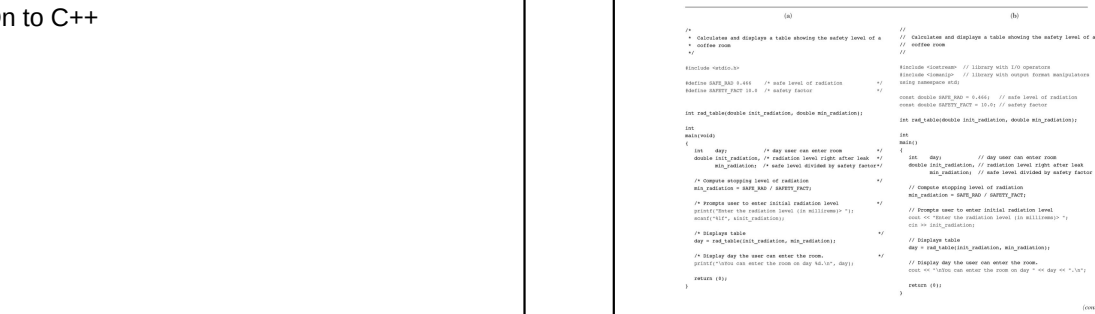
Chapter 15

On to C++

Figure 15.1 Comparison of (a) C and (b) C++ Control Structures



(a)

```
/*
 *  Calculates and displays a table showing the safety level of a
 *  coffee room
 */

#include <stdio.h>

#define SAFE_RAD 9.466    /* safe level of radiation          */
#define SAFETY_FACT 16.8  /* safety factor                    */


int rad_table(double init_radiation, double min_radiation);

int
main(void)
{
    int    day;            /* day user can enter room          */
    double init_radiation, /* radiation level right after leak */
           min_radiation;  /* safe level divided by safety factor*/

    /* Compute stopping level of radiation                     */
    min_radiation = SAFE_RAD / SAFETY_FACT;

    /* Prompts user to enter initial radiation level           */
    printf("Enter the radiation level (in millirems)> ");
    scanf("%lf", &init_radiation);

    /* Displays table                                          */
    day = rad_table(init_radiation, min_radiation);

    /* Display day the user can enter the room.                */
    printf("\nYou can enter the room on day %d.\n", day);

    return (0);
}
```

(b)

```
//
//  Calculates and displays a table showing the safety level of a
//  coffee room
//

#include <iostream>   // library with I/O operators
#include <iomanip>    // library with output format manipulators
using namespace std;

const double SAFE_RAD = 0.466;  // safe level of radiation
const double SAFETY_FACT = 16.0; // safety factor

int rad_table(double init_radiation, double min_radiation);

int
main()
{
    int    day;             // day user can enter room
    double init_radiation,  // radiation level right after leak
           min_radiation;   // safe level divided by safety factor

    // Compute stopping level of radiation
    min_radiation = SAFE_RAD / SAFETY_FACT;

    // Prompts user to enter initial radiation level
    cout << "Enter the radiation level (in millirems)> ";
    cin >> init_radiation;

    // Displays table
    day = rad_table(init_radiation, min_radiation);

    // Display day the user can enter the room.
    cout << "\nYou can enter the room on day " << day << ".\n";

    return (0);
}
```

(continued)

Figure 15.1 Comparison of (a) C and (b) C++ Control Structures (cont'd)

(a)

```
/*
 * Displays a table showing the radiation level and safety status
 * every 3 days until the room is deemed safe to enter.  Returns the
 * day number for the first safe day.
 *   Pre:  min_radiation and init_radiation are defined.
 *   Post: radiation_lev <= min_radiation
 */
int
rad_table(double init_radiation, double min_radiation)
{
    int    day;             /* days elapsed since substance leak    */
    double radiation_lev;   /* current radiation level              */


    day = 0;
    printf("\n  Day   Radiation   Status\n        (millirems)\n");
    for (radiation_lev = init_radiation;
         radiation_lev > min_radiation;
         radiation_lev /= 2.0) {
      if (radiation_lev > SAFE_RAD)
          printf("  %3d%3c%9.4f    Unsafe\n", day, ' ',
                 radiation_lev);
      else
          printf("  %3d%3c%9.4f    Safe\n", day, ' ', radiation_lev);
      day += 3;
    }

    return (day);
}
```

(b)

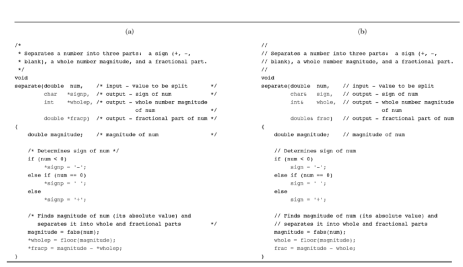
```
//
// Displays a table showing the radiation level and safety status every
// 3 days until the room is deemed safe to enter.  Returns the day
// number for the first safe day.
//   Pre:  min_radiation and init_radiation are defined.
//   Post: radiation_lev <= min_radiation
//
int
rad_table(double init_radiation, double min_radiation)
{

    int    day;             // days elapsed since substance leak
    double radiation_lev;   // current radiation level


    day = 0;
    cout << "\n  Day   Radiation   Status\n        (millirems)\n";
    for (radiation_lev = init_radiation;
         radiation_lev > min_radiation;
         radiation_lev /= 2.0) {
      if (radiation_lev > SAFE_RAD)
        cout << "  " << setw(3) << day << setw(3) << ' ' <<
                fixed << showpoint << setprecision(4)
             << setw(9) << radiation_lev << "   Unsafe\n";
      else
        cout << "  " << setw(3) << day << setw(3) << ' ' <<
            fixed << showpoint << setprecision(4)
            << setw(9) << radiation_lev << "   Safe\n";
      day += 3;
    }

    return (day);
}
```


Figure 15.2 Implementing Output Parameters in C and C++



(a)

```
/*
 * Separates a number into three parts:  a sign (+, -,
 * blank), a whole number magnitude, and a fractional part.
 */
void
separate(double  num,    /* input - value to be split       */
         char    *sign,  /* output - sign of num            */
         int     *whole, /* output - whole number magnitude
                            of num                          */
         double *frac)   /* output - fractional part of num */
{
    double magnitude;    /* magnitude of num                */

    /* Determines sign of num */
    if (num < 0)
        *sign = '-';
    else if (num == 0)
        *sign = ' ';
    else
        *sign = '+';

    /* Finds magnitude of num (its absolute value) and
       separates it into whole and fractional parts        */
    magnitude = fabs(num);
    *whole = floor(magnitude);
    *frac = magnitude - *whole;
}
```

(b)

```
//
// Separates a number into three parts:  a sign (+, -,
// blank), a whole number magnitude, and a fractional part.
//
void
separate(double  num,    // input - value to be split
         char&   sign,   // output - sign of num
         int&    whole,  // output - whole number magnitude
                            of num
         double& frac)   // output - fractional part of num
{
    double magnitude;    // magnitude of num

    // Determines sign of num
    if (num < 0)
        sign = '-';
    else if (num == 0)
        sign = ' ';
    else
        sign = '+';

    // Finds magnitude of num (its absolute value) and
    // separates it into whole and fractional parts
    magnitude = fabs(num);
    whole = floor(magnitude);
    frac = magnitude - whole;
}
```


Figure 15.3 "Donut" Model of an Abstract Data Type

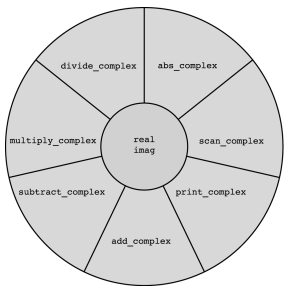


Figure 15.4 "Donut" Model of Standard Type int

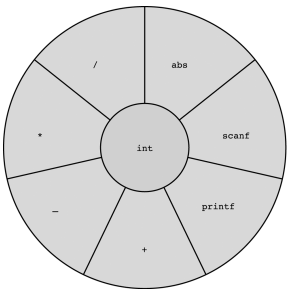


Figure 15.5 Comparison of Models of Standard Type int and Abstract Data Type Complex

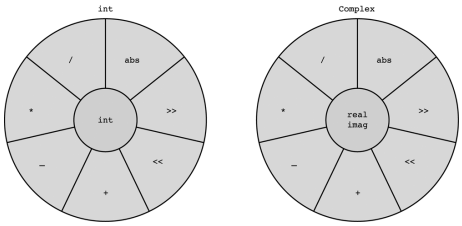


Figure 15.6 Header File for Class Complex

```
//
// header file complex.h
//
#ifndef COMPLEX_H
#define COMPLEX_H
#include <iostream>
using namespace std;

class Complex {
public:
   Complex() { real = 0; imag = 0; }  // default constructor

   Complex(double r1) { real = r1; imag = 0; }   // constructor that
                                                 // converts reals to complex numbers
   Complex(double r1, double im) { real = r1; imag = im; } // constructor
                                 // with 2 parameters corresponding to 2 data members

   Complex abs() const;

   Complex operator+ (Complex operand2) const;
   Complex operator- (Complex operand2) const;
   Complex operator* (Complex operand2) const;
   Complex operator/ (Complex operand2) const;

private:
   double real;  // real and imaginary parts
   double imag;  //    of a complex number

   friend istream& operator>> (istream& is, Complex& inum);
   friend ostream& operator<< (ostream& os, Complex& onum);
};

#endif
```


Figure 15.7 Implementation File for Class Complex

```
//
//  implementation file complex.cpp
//

#include "complex"
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

//
// absolute value of a complex number
//
Complex Complex::abs() const
{
   Complex cabs( sqrt(real * real + imag * imag), 0);
   return cabs;
}

//
// sum of current complex number and operand2
//
Complex Complex::operator+ ( Complex operand2 ) const
{
   Complex csum( real + operand2.real, imag + operand2.imag );
   return csum;
}
```

(continued)

Figure 15.7 Implementation File for Class Complex (cont'd)

```
//
// product of current complex number and operand2
//
Complex Complex::operator* ( Complex operand2 ) const
{
   Complex cproduct( real * operand2.real - imag * operand2.imag,
                     real * operand2.imag + imag * operand2.real );
   return cproduct;
}

//
// difference of current complex number and operand2
//
Complex Complex::operator- ( Complex operand2 ) const
{
   Complex cdiff( real - operand2.real, imag - operand2.imag );
   return cdiff;
}

//
// quotient of current complex number divided by operand2
//
Complex Complex::operator/ ( Complex operand2 ) const
{
   double divisor = operand2.real * operand2.real +
                    operand2.imag * operand2.imag;
   Complex cquot( (real * operand2.real + imag * operand2.imag) / divisor,
                  (imag * operand2.real - real * operand2.imag) /
                     divisor);
   return cquot;
}

//
// Extract from input source the two components of a complex number
//
istream& operator>> (istream& is, Complex& c)
{
   is >> c.real >> c.imag;
   return is;
}
```

(continued)

Figure 15.7 Implementation File for Class Complex (cont'd)

```
//
// Insert in the output stream a representation of a complex number:
//    either the form (a + bi) or (a - bi), dropping a or b if one of them
//    rounds to zero
//

ostream& operator<< (ostream& os, Complex c)
{
   double a = c.real;
   double b = c.imag;
   char sign;

   os << fixed << showpoint << setprecision(2);
   os << '(';
   if (fabs(a) < .005  && fabs(b) < .005) {
      os << 0.0;
   } else if (fabs(b) < .005) {
      os << a;
   } else if (fabs(a) < .005) {
      os << b;
   } else {
      if (b < 0)
         sign = '-';
      else
         sign = '+';
      os << a << ' ' << sign << ' ' << fabs(b) << 'i';
   }

   os << ')';
   return os;
}
```


Figure 15.8 Driver Function to Test Class Complex

```
//
// Driver for Complex – equivalent to driver of Fig. 11.10
//

#include "complex.h"


int
main()
{
   Complex com1, com2;

   // Gets two complex numbers
   cout << "Enter the real and imaginary parts of a complex number\n";
   cout << "separated by a space> ";
   cin >> com1;
   cout << "Enter a second complex number> ";
   cin >> com2;

   // Forms and displays the sum
   cout << "\n" << com1 << "  +  " << com2 << "  =  " << (com1 + com2);

   // Forms and displays the difference
   cout << "\n\n" << com1 << "  -  " << com2 << "  =  " << (com1 - com2);

   // Forms and displays the absolute value of the first number
   cout << "\n\n|" << com1 << "| = " << com1.abs() << "\n";

   return (0);
}
```

Figure 15.9 Step-by-Step Evaluation of Multiple << Operations

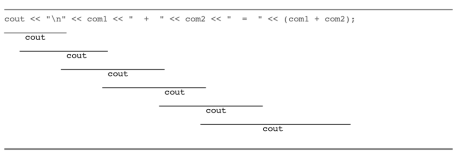


Figure 15.10 Declaration of Class Ratio