Programming Abstraction in C++

Eric S. Roberts and Julie Zelenski

Stanford University
2010
Chapter 9. Classes and Objects
Outline

1. Introduction
2. Implementing stack
3. Implementing Scanner
Outline

1. Introduction
2. Implementing stack
3. Implementing Scanner
Defining a class

Defining a `Point` class

class Point {
public:
    int x, y;
};

Similar to defining a structure

struct pointT {
    int x, y;
};
Hiding implementation

Normally, we want to hide (protect) or restrict the use of the instance variables from clients by exporting getters (accessors). Thus we hide the implementation details (fields \( x \) and \( y \)) and future changes in implementation will not affect user programs.

class Point {
public:
    int getX();
    int getY();
private:
    int x, y;
};
Hiding implementation

Making private instance variables and methods to hide implementation details for simplicity, flexibility, and security.
Hiding implementation

Making private instance variables and methods to hide implementation details for simplicity, flexibility, and security.

Getters (accessors) and setters (mutators).

Immutable classes, impossible to change the values of any instance variables after an object has been constructed, for example, Rational and many more.
class Point { 
public:
    Point(int xc, int yc);
    ~Point();
    int getX();
    int getY();
private:
    int x, y;
};
class Point {
public:
    Point(int xc, int yc);
    ~Point();
    int getX();
    int getY();
private:
    int x, y;
};

For now, the destructor does nothing, since there is no need to free memory allocated on the heap.
Constructor

```cpp
Point::Point(int xc, int yc) {
    x = xc;
    y = yc;
}
```
Constructor initializes instance variables.

```cpp
Point::Point(int xc, int yc) {
    x = xc;
    y = yc;
}
```
**Constructor (cont.)**

Point \( pt(1, 2) \)

<table>
<thead>
<tr>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>yc</td>
</tr>
<tr>
<td>xc</td>
</tr>
<tr>
<td>this</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>x</td>
</tr>
<tr>
<td>pt</td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FFA8</td>
</tr>
<tr>
<td>FFC0</td>
</tr>
<tr>
<td>FFB0</td>
</tr>
<tr>
<td>FFB4</td>
</tr>
<tr>
<td>FFB8</td>
</tr>
<tr>
<td>FFC0</td>
</tr>
</tbody>
</table>
Point pt(1, 2)

```
yc
xc
this
y
x
pt
```

```
2
1
FFC0
2
1
overhead
```

```
FFA
FFA
FFB
FFB
FFB
FFC
```
An ambiguity

Point::Point(int x, int y) {
    x = x;
    y = y;
}
Constructor (cont.)

An ambiguity

```cpp
Point::Point(int x, int y) {
    x = x;
    y = y;
}
```

Resolve the ambiguity using the keyword `this`

```cpp
Point::Point(int x, int y) {
    this->x = x;
    this->y = y;
}
```
Keyword **this**

```cpp
Point::Point(int x, int y) {
    this->x = x;
    this->y = y;
}
```
## Keyword `this`

```cpp
Point::Point(int x, int y) {
    this->x = x;
    this->y = y;
}
```

### Stack Diagram

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pt</strong></td>
<td><strong>overhead</strong></td>
<td></td>
</tr>
<tr>
<td><strong>this</strong></td>
<td><strong>FFC0</strong></td>
<td></td>
</tr>
<tr>
<td><strong>x</strong></td>
<td><strong>1</strong></td>
<td><strong>FFB</strong></td>
</tr>
<tr>
<td><strong>y</strong></td>
<td><strong>2</strong></td>
<td><strong>FFB</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>FFA</strong></td>
</tr>
</tbody>
</table>

This diagram illustrates the stack storage for a `Point` object creation.
Destructor frees any memory stored within the object that has been allocated on the heap.

For example,

expandable character stack
dynamic allocation.
Outline

1. Introduction
2. Implementing stack
3. Implementing Scanner
Implementing stack

Study CharStack.h, Figure 9-1, p. 320-321

- Documentation.
- Class definition.
- Interface design.
  - Use public methods to hide implementation, object encapsulation.
  - The private part on p. 323 can be replaced by the private part in Figure 9-3, p. 325, along with some changes in the method implementations on p. 326, but the public interface remains the same. Thus user programs are not affected.
  - Make private part “invisible” by including the private data file in the header file (p. 322).
Class definition

charstack.h

class Charstack {

public:

    Charstack(); // usage: Charstack cstk
    ~Charstack();

    int size();
    bool isEmpty(); // usage: cstk.isEmpty()
    void clear();
    void push(char ch);
    char pop();
    char peek();

private:

    ...
};
charstack.h

private:
    static const int MAX_STACK_SIZE = 100;
    char elements[MAX_STACK_SIZE];
    int count;
Implementation (static array)

charstack.h

private:
    static const int MAX_STACK_SIZE = 100;
    char elements[MAX_STACK_SIZE];
    int count;

charstack.cpp

Charstack::Charstack() {
    count = 0;
}

Charstack::~Charstack() {
    /* Empty */
}

When an object is constructed, everything is allocated on (system) stack.
charstack.cpp

void Charstack::push(char ch) {
    if (count == MAX_STACK_SIZE)
        Error("push: Stack is full");
    elements[count++] = ch;
}

char Charstack::pop() {
    if (isEmpty())
        Error("pop: Stack is empty");
    return elements[--count];
}

Note: elements[count - 1] is the top.
Implementation (dynamic array)

private:
    #include "cstkpriv.h"

cstkpriv.h

static const int INITIAL_CAPACITY = 100;
char *elements; // dynamic array
int capacity;
int count;

void expandCapacity();
Implementation (dynamic array)

Constructor and destructor

Charstack::Charstack() {
    elements = new char[INITIAL_CAPACITY];
    capacity = INITIAL_CAPACITY;
    count = 0;
}

Charstack::~Charstack() {
    delete[] elements;
}

Array elements is allocated on heap.
Implementation (dynamic array)

```cpp
void Charstack::expandCapacity() {
    capacity *= 2;
    char *array = new char[capacity];
    for (int i = 0; i < count; i++) {
        array[i] = elements[i];
    }
    delete[] elements;
    elements = array;
}
```

Copy elements to the newly allocated location.
Free up memory.
void Charstack::push(char ch) {
    if (count == capacity)
        expandCapacity();
    elements[count++] = ch;
}
Implementing stack

Three files

- `charstack.h`
  - constructor, destructor
  - public: (public methods)
  - private:
    - `#include "cstkpriv.h"

- `cstkpriv.h` (private instance variables, private function prototypes)

- `charstack.cpp` (implementations)
Object copying

Copying an object that has at least one data member of pointer type, such as the one in cstkpriv.h, Figure 9-3, p. 325.

CharStack first;
first.push('A');
first.push('B');

CharStack second = first;
first.push('C');
second.push('Z');

cout << first.pop() << endl;
Object copying

Copying an object that has at least one data member of pointer type, such as the one in `cstkpriv.h`, Figure 9-3, p. 325.

```cpp
CharStack first;
first.push('A);
first.push('B');

CharStack second = first;
first.push('C');
second.push('Z');

cout << first.pop() << endl;
```

What is the output?
Object copying (cont.)

CharStack first;
first.push('A');
first.push('B');

CharStack second = first;
...
Object copying (cont.)

CharStack first;
first.push('A');
first.push('B');

CharStack second = first;
...

<table>
<thead>
<tr>
<th>heap</th>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>'A'</td>
<td>ch</td>
</tr>
<tr>
<td>1000</td>
<td>'A'</td>
</tr>
<tr>
<td>1001</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>this</td>
</tr>
<tr>
<td>1003</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>count</td>
</tr>
<tr>
<td>1005</td>
<td>capacity</td>
</tr>
<tr>
<td></td>
<td>elements</td>
</tr>
<tr>
<td></td>
<td>first</td>
</tr>
<tr>
<td></td>
<td>overhead</td>
</tr>
</tbody>
</table>
CharStack first;
first.push('A');
first.push('B');

CharStack second = first;
...
CharStack first;
first.push('A');
first.push('B');

CharStack second = first;
...

<table>
<thead>
<tr>
<th>heap</th>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>'A'</td>
<td>count</td>
</tr>
<tr>
<td>'B'</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>capacity</td>
</tr>
<tr>
<td>1002</td>
<td>elements</td>
</tr>
<tr>
<td>1005</td>
<td>second</td>
</tr>
<tr>
<td></td>
<td>count</td>
</tr>
<tr>
<td></td>
<td>capacity</td>
</tr>
<tr>
<td></td>
<td>elements</td>
</tr>
<tr>
<td></td>
<td>first</td>
</tr>
</tbody>
</table>
CharStack first;
first.push('A');
first.push('B');

CharStack second = first;
first.push('C');
...

<table>
<thead>
<tr>
<th>heap</th>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>'A'</td>
<td>'C'</td>
</tr>
<tr>
<td>1000</td>
<td>FFC0</td>
</tr>
<tr>
<td>'B'</td>
<td>this</td>
</tr>
<tr>
<td>1001</td>
<td>FF9</td>
</tr>
<tr>
<td>'C'</td>
<td>count</td>
</tr>
<tr>
<td>1002</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>capacity</td>
</tr>
<tr>
<td></td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>elements</td>
</tr>
<tr>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>second</td>
</tr>
<tr>
<td></td>
<td>overhead</td>
</tr>
<tr>
<td></td>
<td>FF9</td>
</tr>
<tr>
<td></td>
<td>'A'</td>
</tr>
<tr>
<td>1003</td>
<td>FFA</td>
</tr>
<tr>
<td></td>
<td>'B'</td>
</tr>
<tr>
<td>1004</td>
<td>FFA</td>
</tr>
<tr>
<td></td>
<td>'C'</td>
</tr>
<tr>
<td>1005</td>
<td>FFA</td>
</tr>
<tr>
<td></td>
<td>count</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>capacity</td>
</tr>
<tr>
<td></td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>elements</td>
</tr>
<tr>
<td></td>
<td>1000</td>
</tr>
<tr>
<td>first</td>
<td>overhead</td>
</tr>
<tr>
<td>FFB</td>
<td>FFC</td>
</tr>
</tbody>
</table>
Object copying (cont.)

CharStack first;
...

CharStack second = first;
first.push(‘C’);
second.push(‘Z’);
...

```

<table>
<thead>
<tr>
<th></th>
<th>heap</th>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>‘A’</td>
<td>‘Z’</td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td>FF9</td>
</tr>
<tr>
<td></td>
<td>‘B’</td>
<td>this</td>
</tr>
<tr>
<td>1001</td>
<td></td>
<td>FFAC</td>
</tr>
<tr>
<td></td>
<td>‘Z’</td>
<td>count</td>
</tr>
<tr>
<td>1002</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>capacity</td>
</tr>
<tr>
<td>1003</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>elements</td>
</tr>
<tr>
<td>1004</td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>second</td>
</tr>
<tr>
<td>1005</td>
<td></td>
<td>overhead</td>
</tr>
<tr>
<td></td>
<td></td>
<td>count</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>capacity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>elements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>first</td>
</tr>
<tr>
<td></td>
<td></td>
<td>overhead</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FFC</td>
</tr>
</tbody>
</table>
```

1000 elements, capacity 1000
1000 elements, capacity 1000
1000 elements, capacity 1000
1000 elements, capacity 1000
1000 elements, capacity 1000
Use `DISALLOW_COPYING` macro to prevent inadvertent copying: assignment, parameter passing, or function return.

Add

```
DISALLOW_COPYING(CharStack)
```

to the `cstkpriv.h` file.
Outline

1. Introduction
2. Implementing stack
3. Implementing Scanner
Implementing the **Scanner class**

Three files

* scanner.h

Interface, Figure 9-4, p. 329-330

Constructor, destructor

```cpp
public: (methods)
A setter setSpaceOption
A getter getSpaceOption
```

```cpp
private:
#include "scanpriv.h"
```

Hiding implementation.
Implementing the **Scanner** class

scanpriv.h, Figure 9-5, p. 331

Hides the implementation (string)

```c
/* instance variables */
string buffer;
int len;
int cp;               /* index */
spaceOptionT spaceOption; /* space option */

/* private method prototypes */
...
```
Implementing the \texttt{Scanner} class

\texttt{scanner.cpp}, Figure 9-6, p. 332-333

\begin{verbatim}
Scanner::Scanner() {
    buffer = "";
    spaceOption = PreserveSpaces; /* default */
    cp = -1; /* input string not set */
}

void Scanner::setInput(string str) {
    buffer = str;
    len = buffer.length();
    cp = 0;
}
\end{verbatim}
Implementing the Scanner class

scanner.cpp, Figure 9-6, p. 332-333

```cpp
string Scanner::nextToken() {
    if (cp == -1) {
        Error("setInput not called");
    }
    if (spaceOption == IgnoreSpaces) skipSpaces();
    int start = cp;
    if (start >= len) return "";
    if (isalnum(buffer[cp])) {
        int finish = scanToEndOfIndetifier();
        return buffer.substr(start, finish - start + 1);
    }
    cp++;
    return buffer.substr(start, 1);
}
```