L ISTING 1-1: T HE S TRING C ODING C LASS

#include <stdio.h>

#include <string>

class StringCoding

{

private:

// The key to use in encrypting the string

std::string sKey;

public:

// The constructor, uses a preset key

StringCoding( void )

{

sKey = “ATest”;

}

// Main constructor, allows the user to specify a key

StringCoding( const char \*strKey )

{

if ( strKey )

sKey = strKey;

else

sKey = “ATest”;

}

// Copy constructor

StringCoding( const StringCoding& aCopy )

{

sKey = aCopy.sKey;

}

public:

// Methods

std::string Encode( const char \*strIn );

std::string Decode( const char \*strIn );

private:

std::string Xor( const char \*strIn );

};

std::string StringCoding::Xor( const char \*strIn )

{

std::string sOut = “”;

int nIndex = 0;

for ( int i=0; i<(int)strlen(strIn); ++i )

{

char c = (strIn[i] ^ sKey[nIndex]);

sOut += c;

nIndex ++;

if ( nIndex == sKey.length() )

nIndex = 0;

}

return sOut;

}

// For XOR encoding, the encode and decode methods are the same.

std::string StringCoding::Encode( const char \*strIn )

{

return Xor( strIn );

}

std::string StringCoding::Decode( const char \*strIn )

{

return Xor( strIn );

}

int main(int argc, char \*\*argv)

{

if ( argc < 2 )

{

printf(“Usage: ch1\_1 inputstring1 [inputstring2...]\n”);

exit(1);

}

StringCoding key(“XXX”);

for ( int i=1; i<argc; ++i )

{

std::string sEncode = key.Encode( argv[i] );

printf(“Input String : [%s]\n”, argv[i] );

printf(“Encoded String: [%s]\n”, sEncode.c\_str() );

std::string sDecode = key.Decode( sEncode.c\_str() );

printf(“Decoded String: [%s]\n”, sDecode.c\_str() );

}

printf(“%d strings encoded\n”, argc-1);

return 0;

}

Making Updates to an Encapsulated Class ([1], p. 10)

One of the benefits of encapsulation is that it makes updating your hidden data simple and convenient. With encapsulation, you can easily replace the underlying encryption algorithm in Listing 1-1 with an alternative if one is found to work better. In our original algorithm, we did an “exclusive logical or” to convert a character to another character. In the following example, suppose that we want to use a different method for encrypting strings. For simplicity, suppose that this new algorithm encrypts strings simply by changing each character in the input string to the next letter position in the alphabet: An a becomes a b, a c becomes a d, and so on. Obviously, our decryption algorithm would have to do the exact opposite, subtracting one letter position from the input string to return a valid output string. We could then modify the Encode method in Listing 1-1 to reflect this change. The following steps show how:

1. Reopen the source file in your code editor. In this example, we called the source file ch01.cpp .

2. Modify the code as shown in Listing 1-2.

3. Save the source code as a file in the code editor and then close the code editor.

4. Compile the application, using your favorite compiler on your favorite operating system.

5. Run the application on your favorite operating system.

L ISTING 1-2: U PDATING THE S TRING C ODING C LASS

std::string StringCoding::Encode( const char \*strIn )

{

std::string sOut = “”;

for ( int i=0; i<(int)strlen(strIn); ++i )

{

char c = strIn[i];

c ++;

sOut += c;

}

return sOut;

}

std::string StringCoding::Decode( const char \*strIn )

{

std::string sOut = “”;

for ( int i=0; i<(int)strlen(strIn); ++i )

{

char c = strIn[i];

c --;

sOut += c;

}

return sOut;

}

You might think that this approach would have an impact on the developers who were using our class. The developers don’t have to worry about it. When we compile and run this application, we get the following output:

$ ./ch1\_1a.exe “hello”

Input String : [hello]

Encoded String: [ifmmp]

Decoded String: [hello]

1 strings encoded

As you can see, the algorithm changed, yet the encoding and decoding still worked and the application code didn’t change at all. This, then, is the real power of encapsulation: It’s a black box. The end users have no need to know how something works in order to use it; they simply need to know what it does and how to make it do its thing. Encapsulation also solves two other big problems in the programming world:

By putting all the code to implement specific functionality in one place, you know exactly where to go when a bug crops up in that functionality. Rather than having to chase the same code in a hundred scattered places, you have it in one place.

You can change how your data is internally stored without affecting the program external to that class. For example, imagine that in the first version of the code just given, we chose to use an integer value rather than the string key. The outside application would never know, or care.

If you really want to “hide” your implementation from the user — yet still give the end user a chance to customize your code — implement your own types for the values to be passed in. Doing so requires your users to use your specific data types, rather than more generic ones.

REFERENCIAS

[1] Telles, M. (2005). C++ Timesaving Techniques For Dummies (1/a edición). Wiley Publishing Inc.

(For dummies) Matthew Telles - C++ Timesaving Techniques For Dummies-Wiley (2005).pdf