Chapter - 21 Advanced Classes

Derived classes

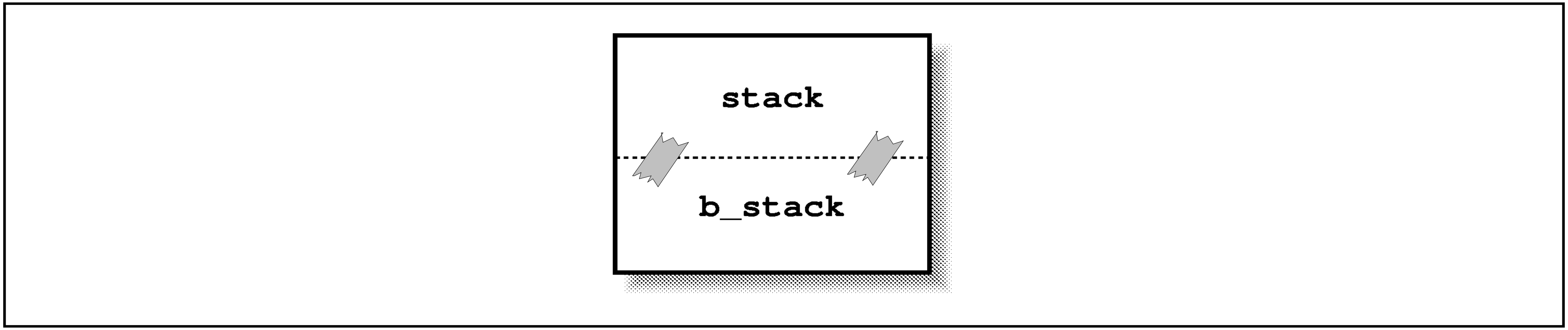
Defining a bounds checking stack:

```
class b_stack: public stack {
    public:
        // b_stack -- default constructor
        // ~b_stack -- default destructor
        // copy constructor defaults

        // Push an item on the stack
        void push(const int item);

        // Remove an item from the stack
        int pop(void);
};
```

Bound check stack (cont.)

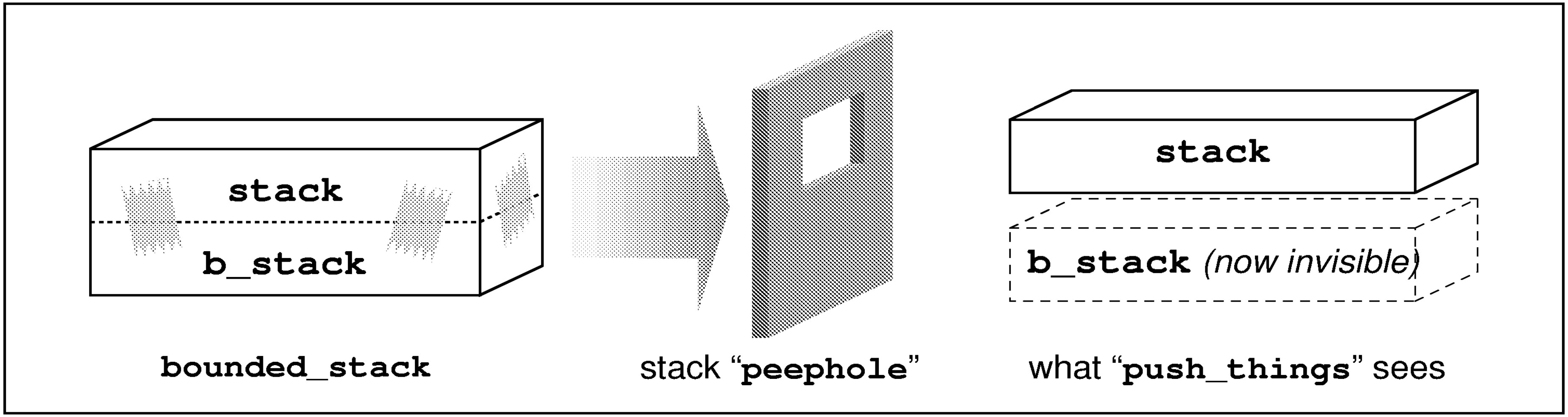
```
inline void b_stack::push(const int item) {
    if (count >= STACK_SIZE) {
        std::cerr <<
             "Error: Push overflows stack\n";
        exit (8);
    }
    stack::push(item);
}

inline int b_stack::pop(void) {
    if (count <= 0) {
        std::cerr <<
             "Error: Pop causes stack underflow\n";
        exit (8);
    }
    return (stack::pop());
}
```

Derived Classes are like the base classes only with something extra



Derived classes can be used anywhere you can use a base class

```
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);
```



Dynamically Allocated stack

```
class stack {
    private:
        int *data;    // Pointer to the data in the stack
    protected:
        int count;    // Current item on the stack
    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
        const unsigned int stack_size;

    public:
        b_stack(const unsigned int size) :
                stack(size), stack_size(size) {
        }
```

Protections

```
class a {
    private:        int a_private;
    protected:      int a_protected;
    public:         int a_public;
};

class b {
    private:        int b_private;
    protected:      int b_protected;
    public:         int b_public;
};

class c : public a, private b {
    private:        int c_private;
    protected:      int c_protected;
    public:         int c_public;

    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”

Class Type	**Member Function type**	**Search order**
Derived	Normal	Derived->Base
Base	Normal	Base
Base	virtual	Derived->Base

virtual usage

```
#include <iostream>
class base {
    public:
        void a(void) {
            std::cout << "base::a called\n";
        }
        virtual void b(void) {
            std::cout << "base::b called\n";
        }
        virtual void c(void) {
            std::cout << "base::c called\n";
        }
};
class derived: public base {
    public:
        void a(void) {
            std::cout << "derived::a called\n";
        }
        void b(void) {
            std::cout << "derived::b called\n";
        }
};
```

```
void do_base(base &a_base)
{
    std::cout <<
        "Call functions in the base class\n";
    a_base.a();
    a_base.b();
    a_base.c();
}
int main()
{
    derived a_derived;

    std::cout <<
        "Calling functions in the derived class\n";

    a_derived.a();
    a_derived.b();
    a_derived.c();

    do_base(a_derived);
    return (0);
}
```

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

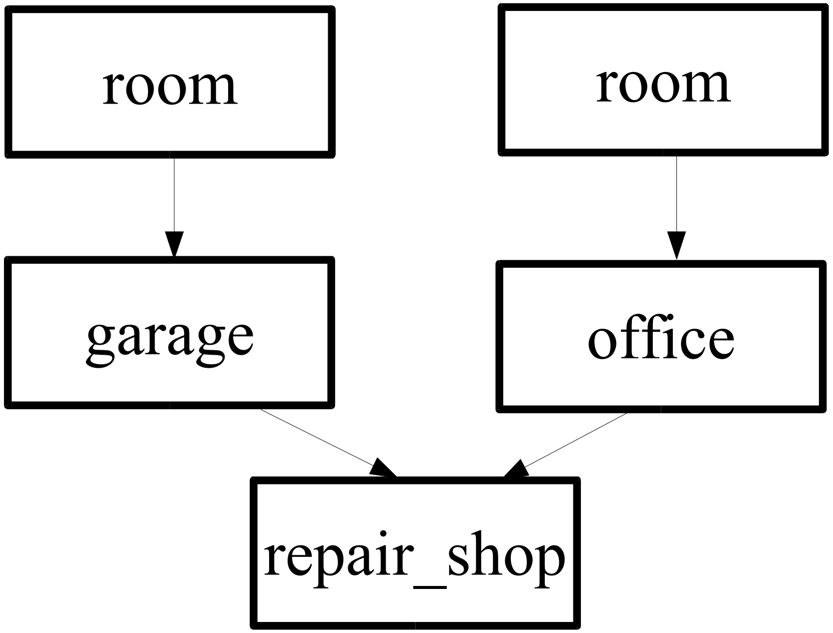
```
class post_office: public mail {
    public:
        void send_it(void) {
            put_letter_in_box();
        }
        int cost(void) {
            return (29);
        }
};
```

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;
};
```

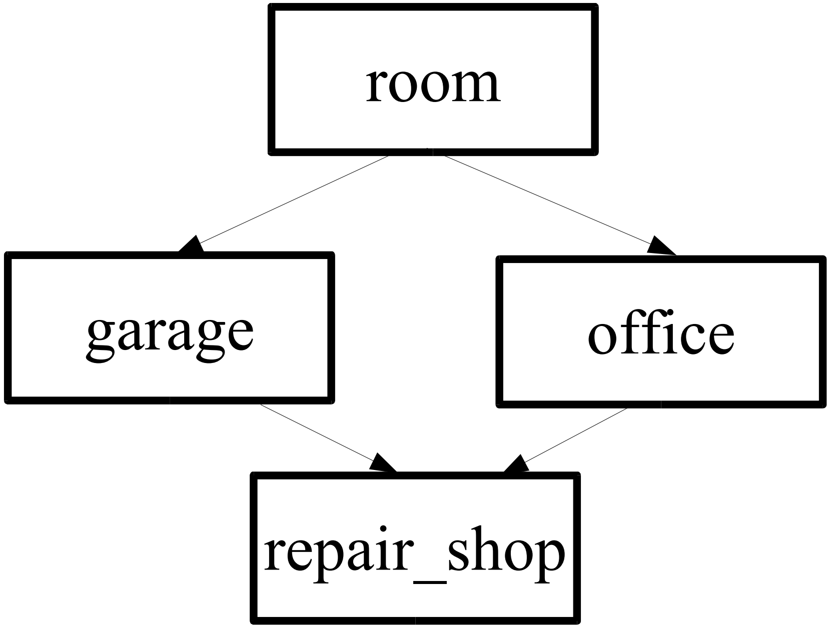
Two room repair shop

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



One room repair shop

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



Function Hiding in Derived Classes

```
class simple {
    public:
        int do_it(int i, int j) { return (i*j); }
        float do_it(float f) { return (f*2);}
};
class derived: public simple {
    public:
        int do_it(int i, int j) { return (i+j); }
};


int main() {
    derived test;       // Define a class for our testing
    int i;              // Test variable
    float f;            // Test variable


    i = test.do_it(1, 3);   // Legal, returns 4 (1+3)
    f = test.do_it(4.0);    // Illegal "do_it(float)"
                            // 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 Derived class's destruc-tor.

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.

```
 1 #include <iostream>
 2 #include <stdlib.h>
 4 class list {
 5     private:
 6      int item;       // Current item number
 8     public:
 9      virtual void clear() = 0;
11      void next_item(void) {
12          ++item;
13      }
15      list(void) {
16          item = 0;
17      }
19      virtual ~list() {
20         clear();
21      }
22 };
```

Question (continued)

```
class list_of_integers : public list {
    public:
        int array[100];   // Place to store the items
     void clear(void) {
         int i;      // Array index
         for (i = 0; i < 100; ++i)
             array[i] = 0;
     }
};
int main()
{
    list_of_integers *list_ptr = new list_of_integers;

    // Cause problems
    delete list_ptr;
    list_ptr = NULL;
    return (0);
}
```