Chapter 5

HIDING THE IMPLEMENTATION

***Listing 5-1. C++’s public is Just Like C’s struct***

**//: C05:Public.cpp**

**// Uses identical struct declarations**

**struct A {**

**int i;**

**char j;**

**float f;**

**void func();**

**};**

**void A::func() {}**

**struct B {**

**public:**

**int i;**

**char j;**

**float f;**

**void func();**

**};**

**void B::func() {}**

**int main() {**

**A a; B b;**

**a.i = b.i = 1;**

**a.j = b.j = 'c';**

**a.f = b.f = 3.14159;**

**a.func();**

**b.func();**

**} ///:~**

***Listing 5-2. The private Access Specifier***

**//: C05:Private.cpp**

**// Setting the Boundary**

**// and Hiding Portions of the Representation**

**struct B {**

**private:**

**char j;**

**float f;**

**public:**

**int i;**

**void func();**

**};**

**void B::func() {**

**i = 0;**

**j = '0';**

**f = 0.0;**

**};**

**int main() {**

**B b;**

**b.i = 1; // OK, public**

**//! b.j = '1'; // Illegal, private**

**//! b.f = 1.0; // Illegal, private**

**} ///:~**

***Listing 5-3. Declaring a Friend***

**//: C05:Friend.cpp**

**// Friend allows special access**

**// Declaration (incomplete type specification):**

**struct X;**

**struct Y {**

**void f(X\*);**

**};**

**struct X { // Definition**

**private:**

**int i;**

**public:**

**void initialize();**

**friend void g(X\*, int); // Global friend**

**friend void Y::f(X\*); // Struct member friend**

**friend struct Z; // Entire struct is a friend**

**friend void h();**

**};**

**void X::initialize() {**

**i = 0;**

**}**

**void g(X\* x, int i) {**

**x->i = i;**

**}**

**void Y::f(X\* x) {**

**x->i = 47;**

**}**

**struct Z {**

**private:**

**int j;**

**public:**

**void initialize();**

**void g(X\* x);**

**};**

**void Z::initialize() {**

**j = 99;**

**}**

**void Z::g(X\* x) {**

**x->i += j;**

**}**

**void h() {**

**X x;**

**x.i = 100; // Direct data manipulation**

**}**

**int main() {**

**X x;**

**Z z;**

**z.g(&x);**

**} ///:~**

***Listing 5-4. Nested Friends***

**//: C05:NestFriend.cpp**

**// Demonstrates Nested friends**

**#include <iostream>**

**#include <cstring> // memset()**

**using namespace std;**

**const int sz = 20;**

**struct Holder {**

**private:**

**int a[sz];**

**public:**

**void initialize();**

**struct Pointer;**

**friend struct Pointer;**

**struct Pointer {**

**private:**

**Holder\* h;**

**int\* p;**

**public:**

**void initialize(Holder\* h);**

**// Move around in the array:**

**void next();**

**void previous();**

**void top();**

**void end();**

**// Access values:**

**int read();**

**void set(int i);**

**};**

**};**

**void Holder::initialize() {**

**memset(a, 0, sz \* sizeof(int));**

**}**

**void Holder::Pointer::initialize(Holder\* rv) {**

**h = rv;**

**p = rv->a;**

**}**

**void Holder::Pointer::next() {**

**if(p <&(h->a[sz - 1])) p++;**

**}**

**void Holder::Pointer::previous() {**

**if(p >&(h->a[0])) p--;**

**}**

**void Holder::Pointer::top() {**

**p = &(h->a[0]);**

**}**

**void Holder::Pointer::end() {**

**p = &(h->a[sz - 1]);**

**}**

**int Holder::Pointer::read() {**

**return \*p;**

**}**

**void Holder::Pointer::set(int i) {**

**\*p = i;**

**}**

**int main() {**

**Holder h;**

**Holder::Pointer hp, hp2;**

**int i;**

**h.initialize();**

**hp.initialize(&h);**

**hp2.initialize(&h);**

**for(i = 0; i < sz; i++) {**

**hp.set(i);**

**hp.next();**

**}**

**hp.top();**

**hp2.end();**

**for(i = 0; i < sz; i++) {**

**cout << "hp = " << hp.read()**

**<< ", hp2 = " << hp2.read() << endl;**

**hp.next();**

**hp2.previous();**

**}**

**} ///:~**

***Listing 5-5. Comparing struct and class***

**//: C05:Class.cpp**

**// Similarity of struct and class**

**struct A {**

**private:**

**int i, j, k;**

**public:**

**int f();**

**void g();**

**};**

**int A::f() {**

**return(i + j + k);**

**}**

**void A::g() {**

**i = j = k = 0;**

**}**

**// Identical results are produced with:**

**class B {**

**int i, j, k;**

**public:**

**int f();**

**void g();**

**};**

**int B::f() {**

**return(i + j + k);**

**}**

**void B::g() {**

**i = j = k = 0;**

**}**

**int main() {**

**A a;**

**B b;**

**a.f(); a.g();**

**b.f(); b.g();**

**} ///:~**

***Listing 5-6. Updating Stash to Use Access Control***

**//: C05:Stash.h**

**// Converted to use access control**

**#ifndef STASH\_H**

**#define STASH\_H**

**class Stash {**

**int size; // Size of each space**

**int quantity; // Number of storage spaces**

**int next; // Next empty space**

**// Dynamically allocated array of bytes:**

**unsigned char\* storage;**

**void inflate(int increase);**

**public:**

**void initialize(int size);**

**void cleanup();**

**int add(void\* element);**

**void\* fetch(int index);**

**int count();**

**};**

**#endif // STASH\_H ///:~**

***Listing 5-7. Turning Stack into a class***

**//: C05:Stack2.h**

**// Nested structs via linked list**

**#ifndef STACK2\_H**

**#define STACK2\_H**

**class Stack {**

**struct Link {**

**void\* data;**

**Link\* next;**

**void initialize(void\* dat, Link\* nxt);**

**}\* head;**

**public:**

**void initialize();**

**void push(void\* dat);**

**void\* peek();**

**void\* pop();**

**void cleanup();**

**};**

**#endif // STACK2\_H ///:~**

***Listing 5-8. Handling Classes***

**//: C05:Handle.h**

**// Handle classes header file**

**#ifndef HANDLE\_H**

**#define HANDLE\_H**

**class Handle {**

**struct Hire; // Class declaration only**

**Hire\* smile;**

**public:**

**void initialize();**

**void cleanup();**

**int read();**

**void change(int);**

**};**

**#endif // HANDLE\_H ///:~**

***Listing 5-9. Handling Implementation***

**//: C05:Handle.cpp {O}**

**// Handle implementation**

**#include "Handle.h" // To be INCLUDED from Header FILE above**

**#include "../require.h" // To be INCLUDED from Header FILE in *Chapter 3***

**// Define Handle's implementation:**

**struct Handle::Hire {**

**int i;**

**};**

**void Handle::initialize() {**

**smile = new Hire;**

**smile->i = 0;**

**}**

**void Handle::cleanup() {**

**delete smile;**

**}**

**int Handle::read() {**

**return smile->i;**

**}**

**void Handle::change(int x) {**

**smile->i = x;**

**} ///:~**

***Listing 5-10. Using the Handle Class***

**//: C05:UseHandle.cpp**

**//{L} Handle**

**// Use the Handle class**

**#include "Handle.h"**

**int main() {**

**Handle u;**

**u.initialize();**

**u.read();**

**u.change(1);**

**u.cleanup();**

**} ///:~**