Chapter - 21
Advanced Classes
Derived classes

Defining a bounds checking stack:

```cpp
public:

};
```
Bound check stack (cont.)

```cpp
if (count > STACSIZE) {
    stck::push(item);
    return(stck::pop(viod));
}
```

```cpp
}
```
Derived Classes are like the base classes only with something extra
Derived classes can be used anywhere you can use a base class

```cpp
void push_things(stack &a_stack) {
    a_stack.push(1);
    a_stack.push(2);
}

// ...
b_stack bounded_stack; // A random stack
// ....
push_things(bounded_stack);
```

<table>
<thead>
<tr>
<th>Stack</th>
<th>Stack (now invisible)</th>
</tr>
</thead>
<tbody>
<tr>
<td>b_stack</td>
<td>bounded_stack</td>
</tr>
</tbody>
</table>

Stack "peephole"
Dynamically Allocated stack

class stack {
  private:

  stack
    protected:

  public:
    stack(const unsigned int size) {
      data = new int[size];
      count = 0;
    }
    virtual ~stack(void) {
      delete data;
      data = NULL;
    }

    // ...

   .Usage:
    stack big_stack(1000);
    stack small_stack(10);
    stack bad_stack; // Illegal, size required
Derived Class

We have Derived class. How do we call the parameterized constructor in the base class?

class b_stack: public stack {
    private:
        // Size of the simple stack

    public:
        b_stack(const unsigned int size) :
            }

Protections

class a {
}

class b {
}

class c : public a, private b {

public:
    void function(void) {
        // Legal or Illegal?
        a_private = 1;
        a_protected = 1;
        a_public = 1;

        b_private = 1;
        b_protected = 1;
        b_public = 1;
    }

}

main() {
    class c c_var;

    c_var.a_private = 1;
    c_var.a_protected = 1;
    c_var.a_public = 1;

    c_var.b_private = 1;
    c_var.b_protected = 1;
    c_var.b_public = 1;

    c_var.c_private = 1;
    c_var.c_protected = 1;
    c_var.c_public = 1;
}
Sending mail the hard way

Let’s define a class to mail a letter:

class mail {
    public:
        address sender; // Who's sending the mail
                        // (return address)
        address receiver; // Who's getting the mail

        // Send the letter
        void send_it(void) {
            ... Some magic happens here
        }
};

void mail::send_it(void) {
    switch (service) {
        case POST_OFFICE:
            put_in_local_mail_box();
            break;

        case FEDERAL_EXPRESS:
            fill_out_waybill();
            call_federal_for_pickup();
            break;

        case UPS:
            put_out_ups_yes_sign();
            give_package_to_driver();
            break;

        //... and so on for every service in the universe
    }
}
Simple post_office class

class post_office: public mail{
    public:
        // Send the letter
        void send_it(void) {
            put_in_local_mail_box();
        }
        // Cost returns cost of sending a letter in cents
        int cost(void) {
            // Costs 32 cents to mail a letter
            // WARNING: This can easily become dated
            return (32);
        }
    }

    Example:
    void get_address_and_send(mail &letter){
        letter.from = my_address.
        letter.to = get_to_address();
        letter.send_it();
    }
    //...
    class post_office simple_letter;
    get_address_and_send(simple_letter);

    Nice idea, but it doesn’t work
**virtual functions**

The keyword `virtual` tells C++ “Look for the function in the Derived class first”

<table>
<thead>
<tr>
<th>Class Type</th>
<th>Member Function type</th>
<th>Search order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derived</td>
<td>Normal</td>
<td>Derived-&gt;Base</td>
</tr>
<tr>
<td>Base</td>
<td>Normal</td>
<td>Base</td>
</tr>
<tr>
<td>Base</td>
<td>virtual</td>
<td>Derived-&gt;Base</td>
</tr>
</tbody>
</table>
virtual usage

public:

};

public:

};

}
{  
  a_base.a();  
  a_base.b();  
  a_base.c();  
}

int main()
{
  a_derived.a();  
  a_derived.b();  
  a_derived.c();  
  do_base(a-derived);  
}
Virtual class mail

class mail {
    public:
        address sender; // Who is sending the mail
        address receiver; // Who is getting the mail

    // Send the letter
    virtual void send_it(void) {
        std::cout << "Error: send_it not defined" << 
            " in derived class.\n"
        exit (8);
    }

    // Cost of sending a letter in pennies
    virtual int cost(void) {
        std::cout << "Error: cost not defined " << 
            " in derived class.\n"
        exit (8);
    }
};
Post Office Derivation

class post_office: public mail {
    public:
        void send_it(void) {
            put_letter_in_box();
        }
        int cost(void) {
            return (29);
        }
};
Abstract mail class

class mail {
    public:
        address sender; // Who is sending the mail
        // (return address)

        // Who is getting the mail
        address receiver;

        // Send the letter
        virtual void send_it(void) = 0;

        // Cost of sending a letter in pennies
        virtual int cost(void) = 0;
};
Two room repair shop

class room { ... };
class garage: public room { ... };
class office: public room { ... };
class repair_shop: public garage, office { .... }

Diagram:

- room
  - garage
  - repair_shop
- room
  - office
  - repair_shop
class room { ... };
class garage: virtual public room { ... };
class office: virtual public room { ... };
class repair_shop: public garage, office { .... }

One room repair shop
Function Hiding in Derived Classes

public:

};

public:

};

int main() {

// not defined in
// the class "derived"
Constructors, Destructors, Derived Classes

Constructor order: Base class, Derived Class
Destruction order: Derived Class, Base class

If the destructor of a base class is not declared virtual, then deleting a pointer to the base class will cause C++ to skip the calling of the

When in doubt, declare the destructor virtual.
Question:

Why does the following program fail when we delete the variable list_ptr? The program seems to get upset when it tries to call clear at line 17.
Question (continued)