03 - Formal Introduction to OOP

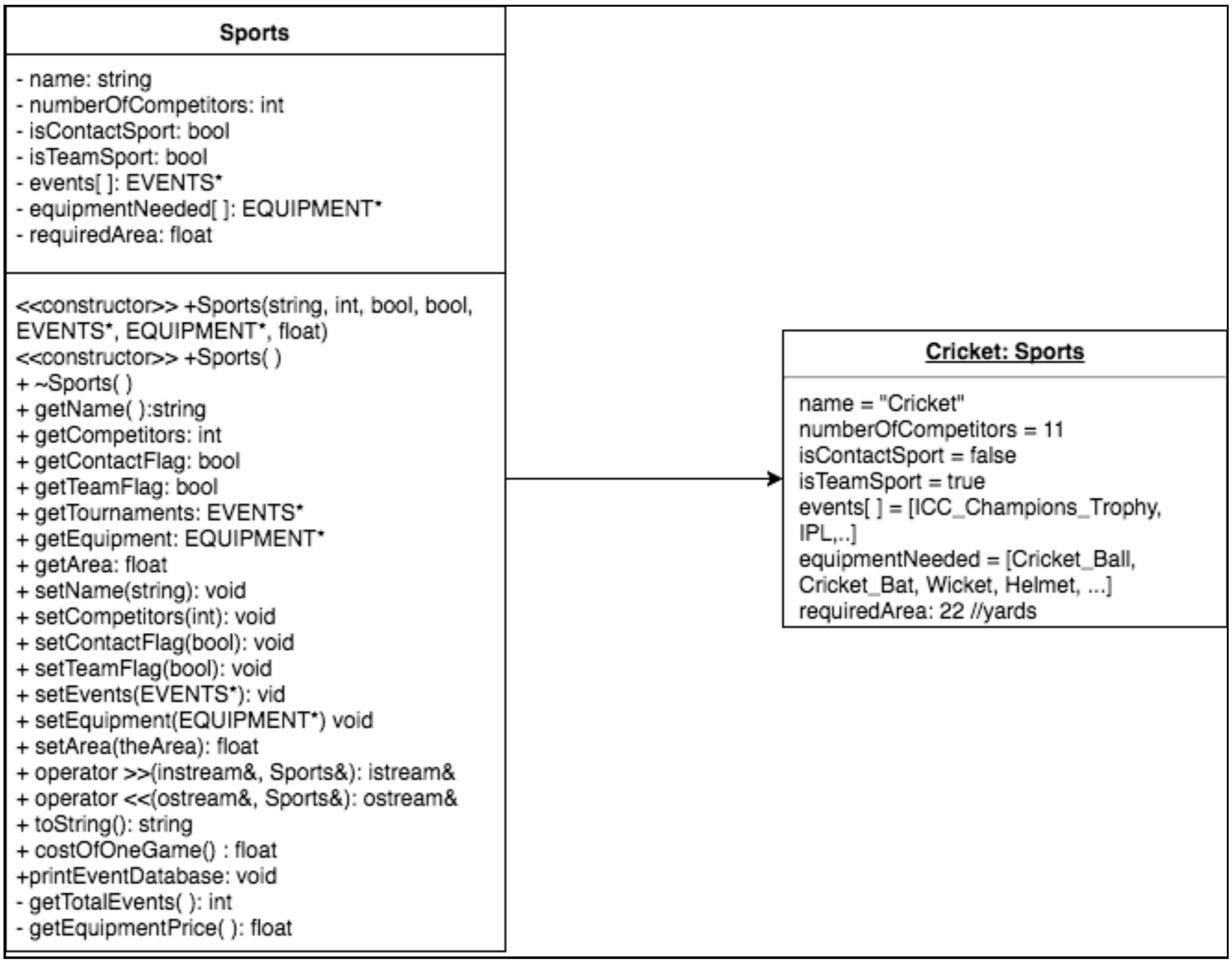
Pre-Lab Exercises

Exercise 1

Considering ‘`Sports`’ as a general class encompassing a variety of games in it, describe the ‘`Cricket`’ object that belongs to this class. You must specify all possible public data members and member functions as well as two example of private data members and member functions.

```
#ifndef Sports_h
#define Sports_h

#include <string>
#include <fstream>
#include <iostream>

class Sports
{
private:
    string      name;
    int         numberOfCompetitors;
    bool        isContactSport;
    bool        isTeamSport;
    Events*     events[];
    Equipment*  equipmentNeeded[];
    float       requiredArea;

    //PRIVATE METHODS - ONLY FACILITATE PUBLIC FUNCTIONS
    int         getTotalEvents();
    float       getEquipmentPrice();
public:
    //CONSTRUCTORS
    Sports(string myName,int theCompetitors, bool contactFlag,
           bool teamFlag Events* theEvents[],
           Equipment* theEquipment[], float theArea);
    Sports();

    //DESTRUCTOR
    ~Sports();

    //GETTERS
    string      getName();
    int         getCompetitors();
    bool        getContactFlag();
    bool        getTeamFlag();
    Events*     getTournaments();
    Equipment*  getEquipment();
    float       getArea();
```

```
 //SETTERS
    void        setName(string theName);
    void        setCompetitors(int theCompetitors);
    void        setContactFlag(bool theContactFlag);
    void        setTeamFlag(bool theTeamFlag);
    void        setEvents(Events∗ myEventsArray);
    void        setEquipment(Equipment∗ myEquipmentArray);
    void        setArea(float theArea);

    //OPERATORS
    friend istream& operator >>(istream& ins, Sports& mySport);
    friend ostream& operator <<(ostream& outs, const Sports& mySport);

    //OTHER METHODS
    string      toString();
    float       costOfOneGame();
    void        printEventDatabase();
}

#endif /∗ Sports_h ∗/
```


Sports
- name: string
- numberOfCompetitors: int
- isContactSport: bool
- isTeamSport: bool
- events[ ]: EVENTS*
- equipmentNeeded[ ]: EQUIPMENT*
- requiredArea: float
<<constructor>> +Sports(string, int, bool, bool, EVENTS*, EQUIPMENT*, float)
<<constructor>> +Sports( )
+ ~Sports( )
+ getName( ):string
+ getCompetitors: int
+ getContactFlag: bool
+ getTeamFlag: bool
+ getTournaments: EVENTS*
+ getEquipment: EQUIPMENT*
+ getArea: float
+ setName(string): void
+ setCompetitors(int): void
+ setContactFlag(bool): void
+ setTeamFlag(bool): void
+ setEvents(EVENTS*): vid
+ setEquipment(EQUIPMENT*) void
+ setArea(theArea): float
+ operator >>(instream&, Sports&): istream&
+ operator <<(ostream&, Sports&): ostream&
+ toString(): string
+ costOfOneGame() : float
+printEventDatabase: void
- getTotalEvents( ): int
- getEquipmentPrice( ): float
Cricket: Sports
name = "Cricket"
numberOfCompetitors = 11
isContactSport = false
isTeamSport = true
events[ ] = [ICC_Champions_Trophy, IPL,..]
equipmentNeeded = [Cricket_Ball, Cricket_Bat, Wicket, Helmet, ...]
requiredArea: 22 //yards

Exercise 2

Define the following concepts from OOP realm:

1. **encapsulation**
2. **inheritance**
3. **polymorphism**
4. **accessor functions**
5. **function and operator overloading**

Give references for the sources of your definitions.

Encapsulation
In programming, encapsulation is the process of enclosing or wrapping code into a single, self-contained package. For instance, modularising a procedural programme by replacing the code for printing an array with a function is one of the simplest forms of encapsulation.

In OOP, encapsulation means bundling variables and methods that operate on these variables into a class. This creates a self-contained template for objects of the class whose implementation is usually hidden from the user.

Inheritance
An OOP concept which establishes an 'is a' relationship between two classes. Class B is said to inherit from Class A if the B is a more specialised version of A. This means B not only has all the attributes and methods of class A, but also its own specialised attributes and methods. In this case, Class B is said to be a subclass of A. Equivalently, A is said to be the parent class of B.

Polymorphism
An OOP principle which is used to describe how the implementation of a function common to many subclasses is unique for each subclass - the same function takes 'many forms' depending on the class of the instance on which the function is called.

For instance, if we were to write an `Animal` class, one of the functions we would define would be '`makeSound( )`' - making a sound is a behaviour that is common to most, if not all, animals. Subclasses of the `Animal` class include a `Dog` class, a `Mouse` class, and a `Cat` class, each of which will inherit the `makeSound()` function from Animal.

However, the implementation of the `makeSound()` method will be different for each of these subclasses, which makes sense because even though dogs, cats, and mice all make sounds, *how* they make these sounds is different. As such, the `Dog` class' `makeSound( )` method may in turn call a `Dog`-specific function like '`bark()`' or '`woof()`', the `Cat` class' `makeSound()` method may implement something equivalent to '`meow()`', and the `Mouse` class may define '`makeSound()`' in terms of '`squeak()`'.

Accessor Functions
Public functions of a class that are used to 'access' private member variables. Accessor functions are also called 'getter' functions.

Function and Operator Overloading
Operator overloading is the process of extending or redefining the behaviour of an existing C++ operator such as +, -, or <<, often for use with new classes.

Similarly, function overloading is the process of defining more than one implementation for a function with the same name.