Procedural and Object Oriented programming

Procedural – what we have done up to now. Data and functions are defined separately.

Object oriented – based around creating objects that contain both data and procedures.

The data in an object is called attributes. Procedures in an object are called member functions. Sometimes these are called properties and methods in other languages.

Encapsulation – combining data and code into a single object.

Data hiding – hiding attributes that aren't exposed to outside functions. Only member functions can have access to these attributes.

Object reusability – by packaging code into objects, it is easier to reuse.

Class- code that specifies both attributes and member functions that make up an object.

Like a struct, you still need to create an instance of a class.

Classes are a data type defined by you. Classes consist of variables and functions (usually doing operations on those variables).

```
class Student{
        string name;
        int id;
        string GetName(){
                return name;
        }
};
```

By default, all members of a class are hidden (private). To make them available, you need to change the access specifier.

Access Specifiers:

- private – only member functions can access
- protected – member functions, friends, and derived classes can access
- public – if you can access the instantiated class, you can access these members.

```
class Student{
        public:
                string getName(); //This is a function prototype
        private:
                string name;
                void setName(string name);
};
```

It is not required to group your public and private sections together. It helps to do this for readability. Frequently, attributes will be defined as private with member functions to get the value and set the value. These are called getters and setters.

Adding the const keyword to the end of a function prototype tells the compiler the function shouldn't modify member variables.

To define member functions outside the class definition, do the following:

```
string Student::getName(){
        return name;
}
```

Defining a class function outside the class declaration isn't required. My rule of thumb is define it outside if it is more than 5 lines of code.
Code defined inside the class is called inline.

Note, it would be perfectly legal to define getName without the Student::. However, it wouldn't do what you wanted it to. Leaving off the Student:: would create a function that was independent of the class.

The two colons are called the scope resolution operator.

To actually use a class, you still need to create an instance of it.

```
Student Sue;
```

Accessing object members of classes is just like accessing object members of structures.

```
Sue.name = "Susan"; //This would not compile. It would tell you that name is recognized
//because name is private.
cout << Sue.getName(); //This is perfectly legal.
```

For member functions that use member data, it is best to re-fetch it every time the function has been called to avoid missing out on updates. This really is a concern if you use the static keyword.

Pointers to classes – pointers to classes work essentially the same as pointer to structures.
Use the -> operator to access member data/functions.

```
Student *currStudent = &Sue;
cout << currStudent -> getName();
```

This is equivalent to

```
cout << *(currStudent).getName();
```

The new operator works with classes the same as structures. So does the delete operator.
new – creates new memory on the heap. Then it returns a pointer to the object.
delete – marks the memory for that object as available for use.

Checkpoint
13.1 – false.
13.2 – B
13.3 – A
13.4 – C
13.5 –

```
class Date{
        public:
                string getDate();
                void setDate(int m, int d, int y){
                        month = m;
                        }
        private:
                int month, day, year;
};
```

Why private members?
It allows you to provide data validation.
It allows us to control the state of our class without worries about someone else modifying it.

Typically classes are defined in their own separate file. This is called the "class specification" file. Typically this file has a .h extension with name the same as the class name. Student.h.

It is common to put your class member function's definitions in a separate file from the class specification. This is called the "class implementation" file. This typically has a .cpp extension. This .cpp will need to include your .h file.

Any program using the class will include the .h file, not the .cpp file.

Preprocessor statements
#ifndef – used to check if a header file has already been included so it doesn't get included twice.

#ifndef – if not defined. This is an if statement executed by the preprocessor. It is checking if a variable is defined.
#define – defines a variable.
#endif – end of our if statement.