Chapter 3

THE C IN C++

***Listing 3-1. Several return Statements***

**//: C03:Return.cpp**

**// Use of "return"**

**#include <iostream>**

**using namespace std;**

**char cfunc(int i) {**

**if(i == 0)**

**return 'a';**

**if(i == 1)**

**return 'g';**

**if(i == 5)**

**return 'z';**

**return 'c';**

**}**

**int main() {**

**cout << "type an integer: ";**

**int val;**

**cin >> val;**

**cout << cfunc(val) << endl;**

**} ///:~**

***Listing 3-2. Using if and if-else***

**//: C03:Ifthen.cpp**

**// Demonstration of if and if-else conditionals**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int i;**

**cout << "type a number and 'Enter'" << endl;**

**cin >> i;**

**if(i > 5)**

**cout << "It's greater than 5" << endl;**

**else**

**if(i < 5)**

**cout << "It's less than 5 " << endl;**

**else**

**cout << "It's equal to 5 " << endl;**

**cout << "type a number and 'Enter'" << endl;**

**cin >> i;**

**if(i < 10)**

**if(i > 5) // "if" is just another statement**

**cout << "5 < i < 10" << endl;**

**else**

**cout << "i <= 5" << endl;**

**else // Matches "if(i < 10)"**

**cout << "i >= 10" << endl;**

**} ///:~**

***Listing 3-3. Using while***

**//: C03:Guess.cpp**

**// Guess a number (demonstrates "while")**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int secret = 15;**

**int guess = 0;**

**// "!=" is the "not-equal" conditional:**

**while(guess != secret) { // Compound statement**

**cout << "guess the number: ";**

**cin >> guess;**

**}**

**cout << "You guessed it!" << endl;**

**} ///:~**

***Listing 3-4. Using do-while***

**//: C03:Guess2.cpp**

**// The guess program using do-while**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int secret = 15;**

**int guess; // No initialization needed here**

**do {**

**cout << "guess the number: ";**

**cin >> guess; // Initialization happens**

**} while(guess != secret);**

**cout << "You got it!" << endl;**

**} ///:~**

***Listing 3-5. Using for***

**//: C03:Charlist.cpp**

**// Display all the ASCII characters**

**// Demonstrates "for"**

**#include <iostream>**

**using namespace std;**

**int main() {**

**for(int i = 0; i < 128; i = i + 1)**

**if (i != 26) // ANSI Terminal Clear screen**

**cout << " value: " << i**

**<< " character: "**

**<< char(i) // Type conversion**

**<< endl;**

**} ///:~**

***Listing 3-6. Using the break and continue Keywords***

**//: C03:Menu.cpp**

**// Simple menu program demonstrating**

**// the use of "break" and "continue"**

**#include <iostream>**

**using namespace std;**

**int main() {**

**char c; // To hold response**

**while(true) {**

**cout << "MAIN MENU:" << endl;**

**cout << "l: left, r: right, q: quit -> ";**

**cin >> c;**

**if(c == 'q')**

**break; // Out of "while(1)"**

**if(c == 'l') {**

**cout << "LEFT MENU:" << endl;**

**cout << "select a or b: ";**

**cin >> c;**

**if(c == 'a') {**

**cout << "you chose 'a'" << endl;**

**continue; // Back to main menu**

**}**

**if(c == 'b') {**

**cout << "you chose 'b'" << endl;**

**continue; // Back to main menu**

**}**

**else {**

**cout << "you didn't choose a or b!"**

**<< endl;**

**continue; // Back to main menu**

**}**

**}**

**if(c == 'r') {**

**cout << "RIGHT MENU:" << endl;**

**cout << "select c or d: ";**

**cin >> c;**

**if(c == 'c') {**

**cout << "you chose 'c'" << endl;**

**continue; // Back to main menu**

**}**

**if(c == 'd') {**

**cout << "you chose 'd'" << endl;**

**continue; // Back to main menu**

**}**

**else {**

**cout << "you didn't choose c or d!"**

**<< endl;**

**continue; // Back to main menu**

**}**

**}**

**cout << "you must type l or r or q!" << endl;**

**}**

**cout << "quitting menu..." << endl;**

**} ///:~**

***Listing 3-7. Using switch***

**//: C03:Menu2.cpp**

**// A menu using a switch statement**

**#include <iostream>**

**using namespace std;**

**int main() {**

**bool quit = false; // Flag for quitting**

**while(quit == false) {**

**cout << "Select a, b, c or q to quit: ";**

**char response;**

**cin >> response;**

**switch(response) {**

**case 'a' : cout << "you chose 'a'" << endl;**

**break;**

**case 'b' : cout << "you chose 'b'" << endl;**

**break;**

**case 'c' : cout << "you chose 'c'" << endl;**

**break;**

**case 'q' : cout << "quitting menu" << endl;**

**quit = true;**

**break;**

**default : cout << "Please use a,b,c or q!"**

**<< endl;**

**}**

**}**

**} ///:~**

***Listing 3-8. Using goto***

**//: C03:gotoKeyword.cpp**

**// The infamous goto is supported in C++**

**#include <iostream>**

**using namespace std;**

**int main() {**

**long val = 0;**

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

**for(int j = 1; j < 100; j += 10) {**

**val = i \* j;**

**if(val > 47000)**

**goto DOWN;**

**// Break would only go to the outer 'for'**

**}**

**}**

**DOWN: // A label**

**cout << val << endl;**

**} ///:~**

***Listing 3-9. Using Recursion***

**//: C03:CatsInHats.cpp**

**// Simple demonstration of recursion**

**#include <iostream>**

**using namespace std;**

**void removeHat(char cat) {**

**for(char c = 'A'; c < cat; c++)**

**cout << " ";**

**if(cat <= 'Z') {**

**cout << "cat " << cat << endl;**

**removeHat(cat + 1); // Recursive call**

**} else**

**cout << "VOOM!!!" << endl;**

**}**

**int main() {**

**removeHat('A');**

**} ///:~**

***Listing 3-10. Auto-Increment and Auto-Decrement***

**//: C03:AutoIncrement.cpp**

**// Shows use of auto-increment**

**// and auto-decrement operators.**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int i = 0;**

**int j = 0;**

**cout << ++i << endl; // Pre-increment**

**cout << j++ << endl; // Post-increment**

**cout << --i << endl; // Pre-decrement**

**cout << j-- << endl; // Post decrement**

**} ///:~**

***Listing 3-11. Basic Data Types***

**//: C03:Basic.cpp**

**// Defining the four basic data**

**// types in C and C++**

**int main() {**

**// Definition without initialization:**

**char protein;**

**int carbohydrates;**

**float fiber;**

**double fat;**

**// Simultaneous definition & initialization:**

**char pizza = 'A', pop = 'Z';**

**int dongdings = 100, twinkles = 150, heehos = 200;**

**float chocolate = 3.14159;**

**// Exponential notation:**

**double fudge\_ripple = 6e-4;**

**} ///:~**

***Listing 3-12. Using Specifiers***

**//: C03:Specify.cpp**

**// Demonstrates the use of specifiers**

**#include <iostream>**

**using namespace std;**

**int main() {**

**char c;**

**unsigned char cu;**

**int i;**

**unsigned int iu;**

**short int is;**

**short iis; // Same as short int**

**unsigned short int isu;**

**unsigned short iisu;**

**long int il;**

**long iil; // Same as long int**

**unsigned long int ilu;**

**unsigned long iilu;**

**float f;**

**double d;**

**long double ld;**

**cout**

**<< "\n char = " << sizeof(c)**

**<< "\n unsigned char = " << sizeof(cu)**

**<< "\n int = " << sizeof(i)**

**<< "\n unsigned int = " << sizeof(iu)**

**<< "\n short = " << sizeof(is)**

**<< "\n unsigned short = " << sizeof(isu)**

**<< "\n long = " << sizeof(il)**

**<< "\n unsigned long = " << sizeof(ilu)**

**<< "\n float = " << sizeof(f)**

**<< "\n double = " << sizeof(d)**

**<< "\n long double = " << sizeof(ld)**

**<< endl;**

**} ///:~**

***Listing 3-13. A Simple Program***

**//: C03:YourPets1.cpp**

**#include <iostream>**

**using namespace std;**

**int dog, cat, bird, fish;**

**void f(int pet) {**

**cout << "pet id number: " << pet << endl;**

**}**

**int main() {**

**int i, j, k;**

**} ///:~**

***Listing 3-14. Modifying the Program***

**//: C03:YourPets2.cpp**

**#include <iostream>**

**using namespace std;**

**int dog, cat, bird, fish;**

**void f(int pet) {**

**cout << "pet id number: " << pet << endl;**

**}**

**int main() {**

**int i, j, k;**

**cout << "f(): " << (long)&f << endl;**

**cout << "dog: " << (long)&dog << endl;**

**cout << "cat: " << (long)&cat << endl;**

**cout << "bird: " << (long)&bird << endl;**

**cout << "fish: " << (long)&fish << endl;**

**cout << "i: " << (long)&i << endl;**

**cout << "j: " << (long)&j << endl;**

**cout << "k: " << (long)&k << endl;**

**} ///:~**

***Listing 3-15. Pass-by-value***

**//: C03:PassByValue.cpp**

**#include <iostream>**

**using namespace std;**

**void f(int a) {**

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

**a = 5;**

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

**}**

**int main() {**

**int x = 47;**

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

**f(x);**

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

**} ///:~**

***Listing 3-16. Passing an Alias***

**//: C03:PassAddress.cpp**

**#include <iostream>**

**using namespace std;**

**void f(int\* p) {**

**cout << "p = " << p << endl;**

**cout << "\*p = " << \*p << endl;**

**\*p = 5;**

**cout << "p = " << p << endl;**

**}**

**int main() {**

**int x = 47;**

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

**cout << "&x = " <<&x << endl;**

**f(&x);**

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

**} ///:~**

***Listing 3-17. Pass-by-reference***

**//: C03:PassReference.cpp**

**#include <iostream>**

**using namespace std;**

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

**cout << "r = " << r << endl;**

**cout << "&r = " <<&r << endl;**

**r = 5;**

**cout << "r = " << r << endl;**

**}**

**int main() {**

**int x = 47;**

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

**cout << "&x = " <<&x << endl;**

**f(x); // Looks like pass-by-value,**

**// is actually pass by reference**

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

**} ///:~**

***Listing 3-18. All Possible Combinations***

**//: C03:AllDefinitions.cpp**

**// All possible combinations of basic data types,**

**// specifiers, pointers and references**

**#include <iostream>**

**using namespace std;**

**void f1(char c, int i, float f, double d);**

**void f2(short int si, long int li, long double ld);**

**void f3(unsigned char uc, unsigned int ui,**

**unsigned short int usi, unsigned long int uli);**

**void f4(char\* cp, int\* ip, float\* fp, double\* dp);**

**void f5(short int\* sip, long int\* lip,**

**long double\* ldp);**

**void f6(unsigned char\* ucp, unsigned int\* uip,**

**unsigned short int\* usip,**

**unsigned long int\* ulip);**

**void f7(char& cr, int& ir, float& fr, double& dr);**

**void f8(short int& sir, long int& lir,**

**long double& ldr);**

**void f9(unsigned char& ucr, unsigned int& uir,**

**unsigned short int& usir,**

**unsigned long int& ulir);**

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

***Listing 3-19. void Pointer***

**//: C03:VoidPointer.cpp**

**int main() {**

**void\* vp;**

**char c;**

**int i;**

**float f;**

**double d;**

**// The address of ANY type can be**

**// assigned to a void pointer:**

**vp = &c;**

**vp = &i;**

**vp = &f;**

**vp = &d;**

**} ///:~**

***Listing 3-20. Cast from void Pointer***

**//: C03:CastFromVoidPointer.cpp**

**int main() {**

**int i = 99;**

**void\* vp = &i;**

**// Can't dereference a void pointer:**

**// \*vp = 3; // Compile time error**

**// Must cast back to int before dereferencing:**

**\*((int\*)vp) = 3;**

**} ///:~**

***Listing 3-21. Scoping***

**//: C03:Scope.cpp**

**// How variables are scoped**

**int main() {**

**int scp1;**

**// scp1 visible here**

**{**

**// scp1 still visible here**

**//.....**

**int scp2;**

**// scp2 visible here**

**//.....**

**{**

**// scp1 & scp2 still visible here**

**//..**

**int scp3;**

**// scp1, scp2 & scp3 visible here**

**// ...**

**} // <-- scp3 destroyed here**

**// scp3 not available here**

**// scp1 & scp2 still visible here**

**// ...**

**} // <-- scp2 destroyed here**

**// scp3 & scp2 not available here**

**// scp1 still visible here**

**//..**

**} // <-- scp1 destroyed here**

**///:~**

***Listing 3-22. On-the-fly Variable Definitions***

**//: C03:OnTheFly.cpp**

**// On-the-fly variable definitions**

**#include <iostream>**

**using namespace std;**

**int main() {**

**//..**

**{ // Begin a new scope**

**int q = 0; // C requires definitions here**

**//..**

**// Define at point of use:**

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

**q++; // q comes from a larger scope**

**// Definition at the end of the scope:**

**int p = 12;**

**}**

**int p = 1; // A different p**

**} // End scope containing q & outer p**

**cout << "Type characters:" << endl;**

**while(char c = cin.get() != 'q') {**

**cout << c << " wasn't it" << endl;**

**if(char x = c == 'a' || c == 'b')**

**cout << "You typed a or b" << endl;**

**else**

**cout << "You typed " << x << endl;**

**}**

**cout << "Type A, B, or C" << endl;**

**switch(int i = cin.get()) {**

**case 'A': cout << "Snap" << endl; break;**

**case 'B': cout << "Crackle" << endl; break;**

**case 'C': cout << "Pop" << endl; break;**

**default: cout << "Not A, B or C!" << endl;**

**}**

**} ///:~**

***Listing 3-23. Using Global Variables***

**//: C03:Global.cpp**

**//{L} Global2**

**// Demonstration of global variables**

**#include <iostream>**

**using namespace std;**

**int globe;**

**void func();**

**int main() {**

**globe = 12;**

**cout << globe << endl;**

**func(); // Modifies globe**

**cout << globe << endl;**

**} ///:~**

***Listing 3-24. Accessing Global Variables***

**//: C03:Global2.cpp {O}**

**// Accessing external global variables**

**extern int globe;**

**// (The linker resolves the reference)**

**void func() {**

**globe = 47;**

**} ///:~**

***Listing 3-25. Static Variables***

**//: C03:Static.cpp**

**// Using a static variable in a function**

**#include <iostream>**

**using namespace std;**

**void func() {**

**static int i = 0;**

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

**}**

**int main() {**

**for(int x = 0; x < 10; x++)**

**func();**

**} ///:~**

***Listing 3-26. File Scope Demonstration***

**//: C03:FileStatic.cpp**

**// File scope demonstration. Compiling and**

**// linking this file with FileStatic2.cpp**

**// will cause a linker error**

**// File scope means only available in this file:**

**static int fs;**

**int main() {**

**fs = 1;**

**} ///:~**

***Listing 3-27. More of the Demonstration***

**//: C03:FileStatic2.cpp {O}**

**// Trying to reference fs**

**extern int fs;**

**void func() {**

**fs = 100;**

**} ///:~**

***Listing 3-28. The extern Keyword***

**//: C03:Forward.cpp**

**// Forward function & data declarations**

**#include <iostream>**

**using namespace std;**

**// This is not actually external, but the**

**// compiler must be told it exists somewhere:**

**extern int i;**

**extern void func();**

**int main() {**

**i = 0;**

**func();**

**}**

**int i; // The data definition**

**void func() {**

**i++;**

**cout << i;**

**} ///:~**

***Listing 3-29. Using Mathematical Operators***

**//: C03:Mathops.cpp**

**// Mathematical operators**

**#include <iostream>**

**using namespace std;**

**// A macro to display a string and a value.**

**#define PRINT(STR, VAR) \**

**cout << STR " = " << VAR << endl**

**int main() {**

**int i, j, k;**

**float u, v, w; // Applies to doubles, too**

**cout << "enter an integer: ";**

**cin >> j;**

**cout << "enter another integer: ";**

**cin >> k;**

**PRINT("j",j); PRINT("k",k);**

**i = j + k; PRINT("j + k",i);**

**i = j - k; PRINT("j - k",i);**

**i = k / j; PRINT("k / j",i);**

**i = k \* j; PRINT("k \* j",i);**

**i = k % j; PRINT("k % j",i);**

**// The following only works with integers:**

**j %= k; PRINT("j %= k", j);**

**cout << "Enter a floating-point number: ";**

**cin >> v;**

**cout << "Enter another floating-point number:";**

**cin >> w;**

**PRINT("v",v); PRINT("w",w);**

**u = v + w; PRINT("v + w", u);**

**u = v - w; PRINT("v - w", u);**

**u = v \* w; PRINT("v \* w", u);**

**u = v / w; PRINT("v / w", u);**

**// The following works for ints, chars,**

**// and doubles too:**

**PRINT("u", u); PRINT("v", v);**

**u += v; PRINT("u += v", u);**

**u -= v; PRINT("u -= v", u);**

**u \*= v; PRINT("u \*= v", u);**

**u /= v; PRINT("u /= v", u);**

**} ///:~**

***Listing 3-30. Relational and Logical Operators***

**//: C03:Boolean.cpp**

**// Relational and logical operators.**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int i,j;**

**cout << "Enter an integer: ";**

**cin >> i;**

**cout << "Enter another integer: ";**

**cin >> j;**

**cout << "i > j is " << (i > j) << endl;**

**cout << "i < j is " << (i < j) << endl;**

**cout << "i >= j is " << (i >= j) << endl;**

**cout << "i <= j is " << (i <= j) << endl;**

**cout << "i == j is " << (i == j) << endl;**

**cout << "i != j is " << (i != j) << endl;**

**cout << "i && j is " << (i && j) << endl;**

**cout << "i || j is " << (i || j) << endl;**

**cout << " (i < 10) && (j < 10) is "**

**<< ((i < 10) && (j < 10)) << endl;**

**} ///:~**

***Listing 3-31. All of the Operators Involving Bits***

**//: C03:printBinary.h**

**// Display a byte in binary**

**void printBinary(const unsigned char val);**

**///:~**

//Here’s the implementation of the function:

**//: C03:printBinary.cpp {O}**

**#include <iostream>**

**void printBinary(const unsigned char val) {**

**for(int i = 7; i >= 0; i--)**

**if(val & (1 << i))**

**std::cout << "1";**

**else**

**std::cout << "0";**

**} ///:~**

***Listing 3-32. Bit-Manipulation Operators***

**//: C03:Bitwise.cpp**

**//{L} printBinary**

**// Demonstration of bit manipulation**

**#include "printBinary.h"**

**#include <iostream>**

**using namespace std;**

**// A macro to save typing:**

**#define PR(STR, EXPR) \**

**cout << STR; printBinary(EXPR); cout << endl;**

**int main() {**

**unsigned int getval;**

**unsigned char a, b;**

**cout << "Enter a number between 0 and 255: ";**

**cin >> getval; a = getval;**

**PR("a in binary: ", a);**

**cout << "Enter a number between 0 and 255: ";**

**cin >> getval; b = getval;**

**PR("b in binary: ", b);**

**PR("a | b = ", a | b);**

**PR("a & b = ", a & b);**

**PR("a ^ b = ", a ^ b);**

**PR("~a = ", ~a);**

**PR("~b = ", ~b);**

**// An interesting bit pattern:**

**unsigned char c = 0x5A;**

**PR("c in binary: ", c);**

**a |= c;**

**PR("a |= c; a = ", a);**

**b &= c;**

**PR("b &= c; b = ", b);**

**b ^= a;**

**PR("b ^= a; b = ", b);**

**} ///:~**

***Listing 3-33. Rotations***

**//: C03:Rotation.cpp {O}**

**// Perform left and right rotations**

**unsigned char rol(unsigned char val) {**

**int highbit;**

**if(val & 0x80) // 0x80 is the high bit only**

**highbit = 1;**

**else**

**highbit = 0;**

**// Left shift (bottom bit becomes 0):**

**val <<= 1;**

**// Rotate the high bit onto the bottom:**

**val |= highbit;**

**return val;**

**}**

**unsigned char ror(unsigned char val) {**

**int lowbit;**

**if(val & 1) // Check the low bit**

**lowbit = 1;**

**else**

**lowbit = 0;**

**val >>= 1; // Right shift by one position**

**// Rotate the low bit onto the top:**

**val |= (lowbit << 7);**

**return val;**

**} ///:~**

***Listing 3-34. Using the Comma Operator***

**//: C03:CommaOperator.cpp**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int a = 0, b = 1, c = 2, d = 3, e = 4;**

**a = (b++, c++, d++, e++);**

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

**// The parentheses are critical here. Without**

**// then, the statement will evaluate to:**

**(a = b++), c++, d++, e++;**

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

**} ///:~**

***Listing 3-35. Common Pitfall***

**//: C03:Pitfall.cpp**

**// Operator mistakes**

**int main() {**

**int a = 1, b = 1;**

**while(a = b) {**

**// ....**

**}**

**} ///:~**

***Listing 3-36. Simple Cast***

**//: C03:SimpleCast.cpp**

**int main() {**

**int b = 200;**

**unsigned long a = (unsigned long int)b;**

**} ///:~**

***Listing 3-37. Function Call Cast***

**//: C03:FunctionCallCast.cpp**

**int main() {**

**float a = float(200);**

**// This is equivalent to:**

**float b = (float)200;**

**} ///:~**

***Listing 3-38. Using static\_cast***

**//: C03:static\_cast.cpp**

**void func(int) {}**

**int main() {**

**int i = 0x7fff; // Max pos value = 32767**

**long l;**

**float f;**

**// (1) Typical castless conversions:**

**l = i;**

**f = i;**

**// Also works:**

**l = static\_cast<long>(i);**

**f = static\_cast<float>(i);**

**// (2) Narrowing conversions:**

**i = l; // May lose digits**

**i = f; // May lose info**

**// Says "I know," eliminates warnings:**

**i = static\_cast<int>(l);**

**i = static\_cast<int>(f);**

**char c = static\_cast<char>(i);**

**// (3) Forcing a conversion from void\* :**

**void\* vp = &i;**

**// Old way produces a dangerous conversion:**

**float\* fp = (float\*)vp;**

**// The new way is equally dangerous:**

**fp = static\_cast<float\*>(vp);**

**// (4) Implicit type conversions, normally**

**// performed by the compiler:**

**double d = 0.0;**

**int x = d; // Automatic type conversion**

**x = static\_cast<int>(d); // More explicit**

**func(d); // Automatic type conversion**

**func(static\_cast<int>(d)); // More explicit**

**} ///:~**

***Listing 3-39. Using const\_cast***

**//: C03:const\_cast.cpp**

**int main() {**

**const int i = 0;**

**int\* j = (int\*)&i; // Deprecated form**

**j = const\_cast<int\*>(&i); // Preferred**

**// Can't do simultaneous additional casting:**

**//! long\* l = const\_cast<long\*>(&i); // Error**

**volatile int k = 0;**

**int\* u = const\_cast<int\*>(&k);**

**} ///:~**

***Listing 3-40. Using reinterpret\_cast***

**//: C03:reinterpret\_cast.cpp**

**#include <iostream>**

**using namespace std;**

**const int sz = 100;**

**struct X { int a[sz]; };**

**void print(X\* x) {**

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

**cout << x->a[i] << ' ';**

**cout << endl << "--------------------" << endl;**

**}**

**int main() {**

**X x;**

**print(&x);**

**int\* xp = reinterpret\_cast<int\*>(&x);**

**for(int\* i = xp; i < xp + sz; i++)**

**\*i = 0;**

**// Can't use xp as an X\* at this point**

**// unless you cast it back:**

**print(reinterpret\_cast<X\*>(xp));**

**// In this example, you can also just use**

**// the original identifier:**

**print(&x);**

**} ///:~**

***Listing 3-41.Using sizeof***

**//: C03:sizeof.cpp**

**#include <iostream>**

**using namespace std;**

**int main() {**

**cout << "sizeof(double) = " << sizeof(double);**

**cout << ", sizeof(char) = " << sizeof(char);**

**} ///:~**

***Listing 3-42. Using sizeof with a Variable***

**//: C03:sizeofOperator.cpp**

**int main() {**

**int x;**

**int i = sizeof x;**

**} ///:~**

***Listing 3-43. A Simple struct***

**//: C03:SimpleStruct.cpp**

**struct Structure1 {**

**char c;**

**int i;**

**float f;**

**double d;**

**};**

**int main() {**

**struct Structure1 s1, s2;**

**s1.c = 'a'; // Select an element using a '.'**

**s1.i = 1;**

**s1.f = 3.14;**

**s1.d = 0.00093;**

**s2.c = 'a';**

**s2.i = 1;**

**s2.f = 3.14;**

**s2.d = 0.00093;**

**} ///:~**

***Listing 3-44. Another Simple struct***

**//: C03:SimpleStruct2.cpp**

**// Using typedef with struct**

**typedef struct {**

**char c;**

**int i;**

**float f;**

**double d;**

**} Structure2;**

**int main() {**

**Structure2 s1, s2;**

**s1.c = 'a';**

**s1.i = 1;**

**s1.f = 3.14;**

**s1.d = 0.00093;**

**s2.c = 'a';**

**s2.i = 1;**

**s2.f = 3.14;**

**s2.d = 0.00093;**

**} ///:~**

***Listing 3-45. Allowing a struct to Refer to Itself***

**//: C03:SelfReferential.cpp**

**// Allowing a struct to refer to itself**

**typedef struct SelfReferential {**

**int i;**

**SelfReferential\* sr; // Head spinning yet?**

**} SelfReferential;**

**int main() {**

**SelfReferential sr1, sr2;**

**sr1.sr = &sr2;**

**sr2.sr = &sr1;**

**sr1.i = 47;**

**sr2.i = 1024;**

**} ///:~**

***Listing 3-46. Using Pointers to structs***

**//: C03:SimpleStruct3.cpp**

**// Using pointers to structs**

**typedef struct Structure3 {**

**char c;**

**int i;**

**float f;**

**double d;**

**} Structure3;**

**int main() {**

**Structure3 s1, s2;**

**Structure3\* sp = &s1;**

**sp->c = 'a';**

**sp->i = 1;**

**sp->f = 3.14;**

**sp->d = 0.00093;**

**sp = &s2; // Point to a different struct object**

**sp->c = 'a';**

**sp->i = 1;**

**sp->f = 3.14;**

**sp->d = 0.00093;**

**} ///:~**

***Listing 3-47. Using enum***

**//: C03:Enum.cpp**

**// Keeping track of shapes**

**enum ShapeType {**

**circle,**

**square,**

**rectangle**

**}; // Must end with a semicolon like a struct**

**int main() {**

**ShapeType shape = circle;**

**// Activities here....**

**// Now do something based on what the shape is:**

**switch(shape) {**

**case circle: /\* circle stuff \*/ break;**

**case square: /\* square stuff \*/ break;**

**case rectangle: /\* rectangle stuff \*/ break;**

**}**

**} ///:~**

***Listing 3-48. The Size and Simple Use of a union***

**//: C03:Union.cpp**

**// The size and simple use of a union**

**#include <iostream>**

**using namespace std;**

**union Packed { // Declaration similar to a class**

**char i;**

**short j;**

**int k;**

**long l;**

**float f;**

**double d;**

**// The union will be the size of a**

**// double, since that's the largest element**

**}; // Semicolon ends a union, like a struct**

**int main() {**

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

**<< sizeof(Packed) << endl;**

**Packed x;**

**x.i = 'c';**

**cout << x.i << endl;**

**x.d = 3.14159;**

**cout << x.d << endl;**

**} ///:~**

***Listing 3-49. Arrays***

**//: C03:Arrays.cpp**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int a[10];**

**for(int i = 0; i < 10; i++) {**

**a[i] = i \* 10;**

**cout << "a[" << i << "] = " << a[i] << endl;**

**}**

**} ///:~**

***Listing 3-50. An Array of structs***

**//: C03:StructArray.cpp**

**// An array of struct**

**typedef struct {**

**int i, j, k;**

**}**

**ThreeDpoint;**

**int main() {**

**ThreeDpoint p[10];**

**for(int i = 0; i < 10; i++) {**

**p[i].i = i + 1;**

**p[i].j = i + 2;**

**p[i].k = i + 3;**

**}**

**} ///:~**

***Listing 3-51. Array Addresses***

**//: C03:ArrayAddresses.cpp**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int a[10];**

**cout << "sizeof(int) = " << sizeof(int) << endl;**

**for(int i = 0; i < 10; i++)**

**cout << "&a[" << i << "] = "**

**<< (long)&a[i] << endl;**

**} ///:~**

***Listing 3-52. Array Identifier***

**//: C03:ArrayIdentifier.cpp**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int a[10];**

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

**cout << "&a[0] =" <<&a[0] << endl;**

**} ///:~**

***Listing 3-53. Square-Bracket Syntax***

**//: C03:PointersAndBrackets.cpp**

**int main() {**

**int a[10];**

**int\* ip = a;**

**for(int i = 0; i < 10; i++)**

**ip[i] = i \* 10;**

**} ///:~**

***Listing 3-54. Array Arguments***

**//: C03:ArrayArguments.cpp**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**void func1(int a[], int size) {**

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

**a[i] = i \* i - i;**

**}**

**void func2(int\* a, int size) {**

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

**a[i] = i \* i + i;**

**}**

**void print(int a[], string name, int size) {**

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

**cout << name << "[" << i << "] = "**

**<< a[i] << endl;**

**}**

**int main() {**

**int a[5], b[5];**

**// Probably garbage values:**

**print(a, "a", 5);**

**print(b, "b", 5);**

**// Initialize the arrays:**

**func1(a, 5);**

**func1(b, 5);**

**print(a, "a", 5);**

**print(b, "b", 5);**

**// Notice the arrays are always modified:**

**func2(a, 5);**

**func2(b, 5);**

**print(a, "a", 5);**

**print(b, "b", 5);**

**} ///:~**

***Listing 3-55. Command-Line Arguments***

**//: C03:CommandLineArgs.cpp**

**#include <iostream>**

**using namespace std;**

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

**cout << "argc = " << argc << endl;**

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

**cout << "argv[" << i << "] = "**

**<< argv[i] << endl;**

**} ///:~**

***Listing 3-56. Using atoi( )***

**//: C03:ArgsToInts.cpp**

**// Converting command-line arguments to ints**

**#include <iostream>**

**#include <cstdlib>**

**using namespace std;**

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

**for(int i = 1; i < argc; i++)**

**cout << atoi(argv[i]) << endl;**

**} ///:~**

***Listing 3-57. Floating As Binary***

**//: C03:FloatingAsBinary.cpp**

**//{L} printBinary**

**//{T} 3.14159**

**#include "printBinary.h"**

**#include <cstdlib>**

**#include <iostream>**

**using namespace std;**

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

**if(argc != 2) {**

**cout << "Must provide a number" << endl;**

**exit(1);**

**}**

**double d = atof(argv[1]);**

**unsigned char\* cp =**

**reinterpret\_cast<unsigned char\*> (&d);**

**for(int i = sizeof(double)-1; i >= 0 ; i -= 2) {**

**printBinary(cp[i-1]);**

**printBinary(cp[i]);**

**}**

**} ///:~**

***Listing 3-58. Pointer Increment***

**//: C03:PointerIncrement.cpp**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int i[10];**

**double d[10];**

**int\* ip = i;**

**double\* dp = d;**

**cout << "ip = " << (long)ip << endl;**

**ip++;**

**cout << "ip = " << (long)ip << endl;**

**cout << "dp = " << (long)dp << endl;**

**dp++;**

**cout << "dp = " << (long)dp << endl;**

**} ///:~**

***Listing 3-59. Pointer Increment and an Array of structs***

**//: C03:PointerIncrement2.cpp**

**#include <iostream>**

**using namespace std;**

**typedef struct {**

**char c;**

**short s;**

**int i;**

**long l;**

**float f;**

**double d;**

**long double ld;**

**} Primitives;**

**int main() {**

**Primitives p[10];**

**Primitives\* pp = p;**

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

**<< sizeof(Primitives) << endl;**

**cout << "pp = " << (long)pp << endl;**

**pp++;**

**cout << "pp = " << (long)pp << endl;**

**} ///:~**

***Listing 3-60. Pointer Arithmetic***

**//: C03:PointerArithmetic.cpp**

**#include <iostream>**

**using namespace std;**

**#define P(EX) cout << #EX << ": " << EX << endl;**

**int main() {**

**int a[10];**

**for(int i = 0; i < 10; i++)**

**a[i] = i; // Give it index values**

**int\* ip = a;**

**P(\*ip);**

**P(\*++ip);**

**P(\*(ip + 5));**

**int\* ip2 = ip + 5;**

**P(\*ip2);**

**P(\*(ip2 - 4));**

**P(\*--ip2);**

**P(ip2 - ip); // Yields number of elements**

**} ///:~**

***Listing 3-61. Dynamic Debugging Flags***

**//: C03:DynamicDebugFlags.cpp**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**// Debug flags aren't necessarily global:**

**bool debug = false;**

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

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

**if(string(argv[i]) == "--debug=on")**

**debug = true;**

**bool go = true;**

**while(go) {**

**if(debug) {**

**// Debugging code here**

**cout << "Debugger is now on!" << endl;**

**} else {**

**cout << "Debugger is now off." << endl;**

**}**

**cout << "Turn debugger [on/off/quit]: ";**

**string reply;**

**cin >> reply;**

**if(reply == "on") debug = true; // Turn it on**

**if(reply == "off") debug = false; // Off**

**if(reply == "quit") break; // Out of 'while'**

**}**

**} ///:~**

***Listing 3-62. Stringized Expressions***

**//: C03:StringizingExpressions.cpp**

**#include <iostream>**

**using namespace std;**

**#define P(A) cout << #A << ": " << (A) << endl;**

**int main() {**

**int a = 1, b = 2, c = 3;**

**P(a); P(b); P(c);**

**P(a + b);**

**P((c - a)/b);**

**} ///:~**

***Listing 3-63. Using Assert***

**//: C03:Assert.cpp**

**// Use of the assert() debugging macro**

**#include <cassert> // Contains the macro**

**using namespace std;**

**int main() {**

**int i = 100;**

**assert(i != 100); // Fails**

**} ///:~**

***Listing 3-64. Complicated Definitions***

**//: C03:ComplicatedDefinitions.cpp**

**/\* 1. \*/ void \* (\*(\*fp1)(int))[10];**

**/\* 2. \*/ float (\*(\*fp2)(int,int,float))(int);**

**/\* 3. \*/ typedef double (\*(\*(\*fp3)())[10])();**

**fp3 a;**

**/\* 4. \*/ int (\*(\*f4())[10])();**

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

***Listing 3-65. Pointer to Function***

**//: C03:PointerToFunction.cpp**

**// Defining and using a pointer to a function**

**#include <iostream>**

**using namespace std;**

**void func() {**

**cout << "func() called..." << endl;**

**}**

**int main() {**

**void (\*fp)(); // Define a function pointer**

**fp = func; // Initialize it**

**(\*fp)(); // Dereferencing calls the function**

**void (\*fp2)() = func; // Define and initialize**

**(\*fp2)();**

**} ///:~**

***Listing 3-66. Using an Array of Pointers to Functions***

**//: C03:FunctionTable.cpp**

**// Using an array of pointers to functions**

**#include <iostream>**

**using namespace std;**

**// A macro to define dummy functions:**

**#define DF(N) void N() { \**

**cout << "function " #N " called..." << endl; }**

**DF(a); DF(b); DF(c); DF(d); DF(e); DF(f); DF(g);**

**void (\*func\_table[])() = { a, b, c, d, e, f, g };**

**int main() {**

**while(1) {**

**cout << "press a key from 'a' to 'g' "**

**"or q to quit" << endl;**

**char c, cr;**

**cin.get(c); cin.get(cr); // second one for CR**

**if ( c == 'q' )**

**break; // ... out of while(1)**

**if ( c < 'a' || c > 'g' )**

**continue;**

**(\*func\_table[c - 'a'])();**

**}**

**} ///:~*Listing 3-67. Example makefile***

**CPP = g++**

**OFLAG = -o**

**.SUFFIXES : .o .cpp .c**

**.cpp.o :**

**$(CPP) $(CPPFLAGS) -c $<**

**.c.o :**

**$(CPP) $(CPPFLAGS) -c $<**

**all: \**

**Return \**

**Declare \**

**Ifthen \**

**Guess \**

**Guess2**

**# Rest of the files for this chapter not shown**

**Return: Return.o**

**$(CPP) $(OFLAG)Return Return.o**

**Declare: Declare.o**

**$(CPP) $(OFLAG)Declare Declare.o**

**Ifthen: Ifthen.o**

**$(CPP) $(OFLAG)Ifthen Ifthen.o**

**Guess: Guess.o**

**$(CPP) $(OFLAG)Guess Guess.o**

**Guess2: Guess2.o**

**$(CPP) $(OFLAG)Guess2 Guess2.o**

**Return.o: Return.cpp**

**Declare.o: Declare.cpp**

**Ifthen.o: Ifthen.cpp**

**Guess.o: Guess.cpp**

**Guess2.o: Guess2.cpp**

***Listing 3-68. The Header File <require.h>***

***///: :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);***

***}***

***9***

***}***

***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(int argc, int minArgs,***

***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 ///:~***