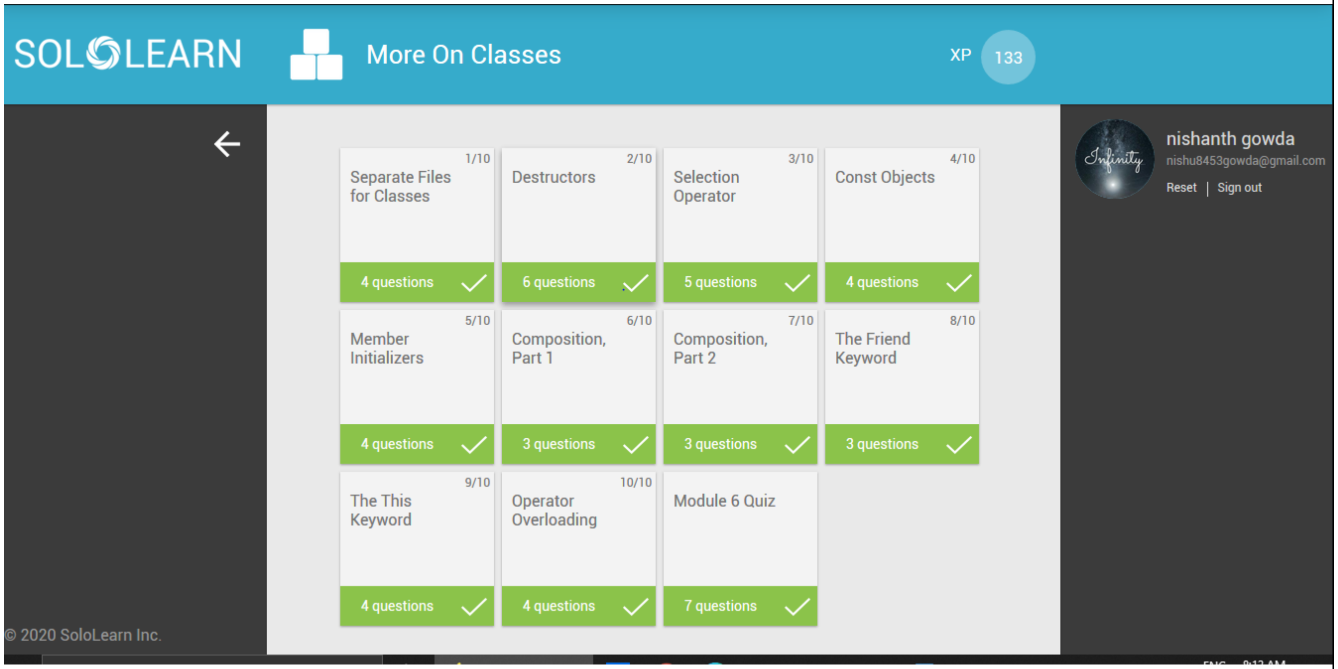
DAILY ASSESSMENT FORMAT

Date:	**26/June/2020**	**Name:**	**nishanth**
Course:	**C++ programming**	**USN:**	**4al17ec063**
Topic:	**Destructors** **Operator overloading**	**Semester & Section:**	**6th b**
GitHub Repository:	**nishanthvr**		

FORENOON SESSION DETAILS

Image of session



1/10 Separate Files for Classes — 4 questions ✓	2/10 Destructors — 6 questions ✓	3/10 Selection Operator — 5 questions ✓	4/10 Const Objects — 4 questions ✓
5/10 Member Initializers — 4 questions ✓	6/10 Composition, Part 1 — 3 questions ✓	7/10 Composition, Part 2 — 3 questions ✓	8/10 The Friend Keyword — 3 questions ✓
9/10 The This Keyword — 4 questions ✓	10/10 Operator Overloading — 4 questions ✓	Module 6 Quiz — 7 questions ✓	

Destructors

Remember constructors? They're special member functions that are automatically called when an object is created.
Destructors are special functions, as well. They're called when an object is **destroyed** or **deleted**.
Objects are destroyed when they go out of scope, or whenever the **delete** expression is applied to a pointer directed at an object of a class.

#ifndef & #define

We created separate header and source files for our class, which resulted in this header file.**#ifndef** MYCLASS_H
#define MYCLASS_H

```
class MyClass
{
public:
MyClass();
protected:
private:
};

#endif // MYCLASS_H
```

ifndef stands for "if not defined". The first pair of statements tells the program to define the **MyClass** header file if it has not been defined already.
endif ends the condition.
This prevents a header file from being included more than once within one file.

Dot Operator

Next, we'll create an object of the type **MyClass**, and call its **myPrint()** function using the dot (.) operator:

```
#include "MyClass.h"

int main() {
MyClass obj;
obj.myPrint();
}

// Outputs "Hello"
```

Pointers

We can also use a **pointer** to access the object's members.
The following pointer points to the **obj** object:MyClass obj;
MyClass *ptr = &obj;

Selection Operator

The **arrow member selection operator (->)** is used to access an object's members with a pointer.

```
MyClass obj;
MyClass *ptr = &obj;
ptr->myPrint();
```

Friend Functions

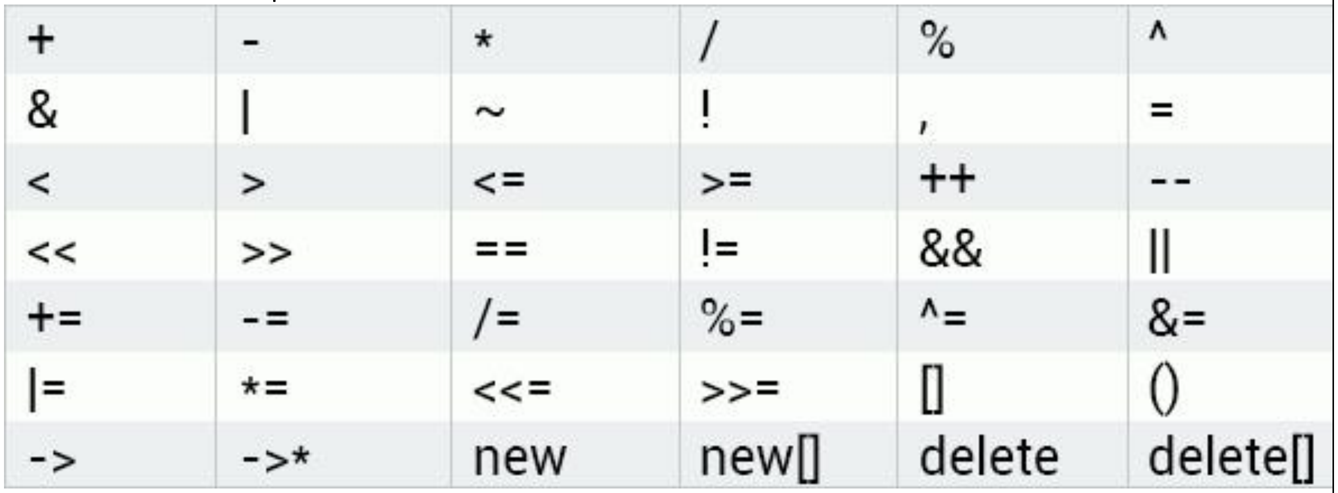
Normally, private members of a class cannot be accessed from outside of that class.
However, declaring a **non-member** function as a **friend** of a class allows it to access the class' private members. This is accomplished by including a declaration of this external function within the class, and preceding it with the keyword **friend**.
In the example below, **someFunc()**, which is not a member function of the class, is a friend of **MyClass** and

can access its private members. class MyClass {
```
public:
MyClass() {
regVar = 0;
}
private:
int regVar;

friend void someFunc(MyClass &obj);
};
```
The function **someFunc()** is defined as a regular function outside the class. It takes an object of type **MyClass** as its parameter, and is able to access the private data members of that object.class MyClass {
```
public:
MyClass() {
regVar = 0;
}
private:
int regVar;

friend void someFunc(MyClass &obj);
};

void someFunc(MyClass &obj) {
obj.regVar = 42;
cout << obj.regVar;
}
```

Operator Overloading

Most of the C++ built-in operators can be redefined or **overloaded**.
Thus, operators can be used with user-defined types as well (for example, allowing you to **add** two objects together).

This chart shows the operators that can be overloaded.



+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new[]	delete	delete[]

Operators that can't be overloaded include :: | .* | . | ?: