- What does **this** represent?
  - When a member function is called, to keep track of which object was called we utilize this.
  - this is a C++ keyword, and every object has access to its own address through a pointer called this.
  - The this pointer is not part of the object itself, but it is passed by the compiler as an implicit argument to each of the object's non static member functions
  - Since every object has access to its own address, this will be a pointer that allows us to have access to that address.
- Can you explain why we would make a function a friend instead of a member function?
  - Allows us to extract functionality from a class and are kept in a non-member function for use by multiple classes.
  - **Multiple classes can use the same function when utilizing friends.**
  - It would be like a library function
  - Member functions can only be used in the class in which they are defined.
- What makes a friend function different from a regular non member function?
  - friends are able to access both public and private data members of a class, whereas, non-members can only access the public data members.
- How do you make a function a friend?
  - put the keyword *friend* in front of the function prototype.
  - We put the function prototype inside of the class

```
class ClassABC
{
  friend void setZ(ClassABC&, int);

  public :
    int getZ() const
    {
      return Z;
    }
  private :
    int Z{0};
};
```

  -
- Operator Overloading
  - Given a class containing an operator overload function as a member function, can you make the necessary changes to move that operator overload function out of the class?

- The first piece of code here shows the operator overload function as a member function.
  - Overloaded < defined as a member function with one parameter

```
class Quarterback
{
    public :
        Quarterback(std::string name, int att, int comp, int yds, int td)
        : qbName{name}, qbAtt{att}, qbComp{comp}, qbYds{yds}, qbTd{td}
        {
        }

        bool operator<(const Quarterback& QB)
        {
            if (qbAtt  < QB.qbAtt  &&
                qbComp < QB.qbComp &&
                qbYds  < QB.qbYds  &&
                qbTd   < QB.qbTd)
                return true;
            else
                return false;
        }

    private :
        std::string qbName;
        int qbAtt;
        int qbComp;
        int qbYds;
        int qbTd;
};
```

- The second section of code show the same operator overload function but OUTSIDE of the class, with different parameters and declared a friend.
  - Overloaded < defined a non member friend function with one parameter

```
class Quarterback
{
    friend bool operator<(const Quarterback& QB1, const Quarterback& QB2);

    public :
        Quarterback(std::string name, int att, int comp, int yds, int td)
        : qbName{name}, qbAtt{att}, qbComp{comp}, qbYds{yds}, qbTd{td}
        {
        }


    private :
        std::string qbName;
        int qbAtt;
        int qbComp;
        int qbYds;
        int qbTd;
};


bool operator<(const Quarterback& QB1, const Quarterback& QB2)
{
    if (QB1.qbAtt  < QB2.qbAtt  &&
        QB1.qbComp < QB2.qbComp &&
        QB1.qbYds  < QB2.qbYds  &&
        QB1.qbTd   < QB2.qbTd)
        return true;
    else
        return false;
}
```