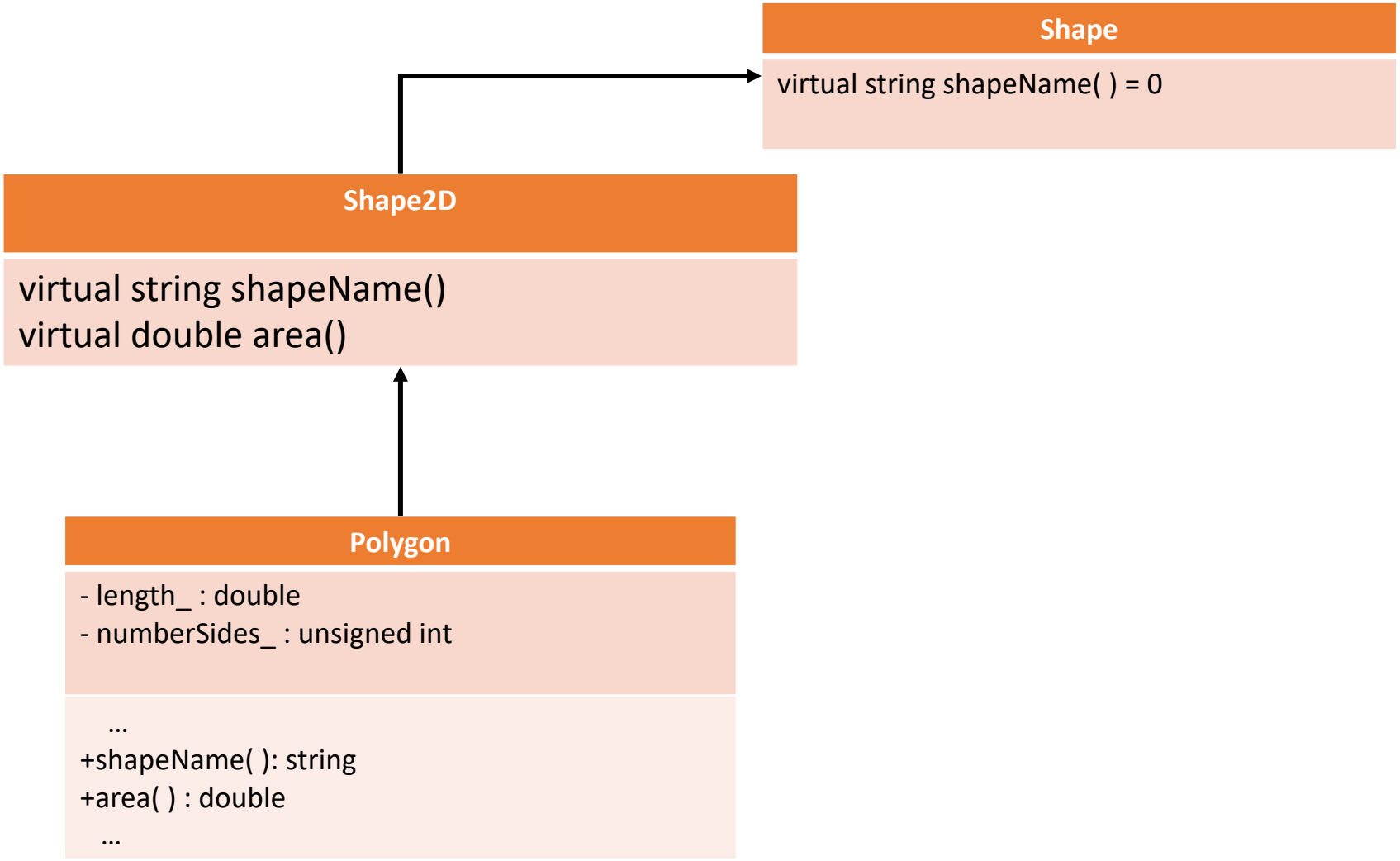
Review

Object-Oriented Programming Concepts

&

Lifecycle of a C++ object.


Shape
virtual string shapeName( ) = 0
Shape2D
virtual string shapeName()
virtual double area()
Polygon
- length_ : double
- numberSides_ : unsigned int
...
+shapeName( ): string
+area( ) : double
...

Abstraction

Abstract class in C++ is the one which is **not used** to create objects. These type of classes are designed only to treat like a base class (to be inherited by other classes). It is a designed technique for program development which allows making a base upon which other classes may be built.

Abstract class in C++ is a class that has at least *one* pure virtual function (i.e., a function that has no definition). The classes inheriting the abstract class *must* provide a definition for the pure virtual function; otherwise, the subclass would become an abstract class itself.

Pure Virtual Function & Virtual Function

Shape.h | main.cpp | libraries.h | Makefile | Polygon.h | Polygon.cpp

Shape.h > ...

```cpp
#pragma once
// #include is a way of including a standard or user-defined file in the program
#include "libraries.h"

// Abstract class
class Shape{
public:

  // Pure Virtual Function
  virtual string shapeName() = 0;

};

class Shape2D : public Shape{

public:
    //Virtual Function
    virtual string shapeName(){ return "Shape2D";}
    //Pure Virtual Function
    virtual double area(){ return 0.0;}

};
```

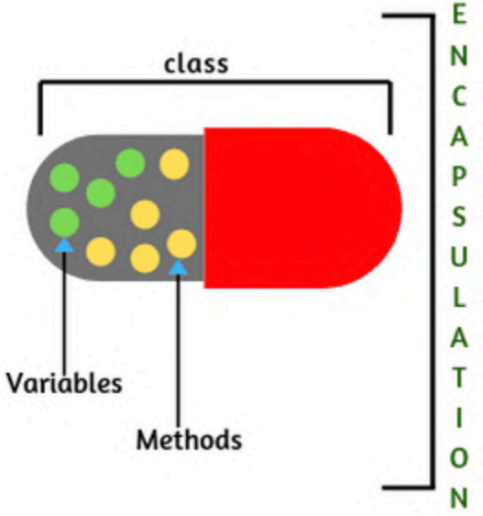
A pure virtual function (or abstract function) in C++ is a virtual function for which we don't have an implementation, we only declare it. A pure virtual function is declared by assigning 0 in the declaration.

A virtual function is a member function re-defined(Overriden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

2022-06-21

master*+ | 0 ⚠ 0 | Live Share | Ln 34, Col 3 | Spaces: 2 | UTF-8 | LF | C++ | Mac

Encapsulation

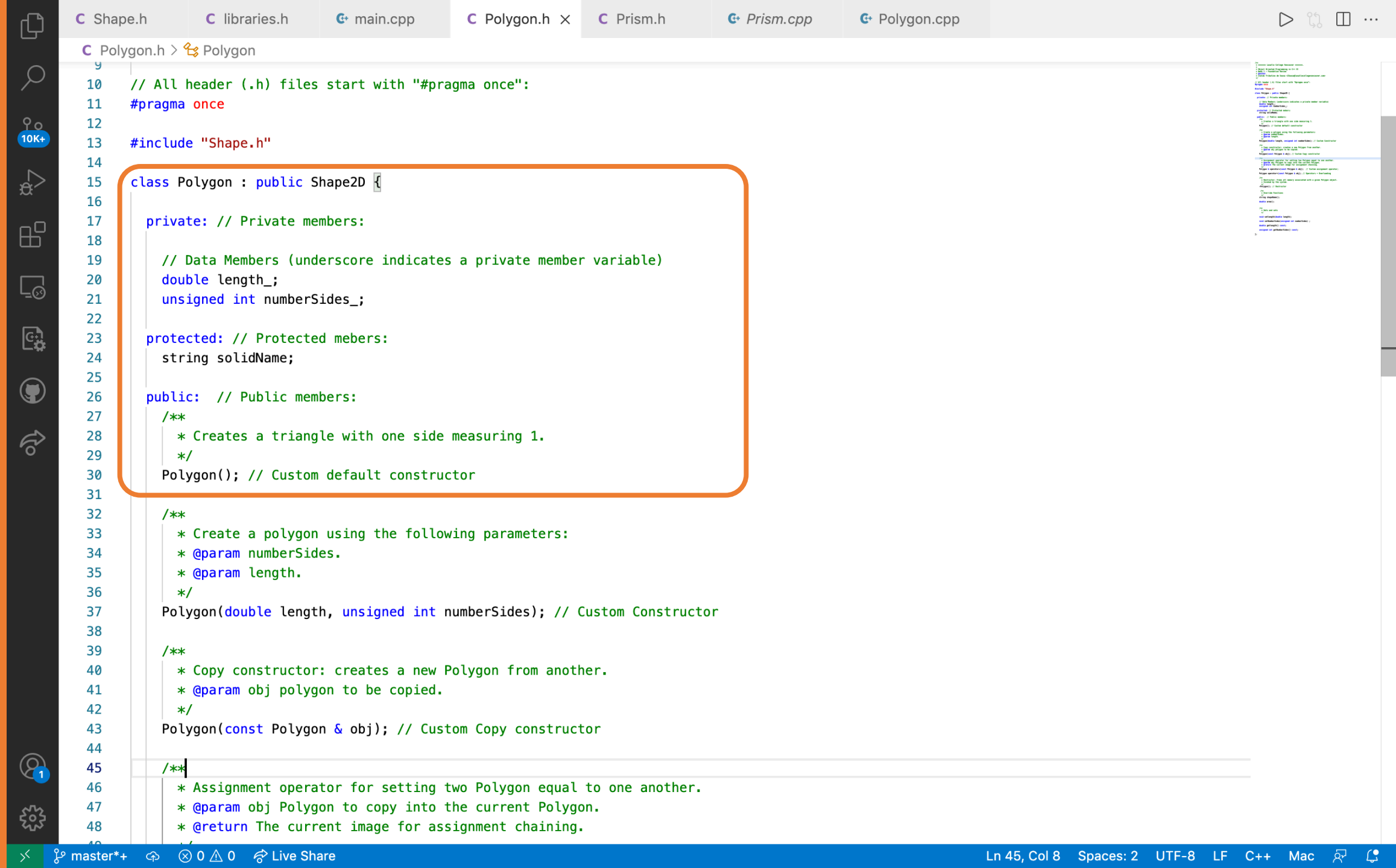


Encapsulation is the process of hiding information details and protecting data and behavior of an object from misuse by other objects. In C++ encapsulation can be implemented using Class and access modifiers.

There are 3 types of access modifiers available in C++:

1. Public	All the class members declared under the public specifier will be available to everyone.
2. Private	The class members declared as private can be accessed only by the member functions inside the class.
3. Protected	Protected access modifier is similar to private access modifier. However, it can be accessed by any subclass(derived class) of that class as well.

Encapsulation



Encapsulation

Shape.h | libraries.h | main.cpp | Makefile | Polygon.h | Prism.h | Polygon.cpp

main.cpp > main()

```
43
44  int main() {
45
46    //************************************************
47    //       -------- Encapsulation --------
48    //************************************************
49    cout << "-------- Encapsulation --------" << endl;
50
51    Polygon poly;
52
53    cout << poly.length_ << endl;
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
```

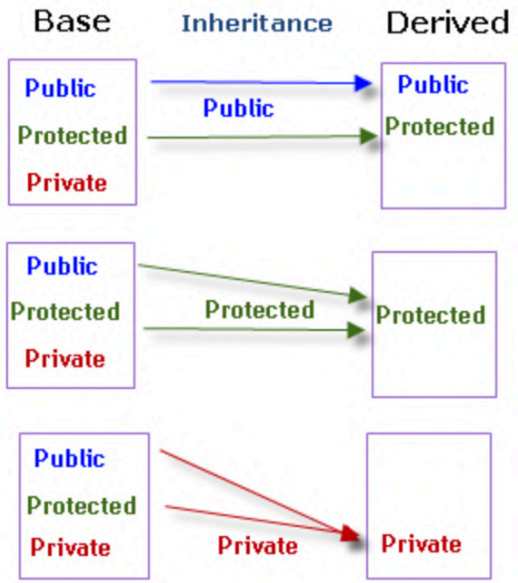
PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
main.cpp:53:16: error: 'length_' is a private member of 'Polygon'
  cout << poly.length_ << endl;
               ^
./Polygon.h:20:12: note: declared private here
    double length_;
           ^
1 error generated.
make: *** [main.o] Error 1
(base) Ivaldo:Week1 admin$
```

master*+ | 0 ⚠ 0 | Live Share | Ln 53, Col 32 | Spaces: 2 | UTF-8 | LF | C++ | Mac

Inheritance



Inheritance is a mechanism where a new class is derived from an existing class.

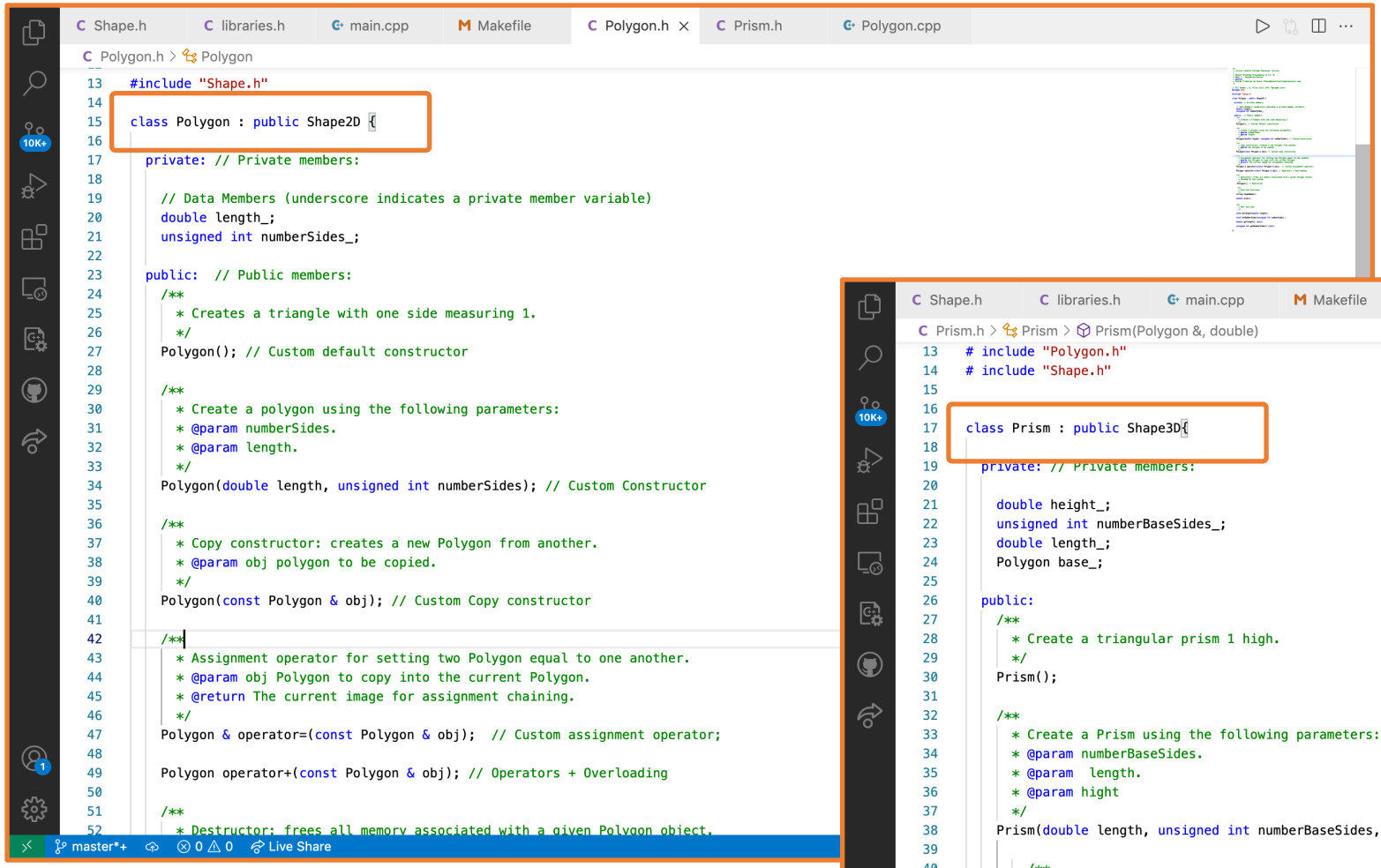
Sub Class: The class that inherits properties from another class is called Sub class or Derived Class.
Super Class: The class whose properties are inherited by sub class is called Base Class or Super class.

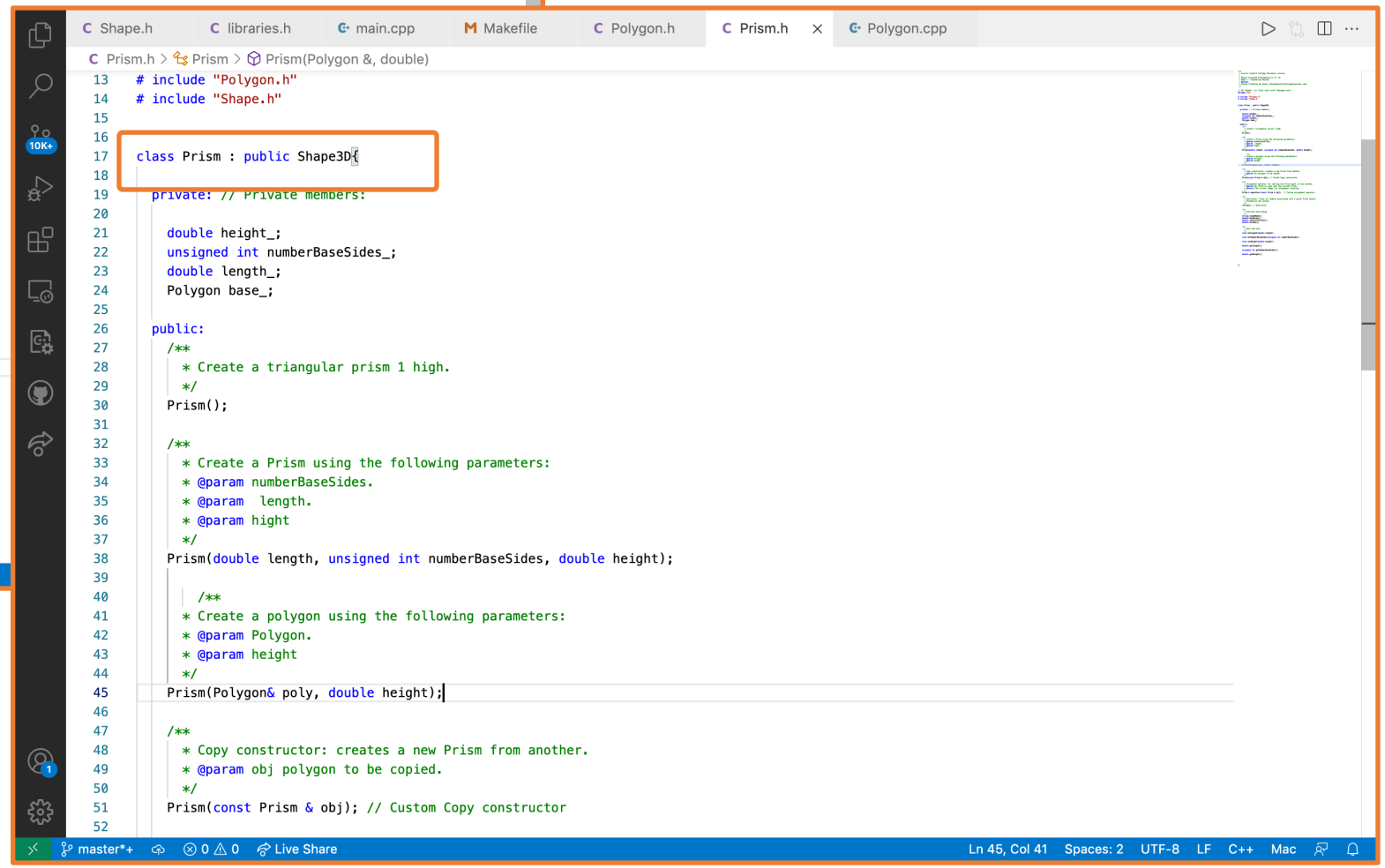
Modes of Inheritance:

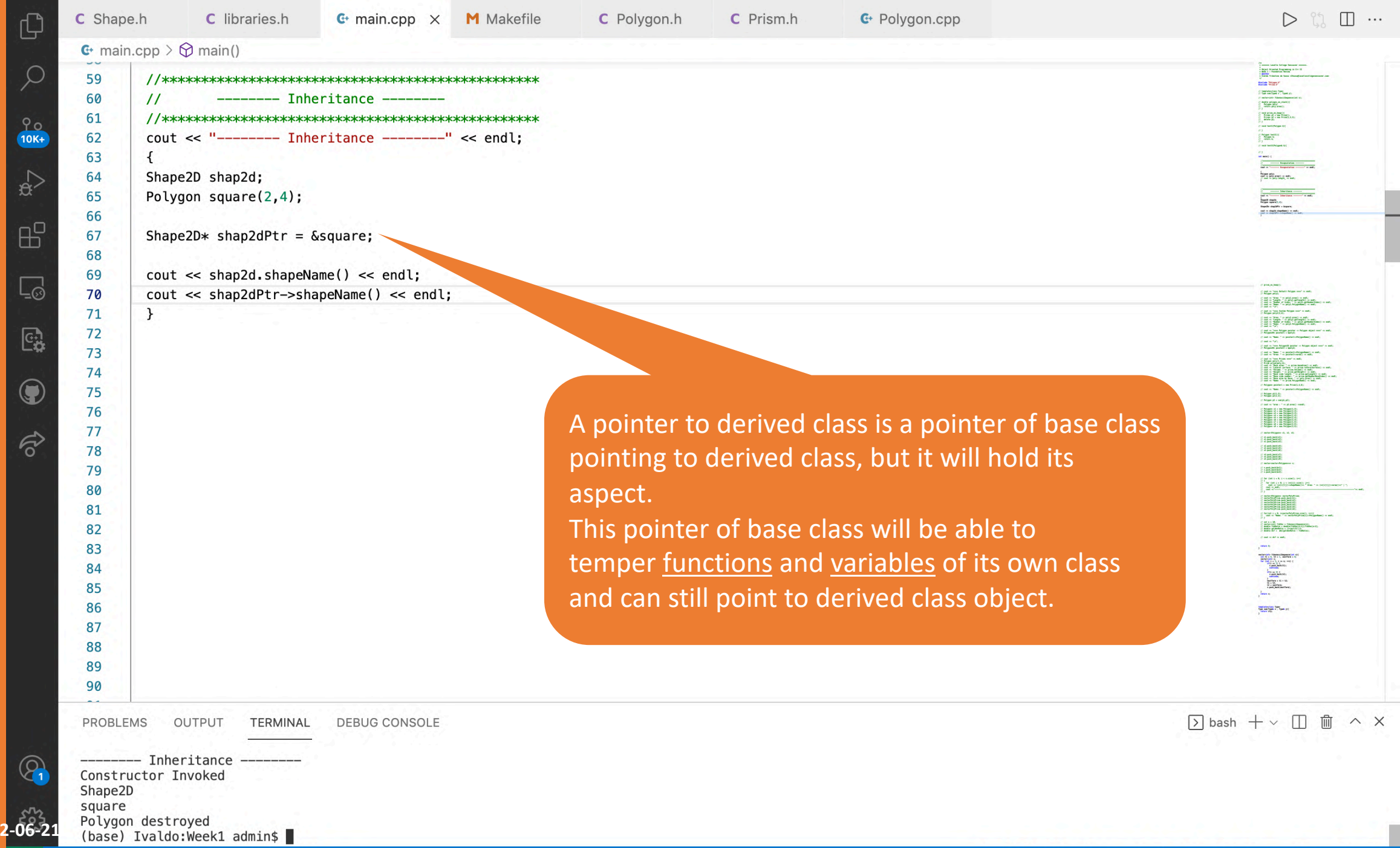
1.Public mode: If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.

2.Protected mode: If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.

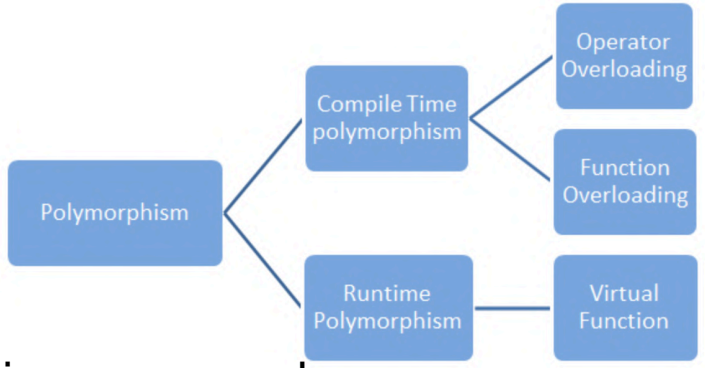
3.Private mode: If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in derived class.


Shape.h
libraries.h
main.cpp
Makefile
Polygon.h
Prism.h
Polygon.cpp
Polygon.h > Polygon
#include "Shape.h"
class Polygon : public Shape2D {
private: // Private members:
// Data Members (underscore indicates a private member variable)
double length_;
unsigned int numberSides_;
public: // Public members:
/**
* Creates a triangle with one side measuring 1.
*/
Polygon(); // Custom default constructor
/**
* Create a polygon using the following parameters:
* @param numberSides.
* @param length.
*/
Polygon(double length, unsigned int numberSides); // Custom Constructor
/**
* Copy constructor: creates a new Polygon from another.
* @param obj polygon to be copied.
*/
Polygon(const Polygon & obj); // Custom Copy constructor
/**
* Assignment operator for setting two Polygon equal to one another.
* @param obj Polygon to copy into the current Polygon.
* @return The current image for assignment chaining.
*/
Polygon & operator=(const Polygon & obj); // Custom assignment operator;
Polygon operator+(const Polygon & obj); // Operators + Overloading
/**
* Destructor: frees all memory associated with a given Polygon object.
master*+
0 0
Live Share


Shape.h
libraries.h
main.cpp
Makefile
Polygon.h
Prism.h
Polygon.cpp
Prism.h > Prism > Prism(Polygon &, double)
# include "Polygon.h"
# include "Shape.h"
class Prism : public Shape3D{
private: // Private members:
double height_;
unsigned int numberBaseSides_;
double length_;
Polygon base_;
public:
/**
* Create a triangular prism 1 high.
*/
Prism();
/**
* Create a Prism using the following parameters:
* @param numberBaseSides.
* @param length.
* @param hight
*/
Prism(double length, unsigned int numberBaseSides, double height);
/**
* Create a polygon using the following parameters:
* @param Polygon.
* @param height
*/
Prism(Polygon& poly, double height);
/**
* Copy constructor: creates a new Prism from another.
* @param obj polygon to be copied.
*/
Prism(const Prism & obj); // Custom Copy constructor
master*+
0 0
Live Share
Ln 45, Col 41
Spaces: 2
UTF-8
LF
C++
Mac


Shape.h
libraries.h
main.cpp
Makefile
Polygon.h
Prism.h
Polygon.cpp
main.cpp > main()
59 //***
60 //   -------- Inheritance --------
61 //***
62 cout << "-------- Inheritance --------" << endl;
63 {
64 Shape2D shap2d;
65 Polygon square(2,4);
66
67 Shape2D* shap2dPtr = □
68
69 cout << shap2d.shapeName() << endl;
70 cout << shap2dPtr->shapeName() << endl;
71 }
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
A pointer to derived class is a pointer of base class pointing to derived class, but it will hold its aspect.
This pointer of base class will be able to temper functions and variables of its own class and can still point to derived class object.
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
bash
-------- Inheritance --------
Constructor Invoked
Shape2D
square
Polygon destroyed
(base) Ivaldo:Week1 admin$
master*+
0 0
Live Share
Ln 70, Col 32
Spaces: 2
UTF-8
LF
C++
Mac

Polymorphism



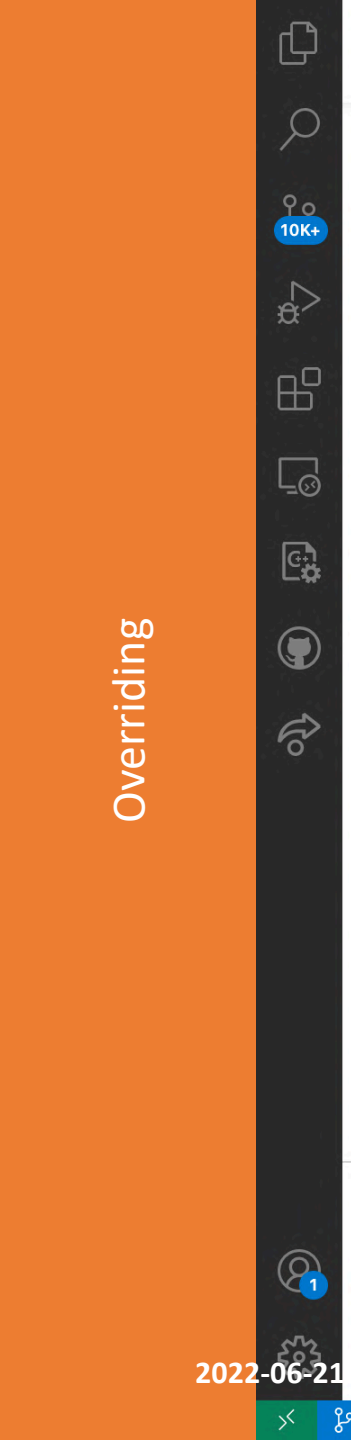
The word **polymorphism** means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

In C++ polymorphism is mainly divided into two types:

Polymorphism	Compile time	Runtime
Basis	**Function Overloading**	**Function Overriding**
Purpose	To have multiple functions with same name that act differently depending on parameters	To perform additional or different tasks than the base class function

Overriding



Shape.h | libraries.h | main.cpp | Makefile | Polygon.h | Prism.h | Polygon.cpp

Shape.h > ...

```
13  #pragma once
14  // #include is a way of including a standard or user-defined file in the program
15  #include "libraries.h"
16
17  // Abstract class
18  class Shape{
19  public:
20
21    // Pure Virtual Function
22    virtual string shapeName() = 0;
23
24  };
25
26  class Shape2D : public Shape{
27
28  public:
29      //Virtual Function
30      virtual string shapeName(){ return "Shape2D";}
31      //Pure Virtual Function
32      virtual double area(){ return 0.0;}
33
34  };
35
36  class Shape3D : public Shape{
37  public:
38      //Virtual Function
39      virtual string shapeName(){return "Shape3D";};
40      //Pure Virtual Functions
41      virtual double baseArea()=0;
42      virtual double lateralSurface()=0;
43      virtual double Volume()=0;
44
45  };
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
-------- Inheritance --------
Constructor Invoked
Shape2D
square
Polygon destroyed
(base) Ivaldo:Week1 admin$
```

2022-06-21

master*+ | 0 ⚠ 0 | Live Share | Ln 45, Col 3 | Spaces: 2 | UTF-8 | LF | C++ | Mac

Overriding

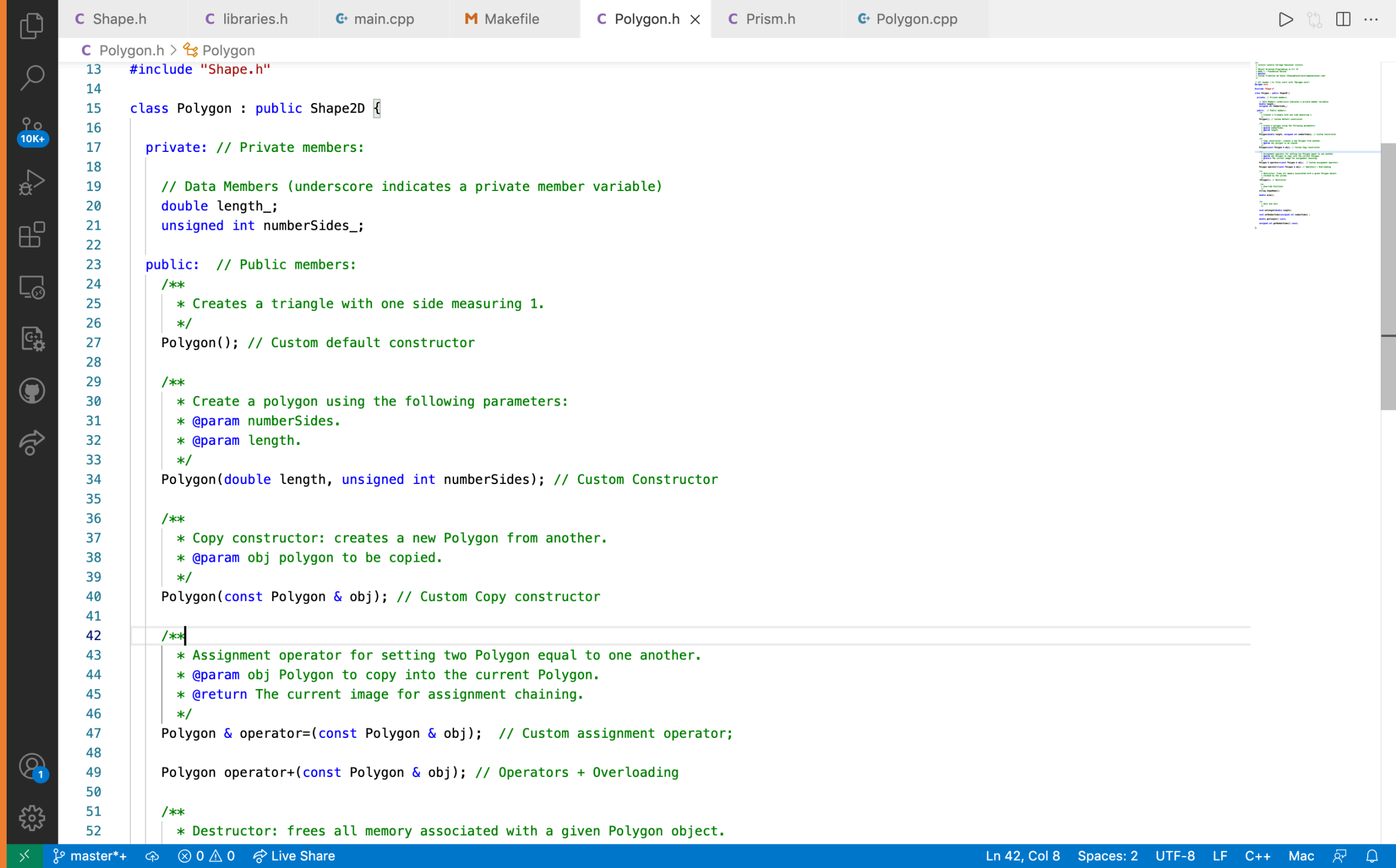
Shape.h | libraries.h | main.cpp | Makefile | Polygon.h | Prism.h | Polygon.cpp

Polygon.cpp > setNumberSides(unsigned int)

```
45    A.numberSides_ = this->numberSides_ + obj.numberSides_;
46    return A;
47  }
48
49  Polygon::~Polygon() {
50      std::cout << "Polygon destroyed" << std::endl;
51  }
52
53  double Polygon:: area(){
54    double perimeter = numberSides_*length_;
55    double apothem = (length_)/(2*tan(PI/numberSides_));
56    return perimeter*apothem/2;
57  }
58
59  string Polygon::shapeName() {
60
61    string arrayName[6] = {"triangle" , "square", "pentagon",
62    "hexagon", "heptagon", "octagon"};
63
64    string name = (numberSides_<9)? arrayName[numberSides_-3]: to_string(numberSides_)+"_polygon";
65
66    return name;
67  }
68
69  void Polygon::setlength(double length) {
70    if (length>0){
71      length_ = length;
72    }
73    else{
74      std::cout << "Please, set a value greater than 0" << std::endl;
75    };
76  }
77
78  void Polygon :: setNumberSides(unsigned int numberSides) {
79
80    if (numberSides>2){
81      numberSides_ = numberSides;
82    }
83    else{
84      std::cout << "Please, only set values above 2." << std::endl;
```

master*+ | 0 ⚠ 0 | Live Share | Ln 80, Col 22 | Spaces: 2 | UTF-8 | LF | C++ | Mac

Overloading



Overloading

Shape.h | main.cpp | libraries.h | Makefile | Polygon.h | Polygon.cpp

Polygon.cpp > operator+(const Polygon &)

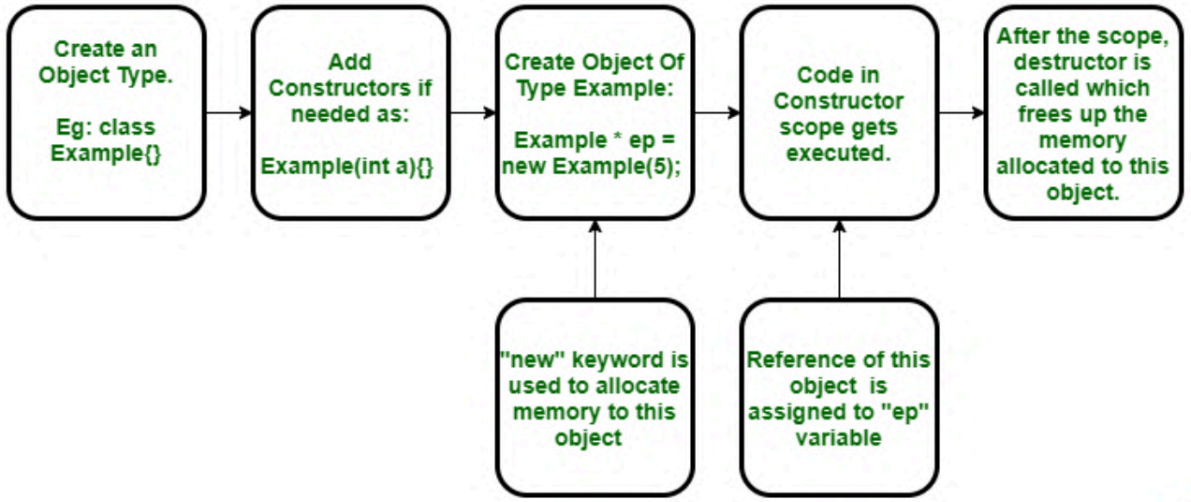
```cpp
// #define Allows the programmer to give a name to a constant value before the program is compiled
#define PI 3.14159265


Polygon :: Polygon(){
  length_ = 1;
  numberSides_ = 3;
  std::cout << "Default Constructor Invoked"  << std::endl;
}


Polygon :: Polygon(double length, unsigned int numberSides){
  length_ = (length>0)? length: 1;
  numberSides_ = (numberSides>2)? numberSides : 3;
  std::cout << "Constructor Invoked"  << std::endl;
}

Polygon :: Polygon(const Polygon & obj){
  length_ = obj.length_;
  std::cout << "Copy Constructor Invoked"  << std::endl;
}

Polygon & Polygon :: operator=(const Polygon & obj){
  length_ = obj.length_;
  numberSides_ = obj.numberSides_;
  std::cout << "Assignment operator invoked"  << std::endl;
  return *this; // dereferenced pointer
}

Polygon Polygon :: operator+(const Polygon & obj){
  Polygon A;
  A.length_ = this->length_ + obj.length_;
  A.numberSides_ = this->numberSides_ + obj.numberSides_;
  return A;
}

Polygon::~Polygon() {
    std::cout << "Polygon destroyed" << std::endl;
}

```

master*+ | 0 ⚠ 0 | Live Share | Ln 44, Col 13 | Spaces: 2 | UTF-8 | LF | C++ | Mac

Lifecycle of a C++ object.

Key Concepts

- Constructors and destructors
- Methods and operators
- Access through pointers and references



Class Constructor

Custom Default Constructor	Custom Constructor
Specifies the states of the object when the object is constructed. It is defined by creating: • A member function with the same name of the class; • The function take zero parameters; • The function does not have a return type. **Polygon :: Polygon( );**	Arguments will be provided. **Polygon :: Polygon(double length, int numberSides);**

Shape.h | libraries.h | main.cpp | Makefile | Polygon.h | Prism.h | Polygon.cpp

Polygon.h > Polygon

```
14
15  class Polygon : public Shape2D {
16
17    private: // Private members:
18
19      // Data Members (underscore indicates a private member variable)
20      double length_;
21      unsigned int numberSides_;
22
23    public:  // Public members:
24      /**
25        * Creates a triangle with one side measuring 1.
26        */
27      Polygon(); // Custom default constructor
28
29      /**
30        * Create a polygon using the following parameters:
31        * @param numberSides.
32        * @param length.
33        */
34      Polygon(double length, unsigned int numberSides); // Custom Constructor
35
36      /**
37        * Copy constructor: creates a new Polygon from another.
38        * @param obj polygon to be copied.
39        */
40      Polygon(const Polygon & obj); // Custom Copy constructor
41
42      /**
43        * Assignment operator for setting two Polygon equal to one another.
44        * @param obj Polygon to copy into the current Polygon.
45        * @return The current image for assignment chaining.
46        */
47      Polygon & operator=(const Polygon & obj);  // Custom assignment operator;
48
49      Polygon operator+(const Polygon & obj); // Operators + Overloading
50
51      /**
52        * Destructor: frees all memory associated with a given Polygon object.
53        * Invoked by the system.
```

master*+ ⊗ 0 ⚠ 0 Live Share

Shape.h | libraries.h | main.cpp | Makefile | Polygon.h | Prism.h | Polygon.cpp

Polygon.cpp > setNumberSides(unsigned int)

```
7   * Ivaldo Tributino de Sousa <ISousa@lasallecollegevancouver.com>
8   */
9
10  // #include is a way of including a standard or user-defined file in the program
11
12  #include "Polygon.h"
13
14  // #define Allows the programmer to give a name to a constant value before the program is compiled
15  #define PI 3.14159265
16
17
18  Polygon :: Polygon(){
19    length_ = 1;
20    numberSides_ = 3;
21    std::cout << "Default Constructor Invoked"  << std::endl;
22  }
23
24
25  Polygon :: Polygon(double length, unsigned int numberSides){
26    length_ = (length>0)? length: 1;
27    numberSides_ = (numberSides>2)? numberSides : 3;
28    std::cout << "Constructor Invoked"  << std::endl;
29  }
30
31  Polygon :: Polygon(const Polygon & obj){
32    length_ = obj.length_;
33    std::cout << "Copy Constructor Invoked"  << std::endl;
34  }
35
36  Polygon & Polygon :: operator=(const Polygon & obj){
37    length_ = obj.length_;
38    std::cout << "Assignment operator invoked"  << std::endl;
39    return *this; // dereferenced pointer
40  }
41
42  Polygon Polygon :: operator+(const Polygon & obj){
43    Polygon A;
44    A.length_ = this->length_ + obj.length_;
45    A.numberSides_ = this->numberSides_ + obj.numberSides_;
46    return A;
```

master*+ ⊗ 0 ⚠ 0 Live Share Ln 80, Col 22 Spaces: 2 UTF-8 LF C++ Mac

Constructor

Shape.h | libraries.h | main.cpp | Makefile | Polygon.h | Prism.h | Polygon.cpp

main.cpp > main()

```
58
59  //************************************************
60  //        -------- Inheritance --------
61  //************************************************
62  cout << "-------- Inheritance --------" << endl;
63  {
64  Shape2D shap2d;
65  Polygon square(2,4);
66
67  Shape2D* shap2dPtr = &square;
68
69  cout << shap2d.shapeName() << endl;
70  cout << shap2dPtr->shapeName() << endl;
71  }
72
73  //************************************************
74  //        -------- Constructor --------
75  //************************************************
76  cout << "-------- Constructor --------" << endl;
77
78  {
79    Polygon poly1;
80    cout << poly1.getNumberSides() << endl;
81  }
82
83  {
84    Polygon poly2(2,5);
85    cout << poly2.getNumberSides() << endl;
86  }
87
88
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
-------- Constructor --------
Default Constructor Invoked
3
Polygon destroyed
Constructor Invoked
5
Polygon destroyed
(base) Ivaldo:Week1 admin$
```

master*+ 0 0 Live Share Ln 85, Col 18 Spaces: 2 UTF-8 LF C++ Mac

Copy Constructor

- Is a special constructor that exists to make a copy of an existing object.
- If we do not provide a custom copy constructor, the C++ compiler will provide one automatically.
- Has exactly one argument (constant reference to the same type as the class itself).
- **Polygon :: Polygon (const Polygon & object)**

Copy constructor is invoked when:

- Passing an object as a parameter;
- Returning an object from a function;
- Initializing a new object.

Copy assignment Operator

The assignment operator for a class is what allows you to use = to assign one instance to another.

Polygon & operator=(const Polygon & obj);

In general, any time you need to write your own custom copy constructor, you also need to write a custom assignment operator.

Copy Assignment Operator

Shape.h | libraries.h | main.cpp | Makefile | Polygon.h | Prism.h | Polygon.cpp

Polygon.cpp > operator=(const Polygon &)

```
    numberSides_ = 3;
    std::cout << "Default Constructor Invoked"  << std::endl;
}


Polygon :: Polygon(double length, unsigned int numberSides){
    length_ = (length>0)? length: 1;
    numberSides_ = (numberSides>2)? numberSides : 3;
    std::cout << "Constructor Invoked"  << std::endl;
}

Polygon :: Polygon(const Polygon & obj){
    length_ = obj.length_;
    std::cout << "Copy Constructor Invoked"  << std::endl;
}

Polygon & Polygon :: operator=(const Polygon & obj){
    length_ = obj.length_;
    numberSides_ = obj.numberSides_;
    std::cout << "Assignment operator invoked"  << std::endl;
    return *this; // dereferenced pointer
}

Polygon Polygon :: operator+(const Polygon & obj){
    Polygon A;
    A.length_ = this->length_ + obj.length_;
    A.numberSides_ = this->numberSides_ + obj.numberSides_;
    return A;
}
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
-------- Copy Constructor --------
Default Constructor Invoked
Constructor Invoked
Copy Constructor Invoked
Polygon destroyed
Constructor Invoked
Assignment operator invoked
Polygon destroyed
Polygon destroyed
Polygon destroyed
(base) Ivaldo:Week1 admin$
```

master*+ | 0 ⚠ 0 | Live Share | Ln 38, Col 35 | Spaces: 2 | UTF-8 | LF | C++ | Mac

Copy Constructor & Copy assignment Operator

Shape.h | libraries.h | main.cpp | Makefile | Polygon.h | Prism.h | Polygon.cpp

main.cpp > main()

```
{
  Polygon poly1;
  cout << poly1.getNumberSides() << endl;
}

{
  Polygon poly2(2,5);
  cout << poly2.getNumberSides() << endl;
}


//************************************************************
//    --- Copy Constructor & Copy assignment Operator ---
//************************************************************
cout << "--- Copy Constructor & Copy assignment Operator ---" << endl;

{
  Polygon poly1;
  Polygon poly2(2,5);
  test1(poly2);
  Polygon poly3 = test2();
  poly1 = poly3;
}
```

```
void test1(Polygon b){}

Polygon test2(){
  Polygon square(2,4);
  return square;
}
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
--- Copy Constructor & Copy assignment Operator ---
Default Constructor Invoked
Constructor Invoked
Copy Constructor Invoked
Polygon destroyed
Constructor Invoked
Assignment operator invoked
Polygon destroyed
Polygon destroyed
Polygon destroyed
(base) Ivaldo:Week1 admin$
```

2022-06-21

master*+ | 0 ⚠ 0 | Live Share | Ln 87, Col 63 | Spaces: 2 | UTF-8 | LF | C++ | Mac

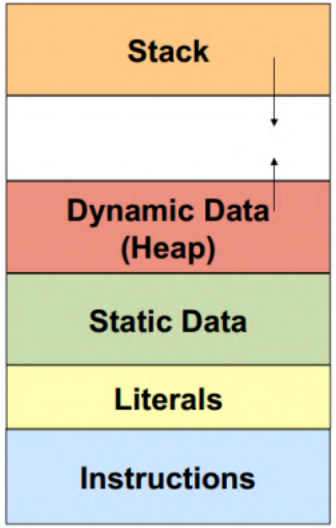
Class Destructor

If you don't have any destructors defined, one will be provided automatically.

The custom constructor must be the name of the class itself, preceded by a tilde(~) and have no arguments and no return types.

Polygon ::~Polygon();

If your object is in the **stack** memory,
your destructor is going to be called as soon as the function returns.
If it's on the **heap**, the destructor is only called when that **delete** keyword is used.



Class Destructor

Shape.h | libraries.h | main.cpp | Makefile | Polygon.h | Prism.h | Polygon.cpp

Polygon.cpp > operator=(const Polygon &)

```
41  }
42
43  Polygon Polygon :: operator+(const Polygon & obj){
44    Polygon A;
45    A.length_ = this->length_ + obj.length_;
46    A.numberSides_ = this->numberSides_ + obj.numberSides_;
47    return A;
48  }
49
50  Polygon::~Polygon() {
51      std::cout << "Polygon destroyed" << std::endl;
52  }
53
54  double Polygon:: area(){
55    double perimeter = numberSides_*length_;
56    double apothem = (length_)/(2*tan(PI/numberSides_));
57    return perimeter*apothem/2;
58  }
59
60  string Polygon::shapeName() {
61
62    string arrayName[6] = {"triangle" , "square", "pentagon",
63    "hexagon", "heptagon", "octagon"};
64
65    string name = (numberSides_<9)? arrayName[numberSides_-3]: to_string(numberSides_)+"_polygon";
66
67    return name;
68  }
69
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
--- Copy Constructor & Copy assignment Operator ---
Default Constructor Invoked
Constructor Invoked
Copy Constructor Invoked
Polygon destroyed
Constructor Invoked
Assignment operator invoked
Polygon destroyed
Polygon destroyed
Polygon destroyed
(base) Ivaldo:Week1 admin$
```

master*+ | 0 ⚠ 0 | Live Share | Ln 38, Col 35 | Spaces: 2 | UTF-8 | LF | C++ | Mac

Shape.h | libraries.h | main.cpp | Makefile | Polygon.h | Polygon.cpp

main.cpp > main()

```
75    Polygon poly3 = test2();
76    poly1 = poly3;
77  }
78
79  //************************************************
80  //       -------- Class Destructor --------
81  //************************************************
82  cout << "-------- Class Destructor --------" << endl;
83
84  {
85    Polygon poly(10,11);
86    cout << poly.area() << endl;
87  }
88
89  {
90    Polygon* p1 = new Polygon();
91    Polygon* p2 = new Polygon(3,6);
92    cout << p1->shapeName() << endl;
93    cout << p2->shapeName() << endl;
94
95    delete p1;
96  }
97
98
99
100
101
102  return 0;
103 }
```

Polygon* p1 = new Polygon;

The code above allocates two chunks of memory:
1 – Memory on the stack to store the pointer;
2 – Memory on the heap to store the object.

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
-------- Class Destructor --------
Constructor Invoked
936.564
Polygon destroyed
Default Constructor Invoked
Constructor Invoked
triangle
hexagon
Polygon destroyed
(base) Ivaldo:Week1 admin$
```


master*+ | 0 ⚠ 0 | Live Share | Ln 90, Col 31 | Spaces: 2 | UTF-8 | LF | C++ | Mac

Variable storage

Now that we understand how a class exists in C++ and also how variables are created and stored. Let's talk about three different ways to store a variable.

	Directly in memory	accessed via a pointer	accessed by a reference
Size	exactly the size it takes up to store	The exact length of that memory address (Ex 64 bit)	It has no size
type	Polygon b;	Polygon* p;	Polygon& r = Polygon b; we have to alias that reference variable at the moment we create it.
Pass by	Value(default)	Pointer(modified with *)	Reference(modified by & , acts as an alias)

Variable Storage

Shape.h | libraries.h | main.cpp | Makefile | Polygon.h | Polygon.cpp

main.cpp > main()

```
100   //***********************************************
101   //       -------- Variable storage --------
102   //***********************************************
103   cout << "-------- Variable storage --------" << endl;
104
105   {
106     Polygon poly(5,5);
107     test1(poly);
108     test3(poly);
109     test4(&poly);
110   }
111
112
113
114
115   return 0;
116 }
117
118
119
```

```
void test1(Polygon b){}

Polygon test2(){
  Polygon square(2,4);
  return square;
}

void test3(Polygon& b){}

void test4(Polygon* b){}
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
-------- Variable storage --------
Constructor Invoked
Copy Constructor Invoked
Polygon destroyed
Polygon destroyed
(base) Ivaldo:Week1 admin$
```

2-06-21

master*+ | 0 ⚠ 0 | Live Share | Ln 109, Col 12 | Spaces: 2 | UTF-8 | LF | C++ | Mac

Now it's your turn: what will be the output?

```
//*************************************************
//       -------- Now it's your turn --------
//*************************************************
cout << "-------- Now it's your turn --------" << endl;

  {
  Polygon poly(5,5);
  test5(poly, 10);
  cout << poly.getlength() << endl;
  }
```

```
void test5(Polygon& b, double length){
  if (length > 0.0){
    b.setlength(length);
  }
}
```

```
-------- Now it's your turn --------
Constructor Invoked
10
Polygon destroyed
```

References

- Geeksforgeeks (2022, Jan 3) **Difference between Virtual function and Pure virtual function in C++.** https://www.geeksforgeeks.org/difference-between-virtual-function-and-pure-virtual-function-in-c/

Appendix

- Libraries.h
- Shape.h
- Polygon.h
- Polygon.cpp
- Main.cpp