- Solution

In this lesson, we'll look at the solution review of the problem statement 2 form last exercise.

WE'LL COVER THE FOLLOWING

Solution Review of Problem Statement 2

```
// printf.cpp

#include <iostream>

void myPrintf(const char* format)
{
    std::cout << format;
}

template<typename T, typename... Args>
void myPrintf(const char* format, T value, Args... args)
{
    for ( ; *format != '\0'; format++ ) {
        if ( *format == '%' ) {
           std::cout << value;
           myPrintf(format+1, args...);
           return;
        }
        std::cout << *format;
    }
}

int main(){
    myPrintf("% world% %\n","Hello",'!',2011);
}
```

Explanation

The `printf` function uses variadic templates to implement a type-safe `printf`

variant. The format string is specified and thanks to function template argument deduction, the compiler deduces the types.

The function templates loops (line 13) as long as the format symbol is not equal to `\0`. When the format symbol is not equal to `\0`, two control flows are possible. First, if the format starts with `%` (line 14), the first argument value is displayed, and `myPrintf` is invoked once more but with a new format symbol and an argument less (line 16). Second, if the format string does not start with `%`, the format symbol is just displayed (line 19). The function `myPrintf` (line 5) is the boundary condition for the recursive calls.

Let's study fold expressions in the next lesson.