Programming Abstraction in C++

Eric S. Roberts and Julie Zelenski

Stanford University
2010.

Chapter 1. An Overview of C++

Chapter 1. An Overview of C++

You should read Chapter 1. We won't teach the basic syntax and constructs. We'll just highlight some of the common programming idioms and C++ characteristics.

Outline

1. Structure of a C++ program
2. Variables, values, and types
3. Statements
4. Functions

Outline

1 Structure of a C++ program

2 Variables, values, and types

3 Statements

4 Functions

Structure of a C++ program

Comments

- Program: Operation of the program as a whole.
- Function: What the function does.

```
/* multiline
 * comments
 */

// single line comments
```

Structure of a C++ program (cont.)

Library inclusions

```
#include "private library"
#include <system library>
```

header files containing definitions.

Structure of a C++ program (cont.)

Library inclusions

```
#include "private library"
#include <system library>
```

header files containing definitions.

constant definitions

Structure of a C++ program (cont.)

Library inclusions

```
#include "private library"
#include <system library>
```

header files containing definitions.

constant definitions

function prototypes

Structure of a C++ program (cont.)

Library inclusions

```
#include "private library"
#include <system library>
```

header files containing definitions.

constant definitions

function prototypes

main

Structure of a C++ program (cont.)

Library inclusions

```
#include "private library"
#include <system library>
```

header files containing definitions.

constant definitions

function prototypes

main

function definitions

Example

Program comments

```
/*
 * File: powertab.cpp
 * ------------------
 * This program generates a table comparing
 * values of the functions n^2 and 2^n.
 */
```

Example

Program comments

```
/*
 * File: powertab.cpp
 * ------------------
 * This program generates a table comparing
 * values of the functions n^2 and 2^n.
 */
```

Library inclusions

```
#include "genlib.h"
#include <iostream>
#include <iomanip>
```

Example (cont.)

section comment

```
/* Constants
 * ---------
 * LOWER_LIMIT -- starting value for the table
 * UPPER_LIMIT -- final value for the table
 */
```

Example (cont.)

section comment

```
/* Constants
 * ---------
 * LOWER_LIMIT -- starting value for the table
 * UPPER_LIMIT -- final value for the table
 */
```

constant definitions

```
const int LOWER_LIMIT = 0;
const int UPPER_LIMIT = 12;
```

Example (cont.)

function prototype

```
/* Private function prototypes */
int RaiseIntToPower(int n, int k);
```

Example (cont.)

main program

```
int main() {
   cout << "    |   2 |    N " << endl;
   cout << "  N |  N  |   2  " << endl;
   cout << "----|-----|------" << endl;
   for (int n = LOWER_LIMIT; n <= UPPER_LIMIT; n++) {
      cout << setw(3) << n << " |";
      cout << setw(4) << RaiseIntToPower(n, 2) << " |";
      cout << setw(5) << RaiseIntToPower(2, n) << endl;
   }
   return 0;
}
```

Example (cont.)

function comments

```
/*
 * Function: RaiseIntToPower
 * Usage: p = RaiseIntToPower(n, k);
 * ---------------------------------
 * This function returns n to the kth power.
 */
```

Example (cont.)

function definition

```
int RaiseIntToPower(int n, int k) {
   int result;

   result = 1;
   for (int i = 0; i < k; i++) {
      result *= n;
   }
   return result;
}
```

Example (cont.)

function definition

```
int RaiseIntToPower(int n, int k) {
   int result;

   result = 1;
   for (int i = 0; i < k; i++) {
      result *= n;
   }
   return result;
}
```

Style: Page 6

Outline

1. Structure of a C++ program
2. Variables, values, and types
3. Statements
4. Functions

Variables and values

Declaration: four properties

- type: (`int i; double x; char c;` ...)
- name: Naming conventions
  - start with a letter or underscore, others are letters, digits, or underscores, no spaces or special characters
  - No reserved keywords (Table 1-1, p. 11)
  - Case sensitive

Variables and values

Declaration: four properties

- type: (`int i; double x; char c;` ...)
- name: Naming conventions
  - start with a letter or underscore, others are letters, digits, or underscores, no spaces or special characters
  - No reserved keywords (Table 1-1, p. 11)
  - Case sensitive

Examples

variables: `totalTime`
functions: `RaiseIntToPower`
constants: `UPPER_LIMIT`

Variables and values

Declaration: four properties (cont.)

- life time: How long a varible persists. The lifetime of a variable declared in a function (local variable) is the time when the function is active
- scope: accessibility. The scope of a local variable extends to the end of the block where it is declared.

Variables and values

Declaration: four properties (cont.)

- life time: How long a varible persists. The lifetime of a variable declared in a function (local variable) is the time when the function is active
- scope: accessibility. The scope of a local variable extends to the end of the block where it is declared.

We rarely, if ever, use global variables (declared outside any function).

Variables and values

Variables must be declared before they are used.

Variables and values

Variables must be declared before they are used.

All values have a type, and every variable has a declared type.

Example: 2 (`int`), 2.0 (`double`)

Variables and values

Variables must be declared before they are used.

All values have a type, and every variable has a declared type.

Example: 2 (`int`), 2.0 (`double`)

Local variable can be declared anywhere with a block of statements.

Example:

```
for (int i = 0; ...) {
    ...
}
```

Data types

Two attributes: Domain and operations.

Data types

Two attributes: Domain and operations.

Atomic types

- integer: `short`, `int`, `long`
- floating-point: `float`, `double`, `long double`
- text: `char` (ASCII code, Table 1-2, p. 14), `string`
- Boolean: `bool`

Operations

- Precedence and associativity (Table 1-4, p. 17).
  Example:
  `7 + 6 / 3 * 2` or `7 + ((6 / 3) * 2)`
  In general, put extra parentheses.

Operations

- Precedence and associativity (Table 1-4, p. 17).
  Example:
  `7 + 6 / 3 * 2` or `7 + ((6 / 3) * 2)`
  In general, put extra parentheses.
- Mixing types (automatic conversion, Table 1-5, p. 18).
  Example: `9 / 4.0`
  Values are promoted to the richer type.

Operations

- Precedence and associativity (Table 1-4, p. 17).
  Example:
  `7 + 6 / 3 * 2` or `7 + ((6 / 3) * 2)`
  In general, put extra parentheses.
- Mixing types (automatic conversion, Table 1-5, p. 18).
  Example: `9 / 4.0`
  Values are promoted to the richer type.
- Type casts: `int num, den; double (num) / den;`

Operations

- Assignments: multiple assignments (`n1 = n2 = 0`).
  It works because an assignment is an expression that has as its result the value assigned.
  shorthand assignments (`x += 2`)

Operations

- Assignments: multiple assignments (`n1 = n2 = 0`).
  It works because an assignment is an expression that has as its result the value assigned.
  shorthand assignments (`x += 2`)
- Increments and decrements (`i++`, `++i`, `j--`)
  Be sure you understand their meanings. (P. 22)

Operations

- Assignments: multiple assignments (`n1 = n2 = 0`).
  It works because an assignment is an expression that has as its result the value assigned.
  shorthand assignments (`x += 2`)
- Increments and decrements (`i++`, `++i`, `j--`)
  Be sure you understand their meanings. (P. 22)
- Boolean
  relational operators: `==`, `!=`, `<`, `>`, `<=`, `>=`
  short-circuit evaluation:
  `if ((y != 0) && (x % y == 0))`
  logical operators: `!`, `&&`, `||`
  bitwise operators: `&`, `|`
  Don't confuse Boolean logic with bitwise operators.

Outline

1 Structure of a C++ program

2 Variables, values, and types

3 Statements

4 Functions

Simple I/O

Simplified I/O

```
#include "simpio.h"
```

Stream manipulators, Table 1-3, p. 16

```
#include <iomanip>
```

Simple I/O

Simplified I/O

```
#include "simpio.h"
```

Stream manipulators, Table 1-3, p. 16

```
#include <iomanip>
```

```
cout << "Enter an integer: " << endl;
int n1 = GetInteger();
cout << "Enter a floating-point: " << endl;
float x = GetReal();
```

Statements

- Simple statements

```
a = b + c;
```

- Compound statements (blocks): indentation (four spaces)

```
{
    y = x;
    x += 1;
}
```

- Terminator `;`

Control statements: `if`

`if` (condition) statement
The test must always be enclosed in parentheses.

`if` (condition) statement `else` statement

```
if (n % 2 == 0) {
    cout << "That number is even." << endl;
} else {
    cout << "That number is odd." << endl;
}
```

Control statements: `if`

`if` (condition) statement
The test must always be enclosed in parentheses.

`if` (condition) statement `else` statement

```
if (n % 2 == 0) {
    cout << "That number is even." << endl;
} else {
    cout << "That number is odd." << endl;
}
```

Any non-zero expression is true

`if (x)` means the same as `if (x != 0)`

Control statements: switch

```
switch (d) {
  case 0: cout << "zero"; break;
  case 1: cout << "one"; break;
  case 2: cout << "two"; break;
  case 3: cout << "three"; break;
  case 4: cout << "four"; break;
  case 5: cout << "five"; break:
  case 6: cout << "six"; break;
  case 7: cout << "seven"; break;
  case 8: cout << "eight"; break;
  case 9: cout << "nine"; break;
  default: Error("Illegal call to PrintOneDigit");
}
```

Control statements: `switch`

```
switch (d) {
  case 0: cout << "zero"; break;
  case 1: cout << "one"; break;
  case 2: cout << "two"; break;
  case 3: cout << "three"; break;
  case 4: cout << "four"; break;
  case 5: cout << "five"; break:
  case 6: cout << "six"; break;
  case 7: cout << "seven"; break;
  case 8: cout << "eight"; break;
  case 9: cout << "nine"; break;
  default: Error("Illegal call to PrintOneDigit");
}
```

use `break` and `default`

Control statements: `while`

Digit sum

```
sum = 0;
while (n > 0) {
    sum += n % 10;
    n /= 10;
}
```

Control statements: `while`

Digit sum

```
sum = 0;
while (n > 0) {
    sum += n % 10;
    n /= 10;
}
```

Solving the loop-and-half problem with `while (true)` and `break`

```
while (true) {
    ...
    if (value == sentinel) break;
    ...
}
```

Control statements: `while`

Digit sum

```
sum = 0;
while (n > 0) {
    sum += n % 10;
    n /= 10;
}
```

Solving the loop-and-half problem with `while (true)` and `break`

```
while (true) {
    ...
    if (value == sentinel) break;
    ...
}
```

Programming style: Use at most one `break` in any given loop.

Example

Echo an integer until -1

```
const int SENTINEL = -1;

while (true) {
    cout << " ? ";
    int value = GetInteger();
    if (value == SENTINEL) break;
    cout << value << endl;
}
```

Control statements: `for`

```
for (int t = 10; t >= 0; t--) {
    cout << t << endl;
}
```

Control statements: `for`

```
for (int t = 10; t >= 0; t--) {
    cout << t << endl;
}
```

The expressions *init*, *test*, and *step* are each optional, but the semicolons must appear.

- If `init` is missing, no initialization;
- If `test` is missing, assumed to be `true`;
- If `step` is missing, no action between loop cycles.

Control statements: `for`

```
for (int t = 10; t >= 0; t--) {
    cout << t << endl;
}
```

The expressions *init*, *test*, and *step* are each optional, but the semicolons must appear.

- If `init` is missing, no initialization;
- If `test` is missing, assumed to be `true`;
- If `step` is missing, no action between loop cycles.

Use `for` loop for straightforward iterative tasks;
`while` loop for indefinite iteration.

Outline

1 Structure of a C++ program

2 Variables, values, and types

3 Statements

4 Functions

Functions

Prototype and definition must match exactly.

Functions

Prototype and definition must match exactly.

function-calling mechanism

1. Evaluate arguments;
2. Creat a frame on the stack for local variables, including arguments;
3. Copy the values of the arguments in order;
4. Execute the function;
5. Return the value of the function, if any;
6. Discard the frame for the function;
7. Continue the calling function with the returned value, if any.

Function (cont.)

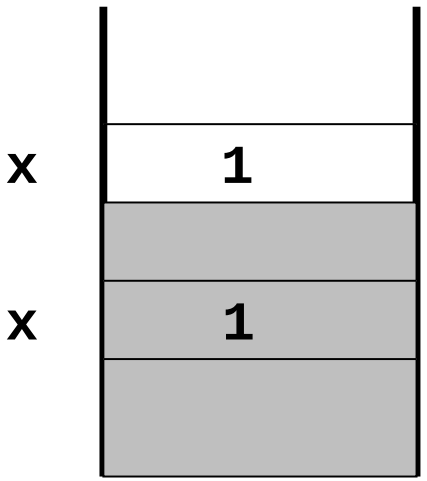
Call by value.

```
void SetToZero(int x) {
    x = 0;
}

...
x = 1;
SetToZero(x);
cout << x << endl;
```

Function (cont.)

Call by value.

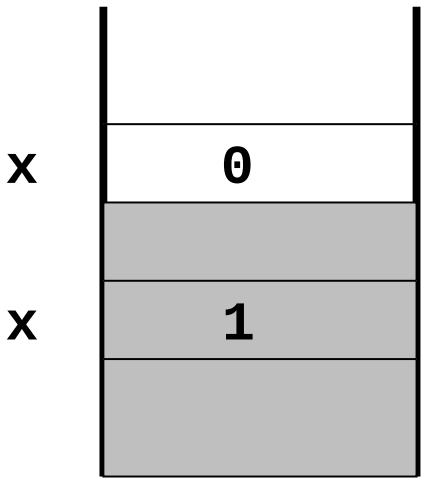
```
void SetToZero(int x) {
    x = 0;
}

...
x = 1;
SetToZero(x);
cout << x << endl;
```



Function (cont.)

Call by value.

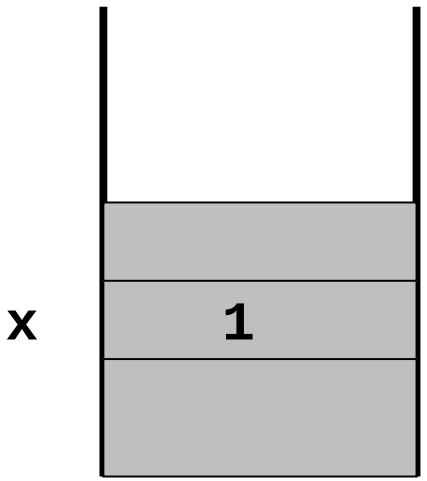
```
void SetToZero(int x) {
    x = 0;
}

...
x = 1;
SetToZero(x);
cout << x << endl;
```



Function (cont.)

Call by value.

```
void SetToZero(int x) {
    x = 0;
}

...
x = 1;
SetToZero(x);
cout << x << endl;
```



Function (cont.)

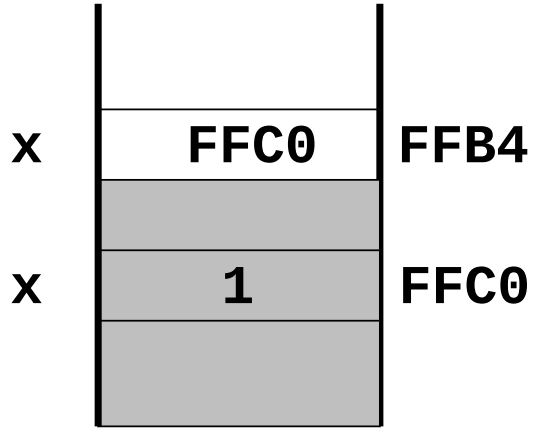
Call by reference.

```
void SetToZero(int & x) {
    x = 0;
}


...
x = 1;
SetToZero(x);
cout << x << endl;
```

Function (cont.)

Call by reference.

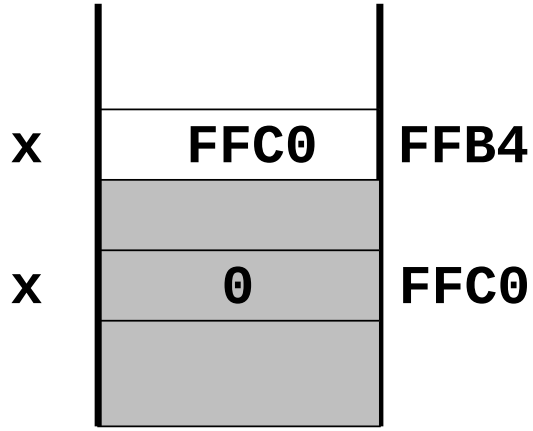
```
void SetToZero(int & x) {
    x = 0;
}


...
x = 1;
SetToZero(x);
cout << x << endl;
```



Function (cont.)

Call by reference.

```
void SetToZero(int & x) {
    x = 0;
}


...
x = 1;
SetToZero(x);
cout << x << endl;
```



Function (cont.)

In general

- If you only use the value of an argument in the function (on the right-side of assignments), call it by value.
- If you want to reflect the change of the value of an argument in the function to the caller (on the left-side of assignments), call it by reference.

Example

Program decomposition: Input-Computation-Output

Example

Program decomposition: Input-Computation-Output

Solving quadratic equations

$$ax^2 + bx + c = 0, \quad a \neq 0.$$

Textbook formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Example

Program decomposition: Input-Computation-Output

Solving quadratic equations

$$ax^2 + bx + c = 0, \quad a \neq 0.$$

Textbook formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Computer method

$$x_1 = \frac{2c}{-b - \text{sign}(b)\sqrt{b^2 - 4ac}}, \quad x_2 = \frac{c}{ax_1}.$$

Example (cont.)

```
void SolveQuadEqn(double a, doule b, double c,
                  double &x1, double &x2) {
    if (a == 0)
        Error("Coefficient a is zero.");

    double disc = b * b - 4 * a * c;
    if (disc < 0)
        Error("Solutions are complex.");
    if (disc == 0) {
        x1 = x2 = -b / (2 * a);
    } else {
        x1 = 2*c / (-b - sign(b)*sqrt(disc));
        x2 = c / (a * x1);
    }
}
```