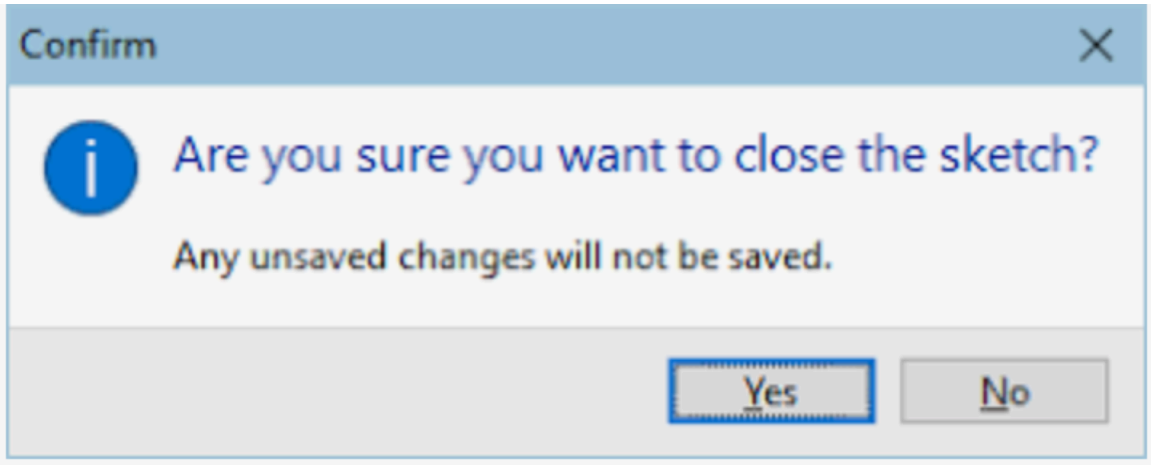
Programming Exercise Exception

In this exercise, you'll implement a guarded switch in software using C++ exception handling. Dangerous machines included guarded switches to ensure that they are not activated accidentally.

Here's one physical implementation of a guarded switch:

You first lift the cover of the switch up, and then you can press the toggle.

Here's another example, this time a software guarded switch:



In this assignment, you'll begin with a working but incomplete program that doesn't protect the device from being activated. Download the file guard_incomplete.cpp. The program is listed below.

```
#include <iostream>
#include <stdexcept>

class Device {
public:
    Device();
    virtual ~Device();
    void enable();
    void activate();

private:
    bool m_bEnabled;
};

Device::Device() : m_bEnabled(false) { } // Default Constructor
Device::~Device() { }                    // Default Destructor

void Device::enable()
{
    m_bEnabled = true;
}

void Device::activate()
{

    // Simulate activing the device.
    // TODO: Replace this with a try catch block
    std::cout << "Device is now activated!" << std::endl;

    return;
}

int main(int argc, const char * argv[]) {
    Device device;
    device.activate();
    return 0;
}
```

Build and compile the program.

The output will appear as follows:

```
Device is now activated!
```

The device is unconditionally activated. We want to make the activation a two-step process. We want to first enable the device, and then activate it.

Our program already includes a function to enable the device, but we're not using it. It sets member variable m_bEnabled to true, indicating the device is enabled.

```
void Device::enable()
{
    m_bEnabled = true;
}
```

We're also not checking if the device is enabled before activating it.

To do this we'll add the following try block in place of the single cout statement in the activate method.

Type in the following code into the activate method:

```
try
{
    // Precondition
    // Must be enabled before activated
    if (!m_bEnabled)
    {
        throw std::runtime_error("Device must be anabled first!");
    }

    std::cout << "Device is now activated!" << std::endl;

} catch(std::runtime_error ex)
{
    std::cout << ex.what() << std::endl;
}
```

When you're done, the function should appear as follows:

```
void Device::activate()
{
    try
    {
        // Precondition
        // Must be enabled before activated
        if (!m_bEnabled)
        {
            throw std::runtime_error("Device must be anabled first!");
        }

        std::cout << "Device is now activated!" << std::endl;

    } catch(std::runtime_error ex)
    {
        std::cout << ex.what() << std::endl;
    }
    return;
}
```

Now, run the program.

You should see the following:

```
Device must be anabled first!
```

Now, change the main function to call the enable method as follows:

```
int main(int argc, const char * argv[]) {
    Device device;
    device.enable();
    device.activate();
    return 0;
}
```

Build and run the program again.

```
Device is now activated!
```

The device can only be activated with a two-step process: enable, then activate.

Appendix: guarded_incomplete.cpp

In case you need it, here is the source for guarded_incomplete.cpp; you can copy it.

```
#include <iostream>
#include <stdexcept>

class Device {
public:
    Device();
    virtual ~Device();
    void enable();
    void activate();

private:
    bool m_bEnabled;
};

Device::Device() : m_bEnabled(false) { } // Default
Constructor
Device::~Device() { }                    // Default
Destructor

void Device::enable()
{
    m_bEnabled = true;
}

void Device::activate()
{

    // Simulate activing the device.
    // TODO: Replace this with a try catch block
    std::cout << "Device is now activated!" << std::endl;

    return;
}

int main(int argc, const char * argv[]) {
    Device device;
    device.activate();
    return 0;
}
```