Chapter 2

Using Objects of A Single Class

Classes and Objects

Defining a class:

```cpp
class Frame {// represent a graphical user
    // interface window
    /* the body of the class definition goes here
    between the curly braces */
};
```

Declaring objects:

```cpp
Frame display(...);
Frame userDisplay(...), aViewer(...);
Frame editorWindow(...);
```
Classes and Objects

Section 2.1:

Defining a class:

```
class Frame {  // represents graphical user
  // interface window
  /* body of class definition goes here,
     between curly braces */
};
```

Declaring Class Instances

```
int counter;
Frame display(...);
Frame userDisplay(...),
aViewer(...);
```

C++ strongly type checks uses of instance names
(You can add ints, but not frames!)
Right or Wrong?

1) Frame frame(...);
2) Frame aFrame(...);
3) Frame Frame(...);
4) class int { ... };
5) class Int { ... };

The Structure of Classes and Objects

Section 2.2:

**Private part:**

Remember what “encapsulation” means?

(Only access “internals” through interface)

Examples: engine in car, pointer manipulation in sorted-list class

**Public part:**

Functions & variables that other classes can use

Examples: gas pedal in car, “insert()” method

```cpp
class Frame {
private:
    // encapsulated
    // implementation goes
    // here

public:
    // interface visible to the
    // user goes here
};
```
The Structure of Classes and Objects

Public interface:
- collection of methods (a.k.a. operations or member functions) giving user ways to operate on object
- 3 kinds of public methods:
  - constructors: how to initialize an object
  - behavioral: how to manipulate an object
  - destructors: how to delete/destroy an object

class Frame {
private:
// encapsulated
// implementation goes
// here
public:
// interface visible to the
// user goes here
};

The Frame Class

class Frame {
private: // added later
public: // added later
Frame (char *name, int initUpperLeftXCoord,
int initUpperLeftYCoord,
int initWidth,
int initHeight);

... // Other methods added later

~Frame ();
};
The Frame Class

```c++
class Frame { // Version 1
  private:
  public:
    Frame(...);
    int IsNamed (char *name);
    void MoveTo (int newXCoord, int newYCoord);
    void Resize (int newHeight, int newWidth);
    void Clear();
    void DrawText (char *text, int atX, int atY);
    void DrawLine (int fromX, int fromY, int toX, int toY);
    void DrawCircle (int centerX, int centerY, int radius);
    ~Frame();
};
```

Try This Now

Write a class declaration for a “stack of integers.”
Creating and Operating on an Object

Section 2.3:

Frame display(“Test”, 10, 20, 100, 200);
...
display.MoveTo(50, 50);
display.Resize(200, 200);
display.DrawText(“Hi World!”, 20, 50);

Simple Programming Environment

// This is the file Program.cpp
#include "Program.h"
// include any necessary header files here (e.g., "Frame.h")
// define here any global objects or variables

void OnStart(void) {}
void OnMouseEvent(char *frameName, int x, int y, int buttonState){}
void OnTimerEvent(void){}
void OnPaint(void){}
Hello World Program

#include "Frame.h"
Frame window("Hello World Program", 200, 200, 400, 400);
void OnStart(void) {
    window.Clear();
    window.DrawText("Hello World!", 20, 20);
}
void OnMouseEvent(char *frameName, int x, int y,
                  int buttonState){}
void OnTimerEvent(void){}
void OnPaint(void){
    window.Clear();
    window.DrawText("Hello World!", 20, 20);
}

Hello World Program with Mouse Events

#include "Program.h"
#include "Frame.h"
Frame window("Hello World Program", 200, 200, 400, 400);
int lastx;
int lasty;
void OnStart(void) {
    window.Clear();
    window.DrawText("Hello World!", 20,20);
    lastx = 20;
    lasty = 20;
}
void OnTimerEvent(void){}
**Hello World Program with Mouse Events**

```c
void OnPaint(void){
    window.Clear();
    window.DrawText("Hello World!", lastx, lasty);
}

void OnMouseEvent(char *frameName, int x, int y,
                  int buttonState){
    if (buttonState & leftButtonDown) {
        window.Clear();
        window.DrawText("Hello World!",x,y);
        lastx = x;    lasty = y;
    }
}
```

**Hello World Program with Mouse and Timer Events**

```c
#include "Program.h"
#include "Frame.h"

Frame window("Hello World Program", 200, 200, 400, 400);
    int lastx;
    int lasty;
    int visible;

void OnStart(void) {
    window.Clear();
    window.DrawText("Hello World!", 20, 20);
    lastx = 20;
    lasty = 20;
    visible = 1;
}
```
Hello World Program with Mouse and Timer Events

```c
void OnMouseEvent(char *frameName, int x, int y,
                  int buttonState){
    if (buttonState & leftButtonDown) {
        window.Clear();
        if (visible)  window.DrawText("Hello World!",x,y);
        lastx = x;    lasty = y;
    }
}

void OnTimerEvent(void){
    window.Clear();
    if (visible) visible = 0;
    else        {  visible = 1;
                   window.DrawText("Hello World!", lastx, lasty);
    }
}
```

Hello World Program with Mouse and Timer Events

```c
void OnPaint(void){
    window.Clear();
    if (visible) window.DrawText("Hello World!", lastx, lasty); 
}
```
Overloaded Constructors

Section 2.4:

class Frame {           // Version 2
    private:
        // encapsulated implementation goes here
    public:
        Frame   (char *name, int initXCoord, int initYCoord,
                int initWidth,  int initHeight);
        Frame   (char *name, int initXCoord, int initYCoord);
        Frame   (char *name);
        Frame   ();
        ...
};

Using Overloaded constructors

Frame exact ("Exact", 50, 60, 100, 200);
Frame here ("No Shape", 50, 50);
Frame any ("Name Only");
Frame anonymous;
Overloaded Methods

class Frame {                   // Version 2 (Cont.)
    private:
        // encapsulated implementation goes here
    public:
        //...
        void Resize ( int newHeight, int newWidth );
        void Resize ( float factor );
        void Clear();                                      // clear entire window
        void Clear(int x, int y, int w, int h); // clear rectangular area
        void TextSize(char* text, int& w, int& h); // find msg size
};

Using Overloaded Methods

Frame window("Testing", 100,100, 200,200);

    window.Resize(100, 100);  // change to a 100 X 100
    window.Resize(1.5);       // enlarge by 50%
    window.Resize(0.5);       // shrink to 50%

    window.DrawText("Hello World", 20, 20);
    window.DrawText("This is Great!", 50,50);

    int w, h;
    window.TextSize("Hello World", w, h);
    window.Clear(20, 20, w, h);       // erase "Hello World"
**Default Arguments**

*Section 2.5:*

class Frame {
    public:
        ...
        void Resize (int width = 100, int height = 150);
        ...
};

Frame window (100, 100, 300, 400);
...

<table>
<thead>
<tr>
<th>Call</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>window.Resize(200, 250);</td>
<td>200 x 250</td>
</tr>
<tr>
<td>window.Resize(200);</td>
<td>200 x 150</td>
</tr>
<tr>
<td>window.Resize();</td>
<td>100 x 150</td>
</tr>
</tbody>
</table>

**Why Is This Illegal?**

class Cube {
    public:
        ...
        void Resize(int height=10, 
                     int width=20, 
                     int depth);
        ...
};
Basic I/O
Section 2.6:

**C++ uses Streams:**

- **Input:**
  Data flows from source into variables

- **Output:**
  Data flows from variables into destination

**I/O can be**

- sequential
- random ← ignore for now

Stream I/O in C++

A “stream” model is used for these purposes:

Interactive I/O

File I/O

Stream Operators

Formatting in Memory

Output to a Window
Chapter 2 27

C++ (Interactive) I/O Streams

<iostream.h> defines

    istream cin;  // standard input
    ostream cout; // standard output

Class istream defines operator >> for input:
    cin >> x;

Class ostream defines operator << for output:
    cout << x;

Chapter 2 28

C++ (Interactive) I/O Streams

#include <iostream.h>

cout << 10; // stream output operator

int i;
cin >> i;   // stream input operator
            // on predefined cin

//Streams using overloading
cout << 10;  // output an integer
cout << 2.5; // output a float
cout << "Now is the time."
            // output a string
cout << '\n';  // output a "newline"
cout << endl;  // output predefined endline
C++ File Stream I/O

Uses these classes:

```cpp
#include <fstream.h>
class ifstream {…}
class ofstream {…}
```

Examples:

```cpp
ifstream inputFile("input.txt");
ofstream resultFile("output.txt");
int x,y
inputFile >> x >> y;
resultFile << "Sum is " << (x+y) << endl;
```

Stream I/O to a Memory Buffer

C++ equivalent of sscanf and sprintf in C:

```cpp
#include <iostream>
#include <sstream>
using namespace std;
class istringstream {…}; // read from memory
class ostringstream {…}; // write to memory
```

What does this program fragment output?

```cpp
char text[100];
ostringstream expression(text, 100);
int i=10, j=2*i;
expression << i << " + " << j;
cout << text << endl;
```

Is this legal?

```cpp
cout << expression;
```
Stream I/O to a Memory Buffer

What does this program fragment output?

```cpp
char* text = "10+20";
istringstream parser(text, 100);
int value1, value2; char op;
parser >> value1
  >> op
  >> value2;
cout << op << '(' << value1 << ',' << value2 << ')';
```

Stream I/O to a Window

`TextFrame` is a variation of `Frame`.

- Allows stream output of built-in types to a window.
- Useful for dumping debug information.
What Does this Program Do?

```c
#include "Program.h", "Frame.h", "TextFrame.h"
Frame window1("Window1", 100, 100, 100, 100);
Frame window2("Window2", 300, 100, 100, 100);
TextFrame out ("Results", 500, 100, 200, 600);
void OnStart() { OnPaint(); }
void OnMouseEvent(char* fName, int x, int y, int buttonState) {
    if (buttonState) out << fName << " was clicked." << endl;
}
void OnPaint() {
    window1.DrawText("Click in here!", 20, 20);
    window2.DrawText("Click in here!", 20, 20);
}
void OnTimerEvent() {}
```

Arrays of Objects

**Section 2.7:**

Arrays of objects can be declared in the same way that arrays of built-in types are declared:

```c
Frame winArray[10]; // declares array of 10 frame objects; uses default constructor for each object
```

Subscripting is used the same way as with built-in types:

```c
int i;
...
winArray[i].MoveTo(200, 200);
```
Dynamic Objects: Two Definitions

Section 2.8:

- **Scope**
  A lexical issue:
  Where can object’s name be used (or *is visible*)?

- **Lifetime**
  A resource issue:
  When does object exist during program execution?

Dynamic Objects: Two Possible Relationships

- **Automatic**: Object lifetime is tied to its scope:
  object is constructed when scope is entered & destructed when scope is exited:

```c
void main() { f(); }  // “Test” only exists during call
void f()        { Frame x(“Test”); }
```

- **Dynamic**: Object lifetime is independent of scope.
  Program(mer) controls when it’s destructed:

```c
void main() { f(); }  // “Test” exists during, after call
void f()        { Frame* x; x = new Frame(“Test”); }
```
Auto Scope: How Many Frame Instances Created?

Frame globalWindow;             // global scope
void function() {
    Frame functionWindow; // start of functionWindow scope
    ...
    for( int i=0; i<10; i++ ) {
        Frame loopWindow;     // start of loopWindow scope
        ...
        if (i < 5) {
            Frame ifWindow;  // start of ifWindow scope
            ...
        }                   // end of ifWindow scope
    }                        // end of loopWindow scope
}                                    // end of functionWindow scope

Creating Objects Dynamically

Section 2.9:

Frame *window;         // declaration of pointer variables
window = new Frame("First", 10, 20, 50, 50);        // create a new Frame object

Frame *edit = new Frame ("Editor", 50, 75, 100, 100);    // combine declaration of pointer
// variable and object construction
Frame *default = new Frame; // use default constructor values

delete window;                  // destruct window Frame
delete edit;                         // destruct edit        Frame
delete default; // destruct default   Frame
Manipulating Dynamically Created Objects

Create objects dynamically:

```cpp
Frame* display = new Frame("Display", 10, 20, 100, 200);
```

Manipulate the object:

```cpp
display->MoveTo(50, 50);
display->Resize(200, 200);
```

---

Dangers with Dynamically Created Objects

```cpp
Frame *display = new Frame("Shared", 10, 20, 100, 100);
Frame *view;
view = display; // both point to same Frame object
```

```cpp
display->MoveTo(50, 50); // OK - moves shared Frame object
view->Resize(200, 200); // OK - resizes shared Frame object
delete display; // delete shared object
```

```cpp
view->MoveTo(20, 20); // ERROR - object already deleted!
```
Another Danger: Memory Leaks

```cpp
{
    for (int i=0; i<100; i++) {
        Frame *display;
        display = new Frame("Memory Leak", 50, 50, 100, 100);
    }
    // "display" was destroyed, but "Memory Leak" still exists!
}
```

Dynamically created objects are \textbf{NOT} automatically destructed when the variable(s) pointing to them go out of scope.