Chapter 8

CONSTANTS

***Listing 8-1. Using const for Safety***

**//: C08:Safecons.cpp**

**/// Using const for safety**

**#include <iostream>**

**using namespace std;**

**const int i = 100; // Typical constant**

**const int j = i + 10; // Value from const expr**

**long address = (long)&j; // Forces storage**

**char buf[j + 10]; // Still a const expression**

**int main() {**

**cout << "type a character & CR:";**

**const char c = cin.get(); // Can't change**

**const char c2 = c + 'a';**

**cout << c2;**

**// ...**

**} ///:~*Listing 8-2. Constants and Aggregates***

**//: C08:Constag.cpp**

**// Constants and aggregates**

**const int i[] = { 1, 2, 3, 4 };**

**//! float f[i[3]]; // Illegal**

**struct S { int i, j; };**

**const S s[] = { { 1, 2 }, { 3, 4 } };**

**//! double d[s[1].j]; // Illegal**

**int main() {} ///:~**

***Listing 8-3. Pointers***

**//: C08:ConstPointers.cpp**

**const int\* u;**

**int const\* v;**

**int d = 1;**

**int\* const w = &d;**

**const int\* const x = &d;  // (1)**

**int const\* const x2 = &d; // (2)**

**int main() {} ///:~**

## 

***Listing 8-4. Pointer Assignment***

**//: C08:PointerAssignment.cpp**

**int d = 1;**

**const int e = 2;**

**int\* u = &d; // OK -- d not const**

**//! int\* v = &e; // Illegal -- e const**

**int\* w = (int\*)&e; // Legal but bad practice**

**int main() {} ///:~**

***Listing 8-5. Returning consts by Value***

**//: C08:Constval.cpp**

**// Returning consts by value**

**// has no meaning for built-in types**

**int f3() { return 1; }**

**const int f4() { return 1; }**

**int main() {**

**const int j = f3(); // Works fine**

**int k = f4(); // But this works fine too!**

**} ///:~**

***Listing 8-6. Constant Returned by Value***

**//: C08:ConstReturnValues.cpp**

**// Constant return by value**

**// Result cannot be used as an lvalue**

**class X {**

**int i;**

**public:**

**X(int ii = 0);**

**void modify();**

**};**

**X::X(int ii) { i = ii; }**

**void X::modify() { i++; }**

**X f5() {**

**return X();**

**}**

**const X f6() {**

**return X();**

**}**

**void f7(X& x) { // Pass by non-const reference**

**x.modify();**

**}**

**int main() {**

**f5() = X(1); // OK -- non-const return value**

**f5().modify(); // OK**

**//! f7(f5()); // Causes warning or error**

**// Causes compile-time errors:**

**//! f7(f5());**

**//! f6() = X(1);**

**//! f6().modify();**

**//! f7(f6());**

**} ///:~**

***Listing 8-7. const Pointers as Function Arguments and Return Values***

**//: C08:ConstPointer.cpp**

**// Constant pointer arg/return**

**void t(int\*) {}**

**void u(const int\* cip) {**

**//! \*cip = 2; // Illegal -- modifies value**

**int i = \*cip; // OK -- copies value**

**//! int\* ip2 = cip; // Illegal: non-const**

**}**

**const char\* v() {**

**// Returns address of static character array:**

**return "result of function v()";**

**}**

**const int\* const w() {**

**static int i;**

**return &i;**

**}**

**int main() {**

**int x = 0;**

**int\* ip = &x;**

**const int\* cip = &x;**

**t(ip); // OK**

**//! t(cip); // Not OK**

**u(ip); // OK**

**u(cip); // Also OK**

**//! char\* cp = v(); // Not OK**

**const char\* ccp = v(); // OK**

**//! int\* ip2 = w(); // Not OK**

**const int\* const ccip = w(); // OK**

**const int\* cip2 = w(); // OK**

**//! \*w() = 1; // Not OK}**

**///:~*Listing 8-8. Temporaries***

**//: C08:ConstTemporary.cpp**

**// Temporaries are const**

**class X {};**

**X f() { return X(); } // Return by value**

**void g1(X&) {} // Pass by non-const reference**

**void g2(const X&) {} // Pass by const reference**

**int main() {**

**// Error: const temporary created by f():**

**//! g1(f());**

**// OK: g2 takes a const reference:**

**g2(f());**

**} ///:~**

***Listing 8-9***. ***Initializing const in Classes***

**//: C08:ConstInitialization.cpp**

**// Initializing const in classes**

**#include <iostream>**

**using namespace std;**

**class Fred {**

**const int size;**

**public:**

**Fred(int sz);**

**void print();**

**};**

**Fred::Fred(int sz) : size(sz) {}**

**void Fred::print() { cout << size << endl; }**

**int main() {**

**Fred a(1), b(2), c(3);**

**a.print(), b.print(), c.print();**

**} ///:~**

***Listing 8-10. Built-in Constructors***

**//: C08:BuiltInTypeConstructors.cpp**

**#include <iostream>**

**using namespace std;**

**class B {**

**int i;**

**public:**

**B(int ii);**

**void print();**

**};**

**B::B(int ii) : i(ii) {}**

**void B::print() { cout << I << endl; }**

**int main() {**

**B a(1), b(2);**

**float pi(3.14159);**

**a.print(); b.print();**

**cout << pi << endl;**

**} ///:~**

***Listing 8-11. Encapsulating***

//: C08:EncapsulatingTypes.cpp

**#include <iostream>**

**using namespace std;**

**class Integer {**

**int i;**

**public:**

**Integer(int ii = 0);**

**void print();**

**};**

**Integer::Integer(int ii) : i(ii) {}**

**void Integer::print() { cout << i << ' '; }**

**int main() {**

**Integer i[100];**

**for(int j = 0; j < 100; j++)**

**i[j].print();**

**} ///:~*Listing 8-12. Using a static const***

**//: C08:StringStack.cpp**

**// Using static const to create a**

**// compile-time constant inside a class**

**#include <string>**

**#include <iostream>**

**using namespace std;**

**class StringStack {**

**static const int size = 100;**

**const string\* stack[size];**

**int index;**

**public:**

**StringStack();**

**void push(const string\* s);**

**const string\* pop();**

**};**

**StringStack::StringStack() : index(0) {**

**memset(stack, 0, size \* sizeof(string\*));**

**}**

**void StringStack::push(const string\* s) {**

**if(index < size)**

**stack[index++] = s;**

**}**

**const string\* StringStack::pop() {**

**if(index > 0) {**

**const string\* rv = stack[--index];**

**stack[index] = 0;**

**return rv;**

**}**

**return 0;**

**}**

**string iceCream[] = {**

**"pralines& cream",**

**"fudge ripple",**

**"jamocha almond fudge",**

**"wild mountain blackberry",**

**"raspberry sorbet",**

**"lemon swirl",**

**"rocky road",**

**"deep chocolate fudge"**

**};**

**const int iCsz =**

**sizeof iceCream / sizeof \*iceCream;**

**int main() {**

**StringStack ss;**

**for(int i = 0; i < iCsz; i++)**

**ss.push(&iceCream[i]);**

**const string\* cp;**

**while((cp = ss.pop()) != 0)**

**cout << \*cp << endl;**

**} ///:~**

***Listing 8-13. enum Hack***

**//: C08:EnumHack.cpp**

**#include <iostream>**

**using namespace std;**

**class Bunch {**

**enum { size = 1000 };**

**int i[size];**

**};**

**int main() {**

**cout << "sizeof(Bunch) = " << sizeof(Bunch)**

**<< ", sizeof(i[1000]) = "**

**<< sizeof(int[1000]) << endl;**

**} ///:~**

***Listing 8-14. const Member Functions***

**//: C08:ConstMember.cpp**

**class X {**

**int i;**

**public:**

**X(int ii);**

**int f() const;**

**};**

**X::X(int ii) : i(ii) {}**

**int X::f() const { return i; }**

**int main() {**

**X x1(10);**

**const X x2(20);**

**x1.f();**

**x2.f();**

**} ///:~**

***Listing 8-15. Contrasting a const and a non-const Member Function***

**//: C08:Quoter.cpp**

**/ Random quote selection**

**#include <iostream>**

**#include <cstdlib> // Random number generator**

**#include <ctime> // To seed random generator**

**using namespace std;**

**class Quoter {**

**int lastquote;**

**public:**

**Quoter();**

**int lastQuote() const;**

**const char\* quote();**

**};**

**Quoter::Quoter(){**

**lastquote = -1;**

**srand(time(0)); // Seed random number generator**

**}**

**int Quoter::lastQuote() const {**

**return lastquote;**

**}**

**const char\* Quoter::quote() {**

**static const char\* quotes[] = {**

**"Are we having fun yet?",**

**"Doctors always know best",**

**"Is it ... Atomic?",**

**"Fear is obscene",**

**"There is no scientific evidence "**

**"to support the idea "**

**"that life is serious",**

**"Things that make us happy, make us wise",**

**};**

**const int qsize = sizeof quotes/sizeof \*quotes;**

**int qnum = rand() % qsize;**

**while(lastquote >= 0 && qnum == lastquote)**

**qnum = rand() % qsize;**

**return quotes[lastquote = qnum];**

**}**

**int main() {**

**Quoter q;**

**const Quoter cq;**

**cq.lastQuote(); // OK**

**//! cq.quote(); // Not OK; non const function**

**for(int i = 0; i < 20; i++)**

**cout << q.quote() << endl;**

**} ///:~*Listing 8-16. Casting Away const Attribute***

**//: C08:Castaway.cpp**

**// "Casting away" const attribute**

**class Y {**

**int i;**

**public:**

**Y();**

**void f() const;**

**};**

**Y::Y() { i = 0; }**

**void Y::f() const {**

**//! i++; // Error -- const member function**

**((Y\*)this)->i++; // OK: cast away const-ness**

**// Better: use C++ explicit cast syntax:**

**(const\_cast<Y\*>(this))->i++;**

**}**

**int main() {**

**const Y yy;**

**yy.f(); // Actually changes it!**

**} ///:~**

***Listing 8-17. The mutable Keyword***

**//: C08:Mutable.cpp**

**// The "mutable" keyword**

**class Z {**

**int i;**

**mutable int j;**

**public:**

**Z();**

**void f() const;**

**};**

**Z::Z() : i(0), j(0) {}**

**void Z::f() const {**

**//! i++; // Error -- const member function**

**j++; // OK: mutable**

**}**

**int main() {**

**const Z zz;**

**zz.f(); // Actually changes it!**

**} ///:~**

***Listing 8-18. The volatile Keyword***

**//: C08:Volatile.cpp**

**// The volatile keyword**

**classComm {**

**const volatile unsigned char byte;**

**volatile unsigned char flag;**

**enum { bufsize = 100 };**

**unsigned char buf[bufsize];**

**int index;**

**public:**

**Comm();**

**void isr() volatile;**

**char read(int index) const;**

**};**

**Comm::Comm() : index(0), byte(0), flag(0) {}**

**// Only a demo; won't actually work**

**// as an interrupt service routine:**

**void Comm::isr() volatile {**

**flag = 0;**

**buf[index++] = byte;**

**// Wrap to beginning of buffer:**

**if(index >= bufsize) index = 0;**

**}**

**charComm::read(int index) const {**

**if(index < 0 || index >= bufsize)**

**return 0;**

**return buf[index];**

**}**

**int main() {**

**volatile Comm Port;**

**Port.isr(); // OK**

**//! Port.read(0); // Error, read() not volatile**

**}**