Chapter 4

DATA ABSTRACTION

*Listing 4-1. CStash*

**//: C04:CLib.h**

**// Header file for a C-like library**

**// An array-like entity created at runtime**

**typedefstructCStashTag {**

**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;**

**} CStash;**

**void initialize(CStash\* s, int size);**

**void cleanup(CStash\* s);**

**int add(CStash\* s, const void\* element);**

**void\* fetch(CStash\* s, int index);**

**int count(CStash\* s);**

**void inflate(CStash\* s, int increase);**

**///:~**

***Listing 4-2. Source Code for the Implementation File***

**//: C04:CLib.cpp {O}**

**// Implementation of example C-like library**

**// Declare structure and functions:**

**#include "CLib.h"**

**#include <iostream>**

**#include <cassert>**

**using namespace std;**

**// Quantity of elements to add**

**// when increasing storage:**

**Const int increment = 100;**

**void initialize(CStash\* s, intsz) {**

**s->size = sz;**

**s->quantity = 0;**

**s->storage = 0;**

**s->next = 0;**

**}**

**int add(CStash\* s, const void\* element) {**

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

**inflate(s, increment);**

**// Copy element into storage,**

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

**int startBytes = s->next \* s->size;**

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

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

**s->next++;**

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

**}**

**void\* fetch(CStash\* s, int index) {**

**// Check index boundaries:**

**assert(0 <= index);**

**if(index >= s->next)**

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

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

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

**}**

**int count(CStash\* s) {**

**return s->next; // Elements in CStash**

**}**

**void inflate(CStash\* s, int increase) {**

**assert(increase > 0);**

**int newQuantity = s->quantity + increase;**

**int newBytes = newQuantity \* s->size;**

**int oldBytes = s->quantity \* s->size;**

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

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

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

**delete [](s->storage); // Old storage**

**s->storage = b; // Point to new memory**

**s->quantity = newQuantity;**

**}**

**void cleanup(CStash\* s) {**

**if(s->storage != 0) {**

**cout << "freeing storage" << endl;**

**delete []s->storage;**

**}**

**} ///:~**

***Listing 4-3. Testing the C-like library with Two CStashes***

**//: C04:CLibTest.cpp**

**//{L} CLib**

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

**#include <fstream>**

**#include <iostream>**

**#include <string>**

**#include <cassert>**

**using namespace std;**

**int main() {**

**// Define variables at the beginning**

**// of the block, as in C:**

**CStashintStash, stringStash;**

**int i;**

**char\* cp;**

**ifstream in;**

**string line;**

**const int bufsize = 80;**

**// Now remember to initialize the variables:**

**initialize(&intStash, sizeof(int));**

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

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

**for(i = 0; i < count(&intStash); i++)**

**cout << "fetch(&intStash, " << i << ") = "**

**<< \*(int\*)fetch(&intStash, i)**

**<< endl;**

**// Holds 80-character strings:**

**initialize(&stringStash, sizeof(char)\*bufsize);**

**in.open("CLibTest.cpp");**

**assert(in);**

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

**add(&stringStash, line.c\_str());**

**i = 0;**

**while((cp = (char\*)fetch(&stringStash, i++))!=0)**

**cout << "fetch(&stringStash, " << i << ") = "**

**<< cp << endl;**

**cleanup(&intStash);**

**cleanup(&stringStash);**

**} ///:~**

***Listing 4-4. Converting C-like library to C++***

**//: C04:CppLib.h**

**struct 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;**

**// Functions!**

**void initialize(int size);**

**void cleanup();**

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

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

**int count();**

**void inflate(int increase);**

**}; ///:~**

***Listing 4-5. Using the Scope Resolution Operator in Function Definitions***

**//: C04:CppLib.cpp {O}**

**// C library converted to C++**

**// Declare structure and functions:**

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

**#include <iostream>**

**#include <cassert>**

**using namespace std;**

**// Quantity of elements to add**

**// when increasing storage:**

**const int increment = 100;**

**void Stash::initialize(int sz) {**

**size = sz;**

**quantity = 0;**

**storage = 0;**

**next = 0;**

**}**

**int Stash::add(const 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::fetch(int index) {**

**// Check index boundaries:**

**assert(0 <= index);**

**if(index >= next)**

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

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

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

**}**

**int Stash::count() {**

**return next; // Number of elements in CStash**

**}**

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

**assert(increase > 0);**

**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; // Old storage**

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

**quantity = newQuantity;**

**}**

**void Stash::cleanup() {**

**if(storage != 0) {**

**cout << "freeing storage" << endl;**

**delete []storage;**

**}**

**} ///:~**

***Listing 4-6. Using the C++ Version of CStash***

**//: C04:CppLibTest.cpp**

**//{L} CppLib**

**// Test of C++ library**

**#include "CppLib.h"**

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

**#include <fstream>**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**int main() {**

**Stash intStash;**

**intStash.initialize(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;**

**// Holds 80-character strings:**

**Stash stringStash;**

**const int bufsize = 80;**

**stringStash.initialize(sizeof(char) \* bufsize);**

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

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

**string line;**

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

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

**int k = 0;**

**char\* cp;**

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

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

**<< cp << endl;**

**intStash.cleanup();**

**stringStash.cleanup();**

**} ///:~**

***Listing 4-7. Finding the Sizes of structs Using the sizeof Operator***

**//: C04:Sizeof.cpp**

**// Sizes of structs**

**#include "CLib.h"**

**#include "CppLib.h"**

**#include <iostream>**

**using namespace std;**

**struct A {**

**int i[100];**

**};**

**struct B {**

**void f();**

**};**

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

**int main() {**

**cout << "sizeof struct A = " << sizeof(A)**

**<< " bytes" << endl;**

**cout << "sizeof struct B = " <<sizeof(B)**

**<< " bytes" << endl;**

**cout << "sizeof CStash in C = "**

**<< sizeof(CStash) << " bytes" << endl;**

**cout << "sizeof Stash in C++ = "**

**<< sizeof(Stash) << " bytes" << endl;**

**} ///:~**

***Listing 4-8. Simple Header that Prevents Re-definition***

**//: C04:Simple.h**

**// Simple header that prevents redefinition**

**#ifndef SIMPLE\_H**

**#define SIMPLE\_H**

**struct Simple {**

**int i,j,k;**

**initialize() { i = j = k = 0; }**

**};**

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

*Listing 4-9. Nested Structures*

**//: C04:Stack.h**

**// Nested struct in linked list**

**#ifndef STACK\_H**

**#define STACK\_H**

**struct Stack {**

**struct Link {**

**void\* data;**

**Link\* next;**

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

**}\* head;**

**void initialize();**

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

**void\* peek();**

**void\* pop();**

**void cleanup();**

**};**

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

***Listing 4-10. Linked List with Nesting including Definitions of Member Functions***

**//: C04:Stack.cpp {O}**

**// Linked list with nesting**

**// Includes definitions of member functions**

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

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

**using namespace std;**

**void**

**Stack::Link::initialize(void\* dat, Link\* nxt) {**

**data = dat;**

**next = nxt;**

**}**

**void Stack::initialize() { head = 0; }**

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

**Link\* newLink = new Link;**

**newLink->initialize(dat, head);**

**head = newLink;**

**}**

**void\* Stack::peek() {**

**require(head != 0, "Stack empty");**

**return head->data;**

**}**

**void\* Stack::pop() {**

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

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

**Link\* oldHead = head;**

**head = head->next;**

**delete oldHead;**

**return result;**

**}**

**void Stack::cleanup() {**

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

**} ///:~**

***Listing 4-11. Testing the Stack***

**//: C04:StackTest.cpp**

**//{L} Stack**

**//{T} StackTest.cpp**

**// Test of nested linked list**

**#include "Stack.h"**

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

**#include<fstream>**

**#include<iostream>**

**#include<string>**

**using namespace std;**

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

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

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

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

**Stack textlines;**

**textlines.initialize();**

**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;**

**}**

**textlines.cleanup();**

**} ///:~**

***Listing 4-12. Global Scope Resolution***

**//: C04:Scoperes.cpp**

**// Global scope resolution for a variable**

**// As well as a function**

**int a;**

**void f() {}**

**struct S {**

**int a;**

**void f();**

**};**

**void S::f() {**

**::f(); // Would be recursive otherwise!**

**::a++; // Select the global a**

**a--; // The a at struct scope**

**}**

**int main() { S s; f(); } ///:~**