PHY 250

Andrew Diggs

Winter 2024, Week 3

C++: Objects-Recap

Two main types of objects in C++, struct and class.They are identical.

Three access qualifiers private, protected, and public.

```
1  class MyClass {
2  private:
3
4  public:
5
6  };
```

C++: Objects

```
1  class Engine {
2  private:
3
4  public:
5      Engine();
6      ~Engine();
7
8  };
```

The **Engine()** is the constructor, This is the code we want to run we we create a new Engine instance.
The ~**Engine()** is the destructor, This is the code we want to run we we destroy an Engine instance.

C++: Objects

```
1  class Engine {
2  private:
3      float RPM;
4  public:
5      Engine();
6      ~Engine();
7
8      void Set_RPM(float HowMuch);
9      float Get_RPM();
10 };
```

The float RPM is a private member. Private members can only be accessed by the class instance.
We can safely access private members with public Getters and Setters.

C++: Objects

Last class we wrote a class for a 1D float array called fArray.

```
1  class fArray{
2  private:
3      int m_num_el = 0;
4      float* m_vals = NULL;
5      bool is_init = false;
6  public:
7      fArray();
8      ~fArray();
9
10     void INIT(int num);
11     float* Get_Vals();
12     int Get_Num();
13 };
```

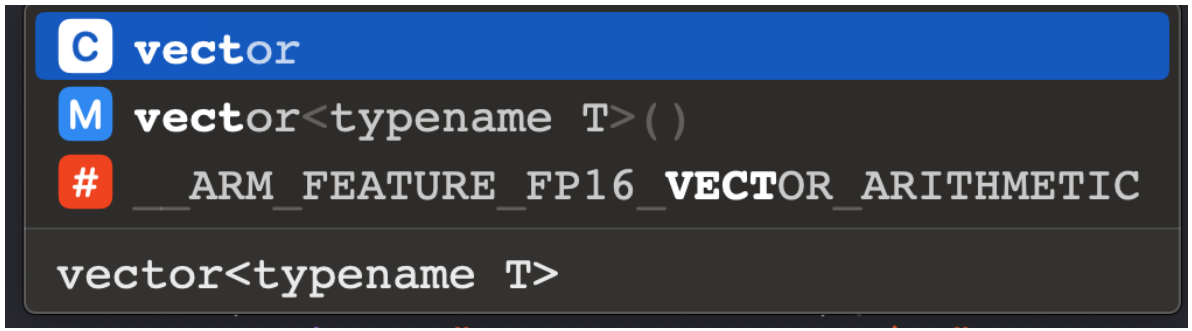
C++: Objects

A more common name for this class would be a vector.

```
1  class vector{
2  private:
3      int m_num_el = 0;
4      float* m_vals = NULL;
5      bool is_init = false;
6  public:
7      vector();
8      ~vector();
9
10     void INIT(int num);
11     float* Get_Vals();
12     int Get_Num();
13 };
```

C++: Objects

With this class name auto-complete gives me

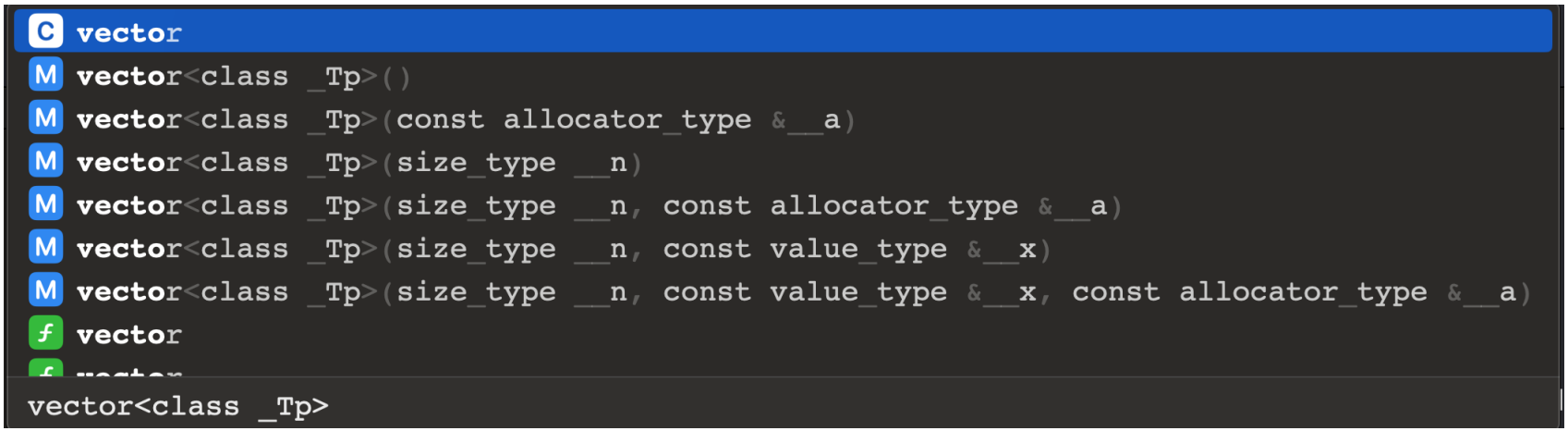


C++: Objects

However, if I include these lines in my main file

```
1 #include <vector>
2 using namspace std;
```

I get



This would get very confusing

C++: Namespaces

We can fix this problem by using a namespace.

```
1  namespace PHY{
2
3  class vector{
4  private:
5      int m_num_el = 0;
6      float* m_vals = NULL;
7      bool is_init = false;
8  public:
9      vector();
10     ~vector();
11     void INIT(int num);
12     float* Get_Vals();
13     int Get_Num();
14 };
15 }
```

C++: Namespaces

By encapsulating our vector class in a namespace it has a unique definition so it won't be confused with any other definition of vector. More importantly it wont override any other defined objects.

```
1  int main(){
2
3      PHY::vector vec1;
4      std::vector vec2;
5      vec1 != vec2;
6
7  return 0;
8  }
```

C++: Lab 3

1. Add the following to your MyClass.hpp file.

```
1     fArray();
2     fArray(int num);
3     ~fArray();
```

And in your .cpp file

```
1 fArray::fArray(int num){
2     INIT(num);
3 }
```

C++: Lab 3

4. In your myclass.hpp define a namespace with whatever name you choose. Keep in mind you will have to type this many times.
5. Encapsulate your fArray class in your namespace.
6. rename your fArray class to vecN, and add the following to your .hpp file.
7. Go through your myclass.cpp and change vecN → <your namespace>::vecN ;
8. Create an instance using the new constructor of <your namespace>::vecN in main to make sure it works.

```
1  int main(){
2      int rows = 10;
3      YourNameSpace::vecN vec(rows);
4  return 0;
5  }
```

C++: Operators

Something we have all used many times

```
1  int main(){
2
3      int a = 3;
4      int b = 1;
5      int c = a + b;
6
7  return 0;
8  }
```

Have you ever asked yourself what this really means?

C++: Operators

In C/C++ we need to declare a type for all variables, why?

```
1  int main(){
2
3      int a = 3;
4
5  return 0;
6  }
```

int a; what does this do?

1. find 4 bytes of continuous memory and assign it to a.
2. define the type assigned to this block of memory to data type int .

Both of these are important but I want to focus on the second.

C++: Operators

The reason the data type assignment is important is due to operations.

```
1  int main(){
2
3      int a = 3;
4      int b = 1;
5      int c = a + b;
6
7  return 0;
8  }
```

Let's look at the last line of code.

C++: Operators

int c = a + b;
We created a block of 4 bytes with data type int with variable reference "c".
Because c is an int, using the = operator means that we telling the compiler to copy the value of the int on the RHS of the = to the location of c.
This restricts what we can have on the RHS.

C++: Operators

```
1  int main(){
2
3      char val1[10] = "3";
4      char val2[10] = "1";
5      int val3 = val1 + val2;
6
7  return 0;
8  }
```

This is an error because val1 + val2 does not return an int .

C++: Operators

The + operator preforms different actions depending on the data type it is operating on.

```
1  int rows = 10;
2  int cols = 10;
3  int sum1 = rows + cols;
4
5  std::string val1 = "10";
6  std::string val2 = "10";
7  std::string sum2 = val1 + val2;
8
9  char a = '1'; char b = '2';
10 char sum3 = a + b;
```

sum1 = 20
sum2 = 1010
sum3 = c

C++: Operators + Objects

In C++ all user defined classes, e.g. our vector class, have one operator predefined, =.
This is called the copy constructor, it allows us to copy over the contents of one class instance to another.

```
1  int main(){
2      int a = 10;
3      int b = a;
4
5      PHY::vector vec1;
6      PHY::vector vec2 = vec1;
7
8  return 0;
9  }
```

C++: Operators + Objects

```
1  int main(){
2      int a = 10;
3      int b = a;
4
5      PHY::vector vec1;
6      PHY::vector vec2 = vec1;
7
8  return 0;
9  }
```

C++: Operators + Objects

Copy over the following into your main() replacing PHY with <your namespace>;

```
1  int main(){
2      int rows = 10;
3      PHY::vecN vec(rows);
4      float* vals = vec.Get_Vals();
5      for(int i=0; i<rows; i++){
6          printf("vec[%d] = %.1f\n",i,vals[i]);
7      }
8  return 0;
9  }
```

C++: Operators + Objects

You should get the following output;

$\text{vec}[0] = 0.0$
$\text{vec}[1] = 1.0$
$\text{vec}[2] = 2.0$
$\text{vec}[3] = 3.0$
$\text{vec}[4] = 4.0$
$\text{vec}[5] = 5.0$
$\text{vec}[6] = 6.0$
$\text{vec}[7] = 7.0$
$\text{vec}[8] = 8.0$
$\text{vec}[9] = 9.0$

C++: Operators + Objects

Now insert the additional part as shown;

```
1  int main(){
2      int rows = 10;
3      PHY::vecN vec(rows);
4      {
5          vec = PHY::vecN(rows);
6      }
7      float* vals = vec.Get_Vals();
8      for(int i=0; i<rows; i++){
9          printf("vec[%d] = %.1f\n",i,vals[i]);
10     }
11 return 0;
12 }
```

C++: Operators + Objects

Here is what I get;

vec[0] = -4270243677537828864.0
vec[1] = 0.0
vec[2] = 0.0
vec[3] = 0.0
vec[4] = 0.0
vec[5] = 0.0
vec[6] = 0.0
vec[7] = 0.0
vec[8] = 0.0
vec[9] = 0.0

This is VERY BAD!!!!.

C++: Operators + Objects

In the line

```
1        vec = PHY::vecN(rows);
```

On the RHS we are creating a new instance of vecN that has a float* that gets dynamically allocated using new . Then when the scope ends, i.e. '}', that instance of vecN gets destroyed and the float * is delete .
When we copy the new vecN to our vec using the = operator we are assigning the float * of the new vecN to our vec.
Which means that we we delete the float * of the new vecN we are also deleting the float * that is assigned to old vec.
So then what happens to the memory we allocated for the first vecN?... **NOTHING!** it remains on the heap and does not get released until the program exits.

C++: Operators + Objects

This issue needs to be fixed. Let's redefine the = operator for our vecN class. As a public member add the following to the .hpp file.

```
vecN& operator=(const vecN& other);
```

and the in your .cpp file

```
PHY::vecN& PHY::vecN::operator=(const PHY::vecN& other){
    this->m_num_el = other.m_num_el;
    this->INIT(m_num_el);

    for(int i=0; i<m_num_el; i++) {
        this->m_vals[i] = other.m_vals[i];
    }
    return *this;
}
```

C++: Operators + Objects

This should have fixed the problem we had earlier. vec[0] = 0.0
vec[1] = 1.0
vec[2] = 2.0
vec[3] = 3.0
vec[4] = 4.0
vec[5] = 5.0
vec[6] = 6.0
vec[7] = 7.0
vec[8] = 8.0
vec[9] = 9.0

Looks Good!

C++: Operators + Objects = Power

This is the Power of C++. we get complete control over how we want operators to defined for our data types.

lets add a few more useful operators.

```
1    //Operators to make our lives easy
2    vecN& operator=(const vecN& other);
3    float& operator[](const int index);
4    vecN operator+(vecN& other);
```

C++: Operators + Objects = Power

This is why it works to add numpy arrays but not python lists. This can be used to make our lives very easy

```
1  struct Vec3{
2      float x,y,z;
3      Vec3();
4      Vec3(float s);
5      Vec3(float e_x, float e_y, float e_z);
6
7      Vec3 add(const Vec3& other) const;
8      float& operator[](const int index);
9      float* get();
10
11     float dot(const Vec3& other) const;
12     Vec3 cross(const Vec3& other) const;
13     float len() const;
14
```

C++: Operators + Objects = Power

```
1     Vec3 operator+(const Vec3& other) const;
2     Vec3 operator-(const Vec3& other) const;
3     Vec3& operator=(const Vec3& other);
4     Vec3& operator=(const Quat& other);
5     Vec3 operator*(const Vec3& other) const;
6     Vec3 operator/(float div) const;
7     Vec3 operator+=(const Vec3& other);
8     Vec3 operator*=(float scale);
9
10    bool operator==(const Vec3& other) const;
11    bool Is_Parallel(const Vec3& other) const;
12    void Rotate_Quaternion(const Vec3& axis, float ang);
13    void Reset();
14    void print();
15    void Normalize();
16    void Vround(int decimals);
17
```