

ENGG 233 Fall 2004 L02: Handout for lecture of Oct. 5 (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/>

Today's problem is: Write a program to convert a height in meters to the equivalent in feet and inches, accurate to the nearest half-inch. Here's an example dialogue:

```
Enter a positive number of meters (such as 1.71 or 2 or 2.07):
2
Using a value of 2 meters ...
To the nearest half-inch, that is 6 feet, 6 and 1/2 inches.
```

And here's another example dialogue:

```
Enter a positive number of meters (such as 1.71 or 2 or 2.07):
1.65
Using a value of 1.65 meters ...
To the nearest half-inch, that is 5 feet, 5 inches.
```

Here's a *sketch* of a solution. Some decisions have been made about what functions will do, but a lot of coding is left to do ...

```
#include <iostream>
using namespace std;
const double METERS_PER_INCH = 0.0254;

double get_meters();
// Prompt user to enter a positive number of meters, read the
// user's input from cin, and return that value.

void
convert(double meters,          // must be positive
        int& feet,             // will be set to number of feet in meters
        int& inches,          // number of inches left after computing feet
        bool& plus_a_half); // true means 1/2 inch left after computing inches
// Convert meters to nearest height in feet, inches, and half-inches.

void display(int feet, int inches, bool plus_a_half);
// Use cout to display height.  Examples:
//   display(5, 10, true) would show "5 feet, 10 and 1/2 inches"
//   display(6, 2, false) would show "6 feet, 2 inches"

int main()
{
    // Call get_meters, then convert, then display.
}

void
convert(double meters, int& feet, int& inches, bool& plus_a_half)
{
    // Find the whole numbers of feet and inches in meters.

    // Is there an extra 1/2 inch?  Or should the number of inches be
    // rounded upwards?
}

void display(int feet, int inches, bool plus_a_half)
{
    // Need to deal properly with plus_a_half.
}

double get_meters()
{
    // Prompt, get input, return it.
}
```

Some notes on C++ code layout. For the most part, a C++ compiler does not care about space between pieces of a program, or how code is broken into lines. For example, this is a silly but valid way to code a program that prints a hello world message:

```
#include<iostream>
using namespace std;int main(){cout<<"hello world"<<endl;return 0;}
```

And here is another silly but valid way to do the same thing:

```
#include <iostream>
using
namespace
std
;
int
main
(
)
{
cout
<<
"hello world"
<<
endl
;
return
0
;
}
```

The above two examples should be enough to convince you that C++ code layout is mainly for human readers, not for compilers.

There are a few important restrictions. Preprocessor directives (lines that start with #) must be on lines by themselves and can't be split up:

```
// The following will not be accepted by the compiler.
#include
<iostream>

// Nor will the next line be accepted.
#include <cmath> using namespace std;
```

And you can't start a string literal on one line and finish it on another:

```
cout << "hello
world" << endl; // Will not compile!
```

When coding things like if-else statements, what matters is the number of statements, not the number of lines:

```
if (x < 0)
    y = -x; z = 0; // No good: two statements between if and else!
else
    y = x; z = 1; // z = 1; does not belong to the else.
```

Space around operators doesn't matter to the compiler. For example, the next four lines have identical meanings:

```
void my_function(double&d);
void my_function(double& d);
void my_function(double &d);
void my_function(double & d);
```

But careful use of space enhances readability of code by humans:

```
a=2*(b+c)+(d+e)*f;           // Brutal.
a = 2 * (b + c) + (d + e) * f; // Much nicer.
```