Chapter - 13
Simple Classes
Stack Definition

Data:

A place to store the items put on and taken from the stack. (Implemented as an array).

Obvious operations:

Push -- Add an element to the top of the stack (other elements are pushed down).
Pop -- Remove the top element from the stack (other elements are popped up).

Hidden operations:

Construction -- The creation and initialization of the stack
Destruction -- The clean up done when the stack is destroyed.
Stack Implementation as Struct

/

*******************************************************/

*******************************************************/

};

};//***********************************************************

};//***********************************************************
Stack Program (cont.)

/**************************************************

/**************************************************

{ 
  ++the_stack.count;
} 
/**********************************************************

/**********************************************************

{ 
  --the_stack.count;
}
Using the Stack

```cpp
main()
{

    stack_init(a_stack);
}
```
Stack as a Class

private:

public:

};
Stack member functions

```cpp
{
}

{
    ++count;
}

{
    --count;
}
```
Using a class

Declaring a class variable (called an instance of a class):

```cpp
class stack a_stack;  // Stack we want to use
```

or more commonly:

```cpp
stack a_stack;  // Stack we want to use
```

Calling member functions:

```cpp
a_stack.init();
a_stack.push(1);
result = a_stack.pop();
```
Constructor

A constructor is called when a variable is created. The member function for the constructor is the same as the class’s name.

```cpp
// ...
public:
    stack(void);
    // ...
};
{
}
main()
{
    // Calls stack::stack()
```
Destructor

A destructor is called when a variable is destroyed (goes out of scope). The member function for the destructor is named the same as the class with a tilde (~) in front of it.

```cpp
class stack {
    // ... 

    public:
    ~stack() {
        if (count != 0) {
            std::cerr <<
                "Error: Destroying a non-empty stack\n";
        }
    }

    // other destructor methods ...
};
```
Parametrized Constructors

class person {
    public:

        // ... 
    public:
        person(const std::string& i_name,
                const std::string& i_phone);
        // ... rest of class
    }

    person::person(const std::string& i_name,
                    const std::string& i_phone)
    {
        name = i_name;
        phone = i_phone;
    }

    main()
    {
        person sam("Sam Jones", "555-1234");
        person sam; // Illegal
Overloaded Constructors

class person {
    public:

        // ..... 
        public:
            person(const std::string& i_name, 
                    const std::string& i_phone); 
            person(const std::string& i_name); 
        // ... rest of class 
    };
    person::person(const std::string& i_name) 
    { 
        name = i_name; 
        phone = "No Phone";
    }

main()
{
    person sam("Sam Jones", "555-1212");
    person john("John Smith");
Parameterized Destructors

No such thing.
Copy Constructor

```cpp
stack::stack(const stack &old_stack) {
    for (i = 0; i < old_stack.count; ++i) {
        data[i] = old_stack.data[i];
        count = old_stack.count;
    }
}

main()
{
    stack old_stack;
    
    old_stack.push(1);
    old_stack.push(2);
    
    stack new_stack(old_stack);
}
```
Hidden Member Function Calls

```cpp
void use_stack(stack local_stack) {
    local_stack.push(9);
    local_stack.push(10);
    .. Do something with local_stack
}

main() {

    a_stack.push(1);
    a_stack.push(2);

    use_stack(a_stack);

    // Prints "2"
    std::cout << a_stack.pop() << '\n';
```
Automatically Generated Member Functions

class::class()
    Default constructor

class::class(const class &old_class)
    Copy constructor

class::~class()    
    Destructor

class class::operator = (const class &old_class)
    Assignment operator.
Shortcuts

class stack {
    public:
        // .... rest of class
        // Push an item on the stack
    void push(const int item) {
        data[count] = item;
        ++count;
    }
};
Class Style

• Use the “short form” only for very short functions who’s purpose is obvious.
• Use the “short form” only if you can use it and keep the structure of the class clear and easy to understand.
• Remember the “big 4”. These four member functions should be explicitly supplied, else include or a comment indicating that you are using the default.

Big 4:
  1. Default constructor
  2. Destructor
  3. Copy constructor
  4. Assignment operator
Style Example

// Comments describing the class
class queue {
    private:

    public:

        // queue(const queue &old_queue)

        // queue operator = (const queue &old_queue)

        // ~queue()

        void put(int item);// Put an item in the queue

};
Classes that can't be copied

If you want a class that does not contain a copy constructor, you can’t just leave the constructor out. C++ will generate a default.

The trick is to declare the constructor private:

```cpp
class no_copy {
    // Body of the class
    private:
    // There is no copy constructor
    no_copy(const no_copy &old_class);
};
```