UNIT 16

Lab Exercises (Answers)

1. The new class and its implementation:

FILE: **Term_Buffer.h**

```
#include "Buffer.h"

class Terminal_Buffer : public Buffer {
public:
     Terminal_Buffer(int size);
     ~Terminal_Buffer();

private:
     void flush();
};
```

FILE: **term_buffer.c**

```
#include "Term_Buffer.h"
#include <iostream.h>

Terminal_Buffer::Terminal_Buffer(int size)
                 : Buffer(size)
{
}

Terminal_Buffer::~Terminal_Buffer()
{
     if (contains() > 0) flush();
}

void Terminal_Buffer::flush()
{
     int i;
     for (i = 0; i < contains(); i++)
     {
         cerr << get(i);
     }
}
```

UNIT 16

Lab Exercises

1. Change to the *unit16/buffer* directory. Create the class **Terminal_Buffer**, that buffers characters to be output to the terminal. Derive this class from class **Buffer**. Declare the class in the file *Term_Buffer.h*, and write the member functions in *term_buffer.c*.

   When a **Term_Buffer** is full, its data are written to the **ostream cerr** (you may assume **cerr** will output to the terminal). When a **Terminal_Buffer** is created, the size must be given, but no file name is needed. Test the class with the test program in the file *term_test.c*.

   You can compile and execute the test program by entering '**make prob1**' or '**make term_test**' or you can compile and execute it directly using the commands:

```
$ CC term_test.c term_buffer.c buf_prot.c \
     buf_public.c -o term_test
$ term_test
```

SUMMARY	
DIRECTORY	unit16/buffer
DECLARATION	Buffer.h, Term_Buffer.h (new)
IMPLEMENTATION	buf_prot.c buf_public.c, term_buffer.c (new)
TEST PROGRAM	term_test.c

FILE: **term_test.c**

```
#include "Term_Buffer.h"
#include <iostream.h>
#include <libc.h>

main(int, char *[])
{
     Terminal_Buffer test(7);
     char ch;

     for (ch = 'a' ; ch <= 'z' ; ch++) {
                test.add(ch);
                sleep(1);
          }

     return 0;
}
```

Lab Exercises

Summary

A derived class can

- access **protected** members of base
- provide an implementation for base

All users of a class should be aware of

- the class's public members, and friend functions

Users creating derived classes should also be aware of

- the class's protected members
- the class's private virtual functions

Summary

Using a Buffer

```
#include "File_Buffer.h"
#include <iostream.h>
#include <libc.h>

void do_test()
{
    File_Buffer test("test_file", 5);
    char ch;

    cout << "initially: \n";
    system("ls -l test_file");

    for (ch = 'a' ; ch <= 'e' ; ch++)
      test.add(ch);

    cout << "after 5 characters: \n";
    system("ls -l test_file");

    test.add('f');
    cout << "after 6 characters: \n";
    system("ls -l test_file");
}

main(int, char *[])
{
    do_test();
    cout << "after do_test:\n";
    system("ls -l test_file");

    return 0;
}
```

Using a Buffer

If we create a 5 character buffer, and add up to five characters, no data will be written to the disk. As soon as we add a sixth character, the original contents of the buffer are written to the disk, so that the sixth character can be put into the buffer. When the buffer is destroyed, the remaining data are flushed to the disk.

File_Buffer::flush

```
#include "File_Buffer.h"

File_Buffer::File_Buffer(const char *name, int size)
          : Buffer(size), str(name)
{
}

File_Buffer::~File_Buffer()
{
    if (contains() > 0) flush();
}

void File_Buffer::flush()
{
    int i;
    for (i = 0; i < contains(); i++)
    {
      str << get(i);
    }
    str.flush();
}
```

File_Buffer::flush

The `File_Buffer::flush` function uses the `contains` and `get` functions from the base class to extract the data from the buffer, and puts the data into the file with the "<<" operator.

Class File_Buffer

```
#include "Buffer.h"
#include <fstream.h>

class File_Buffer : public Buffer {
public:
    File_Buffer(const char *name, int size);
    ~File_Buffer();

private:
    void flush();
    ofstream str;
};
```

Class File_Buffer

Class `File_Buffer` provides an overriding `flush` function, to write out characters into a file. If this class were written as part of an operating system, it would need private data to keep track of what part of the disk it should use, but since we are constructing a simplified example, we'll just declare an output file stream.

The `File_Buffer` constructor will initialize the output file stream, and the destructor will ensure that any data remaining in the `File_Buffer` is written out before the buffer is destroyed.

Protected Member Functions

```
#include "Buffer.h"

int Buffer::contains()
{
    return _contains;
}

char Buffer::get(int index)
{
    return data[index];
}
```

Protected Member Functions

The `contains` and `get` functions simply return the appropriate information. Since they are listed in the protected section of the class, they can be used by the derived class member functions, or the member functions of class `Buffer`, but not by other functions.

Class Buffer's Protected Members

```
class Buffer {
public:
    Buffer(int size);
    virtual ~Buffer();

    void add(char);

protected:
    int contains();
    char get(int index);

private:
    virtual void flush() = 0;
    // flush() copies all characters
    // from the buffer to the device

    int _size_limit;
    char *data;
    // data must always point to
    // an array of _size_limit chars

    int _contains;
    // _contains gives the number of
    // characters in the buffer
};
```

Class Buffer's Protected Members

Class `Buffer` provides two functions for use by the derived class `flush` function: `contains` and `get`. `contains` returns the number of characters contained in the `Buffer`, and `get` gives the derived class functions access to the characters.

Protected Members

- **public** members
  - — accessible from any function
  - — provides the class's interface for users
- **protected** members
  - — accessible from class and derived classes
  - — interface for derived classes
- **private** members
  - — only accessible from the class itself
  - — describe the representation

Protected Members

In situations where special features must be provided for derived classes, the base class can declare **protected** members. A protected member can only be accessed by the defining operations of the class itself and the derived classes.

Overriding flush

The derived class `flush` functions

- obey rules in base class
- must take characters from buffer

How can they read the characters?

- no access to private data
- no public functions provided

Overriding flush

The `flush` functions in the derived classes must obey the rules that our base class gives in its comments about the flush function. They must therefore take the characters from the buffer, and write them into whatever device they work with.

The problem we face now is: How can the derived class functions access the data in the base class? They do not have direct access to the base class data, and there are no public member functions that return the characters.

Buffer Member Functions

```
#include "Buffer.h"
#include <iostream.h>
#include <libc.h>

Buffer::Buffer(int size)
{
    data = new char[size];
    if (data == 0) {
      cerr << "Couldn't allocate space.\n";
      exit(1);
    }
    _size_limit = size;
    _contains = 0;
}

Buffer::~Buffer()
{
    delete data;
}

void Buffer::add(char ch)
{
    if (_contains == _size_limit) {
      flush();
      _contains = 0;
    }
    data[_contains++] = ch;
}
```

Buffer Member Functions

The `Buffer` constructor allocates an array to hold the characters, and sets the `_size_limit` and `_contains` members. Therefore, when a `Buffer` is created, it will obey all the rules stated in the private section.

The `Buffer` destructor frees the storage for the buffer, so that all the memory for a `Buffer` object will be freed when a `Buffer` is destroyed.

The `add` member function checks to see if the `Buffer` is full before it adds the character. If the `Buffer` is full, `add` calls the `flush` function to empty it. Even though the code for the `flush` has not been written yet, we are relying on the fact that it should follow the rules listed below its declaration in the base class: it must write out the characters in the `Buffer`.

class Buffer

```
class Buffer {
public:
    Buffer(int size);
    virtual ~Buffer();

    void add(char);


private:
    virtual void flush() = 0;
    // flush() copies all characters
    // from the buffer to the device

    int _size_limit;
    char *data;
    // data must always point to
    // an array of _size_limit chars

    int _contains;
    // _contains gives the number of
    // characters in the buffer
};
```

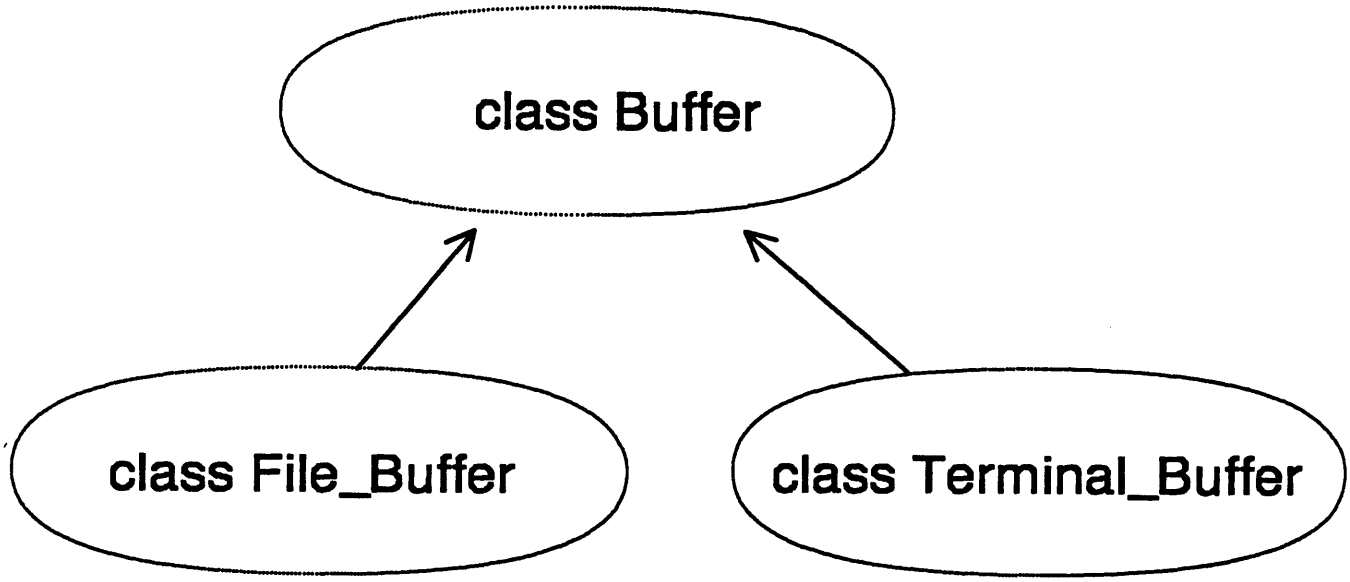
class Buffer

The abstract class `Buffer` provides a constructor, a destructor, and one additional member function for the users. The constructor and destructor will allocate and free storage to hold the characters in the buffer, and the `add` function will add data into the buffer. When the buffer is full, the `add` function will automatically call `flush`. In this example, flushing is handled automatically, so the `flush` function is not listed in the public section. These buffer classes will be used only for output, so there are no member functions for reading data in from the buffer.

`flush` is a pure virtual function, because we can not write a `flush` function for the base class. When we derive the class `File_Buffer`, we will provide the `flush` member function.

The member named `data` must always point to an array of at least `_size_limit` characters, and the member `_contains` must count the number of characters in the buffer. Even though we have not written a `flush` function yet, we are stating exactly what it must do in the comment that follows it. We will expect the `flush` functions in the various derived classes to obey this rule.

Example Classes



Example Classes

In this unit, we will see that a derived class can access its base class in ways that other code can not. Our example will focus on a group of classes that represent different kinds of buffers in an operating system. A `File_Buffer` holds characters that are to be written into a disk file. A `Terminal_Buffer` holds characters that will be output to a terminal. If new devices are added to the system, new classes can be added to handle their output.

To avoid getting distracted by the details of writing operating system code, we will simply implement the class `File_Buffer` by using the stream I/O library.

Objectives

At the end of this unit we will be able to:

- Describe the rules for access of a derived class to its base
- Create and use protected member functions
- Override a private function from the base

CONTENTS

Unit 16

Object-Oriented Programming in C++

Derived Class Access