Queens College, CUNY, Department of Computer Science

Object Oriented Programming in C++

CSCI 211 / 611

Summer 2018

Instructor: Dr. Sateesh Mane

June 28, 2018

Namespaces

- In this lecture we shall learn about **namespaces**.
- Note for example that all of the progrms we have written thus far contain the statement **using namespace std;**
- Clearly "`std`" is a namespace (certainly the most widely used in C++).

Class Point1: review for later use

- Recall the class Point1 which we have used before and the function distance.

```
class Point1 {
public:
  void set(const double &a, const double &b)
  {
    x = a;
    y = b;
  }

  const double& getx() const { return x; }
  const double& gety() const { return y; }

  void print() const
  {
    cout << "print x,y    " << x << "    " << y << endl;
  }

private:
  double x, y;
};

double distance(const Point1 &a, const Point1 &b)
{
  double dx = a.getx() - b.getx();
  double dy = a.gety() - b.gety();
  return sqrt(dx*dx + dy*dy);
}
```

1 Ambiguity of names

- The problem is this:

  1. **Two students write the above code for `Point1` and `distance`.**
  2. Call the students Alice and Bob.
  3. In and of itself, this is not a problem: Alice and Bob wrote their code independently.
  4. However, the instructor receives submissions from both Alice and Bob.
  5. ***How will we distinguish the class `Point1` and function `distance` written by Alice and Bob?***

- One solution is to tell Alice and/or Bob to rewrite their code with different (unique) names for their classes, say `AlicePoint1` and `BobPoint1`.

  1. This is impractical, especially in a large project with multiple software developers.
  2. Furthermore, it is good programming practice to choose names (for classes, functions, methods, data members, etc.) which reflect their functionality.
  3. Names such as `AlicePoint1` and `BobPoint1` have no meaning and are confusing to understand.
  4. Such a solution leads to code that is difficult to understand and debug.

- *Why not load only one student submission at a time?*

  1. That would work for the simple situation presented above.
  2. In a large project, the project would have to be recompiled every time a different copy of the code was loaded.
  3. It would not work if both Alice's and Bob's functionalities were required simultaneously.

- A better solution is to employ **namespaces.**

2 Namespaces

- The concept of a namespace is simple.
- We select a unique name (alphanumeric) and enclose the relevant code in it.
- The code will be tagged with the namespace as explained below.
- **We create namespaces “Alice” and “Bob” and place the relevant code inside them as follows.**
- The syntax is the name of the namespace and the code enclosed in braces:

```
namespace Alice
{
  class Point1 { // etc };
  double distance(const Point1 &a, const Point1 &b) { // etc }
}
namespace Bob
{
  class Point1 { // etc };
  double distance(const Point1 &a, const Point1 &b) { // etc }
}
```

- To instantiate objects of the class `Point1`, we must now write

```
  Alice::Point1 p1, p2;
```

- To call the function `distance` written by Alice, we must write

```
  double d = Alice::distance(p1, p2);
```

- Here is an example main program with code from both Alice and Bob.

```
// include header files and namespace code for Alice and Bob

int main()
{
  Alice::Point1 ap1, ap2;
  ap1.set(1.1, 2.2);
  ap2.set(2.1, 3.2);

  Bob::Point1 bp1, bp2;
  bp1.set(-1, -2);
  bp2.set(-3, -4);

  cout << "Alice: " << Alice::distance(ap1, ap2) << endl;
  cout << "Bob:   " <<   Bob::distance(bp1, bp2) << endl;
  return 0;
}
```

2.1 Namespace Alice

- The code for the namespace Alice is given below.
- This is not necessary, but personally I like to append a comment "// Alice" at the closing brace just as a reminder of which namespace the brace is closing. Frequently the block of code is so long that it is easy to forget. It is only a personal habit and is not necessary.

```
namespace Alice
{
  class Point1 {
  public:
    void set(const double &a, const double &b)
    {
      x = a;
      y = b;
    }

    const double& getx() const { return x; }
    const double& gety() const { return y; }

    void print() const
    { cout << "print x,y    " << x << "    " << y << endl; }

  private:
    double x, y;
  };

  double distance(const Point1 &a, const Point1 &b)
  {
    double dx = a.getx() - b.getx();
    double dy = a.gety() - b.gety();
    return sqrt(dx*dx + dy*dy);
  }
}  // namespace Alice
```

2.2 Namespace `Bob`

- Working code for the namespace `Bob` is given below.
- However, we change data members x and y in the class `Point1` to the type `long`.
- **This is to demonstrate that although the class name may be `Point1` in both cases, the internal functionality may be different.**
- This is just a simple example to justify why we want the functionalities of both `Alice` and `Bob` in the project simultaneously.

```
namespace Bob
{
  class Point1 {
  public:
    void set(const long &a, const long &b)
    {
      x = a;
      y = b;
    }

    const long& getx() const { return x; }
    const long& gety() const { return y; }

    void print() const
    { cout << "print x,y    " << x << "    " << y << endl; }

  private:
    long x, y;
  };

  double distance(const Point1 &a, const Point1 &b)
  {
    long dx = a.getx() - b.getx();
    long dy = a.gety() - b.gety();
    return sqrt(dx*dx + dy*dy);
  }
}  // namespace Bob
```

3 Mix & match: code from different namespaces

- Let us modify the main program in Sec. 2 to mix objects by Alice and Bob.
- The resulting code will generate a compiler error.

```
// include header files and namespace code for Alice and Bob

int main()
{
  Alice::Point1 ap1, ap2;
  ap1.set(1.1, 2.2);
  ap2.set(2.1, 3.2);

  Bob::Point1 bp1, bp2;
  bp1.set(-1, -2);
  bp2.set(-3, -4);

  cout << "mix and match: " << Alice::distance(ap1, bp2) << endl  // compiler error
  cout << "mix and match: " <<   Bob::distance(ap2, bp1) << endl; // compiler error
  return 0;
}
```

- The function calls generate compilation errors because there is no "`distance`" function with a signature as below.

```
  double distance(const Alice::Point1 &a, const Bob::Point1 &b);
```

- **Such a function can be written.**

```
double distance(const Alice::Point1 &a, const Bob::Point1 &b)
{
  long dx = a.getx() - b.getx();
  long dy = a.gety() - b.gety();
  return sqrt(dx*dx + dy*dy);
}
```

- **Without the above function, the compiler genererates an error.**

4 Splitting a namespace

- **We do not have to write all the code of a namespace in a single block.**
- If a code in namespace block is too long, we can split the code into multiple blocks.
- *The individual blocks may be placed in separate files of the project.*
- We split the code in the namespace `Alice` into two blocks.
  1. The declaration for the class `Point1` is placed in the first block.
  2. The code for the function `distance` is placed in the second block.

```
namespace Alice   // start of block 1
{
  class Point1 {
  public:
    void set(const double &a, const double &b)
    {
      x = a;
      y = b;
    }

    const double& getx() const { return x; }
    const double& gety() const { return y; }

    void print() const
    { cout << "print x,y    " << x << "    " << y << endl; }

  private:
    double x, y;
  };
}  // namespace Alice, end of block 1

namespace Alice   // start of block 2
{
  double distance(const Point1 &a, const Point1 &b)
  {
    double dx = a.getx() - b.getx();
    double dy = a.gety() - b.gety();
    return sqrt(dx*dx + dy*dy);
  }
}  // namespace Alice, end of block 2
```

- **The two blocks are both tagged “`namespace Alice`” and nothing else**
- The words/numbers “block 1” and “block 2” are simply comments.

5 Keyword: “using” a namespace

- Consider the following main program.

```
int main()
{
  Alice::Point1 ap1, ap2;
  ap1.set(1.1, 2.2);
  ap2.set(2.1, 3.2);

  cout << Alice::distance(ap1, ap2) << endl;
  ap1.print();
  ap1.print();
  return 0;
}
```

- All the objects in the program (also “distance”) are from the namespace Alice.
- In that case, it is inconvenient to prepend “Alice::” everywhere, because there is no ambiguity what “Point1” or “distance” mean.
- We can insert s statement “**using namespace Alice;**” before the main program (or anywhere in the program, as long as it is before we wish to use the namespace Alice).
- Then we can omit the prefix “Alice::” and the compiler will automatically search for a match in the namespace Alice.
- Here is the C++ code with the keyword “using namespace Alice” before the main program.

```
#include <iostream>
#include <cmath>
using namespace std;

// namespace Alice, blocks 1 and 2

using namespace Alice;                     // keyword "using"

int main()
{
  Point1 ap1, ap2;       // compiler searches for "Point1" in namespace Alice
  ap1.set(1.1, 2.2);
  ap2.set(2.1, 3.2);

  cout << distance(ap1, ap2) << endl;
  ap1.print();
  ap1.print();
  return 0;
}
```

6 Namespace "std"

- In fact, we have been using the "standard namespace" `std` in all our programs.
- All of our programs contain the statement **using namespace std;**
- The namespace "`std`" is heavily used.
- It is a huge namespace, with the code in many blocks.
- Without the statement "**using namespace std;**" we must write "`std::`" explicitly.

```
#include <iostream>

int main()
{
  std::cout << "hello, world!" << std::endl;          // tag "std::" required
  return 0;
}
```

7 Nested namespaces

- Namespaces can be nested.
- Suppose there are two students named Alice, in different sections with lab instructors `Alpha` and `Beta`.
- Also students Bob in namespace Alpha and Charlie in namespace Beta (see below).
- Then we can define nested namespaces as follows.

```
#include <iostream>
using namespace std;

namespace Alpha
{
  namespace Alice
  {
    void print() { cout << "Alice, Alpha" << endl; }
  }
  namespace Bob
  {
    void print() { cout << "Bob, Alpha" << endl; }
  }
}

namespace Beta
{
  namespace Alice
  {
    void print() { cout << "Alice, Beta" << endl; }
  }
  namespace Charlie
  {
    void print() { cout << "Charlie, Beta" << endl; }
  }
}

int main()
{
  Alpha::Alice::print();
  Beta::Alice::print();
  Alpha::Bob::print();
  Beta::Charlie::print();
  return 0;
}
```

8 Summary

- A **namespace** is a unique identifier to tag a block of code.
- The code enclosed in a namespace can be anything: classes, functions, etc.
- If there are two or more classes (or two or more functions) with the same name, the namespace is used to resolve the ambiguity.
  1. As an example we have the class `Point1` in the namespaces `Alice` and `Bob`.
  2. As another example we have the function `distance` in the namespaces `Alice` and `Bob`.
  3. We write `Alice::Point1`, `Bob::Point1`, `Alice::distance` and `Bob::distance`, to resolve the ambiguity of what we mean to use.
- **A namespace can be split into multiple blocks.**
- The individual blocks of a namespace can be placed in different project files.
- Namespaces can be nested.
- If no ambiguity will result if the namespace is not explicitly mentioned, we can employ the keyword **using.**
  1. The standard namespace `std` is a very heavily used namespace in C++.
  2. The statement "**using namespace std;**" is ubiquitous in C++ programs.