Chapter 11

REFERENCES AND THE COPY CONSTRUCTOR

***Listing 11-1. Illustrating A Freestanding Reference***

**//: C11:FreeStandingReferences.cpp**

**#include <iostream>**

**using namespace std;**

**// Ordinary free-standing reference:**

**int y;**

**int& r = y;**

**// When a reference is created, it must**

**// be initialized to a live object.**

**// However, you can also say:**

**const int& q = 12; // (1)**

**// References are tied to someone else's storage:**

**int x = 0; // (2)**

**int& a = x; // (3)**

**int main() {**

**cout << "x = " << x << ", a = " << a << endl;**

**a++;**

**cout << "x = " << x << ", a = " << a << endl;**

**} ///:~*****Listing 11-2. Illustrating Simple C++ References***

**//: C11:Reference.cpp**

**// Simple C++ references**

**int \*f(int \*x) {**

**(\*x)++;**

**return x; // Safe, x is outside this scope**

**}**

**int &g(int &x) {**

**x++; // Same effect as in f()**

**return x; // Safe, outside this scope**

**}**

**int &h() {**

**int q;**

**//! return q; // Error**

**static int x;**

**return x; // Safe, x lives outside this scope**

**}**

**int main() {**

**int a = 0;**

**f(&a); // Ugly (but explicit)**

**g(a); // Clean (but hidden)**

**} ///:~**

***Listing 11-3. Illustrating Passing of References as const***

**//: C11:ConstReferenceArguments.cpp**

**// Passing references as const**

**void f(int&) {}**

**void g(const int&) {}**

**int main() {**

**//! f(1); // Error**

**g(1);**

**} ///:~**

***Listing 11-4. Illustrating Reference to A Pointer***

**//: C11:ReferenceToPointer.cpp**

**#include <iostream>**

**using namespace std;**

**void increment(int \*&i) { i++; }**

**int main() {**

**int \*i = 0;**

**cout << "i = " << i<< endl;**

**increment(i);**

**cout << "i = " << i << endl;**

**} ///:~**

***Listing 11-5. Illustrating Passing of Big Structures***

**//: C11:PassingBigStructures.cpp**

**struct Big {**

**char buf[100];**

**int i;**

**long d;**

**} B, B2;**

**Big bigfun(Big b) {**

**b.i = 100; // Do something to the argument**

**return b;**

**}**

**int main() {**

**B2 = bigfun(B);**

**} ///:~*****Listing 11-6. Illustrating a Class that Counts its Objects***

***(by Including a Static Data Member)*//: C11:HowMany.cpp**

**// A class that counts its objects**

**#include <fstream>**

**#include <string>**

**using namespace std;**

**ofstream out("HowMany.out");**

**classHowMany {**

**static int objectCount;**

**public:**

**HowMany() { objectCount++; }**

**static void print(const string& msg = "") {**

**if(msg.size() != 0) out << msg << ": ";**

**out << "objectCount = "**

**<< objectCount << endl;**

**}**

**~HowMany() {**

**objectCount--;**

**print("~HowMany()");**

**}**

**};**

**intHowMany::objectCount = 0;**

**// Pass and return BY VALUE:**

**HowManyf(HowMany x) {**

**x.print("x argument inside f()");**

**return x;**

**}**

**int main() {**

**HowMany h;**

**HowMany::print("after construction of h");**

**HowMany h2 = f(h);**

**HowMany::print("after call to f()");**

**} ///:~*****Listing 11-7. Illustrating How to Fix the Problem***

**//: C11:HowMany2.cpp**

**// The copy-constructor**

**#include <fstream>**

**#include <string>**

**using namespace std;**

**ofstream out("HowMany2.out");**

**class HowMany2 {**

**string name; // Object identifier**

**static int objectCount;**

**public:**

**HowMany2(const string &id = "") : name(id) {**

**++objectCount;**

**print("HowMany2()");**

**}**

**~HowMany2() {**

**--objectCount;**

**print("~HowMany2()");**

**}**

**// The copy-constructor:**

**HowMany2(const HowMany2 &h) : name(h.name) {**

**name += " copy";**

**++objectCount;**

**print("HowMany2(const HowMany2&)");**

**}**

**void print(const string &msg = "") const {**

**if(msg.size() != 0)**

**out << msg << endl;**

**out << '\t' << name << ": "**

**<< "objectCount = "**

**<< objectCount << endl;**

**}**

**};**

**int HowMany2::objectCount = 0;**

**// Pass and return BY VALUE:**

**HowMany2 f(HowMany2 x) {**

**x.print("x argument inside f()");**

**out << "Returning from f()" << endl;**

**return x;**

**}**

**int main() {**

**HowMany2 h("h");**

**out << "Entering f()" << endl;**

**HowMany2 h2 = f(h);**

**h2.print("h2 after call to f()");**

**out << "Call f(), no return value" << endl;**

**f(h);**

**out << "After call to f()" << endl;**

**} ///:~**

**(using iostream)*Listing 11-8. Illustrating Addition of Line Numbers to Any File***

***(using iostream)***

**//: C11:Linenum.cpp**

**//{T} Linenum.cpp**

**// Add line numbers**

**#include "../require.h" // To be INCLUDED from Header FILE in *Chapter 9***

**#include <vector>**

**#include <string>**

**#include <fstream>**

**#include <iostream>**

**#include <cmath>**

**using namespace std;**

**int main(intargc, char\* argv[]) {**

**requireArgs(argc, 1, "Usage: linenum file\n"**

**"Adds line numbers to file");**

**ifstream in(argv[1]);**

**assure(in, argv[1]);**

**string line;**

**vector<string> lines;**

**while(getline(in, line)) // Read in entire file**

**lines.push\_back(line);**

**if(lines.size() == 0) return 0;**

**intnum = 0;**

**// Number of lines in file determines width:**

**const int width =**

**int(log10((double)lines.size())) + 1;**

**for(int i = 0; i < lines.size(); i++) {**

**cout.setf(ios::right, ios::adjustfield);**

**cout.width(width);**

**cout << ++num << ") " << lines[i] << endl;**

**}**

**} ///:~**

***Listing 11-9. Illustrating Creation of The Default Copy-Constructor***

**//: C11:DefaultCopyConstructor.cpp**

**/// Automatic creation of the copy-constructor**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**class WithCC { // With copy-constructor**

**public:**

**// Explicit default constructor required:**

**WithCC() {}**

**WithCC(const WithCC&) {**

**cout << "WithCC(WithCC&)" << endl;**

**}**

**};**

**class WoCC { // Without copy-constructor**

**string id;**

**public:**

**WoCC(const string& ident = "") : id(ident) {}**

**void print(const string& msg = "") const {**

**if(msg.size() != 0) cout << msg << ": ";**

**cout << id << endl;**

**}**

**};**

**class Composite {**

**WithCC withcc; // Embedded objects**

**WoCC wocc;**

**public:**

**Composite() : wocc("Composite()") {}**

**void print(const string& msg = "") const {**

**wocc.print(msg);**

**}**

**};**

**int main() {**

**Composite c;**

**c.print("Contents of c");**

**cout << "Calling Composite copy-constructor"**

**<< endl;**

**Composite c2 = c; // Calls copy-constructor**

**c2.print("Contents of c2");**

**} ///:~*****Listing 11-10. Illustrating Prevention of Copy-Construction***

**//: C11:NoCopyConstruction.cpp**

**/// Preventing copy-construction**

**class NoCC {**

**int i;**

**NoCC(const NoCC&); // No definition**

**public:**

**NoCC(int ii = 0) : i(ii) {}**

**};**

**void f(NoCC);**

**int main() {**

**NoCC n;**

**//! f(n); // Error: copy-constructor called**

**//! NoCC n2 = n; // Error: c-c called**

**//! NoCC n3(n); // Error: c-c called**

**} ///:~*****Listing 11-11. Illustrating Syntax for Selection of Members in a Simple Structure***

**//: C11:SimpleStructure.cpp**

**struct Simple { int a; };**

**int main() {**

**Simple so, \*sp = &so;**

**sp->a;**

**so.a;**

**} ///:~**

***Listing 11-12. Illustrating Pointer-to-Member Syntax for Data Members***  ***(also, demonstrates creation & use of pointers-to-members)***

**//: C11:PointerToMemberData.cpp**

**#include <iostream>**

**using namespace std;**

**class Data {**

**public:**

**int a, b, c;**

**void print() const {**

**cout << "a = " << a << ", b = " << b**

**<< ", c = " << c << endl;**

**}**

**};**

**int main() {**

**Data d, \*dp = &d;**

**int Data::\*pmInt = &Data::a;**

**dp->\*pmInt = 47;**

**pmInt = &Data::b;**

**d.\*pmInt = 48;**

**pmInt = &Data::c;**

**dp->\*pmInt = 49;**

**dp->print();**

**} ///:~**

***Listing 11-13. Illustrating Pointer-to-Member syntax for Member Functions***

**//: C11:PmemFunDefinition.cpp**

**class Simple2 {**

**public:**

**int f(float) const { return 1; }**

**};**

**int (Simple2::\*fp)(float) const;**

**int (Simple2::\*fp2)(float) const = &Simple2::f;**

**int main() {**

**fp = &Simple2::f;**

**} ///:~**

***Listing 11-14. Illustrating Selection of Member Functions at Runtime***

**//: C11:PointerToMemberFunction.cpp**

**#include <iostream>**

**using namespace std;**

**class Widget {**

**public:**

**void f(int) const { cout << "Widget::f()\n"; }**

**void g(int) const { cout << "Widget::g()\n"; }**

**void h(int) const { cout << "Widget::h()\n"; }**

**void i(int) const { cout << "Widget::i()\n"; }**

**};**

**int main() {**

**Widget w;**

**Widget \*wp = &w;**

**void (Widget::\*pmem)(int) const = &Widget::h;**

**(w.\*pmem)(1);**

**(wp->\*pmem)(2);**

**} ///:~**

***Listing 11-15. Illustrating Use of a Pointer-to-Member Inside the Class***

**//: C11:PointerToMemberFunction2.cpp**

**#include <iostream>**

**using namespace std;**

**class Widget {**

**void f(int) const { cout<< "Widget::f()\n"; }**

**void g(int) const { cout<< "Widget::g()\n"; }**

**void h(int) const { cout<< "Widget::h()\n"; }**

**void i(int) const { cout<< "Widget::i()\n"; }**

**enum { cnt = 4 };**

**void (Widget::\*fptr[cnt])(int) const;**

**public:**

**Widget() {**

**fptr[0] = &Widget::f; // Full spec required**

**fptr[1] = &Widget::g;**

**fptr[2] = &Widget::h;**

**fptr[3] = &Widget::i;**

**}**

**void select(int i, int j) {**

**if(i < 0 || i >= cnt) return;**

**(this->\*fptr[i])(j);**

**}**

**int count() { return cnt; }**

**};**

**int main() {**

**Widget w;**

**for(int i = 0; i < w.count(); i++)**

**w.select(i, 47);**

**} ///:**