

ENGG 233 Fall 2004 L02: Examples for Oct. 4

Author: Dr. S. A. Norman

Electronic copies of handouts for L02 can be found at

<http://www.enel.ucalgary.ca/People/Norman/engg233-L02-fall2004/>

switch-example.cc: I can't think of a practical use for a function that prints the number of letters in the English spelling of an integer, but it does allow for a clear and simple demonstration of how a `switch` statement works ...

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

void f(int i);
int main()
{
    f(3);
    f(6);
    f(8);
    f(11);
    return 0;
}

void f(int i)
{
    cout << "The number " << i << " has ";
    switch(i) {
        case 1: case 2: case 6: case 10:
            cout << "three letters ";
            break;
        case 0: case 4: case 5: case 9:
            cout << "four letters ";
            break;
        case 3: case 7: case 8:
            cout << "five letters ";
            break;
        default:
            cout << "six or more letters ";
    }
    cout << "in its English spelling." << endl;
}
```

When the above program is compiled and run, the output is ...

```
The number 3 has five letters in its English spelling.
The number 6 has three letters in its English spelling.
The number 8 has five letters in its English spelling.
The number 11 has six or more letters in its English spelling.
```

The definition of the function `f` in the previous example could alternatively be written using `if`, `else`, the `==` operator and the `||` operator...

```
void f(int i)
{
    cout << "The number " << i << " has ";
    if(i == 1 || i == 2 || i == 6 || i == 10)
        cout << "three letters ";
    else if(i == 0 || i == 4 || i == 5 || i == 9)
        cout << "four letters ";
    else if(i == 3 || i == 7 || i == 8)
        cout << "five letters ";
    else
        cout << "six or more letters ";
    cout << "in its English spelling." << endl;
}
```

drop-through-example.cc: A `break` statement within a `switch` statement says, “Skip to the end of the `switch` statement.” If a `break` statement is left out, you get a behaviour—usually undesired—called “drop-through” ...

```
#include <iostream>
using namespace std;
void f(int i);
int main()
{
    f(4);
    return 0;
}

void f(int i)
{
    cout << "The number " << i << " has ";
    // Note the absence of break statements.
    switch(i) {
        case 1: case 2: case 6: case 10:
            cout << "three letters ";
        case 0: case 4: case 5: case 9:
            cout << "four letters ";
        case 3: case 7: case 8:
            cout << "five letters ";
        default:
            cout << "six or more letters ";
    }
    cout << endl << "in its English spelling." << endl;
}
```

When the above program is compiled and run, the output is ...

```
The number 4 has four letters five letters six or more letters
in its English spelling.
```

Note how the `switch` statement started with the code for case 4, and then ran through the statements for all the remaining cases instead of skipping them.

“Short-circuit” evaluation rules for `||` and `&&`:

In evaluating `expr1 || expr2`, `expr1`—the expression on the left side—is evaluated first. If `expr1` has a value of `true`, then evaluation of `expr2` will be skipped entirely.

In evaluating `expr1 && expr2`, `expr1`—the expression on the left side—is evaluated first. If `expr1` has a value of `false`, then evaluation of `expr2` will be skipped entirely.

What values will all of the variables have at the end of this code fragment?

```
int main()
{
    bool b1, b2, b3, b4, b5, b6;
    int i0 = 0, i1 = 10, i2 = 20, i3 = 30, i4 = 40, i5 = 50, i6 = 60;
    b1 = (i0 < 0) || (i1++ > 5);
    b2 = (i0 == 0) || (i2++ > 15);
    b3 = (i3++ > 100) || (i0 != 0);
    b4 = !(i0 == 0) && (0 < i4++);
    b5 = (i0 < 100) && (100 < i5++);
    b6 = i6++ && (i0 == 0);
    return 0;
}
```

Answers—make sure you know why these are correct ...

```
i0: 0    i1: 11    i2: 20    i3: 31    i4: 40    i5: 51    i6: 61
          b1: true   b2: true   b3: false b4: false b5: false b6: true
```