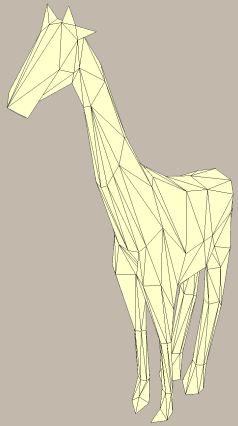


Lecture 03 - Templates

Philip Caplan

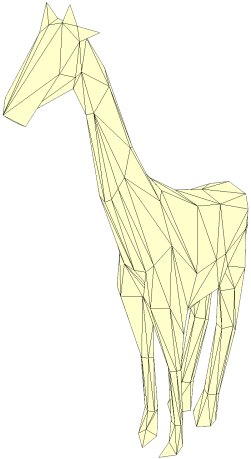
CSCI 0422 - Geometric Modeling (Spring 2022)

Middlebury

Learning objectives

By the end of this lecture you will be able to:

- use `CMake` to automatically generate Makefiles for `C++` projects,
- inherit from a base class,
- create a smart pointer,
- use templates to re-use functions & classes in which the computation, or underlying data is the same, but the type of the data is different
- practice by writing your own `Matrix` class.



Socrative quiz

cs422-class03

Save and Exit

Align Quiz to Standard

Share

1. Consider the following C++ code:

```
int value1 = 5;
int value2 = 8;

int *x = &value1;
const int* y = &value1;
int* const z = &value1;
```

Which of the following lines (immediately following this block of code) would result in a compiler error?

A. *x = value2;

B. x = &value2;

C. y = &value2;

D. *y = value2;

E. z = &value2;

F. *z = value2;

Getting started

- Pick a partner (person sitting closest to you, but try to work with someone new today).
- Open VS Code and click on Terminal.
- Navigate to where you want to store the exercises for today.
- Clone the repository using:

```
$ git clone https://gitlab.com/csci422-s22/class03.git class03
```

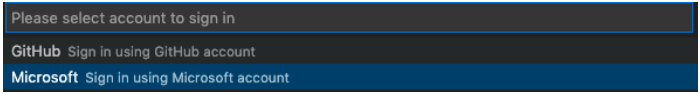
- Build and run:

```
$ cd class03
$ mkdir build
$ cd build
$ cmake ../
$ make
```

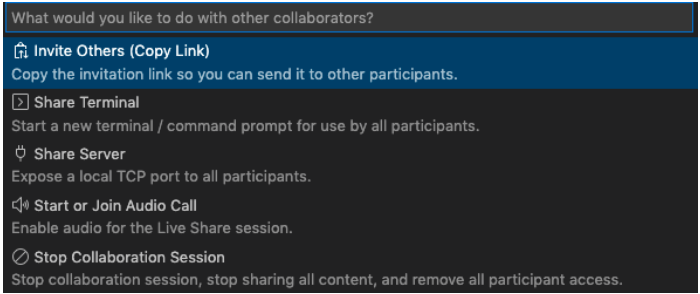
With the person sitting next to you...

In VS Code:

- Click on the Live Share button at the bottom-left.



- Sign in with **Middlebury email address even if you have a GitHub account**.
- Your name should then appear at the bottom left where it previously said Live Share.
- Click on your name, you should see this:



- Person 1: Send the the invite link to Person 2 (via private message in Slack).

Inheriting from a base class.

```
class Shape {
public:
  // shape is an abstract class because it has a purely virtual function
  virtual double area() const = 0;

  const std::string& name() const { return name_; }

protected:
  std::string name_;
};

class Rectangle : public Shape {
public:
  Rectangle( double length , double width ) :
    length_(length),
    width_(width) {

    name_ = "rectangle";
  }

  double area() const {
    return length_*width_;
  }
private:
  double length_, width_;
};
```

Using smart pointers.

```
    // uncommenting the following creates a compiler error
    // because Shape is an abstract class since area() is a pure virtual function
    //Shape shape;

    Circle circle(2.0);
    std::cout << circle.name() << " has area " << circle.area() << std::endl;

    // example of using a unique_ptr
    std::unique_ptr<Rectangle> r_ptr = std::make_unique<Rectangle>(10,20);
    Rectangle& r = * r_ptr.get();

    std::cout << r.name() << " has area " << r.area() << std::endl;
    std::cout << r_ptr->name() << " has area " << r_ptr->area() << std::endl;

    // r_ptr is responsible for the memory associated with Rectangle, and will
    // delete the memory when it goes out of scope
```

Templating a function.

Adding two numbers together:

```
int
add( int x , int y ) {
  return x + y;
}
```

```
double
add( double x , double y ) {
  return x + y;
}
```

```
template<typename T>
T
add( T x , T y ) {
  return x + y;
}
```

Implementing a class for *N*-dimensional vectors.

```
template<int N>
class Vector {
public:
  Vector();                        // constructor that will set vector to 0
  Vector( double* x );             // constructor from an array
  Vector( const Vector<N>& v ); // copy constructor

  double  operator() (int i ) const;
  double& operator() (int i );

  void print() const;

private:
  double data_[N];
};
```

Implementing a class for *N*-dimensional vectors.

```
template<int N>
Vector<N>::Vector() {
  for (int i = 0; i < N; i++) {
    data_[i] = 0;
  }
}

template<int N>
double&
Vector<N>::operator() (int i) {
  assert( i >= 0 && i < N );
  return data_[i];
}

template<int N>
void
Vector<N>::print() const {
  for (int i = 0; i < N; i++)
    std::cout << "entry[" << i << "] = " << data_[i] << std::endl;
}
```

Using our `Vector` class.

```
Vector<3> v;
v(0) = 2;

v.print();
```

Using our `Vector` class.

```
Vector<3> v;
v(0) = 2;

v.print();
```

why doesn't this compile???

Using our `Vector` class.

```
Vector<3> v;
v(0) = 2;

v.print();
```

why doesn't this compile???

instantiate your templates!

```
template class Vector<3>;
```

Using our `Vector` class.

```
Vector<3> v;
v(0) = 2;

v.print();
```

why doesn't this compile???

instantiate your templates!

```
template class Vector<3>;
```

What if you want your class/function to be implemented differently for a certain type/value of your templates?

Template specialization.

For example, if we want a 3d vector to be displayed inline, such as `(x,y,z)`.

```
template<>
void
Vector<2>::print() const {
  std::cout << "vec2d = (" << data_[0] << "," << data_[1] << ")" << std::endl;
}

template<>
void
Vector<3>::print() const {
  std::cout << "vec3d = (" << data_[0] << "," << data_[1] <<  "," << data_[2] <<
      ")" << std::endl;
}
```

Exercise 1: create two 2d vectors `u` and `v` and print out their contents.

write a function that adds two vectors, returning a third vector

Exercise 2: implement a class that represents a matrix.

```
template<int M, int N, typename T>
class Matrix {

public:
  // initialize all elements to 0
  Matrix();

  // const access to entry in row i, column j to retrieve entry
  const T& operator() (int i , int j) const;

  // non-const access to entry in row i, column j to modify entry
  T& operator() (int i , int j);

  // print out the contents of this matrix
  void print() const;

private:
  // how will you store the data?
};
```

- how will you store the data?
- complete the functions directly in `matrix.h` (do not create a `matrix.cpp`).
- complete overloaded call operators to retrieve/modify entries.
- **challenge:** write a function that overloads the + operator to add two matrices (returning a third matrix).
- **extra challenge:** write a function that overloads the $*$ operator to multiply two matrices (of any size).

Before next class...

- Please complete the partner form:

  https://forms.gle/QYXbL2URvidLvMyG9

- Install `gdb`, `lcov` and optionally `valgrind`:
  - **Docker**: you should be all set.
  - **Linux/WSL**: `$ sudo apt-get install gdb lcov valgrind`
  - **OS X**: `$ brew install lcov`. You can try to install `valgrind` as well but I have had issues installing `valgrind` on OS X in the past (maybe things have changed). You can always work from a Docker container since our Docker container will have `valgrind` installed.
- Please come to office hours if you are having trouble installing stuff.
  Tuesdays 1:30pm-3:00pm, Thursdays 11am-12pm, Fridays 11am-12pm.
- Remember to complete Project0 by tonight at 11:59pm.