++ Assignment

i = i++; is undefined behavior. Undefined means undefined – anything is just as good as anything. I will probably mention sequencing points at some time. Remember, false implies true.

See https://stackoverflow.com/questions/4968854/is-i-i-truly-a-undefined-behavior

```
//This onne's really a matter of preference of compiler, but I include it
//simply to show that it causes compatibility issues. The i = i++ line
//increments i by 1 in visual studio, but does not increment i in g++.

#include <iostream>

int main() {
	int i = 1;
	std::cout << "i before incrementation attempt: " << i << std::endl;
	i = i++;
	std::cout << "i after incrementation attempt: " << i << std::endl;

	system("pause");
	return 0;
}
```

AssumedNamespace

```
//Difference: gcc and g++ compilers fail on Rectangle.cpp lines 4 and 5.
//It appears that visual studio assumes the std namespace in files that
//include iostream for many (but not all) objects.
```

There just isn't enough information here. The abs function is not in the std namespace, so something else is going on.

Compute absolute value

Returns the *absolute value* of *x*: |*x*|.

These convenience `abs` overloads are exclusive of C++. In C, abs is only declared in <stdlib.h> (and operates on `int` values).

Since C++11, *additional overloads* are provided in this header (`<cmath>`) for the integral types: These overloads effectively cast *x* to a `double` before calculations (defined for `T` being any *integral type*).

http://www.cplusplus.com/reference/cmath/abs/

https://msdn.microsoft.com/en-us/library/zh80x809.aspx

```
#include "Rectangle.h"
#include <iostream>

int main() {
	Rectangle rec(-1, -2, 2, 3);
	std::cout << rec.getArea() << std::endl;
	system("pause");

	return 0;

}
```

```
#include "Rectangle.h"

int Rectangle::getArea() const {
	int length = abs(p2->x - p1->x);
	int height = abs(p2->y - p1->y);
	return length * height;
}

#pragma once
#include <cmath>

class Rectangle {
private:
	struct Point {
		int x, y;
		Point(int x, int y) {
			this->x = x;
			this->y = y;
		}
	};
	Point* p1;
	Point* p2;

public:
	Rectangle(int x1, int y1, int x2, int y2) {
		p1 = new Point(x1, y1);
		p2 = new Point(x2, y2);
	}
	~Rectangle(){
		if(p1) delete p1;
		if(p2) delete p2;
	}
	int getArea() const;
	int getSize();
};

#pragma once

class Shape {
	virtual int getSize() const = 0;
};
```

ExtraQualifierInHeader

Interesting. This is called class name injection and is legal. However, it's usually g++ and gcc that fail, not VS.
See https://stackoverflow.com/questions/31769853/c-class-name-injection and
https://stackoverflow.com/questions/25549652/why-is-there-an-injected-class-name

Class Name Injection. The C++ standard says that the **name** of a **class** is "**injected**" into the **class** itself. This is a change from earlier C++ rules. Formerly, the **name** of the **class** was not found as a **name** within the **class**.

```
//Difference: g++ and gcc fail to compile on Rectangle.h line 18 because
//of the extra "Rectangle::" qualifier before the function name.


#include "Rectangle.h"
#include <iostream>
```

```
int main() {
	Rectangle rec(-1, -2, 2, 3);
	std::cout << rec.getArea() << std::endl;
	system("pause");

	return 0;

}
```

```
#include "Rectangle.h"

int Rectangle::getArea() const {
	int length = std::abs(p2->x - p1->x);
	int height = std::abs(p2->y - p1->y);
	return length * height;
}
```

```
#pragma once
#include <cmath>
#include "Shape.h"

class Rectangle : public Shape{
private:
	struct Point {
		int x, y;
		Point(int x, int y) {
			this->x = x;
			this->y = y;
		}
	};
	Point* p1;
	Point* p2;

public:
	Rectangle::Rectangle(int x1, int y1, int x2, int y2) {
		p1 = new Point(x1, y1);
		p2 = new Point(x2, y2);
	}
	int getArea() const;
	int getSize() const { return 0;};
};
```

Initializations:

Again, undefined means undefined. All value is valid for an undefined variable. If you are lucky, you'll crash on reference.

```
//Difference: g++ outputs 0 and visual studio outputs a large negative
//number. Not necessarilly wrong, but can cause compatibility difficulties.

#include <iostream>

int main() {

	int* array = new int[5];
	std::cout << "array at 0: " << array[0] << std::endl;

	system("pause");
```

```
	return 0;
}
```

SystemPause:

system(“pause”) is a platform-specific hack and students will lose style points if they include it in their files. A better idea is to put a breakpoint at the end of their program. With that said, we will be using command line arguments for input and output, so this should be a non-issue.

https://stackoverflow.com/questions/1107705/systempause-why-is-it-wrong

```
//Difference:g++ compiles but gives an error message when it arrives at the system pause.
//gcc fails to compile all together.

#include <iostream>


int main() {
	std::cout << "Hello World." << std::endl;

	system("pause");
	return 0;
}
```

IntelliSense engine