Programming Abstraction in C++

Eric S. Roberts and Julie Zelenski

Stanford University
2010

Chapter 3. Libraries and Interfaces

Outline

Outline

Introduction

Clients: Programs that make use of library.

Interface: The boundary between a library and its clients.

An interface provides both a channel of communication and a barrier (hide complex details).

Introduction

Clients: Programs that make use of library.

Interface: The boundary between a library and its clients.

An interface provides both a channel of communication and a barrier (hide complex details).

In C++, an interface is represented by a header file.

Exporting: Putting function prototypes, data type and constant definitions in the interface.

Just as a program implements an algorithm, a header file provides a realization of an interface.

Packages and abstractions

Package: Header file (`.h`), an interface, and its corresponding implementation (`.cpp`).

Abstraction: The conceptual basis of a library.

Example: `iostream` and `simpio`, two different approaches to input operations (powerful and flexible v.s. simple and easy to use).

Good interface design

- Unified. One interface, one theme, one consistent abstraction.
- Simple. Hide as much complexity from the client as possible.
- Sufficient. Enough functionality to meet the needs.
- General. Flexible enough to meet the needs of many different clients
- Stable. Same structure and effect even if the underlying implementation changes.

Good interface design

- Unified. One interface, one theme, one consistent abstraction.
- Simple. Hide as much complexity from the client as possible.
- Sufficient. Enough functionality to meet the needs.
- General. Flexible enough to meet the needs of many different clients
- Stable. Same structure and effect even if the underlying implementation changes.

Extending: Changing an interface without requiring changes to existing programs.

Outline

Random number interface

Figure 3-1, `random.h`, p. 90

interface boilerplate

```
# ifndef _random_h
# define _random_h
   ...
#endif
```

Prevent the compiler from reading the same interface more than once during a single compilation.

Random number interface

Figure 3-1, `random.h`, p. 90

interface boilerplate

```
# ifndef _random_h
# define _random_h
  ...
#endif
```

Prevent the compiler from reading the same interface more than once during a single compilation.

function prototypes

```
int RandomInteger(int low, int high);
double RandomReal(double low, double high);
bool RandomChance(double p);
void Randomize();
```

Implementation

ANSI function

`int rand()`

returns a random integer between 0 and `RAND_MAX` inclusive.

Implementation

ANSI function

```
int rand()
```

returns a random integer between 0 and `RAND_MAX` inclusive.

```
Randomize()
```

hides the implementation detail of initializing a pseudorandom number generator

```
srand(int (time(NULL)));
```

Implementation (cont.)

Figure 3-3, `random.cpp`, p. 97

```
int RandomInteger(int low, int high) {
    double d = double (rand())/(double (RAND_MAX) + 1);
    int k = int (d * (hight - low + 1));
    return low + k;
}
```

1. Normalization. A floating-point number in $[0, 1)$
2. Scaling and truncation. Scale to an integer in $[0, \mathit{high} - \mathit{low}]$
3. Translation. Shift to $[\mathit{low}, \mathit{high}]$

Outline

Strings

Interface

```
#include <string>
```

Domain

All sequences of characters.

Operations

- Initialization with a string literal

  ```
  string str = "Hello";
  ```

- Concatenation

  ```
  string str2 = str + "World";
  ```

- Lexicographical comparison (based on codes)

  ```
  ==, !=, <, >, <=, >=
  ```

Calling member functions

`str.length()`

The object `str` is the receiver (receiving a request to perform an operation).

String methods, Table 3-1, p. 101

Idiom

```
for (i = 0; i < str.length(); i++) {
    ... str[i] ...
}
```

Going through all characters in a string.

C++ and C-style strings

Explicitly convert a C-style string literal into a C++ string using a typecast-like notation:

```
string str = string("Hello");
```

C++ and C-style strings

Explicitly convert a C-style string literal into a C++ string using a typecast-like notation:

```
string str = string("Hello");
```

Convert a C++ string into a C-style string, using the the method `c_str`.

```
string str = "Hello";
char *cstr = str.c_str();
```

Outline

File streams

1. Declare a stream variable

   ```
   ifstream infile;
   ofstream outfile;
   ```

2. Open the file

   `infile.open("fname.txt")`

   The file name must be a string literal or a C-style string.

3. Transfer data from/to the file.

4. Close the file.