Calling Functions

ITP 165 – Fall 2015
Week 5, Lecture 1

Problem: Auto Loan Calculator

- In Homework #2, we had several separate parts of the program…
- First it calculates the loan amount
- Then it determines the length of the loan, and so on
- All that code went inside main, and the code started becoming lengthy and maybe confusing
- Ideally, we want to separate the code into several logical parts in our file. This makes it easier to follow and maintain

Functions

- A ***function*** allows us to break up the program into named sub-sections
- Functions often take in one or more values (called ***parameters***) – though they aren't required to
- Functions often give back (or ***return***) a value – though they aren't required to

Using Functions

- You've already used a function, you just didn't know at the time!

- In Homework #2, you had to include the cmath library:

```
#include <cmath>
```

- And then you used `std::pow`, like this:

```
double result = std::pow(2.0, 5.0);
```

- `std::pow` ***is a function*** (surprise!)
- Using a function is also referred to as ***calling*** a function

The std::pow function

```
double result = std::pow(2.0, 5.0);
```

- It takes two parameters – base and exponent

- It returns the value base$^{\text{exponent}}$

Calling std::pow

Name of function (std::pow)

Zero or more parameters, separated by commas (Must be inside parenthesis!)

```
double result = std::pow(2.0, 5.0);
```

Functions that return values

- A function that returns a value can be used ***anywhere*** that value is valid

- So for example, because this is valid:

```
double value = 6.0;
```

- This is also valid:

```
double value2 = std::pow(2.0, 3.0);
```

Functions that return values, cont'd

- And because this is valid:

```
std::cout << 1000.0 << std::endl;
```

- This is also valid:

```
std::cout << std::pow(10.0, 3.0) << std::endl;
```

Functions that don’t return values

- If a function doesn’t return a value, it can only be used in a statement by itself:

```
// Pretend this is a function that doesn't return
// a value
funcNoReturn(10.0);
```

Other Useful Math Functions

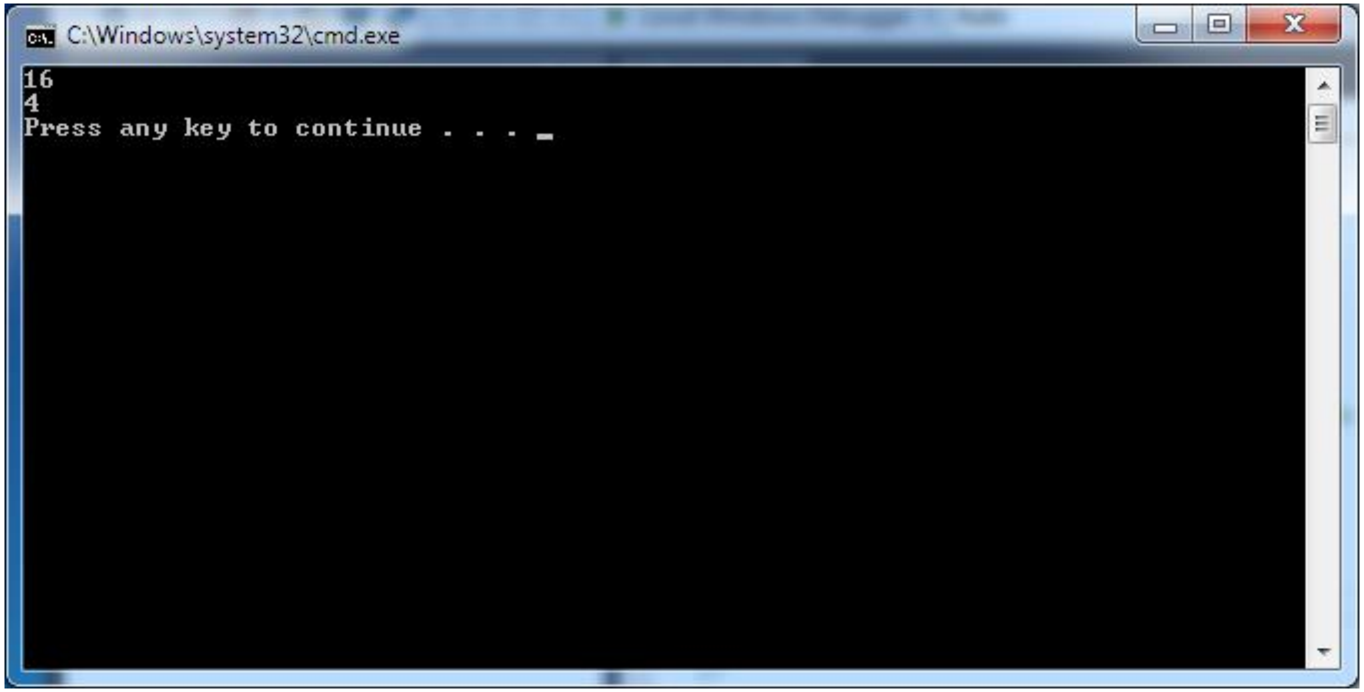
- Here's a list of some of the functions in the `<cmath>` library:

Name	Purpose	Parameter(s)
`std::pow`	Returns base to power of exponent	2 – base and exponent
`std::sqrt`	Returns the square root of the number	1 – a number
`std::cos`	Returns the cosine of the angle	1 – angle (in radians)
`std::sin`	Returns the sine of the angle	1 – angle (in radians)
`std::log10`	Returns base 10 log of number	1 – a number
`std::floor`	Returns the number rounded down to nearest whole number	1 – a number
`std::ceil`	Returns the number rounded up to nearest whole number	1 – a number

Another Math Example

```
// Compute 4 squared
double fourSq = std::pow(4.0, 2.0);

// Compute the square root of previous
// (should be ~4!)
double four = std::sqrt(fourSq);

std::cout << fourSq << std::endl;
std::cout << four << std::endl;
```

Another Math Example, Cont'd



```
16
4
Press any key to continue . . .
```

Functions that return values, cont'd

- Since this is valid:

```
double four = std::sqrt(fourSq);
```

- That also means that this is valid:

```
double four = std::sqrt( std::pow(4.0, 2.0) );
```

Using a Function as a Parameter to Another

- When dealing with a situation like this:

```
double four = std::sqrt( std::pow(4.0, 2.0) );
```

- The `std::sqrt` function can’t run until after `std::pow` function runs, since the value we are taking the square root of depends on the value that `std::pow` returns

- This is handled transparently for you – it knows that it has to call `std::pow` first in this situation

C-Style String Functions

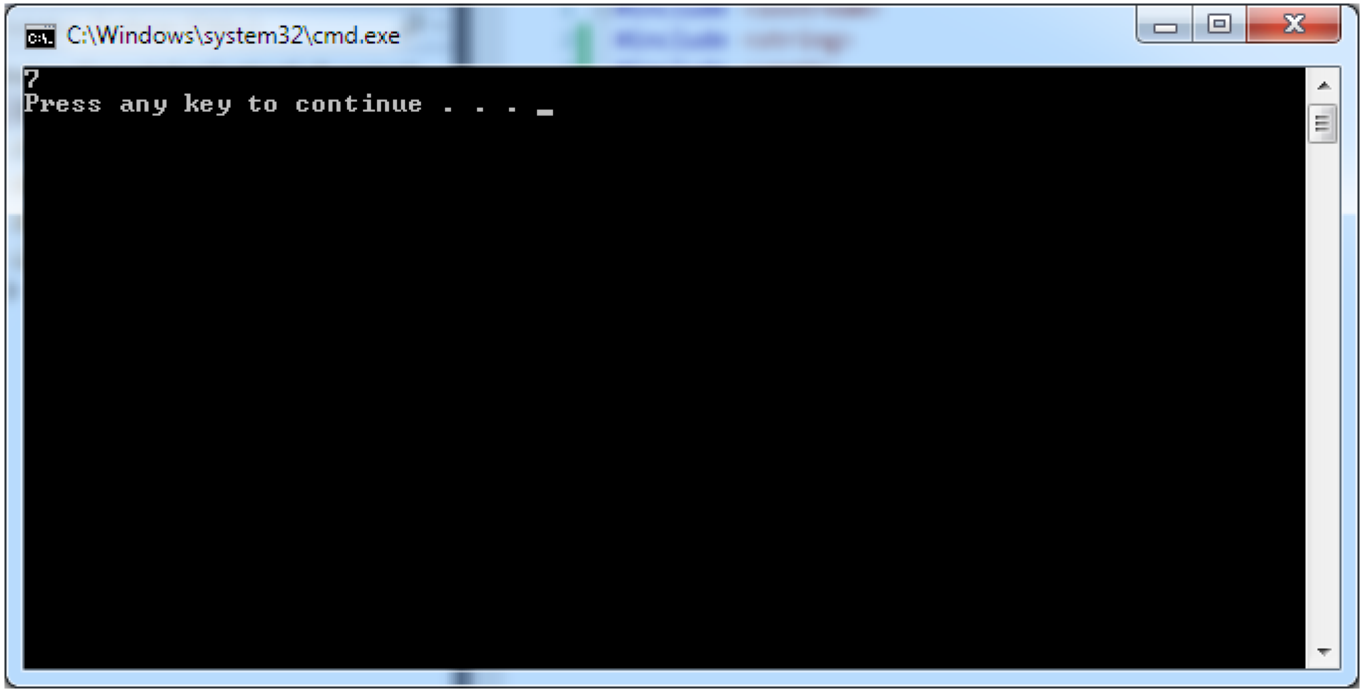
- Functions that can operate on C-style strings are in the `<cstring>` library

- For now, just one function – `std::strlen`
  - One parameter -- the C-style string
  - Returns an `int` with the number of characters in the string (not counting the null terminator)

std::strlen Example

```
char str[] = "Testing";
int x = std::strlen(str);
std::cout << x << std::endl;
```

std::strlen Example in Action

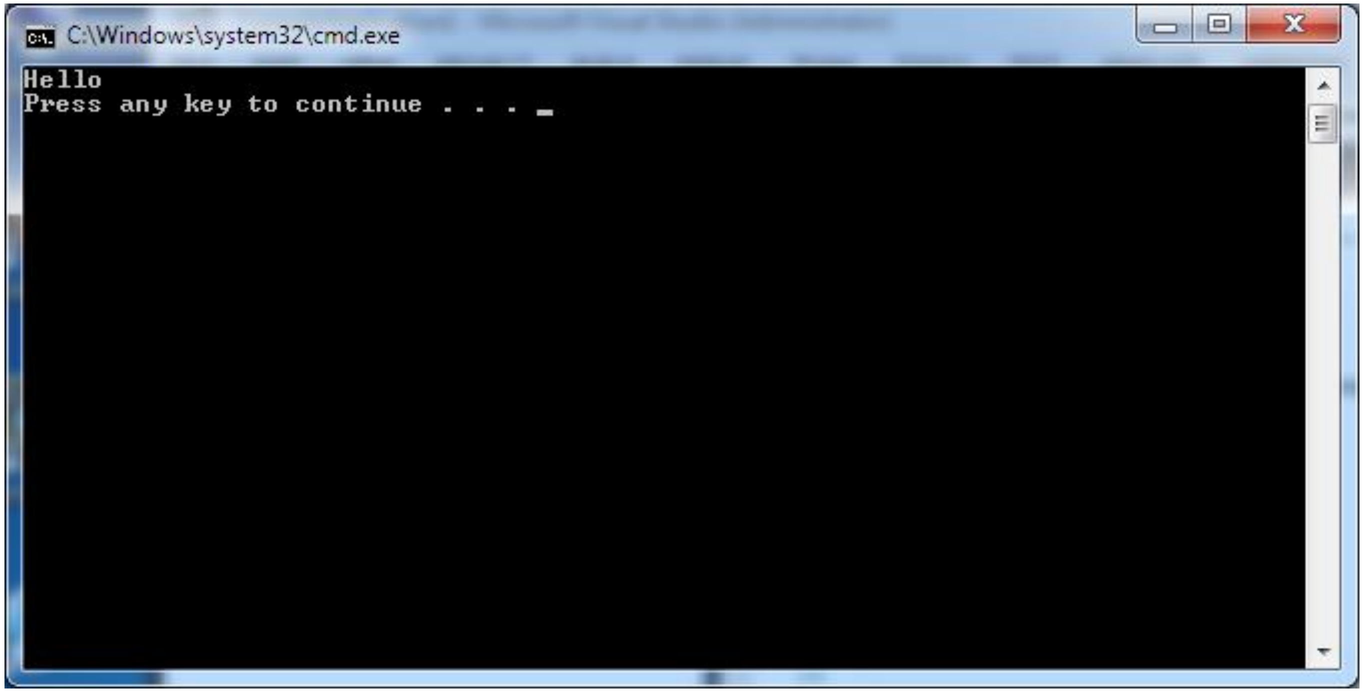


```
7
Press any key to continue . . .
```

A Program w/ a Function

```
#include <iostream>

void SayHello()
{
   std::cout << "Hello" << std::endl;
}

int main()
{
   SayHello();

   return 0;
}
```

In Action



```
Hello
Press any key to continue . . . _
```

Function Declaration

- A ***function declaration*** is what defines a custom function
- In this first sample, our function declaration is:

```
void SayHello()
{
    std::cout << "Hello" << std::endl;
}
```

- Declarations for custom functions are ***outside*** of main

Function Declaration Syntax

The type of value this function returns (void means the function does not return a value)

Name of function (in this case, SayHello)

Parameters, in parenthesis (this function takes no parameters)

Body of function, in braces

```
void SayHello()
{
    std::cout << "Hello" << std::endl;
}
```

Function Declaration Syntax

- Or in other words, we could say that this code:

```
void SayHello()
{
    std::cout << "Hello" << std::endl;
}
```

- Declares a function `SayHello` that:
  - Does not have any parameters
  - Does not have a return value
  - When you call the function, it outputs “Hello”

Commenting Function Declarations

- It is a good coding practice to always put comments right before the declaration of a function
- For example:

```
// Function: SayHello
// Purpose: Outputs "Hello" to cout.
// Parameters: None
// Returns: Nothing
void SayHello()
{
    std::cout << "Hello" << std::endl;
}
```

Function Declaration Order

- Functions must be declared ***before*** any functions they are used in*
- So in our example:

```
void SayHello()
{
   std::cout << "Hello" << std::endl;
}

int main()
{
   SayHello();

   return 0;
}
```

SayHello has to be declared before main, since we use it in main

If You Declare Out of Order…

```
int main()
{
    SayHello();

    return 0;
}

void SayHello()
{
    std::cout << "Hello" << std::endl;
}
```

Error!

It will say that it doesn't know what `SayHello` is at this line.

Using Custom Functions

- Now that we know how to declare a custom function, let's see it in action!
- (To save space I omit the SayHello comments, but they should be there!)

A Program w/ a Function, Step by Step

```
#include <iostream>

void SayHello()
{
    std::cout << "Hello" << std::endl;
}

int main()
{
    SayHello();

    return 0;
}
```

Program still starts at main

A Program w/ a Function, Step by Step

```
#include <iostream>

void SayHello()
{
   std::cout << "Hello" << std::endl;
}

int main()
{
   SayHello();

   return 0;
}
```

We have a call to the `SayHello` function

A Program w/ a Function, Step by Step

```
#include <iostream>

void SayHello()
{
    std::cout << "Hello" << std::endl;
}

int main()
{
    SayHello();

    return 0;
}
```

When the SayHello call happens, main is put on "pause" and we start running the body of SayHello

A Program w/ a Function, Step by Step

```
#include <iostream>

void SayHello()
{
   std::cout << "Hello" << std::endl;
}

int main()
{
   SayHello();

   return 0;
}
```

Just a normal cout

A Program w/ a Function, Step by Step

```
#include <iostream>

void SayHello()
{
    std::cout << "Hello" << std::endl;
}

int main()
{
    SayHello();

    return 0;
}
```

We reached the end of SayHello

A Program w/ a Function, Step by Step

```
#include <iostream>

void SayHello()
{
    std::cout << "Hello" << std::endl;
}

int main()
{
    SayHello();

    return 0;
}
```

When `SayHello` ends, we "resume" `main`, right after the point where `SayHello` was called

A Program w/ a Function, Step by Step

```
#include <iostream>

void SayHello()
{
    std::cout << "Hello" << std::endl;
}

int main()
{
    SayHello();

    return 0;
}
```

The return in main means the program is over!

std::getline()

- For Homework 4, you'll need another function to get more robust input:

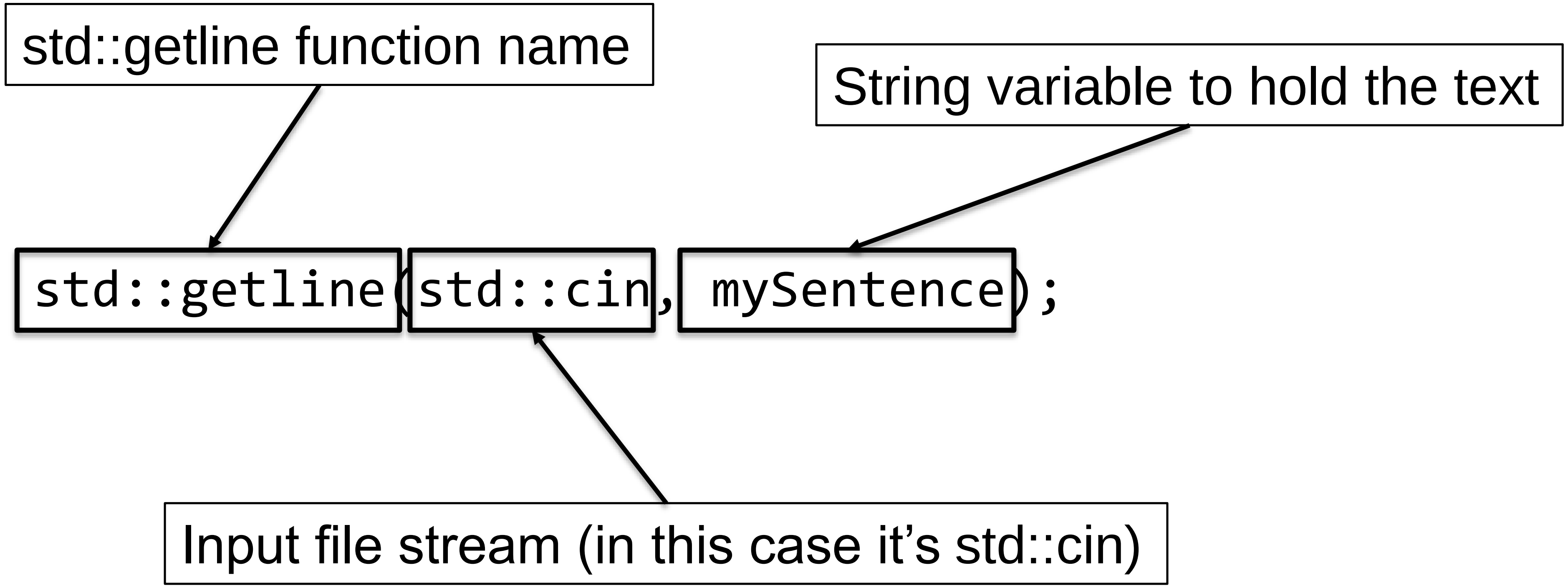
```
#include <iostream>
#include <string>
int main()
{
    std::string mySentence;
    std::cout << "Please enter a sentence: ";
    std::getline (std::cin, mySentence);

    return 0;
}
```

This will take in input that includes white space!

std::getline()

- Previously, we used `std::cin` to get input from the user
- But we were limited to single words (like in our cipher lab)
- Anything after the space character was not included, and it caused some very strange errors in our code
- Now, we have the ability to get an entire line of text (until the user presses the return key)

std::getline()



```
std::getline(std::cin, mySentence);
```

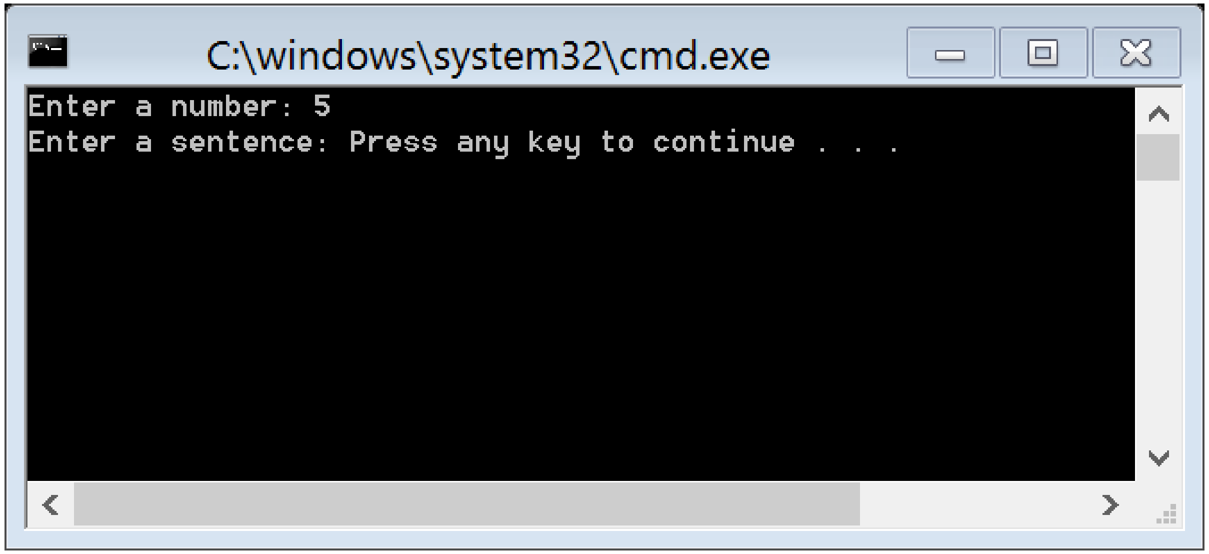
std::getline function name
String variable to hold the text
Input file stream (in this case it’s std::cin)

std::getline()

- If we use `std::cin` and then use `std::getline`, we get issues

```
int myNum;
std::string mySentence

std::cout << "Enter a number: ";
std::cin >> myNum;
std::cout << "Enter a sentence: ";
std::getline(std::cin, mySentence);
```

std::getline()



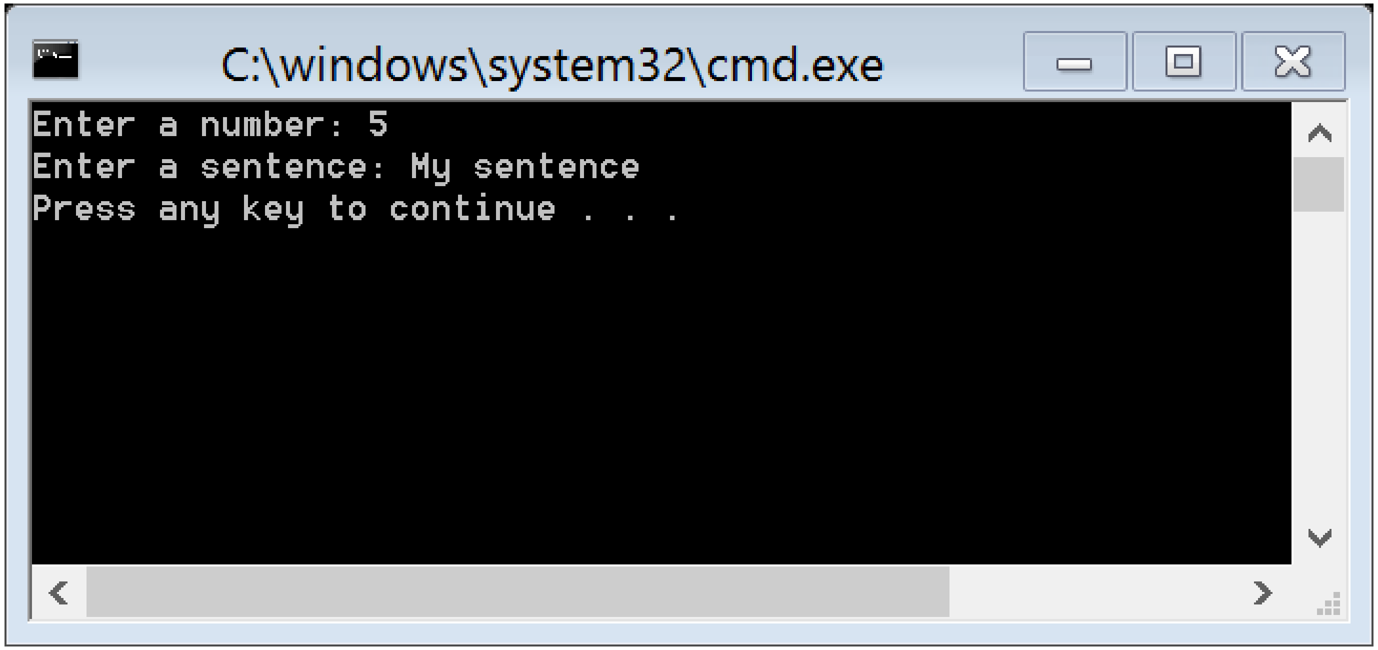
```
Enter a number: 5
Enter a sentence: Press any key to continue . . .
```

std::getline()

- In order to make sure we don't get this error, we must clear the "leftovers" from our `std::cin` call

```
int myNum;
std::string mySentence;

std::cout << "Enter a number: ";
std::cin >> myNum;
std::cin.ignore();
std::cout << "Enter a sentence: ";
std::getline(std::cin, mySentence);
```

std::getline()



```
Enter a number: 5
Enter a sentence: My sentence
Press any key to continue . . .
```

Lab Practical #7