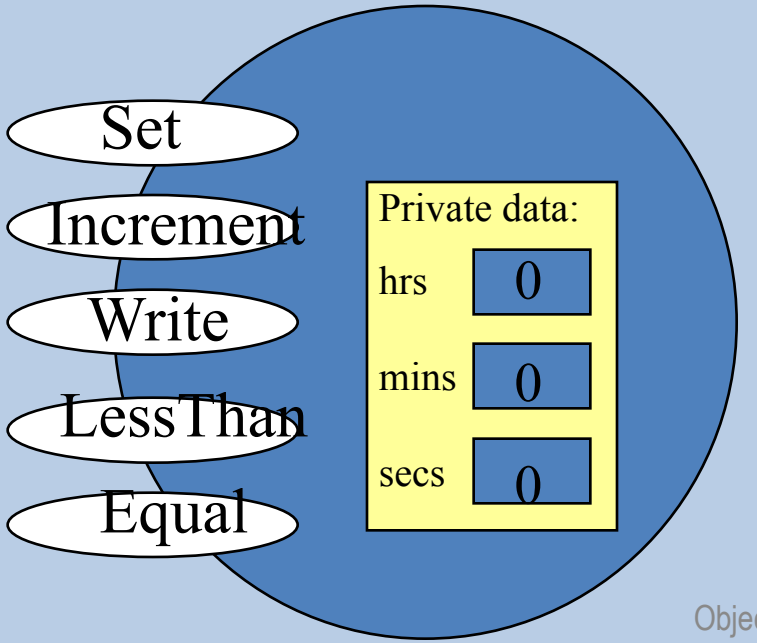
Automatic invocation of constructors occurs

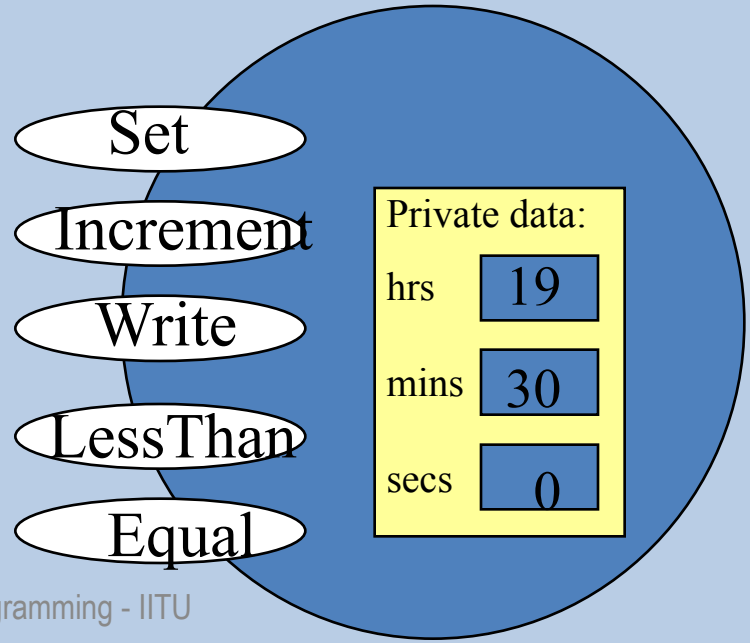
```
Main(){

      TimeType   departureTime ;          // default constructor invoked

      TimeType   movieTime (19, 30, 0 ) ;   // parameterized constructor

      departureTime     movieTime
}
```





The Class Destructor

- A **destructor** is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.
- A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.

Destructor example

```
CDog ::~CDog (void)
 { cout << "Object is being deleted" << endl; }
```

A “real life” example

- The CDog
  - Attributes (characteristics)
    - rabid or not rabid (bool)
    - weight (int or float)
    - name (char [ ])
  - Behaviors
    - growl
    - eat

Step 1: The Skeleton

```
class CDog {
    // attributes will go here – name, weight,
rabid
    // behaviors will go here – growl, eat
};
```

Step 2: The attributes

```
class CDog {
  public:
    boolean rabid;
    int weight;
    char name[255];

    // Behaviors go here
};
```

Step 3: The Constructor

- This is a special function
  - Used to give initial values to ALL attributes
  - Is activated when someone creates a new instance of the class

- The name of this function MUST be the same name as the class

Step 3: Designing the Constructor

- Constructors will vary, depending on design
- Ask questions:

  – Are all CDogs born either rabid or non-rabid?
  (yes – they are all born non-rabid)

  – Are all CDogs born with the same weight?
  (no – they are born with different weights)

  – Are all CDogs born with the same name?
  (no – they all have different names)

- If ever “no”, then you need information passed in as parameters.

Step 3: The Constructor

```
class CDog {
      public:
      boolean rabidOrNot;
      int weight;
      char name [255];
      // Constructor
      CDog::CDog (int x, String y)
      {
            rabid = false;
            weight = x;
            strcpy (name, y);
      }
      // Behaviors go here
};
```

Notice that every CDog we create will be born non-rabid. The weight and name will depend on the values of the parameters

Back to CDog

```
class CDog {
        public:
        boolean rabidOrNot;
        int weight;
        char name [255];
        // Constructor
        CDog::CDog (int x, char y[ ]) {
                rabid = false;
                weight = x;
                strcpy (name, y);
        }
    CDog ::~CDog ()
    { cout << "Object is being deleted" << endl; }
        // Behaviors we still need to eat and growl
};
```

Miscellaneous Methods

- Follow the pattern

```
void CDog::eat ( ) {
    cout << name << " is now eating" << endl;
    weight++;
}

void CDog::growl ( ) {
    cout << "Grrrr" << endl;
}
```

Add Methods

```
class CDog {
        public:
        boolean rabidOrNot;
        int weight;
        char name [255];
        // Constructor
        CDog::CDog (int x, char y[ ]) {
                rabid = false;
                weight = x;
                strcpy (name, y);
        }
        void CDog::eat ( ) {
        cout << name << " is now eating" << endl;
        weight++;
          }

          void CDog::growl ( ) {
        cout << "Grrrr" << endl;
          }
};
```

Create New Object(Instance)

```
Cdog c1 ;  // create an object that run default constructor
CDog c2 (7, “Ethel”); // create an object that run other constructor
CDog* c1 = new CDog (14, “Bob”); // create a pointer object
```

The “.” and “->” operators

- “Dot” operator used for non-pointers to:
  - Get to an instances attributes
  - Get to an instances methods
  - Basically get inside the instance

- Format:
  <instance>.<attribute or method>

- Arrow operator used for pointers

- Format:
  <instance> -> <attribute or method>

Using the “.” and “->” Operators

```
#include <iostream.h>

void main ( ) {
        CDog* c1;
        c1 = new CDog (14, “Bob”);
        CDog c2 (7, “Ethel”);
        c2.bark( );
        c1->growl( );
}
```

Accessors and Modifiers

- Accessor for the rabid attribute

```
bool CDog::getRabid ( ) {
  return rabid;
}
```

- Modifier for the rabid attribute

```
void CDog::setRabid (bool myBoolean) {
  rabid = myBoolean;
}
```

- Put these inside of the CDog class

Using accessors and modifiers

```
#include <iostream.h>

void main ( ) {
        CDog* c1;
        c1 = new CDog (14, “Bob”);
        CDog c2 (7, “Ethel”);
        c1->setRabid (1);
        // prints 1 for true
        cout << c1->getRabid( ) << endl;
}
```

Make a Separate Header File

(for the generic description)

```
class CDog {
  public:
        int weight;
        bool rabid;
        char name [ ];
        CDog (int x, char y[ ]);
        bool getRabid ( );
        void setRabid (bool x);
        char [ ] getName ( );
        void setName (char z[ ]);
        int getWeight ( );
        void setWeight (int x);
        void bark( );
        void growl( );
};
```

Our Final CDog.cpp

Cdog.cpp

```
#include <iostream.h>
#include <CDog.h>

// Constructor
CDog::CDog (int x, char y[ ]) {
        rabid = false;
        weight = x;
        strcpy(name, y);
}
void CDog::eat ( ) {
cout << name << " is eating";
}
void CDog::growl ( ) {
        cout << "Grrrr";
}
```

Cdog.h

```
bool CDog::getRabid ( ) {
    return rabid;
}
void CDog::setRabid (bool
x) {
    rabid = x;
}
int CDog::getWeight ( ) {
    return weight;
}
void CDog::setWeight (int y)
{
    weight = y;
}
char[ ] CDog::getName ( ) {
    return name;
}
void setName (char z[ ]) {
    name = z;
}
```

Hierarchical (Nested) class

```
class Host
{
public:
  class Nested
  {
  public:
    void PrintMe()
    {
      cout << "Printed!\n";
    }
  };
};
```

```
int main()
{
  Host::Nested foo;
  foo.PrintMe();


  Host bar;
  // nothing you can do with bar to call PrintMe
  //   Host::Nested and Host are two separate
classes

  return 0;
}
```

Simple Nested class

```
class A{...};
class B{
   public:
    A a;//declare members

    B() : a(...) {
    }
   // constructors are called here
};
```

Summary of Class Concepts

- A class is a generic description which may have many instances

- When creating classes
  1. Make the constructor
  2. Make the accessors/modifiers/miscellaneous

- Classes go in separate files

- The "." and "->" operators tell the instances which method to run