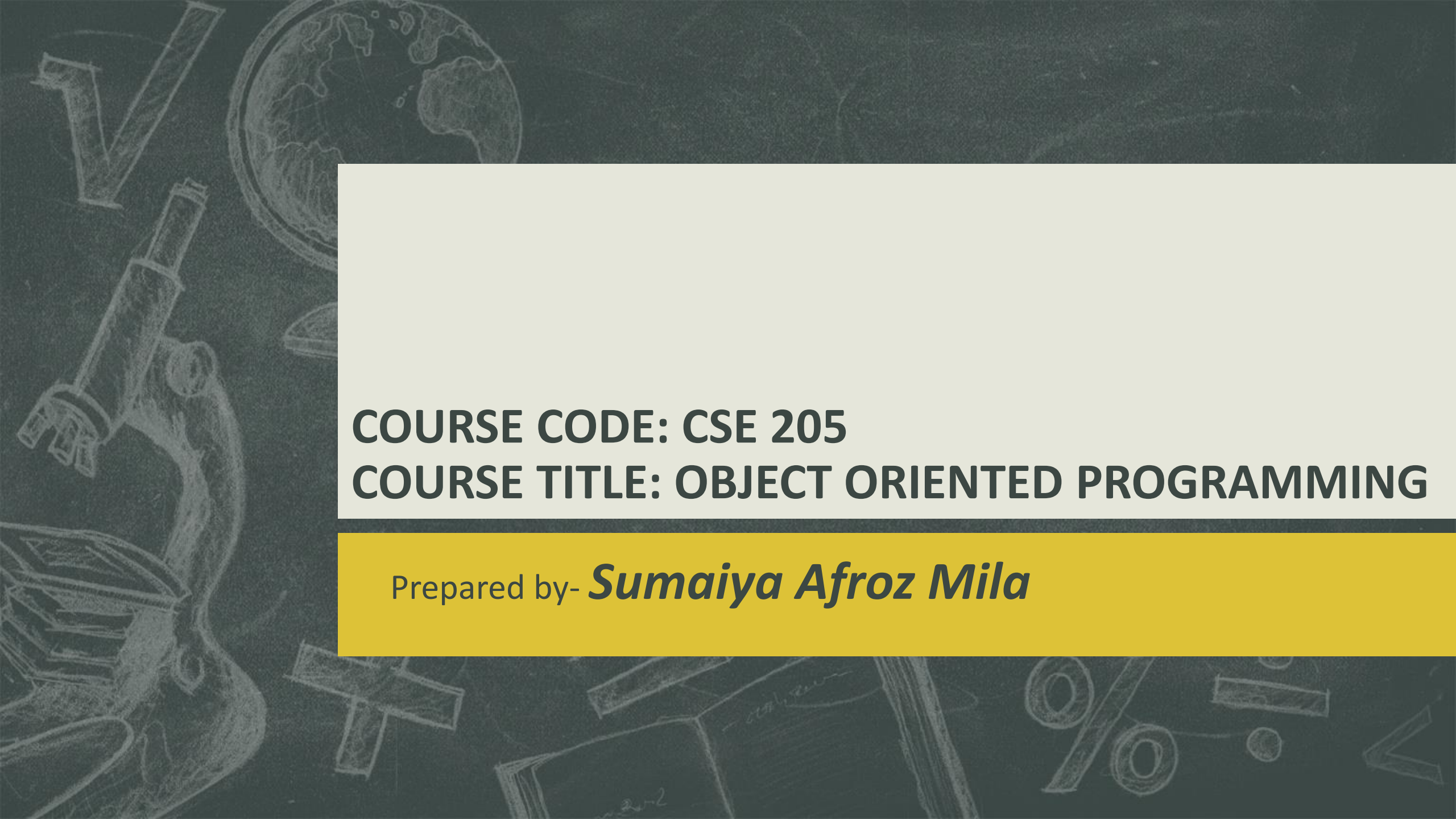
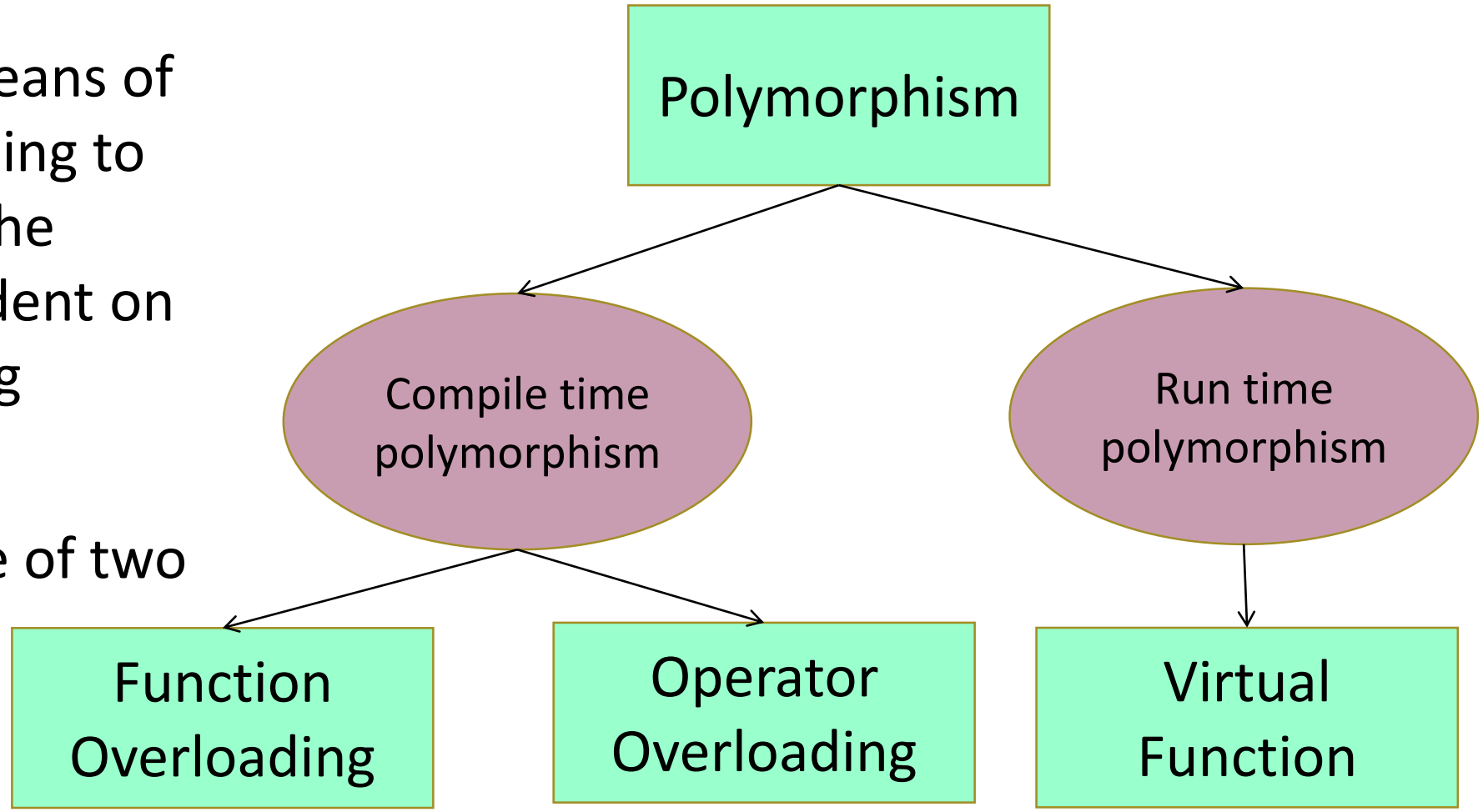


COURSE CODE: CSE 205
COURSE TITLE: OBJECT ORIENTED PROGRAMMING

Prepared by- ***Sumaiya Afroz Mila***

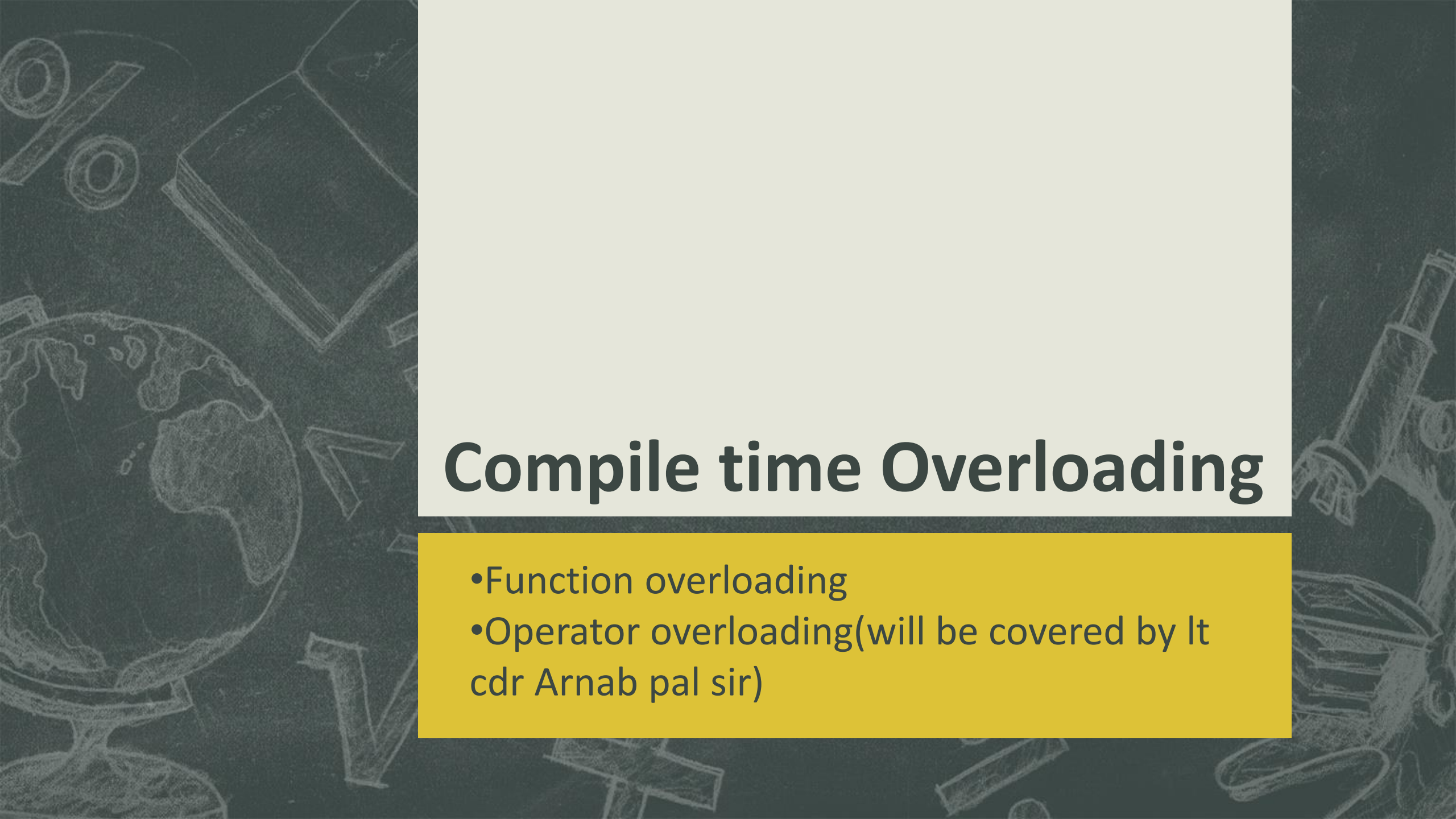
Polymorphism

- Polymorphism is a means of giving different meaning to the same message. The meanings are dependent on the type of data being processed.
- Polymorphism can be of two types-



Polymorphism

- Polymorphism is a means of giving different meaning to the same message. The meanings are dependent on the type of data being processed.
- Polymorphism can be of two types-
- Compile time polymorphism:
  - **Function Overloading**: gives the same name different meaning. The name has several interpretations that depend on function selection.
  - **Operator Overloading**: provides different interpretation of C++ operator depending on type of its argument
- Pure/Run time polymorphism: runtime function selection. Related to inheritance & will be covered in later sessions. Function overriding.



Compile time Overloading

- Function overloading
- Operator overloading(will be covered by lt cdr Arnab pal sir)

Function Overloading

- Functions are either -
  - *member functions—defined inside a class*
  - *free functions—defined in global scope*
- In C, every function has a unique name
- C++ allows several functions with the same name

Early Binding vs Late Binding

- "Early binding refers to events that occur at **compile time**."
- Early binding occurs when all information needed to call a function is known at compile time.
- **Examples :**

  function calls ,overloaded function calls, and overloaded operators.

Early Binding vs Late Binding

- "Dynamic binding occurs at run-time and is also called late-binding."
- "Late binding refers to function calls that are **not resolved until run time**."
- Late binding can make for somewhat **slower** execution times.
- **Example:**

  virtual functions

Function Overloading

C

```
int abs(int i);
float fabs(float f);
long labs(long l);

void main(){
int i=-2;
float f = - 3.4;
long l = -3400l;
Printf("%d",abs(i));
Printf("%f",fabs(f));
Printf("%ld",labs(l));
}
```

C++

```
int abs(int i);
float abs(float f);
long abs(long l);

void main(){
Int i=-2;
float f = - 3.4;
long l = -3400l;
cout<<abs(i);
cout<<abs(f);
cout<<abs(l)
}
```

Overloading mechanism

- Function are differentiated by their names and argument list
- For overloading, Argument list must vary in –
  - Number of arguments
  - Types of arguments
- Only return types can't be used to differentiate functions
  - int func(int i, float f);
  - float func(int i, float f);

Cannot be overloaded

Function Overloading

```
int sum(int a, int b){
        int result = a+b
        return result;
}
float sum(float a, float b){
        float result = a+b
        return result;
}
```

```
void main(){
        int  sumInt = sum(5,6);
        float  sumFloat = sum(4.5,6.0);

        cout<<“Summation of the two
integer numbers : ”<<sumInt<<endl;
        cout<<“Summation of the two
float numbers : ”<<sumFloat<<endl;
}
```

Function Overloading Example

```
int fun(int a, int b){
        return a + b;
}

float fun(int x, int y) {
        return x – y;
}
```

```
int fun(int a, int b){
        return a + b;
}

float fun(float x, float y) {
        return x – y;
}
```

Overloading Example

```
int totalMark (int phy, int chem){
        int total = phy+chem;
        return total;
}
int totalMark(int phy, int chem, int opt){
        int total = phy+chem+opt;
        return total;
}
void main(){
cout << totalMark (80,90);
cout << totalMark (80,90,80);
}
```

Default Argument

- Must be to the right of any parameter that don’t have default value
- Default can not be specified both in prototype and definition
- Default arguments must be constant or global variables

Default Argument Example

```
int totalMark(int phy, int chem, int opt = 0){
        int total = phy+chem+opt;
        return total;
}

void main(){
 cout << totalMark (80,90);
cout << totalMark (80,90,80);
}
```

Default Argument Restriction

```
int totalMark(int opt = 0, int phy, int chem){
        int total = phy+chem+opt;
        return total;
}
```

ERROR
Must be to the right of any parameter that don't have defaults

---

```
int totalMark( int phy, int chem, int opt = 0);

int totalMark( int phy, int chem, int opt = 0){
        int total = phy+chem+opt;
        return total;
}
```

ERROR
Default can't be specified both in prototype and definition

```
int totalMark( int phy, int chem, int opt = phy){
        int total = phy+chem+opt;
        return total;
}
```

ERROR
Default arguments must be constant or global variables


Thank
you!