COM211

Advanced Computer Programming

Classes

- Object-oriented programming (OOP)
  - Involves programming using objects
  - Three fundamental concepts
    - Encapsulation
    - Inheritance
    - Polymorphism

Classes

- Encapsulation
  - The property of being a self-contained unit.
  - The process of separating the user from inner workings of an application.
  - Offers the developer with the ability to change the inner workings of an object without affecting how it's used

Classes

- Inheritance
    - This is the process of modelling objects with a well-defined set of rules.
        - New object types can be declared which extend from existing object types.
        - Promotes code reusability.

Classes

- Polymorphism
  - **“Poly”** means many and **“Morph”** means form.
    - Thus, refers to the idea of an object having multiple forms.
  - This defines OOP’s ability to redefine a class’ characteristics or behaviour depended on usage context

Classes

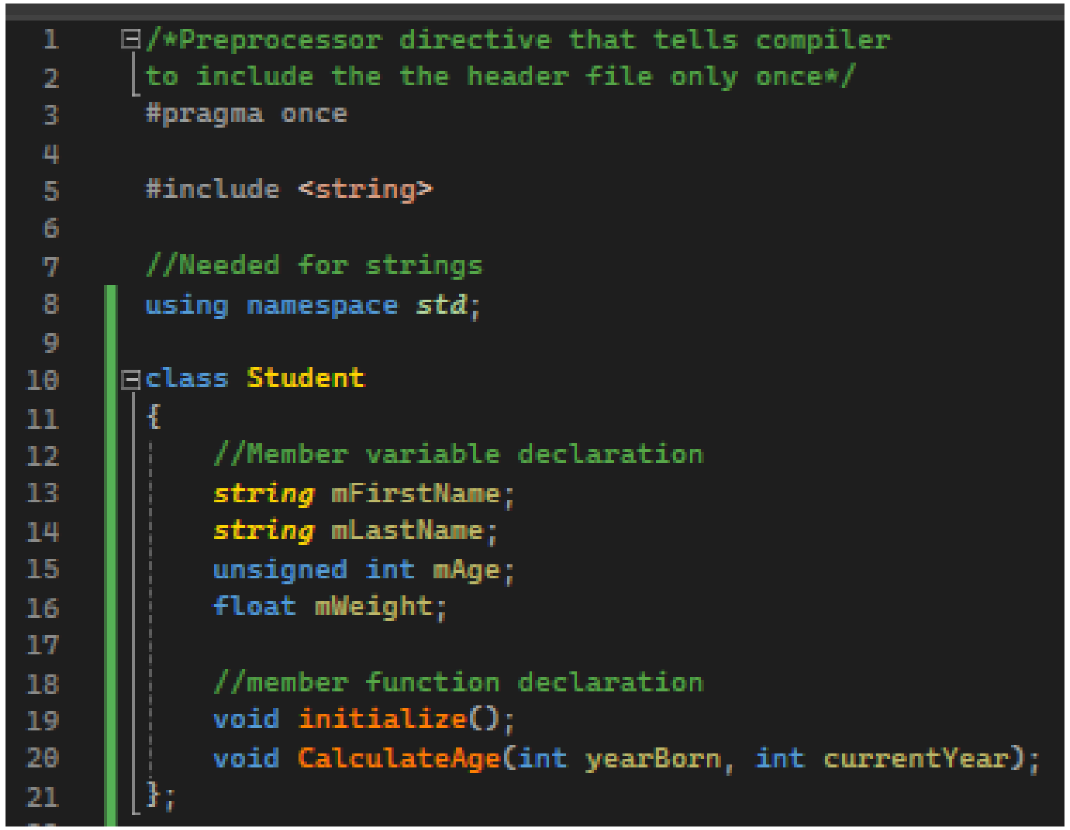
- Class
  - Classes define OOP components (objects).
  - Classes are intended to represent real-life entities
    - A collection of variables (characteristics) and functions (behaviour)
    - Class variables are known as member variables or data members
      - Can consist of any combination of variable types
        - Including other class types
    - Class functions are known as member functions/methods
      - Determine what the class can do
      - Typically manipulate member variables
  - Allow for the creation of new types
    - New types can have the functionality of built-in types

Classes

- In C++ a class is typically declared in a file known as a **header file.**
- A class is typically implemented in a separate file known as a **cpp** file.
- To declare:
  - Use the ***class*** keyword followed by the class name, and opening and closing braces
  - Within open and close braces list data members and member functions
  - Semi colon after the closing brace
- Class declaration **DOES NOT** allocate memory

Classes

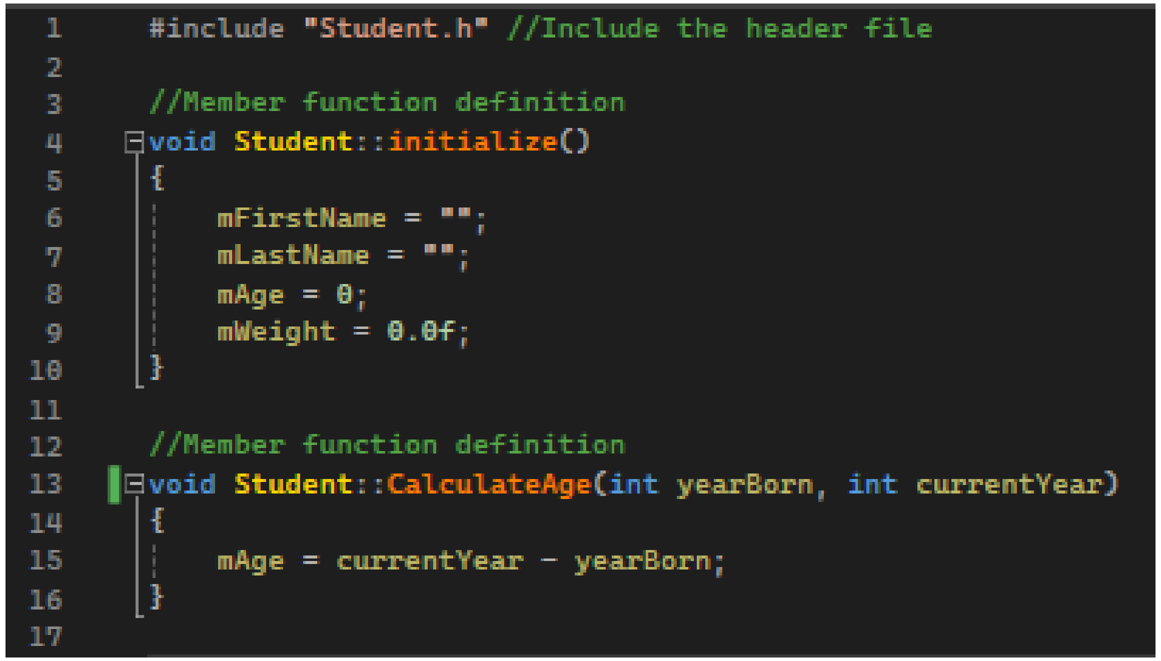
- Class declaration:



```
/*Preprocessor directive that tells compiler
to include the the header file only once*/
#pragma once

#include <string>

//Needed for strings
using namespace std;

class Student
{
    //Member variable declaration
    string mFirstName;
    string mLastName;
    unsigned int mAge;
    float mWeight;

    //member function declaration
    void initialize();
    void CalculateAge(int yearBorn, int currentYear);
};
```

Classes

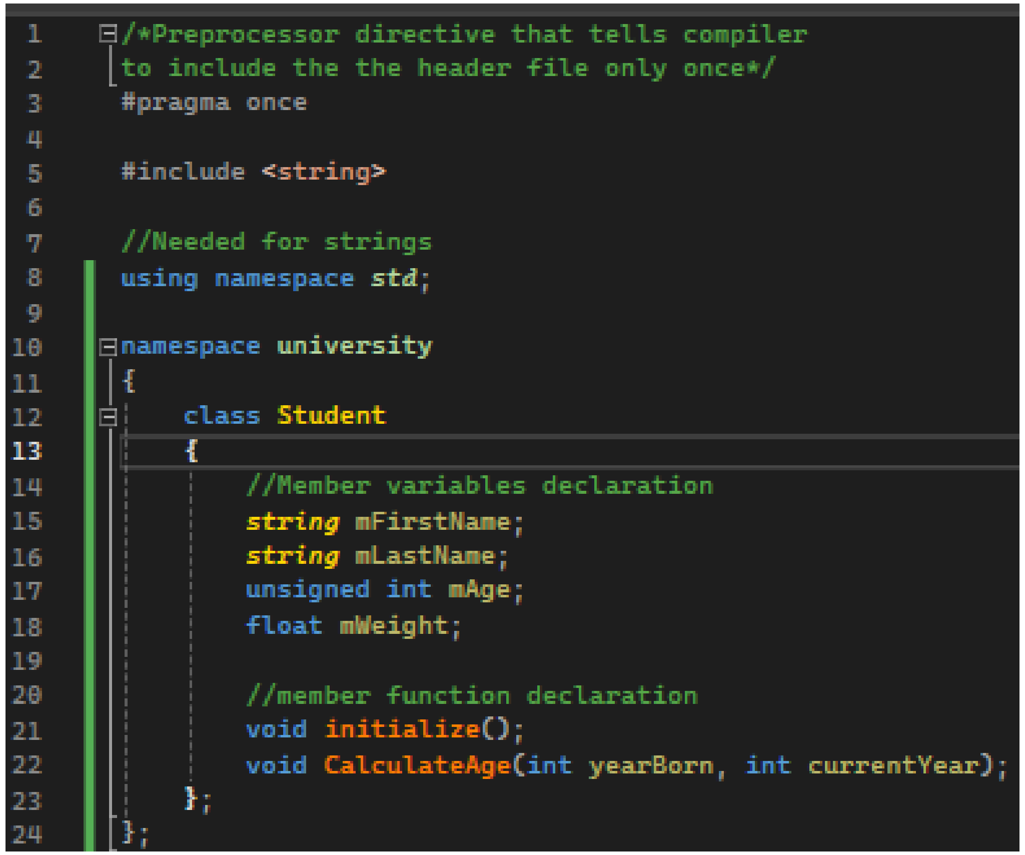
- Implementation of class methods



```cpp
#include "Student.h" //Include the header file

//Member function definition
void Student::initialize()
{
    mFirstName = "";
    mLastName = "";
    mAge = 0;
    mWeight = 0.0f;
}

//Member function definition
void Student::CalculateAge(int yearBorn, int currentYear)
{
    mAge = currentYear - yearBorn;
}

```

Classes

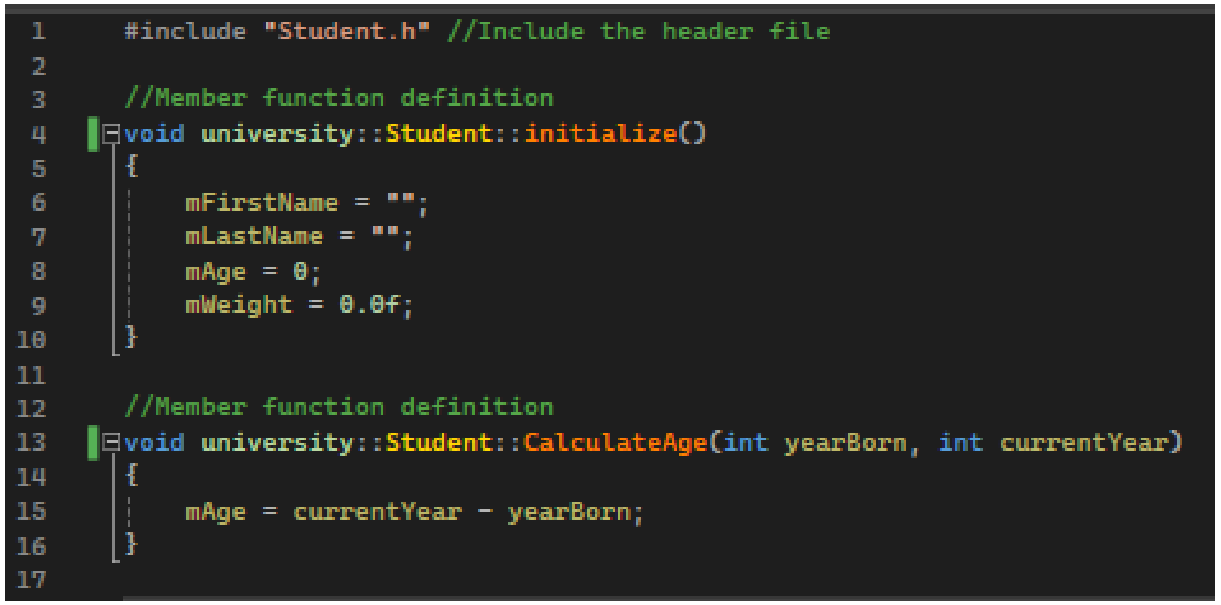
- Class declaration in a namespace:



```cpp
/*Preprocessor directive that tells compiler
to include the the header file only once*/
#pragma once

#include <string>

//Needed for strings
using namespace std;

namespace university
{
    class Student
    {
        //Member variables declaration
        string mFirstName;
        string mLastName;
        unsigned int mAge;
        float mWeight;

        //member function declaration
        void initialize();
        void CalculateAge(int yearBorn, int currentYear);
    };
};
```

Classes

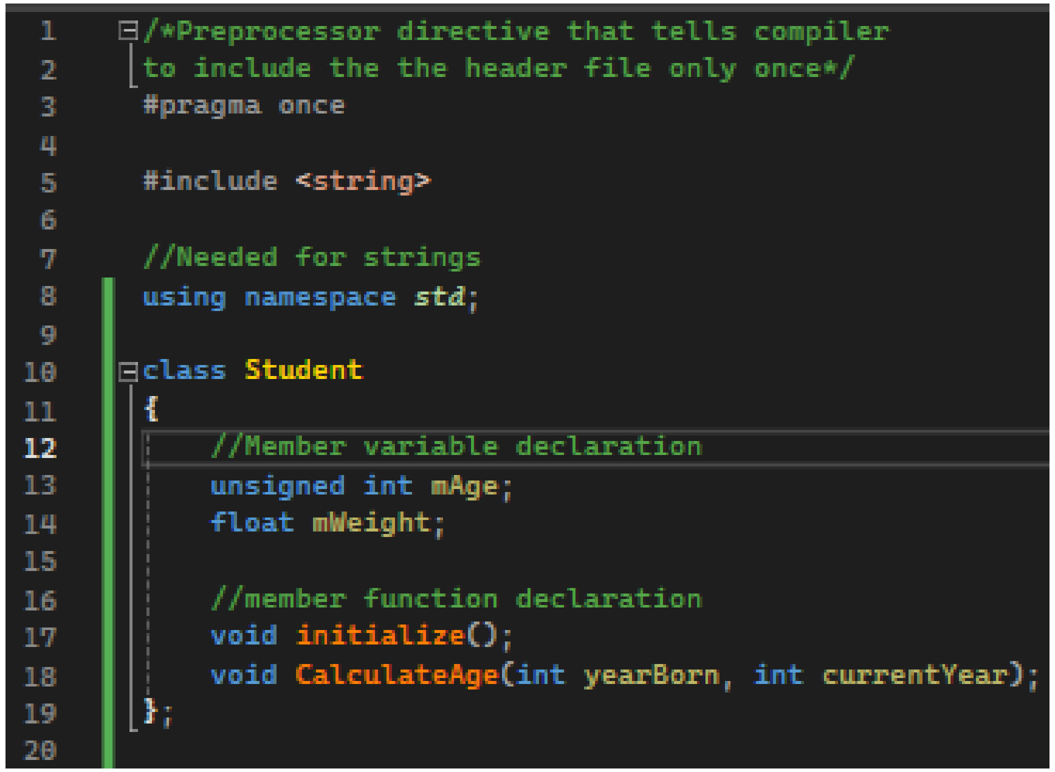
- Implementation of methods of a namespace class:



```cpp
#include "Student.h" //Include the header file

//Member function definition
void university::Student::initialize()
{
    mFirstName = "";
    mLastName = "";
    mAge = 0;
    mWeight = 0.0f;
}

//Member function definition
void university::Student::CalculateAge(int yearBorn, int currentYear)
{
    mAge = currentYear - yearBorn;
}
```

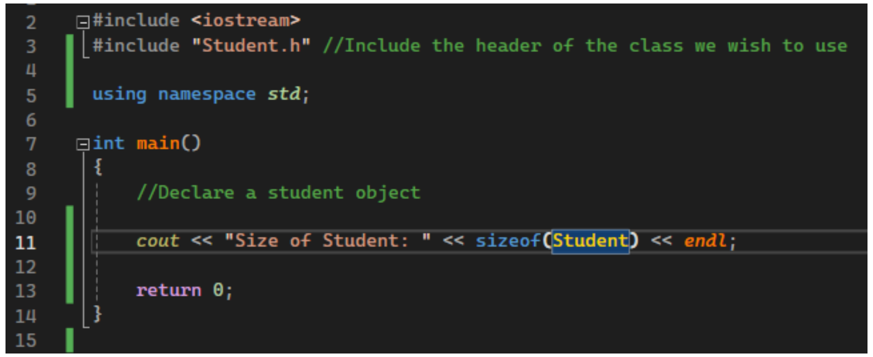
Classes

- Classes
  - Class declaration does not allocate memory
    - It does tell the compiler about how memory the class will require

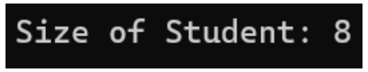


```cpp
/*Preprocessor directive that tells compiler
to include the the header file only once*/
#pragma once

#include <string>

//Needed for strings
using namespace std;

class Student
{
    //Member variable declaration
    unsigned int mAge;
    float mWeight;

    //member function declaration
    void initialize();
    void CalculateAge(int yearBorn, int currentYear);
};
```

- Student class is technically 8 bytes
  - mAge is 4 bytes (unsigned **int**)
  - mWeight is 4 bytes (**float**)
- sizeof(**Student**) is 8 bytes



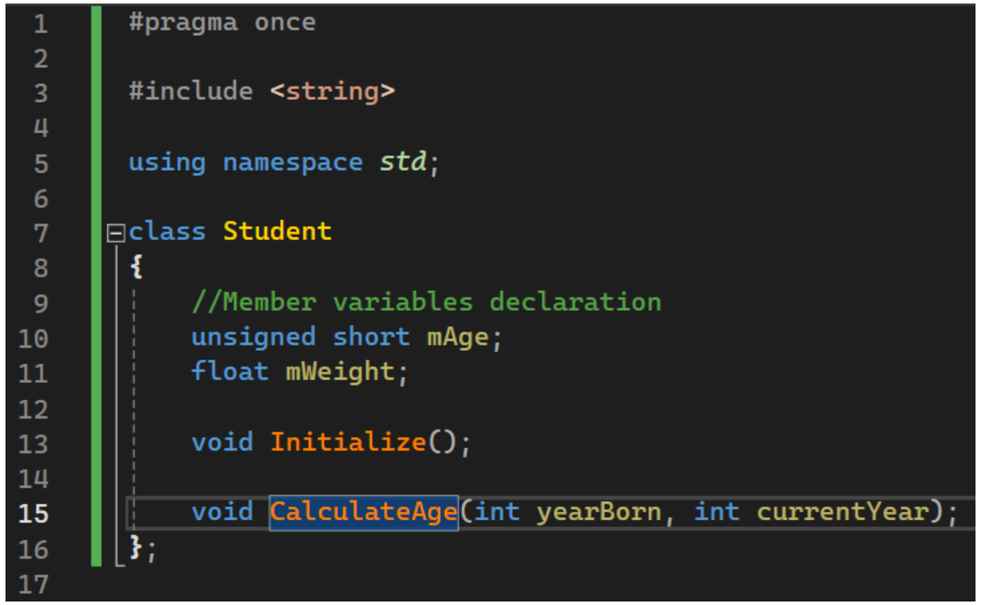
```cpp
#include <iostream>
#include "Student.h" //Include the header of the class we wish to use

using namespace std;

int main()
{
    //Declare a student object

    cout << "Size of Student: " << sizeof(Student) << endl;

    return 0;
}
```



```
Size of Student: 8
```

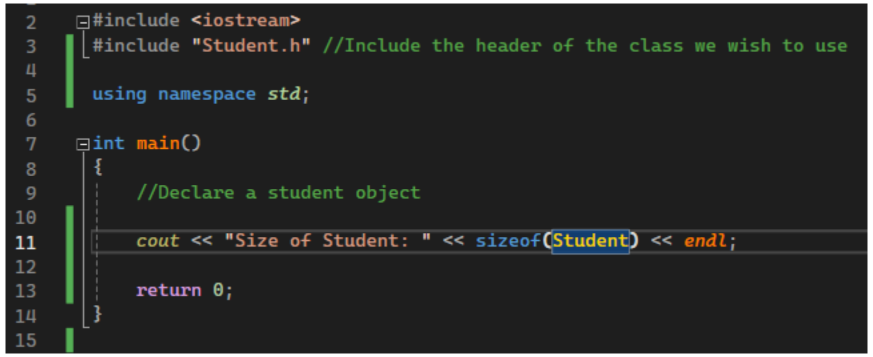
Classes

- Classes

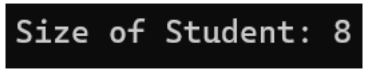


```cpp
#pragma once

#include <string>

using namespace std;

class Student
{
    //Member variables declaration
    unsigned short mAge;
    float mWeight;

    void Initialize();

    void CalculateAge(int yearBorn, int currentYear);
};
```

- Student class is technically 6 bytes
  - mAge is 2 bytes (unsigned **short**)
  - mWeight is 4 bytes (**float**)
- sizeof(**Student**) is 8 bytes
  - Not 6 bytes
  - Due to memory padding



```cpp
#include <iostream>
#include "Student.h" //Include the header of the class we wish to use

using namespace std;

int main()
{
    //Declare a student object

    cout << "Size of Student: " << sizeof(Student) << endl;

    return 0;
}
```



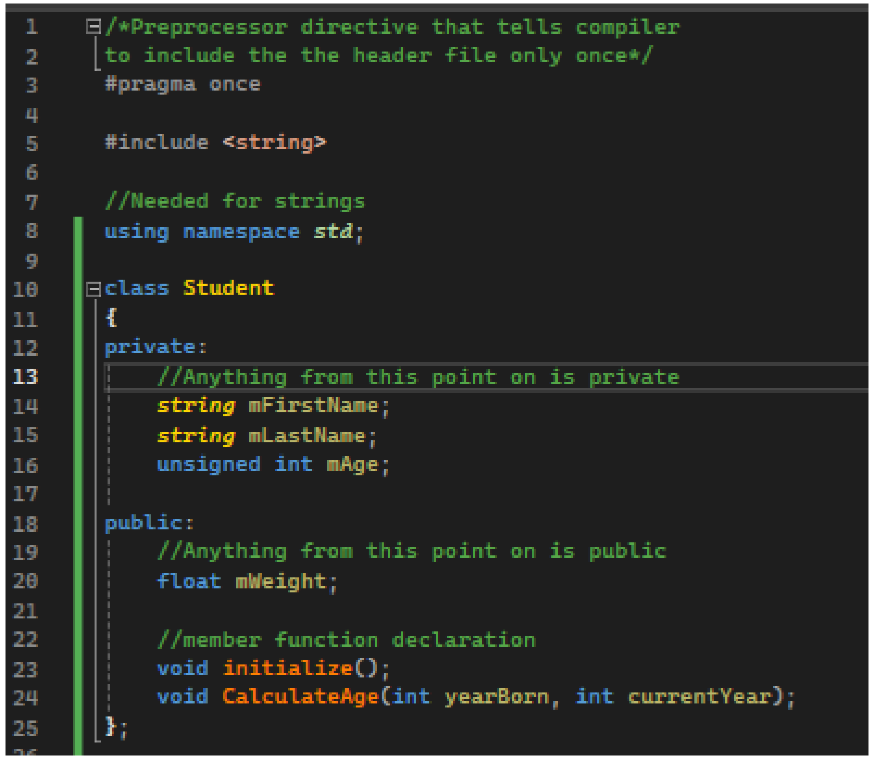
```
Size of Student: 8
```

Classes

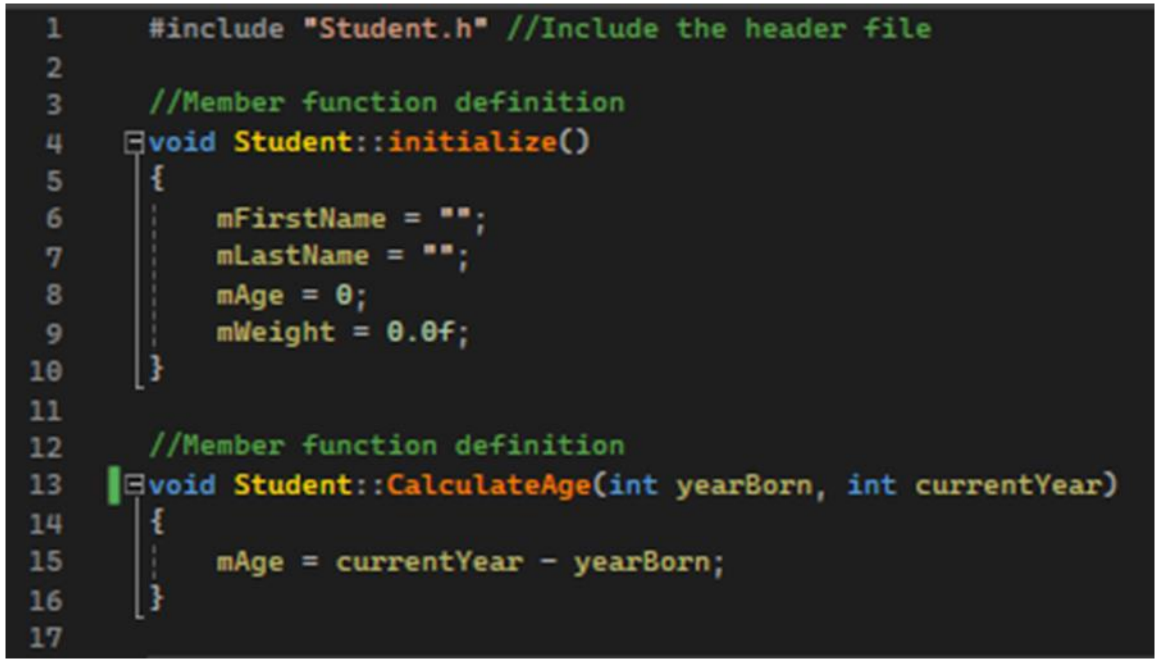
- Member scope
  - Scope determines accessibility of member variables and functions.
  - Class members are usually either private or public.
    - Applies to both methods/functions and variables.
    - The *private* and *public* keywords are applied to members.
    - Public members can be accessed through any instance of the class (using the dot operator).
    - Private members can only be accessed within functions/methods of that class itself.
    - By default, all class members in C++ are private.

Classes

- Member scope



```cpp
/*Preprocessor directive that tells compiler
to include the the header file only once*/
#pragma once

#include <string>

//Needed for strings
using namespace std;

class Student
{
private:
    //Anything from this point on is private
    string mFirstName;
    string mLastName;
    unsigned int mAge;

public:
    //Anything from this point on is public
    float mWeight;

    //member function declaration
    void initialize();
    void CalculateAge(int yearBorn, int currentYear);
};
```



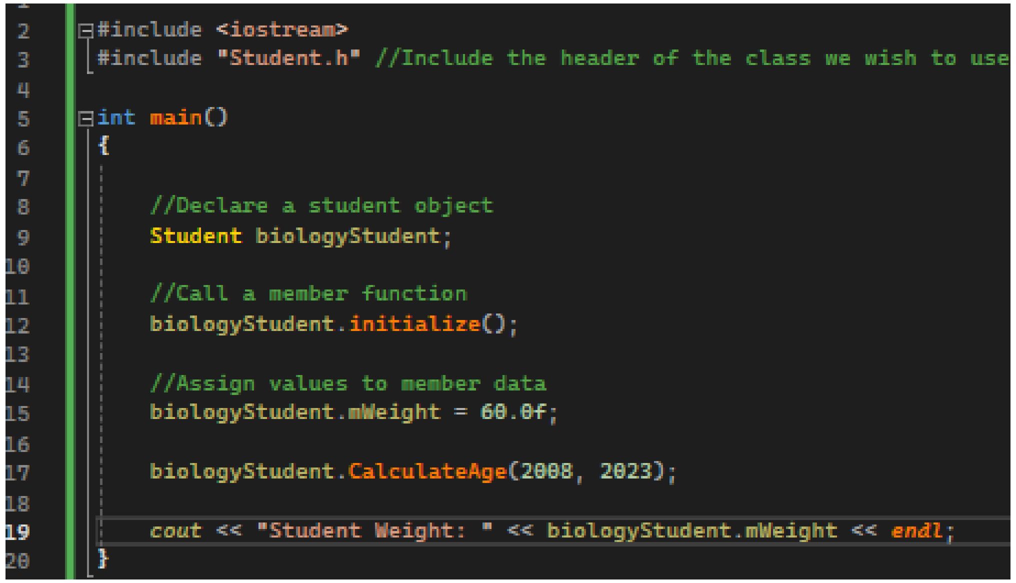
```cpp
#include "Student.h" //Include the header file

//Member function definition
void Student::initialize()
{
    mFirstName = "";
    mLastName = "";
    mAge = 0;
    mWeight = 0.0f;
}

//Member function definition
void Student::CalculateAge(int yearBorn, int currentYear)
{
    mAge = currentYear - yearBorn;
}
```

Classes

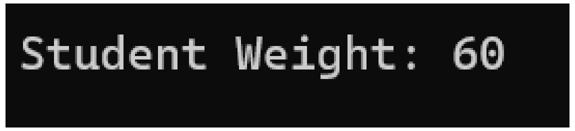
- Classes and Objects
  - After declaration a class can, be used as new type.
    - Variables of that class type can be declared.
      - An object
  - A class is **NOT** an object.
    - A variable of an that class type is an object.
      - Known as an (individual) instance of a class.
      - After instantiation, member variables and functions can be accessed by using the dot (.) operator.
        - Direct member selection operator
      - You **CANNOT** assign values to classes but only to specific objects of the class type.

Classes

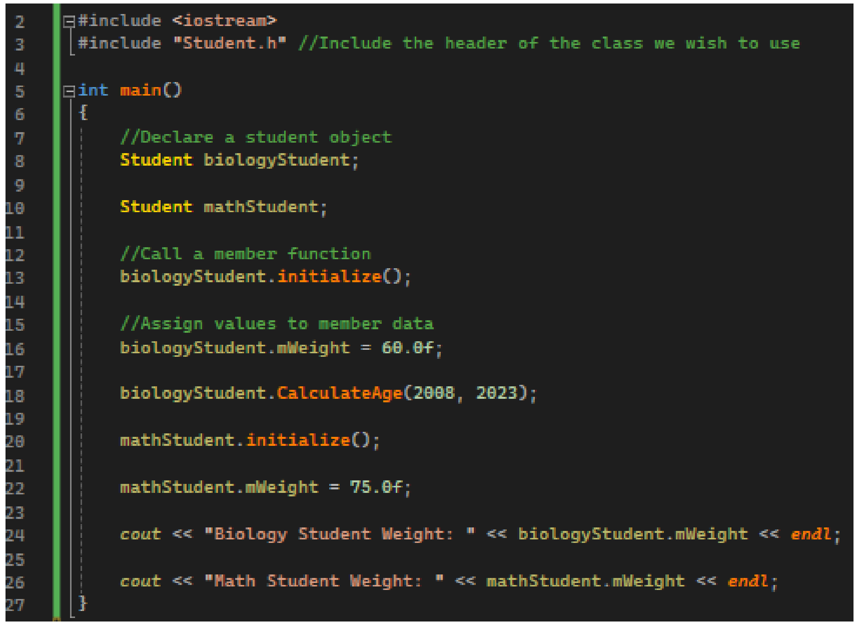
- Object Instantiation



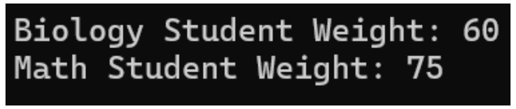
```cpp
#include <iostream>
#include "Student.h" //Include the header of the class we wish to use

int main()
{

    //Declare a student object
    Student biologyStudent;

    //Call a member function
    biologyStudent.initialize();

    //Assign values to member data
    biologyStudent.mWeight = 60.0f;

    biologyStudent.CalculateAge(2008, 2023);

    cout << "Student Weight: " << biologyStudent.mWeight << endl;
}
```



```
Student Weight: 60
```



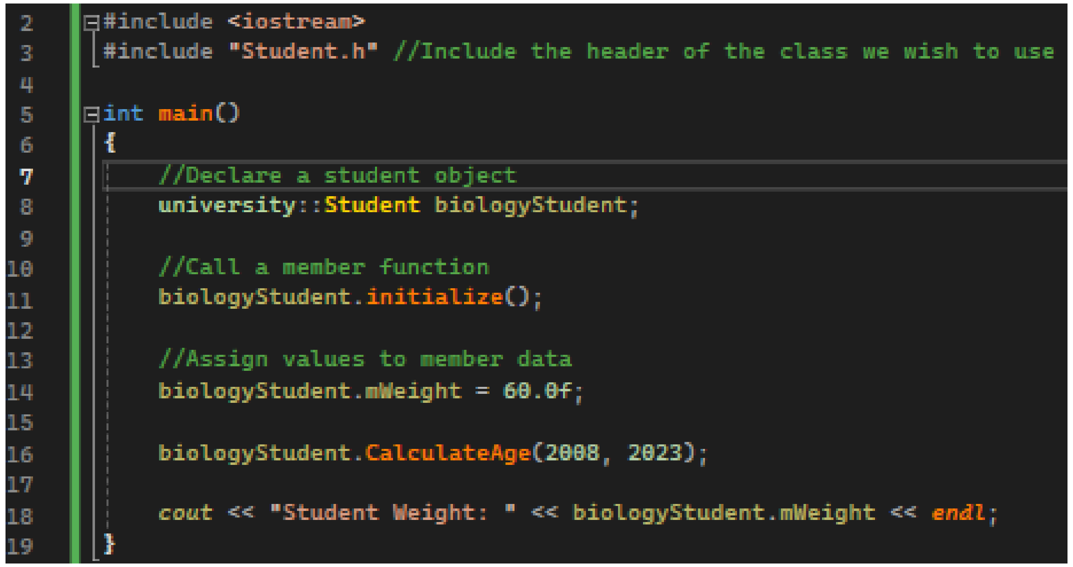
```cpp
#include <iostream>
#include "Student.h" //Include the header of the class we wish to use

int main()
{
    //Declare a student object
    Student biologyStudent;

    Student mathStudent;

    //Call a member function
    biologyStudent.initialize();

    //Assign values to member data
    biologyStudent.mWeight = 60.0f;

    biologyStudent.CalculateAge(2008, 2023);

    mathStudent.initialize();

    mathStudent.mWeight = 75.0f;

    cout << "Biology Student Weight: " << biologyStudent.mWeight << endl;

    cout << "Math Student Weight: " << mathStudent.mWeight << endl;
}
```



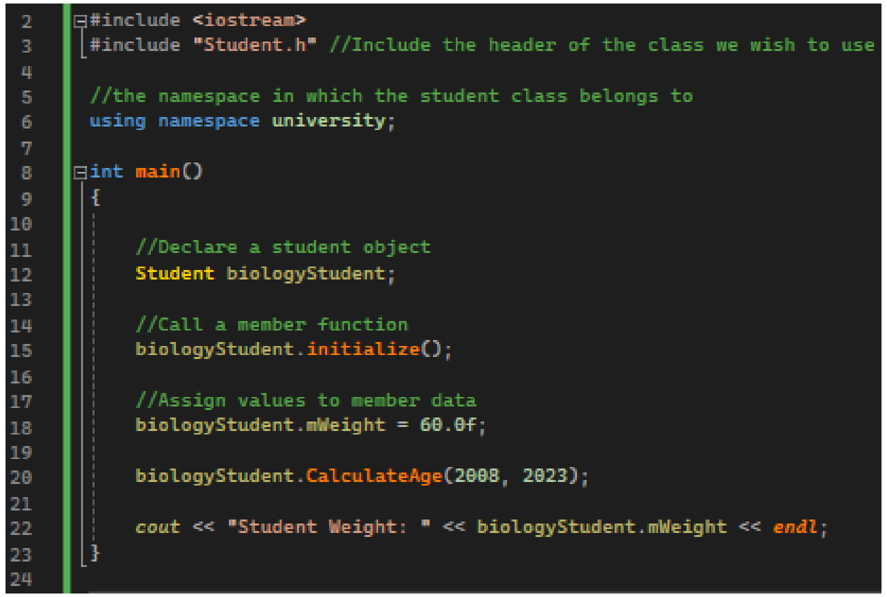
```
Biology Student Weight: 60
Math Student Weight: 75
```

Classes

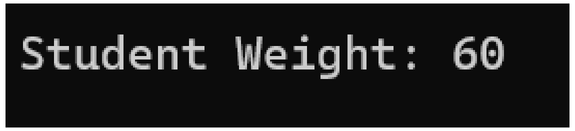
- Object Instantiation of a class declared within a namespace



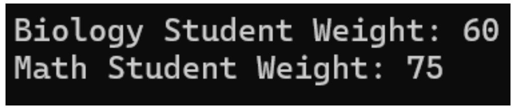
```cpp
#include <iostream>
#include "Student.h" //Include the header of the class we wish to use

int main()
{
    //Declare a student object
    university::Student biologyStudent;

    //Call a member function
    biologyStudent.initialize();

    //Assign values to member data
    biologyStudent.mWeight = 60.0f;

    biologyStudent.CalculateAge(2008, 2023);

    cout << "Student Weight: " << biologyStudent.mWeight << endl;
}
```



```cpp
#include <iostream>
#include "Student.h" //Include the header of the class we wish to use

//the namespace in which the student class belongs to
using namespace university;

int main()
{

    //Declare a student object
    Student biologyStudent;

    //Call a member function
    biologyStudent.initialize();

    //Assign values to member data
    biologyStudent.mWeight = 60.0f;

    biologyStudent.CalculateAge(2008, 2023);

    cout << "Student Weight: " << biologyStudent.mWeight << endl;
}

```



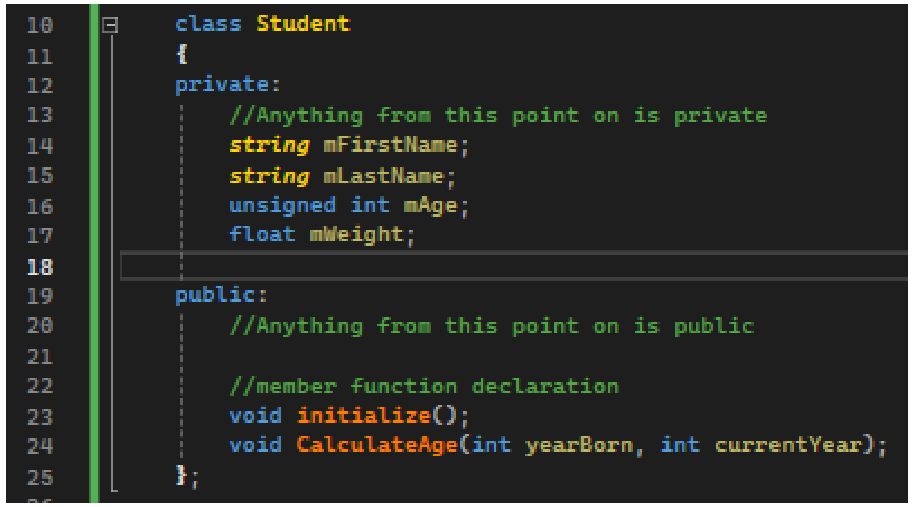
```
Student Weight: 60
```



```
Biology Student Weight: 60
Math Student Weight: 75
```

Classes

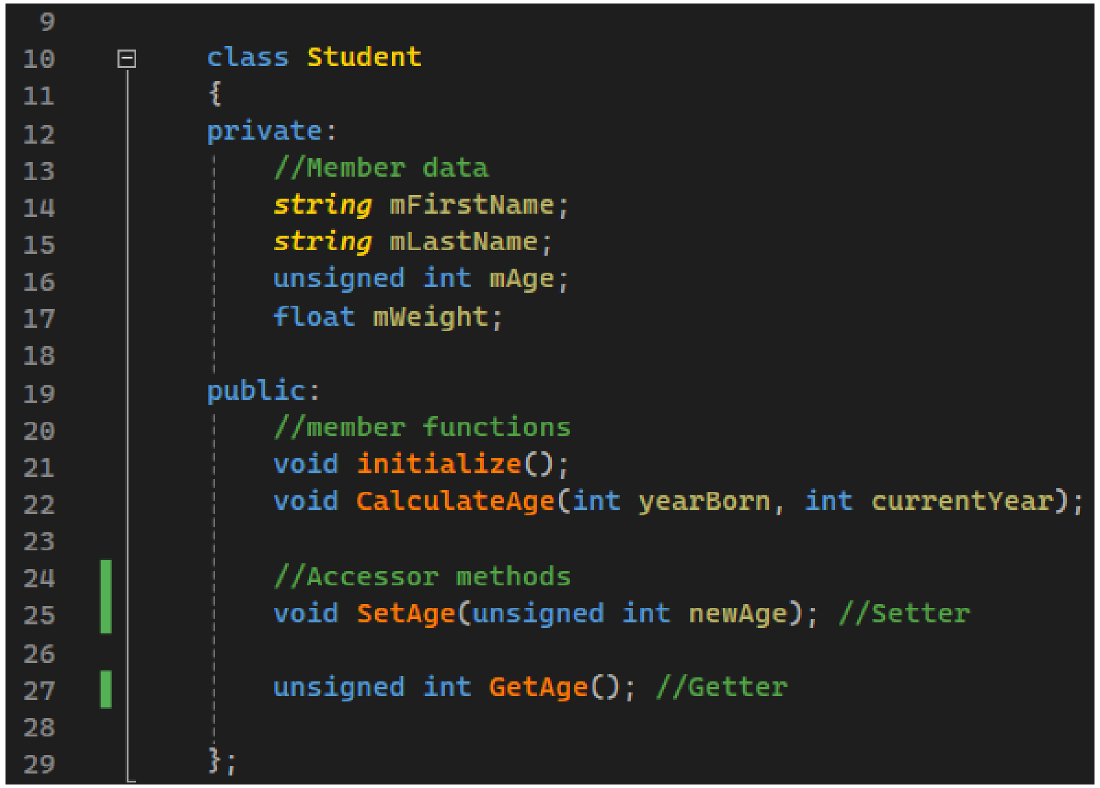
- Accessor Methods
  - A general rule of design all **member variables** should be **private**
    - To access private member variables public functions must be created
      - Known as **accessor methods**
      - Accessor methods are used to either read (get) or set the value of a private member variable



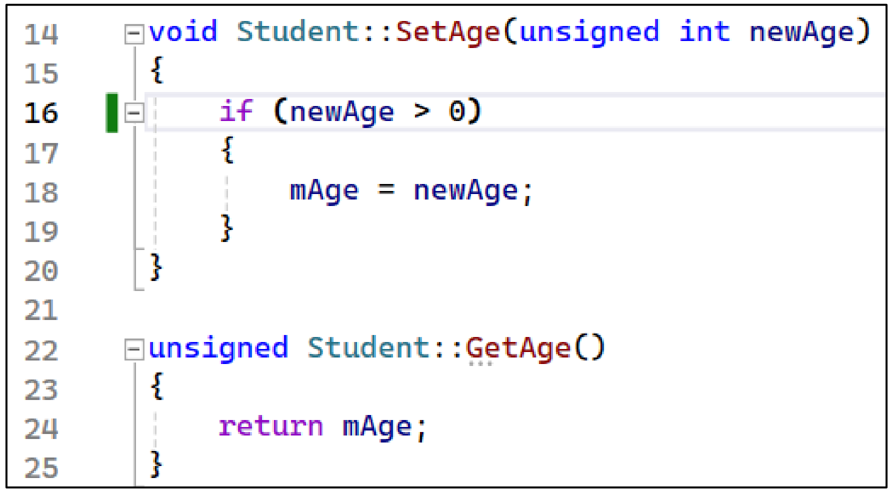
```
class Student
{
private:
    //Anything from this point on is private
    string mFirstName;
    string mLastName;
    unsigned int mAge;
    float mWeight;

public:
    //Anything from this point on is public

    //member function declaration
    void initialize();
    void CalculateAge(int yearBorn, int currentYear);
};
```

Classes

- Accessor Methods



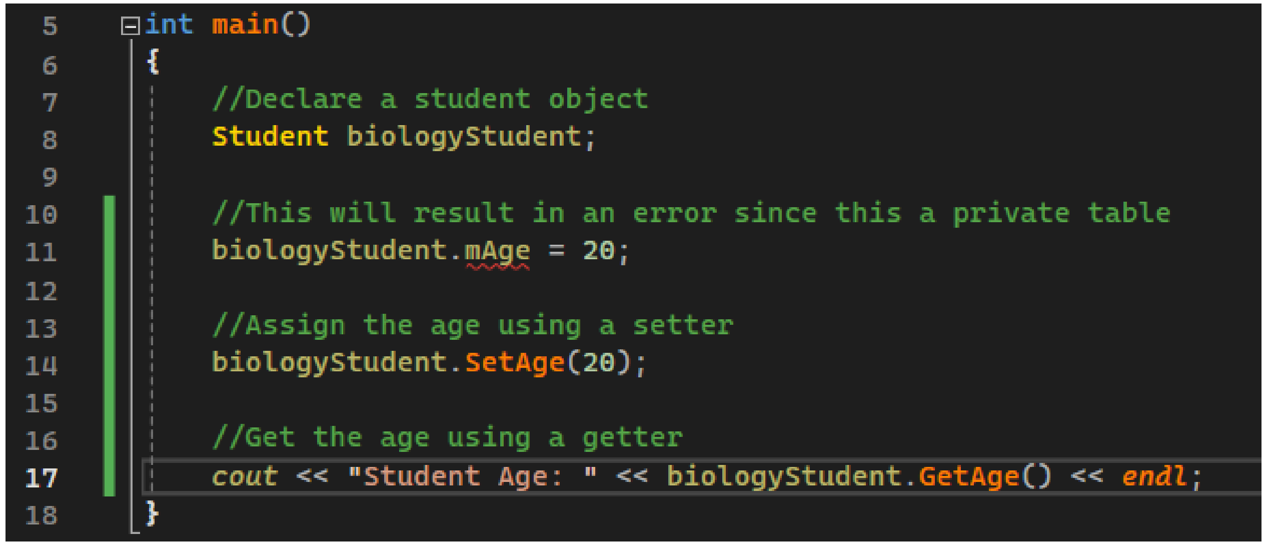
```cpp

class Student
{
private:
    //Member data
    string mFirstName;
    string mLastName;
    unsigned int mAge;
    float mWeight;

public:
    //member functions
    void initialize();
    void CalculateAge(int yearBorn, int currentYear);

    //Accessor methods
    void SetAge(unsigned int newAge); //Setter

    unsigned int GetAge(); //Getter

};
```



```cpp
void Student::SetAge(unsigned int newAge)
{
    if (newAge > 0)
    {
        mAge = newAge;
    }
}

unsigned Student::GetAge()
{
    return mAge;
}
```

Classes

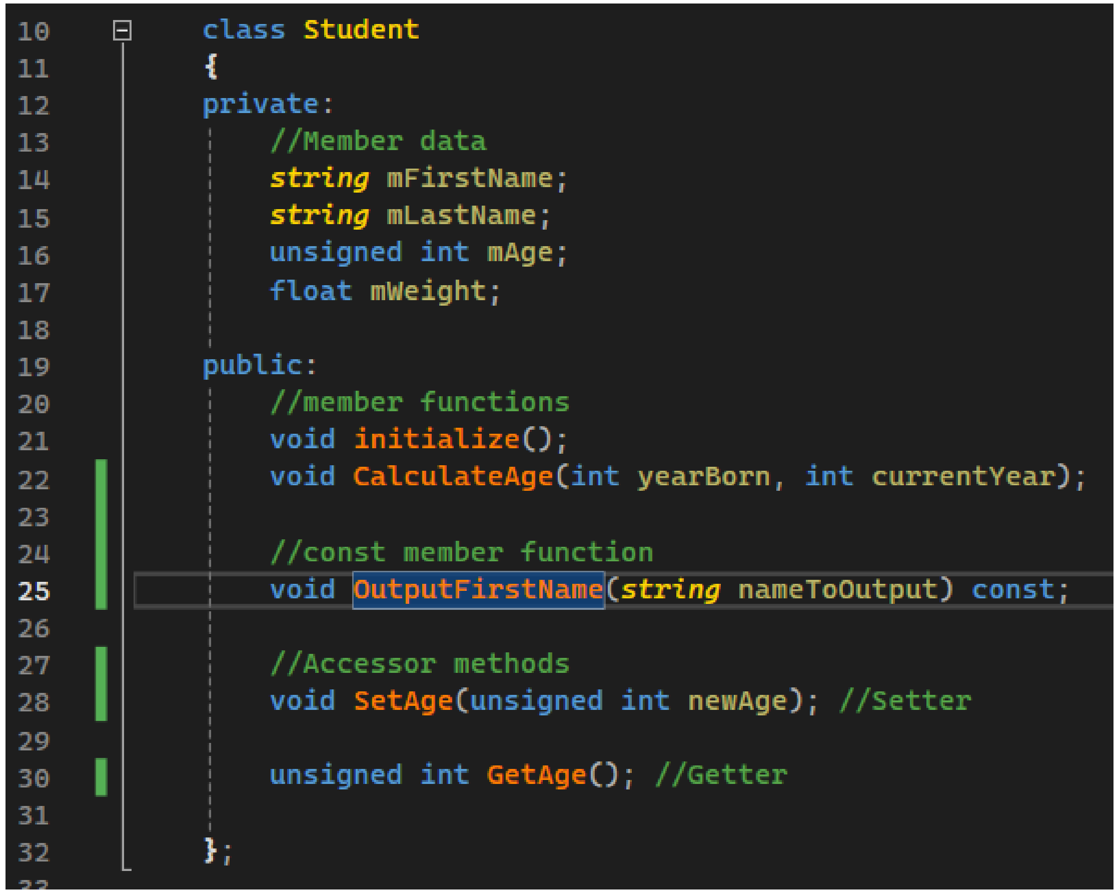
- Accessor Methods
  - Accessor functions enable the separation of details of how the data is used and stored
    - Make program/code easier to maintain



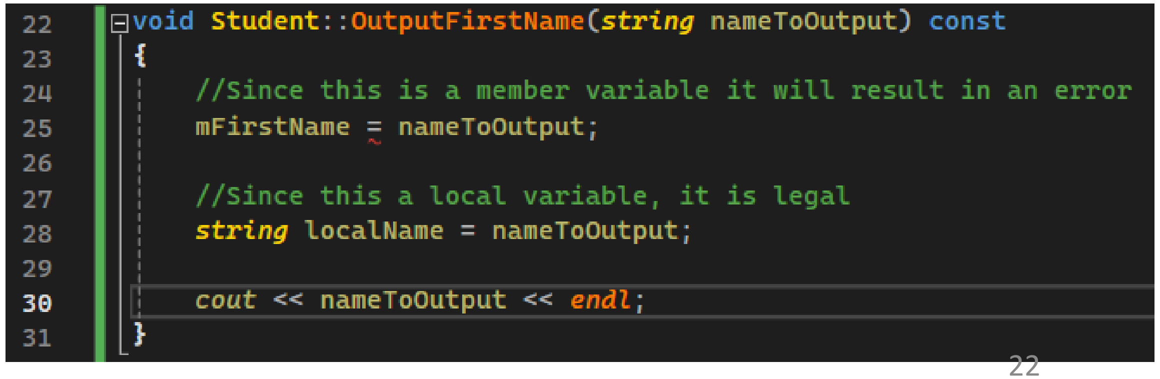
```
int main()
{
    //Declare a student object
    Student biologyStudent;

    //This will result in an error since this a private table
    biologyStudent.mAge = 20;

    //Assign the age using a setter
    biologyStudent.SetAge(20);

    //Get the age using a getter
    cout << "Student Age: " << biologyStudent.GetAge() << endl;
}
```

Classes

- Const Member Functions
  - Ensures that the member method will not make changes any member data



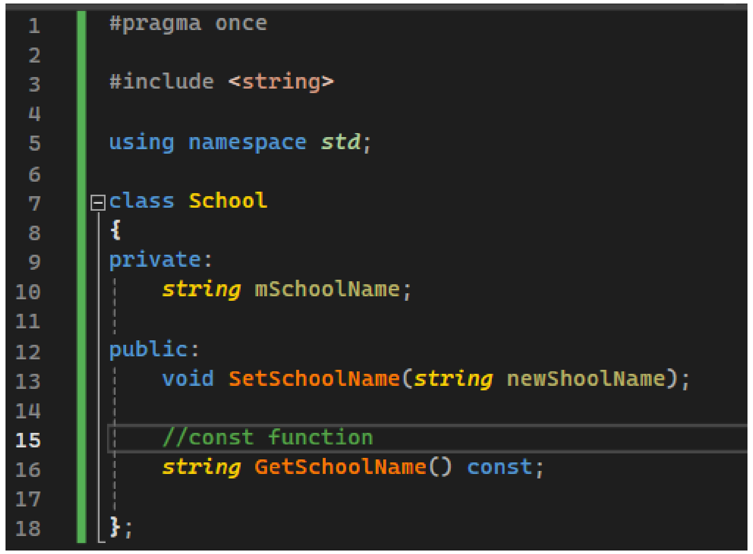
```
class Student
{
private:
    //Member data
    string mFirstName;
    string mLastName;
    unsigned int mAge;
    float mWeight;

public:
    //member functions
    void initialize();
    void CalculateAge(int yearBorn, int currentYear);

    //const member function
    void OutputFirstName(string nameToOutput) const;

    //Accessor methods
    void SetAge(unsigned int newAge); //Setter

    unsigned int GetAge(); //Getter

};
```



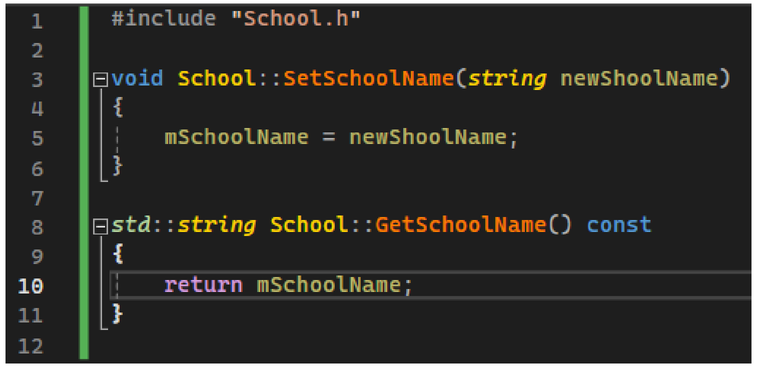
```
void Student::OutputFirstName(string nameToOutput) const
{
    //Since this is a member variable it will result in an error
    mFirstName = nameToOutput;

    //Since this a local variable, it is legal
    string localName = nameToOutput;

    cout << nameToOutput << endl;
}
```

Classes

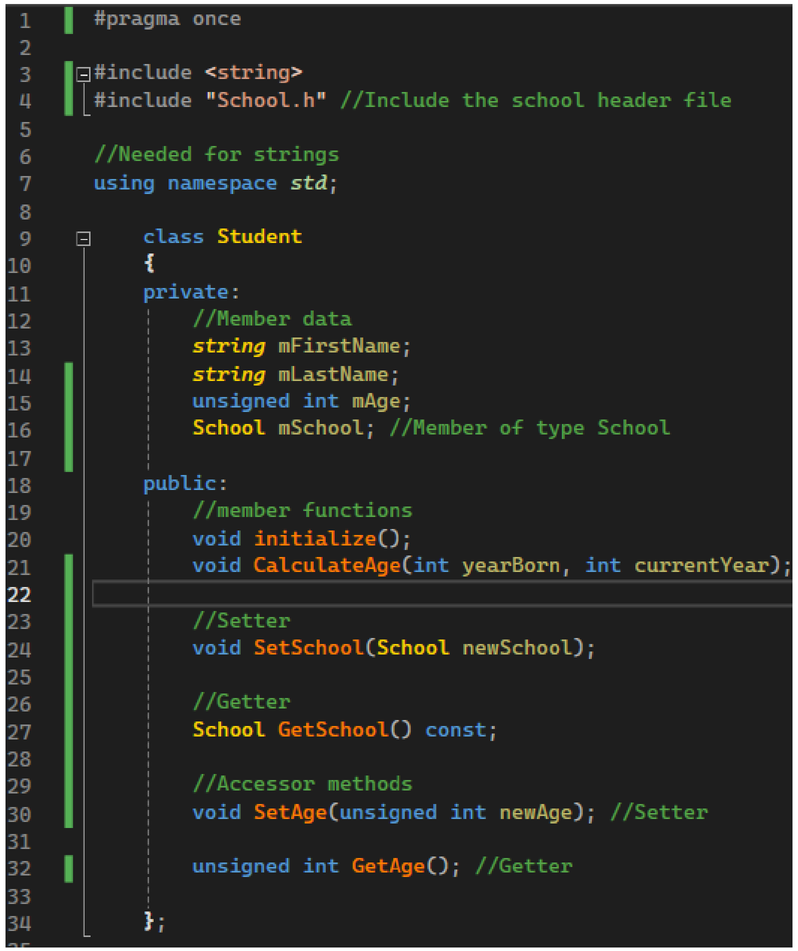
- Other classes as member variables



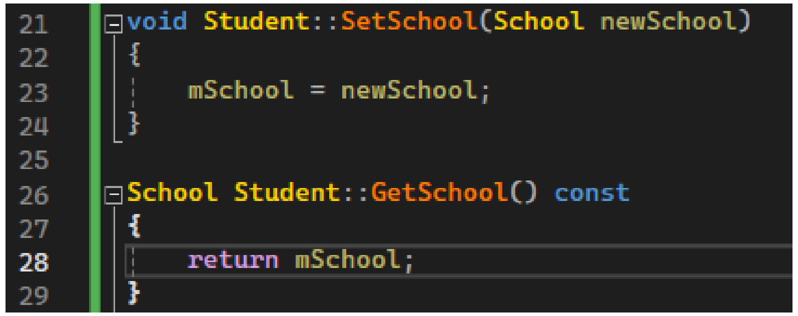
```cpp
#pragma once

#include <string>

using namespace std;

class School
{
private:
    string mSchoolName;

public:
    void SetSchoolName(string newShoolName);

    //const function
    string GetSchoolName() const;

};
```



```cpp
#include "School.h"

void School::SetSchoolName(string newShoolName)
{
    mSchoolName = newShoolName;
}

std::string School::GetSchoolName() const
{
    return mSchoolName;
}

```



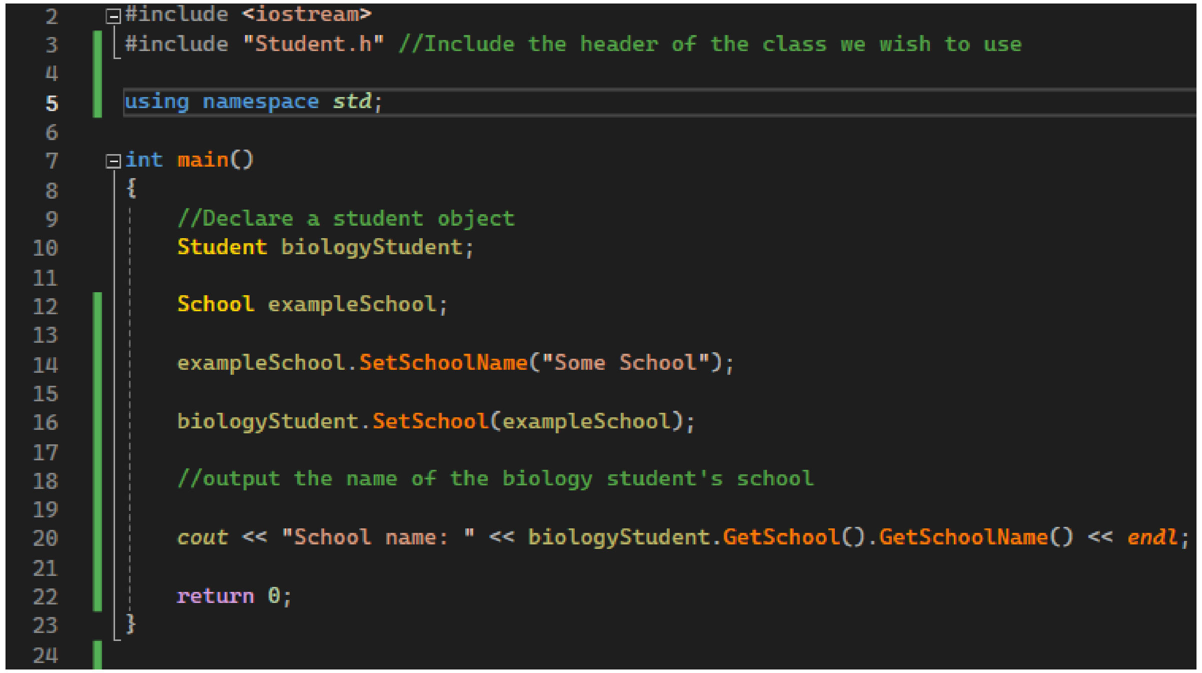
```cpp
#pragma once

#include <string>
#include "School.h" //Include the school header file

//Needed for strings
using namespace std;

    class Student
    {
    private:
        //Member data
        string mFirstName;
        string mLastName;
        unsigned int mAge;
        School mSchool; //Member of type School

    public:
        //member functions
        void initialize();
        void CalculateAge(int yearBorn, int currentYear);

        //Setter
        void SetSchool(School newSchool);

        //Getter
        School GetSchool() const;

        //Accessor methods
        void SetAge(unsigned int newAge); //Setter

        unsigned int GetAge(); //Getter

    };
```



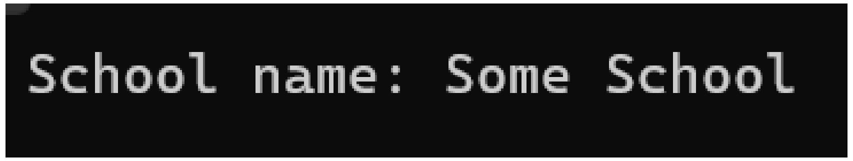
```cpp
void Student::SetSchool(School newSchool)
{
    mSchool = newSchool;
}

School Student::GetSchool() const
{
    return mSchool;
}
```

Classes

- Other classes as member variables



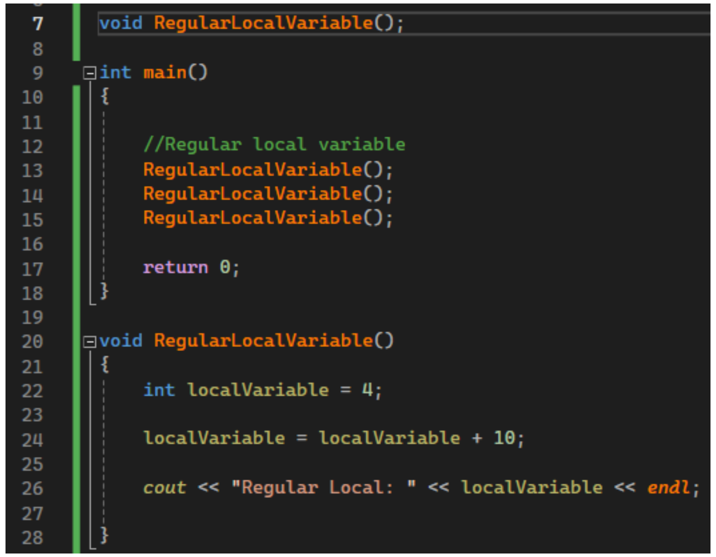
```
#include <iostream>
#include "Student.h" //Include the header of the class we wish to use

using namespace std;

int main()
{
    //Declare a student object
    Student biologyStudent;

    School exampleSchool;

    exampleSchool.SetSchoolName("Some School");

    biologyStudent.SetSchool(exampleSchool);

    //output the name of the biology student's school

    cout << "School name: " << biologyStudent.GetSchool().GetSchoolName() << endl;

    return 0;
}
```



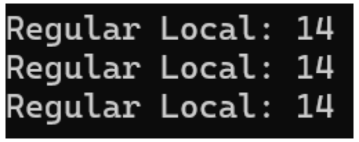
```
School name: Some School
```

Static Variables

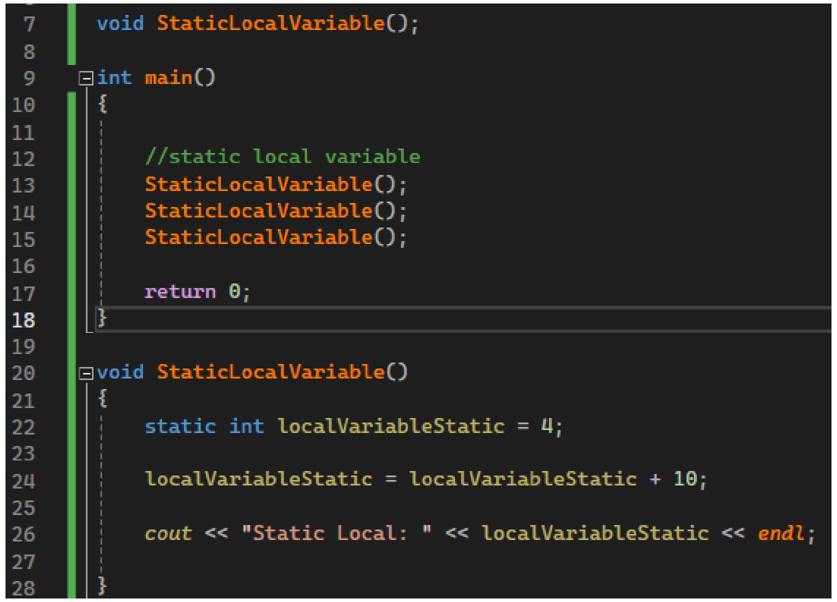
- Stored in the global namespace
- Declared using the ***static*** keyword
- Static Local Variables
  - Permanently stored in the program



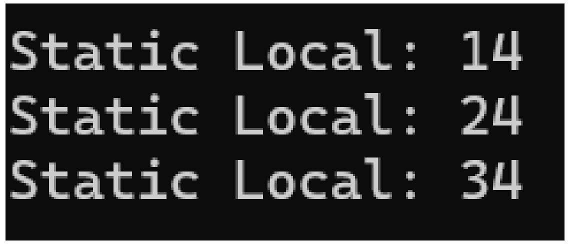
```cpp
void RegularLocalVariable();

int main()
{

    //Regular local variable
    RegularLocalVariable();
    RegularLocalVariable();
    RegularLocalVariable();

    return 0;
}

void RegularLocalVariable()
{
    int localVariable = 4;

    localVariable = localVariable + 10;

    cout << "Regular Local: " << localVariable << endl;

}
```



```
Regular Local: 14
Regular Local: 14
Regular Local: 14
```



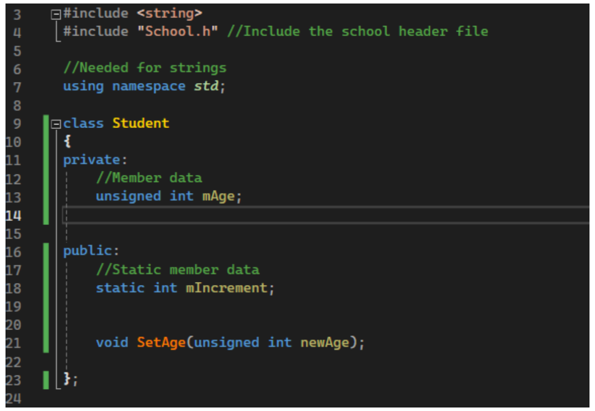
```cpp
void StaticLocalVariable();

int main()
{

    //static local variable
    StaticLocalVariable();
    StaticLocalVariable();
    StaticLocalVariable();

    return 0;
}

void StaticLocalVariable()
{
    static int localVariableStatic = 4;

    localVariableStatic = localVariableStatic + 10;

    cout << "Static Local: " << localVariableStatic << endl;

}
```



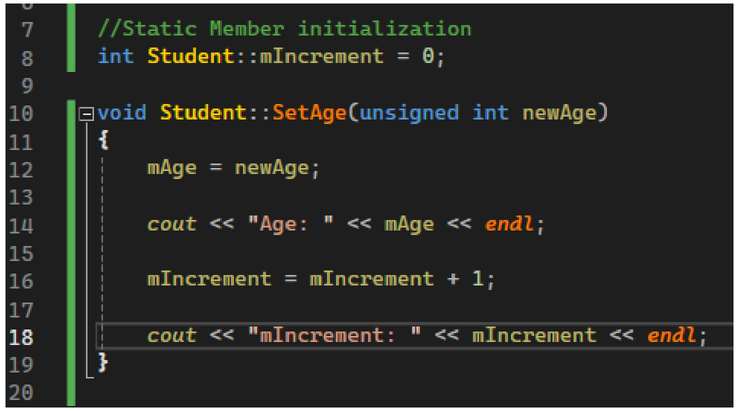
```
Static Local: 14
Static Local: 24
Static Local: 34
```

Static Variables

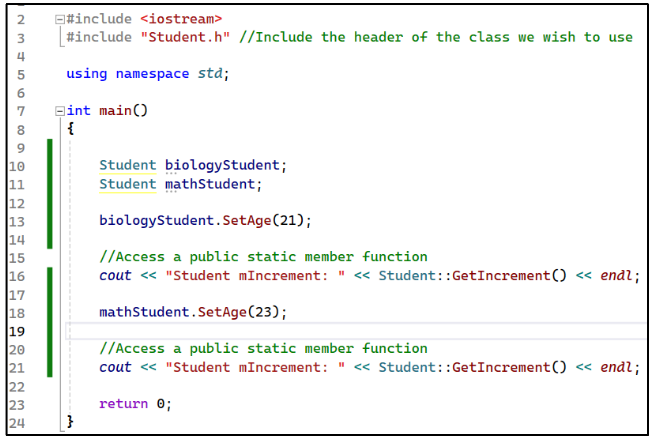
- Static Member Variables
  - A variable that is shared by all objects of a class
  - Do no contribute to the size of class and object



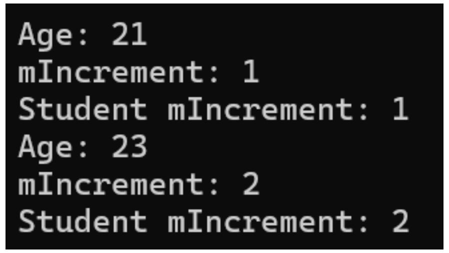
```cpp
#include <string>
#include "School.h" //Include the school header file

//Needed for strings
using namespace std;

class Student
{
private:
	//Member data
	unsigned int mAge;


public:
	//Static member data
	static int mIncrement;


	void SetAge(unsigned int newAge);

};
```



```cpp
//Static Member initialization
int Student::mIncrement = 0;

void Student::SetAge(unsigned int newAge)
{
	mAge = newAge;

	cout << "Age: " << mAge << endl;

	mIncrement = mIncrement + 1;

	cout << "mIncrement: " << mIncrement << endl;
}
```



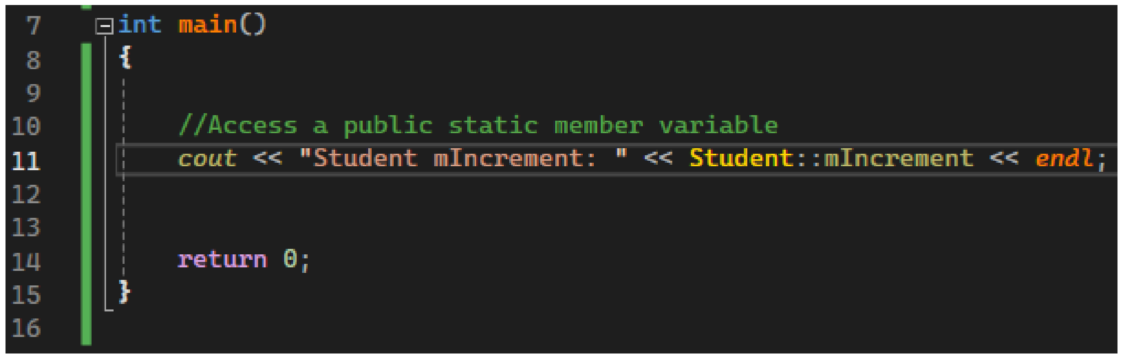
```cpp
#include <iostream>
#include "Student.h" //Include the header of the class we wish to use

using namespace std;

int main()
{

	Student biologyStudent;
	Student mathStudent;

	biologyStudent.SetAge(21);

	//Access a public static member function
	cout << "Student mIncrement: " << Student::GetIncrement() << endl;

	mathStudent.SetAge(23);

	//Access a public static member function
	cout << "Student mIncrement: " << Student::GetIncrement() << endl;

	return 0;
}
```



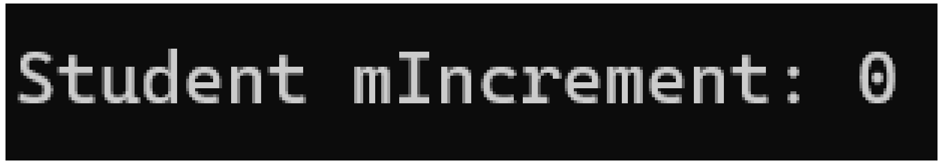
```
Age: 21
mIncrement: 1
Student mIncrement: 1
Age: 23
mIncrement: 2
Student mIncrement: 2
```

Static Variables

- Static Member Variables
  - If public can be accessed without creating an instance
  - Can be accessed from an instance



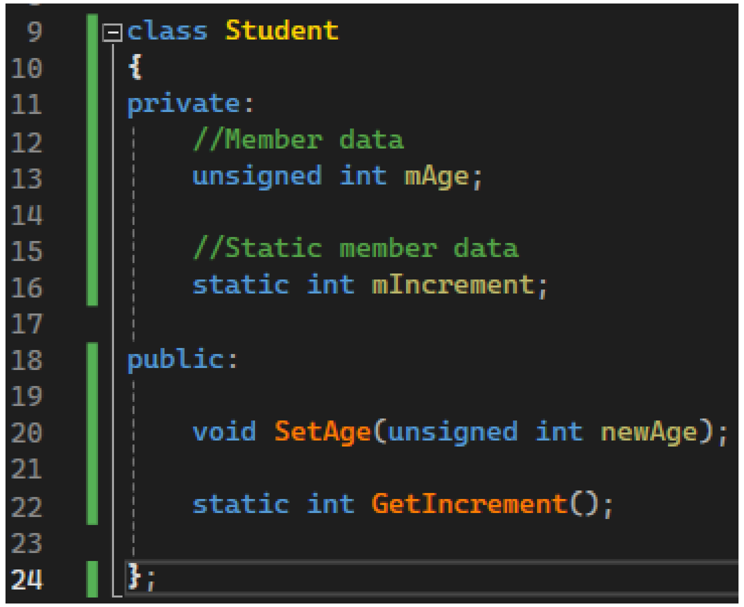
```
int main()
{

    //Access a public static member variable
    cout << "Student mIncrement: " << Student::mIncrement << endl;


    return 0;
}

```



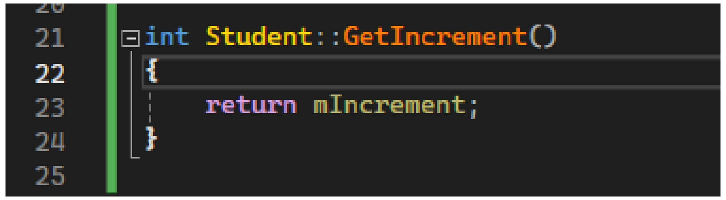
```
Student mIncrement: 0
```

Static Member Functions

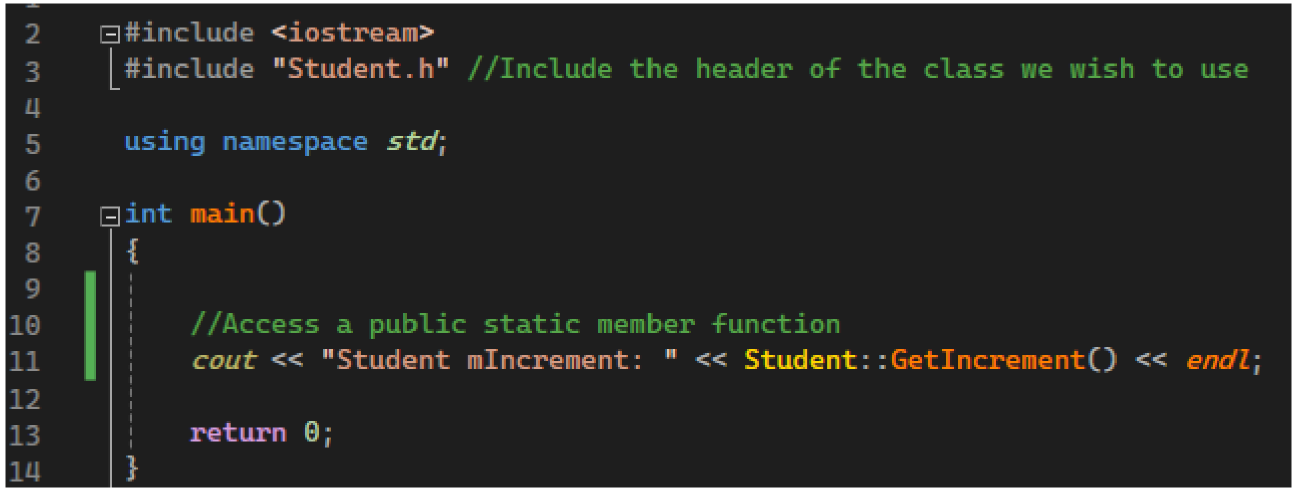
- Static Member Functions
  - Can be accessed without creating objects
  - Can be accessed from an instance
  - Cannot make changes to non-static member variables



```cpp
class Student
{
private:
    //Member data
    unsigned int mAge;

    //Static member data
    static int mIncrement;

public:

    void SetAge(unsigned int newAge);

    static int GetIncrement();

};
```



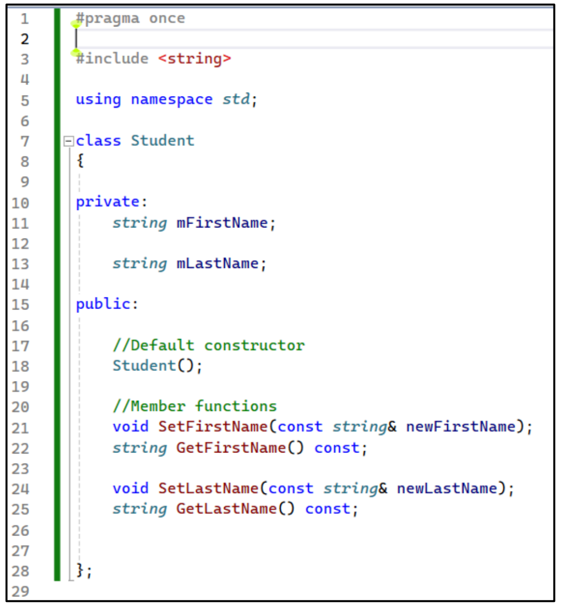
```cpp
int Student::GetIncrement()
{
    return mIncrement;
}

```



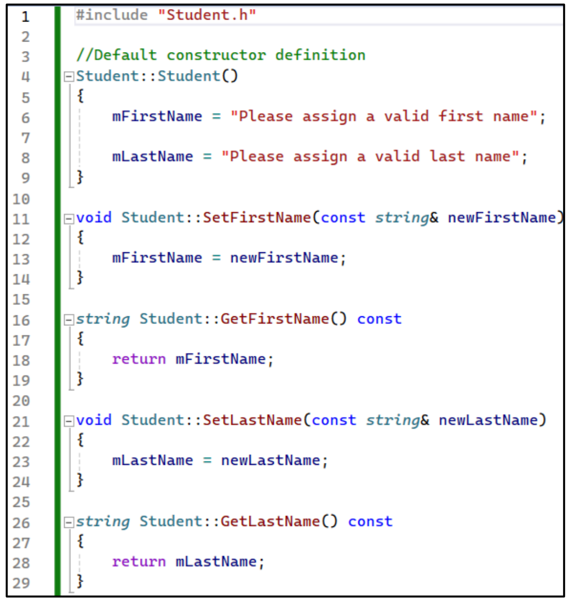
```cpp
#include <iostream>
#include "Student.h" //Include the header of the class we wish to use

using namespace std;

int main()
{

    //Access a public static member function
    cout << "Student mIncrement: " << Student::GetIncrement() << endl;

    return 0;
}
```

Constructor

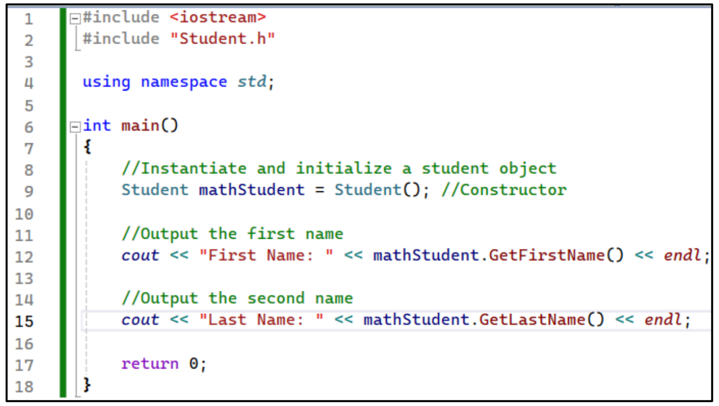
- A constructor is a special member function
  - It has the same name as the class itself
  - Can take parameters
  - Cannot return a value
    - Not even void
- Default constructor
  - Constructor which does not take any arguments



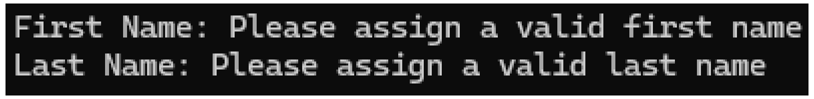
```cpp
#pragma once

#include <string>

using namespace std;

class Student
{

private:
    string mFirstName;

    string mLastName;

public:

    //Default constructor
    Student();

    //Member functions
    void SetFirstName(const string& newFirstName);
    string GetFirstName() const;

    void SetLastName(const string& newLastName);
    string GetLastName() const;


};
```



```cpp
#include "Student.h"

//Default constructor definition
Student::Student()
{
    mFirstName = "Please assign a valid first name";

    mLastName = "Please assign a valid last name";
}

void Student::SetFirstName(const string& newFirstName)
{
    mFirstName = newFirstName;
}

string Student::GetFirstName() const
{
    return mFirstName;
}

void Student::SetLastName(const string& newLastName)
{
    mLastName = newLastName;
}

string Student::GetLastName() const
{
    return mLastName;
}
```



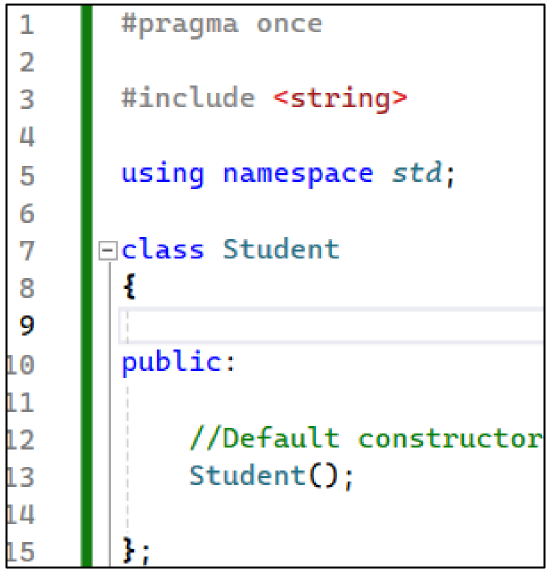
```cpp
#include <iostream>
#include "Student.h"

using namespace std;

int main()
{
    //Instantiate and initialize a student object
    Student mathStudent = Student(); //Constructor

    //Output the first name
    cout << "First Name: " << mathStudent.GetFirstName() << endl;

    //Output the second name
    cout << "Last Name: " << mathStudent.GetLastName() << endl;

    return 0;
}
```



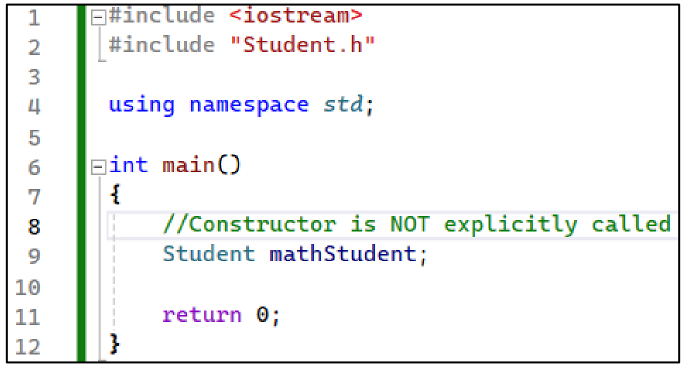
```
First Name: Please assign a valid first name
Last Name: Please assign a valid last name
```

Constructor

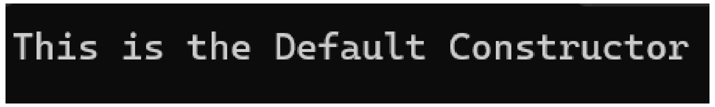
- Default constructor
  - If no constructor is created, then the compiler provides a default constructor
    - Appears to do nothing but is required when creating objects
      - Constructors are called as part of the object creation process
      - The default constructor is called when a constructor is not explicitly called



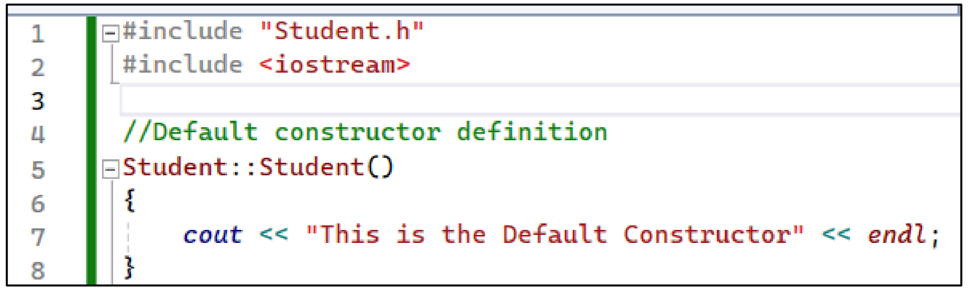
```cpp
#pragma once

#include <string>

using namespace std;

class Student
{

public:

    //Default constructor
    Student();

};
```



```cpp
#include <iostream>
#include "Student.h"

using namespace std;

int main()
{
    //Constructor is NOT explicitly called
    Student mathStudent;

    return 0;
}
```



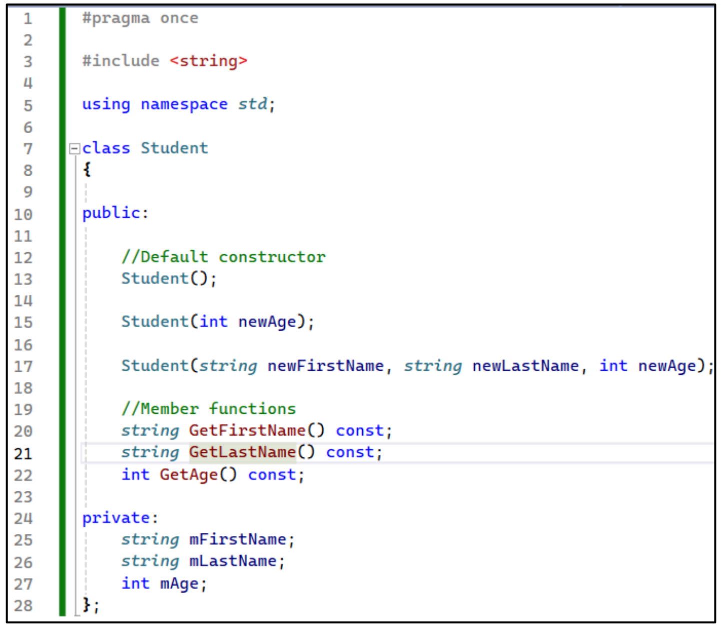
```
This is the Default Constructor
```



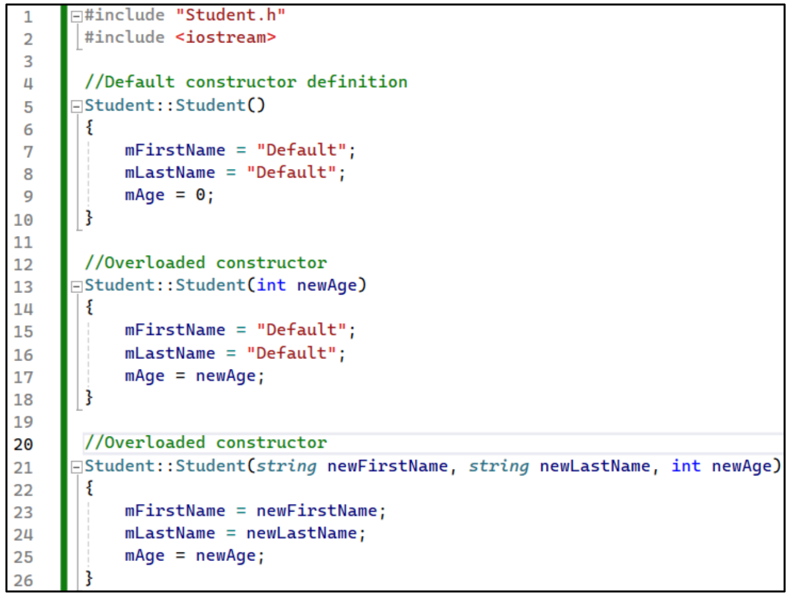
```cpp
#include "Student.h"
#include <iostream>

//Default constructor definition
Student::Student()
{
    cout << "This is the Default Constructor" << endl;
}
```

Constructor

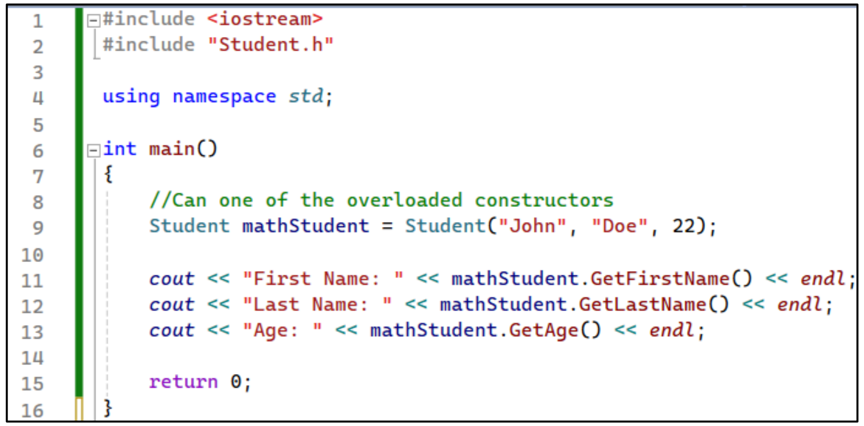
- Constructors with parameters
  - Constructors can be overloaded



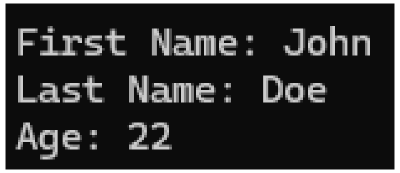
```cpp
#pragma once

#include <string>

using namespace std;

class Student
{

public:

    //Default constructor
    Student();

    Student(int newAge);

    Student(string newFirstName, string newLastName, int newAge);

    //Member functions
    string GetFirstName() const;
    string GetLastName() const;
    int GetAge() const;

private:
    string mFirstName;
    string mLastName;
    int mAge;
};
```



```cpp
#include "Student.h"
#include <iostream>

//Default constructor definition
Student::Student()
{
    mFirstName = "Default";
    mLastName = "Default";
    mAge = 0;
}

//Overloaded constructor
Student::Student(int newAge)
{
    mFirstName = "Default";
    mLastName = "Default";
    mAge = newAge;
}

//Overloaded constructor
Student::Student(string newFirstName, string newLastName, int newAge)
{
    mFirstName = newFirstName;
    mLastName = newLastName;
    mAge = newAge;
}
```



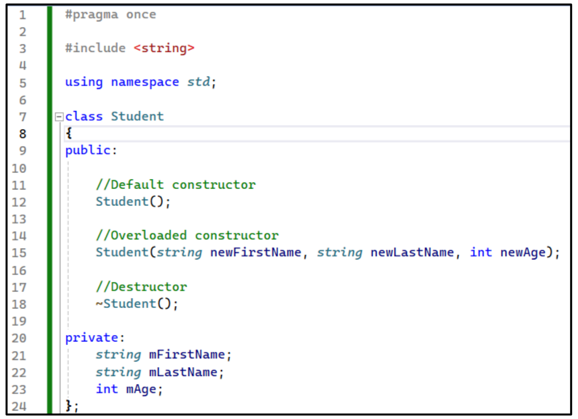
```cpp
#include <iostream>
#include "Student.h"

using namespace std;

int main()
{
    //Can one of the overloaded constructors
    Student mathStudent = Student("John", "Doe", 22);

    cout << "First Name: " << mathStudent.GetFirstName() << endl;
    cout << "Last Name: " << mathStudent.GetLastName() << endl;
    cout << "Age: " << mathStudent.GetAge() << endl;

    return 0;
}
```



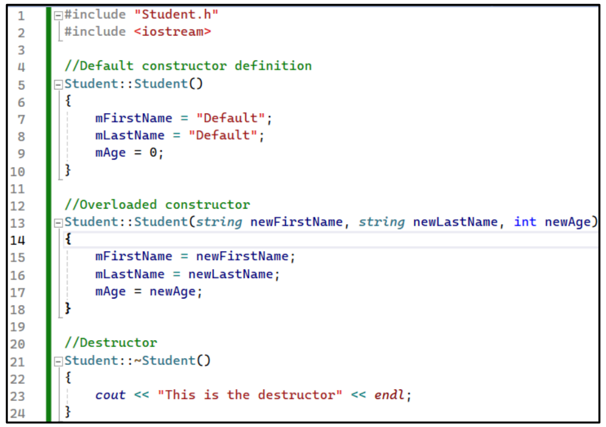
```
First Name: John
Last Name: Doe
Age: 22
```

Destructors

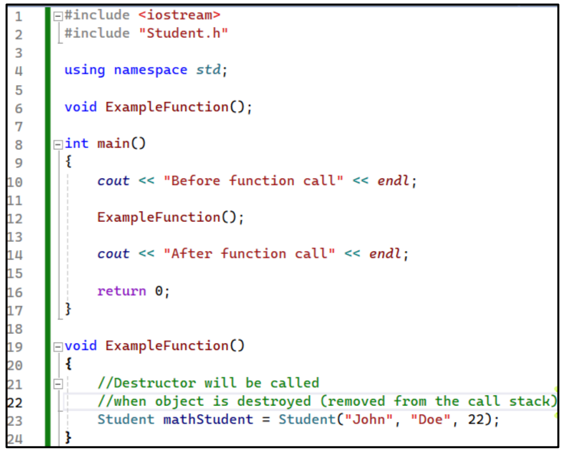
- A destructor is a special member function
  - Cleans up after objects
  - Has the same name as the class itself
    - Name is preceded by a tilde (~)
  - Does not take any arguments
  - Cannot return a value
    - Not even void
  - There is only one destructor
    - Cannot be overloaded
  - Cannot be explicitly called



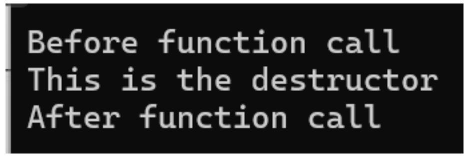
```cpp
#pragma once

#include <string>

using namespace std;

class Student
{
public:

    //Default constructor
    Student();

    //Overloaded constructor
    Student(string newFirstName, string newLastName, int newAge);

    //Destructor
    ~Student();

private:
    string mFirstName;
    string mLastName;
    int mAge;
};
```



```cpp
#include "Student.h"
#include <iostream>

//Default constructor definition
Student::Student()
{
    mFirstName = "Default";
    mLastName = "Default";
    mAge = 0;
}

//Overloaded constructor
Student::Student(string newFirstName, string newLastName, int newAge)
{
    mFirstName = newFirstName;
    mLastName = newLastName;
    mAge = newAge;
}

//Destructor
Student::~Student()
{
    cout << "This is the destructor" << endl;
}
```



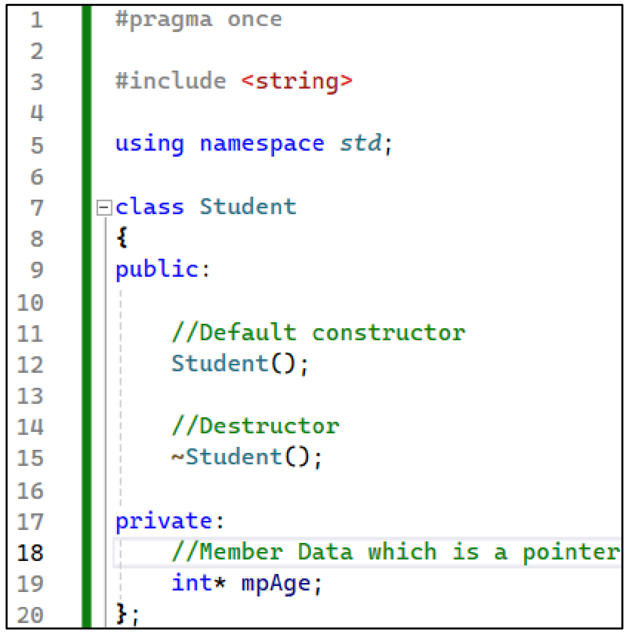
```cpp
#include <iostream>
#include "Student.h"

using namespace std;

void ExampleFunction();

int main()
{
    cout << "Before function call" << endl;

    ExampleFunction();

    cout << "After function call" << endl;

    return 0;
}

void ExampleFunction()
{
    //Destructor will be called
    //when object is destroyed (removed from the call stack)
    Student mathStudent = Student("John", "Doe", 22);
}
```



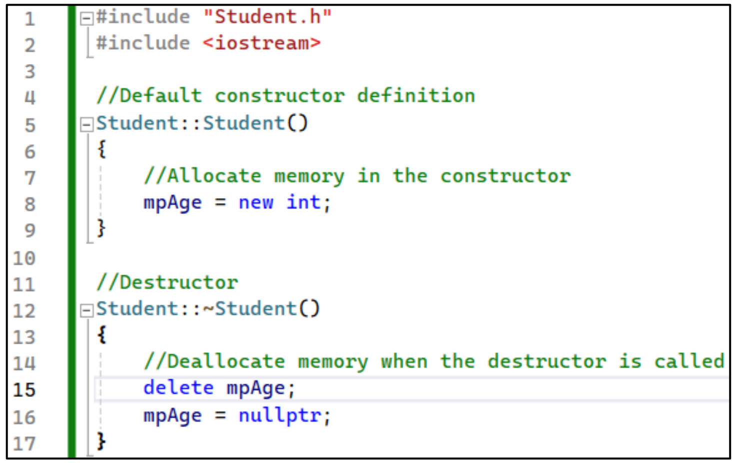
```
Before function call
This is the destructor
After function call
```

Destructors

- Can be used to free up memory allocated within the class



```cpp
#pragma once

#include <string>

using namespace std;

class Student
{
public:

    //Default constructor
    Student();

    //Destructor
    ~Student();

private:
    //Member Data which is a pointer
    int* mpAge;
};
```



```cpp
#include "Student.h"
#include <iostream>

//Default constructor definition
Student::Student()
{
    //Allocate memory in the constructor
    mpAge = new int;
}

//Destructor
Student::~Student()
{
    //Deallocate memory when the destructor is called
    delete mpAge;
    mpAge = nullptr;
}
```

Constructor and Destructor

- When you create a constructor, you must also create a destructor
  - Even if the destructor does nothing
- It is recommended to define a constructor
  - Used to set member variables to appropriate defaults
    - Ensure that the object behaves correctly