Appendix D

Answers

Day 1

Quiz

1. What is the difference between interpreters and compilers?

Interpreters read through source code and translate a program, turning the programmer's code, or program instructions, directly into actions. Compilers translate source code into an executable program that can be run at a later time.

2. How do you compile the source code with your compiler?

Every compiler is different. Be sure to check the documentation that came with your compiler.

3. What does the linker do?

The linker's job is to tie together your compiled code with the libraries supplied by your compiler vendor and other sources. The linker lets you build your program in pieces and then link together the pieces into one big program.

4. What are the steps in the development cycle?

Edit source code, compile, link, test, repeat.

Exercises

1. Initializes two integer variables and then prints out their sum and their product.

2. See your compiler manual.

3. You must put a `#` symbol before the word `include` on the first line.

4. This program prints the words `Hello World` to the screen, followed by a new line (carriage return).

Day 2

Quiz

1. What is the difference between the compiler and the preprocessor?

Each time you run your compiler, the preprocessor runs first. It reads through your source code and includes the files you've asked for, and performs other housekeeping chores. The preprocessor is discussed in detail on Day 18, "Object-Oriented Analysis and Design."

2. Why is the function `main()` special?

`main()` is called automatically, each time your program is executed.

3. What are the two types of comments, and how do they differ?

C++-style comments are two slashes (`//`), and they comment out any text until the end of the line. C-style comments come in pairs (`/* */`), and everything between the matching pairs is commented out. You must be careful to ensure you have matched pairs.

4. Can comments be nested?

Yes, C++-style comments can be nested within C-style comments. You can, in fact, nest C-style comments within C++-style comments, as long as you remember that the C++-style comments end at the end of the line.

5. Can comments be longer than one line?

C-style comments can. If you want to extend C++-style comments to a second line, you must put another set of double slashes (`//`).

Exercises

1. Write a program that writes `I love C++` to the screen.

```
1: #include <iostream.h>
2:
3: int main()
4: {
5:    cout << "I love C++\n";
6:    return 0;
7: }
```

2. Write the smallest program that can be compiled, linked, and run.

```
int main(){}
```

3. BUG BUSTERS: Enter this program and compile it. Why does it fail? How can you fix it?

```
1: #include <iostream.h>
2: main()
3: {
4:      cout << Is there a bug here?";
5: }
```

Line 4 is missing an opening quote for the string.

4. Fix the bug in Exercise 3 and recompile, link, and run it.

```
1: #include <iostream.h>
2: main()
3: {
4:      cout << "Is there a bug here?";
5: }
```

Day 3

Quiz

1. What is the difference between an integral variable and a floating-point variable?

Integer variables are whole numbers; floating-point variables are "reals" and have a "floating" decimal point. Floating-point numbers can be represented using a mantissa and an exponent.

2. What are the differences between an `unsigned short int` and a `long int`?

The keyword `unsigned` means that the integer will hold only positive numbers. On most computers, short integers are 2 bytes and long integers are 4.

3. What are the advantages of using a symbolic constant rather than a literal?

A symbolic constant explains itself; the name of the constant tells what it is for. Also, symbolic constants can be redefined at one location in the source code, rather than the programmer having to edit the code everywhere the literal is used.

4. What are the advantages of using the `const` keyword rather than `#define`?

`const` variables are "typed;" thus the compiler can check for errors in how they are used. Also, they survive the preprocessor; thus the name is available in the debugger.

5. What makes for a good or bad variable name?

A good variable name tells you what the variable is for; a bad variable name has no information. `myAge` and `PeopleOnTheBus` are good variable names, but `xjk` and `prndl` are probably less useful.

6. Given this `enum`, what is the value of `Blue`?

```
enum COLOR { WHITE, BLACK = 100, RED, BLUE, GREEN = 300 };
BLUE = 102
```

7. Which of the following variable names are good, which are bad, and which are invalid?

a. `Age`
Good

b. `!ex`
Not legal

c. `R79J`
Legal, but a bad choice

d. `TotalIncome`
Good

e. `__Invalid`
Legal, but a bad choice

Exercises

1. What would be the correct variable type in which to store the following information?

a. Your age.
`Unsigned short` integer

b. The area of your backyard.
`Unsigned long` integer or `unsigned float`

c. The number of stars in the galaxy.
`Unsigned double`

d. The average rainfall for the month of January.
`Unsigned short` integer

2. Create good variable names for this information.

a. `myAge`

b. `backYardArea`

c. `StarsInGalaxy`

d. `averageRainFall`

3. Declare a constant for pi as 3.14159.

```
const float PI = 3.14159;
```

4. Declare a `float` variable and initialize it using your pi constant.

```
float myPi = PI;
```

Day 4

Quiz

1. What is an expression?

Any statement that returns a value.

2. Is `x = 5 + 7` an expression? What is its value?

Yes. `12`

3. What is the value of `201 / 4`?

`50`

4. What is the value of `201 % 4`?

`1`

5. If `myAge`, `a`, and `b` are all `int` variables, what are their values after:

```
myAge = 39;
a = myAge++;

b = ++myAge;
myAge: 41, a: 39, b: 41
```

6. What is the value of `8+2*3`?

```
14
```

7. What is the difference between `if(x = 3)` and `if(x == 3)`?

The first one assigns `3` to `x` and returns true. The second one tests whether `x` is equal to `3`; it returns true if the value of `x` is equal to 3 and false if it is not.

8. Do the following values evaluate to `TRUE` or `FALSE`?

a. 0
`FALSE`

b. `1`
`TRUE`

c. `-1`
`TRUE`

d. `x = 0`
`FALSE`

e. `x == 0 // assume that x has the value of 0`
`TRUE`

Exercises

1. Write a single `if` statement that examines two integer variables and changes the larger to the smaller, using only one `else` clause.

```
if (x > y)
    x = y;
else           // y > x || y == x
    y = x;
```

2. Examine the following program. Imagine entering three numbers, and write what output you expect.

```
1:   #include <iostream.h>
```

```
2:   int main()
3:   {
4:        int a, b, c;
5:        cout << "Please enter three numbers\n";
6:        cout << "a: ";
7:        cin >> a;
8:        cout << "\nb: ";
9:        cin >> b;
10:       cout << "\nc: ";
11:       cin >> c;
12:
13:       if (c = (a-b))
14        {
15:            cout << "a: " << a << " minus b: ";
16:            cout << b << " _equals c: " << c;
17:       }
15:       else
16:            cout << "a-b does not equal c: ";
17:     return 0;
18:  }
```

3. Enter the program from Exercise 2; compile, link, and run it. Enter the numbers `20`, `10`, and `50`. Did you get the output you expected? Why not?

Enter `20`, `10`, `50`.

Get back `a`: `20` `b: 30` `c: 10`.

Line 13 is assigning, not testing for equality.

4. Examine this program and anticipate the output:

```
1:     #include <iostream.h>
2:      int main()
3:      {
4:           int a = 2, b = 2, c;
5:           if (c = (a-b))
6:                cout << "The value of c is: " << c;
7:     return 0;


8:      }
```

5. Enter, compile, link, and run the program from Exercise 4. What was the output? Why?

Because line 5 is assigning the value of `a-b` to `c`, the value of the assignment is `a` (1) minus `b` (1), or `0`. Because `0` is evaluated as `FALSE`, the `if` fails and nothing is printed.

Day 5

Quiz

1. What are the differences between the function prototype and the function defi-nition?

The function prototype declares the function; the definition defines it. The prototype ends with a semicolon; the definition need not. The declaration can include the keyword `inline` and default values for the parameters; the definition cannot. The declaration need not include names for the parameters; the definition must.

2. Do the names of parameters have to agree in the prototype, definition, and call to the function?

No. All parameters are identified by position, not name.

3. If a function doesn't return a value, how do you declare the function?

Declare the function to return `void`.

4. If you don't declare a return value, what type of return value is assumed?

Any function that does not explicitly declare a return type returns `int`.

5. What is a local variable?

A local variable is a variable passed into or declared within a block, typically a function. It is visible only within the block.

6. What is scope?

Scope refers to the visibility and lifetime of local and global variables. Scope is usually established by a set of braces.

7. What is recursion?

Recursion generally refers to the ability of a function to call itself.

8. When should you use global variables?

Global variables are typically used when many functions need access to the same data. Global variables are very rare in C++; once you know how to create static class variables, you will almost never create global variables.

9. What is function overloading?

Function overloading is the ability to write more than one function with the same name, distinguished by the number or type of the parameters.

10. What is polymorphism?

Polymorphism is the ability to treat many objects of differing but related types without regard to their differences. In C++, polymorphism is accomplished by using class derivation and virtual functions.

Exercises

1. Write the prototype for a function named `Perimeter`, which returns an `unsigned long int` and which takes two parameters, both `unsigned short ints`.
`unsigned long int Perimeter(unsigned short int, unsigned short int);`

2. Write the definition of the function `Perimeter` as described in Exercise 1. The two parameters represent the length and width of a rectangle and have the function return the perimeter (twice the length plus twice the width).

```
unsigned long int Perimeter(unsigned short int length, unsigned short int width)
```

```
{
  return 2*length + 2*width;
}
```

3. BUG BUSTERS: What is wrong with the function?

```
#include <iostream.h>
void myFunc(unsigned short int x);
int main()
{
    unsigned short int x, y;
    y = myFunc(int);
    cout << "x: " << x << " y: " << y << "\n";
return 0;
}

void myFunc(unsigned short int x)
{
    return (4*x);
}
```

The function is declared to return `void` and it cannot return a value.

4. BUG BUSTERS: What is wrong with the function?

```
#include <iostream.h>
int myFunc(unsigned short int x);
int main()
{
    unsigned short int x, y;
    y = myFunc(int);
    cout << "x: " << x << " y: " << y << "\n";
return 0;
}

int myFunc(unsigned short int x)
{
    return (4*x);
}
```

This function would be fine, but there is a semicolon at the end of the function definition's header.

5. Write a function that takes two `unsigned short int` arguments and returns the result of dividing the first by the second. Do not do the division if the second number is `0`, but do return `-1`.
`short int Divider`(`unsigned short int valOne`, `unsigned short int valTwo`)

```
{
    if (valTwo == 0)
          return -1;
    else
          return valOne / valTwo;
}
```

6. Write a program that asks the user for two numbers and calls the function you wrote in Exercise 5. Print the answer, or print an error message if you get `-1`.

```
#include <iostream.h>
typedef unsigned short int USHORT;
typedef unsigned long int ULONG;
short int Divider(
unsigned short int valone,
unsigned short int valtwo);
int main()
```

```
{
    USHORT one, two;
    short int answer;
    cout << "Enter two numbers.\n Number one: ";
    cin >> one;
    cout << "Number two: ";
    cin >> two;
    answer = Divider(one, two);
    if (answer > -1)
       cout << "Answer: " << answer;
    else
       cout << "Error, can't divide by zero!";
return 0;
}
```

7. Write a program that asks for a number and a power. Write a recursive function that takes the number to the power. Thus, if the number is 2 and the power is 4, the function will return `16`.

```
#include <iostream.h>
typedef unsigned short USHORT;
typedef unsigned long ULONG;
ULONG GetPower(USHORT n, USHORT power);
int main()
{
   USHORT number, power;
   ULONG answer;
   cout << "Enter a number: ";
   cin >> number;
   cout << "To what power? ";
   cin >> power;
   answer = GetPower(number,power);
   cout << number << " to the " << power << "th power is " <<
answer << endl;
return 0;
}

ULONG GetPower(USHORT n, USHORT power)
{
    if(power == 1)
     return n;
    else
       return (n * GetPower(n,power-1));
}
```

Day 6

Quiz

1. What is the dot operator, and what is it used for?

The dot operator is the period (`.`). It is used to access the members of the class.

2. Which sets aside memory--declaration or definition?

Definitions of variables set aside memory. Declarations of classes don't set aside memory.

3. Is the declaration of a class its interface or its implementation?

The declaration of a class is its interface; it tells clients of the class how to interact with the class. The implementation of the class is the set of member functions stored--usually in a related CPP file.

4. What is the difference between public and private data members?

Public data members can be accessed by clients of the class. Private data members can be accessed only by member functions of the class.

5. Can member functions be private?

Yes. Both member functions and member data can be private.

6. Can member data be public?

Although member data can be public, it is good programming practice to make it private and to provide public accessor functions to the data.

7. If you declare two `Cat` objects, can they have different values in their `itsAge` member data?

Yes. Each object of a class has its own data members.

8. Do class declarations end with a semicolon? Do class method definitions?

Declarations end with a semicolon after the closing brace; function definitions do not.

9. What would the header for a `Cat` function, `Meow`, that takes no parameters and returns `void` look like?

The header for a `Cat` function, `Meow()`, that takes no parameters and returns `void` looks like this:

```
void Cat::Meow()
```

10. What function is called to initialize a class?

The constructor is called to initialize a class.

Exercises

1. Write the code that declares a class called `Employee` with these data members: `age`, `yearsOfService`, and `Salary`.

```
class Employee
{
    int Age;
    int YearsOfService;
    int Salary;
};
```

2. Rewrite the `Employee` class to make the data members private, and provide public accessor methods to get and set each of the data members.

```
class Employee
{
public:
    int GetAge() const;
    void SetAge(int age);
    int GetYearsOfService()const;
    void SetYearsOfService(int years);
    int GetSalary()const;
    void SetSalary(int salary);

private:
    int Age;
    int YearsOfService;
    int Salary;
};
```

3. Write a program with the `Employee` class that makes two `Employees`; sets their `age`, `YearsOfService`, and `Salary`; and prints their values.

```
main()
{
    Employee John;
    Employee Sally;
    John.SetAge(30);
    John.SetYearsOfService(5);
    John.SetSalary(50000);

    Sally.SetAge(32);
    Sally.SetYearsOfService(8);
    Sally.SetSalary(40000);

    cout << "At AcmeSexist company, John and Sally have the same
job.\n";
    cout << "John is " << John.GetAge() << " years old and he has
been with";
    cout << "the firm for " << John.GetYearsOfService << "
years.\n";
    cout << "John earns $" << John.GetSalary << " dollars per
year.\n\n";
    cout << "Sally, on the other hand is " << Sally.GetAge() << "
years old and has";
    cout << "been with the company " << Sally.GetYearsOfService;
    cout << " years. Yet Sally only makes $" << Sally.GetSalary();
    cout << " dollars per year!  Something here is unfair.";
```

4. Continuing from Exercise 3, provide a method of `Employee` that reports how many thousands of dollars the employee earns, rounded to the nearest 1,000.

```
float Employee:GetRoundedThousands()const
{
    return Salary / 1000;
}
```

5. Change the `Employee` class so that you can initialize `age`, `YearsOfService`, and `Salary` when you create the employee.

```
class Employee
{
public:

    Employee(int age, int yearsOfService, int salary);
    int GetAge()const;
    void SetAge(int age);
    int GetYearsOfService()const;
    void SetYearsOfService(int years);
    int GetSalary()const;
    void SetSalary(int salary);

private:
    int Age;
    int YearsOfService;
    int Salary;
};
```

6. BUG BUSTERS: What is wrong with the following declaration?

```
class Square
{
public:
    int Side;
}
```

Class declarations must end with a semicolon.

7. BUG BUSTERS: Why isn't the following class declaration very useful?

```
class Cat
{
    int GetAge()const;
private:
    int itsAge;
};
```

The accessor `GetAge()` is private. Remember: All class members are private unless you say otherwise.

8. BUG BUSTERS: What three bugs in this code will the compiler find?

```
class  TV
{
public:
    void SetStation(int Station);
    int GetStation() const;
private:
    int itsStation;
};

main()
{
    TV myTV;
    myTV.itsStation = 9;
    TV.SetStation(10);
    TV myOtherTv(2);
}
```

You can't access `itsStation` directly. It is private.
You can't call `SetStation()` on the class. You can call `SetStation()` only on objects.
You can't initialize `itsStation` because there is no matching constructor.

Day 7

Quiz

1. How do I initialize more than one variable in a `for` loop?

Separate the initializations with commas, such as

```
for (x = 0, y = 10; x < 100; x++, y++)
```

2. Why is `goto` avoided?

`goto` jumps in any direction to any arbitrary line of code. This makes for source code that is difficult to understand and therefore difficult to maintain.

3. Is it possible to write a `for` loop with a body that is never executed?

Yes, if the condition is `FALSE` after the initialization, the body of the `for` loop will never execute. Here's an example:

```
for (int x = 100; x < 100; x++)
```

4. Is it possible to nest `while` loops within `for` loops?

Yes. Any loop can be nested within any other loop.

5. Is it possible to create a loop that never ends? Give an example.

Yes. Following are examples for both a `for` loop and a `while` loop:

```
for(;;)
{
    // This for loop never ends!
}
while(1)
{
    // This while loop never ends!


}
```

6. What happens if you create a loop that never ends?

Your program hangs, and you usually must reboot the computer.

Exercises

1. What is the value of `x` when the `for` loop completes?

```
for (int x = 0; x < 100; x++)
100
```

2. Write a nested `for` loop that prints a 10x10 pattern of `0`s.

```
for (int i = 0; i< 10; i++)
{
    for ( int j = 0; j< 10; j++)
       cout << "0";
    cout << "\n";
}
```

3. Write a `for` statement to count from 100 to 200 by 2s.

```
for (int x = 100; x<=200; x+=2)
```

4. Write a `while` loop to count from 100 to 200 by 2s.

```
int x = 100;
while (x <= 200)
    x+= 2;
```

5. Write a `do...while` loop to count from 100 to 200 by 2s.

```
int x = 100;
```

```
do
{
    x+=2;
} while (x <= 200);
```

6. BUG BUSTERS: What is wrong with this code?

```
int counter = 0
while (counter < 10)
{
    cout << "counter: " << counter;
    counter++;
}
```

`counter` is never incremented and the `while` loop will never terminate.

7. BUG BUSTERS: What is wrong with this code?

```
for (int counter = 0; counter < 10; counter++);
    cout << counter << "\n";
```

There is a semicolon after the loop, and the loop does nothing. The programmer may have intended this, but if `counter` was supposed to print each value, it won't.

8. BUG BUSTERS: What is wrong with this code?

```
int counter = 100;
while (counter < 10)
{
    cout << "counter now: " << counter;
    counter--;
}
```

`counter` is initialized to `100`, but the test condition is that if it is less than 10, the test will fail and the body will never be executed. If line 1 were changed to `int counter = 5;`, the loop would not terminate until it had counted down past the smallest possible `int`. Because `int` is `signed` by default, this would not be what was intended.

9. BUG BUSTERS: What is wrong with this code?

```
cout << "Enter a number between 0 and 5: ";
cin >> theNumber;
switch (theNumber)
{
   case 0:
         doZero();
   case 1:              // fall through
   case 2:              // fall through
   case 3:              // fall through
   case 4:              // fall through
   case 5:
         doOneToFive();
         break;
   default:
         doDefault();
         break;
}
```

`Case 0` probably needs a `break` statement. If it does not, it should be documented with a comment.

Day 8

Quiz

1. What operator is used to determine the address of a variable?

The address of operator (`&`) is used to determine the address of any variable.

2. What operator is used to find the value stored at an address held in a pointer?

The dereference operator (`*`) is used to access the value at an address in a pointer.

3. What is a pointer?

A pointer is a variable that holds the address of another variable.

4. What is the difference between the address stored in a pointer and the value at that address?

The address stored in the pointer is the address of another variable. The value stored at that address is any value stored in any variable. The indirection operator (`*`) returns the value stored at the address, which itself is stored in the pointer.

5. What is the difference between the indirection operator and the address of oper-ator?

The indirection operator returns the value at the address stored in a pointer. The address of operator (`&`) returns the memory address of the variable.

6. What is the difference between `const int * ptrOne` and `int * const ptrTwo`?

The `const int * ptrOne` declares that `ptrOne` is a pointer to a constant integer. The integer itself cannot be changed using this pointer.
The `int * const ptrTwo` declares that `ptrTwo` is a constant pointer to an integer. Once it is initialized, this pointer cannot be reassigned.

Exercises

1. What do these declarations do?

a. `int * pOne;`
b. `int vTwo;`
c. `int * pThree = &vTwo;`

a. `int * pOne;` declares a pointer to an integer.
b. `int vTwo;` declares an integer variable.
c. `int * pThree = &vTwo;` declares a pointer to an integer and initializes it with the address of another variable.

2. If you have an `unsigned short` variable named `yourAge`, how would you declare a pointer to manipulate `yourAge`?

```
unsigned short *pAge = &yourAge;
```

3. Assign the value `50` to the variable `yourAge` by using the pointer that you declared in Exercise 2.

```
*pAge = 50;
```

4. Write a small program that declares an integer and a pointer to integer. Assign the address of the integer to the pointer. Use the pointer to set a value in the integer variable.

```
int theInteger;
int *pInteger = &theInteger;
*pInteger = 5;
```

5. BUG BUSTERS: What is wrong with this code?

```
#include <iostream.h>
int main()
{
     int *pInt;
     *pInt = 9;
     cout << "The value at pInt: " << *pInt;
return 0;
}
```

`pInt` should have been initialized. More importantly, because it was not initialized and was not assigned the address of any memory, it points to a random place in memory. Assigning `9` to that random place is a dangerous bug.

6. BUG BUSTERS: What is wrong with this code?

```
int main()
{
    int SomeVariable = 5;
    cout << "SomeVariable: " << SomeVariable << "\n";
    int *pVar = & SomeVariable;
    pVar = 9;
    cout << "SomeVariable: " << *pVar << "\n";
return 0;
}
```

Presumably, the programmer meant to assign `9` to the value at `pVar`. Unfortunately, `9` was assigned to be the value of `pVar` because the indirection operator (`*`) was left off. This will lead to disaster if `pVar` is used to assign a value.

Day 9

Quiz

1. What is the difference between a reference and a pointer?

A reference is an alias, and a pointer is a variable that holds an address. References cannot be null and cannot be assigned to.

2. When must you use a pointer rather than a reference?

When you may need to reassign what is pointed to, or when the pointer may be null.

3. What does `new` return if there is insufficient memory to make your `new` object?

A null pointer (`0`).

4. What is a constant reference?

This is a shorthand way of saying "a reference to a constant object."

5. What is the difference between passing by reference and passing a reference?

Passing by reference means not making a local copy. It can be accomplished by passing a reference or by passing a pointer.

Exercises

1. Write a program that declares an `int`, a reference to an `int`, and a pointer to an `int`. Use the pointer and the reference to manipulate the value in the `int`.

```
int main()
{
int varOne;
int& rVar = varOne;
int* pVar = &varOne;
rVar = 5;
*pVar = 7;
return 0;
}
```

2. Write a program that declares a constant pointer to a constant integer. Initialize the pointer to an integer variable, `varOne`. Assign `6` to `varOne`. Use the pointer to assign `7` to `varOne`. Create a second integer variable, `varTwo`. Reassign the pointer to `varTwo`.

```
int main()
{
   int varOne;
   const int * const pVar = &varOne;
   *pVar = 7;
   int varTwo;
   pVar = &varTwo;
return 0;
}
```

3. Compile the program in Exercise 2. What produces errors? What produces warnings?
You can't assign a value to a constant object, and you can't reassign a constant pointer.

4. Write a program that produces a stray pointer.

```
int main()
{
int * pVar;
*pVar = 9;
return 0;
}
```

5. Fix the program from Exercise 4.

```
int main()
{
int VarOne;
int * pVar = &varOne;
*pVar = 9;
return 0;
}
```

6. Write a program that produces a memory leak.

```
int FuncOne();
int main()
{
   int localVar = FunOne();
   cout << "the value of localVar is: " << localVar;
return 0;
}

int FuncOne()
{
   int * pVar = new int (5);
   return *pVar;
}
```

7. Fix the program from Exercise 6.

```
void FuncOne();
int main()
{
   FuncOne();
return 0;
}

void FuncOne()
{
   int * pVar = new int (5);
   cout << "the value of *pVar is: " << *pVar ;
}
```

8. BUG BUSTERS: What is wrong with this program?

```
#include <iostream.h>

class CAT
{
   public:
      CAT(int age) { itsAge = age; }
      ~CAT(){}
      int GetAge() const { return itsAge;}
   private:
      int itsAge;
};

CAT & MakeCat(int age);
int main()
{
   int age = 7;
   CAT Boots = MakeCat(age);
   cout << "Boots is " << Boots.GetAge() << " years old\n";
 return 0;
}

CAT & MakeCat(int age)
{
   CAT * pCat = new CAT(age);
   return *pCat;
}
```

`MakeCat` returns a reference to the `CAT` created on the free store. There is no way to free that memory, and this produces a memory leak.

9. Fix the program from Exercise 8.

```
#include <iostream.h>

class CAT
{
   public:
      CAT(int age) { itsAge = age; }
      ~CAT(){}
      int GetAge() const { return itsAge;}
   private:
      int itsAge;
};

CAT * MakeCat(int age);
int main()
{
   int age = 7;
   CAT * Boots = MakeCat(age);
   cout << "Boots is " << Boots->GetAge() << " years old\n";
   delete Boots;
 return 0;
}

CAT * MakeCat(int age)
{
   return new CAT(age);
}
```

Day 10

Quiz

1. When you overload member functions, in what ways must they differ?

Overloaded member functions are functions in a class that share a name but differ in the number or type of their parameters.

2. What is the difference between a declaration and a definition?

A definition sets aside memory, but a declaration does not. Almost all declarations are definitions; the major exceptions are class declarations, function prototypes, and `typedef` statements.

3. When is the copy constructor called?

Whenever a temporary copy of an object is created. This happens every time an object is passed by value.

4. When is the destructor called?

The destructor is called each time an object is destroyed, either because it goes out of scope or because you call `delete` on a pointer pointing to it.

5. How does the copy constructor differ from the assignment operator (`=`)?

The assignment operator acts on an existing object; the copy constructor creates a new one.

6. What is the `this` pointer?

The `this` pointer is a hidden parameter in every member function that points to the object itself.

7. How do you differentiate between overloading the prefix and postfix increments?

The prefix operator takes no parameters. The postfix operator takes a single `int` parameter, which is used as a signal to the compiler that this is the postfix variant.

8. Can you overload the operator+ for `short` integers?

No, you cannot overload any operator for built-in types.

9. Is it legal in C++ to overload `operator++` so that it decrements a value in your class?

It is legal, but it is a bad idea. Operators should be overloaded in a way that is likely to be readily understood by anyone reading your code.

10. What return value must conversion operators have in their declaration?

None. Like constructors and destructors, they have no return values.

Exercises

1. Write a `SimpleCircle` class declaration (only) with one member variable: `itsRadius`. Include a default constructor, a destructor, and accessor methods for `itsRadius`.

```
class SimpleCircle
{
public:
     SimpleCircle();
     ~SimpleCircle();
     void SetRadius(int);
     int GetRadius();
private:
     int itsRadius;
};
```

2. Using the class you created in Exercise 1, write the implementation of the default constructor, initializing `itsRadius` with the value `5`.

```
SimpleCircle::SimpleCircle():
itsRadius(5)
{}
```

3. Using the same class, add a second constructor that takes a value as its parameter and assigns that value to `itsRadius`.

```
SimpleCircle::SimpleCircle(int radius):
itsRadius(radius)
{}
```

4. Create a prefix and postfix increment operator for your `SimpleCircle` class that increments `itsRadius`.

```
const SimpleCircle& SimpleCircle::operator++()
{
     ++(itsRadius);
     return *this;
}
```

```
// Operator ++(int) postfix.
// Fetch then increment
const SimpleCircle SimpleCircle::operator++ (int)
{
// declare local SimpleCircle and initialize to value of *this
    SimpleCircle temp(*this);
    ++(itsRadius);
    return temp;
}
```

5. Change `SimpleCircle` to store `itsRadius` on the free store, and fix the existing methods.

```
class SimpleCircle
{
public:
     SimpleCircle();
     SimpleCircle(int);
     ~SimpleCircle();
     void SetRadius(int);
     int GetRadius();
     const SimpleCircle& operator++();
     const SimpleCircle operator++(int);
private:
     int *itsRadius;
};


SimpleCircle::SimpleCircle()
{itsRadius = new int(5);}

SimpleCircle::SimpleCircle(int radius)
{itsRadius = new int(radius);}

const SimpleCircle& SimpleCircle::operator++()
{
     ++(itsRadius);
     return *this;
}

// Operator ++(int) postfix.
// Fetch then increment
const SimpleCircle SimpleCircle::operator++ (int)
{
// declare local SimpleCircle and initialize to value of *this
    SimpleCircle temp(*this);
    ++(itsRadius);
    return temp;
}
```

6. Provide a copy constructor for `SimpleCircle`.

```
SimpleCircle::SimpleCircle(const SimpleCircle & rhs)
{
     int val = rhs.GetRadius();
     itsRadius = new int(val);
}
```

7. Provide an `operator=` for `SimpleCircle`.

```
SimpleCircle& SimpleCircle::operator=(const SimpleCircle & rhs)
{
     if (this == &rhs)
          return *this;
     delete itsRadius;
    itsRadius = new int;
    *itsRadius = rhs.GetRadius();
    return *this;
}
```

8. Write a program that creates two `SimpleCircle` objects. Use the default constructor on one and instantiate the other with the value `9`. Call `increment` on each and then print their values. Finally, assign the second to the first and print its values.

```
#include <iostream.h>

class SimpleCircle
{
public:
      // constructors
     SimpleCircle();
     SimpleCircle(int);
     SimpleCircle(const SimpleCircle &);
     ~SimpleCircle() {}

// accessor functions
     void SetRadius(int);
     int GetRadius()const;

// operators
     const SimpleCircle& operator++();
     const SimpleCircle operator++(int);
     SimpleCircle& operator=(const SimpleCircle &);

private:
     int *itsRadius;
};


SimpleCircle::SimpleCircle()
{itsRadius = new int(5);}

SimpleCircle::SimpleCircle(int radius)
{itsRadius = new int(radius);}

SimpleCircle::SimpleCircle(const SimpleCircle & rhs)
{
     int val = rhs.GetRadius();
     itsRadius = new int(val);
}
SimpleCircle& SimpleCircle::operator=(const SimpleCircle & rhs)
{
     if (this == &rhs)
          return *this;
     *itsRadius = rhs.GetRadius();
     return *this;
}

const SimpleCircle& SimpleCircle::operator++()
{
     ++(itsRadius);
     return *this;
}

// Operator ++(int) postfix.
// Fetch then increment
const SimpleCircle SimpleCircle::operator++ (int)
{
// declare local SimpleCircle and initialize to value of *this
    SimpleCircle temp(*this);
    ++(itsRadius);
    return temp;
}
```

```
int SimpleCircle::GetRadius() const
{
     return *itsRadius;
}
int main()
{
     SimpleCircle CircleOne, CircleTwo(9);
     CircleOne++;
     ++CircleTwo;
     cout << "CircleOne: " << CircleOne.GetRadius() << endl;
     cout << "CircleTwo: " << CircleTwo.GetRadius() << endl;
     CircleOne = CircleTwo;
     cout << "CircleOne: " << CircleOne.GetRadius() << endl;
     cout << "CircleTwo: " << CircleTwo.GetRadius() << endl;
return 0;
}
```

9. BUG BUSTERS: What is wrong with this implementation of the assignment operator?

```
SQUARE SQUARE ::operator=(const SQUARE & rhs)
{
      itsSide = new int;
      *itsSide = rhs.GetSide();
      return *this;
}
```

You must check to see whether `rhs` equals `this`, or the call to `a = a` will crash your program.

10. BUG BUSTERS: What is wrong with this implementation of `operator+`?

```
VeryShort  VeryShort::operator+ (const VeryShort& rhs)
{
   itsVal += rhs.GetItsVal();
   return *this;
}
```

This `operator+` is changing the value in one of the operands, rather than creating a new `VeryShort` object with the sum. The right way to do this is as follows:

```
VeryShort  VeryShort::operator+ (const VeryShort& rhs)
{
   return VeryShort(itsVal + rhs.GetItsVal());
}
```

Day 11

Quiz

1. What are the first and last elements in `SomeArray[25]`?

`SomeArray[0]`, `SomeArray[24]`

2. How do you declare a multidimensional array?

Write a set of subscripts for each dimension. For example, `SomeArray[2][3][2]` is a three-dimensional array. The first dimension has two elements, the second has three, and
the third has two.

3. Initialize the members of the array in Question 2.

```
SomeArray[2][3][2] = { { {1,2},{3,4},{5,6} } , { {7,8},{9,10},{11,12} } };
```

4. How many elements are in the array `SomeArray[10][5][20]`?

10x5x20=1,000

5. What is the maximum number of elements that you can add to a linked list?

There is no fixed maximum. It depends on how much memory you have available.

6. Can you use subscript notation on a linked list?

You can use subscript notation on a linked list only by writing your own class to contain the linked list and overloading the subscript operator.

7. What is the last character in the string "Brad is a nice guy"?

The null character.

Exercises

1. Declare a two-dimensional array that represents a tic-tac-toe game board.

```
int GameBoard[3][3];
```

2. Write the code that initializes all the elements in the array you created in Exercise 1 to the value `0`.

```
int GameBoard[3][3] = { {0,0,0},{0,0,0},{0,0,0} }
```

3. Write the declaration for a `Node` class that holds `unsigned short` integers.

```
 class Node
 {
 public:
    Node ();
    Node (int);
    ~Node();
    void SetNext(Node * node) { itsNext = node; }
    Node * GetNext() const { return itsNext; }
    int GetVal() const { return itsVal; }
    void Insert(Node *);
    void Display();
 private:
    int itsVal;
    Node * itsNext;
 };
```

4. BUG BUSTERS: What is wrong with this code fragment?

```
unsigned short SomeArray[5][4];
for (int i = 0; i<4; i++)
     for (int j = 0; j<5; j++)
          SomeArray[i][j] = i+j;
```

The array is 5 elements by 4 elements, but the code initializes 4x5.

5. BUG BUSTERS: What is wrong with this code fragment?

```
unsigned short SomeArray[5][4];
for (int i = 0; i<=5; i++)
     for (int j = 0; j<=4; j++)
          SomeArray[i][j] = 0;
```

You wanted to write `i<5`, but you wrote `i<=5` instead. The code will run when `i == 5` and `j == 4`, but there is no such element as `SomeArray[5][4]`.

Day 12

Quiz

1. What is a v-table?

A v-table, or virtual function table, is a common way for compilers to manage virtual functions in C++. The table keeps a list of the addresses of all the virtual functions and, depending on the runtime type of the object pointed to, invokes the right function.

2. What is a virtual destructor?

A destructor of any class can be declared to be virtual. When the pointer is deleted, the runtime type of the object will be assessed and the correct derived destructor invoked.

3. How do you show the declaration of a virtual constructor?

There are no virtual constructors.

4. How can you create a virtual copy constructor?

By creating a virtual method in your class, which itself calls the copy constructor.

5. How do you invoke a base member function from a derived class in which you've overridden that function?

```
Base::FunctionName();
```

6. How do you invoke a base member function from a derived class in which you have not overridden that function?

```
FunctionName();
```

7. If a base class declares a function to be virtual, and a derived class does not use the term `virtual` when overriding that class, is it still virtual when inherited by a third-generation class?

Yes, the virtuality is inherited and cannot be turned off.

8. What is the `protected` keyword used for?

`protected` members are accessible to the member functions of derived objects.

Exercises

1. Show the declaration of a virtual function taking an integer parameter and returning `void`.

```
virtual void SomeFunction(int);
```

2. Show the declaration of a class `Square`, which derives from `Rectangle`, which in turn derives from `Shape`.

```
class Square : public Rectangle
{};
```

3. If, in Exercise 2, `Shape` takes no parameters, `Rectangle` takes two (`length` and `width`), and `Square` takes only one (`length`), show the constructor initialization for `Square`.

```
Square::Square(int length):
     Rectangle(length, length){}
```

4. Write a virtual copy constructor for the class `Square` (from the preceding question).

```
class Square
      {
         public:
         // ...
         virtual Square * clone() const { return new Square(*this); }
       // ...
     };
```

5. BUG BUSTERS: What is wrong with this code snippet?

```
void SomeFunction (Shape);
Shape * pRect = new Rectangle;
SomeFunction(*pRect);
```

Perhaps nothing. `SomeFunction` expects a `Shape` object. You've passed it a `Rectangle` "sliced" down to a `Shape`. As long as you don't need any of the `Rectangle` parts, this will be fine. If you do need the `Rectangle` parts, you'll need to change `SomeFunction` to take a pointer or a reference to a `Shape`.

6. BUG BUSTERS: What is wrong with this code snippet?

```
class Shape()
{
public:
     Shape();
     virtual ~Shape();
     virtual Shape(const Shape&);
};
```

You can't declare a copy constructor to be virtual.

Day 13

Quiz

1. What is a down cast?

A down cast (also called "casting down") is a declaration that a pointer to a base class is to be treated as a pointer to a derived class.

2. What is the v-ptr?

The v-ptr, or virtual-function pointer, is an implementation detail of virtual functions. Each object in a class with virtual functions has a v-ptr, which points to the virtual function table for that class.

3. If a round rectangle has straight edges and rounded corners, your `RoundRect` class inherits both from `Rectangle` and from `Circle`, and they in turn both inherit from `Shape`, how many `Shapes` are created when you create a `RoundRect`?

If neither class inherits using the keyword `virtual`, two `Shapes` are created: one for `Rectangle` and one for `Shape`. If the keyword `virtual` is used for both classes, only one shared `Shape` is created.

4. If `Horse` and `Bird` inherit `virtual public` from `Animal`, do their constructors initialize the `Animal` constructor? If `Pegasus` inherits from both `Horse` and `Bird`, how does it initialize `Animal`'s constructor?

Both `Horse` and `Bird` initialize their base class, `Animal`, in their constructors. `Pegasus` does as well, and when a `Pegasus` is created, the `Horse` and `Bird` initializations of `Animal` are ignored.

5. Declare a class `Vehicle` and make it an abstract data type.

```
class Vehicle
{
     virtual void Move() = 0;
}
```

6. If a base class is an ADT, and it has three pure virtual functions, how many of these functions must be overridden in its derived classes?

None must be overridden unless you want to make the class non-abstract, in which case all three must be overridden.

Exercises

1. Show the declaration for a class `JetPlane`, which inherits from `Rocket` and `Airplane`.

```
class JetPlane : public Rocket, public Airplane
```

2. Show the declaration for `747`, which inherits from the `JetPlane` class described in Exercise 1.

```
class 747 : public JetPlane
```

3. Show the declarations for the classes `Car` and `Bus`, which each derive from the class `Vehicle`. Make `Vehicle` an ADT with two pure virtual functions. Make `Car` and `Bus` not be ADTs.

```
class Vehicle
{
     virtual void Move() = 0;
     virtual void Haul() = 0;
};

class Car : public Vehicle
{
     virtual void Move();
     virtual void Haul();
};

class Bus : public Vehicle
{
     virtual void Move();
     virtual void Haul();
};
```

4. Modify the program in Exercise 1 so that `Car` is an ADT, and derive `SportsCar` and `Coupe` from `Car`. In the `Car` class, provide an implementation for one of the pure virtual functions in `Vehicle` and make it non-pure.

```
class Vehicle
{
     virtual void Move() = 0;
     virtual void Haul() = 0;
};

class Car : public Vehicle
{
     virtual void Move();
};

class Bus : public Vehicle
{
     virtual void Move();
     virtual void Haul();
};

class SportsCar : public Car
{
     virtual void Haul();
};

class Coupe : public Car
{
     virtual void Haul();
};
```

Day 14

Quiz

1. Can static member variables be private?

Yes. They are member variables, and their access can be controlled like any other. If they are private, they can be accessed only by using member functions or, more commonly, static member functions.

2. Show the declaration for a static member variable.

```
static int itsStatic;
```

3. Show the declaration for a static function pointer.

```
static int SomeFunction();
```

4. Show the declaration for a pointer `to` function returning `long` and taking an integer parameter.

```
long (* function)(int);
```

5. Modify the pointer in Exercise 4 to be a pointer to member function of class `Car`

```
long ( Car::*function)(int);
```

6. Show the declaration for an array of 10 pointers as defined in Exercise 5.

```
(long ( Car::*function)(int) theArray [10];
```

Exercises

1. Write a short program declaring a class with one member variable and one static member variable. Have the constructor initialize the member variable and increment the static member variable. Have the destructor decrement the member variable.

```
class myClass
{
public:
   myClass();
   ~myClass();
private:
   int itsMember;
   static int itsStatic;
};

myClass::myClass():
 itsMember(1)
{
   itsStatic++;
}

myClass::~myClass()
{
   itsStatic--;
}

int myClass::itsStatic = 0;

int main()
{}
```

2. Using the program from Exercise 1, write a short driver program that makes three objects and then displays their member variables and the static member variable. Then destroy each object and show the effect on the static member variable.

```
#include <iostream.h>

class myClass
{
public:
   myClass();
   ~myClass();
   void ShowMember();
   void ShowStatic();
private:
   int itsMember;
   static int itsStatic;
};

myClass::myClass():
 itsMember(1)
{
   itsStatic++;
}

myClass::~myClass()
{
   itsStatic--;
   cout << "In destructor. ItsStatic: " << itsStatic << endl;
}

void myClass::ShowMember()
{
   cout << "itsMember: " << itsMember << endl;
}

void myClass::ShowStatic()
{
   cout << "itsStatic: " << itsStatic << endl;
}
int myClass::itsStatic = 0;

int main()
{
   myClass obj1;
   obj1.ShowMember();
   obj1.ShowStatic();

   myClass obj2;
   obj2.ShowMember();
   obj2.ShowStatic();

   myClass obj3;
   obj3.ShowMember();
   obj3.ShowStatic();
 return 0;
}
```

3. Modify the program from Exercise 2 to use a static member function to access the static member variable. Make the static member variable private.

```
#include <iostream.h>

class myClass
{
public:
   myClass();
   ~myClass();
   void ShowMember();
   static int GetStatic();
private:
   int itsMember;
   static int itsStatic;
};

myClass::myClass():
 itsMember(1)
{
   itsStatic++;
}

myClass::~myClass()
{
   itsStatic--;
   cout << "In destructor. ItsStatic: " << itsStatic << endl;
}

void myClass::ShowMember()
{
   cout << "itsMember: " << itsMember << endl;
}

int myClass::itsStatic = 0;

void myClass::GetStatic()
{
   return itsStatic;
}

```

```
int main()
{
   myClass obj1;
   obj1.ShowMember();
   cout << "Static: " << myClass::GetStatic() << endl;

   myClass obj2;
   obj2.ShowMember();
   cout << "Static: " << myClass::GetStatic() << endl;

   myClass obj3;
   obj3.ShowMember();
   cout << "Static: " << myClass::GetStatic() << endl;
 return 0;
}
```

4. Write a pointer to a member function to access the non-static member data in the program in Exercise 3, and use that pointer to print the value of that data.

```
#include <iostream.h>

class myClass
{
public:
   myClass();
   ~myClass();
   void ShowMember();
   static int GetStatic();
private:
   int itsMember;
   static int itsStatic;
};

myClass::myClass():
 itsMember(1)
{
   itsStatic++;
}

myClass::~myClass()
{
   itsStatic--;
   cout << "In destructor. ItsStatic: " << itsStatic << endl;
}

void myClass::ShowMember()
{
   cout << "itsMember: " << itsMember << endl;
}

int myClass::itsStatic = 0;

int myClass::GetStatic()
{
   return itsStatic;
}

int main()
{
   void (myClass::*PMF) ();

   PMF=myClass::ShowMember;

   myClass obj1;
   (obj1.*PMF)();
   cout << "Static: " << myClass::GetStatic() << endl;
```

```

      myClass obj2;
      (obj2.*PMF)();
      cout << "Static: " << myClass::GetStatic() << endl;

      myClass obj3;
      (obj3.*PMF)();
      cout << "Static: " << myClass::GetStatic() << endl;
    return 0;
   }
```

5. Add two more member variables to the class from the previous questions. Add accessor functions that get the value of this data and give all the member functions the same return values and signatures. Use the pointer to the member function to access these functions.

```
   #include <iostream.h>

   class myClass
   {
   public:
      myClass();
      ~myClass();
      void ShowMember();
      void ShowSecond();
      void ShowThird();
      static int GetStatic();
   private:
      int itsMember;
      int itsSecond;
      int itsThird;
      static int itsStatic;
   };

   myClass::myClass():
    itsMember(1),
    itsSecond(2),
    itsThird(3)
   {
      itsStatic++;
   }

   myClass::~myClass()
   {
      itsStatic--;
      cout << "In destructor. ItsStatic: " << itsStatic << endl;
   }

   void myClass::ShowMember()
   {
      cout << "itsMember: " << itsMember << endl;
   }

   void myClass::ShowSecond()
   {
      cout << "itsSecond: " << itsSecond << endl;
   }

   void myClass::ShowThird()
   {
      cout << "itsThird: " << itsThird << endl;
   }
   int myClass::itsStatic = 0;

   int myClass::GetStatic()
   {
      return itsStatic;
```

```
}

int main()
{
   void (myClass::*PMF) ();

   myClass obj1;
   PMF=myClass::ShowMember;
   (obj1.*PMF)();
   PMF=myClass::ShowSecond;
   (obj1.*PMF)();
   PMF=myClass::ShowThird;
   (obj1.*PMF)();
   cout << "Static: " << myClass::GetStatic() << endl;

   myClass obj2;
   PMF=myClass::ShowMember;
   (obj2.*PMF)();
   PMF=myClass::ShowSecond;
   (obj2.*PMF)();
   PMF=myClass::ShowThird;
   (obj2.*PMF)();
   cout << "Static: " << myClass::GetStatic() << endl;

   myClass obj3;
   PMF=myClass::ShowMember;
   (obj3.*PMF)();
   PMF=myClass::ShowSecond;
   (obj3.*PMF)();
   PMF=myClass::ShowThird;
   (obj3.*PMF)();
   cout << "Static: " << myClass::GetStatic() << endl;
 return 0;
}
```

Day 15

Quiz

1. How do you establish an is-a relationship?

With public inheritance.

2. How do you establish a has-a relationship?

With containment; that is, one class has a member that is an object of another type.

3. What is the difference between containment and delegation?

Containment describes the idea of one class having a data member that is an object of another type. Delegation expresses the idea that one class uses another class to accomplish a task or goal. Delegation is usually accomplished by containment.

4. What is the difference between delegation and implemented-in-terms-of?

Delegation expresses the idea that one class uses another class to accomplish a task or goal. Implemented-in-terms-of expresses the idea of inheriting implementation from another class.

5. What is a friend function?

A friend function is a function declared to have access to the protected and private members of your class.

6. What is a friend class?

A friend class is a class declared so that all its member functions are friend functions of your class.

7. If `Dog` is a friend of `Boy`, is `Boy` a friend of `Dog`?

No, friendship is not commutative.

8. If `Dog` is a friend of `Boy`, and `Terrier` derives from `Dog`, is `Terrier` a friend of `Boy`?

No, friendship is not inherited.

9. If `Dog` is a friend of `Boy` and `Boy` is a friend of `House`, is `Dog` a friend of `House`?

No, friendship is not associative.

10. Where must the declaration of a friend function appear?

Anywhere within the class declaration. It makes no difference whether you put the declaration within the `public:`, `protected:`, or `private:` access areas.

Exercises

1. Show the declaration of a class `Animal` that contains a data member that is a `String` object.

```
class Animal:
{
private:
   String itsName;
};
```

2. Show the declaration of a class `BoundedArray` that is an array.

```
class boundedArray : public Array
{
//...
}
```

3. Show the declaration of a class `Set` that is declared in terms of an array.

```
class Set : private Array
{
// ...
}
```

4. Modify Listing 15.1 to provide the `String` class with an extraction operator (`>>`).

```
#include <iostream.h>
#include <string.h>

class String
```

```
  {
     public:
        // constructors
        String();
         String(const char *const);
         String(const String &);
        ~String();

        // overloaded operators
        char & operator[](int offset);
        char operator[](int offset) const;
        String operator+(const String&);
        void operator+=(const String&);
        String & operator= (const String &);
        friend ostream& operator<<
            ( ostream&   _theStream,String& theString);
        friend istream& operator>>
           ( istream& _theStream,String& theString);
        // General accessors
        int GetLen()const { return itsLen; }
        const char * GetString() const { return itsString; }
        // static int ConstructorCount;

     private:
        String (int);          // private constructor
        char * itsString;
        unsigned short itsLen;

 };

  ostream& operator<<( ostream& theStream,String& theString)
  {
      theStream << theString.GetString();
      return theStream;
  }

  istream& operator>>( istream& theStream,String& theString)
  {
      theStream >> theString.GetString();
      return theStream;
  }

  int main()
  {
     String theString("Hello world.");
     cout << theString;
return 0;
  }
```

5. BUG BUSTERS: What is wrong with this program?

```
#include <iostream.h>

class Animal;

void setValue(Animal& , int);


class Animal
{
public:
   int GetWeight()const { return itsWeight; }
   int GetAge() const { return itsAge; }
private:
   int itsWeight;
   int itsAge;
};

void setValue(Animal& theAnimal, int theWeight)
{
   friend class Animal;
   theAnimal.itsWeight = theWeight;
}

int main()
{
   Animal peppy;
   setValue(peppy,5);
 return 0;
}
```

You can't put the `friend` declaration into the function. You must declare the function to be a friend in the class.

6. Fix the listing in Exercise 5 so that it will compile.

```
#include <iostream.h>

class Animal;

void setValue(Animal& , int);


class Animal
{
public:
   friend void setValue(Animal&, int);
   int GetWeight()const { return itsWeight; }
   int GetAge() const { return itsAge; }
private:
   int itsWeight;
   int itsAge;
};

void setValue(Animal& theAnimal, int theWeight)
{
   theAnimal.itsWeight = theWeight;
}

int main()
{
   Animal peppy;
   setValue(peppy,5);
 return 0;
}
```

7. BUG BUSTERS: What is wrong with this code?

```
#include <iostream.h>

class Animal;

void setValue(Animal& , int);
void setValue(Animal& ,int,int);

class Animal
{
friend void setValue(Animal& ,int); // here's the change!
private:
   int itsWeight;
   int itsAge;
};
```

```

void setValue(Animal& theAnimal, int theWeight)
{
    theAnimal.itsWeight = theWeight;
}


void setValue(Animal& theAnimal, int theWeight, int theAge)
{
   theAnimal.itsWeight = theWeight;
   theAnimal.itsAge = theAge;
}

int main()
{
   Animal peppy;
   setValue(peppy,5);
   setValue(peppy,7,9);
 return 0;
}
```

The function `setValue(Animal&,int)` was declared to be a friend, but the overloaded function `setValue(Animal&,int,int)` was not declared to be a friend.

8. Fix Exercise 7 so it compiles.

```
#include <iostream.h>

class Animal;

void setValue(Animal& , int);
void setValue(Animal& ,int,int); // here's the change!

class Animal
{
friend void setValue(Animal& ,int);
friend void setValue(Animal& ,int,int);
private:
   int itsWeight;
   int itsAge;
};

void setValue(Animal& theAnimal, int theWeight)
{
    theAnimal.itsWeight = theWeight;
}


void setValue(Animal& theAnimal, int theWeight, int theAge)
{
   theAnimal.itsWeight = theWeight;
   theAnimal.itsAge = theAge;
}

int main()
{
   Animal peppy;
   setValue(peppy,5);
   setValue(peppy,7,9);
 return 0;
}
```

Day 16

Quiz

1. What is the insertion operator and what does it do?

The insertion operator (<<) is a member operator of the `ostream` object and is used for writing to the output device.

2. What is the extraction operator and what does it do?

The extraction operator (>>) is a member operator of the `istream` object and is used for writing to your program's variables.

3. What are the three forms of `cin.get()` and what are their differences?

The first form of `get()` is without parameters. This returns the value of the character found, and will return `EOF` (end of file) if the end of the file is reached.
The second form of `cin.get()` takes a character reference as its parameter; that character is filled with the next character in the input stream. The return value is an `iostream` object.
The third form of `cin.get()` takes an array, a maximum number of characters to get, and a terminating character. This form of `get()` fills the array with up to one fewer characters than the maximum (appending null) unless it reads the terminating character, in which case it immediately
writes a null and leaves the terminating character in the buffer.

4. What is the difference between `cin.read()` and `cin.getline()`?

`cin.read()` is used for reading binary data structures.
`getline()` is used to read from the `istream`'s buffer.

5. What is the default width for ouputting a `long` integer using the insertion operator?

Wide enough to display the entire number.

6. What is the return value of the insertion operator?

A reference to an `istream` object.

7. What parameter does the constructor to an `ofstream` object take?

The filename to be opened.

8. What does the `ios::ate` argument do?

`ios::ate` places you at the end of the file, but you can write data anywhere in the file.

Exercises

1. Write a program that writes to the four standard `iostream` objects: `cin`, `cout`, `cerr`, and `clog`.

```
#include <iostream.h>
int main()
{
   int x;
   cout << "Enter a number: ";
```

```
      cin >> x;
      cout << "You entered: " << x << endl;
      cerr << "Uh oh, this to cerr!" << endl;
      clog << "Uh oh, this to clog!" << endl;
    return 0;
   }
```

2. Write a program that prompts the user to enter her full name and then displays it on the screen.

```
   #include <iostream.h>
   int main()
   {
      char name[80];
      cout << "Enter your full name: ";
      cin.getline(name,80);
      cout << "\nYou entered: " << name << endl;
   return 0;
   }
```

3. Rewrite Listing 16.9 to do the same thing, but without using `putback()` or `ignore()`.

```
    // Listing
    #include <iostream.h>

    int main()
    {
       char ch;
       cout << "enter a phrase: ";
       while ( cin.get(ch) )
       {
          switch (ch)
          {
            case `!':
               cout << `$';
               break;
            case `#':
               break;
            default:
               cout << ch;
               break;
         }
       }
    return 0;
    }
```

4. Write a program that takes a filename as a parameter and opens the file for reading. Read every character of the file and display only the letters and punctuation to the screen. (Ignore all non-printing characters.) Then close the file and exit.

```
   #include <fstream.h>
   enum BOOL { FALSE, TRUE };

   int main(int argc, char**argv)   // returns 1 on error
   {

      if (argc != 2)
      {
         cout << "Usage: argv[0] <infile>\n";
         return(1);
      }

   // open the input stream
      ifstream fin (argv[1],ios::binary);
      if (!fin)
      {
         cout << "Unable to open " << argv[1] << " for reading.\n";
         return(1);
      }

      char ch;
      while ( fin.get(ch))
         if ((ch > 32 && ch < 127) || ch == `\n' || ch == `\t')
            cout << ch;
      fin.close();
   }
```

5. Write a program that displays its command-line arguments in reverse order and does not display the program name.

```
      #include <fstream.h>

      int main(int argc, char**argv)   // returns 1 on error
      {
         for (int ctr = argc; ctr ; ctr--)
           cout << argv[ctr] << " ";
      }
```

Day 17

Quiz

1. What is an inclusion guard?

Inclusion guards are used to protect a header file from being included into a program more than once.

2. How do you instruct your compiler to print the contents of the intermediate file showing the effects of the preprocessor?

This quiz question must be answered by you, depending on the compiler you are using.

3. What is the difference between `#define debug 0` and `#undef debug`?

`#define debug 0` defines the term `debug` to equal 0 (zero). Everywhere the word `debug` is found, the character 0 will be substituted. `#undef debug` removes any definition of `debug`; when the word `debug` is found in the file, it will be left unchanged.

4. Name four predefined macros.

`__DATE__`, `__TIME__`, `__FILE__`, `__LINE__`

5. Why can't you call `invariants()` as the first line of your constructor?

The job of your constructor is to create the object. The class invariants cannot and should not exist before the object is fully created, so any meaningful use of `invariants()` will return false until the constructor is finished.

Exercises

1. Write the inclusion guard statements for the header file `STRING.H`.

```
#ifndef STRING_H
#define STRING_H
```

```
...
#endif
```

2. Write an `assert()` macro that prints an error message and the file and line number if debug level is 2, prints just a message (without file and line number) if the level is 1, and does nothing if the level is 0.

```
  #include <iostream.h>

  #ifndef DEBUG
  #define ASSERT(x)
  #elif DEBUG == 1
  #define ASSERT(x) \
    if (! (x)) \
    { \
       cout << "ERROR!! Assert " << #x << " failed\n"; \
               }
  #elif DEBUG == 2
  #define ASSERT(x) \
    if (! (x) ) \
    { \
     cout << "ERROR!! Assert " << #x << " failed\n"; \
    cout << " on line " << __LINE__  << "\n"; \
     cout << " in file " << __FILE__ << "\n";  \
}
  #endif
```

3. Write a macro `DPrint` that tests whether debug is defined, and if it is, prints the value passed in as a parameter.

```
#ifndef DEBUG
#define DPRINT(string)
#else
#define DPRINT(STRING) cout << #STRING ;
#endif
```

4. Write a function that prints an error message. The function should print the line number and filename where the error occurred. Note that the line number and filename are passed in to this function.

```
#include <iostream.h>

  void ErrorFunc(
     int LineNumber,
     const char * FileName)
 {
     cout << "An error occurred in file ";
      cout << FileName;
      cout << " at line "
      cout << LineNumber << endl;
 }
```

5. How would you call the preceding error function?

```
 // driver program to exercise ErrorFunc
 int main()
 {
      cout << "An error occurs on next line!";
      ErrorFunc(__LINE__, __FILE__);
return 0;
 }
```

Note that the `__LINE__` and `__FILE__` macros are used at the point of the error, and not in the error function. If you used them in the error function, they would report the line and file for the error function itself.

6. Write an `assert()` macro that uses the error function from Exercise 4, and write a driver program that calls that `assert()` macro.

```
1:     #include <iostream.h>
2:
3:      #define DEBUG // turn error handling on
4:
5:      #ifndef DEBUG
6:      #define ASSERT(x)
7:      #else
8:      #define ASSERT(X) \
9:         if (! (X)) \
10:         {   \
11:            ErrorFunc(__LINE__, __FILE__); \
12:         }
13:      #endif
14:
15:      void ErrorFunc(int LineNumber, const char * FileName)
16:      {
17:           cout << "An error occurred in file ";
18:           cout << FileName;
19:           cout << " at line ";
20:           cout << LineNumber << endl;
21:      }
22:
23:      // driver program to exercise ErrorFunc
24:      int main()
25:      {
26:         int x = 5;
27:         ASSERT(x >= 5);  // no error
28:         x = 3;
29:         ASSERT(x >= 5); // error!
30:     return 0;
31:      }
```

Note that in this case, the `__LINE__` and `__FILE__` macros can be called in the `assert()` macro and will still give the correct line (line 29). This is because the `assert()` macro is expanded in place, where it is called. Therefore, this program is evaluated exactly as if `main()` were written as

```
 // driver program to exercise ErrorFunc
 int main()
 {
    int x = 5;
    if (! (x >= 5)) {ErrorFunc(__LINE__, __FILE__);}
    x = 3;
    if (! (x >= 5)) {ErrorFunc(__LINE__, __FILE__);}
return 0;
}
```

Day 18

Quiz

1. What is the difference between object-oriented programming and procedural programming?

Procedural programming focuses on functions separate from data. Object-oriented programming ties data and functionality together into objects, and focuses on the interaction among the objects.

2. To what does "event-driven" refer?

Event-driven programs are distinguished by the fact that action is taken only in response to some form of (usually external) simulation, such as a user's keyboard or mouse input.

3. What are the stages in the development cycle?

Typically, the development cycle includes analysis, design, coding, testing, programming, and interaction and feedback among these stages.

4. What is a rooted hierarchy?

A rooted hierarchy is one in which all the classes in the program derive directly or indirectly from a single base class.

5. What is a driver program?

A driver program is simply a function that is designed to exercise whatever objects and functions you are currently programming.

6. What is encapsulation?

Encapsulation refers to the (desirable) trait of bringing together in one class all the data and functionality of one discrete entity.

Exercises

1. Suppose you had to simulate the intersection of Massachusetts Avenue and Vassar Street--two typical two-lane roads with traffic lights and crosswalks. The purpose of the simulation is to determine whether the timing of the traffic signal allows for a smooth flow of traffic.

What kinds of objects should be modeled in the simulation? What should be the classes defined for the simulation?

Cars, motorcycles, trucks, bicycles, pedestrians, and emergency vehicles all use the intersection. In addition, there is a traffic signal with Walk/Don't Walk lights.

Should the road surface be included in the simulation?

Certainly, road quality can have an effect on the traffic, but for a first design, it may be simpler to leave this consideration aside. The first object is probably the intersection itself. Perhaps the intersection object maintains lists of cars waiting to pass through the signal in each direction, as well as lists of people waiting to cross at the crosswalks. It will need methods to choose which and how many cars and people go through the intersection.

There will be only one intersection, so you may want to consider how you will ensure that only one object is instantiated (hint: think about static methods and protected access).

People and cars are both clients of the intersection. They share a number of characteristics: they can appear at any time, there can be any number of them, and they both wait at the signal (although in different lines). This suggests that you will want to consider a common base class for pedestrians and cars.

The classes would therefore include

```
class Entity;   // a client of the intersection
```

```
// the root of all cars, trucks, bicycles and emergency vehicles.
class Vehicle : Entity ...;

// the root of all People
class Pedestrian : Entity...;

class Car : public Vehicle...;
class Truck : public Vehicle...;
class Motorcycle : public Vehicle...;
class Bicycle : public Vehicle...;
class Emergency_Vehicle : public Vehicle...;

// contains lists of cars and people waiting to pass


class Intersection;
```

2. Suppose the intersections from Exercise 1 were in a suburb of Boston, which has arguably the unfriendliest streets in the United States. At any time, there are three kinds of Boston drivers:

Locals, who continue to drive through intersections after the light turns red

Tourists, who drive slowly and cautiously (in a rental car, typically)

Taxis, which have a wide variation of driving patterns, depending on the kinds of passengers in the cabs

Also, Boston has two kinds of pedestrians:

Locals, who cross the street whenever they feel like it and seldom use the crosswalk buttons

Tourists, who always use the crosswalk buttons and only cross when the Walk/Don't Walk light permits.

Finally, Boston has bicyclists who never pay attention to stoplights

How do these considerations change the model?

A reasonable start on this would be to create derived objects that model the refinements suggested by the problem:

```
class Local_Car : public Car...;
          class Tourist_Car : public Car...;
          class Taxi : public Car...;
          class Local_Pedestrian : public
Pedestrian...;
          class Tourist_Pedestrian : public
Pedestrian...;
          class Boston_Bicycle : public Bicycle...;
```

By using virtual methods, each class can modify the generic behavior to meet its own specifications. For example, the Boston driver can react to a red light differently than a tourist does, while still inheriting the generic behaviors that continue to apply.

3. You are asked to design a group scheduler. The software allows you to arrange meetings among individuals or groups, and to reserve a limited number of conference rooms. Identify the principal subsystems.

Two discrete programs need to be written for this project: the client, which the users run; and the server, which would run on a separate machine. In addition, the client machine would have an administrative component to enable a system administrator to add new people and rooms.

If you decide to implement this as a client/server model, the client would accept input from users and generate a request to the server. The server would service the request and send back the results to the client. With this model, many people can schedule meetings at the same time.

On the client's side, there are two major subsystems in addition to the administrative module: the user interface and the communications subsystem. The server's side consists of three main subsystems: communications, scheduling, and a mail interface that would announce to the user when changes have occurred in the schedule.

4. Design and show the interfaces to the classes in the room-reservation portion of the program discussed in Exercise 3.

A meeting is defined as a group of people reserving a room for a certain amount of time. The person making the schedule might want a specific room, or a specified time; but the scheduler must always be told how long the meeting will last and who is required to attend.

The objects will probably include the users of the system as well as the conference rooms. Don't forget to include classes for the calendar, and perhaps a `Meeting` class that encapsulates all that is known about a particular event.

The prototypes for the classes might include:

```
class Calendar_Class;          // forward reference
class Meeting;               // forward reference
class Configuration
{
public:
     Configuration();
     ~Configuration();
     Meeting Schedule( ListOfPerson&, Delta Time
duration );
     Meeting Schedule( ListOfPerson&, Delta Time
duration, Time );
     Meeting Schedule( ListOfPerson&, Delta Time
duration, Room );
     ListOfPerson&     People();     // public
accessors
     ListOfRoom&     Rooms();     // public accessors
protected:
     ListOfRoom     rooms;
     ListOfPerson     people;
};
typedef long      Room_ID;
class Room
{
public:
     Room( String name, Room_ID id, int capacity,
String directions = "", String description = "" );
     ~Room();
     Calendar_Class Calendar();

protected:
     Calendar_Class     calendar;
     int          capacity;
     Room_ID     id;
     String          name;
     String          directions;          // where is
this room?
     String          description;
};
typedef long Person_ID;
class Person
{
public:
     Person( String name, Person_ID id );
     ~Person();
     Calendar_Class Calendar();          // the access
point to add meetings
protected:
     Calendar_Class     calendar;
     Person_ID     id;
     String          name;
};
class Calendar_Class
{
public:
     Calendar_Class();
     ~Calendar_Class();

     void Add( const Meeting& );               // add a
meeting to the calendar
     void Delete( const Meeting& );
     Meeting* Lookup( Time );               // see if
there is a meeting at the
                                   // given time

     Block( Time, Duration, String reason = "" );
// allocate time to yourself...

protected:
     OrderedListOfMeeting meetings;
};
class Meeting
{
public:
     Meeting( ListOfPerson&, Room room,
          Time when, Duration duration, String purpose
= "" );
     ~Meeting();
protected:
     ListOfPerson     people;
     Room          room;
     Time          when;
     Duration     duration;
     String          purpose;
};
```

Day 19

Quiz

1. What is the difference between a template and a macro?

Templates are built into the C++ language and are type-safe. Macros are implemented by the preprocessor and are not type-safe.

2. What is the difference between the parameter to a template and the parameter to a function?

The parameter to the template creates an instance of the template for each type. If you create six template instances, six different classes or functions are created. The parameters to the function change the behavior or data of the function, but only one function is created.

3. What is the difference between a type-specific template friend class and a general template friend class?

The general template friend function creates one function for every type of the parameterized class; the type-specific function creates a type-specific instance for each instance of the parameterized class.

4. Is it possible to provide special behavior for one instance of a template but not for other instances?

Yes, create a specialized function for the particular instance. In addition to creating `Array<t>::SomeFunction()`, also create `Array<int>::SomeFunction()` to change the behavior for integer arrays.

5. How many static variables are created if you put one static member into a template class definition?
One for each instance of the class.

Exercises

1. Create a template based on this `List` class:

```
class List
{
private:

public:
     List():head(0),tail(0),theCount(0) {}
     virtual ~List();

     void insert( int value );
     void append( int value );
     int is_present( int value ) const;
     int is_empty() const { return head == 0; }
     int count() const { return theCount; }
private:
     class ListCell
     {
     public:
          ListCell(int value, ListCell *cell = 0):val(value),next(cell){}
          int val;
          ListCell *next;
     };
     ListCell *head;
     ListCell *tail;
     int theCount;
};
```

Here is one way to implement this template:

```
template <class Type>
class List
{

public:
     List():head(0),tail(0),theCount(0) { }
     virtual ~List();

     void insert( Type value );
     void append( Type value );
     int is_present( Type value ) const;
     int is_empty() const { return head == 0; }
     int count() const { return theCount; }

private:
     class ListCell
     {
     public:
          ListCell(Type value, ListCell *cell = 0):val(value),next(cell){}
          Type val;
          ListCell *next;
     };

     ListCell *head;
     ListCell *tail;
     int theCount;
};
```

2. Write the implementation for the `List` class (non-template) version.

```
void List::insert(int value)
{
     ListCell *pt = new ListCell( value, head );
     assert (pt != 0);

     // this line added to handle tail
     if ( head == 0 ) tail = pt;

     head = pt;
     theCount++;
}

void List::append( int value )
{
     ListCell *pt = new ListCell( value );
     if ( head == 0 )
          head = pt;
     else
          tail->next = pt;

     tail = pt;
     theCount++;
}

int List::is_present( int value ) const
{
     if ( head == 0 ) return 0;
     if ( head->val == value || tail->val == value )
          return 1;

     ListCell *pt = head->next;
     for (; pt != tail; pt = pt->next)
          if ( pt->val == value )
               return 1;

     return 0;
}
```

3. Write the template version of the implementations.

```
template <class Type>
List<Type>::~List()
{
     ListCell *pt = head;

     while ( pt )
     {
          ListCell *tmp = pt;
          pt = pt->next;
          delete tmp;
     }
     head = tail = 0;
}

template <class Type>
void List<Type>::insert(Type value)
{
     ListCell *pt = new ListCell( value, head );
     assert (pt != 0);

     // this line added to handle tail
     if ( head == 0 ) tail = pt;

     head = pt;
     theCount++;
}

template <class Type>
void List<Type>::append( Type value )
{
     ListCell *pt = new ListCell( value );
     if ( head == 0 )
          head = pt;
     else
          tail->next = pt;

     tail = pt;
     theCount++;
}

template <class Type>
int List<Type>::is_present( Type value ) const
{
     if ( head == 0 ) return 0;
     if ( head->val == value || tail->val == value )
          return 1;

     ListCell *pt = head->next;
     for (; pt != tail; pt = pt->next)
          if ( pt->val == value )
               return 1;

     return 0;
}
```

4. Declare three `List` objects: a list of strings, a list of `Cat`s and a list of `int`s.

```
List<String> string_list;
List<Cat> Cat_List;
List<int> int_List;
```

5. BUG BUSTERS: What is wrong with the following code? (Assume the `List` template is defined, and `Cat` is the class defined earlier in the book.)

```
List<Cat> Cat_List;
Cat Felix;
CatList.append( Felix );
cout << "Felix is " <<
     ( Cat_List.is_present( Felix ) ) ? "" : "not " << "present\n";
```

Hint: (this is tough) *What makes `Cat` different from `int`?*

`Cat` doesn't have `operator ==` defined; all operations that compare the values in the `List` cells, such as `is_present`, will result in compiler errors. To reduce the chance of this, put copious comments before the template definition stating what operations must be defined for the instantiation to compile.

6. Declare friend `operator ==` for `List`.

```
friend int operator==( const Type& lhs, const Type& rhs );
```

7. Implement friend `operator ==` for `List`.

```
template <class Type>
int List<Type>::operator==( const Type& lhs, const Type& rhs )
{
     // compare lengths first
     if ( lhs.theCount != rhs.theCount )
          return 0;     // lengths differ

     ListCell *lh = lhs.head;
     ListCell *rh = rhs.head;

     for(; lh != 0; lh = lh.next, rh = rh.next )
          if ( lh.value != rh.value )
               return 0;

     return 1;          // if they don't differ, they must match
}
```

8. *Does `operator==` have the same problem as in Exercise 4?*

Yes, because comparing the array involves comparing the elements, `operator!=` must be defined for the elements as well.

9. Implement a template function for `swap`, which exchanges two variables.

```
// template swap:
// must have assignment and the copy constructor defined for the Type.
template <class Type>
void swap( Type& lhs, Type& rhs)
{
     Type temp( lhs );
     lhs = rhs;
     rhs = temp;
}
```

Day 20

Quiz

1. What is an exception?

An exception is an object that is created as a result of invoking the keyword `throw`. It is used to signal an exceptional condition, and is passed up the call stack to the first `catch` statement that handles its type.

2. What is a `try` block?

A `try` block is a set of statements that might generate an exception.

3. What is a `catch` statement?

A `catch` statement has a signature of the type of exception it handles. It follows a `try` block and acts as the receiver of exceptions raised within the `try` block.

4. What information can an exception contain?

An exception is an object and can contain any information that can be defined within a user-created class.

5. When are exception objects created?

Exception objects are created when you invoke the keyword `throw`.

6. Should you pass exceptions by value or by reference?

In general, exceptions should be passed by reference. If you don't intend to modify the contents of the exception object, you should pass a `const` reference.

7. Will a `catch` statement catch a derived exception if it is looking for the base class?

Yes, if you pass the exception by reference.

8. If there are two `catch` statements, one for base and one for derived, which should come first?

`catch` statements are examined in the order they appear in the source code. The first `catch` statement whose signature matches the exception is used.

9. What does `catch(...)` mean?

`catch(...)` will catch any exception of any type.

10. What is a breakpoint?

A breakpoint is a place in the code where the debugger will stop execution.

Exercises

1. Create a `try` block, a `catch` statement, and a simple exception.

```
#include <iostream.h>
class OutOfMemory {};
int main()
{

     try
     {
          int *myInt = new int;
          if (myInt == 0)
               throw OutOfMemory();
     }
     catch (OutOfMemory)
     {
          cout << "Unable to allocate memory!\n";
     }
return 0;
```

```
 }
```

2. Modify the answer from Exercise 1, put data into the exception along with an accessor function, and use it in the `catch` block.

```
#include <iostream.h>
#include <stdio.h>
#include <string.h>
class OutOfMemory
{
public:
     OutOfMemory(char *);
     char* GetString() { return itsString; }
private:
     char* itsString;
};

OutOfMemory::OutOfMemory(char * theType)
{
     itsString = new char[80];
     char warning[] = "Out Of Memory! Can't allocate room for: ";
     strncpy(itsString,warning,60);
     strncat(itsString,theType,19);
}

int main()
{

     try
     {
          int *myInt = new int;
          if (myInt == 0)
               throw OutOfMemory("int");
     }
     catch (OutOfMemory& theException)
     {
          cout << theException.GetString();
     }
return 0;
 }
```

3. Modify the class from Exercise 2 to be a hierarchy of exceptions. Modify the `catch` block to use the derived objects and the base objects.

```
#include <iostream.h>

// Abstract exception data type
class Exception
{
public:
   Exception(){}
   virtual ~Exception(){}
   virtual void PrintError() = 0;
};

// Derived class to handle memory problems.
// Note no allocation of memory in this class!
class OutOfMemory : public Exception
{
public:
   OutOfMemory(){}
   ~OutOfMemory(){}
   virtual void PrintError();
private:
};
```

```

void OutOfMemory::PrintError()
{
   cout << "Out of Memory!!\n";
}

// Derived class to handle bad numbers
class RangeError : public Exception
{
public:
   RangeError(unsigned long number){badNumber = number;}
   ~RangeError(){}
   virtual void PrintError();
   virtual unsigned long GetNumber() { return badNumber; }
   virtual void SetNumber(unsigned long number) {badNumber =                        Ânumber;}
private:
   unsigned long badNumber;
};

void RangeError::PrintError()
{
   cout << "Number out of range. You used " << GetNumber() <<                         Â"!!\n";
}

void MyFunction();  // func. prototype

int main()
{
   try
   {
      MyFunction();
   }
   // Only one catch required, use virtual functions to do the
   // right thing.
   catch (Exception& theException)
   {
      theException.PrintError();
   }
   return 0;
 }

 void MyFunction()
 {
      unsigned int *myInt = new unsigned int;
      long testNumber;
      if (myInt == 0)
         throw OutOfMemory();
      cout << "Enter an int: ";
      cin >> testNumber;
      // this weird test should be replaced by a series
      // of tests to complain about bad user input
      if (testNumber > 3768 || testNumber < 0)
         throw RangeError(testNumber);

      *myInt = testNumber;
      cout << "Ok. myInt: " << *myInt;
      delete myInt;
}
```

4. Modify the program from Exercise 3 to have three levels of function calls.

```
#include <iostream.h>

// Abstract exception data type
class Exception
{
 public:
    Exception(){}
    virtual ~Exception(){}
    virtual void PrintError() = 0;
 };

 // Derived class to handle memory problems.
 // Note no allocation of memory in this class!
 class OutOfMemory : public Exception
 {
 public:
    OutOfMemory(){}
    ~OutOfMemory(){}
    virtual void PrintError();
 private:
 };

 void OutOfMemory::PrintError()
 {
    cout << "Out of Memory!!\n";
 }

 // Derived class to handle bad numbers
 class RangeError : public Exception
 {
 public:
    RangeError(unsigned long number){badNumber = number;}
    ~RangeError(){}
    virtual void PrintError();
    virtual unsigned long GetNumber() { return badNumber; }
    virtual void SetNumber(unsigned long number) {badNumber =                       Ânumber;}
 private:
    unsigned long badNumber;
 };

 void RangeError::PrintError()
 {
    cout << "Number out of range. You used " << GetNumber() <<                        Â"!!\n";
 }

 // func. prototypes
 void MyFunction();
 unsigned int * FunctionTwo();
 void FunctionThree(unsigned int *);

 int main()
 {
    try
    {
       MyFunction();
    }
    // Only one catch required, use virtual functions to do the
    // right thing.
    catch (Exception& theException)
    {
       theException.PrintError();
    }
    return 0;
  }

 unsigned int * FunctionTwo()
 {
    unsigned int *myInt = new unsigned int;
   if (myInt == 0)
     throw OutOfMemory();
   return myInt;
}
```

```
 
     void MyFunction()
     {
          unsigned int *myInt = FunctionTwo();

          FunctionThree(myInt);
          cout << "Ok. myInt: " << *myInt;
          delete myInt;
    }

    void FunctionThree(unsigned int *ptr)
    {
          long testNumber;
          cout << "Enter an int: ";
          cin >> testNumber;
          // this weird test should be replaced by a series
          // of tests to complain about bad user input
          if (testNumber > 3768 || testNumber < 0)
             throw RangeError(testNumber);
          *ptr = testNumber;
    }
```

5. BUG BUSTERS: What is wrong with the following code?

```
#include "stringc.h"          // our string class

class xOutOfMemory
{
public:
     xOutOfMemory( const String& where ) : location( where ){}
     ~xOutOfMemory(){}
     virtual String where(){ return location };
private:
     String location;
}

main()
{
     try {
          char *var = new char;
          if ( var == 0 )
               throw xOutOfMemory();
     }
     catch( xOutOfMemory& theException )
     {
          cout << "Out of memory at " << theException.location() << "\n";
     }
}
```

In the process of handling an "out of memory" condition, a `string` object is created by the constructor of `xOutOfMemory`. This exception can only be raised when the program is out of memory, so this allocation must fail.

It is possible that trying to create this string will raise the same exception, creating an infinite loop until the program crashes. If this string is really required, you can allocate the space in a static buffer before beginning the program, and then use it as needed when the exception is thrown.

Day 21

Quiz

1. What is the difference between `strcpy()` and `strncpy()`?

`strcpy(char* destination, char* source)` copies `source` to `destination`, and puts a null at the end of `destination`. `destination` must be large enough to accommodate `source`, or `strcpy()` will simply write past the end of the array. `strncpy(char* destination char* source, int howmany)` will write `howmany` bytes of `source` to `destination`, but will not put a terminating null.

2. What does `ctime()` do?

`ctime()` takes a `time_t` variable and returns an ASCII string with the current time. The `time_t` variable is typically filled by passing its address to `time()`.

3. What is the function to call to turn an ASCII string into a `long`?

```
atol()
```

4. What does the complement operator do?

It flips every bit in a number.

5. What is the difference between `OR` and exclusive `OR`?

`OR` returns `TRUE` if either or both bits are set; exclusive `OR` returns `TRUE` only if one, but not both, is set.

6. What is the difference between `&` and `&&`?

`&` is the bitwise `AND` operator, and `&&` is the logical `AND` operator.

7. What is the difference between | and `||`?

| is the bitwise `OR` operator, and `||` is the logical `OR` operator.

Exercises

1. Write a program to safely copy the contents of a 20-byte string into a 10-byte string, truncating whatever won't fit.

```
#include <iostream.h>
#include <string.h>

int main()
{
   char bigString[21] = "12345678901234567890";
   char smallString[10];
   strncpy(smallString,bigString,9);
   smallString[9]='\0';
   cout << "BigString: " << bigString << endl;
   cout << "smallString: " << smallString << endl;
   return 0;
}
```

2. Write a program that tells the current date in the form `7/28/94`.

```
#include <iostream.h>
#include <time.h>

int main()
{
   time_t currentTime;
7:        struct tm *timeStruct;
8:        time (&currentTime);
9:        timeStruct = localtime(&currentTime);
10:
11:       cout << timeStruct->tm_mon+1 << "/";
12:       cout << timeStruct->tm_mday << "/";
13:       cout << timeStruct->tm_year << " ";
14:       return 0;
15:    }
```

3. Write the definition of a class that uses bit fields to store whether the computer is monochrome or color, a PC or Macintosh, a laptop or a desktop, and whether it has a CD-ROM.

```
#include <iostream.h>
enum Boolean { FALSE = 0, TRUE = 1 };

class Computer
{
public:  // types
     enum Machine { Mac = 0, PC };

public:  // methods
     Computer( Boolean color, Boolean laptop, Machine kind, Boolean cdrom )
          : Color( color ), Laptop( laptop ), Kind( kind ), CDRom( cdrom ){}
     ~Computer(){}

     friend ostream& operator<<( ostream& os, const Computer& computer );

private:
     Boolean Color : 1;
     Boolean Laptop : 1;
     Machine Kind : 1;
     Boolean CDRom : 1;
};

     ostream&
operator<<( ostream& os, const Computer& computer )
{
     os << "[";
     ( computer.Color ) ? os << "color" : os << "monochrome";
     os << ", ";
     ( computer.Laptop ) ? os << "laptop" : os << "desktop";
     os << ", ";
     ( computer.Kind ) ? os << "PC" : os << "Mac";
     os << ", ";
     ( computer.CDRom ) ? os << "" : os << "no ";
     os << "CD-Rom";
     os << "]";
     return os;
}
 int main()
{
     Computer pc( TRUE, TRUE, Computer :: PC, TRUE );

     cout << pc << `\n';
     return 0;
}
```

4. Write a program that creates a 26-bit mask. Prompt the user to enter a word, and then quickly report on which letters were used in setting and reading the bits (one bit per character). The program should treat upper- and lowercase letters as the same.

```
#include <ctype.h>
#include <iostream.h>
#include <string.h>
```

```
class Bits
{
public:
     enum { BITS_PER_INT = 16 };
     Bits( int cnt );
     virtual ~Bits();

     void clear();
     void set( int position );
     void reset( int position );
     int is_set( int position );
private:
     unsigned int * bits;
     int count;
     int Ints_Needed;
};

class AlphaBits : private Bits
{
public:
     AlphaBits() : Bits( 26 ){}
     ~AlphaBits(){}

     void clear() { Bits::clear(); }
     void set( char );
     void reset( char );
     int is_set( char );
};

Bits :: Bits( int cnt ) : count( cnt )
{
     Ints_Needed = count / BITS_PER_INT;

     // if there is a remainder, you need one more member in array
     if ( 0 != count % BITS_PER_INT )
          Ints_Needed++;

     // create an array of ints to hold all the bits
     bits = new unsigned int[ Ints_Needed ];

     clear();
}

Bits :: ~Bits()
{
     delete [] bits;
}

void Bits :: clear()
{
     // clear the bits
     for ( int i = 0; i < Ints_Needed; i++ )
          bits[ i ] = 0;
}

void Bits :: set( int position )
{
     // find the bit to set
     int Int_Number = position / BITS_PER_INT;
     int Bit_Number = position % BITS_PER_INT;

     // create mask with that one bit set
     unsigned int mask = 1 << Bit_Number;

     // set the bit
     bits[ Int_Number ] |= mask;
```

```
}

// clear the bit
void Bits :: reset( int position )
{
     int Int_Number = position / BITS_PER_INT;
     int Bit_Number = position % BITS_PER_INT;

     unsigned int mask = ~( 1 << Bit_Number );

     bits[ Int_Number ] &= mask;
}

int Bits :: is_set( int position )
{
     int Int_Number = position / BITS_PER_INT;
     int Bit_Number = position % BITS_PER_INT;

     unsigned int mask = 1 << Bit_Number;

     return ( 0 != ( bits[ Int_Number ] & mask ) );
}

void AlphaBits :: set( char s )
{

     // make sure the requested character is an alphabetic character
     // if so, force it to lower case, then subtract the ascii value
     // of `a' to get its ordinal (where a = 0, b =1) and set that bit
     if ( isalpha( s ) )
          Bits :: set( tolower( s ) - `a' );
}

void AlphaBits :: reset( char s )
{
     if ( isalpha( s ) )
          Bits :: reset( tolower( s ) - `a' );
}

int AlphaBits :: is_set( char s )
{
     if ( isalpha( s ) )
          return Bits :: is_set( tolower( s ) - `a' );
     else
          return 0;
}

int main()
{
     AlphaBits letters;

     char buffer[512];

     for (;;)
     {
          cout << "\nPlease type a word (0 to quit): ";
          cin >> buffer;

          if (strcmp(buffer,"0") == 0)
             break;

          // set the bits
          for ( char *s = buffer; *s; s++ )
               letters.set( *s );

          // print the results
          cout << "The letters used were: ";
          for ( char c = `a'; c <= `z'; c++ )
               if ( letters.is_set( c ) )
                    cout << c << ` `;
          cout << `\n';

          // clear the bits
          letters.clear();
     }
    return 0;
```

5. Write a program that sorts the command-line parameters. If the user enters `SortFunc cat bird fish dog`, the program prints `bird cat dog fish`.

```
#include <string.h>
#include <iostream.h>

void swap ( char* &s, char* &t )
{
     char* temp = s;
     s = t;
     t = temp;
}

int main( int argc, char* argv[] )
{
     // Since argv[0] is the program name,
     //we don't want to sort or print it;
// we start sorting at element 1 (not 0).

     // a "Bubble Sort" is used because of the small number of items.
     int i,j;
     for ( i = 1; i < argc; i++ )
          for ( j = i + 1; j < argc; j++ )
               if ( 0 < strcmp( argv[i], argv[j] ) )
                    swap( argv[i], argv[j] );


     for ( i = 1; i < argc; i++ )
          cout << argv[i] << ` `;

     cout << `\n';
    return 0;
}
```

6. Write a program that adds two numbers without using the addition operator (+), subtraction operator (-), increment (++), or decrement (--). Hint: Use the bit operators!

If you take a look at the addition of two bits, you'll notice the answer will contain two bits: the result bit and the carry bit. Thus, adding 1 and 1 in binary results in 1 with a carry of 1. If we add 101 to 001, here are the results:

```
101  // 5
001  //1
110  //6
```

If you add two "set" bits (each is valued as one), the result is that the result bit is 0 but the carry bit is 1. If you add two clear bits, both the result and the carry are 0. If you add two bits with one set and the other clear, the result bit is 1, but the carry bit is 0. Here is a table that summarizes these rules:

```
lhs   rhs   |   carry   result
------------+------------------
0     0     |  0       0
0     1     |  0       1
```

```
1     0     |   0       1
1     1     |   1       0
```

Examine the logic of the carry bit. If both bits to be added (`lhs` and `rhs`) are 0 or either side is 0, the answer is 0. Only if both bits are 1 is the answer 1. This is exactly the same as the `AND` operator (`&`).

In the same way, the result is an `XOR` (`^`) operation: if either bit is 1 (but not both), the answer is 1; otherwise, the answer is 0.

When you get a carry, it is added to the next most significant (leftmost) bit. This implies either iterating through each bit or recursion.

```
#include <iostream.h>

unsigned int add( unsigned int lhs, unsigned int rhs )
{
     unsigned int result, carry;

     while ( 1 )
     {
          result = lhs ^ rhs;
          carry = lhs & rhs;

          if ( carry == 0 )
               break;

          lhs = carry << 1;
          rhs = result;
     };

     return result;
}

int main()
{
     unsigned long a, b;
     for (;;)
     {
          cout << "Enter two numbers. (0 0 to stop): ";
          cin >> a >> b;
          if (!a && !b)
               break;
          cout <<a << " + " << b << " =  " << add(a,b) << endl;
     }
    return 0;
}
```

Alternatively, you can solve this problem with recursion:

```
#include <iostream.h>

unsigned int add( unsigned int lhs, unsigned int rhs )
{
     unsigned int carry = lhs & rhs;
     unsigned int result = lhs ^ rhs;

     if ( carry )
          return add( result, carry << 1 );
     else
          return result;
}

int main()
```

```
{
     unsigned long a, b;
     for (;;)
     {
          cout << "Enter two numbers. (0 0 to stop): ";
          cin >> a >> b;
          if (!a && !b)
               break;
          cout <<a << " + " << b << " =  " << add(a,b) << endl;
     }
    return 0;
}
```