Chapter 6

INITIALIZATION AND CLEANUP

***Listing 6-1. Constructors and Destructors***

**//: C06:Constructor1.cpp**

**// Demonstrates features of constructors & destructors**

**#include <iostream>**

**using namespace std;**

**class Tree {**

**int height;**

**public:**

**Tree(int initialHeight); // Constructor**

**~Tree(); // Destructor**

**void grow(int years);**

**void printsize();**

**};**

**Tree::Tree(int initialHeight) {**

**height = initialHeight;**

**}**

**Tree::~Tree() {**

**cout << "inside Tree destructor" << endl;**

**printsize();**

**}**

**void Tree::grow(int years) {**

**height += years;**

**}**

**void Tree::printsize() {**

**cout << "Tree height is " << height << endl;**

**}**

**int main() {**

**cout << "before opening brace" << endl;**

**{**

**Tree t(12);**

**cout << "after Tree creation" << endl;**

**t.printsize();**

**t.grow(4);**

**cout << "before closing brace" << endl;**

**}**

**cout << "after closing brace" << endl;**

**} ///:~**

***Listing 6-2. Defining Variables Anywhere***

**//: C06:DefineInitialize.cpp**

**// Demonstrates that you can define variables anywhere**

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

**#include <iostream>**

**#include <string>**

**using namespace std;**

**class G {**

**int i;**

**public:**

**G(int ii);**

**};**

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

**int main() {**

**cout << "initialization value? ";**

**int retval = 0;**

**cin >> retval;**

**require(retval != 0);**

**int y = retval + 3;**

**G g(y);**

**} ///:~**

***Listing 6-3. Jumping Past Constructors is not Allowed in C++***

**//: C06:Nojump.cpp**

**// Demonstrates that you can't jump past constructors in C++**

**class X {**

**public:**

**X();**

**};**

**X::X() {}**

**void f(int i) {**

**if(i < 10) {**

**//! goto jump1; // Error: goto bypasses init**

**}**

**X x1; // Constructor called here**

**jump1:**

**switch(i) {**

**case 1 :**

**X x2; // Constructor called here**

**break;**

**//! case 2 : // Error: case bypasses init**

**X x3; // Constructor called here**

**break;**

**}**

**}**

**int main() {**

**f(9);**

**f(11);**

**}///:~**

***Listing 6-4. Stash Header Using Constructors and Destructors***

**//: C06:Stash2.h**

**// Demonstrates Stash header file with constructors & destructors**

**#ifndef STASH2\_H**

**#define STASH2\_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 size);**

**~Stash();**

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

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

**int count();**

**};**

**#endif // STASH2\_H ///:~**

***Listing 6-5. Implementing Stash with Constructors & Destructors***

**//: C06:Stash2.cpp {O}**

**// Demonstrates implementation of Stash**

**// with constructors & destructors**

**#include "Stash2.h" // To be INCLUDED from Header FILE above**

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

**#include <iostream>**

**#include <cassert>**

**using namespace std;**

**const int increment = 100;**

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

**size = sz;**

**quantity = 0;**

**storage = 0;**

**next = 0;**

**}**

**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::fetch(int index) {**

**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 Stash::count() {**

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

**}**

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

**require(increase > 0,**

**"Stash::inflate zero or negative increase");**

**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;**

**}**

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

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

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

**delete []storage;**

**}**

**} ///:~**

***Listing 6-6. Testing Stash (with Constructors & Destructors)***

**//: C06:Stash2Test.cpp**

**//{L} Stash2**

**// Demonstrates testing of Stash**

**// (with constructors & destructors)**

**#include "Stash2.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++)**

**int Stash.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);**

**ifstream in("Stash2Test.cpp");**

**assure(in, " Stash2Test.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 6-7. Stack with Constructors/Destructors***

**//: C06:Stack3.h**

**// Demonstrates the modified header file**

**#ifndef STACK3\_H**

**#define STACK3\_H**

**class Stack {**

**struct Link {**

**void\* data;**

**Link\* next;**

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

**~Link();**

**}\* head;**

**public:**

**Stack();**

**~Stack();**

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

**void\* peek();**

**void\* pop();**

**};**

**#endif // STACK3\_H ///:~**

***Listing 6-8. Implementing Stack with Constructors/Destructors***

**//: C06:Stack3.cpp {O}**

**// Demonstrates implementation of Stack**

**// with constructors/destructors**

**#include "Stack3.h" // To be INCLUDED from Header FILE above**

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

**using namespace std;**

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

**data = dat;**

**next = nxt;**

**}**

**Stack::Link::~Link() { }**

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

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

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

**}**

**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;**

**}**

**Stack::~Stack() {**

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

**} ///:~**

***Listing 6-9. Testing Stack (with Constructors/Destructors)***

**//: C06:Stack3Test.cpp**

**//{L} Stack3**

**//{T} Stack3Test.cpp**

**// Demonstrates testing of Stack**

**// (with constructors/destructors)**

**#include "Stack3.h"**

**#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 6-10. Using Multiple Constructor Arguments (with Aggregate Initialization)***

**//: C06:Multiarg.cpp**

**// Demonstrates use of multiple constructor arguments**

**// (with aggregate initialization)**

**#include <iostream>**

**using namespace std;**

**class Z {**

**int i, j;**

**public:**

**Z(int ii, int jj);**

**void print();**

**};**

**Z::Z(int ii, int jj) {**

**i = ii;**

**j = jj;**

**}**

**void Z::print() {**

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

**}**

**int main() {**

**Z zz[] = { Z(1,2), Z(3,4), Z(5,6), Z(7,8) };**

**for(int i = 0; i < (sizeof (zz) / sizeof (\*zz)); i++)**

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

**} ///:~**

***Listing 6-11. Generating Automatic Default Constructor***

**//: C06:AutoDefaultConstructor.cpp**

**// Demonstrates automatically-generated default constructor**

**class V {**

**int i; // private**

**}; // No constructor**

**int main() {**

**V v, v2[10];**

**}**

**///:~**