Object-Oriented Programming

Lecture 2: Elements of a Program

01286131 Object-Oriented Programming

Software Engineering Programme
International College, KMITL

Programming Paradigms

Programming Paradigms

Programming languages are classified in many ways based on their features:

- Imperative
  - Structured/Procedural
  - Object-Oriented
- Declarative
  - Functional
  - Logic

Procedural Programming

In **procedural programming**, programs are mainly composed of three basic control structures:

- **Sequence** statements (execution in order)
- **Selection** statements (branching, if-else/switch)
- **Iteration** statements (looping, for/while)

After the source code get transformed to machine code, all program structures are reduced to simple and jumping/branching instructions.

Blocks and Subroutines

In addition to the basic control structures:

- **Blocks** are used to enable groups of statements to be treated as if they were one statement
- **Subroutines** (procedures, functions, methods, or subprograms) are used to allow a sequence to be referred to by a single statement

Object-Oriented Programming

In **object-oriented** programming, programs are mainly composed of interrelated objects:

- An **object** is a computational entity which has both **state** and **behavior**
- Objects are self-contained by bundling data and operations together, this notion is called **encapsulation**
- By encapsulation, an object is mainly used by sending it a message via its **methods** (or member functions)
- Objects are often used for modeling things found in the real world

Objects

- An **object** can be created, stored and manipulated, **without** knowing its internal structure
- In C++, an object is created from a **class** and is an ***instance*** of a class
  - A **class** is a user-defined type which is used as a blueprint for creating objects
  - A **class** in C++ can be designed in a way that it behaves **"just like a built-in type"**

A **variable** (of a built-in type) can be considered an object. We sometimes used the term **"object"** to refer to a variable.

Extending Classes

In OOP, a **class** can be defined and extended in many ways:

- By **inheritance**, a class can be defined as an extension of existing classes, forming a class hierarchy
- By overriding existing methods of existing classes, **polymorphism** can be achieved and objects from related types can be used in the same way with varying behavior

Encapsulation, **inheritance**, and **polymorphism** are major concepts in OOP. We will explore more on these topics later.

Working with Objects

Example: Interacting with Strings

```cpp
  // 1. What are you expecting the program to do?
  // 2. What's wrong with the code?
  // 3. Does it compile?
  // 4. Does it work as expected?
  // 5. If not, what would you do to correct the program?

#include <iostream>

int main()
{
    std::cout << "Please enter P1 name: ";
    std::string p1_name;
    std::cin >> p1_name;
    std::cout << "Player 1: " << p1_name << std::endl;

    std::cout << "Please enter P2 name: ";
    std::string p2_name;
    std::cin >> p2_name;
    std::cout << "Player 2: " << p2_name << std::endl;
    return 0;
}
```

Blocks and Scope of Variables

A block defines an extent to which an inner object/variable exists:

```cpp
#include <iostream>

int main()
{
    {
        std::cout << "Please enter P1 name: ";
        std::string p1_name;
        std::cin >> p1_name;

        std::cout << "Please enter P2 name: ";
        std::string p2_name;
        std::cin >> p2_name;
    }

    // `p1_name` and `p2_name` doesn't exist here!
    std::cout << "Player 1: " << p1_name << std::endl;
    std::cout << "Player 2: " << p2_name << std::endl;
    return 0;
}
```

Objects vs Variables

In some language (e.g. Rust), a primitive variable can be used like an object.

```rust
println!("{}", (-10_i32).abs());
```

In C++, some object can be used like a variable.

```cpp
std::complex<double> a = 2;
std::complex<double> b = 3i;
a += b; // `a` becomes `2 + 3i`
```

Object Creation

In C++, an object/variable is created at the point of its **definition**.

```
auto spaces = string(5, ' ');
auto ext = string{'t', 'x', 't'};
```

More examples:
(with **auto**)

```
string s1;   // default init.
string s2{}; // value init.
auto t1 = s1; // copy init.
auto t2 = "Hello"s; // also copy

string stars(5, '*');          // direct init.
auto dashes = string(5, '-'); // ?

  // list initialization
string bom{'\xEF', '\xBB', '\xBF'};
```

References: https://en.cppreference.com/w/cpp/language/initialization

C++ Pragramming Environments

C++ Online Tools

- Online IDE: https://replit.com/languages/cpp
  - Editing and Running the code in an REPL is a quick way to get started
- Compiler Explorer: https://godbolt.org
  - Useful for getting to know how the code is transformed to the machine instructions

Program Exit Code

```cpp
  // success.cpp
int main()
{
    return 0;
}
```

```cpp
  // failed.cpp
int main()
{
    return 1;
}
```

Example Session

```
$ g++ -o success -Wall -Wextra success.cpp
$ ./success
$ echo $?
0
$ g++ -o failed -Wall -Wextra failed.cpp
$ ./failed
$ echo $?
1
```

Tips: For Windows, check for **ERRORLEVEL**.

Improving the Frame Program

Overall structure (starting from previous version):

```
#include <iostream>
#include <string>

int main()
{
    // ask for a person's name
    std::cout << "Please enter your first name: ";

    // read the name
    std::string name;
    std::cin >> name;

    // build the message that we intend to write
    const std::string greeting = "Hello, " + name + "!";

    // we have to rewrite this part ...

    return 0;
}
```

Compute the Number of Rows

```
  // the number of blanks
  // surrounding the greeting
const int pad = 1;

  // the number of rows and columns to write
const int rows = pad * 2 + 3;
```

```
*****  | top border
*   *->| top pad
* x *
*   *--->| bottom pad
*****    | bottom border

Rows = (1 * 2) + 3 = 5
```

```
  // constexpr is also `const`
constexpr int pad = 2;
constexpr int rows = pad * 2 + 3;
```

```
*******
*     *->| top pad
*     *  |
* xxx *
*     *--->|
*     *    | bottom pad
*******

Rows = (2 * 2) + 3 = 7
```

The `while` Statement

Print rows of output

```
constexpr int pad = 1;
constexpr int rows = pad * 2 + 3;

// separate the output from the input
cout << endl;

// write `rows' rows of output
int r = 0;

// invariant: we have written `r' rows so far
while (r != rows) {
    // write a row of output
    std::cout << std::endl;
    ++r;
}
```

```
while (condition)
    statement
```

```
while (condition) statement
```

See "Accelerated C++, section 2.3" for a reference

The `if` Statement

Print a row

```
// invariant: we have written `c' characters
//            so far in the current row
while (c != cols) {
    // is it time to write the greeting?
    if (r == pad + 1 && c == pad + 1) {
        cout << greeting;
        c += greeting.size();
    }
    else {
        // are we on the border?
        if (r == 0 || r == rows - 1 ||
            c == 0 || c == cols - 1)
            cout << "*";
        else
            cout << " ";
        ++c;
    }
}
```

```
if (condition)
    statement
```

```
if (condition)
    statement1
else
    statement2
```

The for Statement

```
for (init-statement condition; expression)
    statement
```

```
{
    init-statement
    while (condition) {
        statement
        expression;
    }
}
```

```
  // `r' takes on the values in [0, rows)
for (int r = 0; r != rows; ++r) {
    // stuff that doesn't change
    // the value of `r'
}
```

```
{
    int r = 0;
    while (r != rows) {
        // ...
        ++r;
    }
}
```

for statements is used as a shorthand way of writing a loop

The Complete Framing Program (1)

```
#include <iostream>
#include <string>

  // say what standard-library names we use
using std::cin;          using std::endl;
using std::cout;         using std::string;

int main()
{
    // ask for the person's name
    cout << "Please enter your first name: ";

    // read the name
    string name;
    cin >> name;

    // build the message that we intend to write
    const string greeting = "Hello, " + name + "!";

    // the number of blanks surrounding the greeting
    constexpr int pad = 1;

    // the number of rows and columns to write
    constexpr int rows = pad * 2 + 3;
    const string::size_type cols = greeting.size() + pad * 2 + 2;

    // write a blank line to separate the output from the input
    cout << endl;

    // ...
```

The Complete Framing Program (2)

```
// ...

// write `rows' rows of output
// invariant: we have written `r' rows so far
for (int r = 0; r != rows; ++r) {
    string::size_type c = 0;

    // invariant: we have written `c' characters
    //            so far in the current row
    while (c != cols) {
        // is it time to write the greeting?
        if (r == pad + 1 && c == pad + 1) {
            cout << greeting;
            c += greeting.size();
        }
        else {
            // are we on the border?
            if (r == 0 || r == rows - 1 ||
                c == 0 || c == cols - 1)
                cout << "*";
            else
                cout << " ";
            ++c;
        }
    }

    cout << endl;
}
return 0;
}
```

Loop Counter

`r` takes on the values in **`[0, rows)`**

```
for (int r = 0; r != rows; ++r) {
    // write a row
}
```

`r` takes on the values in **`[1, rows]`**

```
for (int r = 1; r <= rows; ++r) {
    // write a row
}
```

The number of iterations is the same in both cases.

Tips: In C++, we often **prefer** to count from **`0`** and use the half-open range **`[0, n)`** to control the loop. We also **prefer `++r`** over **`r++`** whenever we have a choice.

Computing Student Grades (1)

```cpp
#include <iomanip>
#include <ios>
#include <iostream>
#include <string>

using std::cin;                 using std::setprecision;
using std::cout;                using std::string;
using std::endl;                using std::streamsize;

int main()
{
    // ask for and read the student's name
    cout << "Please enter your first name: ";
    string name;
    cin >> name;
    cout << "Hello, " << name << "!" << endl;

    // ask for and read the midterm and final grades
    cout << "Please enter your midterm and final exam grades: ";
    double midterm, final;
    cin >> midterm >> final;

    // ask for the homework grades
    cout << "Enter all your homework grades, "
            "followed by end-of-file: ";

    // ...
```

Computing Student Grades (2)

```
    // ...

    // the number and sum of grades read so far
    int count = 0;
    double sum = 0;

    // a variable into which to read
    double x;

    // invariant:
    //     we have read `count' grades so far, and
    //     `sum' is the sum of the first `count' grades
    while (cin >> x) {
        ++count;
        sum += x;
    }

    // write the result
    streamsize prec = cout.precision();
    cout << "Your final grade is " << setprecision(3)
         << 0.2 * midterm + 0.4 * final + 0.4 * sum / count
         << setprecision(prec) << endl;
    return 0;
}
```

Using Medians to Compute Grades (1)

```
#include <algorithm>
#include <iomanip>
#include <ios>
#include <iostream>
#include <string>
#include <vector>

using std::cin;              using std::sort;
using std::cout;             using std::streamsize;
using std::endl;             using std::string;
using std::setprecision;     using std::vector;

int main()
{
    // ask for and read the student's name
    cout << "Please enter your first name: ";
    string name;
    cin >> name;
    cout << "Hello, " << name << "!" << endl;

    // ask for and read the midterm and final grades
    cout << "Please enter your midterm and final exam grades: ";
    double midterm, final;
    cin >> midterm >> final;

    // ask for and read the homework grades
    cout << "Enter all your homework grades, "
            "followed by end-of-file: ";

    // ...
```

Using Medians to Compute Grades (2)

```
// ...

vector<double> homework;
double x;
// invariant: `homework' contains all the homework grades read so far
while (cin >> x)
    homework.push_back(x);

// check that the student entered some homework grades
typedef vector<double>::size_type vec_sz;
vec_sz size = homework.size();
if (size == 0) {
    cout << endl << "You must enter your grades.  "
                    "Please try again." << endl;
    return 1;
}

// sort the grades
sort(homework.begin(), homework.end());

// compute the median homework grade
auto mid = size / 2;
double median;
median = size % 2 == 0 ? (homework[mid] + homework[mid-1]) / 2
                       : homework[mid];

// compute and write the final grade
auto prec = cout.precision();
cout << "Your final grade is " << setprecision(3)
     << 0.2 * midterm + 0.4 * final + 0.4 * median
     << setprecision(prec) << endl;
return 0;
}
```