Chapter 9

# **INLINE FUNCTIONS**

***Listing 9-1. Macro Side Effects***

**//: C09:MacroSideEffects.cpp**

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

**// *ahead* (Section: Improved error**

**// checking) Or *Chapter 3***

**#include <fstream>**

**using namespace std;**

**#define BAND(x) (((x)>5 && (x)<10) ? (x) : 0)**

**int main() {**

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

**assure(out, "macro.out");**

**for(int i = 4; i < 11; i++) {**

**int a = i;**

**out << "a = " << a << endl << '\t';**

**out << "BAND(++a)=" << BAND(++a) << endl;**

**out << "\t a = " << a << endl;**

**}**

**} ///:~**

***Listing 9-2. Inlines Inside Classes***

**//: C09:Inline.cpp**

**// Inlines inside classes**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**class Point {**

**int i, j, k;**

**public:**

**Point(): i(0), j(0), k(0) {}**

**Point(int ii, int jj, int kk)**

**: i(ii), j(jj), k(kk) {}**

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

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

**cout << "i = " << i << ", "**

**<< "j = " << j << ", "**

**<< "k = " << k << endl;**

**}**

**};**

**int main() {**

**Point p, q(1,2,3);**

**p.print("value of p");**

**q.print("value of q");**

**} ///:~*****Listing 9-3. Inline Access Functions***

**//: C09:Access.cpp**

**// Inline access functions**

**class Access {**

**int i;**

**public:**

**int read() const { return i; }**

**void set(int ii) { i = ii; }**

**};**

**int main() {**

**Access A;**

**A.set(100);**

**int x = A.read();**

**} ///:~**

***Listing 9-4. Accessors and Mutators***

**//: C09:Rectangle.cpp**

**// Accessors & mutators**

**class Rectangle {**

**int wide, high;**

**public:**

**Rectangle(int w = 0, int h = 0)**

**: wide(w), high(h) {}**

**int width() const { return wide; } // Read**

**void width(int w) { wide = w; } // Set**

**int height() const { return high; } // Read**

**void height(int h) { high = h; } // Set**

**};**

**int main() {**

**Rectangle r(19, 47);**

**// Change width & height:**

**r.height(2 \* r.width());**

**r.width(2 \* r.height());**

**} ///:~**

***Listing 9-5. Using get and set***

**//: C09:Rectangle2.cpp**

**// Accessors & mutators with "get" and "set"**

**class Rectangle {**

**int width, height;**

**public:**

**Rectangle(int w = 0, int h = 0)**

**: width(w), height(h) {}**

**int getWidth() const { return width; }**

**void setWidth(int w) { width = w; }**

**int getHeight() const { return height; }**

**void setHeight(int h) { height = h; }**

**};**

**int main() {**

**Rectangle r(19, 47);**

**// Change width & height:**

**r.setHeight(2 \* r.getWidth());**

**r.setWidth(2 \* r.getHeight());**

**} ///:~**

***Listing 9-6. Using Time Functions***

**//: C09:Cpptime.h**

**// A simple time class**

**#ifndef CPPTIME\_H**

**#define CPPTIME\_H**

**#include <ctime>**

**#include <cstring>**

**class Time {**

**std::time\_t t;**

**std::tm local;**

**char asciiRep[26];**

**unsigned char lflag, aflag;**

**void updateLocal() {**

**if(!lflag) {**

**local = \*std::localtime(&t);**

**lflag++;**

**}**

**}**

**void updateAscii() {**

**if(!aflag) {**

**updateLocal();**

**std::strcpy(asciiRep,std::asctime(&local));**

**aflag++;**

**}**

**}**

**public:**

**Time() { mark(); }**

**void mark() {**

**lflag = aflag = 0;**

**std::time(&t);**

**}**

**const char\* ascii() {**

**updateAscii();**

**return asciiRep;**

**}**

**// Difference in seconds:**

**int delta(Time\* dt) const {**

**return int(std::difftime(t, dt->t));**

**}**

**int daylightSavings() {**

**updateLocal();**

**return local.tm\_isdst;**

**}**

**int dayOfYear() { // Since January 1**

**updateLocal();**

**return local.tm\_yday;**

**}**

**int dayOfWeek() { // Since Sunday**

**updateLocal();**

**return local.tm\_wday;**

**}**

**int since1900() { // Years since 1900**

**updateLocal();**

**return local.tm\_year;**

**}**

**int month() { // Since January**

**updateLocal();**

**return local.tm\_mon;**

**}**

**int dayOfMonth() {**

**updateLocal();**

**return local.tm\_mday;**

**}**

**int hour() { // Since midnight, 24-hour clock**

**updateLocal();**

**return local.tm\_hour;**

**}**

**int minute() {**

**updateLocal();**

**return local.tm\_min;**

**}**

**int second() {**

**updateLocal();**

**return local.tm\_sec;**

**}**

**};**

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

***Listing 9-7. Testing a Simple Time Class***

**//: C09:Cpptime.cpp**

**// Testing a simple time class**

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

**#include <iostream>**

**using namespace std;**

**int main() {**

**Time start;**

**for(int i = 1; i < 1000; i++) {**

**cout << i << ' ';**

**if(i%10 == 0) cout << endl;**

**}**

**Time end;**

**cout << endl;**

**cout << "start = " << start.ascii();**

**cout << "end = " << end.ascii();**

**cout << "delta = " << end.delta(&start);**

**} ///:~**

***Listing 9-8. Stash Header File (with Inline Functions)***

**//: C09:Stash4.h**

**// Inline functions**

**#ifndef STASH4\_H**

**#define STASH4\_H**

**#include "../require.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:**

**Stash(int sz) : size(sz), quantity(0),**

**next(0), storage(0) {}**

**Stash(int sz, int initQuantity) : size(sz),**

**quantity(0), next(0), storage(0) {**

**inflate(initQuantity);**

**}**

**Stash::~Stash() {**

**if(storage != 0)**

**delete []storage;**

**}**

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

**void\* fetch(int index) const {**

**require(0 <= index, "Stash::fetch (-)index");**

**if(index >= next)**

**return 0; // To indicate the end**

**// Produce pointer to desired element:**

**return &(storage[index \* size]);**

**}**

**int count() const { return next; }**

**};**

**#endif // STASH4\_H ///:~**

***Listing 9-9. Stash Source Code cpp File (with Inline Functions)***

**//: C09:Stash4.cpp {O}**

**#include "Stash4.h" // To be INCLUDED from Header FILE above**

**#include <iostream>**

**#include <cassert>**

**using namespace std;**

**const int increment = 100;**

**int Stash::add(void\* element) {**

**if(next >= quantity) // Enough space left?**

**inflate(increment);**

**// Copy element into storage,**

**// starting at next empty space:**

**int startBytes = next \* size;**

**unsigned char\* e = (unsigned char\*)element;**

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

**storage[startBytes + i] = e[i];**

**next++;**

**return(next - 1); // Index number**

**}**

**void Stash::inflate(int increase) {**

**assert(increase >= 0);**

**if(increase == 0) return;**

**int newQuantity = quantity + increase;**

**int newBytes = newQuantity \* size;**

**int oldBytes = quantity \* size;**

**unsigned char\* b = new unsigned char[newBytes];**

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

**b[i] = storage[i]; // Copy old to new**

**delete [](storage); // Release old storage**

**storage = b; // Point to new memory**

**quantity = newQuantity; // Adjust the size**

**} ///:~**

***Listing 9-10. Testing the Stash (with Inline Functions)***

**//: C09:Stash4Test.cpp**

**//{L} Stash4**

**#include "Stash4.h"**

**#include "../require.h"**

**#include <fstream>**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**int main() {**

**Stash intStash(sizeof(int));**

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

**intStash.add(&i);**

**for(int j = 0; j <intStash.count(); j++)**

**cout << "intStash.fetch(" << j << ") = "**

**<< \*(int\*)intStash.fetch(j)**

**<< endl;**

**const int bufsize = 80;**

**Stash stringStash(sizeof(char) \* bufsize, 100);**

**ifstream in("Stash4Test.cpp");**

**assure(in, "Stash4Test.cpp");**

**string line;**

**while(getline(in, line))**

**stringStash.add((char\*)line.c\_str());**

**int k = 0;**

**char\* cp;**

**while((cp = (char\*)stringStash.fetch(k++))!=0)**

**cout << "stringStash.fetch(" << k << ") = "**

**<< cp << endl;**

**} ///:~**

***Listing 9-11. Stack Header File (with Inline Functions***

**//: C09:Stack4.h**

**// With inlines**

**#ifndef STACK4\_H**

**#define STACK4\_H**

**#include "../require.h"**

**class Stack {**

**struct Link {**

**void\* data;**

**Link\* next;**

**Link(void\* dat, Link\* nxt):**

**data(dat), next(nxt) {}**

**}\* head;**

**public:**

**Stack() : head(0) {}**

**~Stack() {**

**require(head == 0, "Stack not empty");**

**}**

**void push(void\* dat) {**

**head = new Link(dat, head);**

**}**

**void\* peek() const {**

**return head ? head->data : 0;**

**}**

**void\* pop() {**

**if(head == 0) return 0;**

**void\* result = head->data;**

**Link\* oldHead = head;**

**head = head->next;**

**delete oldHead;**

**return result;**

**}**

**};**

**#endif // STACK4\_H ///:~**

***Listing 9-12. Testing the Stack (with Inline Functions)***

**//: C09:Stack4Test.cpp**

**//{T} Stack4Test.cpp**

**#include "Stack4.h" // To be INCLUDED from Header FILE above**

**#include "../require.h"**

**#include <fstream>**

**#include <iostream>**

**#include <string>**

**using namespace std;**

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

**requireArgs(argc, 1); // File name is argument**

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

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

**Stack textlines;**

**string line;**

**// Read file and store lines in the stack:**

**while(getline(in, line))**

**textlines.push(new string(line));**

**// Pop the lines from the stack and print them:**

**string\* s;**

**while((s = (string\*)textlines.pop()) != 0) {**

**cout << \*s << endl;**

**delete s;**

**}**

**} ///:~**

***Listing 9-13. Inline Evaluation Order***

**//: C09:EvaluationOrder.cpp**

**class Forward {**

**int i;**

**public:**

**Forward() : i(0) {}**

**// Call to undeclared function:**

**int f() const { return g() + 1; }**

**int g() const { return i; }**

**};**

**int main() {**

**Forward frwd;**

**frwd.f();**

**} ///:~**

***Listing 9-14. Illustrating Hidden Activities in Inlines (for a Class with Member Objects)***

**/// Hidden activities in inlines**

**#include <iostream>**

**using namespace std;**

**class Member {**

**int i, j, k;**

**public:**

**Member(int x = 0) : i(x), j(x), k(x) {}**

**~Member() { cout << "~Member" << endl; }**

**};**

**class WithMembers {**

**Member q, r, s; // Have constructors**

**int i;**

**public:**

**WithMembers(int ii) : i(ii) {} // Trivial?**

**~WithMembers() {**

**cout << "~WithMembers" << endl;**

**}**

**};**

**int main() {**

**WithMembers wm(1);**

**} ///:~*****Listing 9-15. Using the inline Keyword***

**//: C09:Noinsitu.cpp**

**// Removing in situ functions**

**class Rectangle {**

**int width, height;**

**public:**

**Rectangle(int w = 0, int h = 0);**

**int getWidth() const;**

**void setWidth(int w);**

**int getHeight() const;**

**void setHeight(int h);**

**};**

**inline Rectangle::Rectangle(int w, int h)**

**: width(w), height(h) {}**

**inline int Rectangle::getWidth() const {**

**return width;**

**}**

**inline void Rectangle::setWidth(int w) {**

**width = w;**

**}**

**inline int Rectangle::getHeight() const {**

**return height;**

**}**

**inline void Rectangle::setHeight(int h) {**

**height = h;**

**}**

**int main() {**

**Rectangle r(19, 47);**

**// Transpose width & height:**

**int iHeight = r.getHeight();**

**r.setHeight(r.getWidth());**

**r.setWidth(iHeight);**

**} ///:~*****Listing 9-16. The require.h Header File***

**// : : require.h**

**// Test for error conditions in programs**

**// Local "using namespace std" for old compilers**

**#ifndef REQUIRE\_H**

**#define REQUIRE\_H**

**#include <cstdio>**

**#include <cstdlib>**

**#include <fstream>**

**#include <string>**

**inline void require(bool requirement,**

**const std::string& msg = "Requirement failed"){**

**using namespace std;**

**if (!requirement) {**

**fputs(msg.c\_str(), stderr);**

**fputs("\n", stderr);**

**exit(1);**

**}**

**}**

**inline void requireArgs(int argc, int args,**

**const std::string& msg =**

**"Must use %d arguments") {**

**using namespace std;**

**if (argc != args + 1) {**

**fprintf(stderr, msg.c\_str(), args);**

**fputs("\n", stderr);**

**exit(1);**

**}**

**}**

**inline void requireMinArgs(intargc, intminArgs,**

**const std::string& msg =**

**"Must use at least %d arguments") {**

**using namespace std;**

**if(argc < minArgs + 1) {**

**fprintf(stderr, msg.c\_str(), minArgs);**

**fputs("\n", stderr);**

**exit(1);**

**}**

**}**

**inline void assure(std::ifstream& in,**

**const std::string& filename = "") {**

**using namespace std;**

**if(!in) {**

**fprintf(stderr, "Could not open file %s\n",**

**filename.c\_str());**

**exit(1);**

**}**

**}**

**inline void assure(std::ofstream& out,**

**const std::string& filename = "") {**

**using namespace std;**

**if(!out) {**

**fprintf(stderr, "Could not open file %s\n",**

**filename.c\_str());**

**exit(1);**

**}**

**}**

**#endif // REQUIRE\_H ///:~*****Listing 9-17. Testing require.h***

**//: C09:ErrTest.cpp**

**//{T} ErrTest.cpp**

**// Testing require.h**

**#include "../require.h"**

**#include <fstream>**

**using namespace std;**

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

**int i = 1;**

**require(i, "value must be nonzero");**

**requireArgs(argc, 1);**

**requireMinArgs(argc, 1);**

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

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

**// Use the file name**

**ifstream nofile("nofile.xxx");**

**// Fails:**

**//! assure(nofile);**

**// The default argument**

**ofstream out("tmp.txt");**

**assure(out);**

**} ///:~**