PIC 10A 1C: Week 2b Discussion Problems (Thursday, 1/14/2016)

[**Solutions**] TA: Eric Kim *[Updated: 2/9/2016 v3]*

1. Spot the Error

Louis Reasoner is trying to write a program, but there are some issues. Identify the mistakes, and fix them.

Desired Output:

```
The falling leaves...
```

Louis' Code:

```
include iostream

int main() {

    / displays "The falling leaves..." to the screen

    cout << "The falling"

    cout << "leaves..."

    return "goodbye!";
}
```

Answer: [corrections in **red**]

```
#include <iostream>
using namespace std;
int main() {

    // displays "The falling leaves..." to the screen

    cout << "The falling";

    cout << " leaves...";

    return 0;
}
```

2. Code Jeopardy

Write C++ code that, when run, outputs the desired output.

Assume that we are using the standard namespace (`using namespace std;`), and that the iostream library is included (`#include <iostream>`).

Example:

Desired Output:

```
Hello World!
```

C++ Code (you fill this in):

```
cout << "Hello World!";
```

2.a.

Desired Output:

```
For You, "Blue"
```

C++ Code (you fill this in):

```
cout << "For You, \"Blue\"";
```

2.b.

Desired Output:

```
Throw the \m/ horns!
```

C++ Code (you fill this in):

```
cout << "Throw the \\m/ horns!";
```

2.c.

Desired Output:

```
A newline \n
does that!
```

C++ Code (you fill this in):

```
cout << "A newline \\n\n";
cout << "does that!";
```

Desired Output:

```
He "said", but 'she' said.
```

C++ Code (you fill this in):

```
cout << "He \"said\", but \'she\' said.";
```

3. To truly understand, one must become the compiler…

For each code fragment, write what the output is. If it crashes, explain why it crashes.
Assume that we are using the standard namespace, and that the iostream library is included.

3.a.

```
cout << "T" << "h" << "e" << " \'best";
```

Output:

```
The 'best
```

3.b.

```
cout << "a bird \nest/";
```

Output:

```
a bird
est/
```

3.c.

```
cout << "To be, or to " " be " ";
```

Output:

```
Compile error! Unmatched double-quotation marks.
```

3.d.

```
cout << "To be, or to " " be "  " ";
```

Output:

```
To be, or to  be
[note: two spaces between 'to' and 'be']
```

3.e.

```
cout << "There are " << "44+55" << " red balloons.";
```

Output:

```
There are 44+55 red balloons.
```

3.f.

```
cout << "w" << endl << "a\n" << "t" << "endl" << "!";
```

Output:

```
w
a
tendl!
```

3.g.

```
cout << "" "" "";
cout << "\\\\\\" << "///";
```

Output:

```
\\\///
```

4. Bases loaded!

Complete the following table, converting to/from binary/decimal as needed. The first row has been completed for you.

Decimal (Base 10)	Binary (Base 2)
3	0011
2	**0010**
8	1000
15	**1111**
9	1001
7	0111

4. From source to executable: A source file's compilation journey

Recall that a source file (ie .cpp file) is transformed into an executable via a series of steps: the preprocessor, the compiler, the assembler, and finally the linker.
For each component of the compilation process, describe the inputs and outputs, and briefly describe what each component does.

Preprocessor Input: C++ Code Output: Transformed/Expanded C++ Code Purpose: Takes your input C++ code, and outputs an equivalent C++ file, but with some changes. For instance, the preprocessor takes your include statements (#include <iostream>) and replaces it with the actual contents of the iostream library (specifically, the iostream header). Tip: Any line that starts with "#" is often a signal that the preprocessor uses to expand/transform code.
Compiler Input: C++ Code (ie the output of the preprocessor) Output: Assembly Language Purpose: First major step to turn C++ code into an executable. Performs syntax checks on the code, and outputs human-readable error messages if the compiler can't compile the code (ie undefined variables, etc.). Note: There are errors that the compiler can't detect. For instance: logic errors, human errors. But it can (and will) catch errors such as: typos, undefined variables, missing semi-colons.
Assembler Input: Assembly Language Output: Machine code (ie 0's and 1's, binary) Purpose: Convert assembly language (in a human-readable format!) to binary (a machine-readable format). The resulting binary code is called an object file, and is simply a bunch of 0's and 1's. The object file is not yet an executable, though it is close to being one. It needs to be linked to become a full-fledged executable.
Linker Input: Object files Output: Executable Purpose: Packages together ("links") multiple object files into a single, final executable. One important job a linker does is package any required libraries that a program may requires. For instance, if my code uses a face-detection library, then the linker will link my program's object file with the face-detection library's object file.

There are two styles of linking: static and dynamic linking. To statically link a library, the library's object file is packaged into the executable at link time.
A dynamically-linked library, however, does not get packaged into the executable. Instead, it will be loaded while running the executable.