Namespaces

Modified from Section 12.2

Namespaces

- A namespace is a collection of name definitions, such as class definitions and variable declarations
  - If a program uses classes and functions written by different programmers, it may be that the same name is used for different things
  - Namespaces help us deal with this problem

The Using Directive

- #include <iostream> places names such as cin and cout in the std namespace
- The program does not know about names in the std namespace until you add

  using namespace std;

  (if you do not use the std namespace, you can define cin and cout to behave differently)

The Global Namespace

- Code you write is in a namespace
  - It is in the global namespace unless you specify a namespace
  - The global namespace does not require the using directive

Name Conflicts

- If the same name is used in two namespaces
  - The namespaces cannot be used at the same time
  - Example: If my_function is defined in namespaces ns1 and ns2, the two versions of my_function could be used in one program by using local using directives this way:

```
{
  using namespace ns1;
  my_function( );
}
```

```
{
  using namespace ns2;
  my_function( );
}
```

Scope Rules For using

- A block is a list of statements enclosed in { }s
- The scope of a using directive is the block in which it appears
- A using directive placed at the beginning of a file, outside any block, applies to the entire file

Creating a Namespace

- To place code in a namespace
  - Use a namespace grouping
    - namespace Name_Space_Name
      {
      Some_Code
      }
- To use the namespace created
  - Use the appropriate using directive
    - using namespace Name_Space_Name;

Namespaces: Declaring a Function

- To add a function to a namespace
  - Declare the function in a namespace grouping

```
namespace savitch1
{
    void greeting( );
}
```

Namespaces: Defining a Function

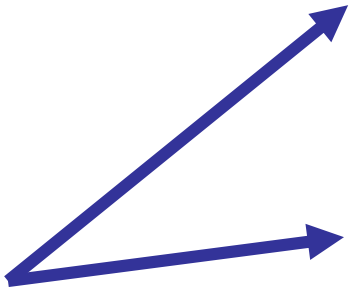
- To define a function declared in a namespace
  - Define the function in a namespace grouping

```
namespace savitch1
{
      void greeting( )
      {
            cout << "Hello from namespace savitch1.\n";
      }
 }
```

Namespaces: Using a Function

- To use a function defined in a namespace
  - Include the using directive in the program where the namespace is to be used
  - Call the function as the function would normally be called

```
int main( )
{
    {
        using namespace savitch1;
        greeting( );
    }
}
```



Using directive's scope

Display 12.5 (1-2)

Display 12.5 (1/2)

Back Next

Namespace Demonstration (*part 1 of 2*)

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

namespace savitch1
{
    void greeting( );
}

namespace savitch2
{
    void greeting( );
}

void big_greeting( );

int main( )
{
    {
        using namespace savitch2;
        greeting( );
    }

    {
        using namespace savitch1;
        greeting( );
    }

    big_greeting( );

    return 0;
}
```

Names in this block use definitions in namespaces `savitch2`, `std`, *and the global namespace.*

Names in this block use definitions in namespaces `savitch1`, `std`, *and the global namespace.*

Names out here only use definitions in namespace `std` *and the global namespace.*

Display 12.5 (2/2)

Namespace Demonstration (*part 2 of 2*)

```
namespace savitch1
{
    void greeting( )
    {
        cout << "Hello from namespace savitch1.\n";
    }
}

namespace savitch2
{
    void greeting( )
    {
        cout << "Greetings from namespace savitch2.\n";
    }
}

void big_greeting( )
{
    cout << "A Big Global Hello!\n";
}
```

Sample Dialogue

```
Greetings from namespace savitch2.
Hello from namespace savitch1.
A Big Global Hello!
```

A Namespace Problem

- Suppose you have the namespaces below:

```
namespace ns1
{
      fun1( );
      my_function( );
 }
```

```
namespace ns2
{
      fun2( );
      my_function( );
}
```

- Is there an easier way to use both namespaces considering that my_function is in both?

Qualifying Names

- Using declarations (not directives) allow us to select individual functions to use from namespaces
  - using ns1::fun1; //makes only fun1 in ns1 avail
    - The scope resolution operator identifies a namespace here
    - Means we are using only namespace ns1's version of fun1
  - If you only want to use the function once, call it like this

```
ns1::fun1( );
```

Qualifiying Parameter Names

- To qualify the type of a parameter with a using declaration
  - Use the namespace and the type name

    ```
    int get_number (std::istream input_stream)
      …
    ```

    - istream is the istream defined in namespace std
    - If istream is the only name needed from namespace std, then you do not need to use

      ```
      using namespace std;
      ```

Unnamed Namespaces

- As we have done helper functions so far, they are not really hidden (Display 12.2)
  - We would like them to be local to the implementation file to implement information hiding
- The unnamed namespace can hide helper functions
  - Names defined in the unnamed namespace are local to the compilation unit
    - A compilation unit is a file (such as an implementation file) plus any file(s) #included in the file

The unnamed grouping

- Every compilation unit has an unnamed namespace
  - The namespace grouping is written as any other namespace, but no name is given:

```
namespace
{
      void sample_function( )
       …
 }  //unnamed namespace
```

Names In The unnamed namespace

- Names in the unnamed namespace
  - Cannot be reused outside the compilation unit
  - Can be used in the compilation unit without a namespace qualifier
- The rewritten version of the DigitalTime interface is found in **Display 12.6** while the implementation file is shown in **Display 12.7 (1)** **Display 12.7 (2)**

Display 12.6

Back Next

DISPLAY 12.6 Placing a Class in a Namespace—Header File

```
1  //Header file dtime.h: This is the interface for the class DigitalTime.
2  //Values of this type are times of day. The values are input and output in
3  //24-hour notation, as in 9:30 for 9:30 AM and 14:45 for 2:45 PM.
4
5  #ifndef DTIME_H
6  #define DTIME_H
7
8  #include <iostream>
9  using namespace std;
10
11 namespace dtimesavitch
12 {
13
14      class DigitalTime
15      {
16
     <The definition of the class DigitalTime is the same as in Display 12.1.>
17      };
18 }//end dtimesavitch
19
20 #endif //DTIME_H
```

One grouping for the namespace `dtimesavitch`*.*
Another grouping for the namespace `dtimesavitch`
is in the implementation file `dtime.cpp`*.*

DISPLAY 12.7 Placing a Class in a Namespace—Implementation File *(part 1 of 2)*

```
1  //Implementation file dtime.cpp (your system may require some
2  //suffix other than .cpp): This is the IMPLEMENTATION of the ADT DigitalTime.
3  //The interface for the class DigitalTime is in the header file dtime.h.
4  #include <iostream>
5  #include <cctype>
6  #include <cstdlib>
7  #include "dtime.h"
8  using namespace std;
9
10 namespace
11 {
12     //These function declarations are for use in the definition of
13     //the overloaded input operator >>:
14
15     void read_hour(istream& ins, int& the_hour);
16     //Precondition: Next input in the stream ins is a time in 24-hour notation,
17     //like 9:45 or 14:45.
18     //Postcondition: the_hour has been set to the hour part of the time.
19     //The colon has been discarded and the next input to be read is the minute.
20
21     void read_minute(istream& ins, int& the_minute);
22     //Reads the minute from the stream ins after read_hour has read the hour.
23
24     int digit_to_int(char c);
25     //Precondition: c is one of the digits '0' through '9'.
26     //Returns the integer for the digit; for example, digit_to_int('3')
27     //returns 3.
28 }//unnamed namespace
29
30
31 namespace dtimesavitch
32 {
33     bool operator ==(const DigitalTime& time1, const DigitalTime& time2)
    <The rest of the definition of == is the same as in Display 12.2.>
34
35     DigitalTime::DigitalTime( )
    <The rest of the definition of this constructor is the same as in Display 12.2.>
36
37     DigitalTime::DigitalTime(int the_hour, int the_minute)
    <The rest of the definition of this constructor is the same as in Display 12.2.>
38     void DigitalTime::advance(int minutes_added)
    <The rest of the definition of this advance function is the same as in Display 12.2.>
39
40     void DigitalTime::advance(int hours_added, int minutes_added)
    <The rest of the definition of this advance function is the same as in Display 12.2.>
41
```

One grouping for the unnamed namespace

One grouping for the namespace dtimesavitch. Another grouping is in the file dtime.h.

(continued)

Display 12.7 (2/2)

DISPLAY 12.7 Placing a Class in a Namespace—Implementation File *(part 2 of 2)*

```
42      ostream& operator <<(ostream& outs, const DigitalTime& the_object)
      <The rest of the definition of << is the same as in Display 12.2.>
43
44      //Uses iostream and functions in the unnamed namespace:
45      istream& operator >>(istream& ins, DigitalTime& the_object)
46      {
47          read_hour(ins, the_object.hour);
48          read_minute(ins, the_object.minute);
49          return ins;
50      }
51  } //dtimesavitch
52
53
54  namespace
55  {
56      int digit_to_int(char c)
      <The rest of the definition of digit_to_int is the same as in Display 12.2.>
57
58      void read_minute(istream& ins, int& the_minute)
      <The rest of the definition of read_minute is the same as in Display 12.2.>
59
60      void read_hour(istream& ins, int& the_hour)
      <The rest of the definition of read_hour is the same as in Display 12.2.>
61
62  }//unnamed namespace
```

Functions defined in the unnamed namespace are local to this compilation unit (this file and included files). They can be used anywhere in this file, but have no meaning outside this compilation unit.

Another grouping for the unnamed namespace.

Namespaces In An Application

- The application file for the DigitalTime ADT is shown in **Display 12.8 (1)** **Display 12.8 (2)**

Display 12.8 (1/2)

DISPLAY 12.8 Placing a Class in a Namespace—Application Program *(part 1 of 2)*

```
1  //This is the application file: timedemo.cpp. This program
2  //demonstrates hiding the helping functions in an unnamed namespace.
3
4  #include <iostream>
5  #include "dtime.h"
6
7  void read_hour(int& the_hour);
8
9  int main( )
10 {
11     using namespace std;
12
13     using namespace dtimesavitch;
14
15     int the_hour;
16     read_hour(the_hour);
17
18     DigitalTime clock(the_hour, 0), old_clock;
19
20     old_clock = clock;
21     clock.advance(15);
22     if (clock == old_clock)
23         cout << "Something is wrong.";
24     cout << "You entered " << old_clock << endl;
25     cout << "15 minutes later the time will be "
26          << clock << endl;
27
28     clock.advance(2, 15);
29     cout << "2 hours and 15 minutes after that\n"
30          << "the time will be "
31          << clock << endl;
32
33     return 0;
34 }
35 void read_hour(int& the_hour)
36 {
37      using namespace std;
38
39      cout << "Let's play a time game.\n"
40           << "Let's pretend the hour has just changed.\n"
```

If you place the using directives here, then the program behavior will be the same.

This is a different function `read_hour` *than the one in the implementation file* `dtime.cpp` *(shown in Display 12.7).*

(continued)

Display 12.8 (2/2)

Back Next

DISPLAY 12.8 Placing a Class in a Namespace—Application Program *(part 2 of 2)*

```
41            << "You may write midnight as either 0 or 24,\n"
42            << "but I will always write it as 0.\n"
43            << "Enter the hour as a number (0 to 24): ";
44        cin >> the_hour;
45        if (the_hour == 24)
46           the_hour = 0;
47    }
```

Sample Dialogue

```
Let's play a time game.
Let's pretend the hour has just changed.
You may write midnight as either 0 or 24,
but I will always write it as 0.
Enter the hour as a number (0 to 24): 11
You entered 11:00
15 minutes later the time will be 11:15
2 hours and 15 minutes after that
the time will be 13:30
```

Compilation Units Overlap

- A header file can be #included in two files (for example, the class implementation file and an application file)
  - It is in two compilation units
  - Participates in two unnamed namespaces!
  - This is OK as long as each of the compilation units makes sense independent of the other
    - A name in the header file's unnamed namespace cannot be defined again in the unnamed namespace of the implementation or application file

Naming Namespaces

- To avoid choosing a name for a namespace that has already been used
  - Use some other unique string

Global or unnamed?

- Names in the global namespace have global scope (all files)
  - They are available without a qualifier to all the program files
- Names in the unnamed namespace are local to a compilation unit
  - They are available without a qualifier within the compilation unit