Inline member functions
We can put the code for a function of a class member inside the class instead of separately.

```
class Student{
        ….
        int getID(){
                return id;
        }
…
}
```

Inline functions replaces it with the code rather than calling that function.
Inline functions are to be used only when the definitions are really short.
When you have an inline function, you don't scope resolution operator.

Constructors – a function that a class uses to initialize class member variables and any other setup that is required.

A constructor has the same name as the class and is called automatically when the class is Created in memory.

- new operator
- declare an instance.
- It does not happen when we declare a pointer.

```
class Student{
        public: //We have to make it public.
                Student();  //no return type specified.
…
};
Student::Student(){
        …
}
```

You should consider writing a constructor for every class. If you don't, a default constructor will be created by the compiler.

Constructors can have arguments passed in as parameters. If a constructor has no arguments, It is considered the "default constructor"
We can have multiple constructors. A default constructor is not required and if there is a constructor, a default one will not be supplied for you.

```
class Student{
        public: //We have to make it public.
                Student(string name);  //no return type specified.
                Student(string name, int MNumber);
…
};
Student::Student(string name){
        …
}
```

13.6 – so they can't be changed by the user. Can make sure values assigned are appropriate and legal.
13.7 – getters and setters
13.8 – specification is the .h and the implementation is the .cpp file.
13.9 – keeps us from declaring something twice.
13.10 –
BasePay Declaration – BasePay.h
BasePay member function definitions – BasePay.cpp
Overtime Declaration – Overtime.h
Overtime member function definitions – Overtime.cpp

13.11 – write the code for a class member function in the class itself.

Passing arguments to the constructor

```
int i;
Student Sam("Samantha");
Student s = new Student("Billy");

class Student{
        public: //We have to make it public.

        Student(string name, int MNumber = 123456);
…
};

Student::Student(string name, int MNumber = 123456){
        …
}
```

We can have default arguments in constructors.
If we have a constructor that looks like:
    Student::Student(string name="No Name Given", int MNumber=12345){}
If all parameters are optional, it becomes the default constructor.
    Student::Student(){} //This would be illegal because wouldn't know which to call.

If you don't have a default constructor or a constructor with all optional parameters and you try to create an instance with no parameters, it will be a compiler error.

Destructors – Destructor is like a constructor except it is called when an object is removed from memory (destroyed).

Destructors are called automatically. They give you a chance to cleanup such as closing files or DB connections or freeing memory used inside the class.

Destructors have the same name as the class (like a constructor) except they have the ~ prefix.

```
class Student{
        public:
                ~Student();
…
}
```

Checkpoint
13.12 – allocates memory for a class object. Initializes fields for a class object.
Gets a class ready to be used.
13.13 – Removes a class object from memory. Cleans up before throwing the memory away.
13.14 – A
13.15 – B
13.16 – B
13.17 – A

Private member functions
Private member functions can only be called from inside the class. These functions that are operating on the private members can be public or private member functions. These can modify public or private member variables.

We can create arrays of objects.
When you create an array of objects that are classes, the constructor is called when the array is created.

```
Student cs2028C[70];  //The constructor is called 70 times here.
```

We can initialize the class instances when we initialize our array.

```
//1 parameter
Student cs2028C[70] = {“Mary”, “Scott”, “Patty”, …};
```

Or

```
//Multiple parameters
Student cs2028C[70] = {Student(“Bob”, 123), Student(“Jill”, 456), …};
```

You can mix which constructor you use in a single call.

To access class members in an array

```
cs2028c[3].Name = “Patricia”;
cout << cs2028c[2].mNumber << endl;
```

Checkpoint
13.21 – 10
20
50
13.22 – 4
7
2