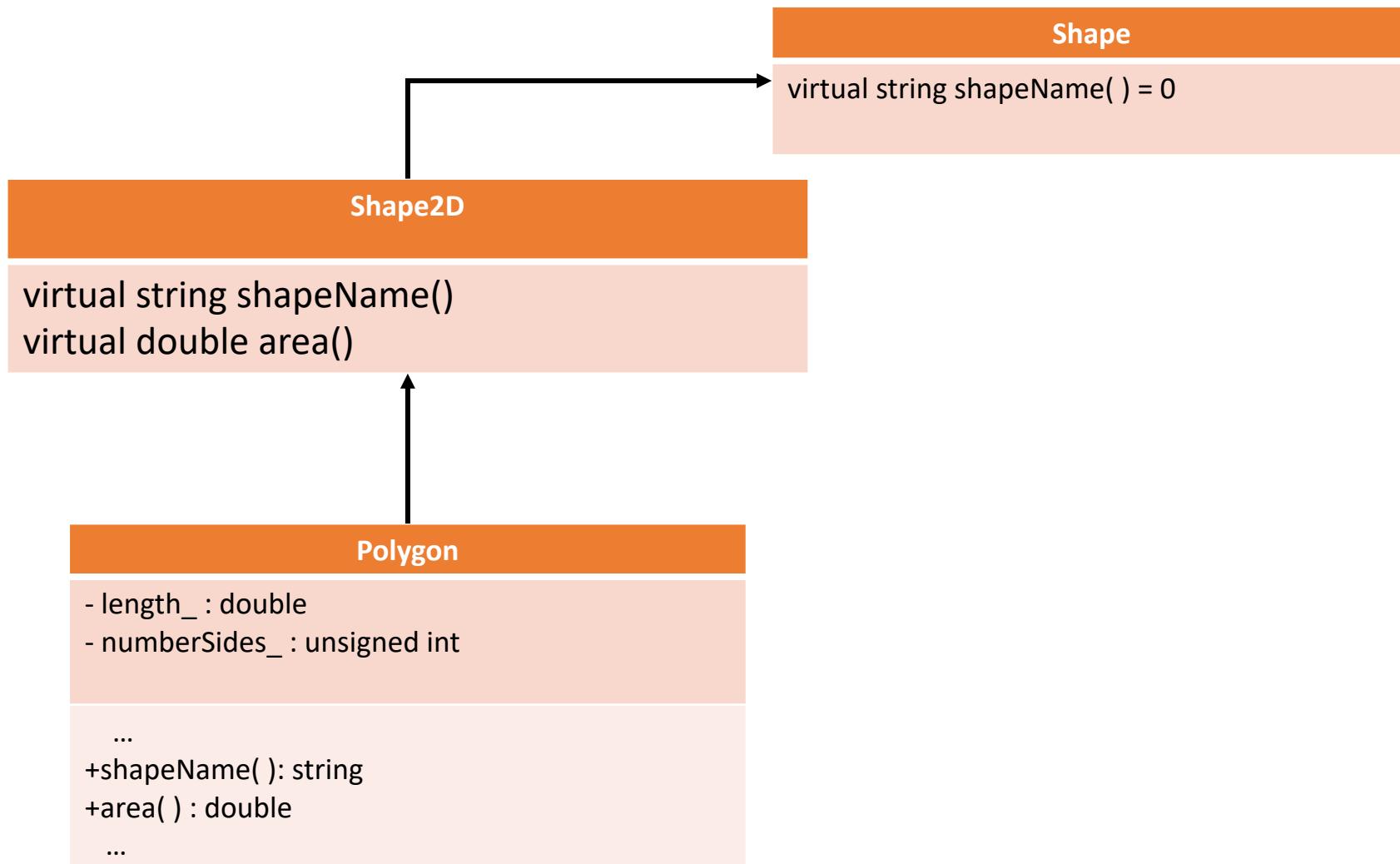


Review

Object-Oriented Programming Concepts & Lifecycle of a C++ object.



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.

```
class Myclass
{
public:
    virtual ReturnType Function(Argumet) = 0;
};
```

Keyword

Null Function Body

The diagram shows a code snippet for an abstract class named 'Myclass'. It includes a public section with a single pure virtual function 'Function' that returns 'ReturnType' and takes an argument. The function body is explicitly defined as '0', which is highlighted with a red box and labeled 'Null Function Body'. A blue arrow points from the word 'Keyword' to the '=' sign in the function declaration, indicating that the assignment operator is being used to denote a pure virtual function. The entire code block is enclosed in a light gray rounded rectangle.

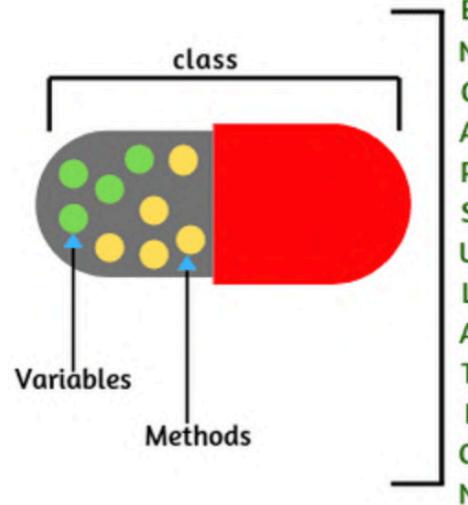
```
11  
12  
13 #pragma once  
14 // #include is a way of including a standard or user-defined file in the program  
15 #include "libraries.h"  
16  
17 // Abstract class  
18 class Shape{  
public:  
19  
    // Pure Virtual Function  
20    virtual string shapeName() = 0;  
21};  
22  
23  
24  
25  
26 class Shape2D : public Shape{  
27  
public:  
28    //Virtual Function  
29    virtual string shapeName(){ return "Shape2D";}  
30    //Pure Virtual Function  
31    virtual double area(){ return 0.0;}  
32};  
33  
34};
```

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.

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

```
C Shape.h      C libraries.h    C main.cpp    C Polygon.h X  C Prism.h     C Prism.cpp   C Polygon.cpp
C Polygon.h > Polygon
y
10 // All header (.h) files start with "#pragma once":
11 #pragma once
12
13 #include "Shape.h"
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     protected: // Protected members:
24         string solidName;
25
26     public: // Public members:
27     /**
28     * Creates a triangle with one side measuring 1.
29     */
30     Polygon(); // Custom default constructor
31
32     /**
33     * Create a polygon using the following parameters:
34     * @param numberSides.
35     * @param length.
36     */
37     Polygon(double length, unsigned int numberSides); // Custom Constructor
38
39     /**
40     * Copy constructor: creates a new Polygon from another.
41     * @param obj polygon to be copied.
42     */
43     Polygon(const Polygon & obj); // Custom Copy constructor
44
45     /**
46     * Assignment operator for setting two Polygon equal to one another.
47     * @param obj Polygon to copy into the current Polygon.
48     * @return The current image for assignment chaining.
49     */
50 }
```

Encapsulation

The screenshot shows a code editor interface with the following details:

- Top Bar:** Shows tabs for Shape.h, libraries.h, main.cpp (active), Makefile, Polygon.h, Prism.h, and Polygon.cpp.
- Sidebar:** Contains icons for file operations, search, and other development tools.
- Code Area:** Displays the main.cpp file content. The code includes comments for encapsulation and attempts to print the private member length_.
- Terminal Area:** Shows the output of the compilation process, indicating an error due to attempting to access a private member from outside its class.
- Status Bar:** Shows the current file path (master*+), line (Ln 53, Col 32), and other system information.

```
43
44 int main() {
45
46     //***** Encapsulation *****
47     // ----- Encapsulation -----
48     //***** Encapsulation *****
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
```

```
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$
```

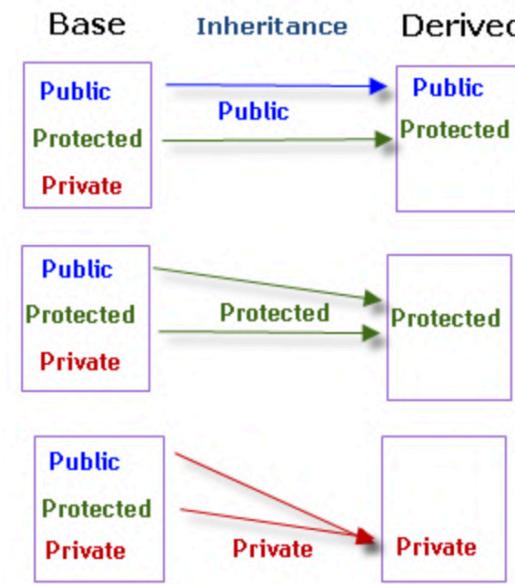
Ln 53, Col 32 Spaces: 2 UTF-8 LF C++ Mac Live Share

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.

The screenshot displays a dual-monitor setup with two instances of a code editor, likely Visual Studio Code, showing the same C++ codebase. The code is organized into two main classes: `Polygon` and `Prism`.

Polygon Class (Left Monitor):

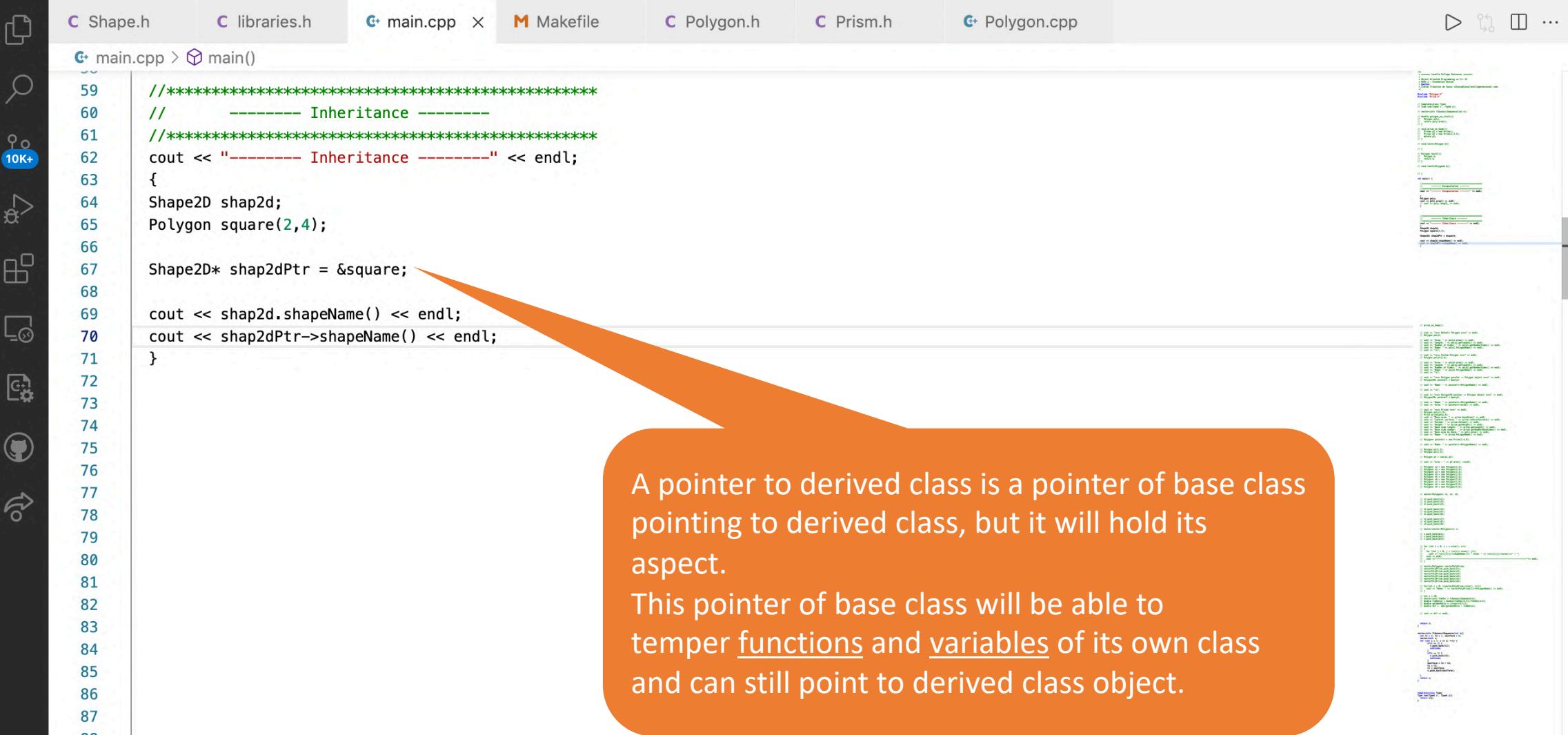
```
13 #include "Shape.h"
14
15 class Polygon : public Shape2D {
16
17     private: // Private members:
18
19     // Data Members (underscore indicates a private member variable)
20     double length_;
21     unsigned int numberSides_;
22
23 public: // Public members:
24
25     /**
26      * Creates a triangle with one side measuring 1.
27      */
28     Polygon(); // Custom default constructor
29
30     /**
31      * Create a polygon using the following parameters:
32      * @param numberSides.
33      * @param length.
34      */
35     Polygon(double length, unsigned int numberSides); // Custom Constructor
36
37     /**
38      * Copy constructor: creates a new Polygon from another.
39      * @param obj polygon to be copied.
40      */
41     Polygon(const Polygon & obj); // Custom Copy constructor
42
43     /**
44      * Assignment operator for setting two Polygon equal to one another.
45      * @param obj Polygon to copy into the current Polygon.
46      * @return The current image for assignment chaining.
47      */
48     Polygon & operator=(const Polygon & obj); // Custom assignment operator;
49
50     Polygon operator+(const Polygon & obj); // Operators + Overloading
51
52     /**
53      * Destructor: frees all memory associated with a given Polygon object.
54      */
55 }
```

Prism Class (Right Monitor):

```
13 # include "Polygon.h"
14 # include "Shape.h"
15
16
17 class Prism : public Shape3D{
18
19     private: // Private members:
20
21     double height_;
22     unsigned int numberBaseSides_;
23     double length_;
24     Polygon base_;
25
26 public:
27
28     /**
29      * Create a triangular prism 1 high.
30      */
31     Prism();
32
33     /**
34      * Create a Prism using the following parameters:
35      * @param numberBaseSides.
36      * @param length.
37      * @param height
38      */
39     Prism(double length, unsigned int numberBaseSides, double height);
40
41     /**
42      * Create a polygon using the following parameters:
43      * @param Polygon.
44      * @param height
45      */
46     Prism(Polygon& poly, double height);
47
48     /**
49      * Copy constructor: creates a new Prism from another.
50      * @param obj polygon to be copied.
51      */
52     Prism(const Prism & obj); // Custom Copy constructor
53 }
```

The code includes detailed comments for each constructor, operator, and method, specifying parameters and return values. The `Polygon` class has a custom default constructor, a constructor taking `length` and `numberSides`, a copy constructor, an assignment operator, an overloaded addition operator, and a destructor. The `Prism` class has a constructor, a constructor taking `length`, `numberBaseSides`, and `height`, a constructor taking a `Polygon` and `height`, and a copy constructor.

Inheritance



The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure with files like Shape.h, libraries.h, main.cpp, Makefile, Polygon.h, Prism.h, and Polygon.cpp.
- Code Editor:** The main editor window displays the `main.cpp` file. It contains code demonstrating inheritance://*****
// ----- Inheritance -----
//*****
cout << "----- Inheritance -----" << endl;
{
Shape2D shap2d;
Polygon square(2,4);

Shape2D* shap2dPtr = □

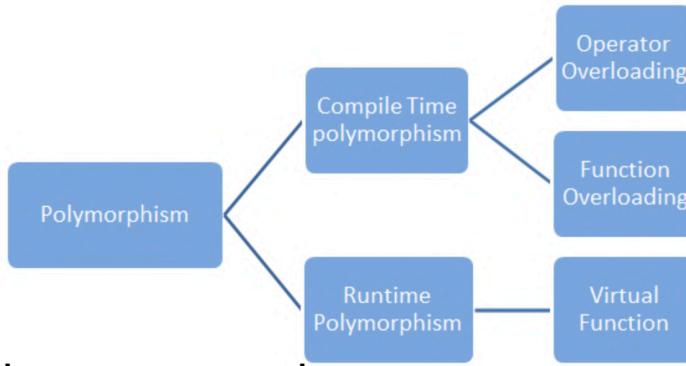
cout << shap2d.shapeName() << endl;
cout << shap2dPtr->shapeName() << endl;
}

72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
- Output Panel:** Shows the terminal output:

```
----- Inheritance -----  
Constructor Invoked  
Shape2D  
square  
Polygon destroyed  
(base) Ivaldo:Week1 admin$
```
- Bottom Status Bar:** Shows the date (2022-06-21), file name (master*+), and other status information.
- Callout Bubble:** An orange callout bubble points from the line `Shape2D* shap2dPtr = □` to the explanatory text below.
- Text Block:** Inside the callout bubble, the following text is displayed:

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.

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

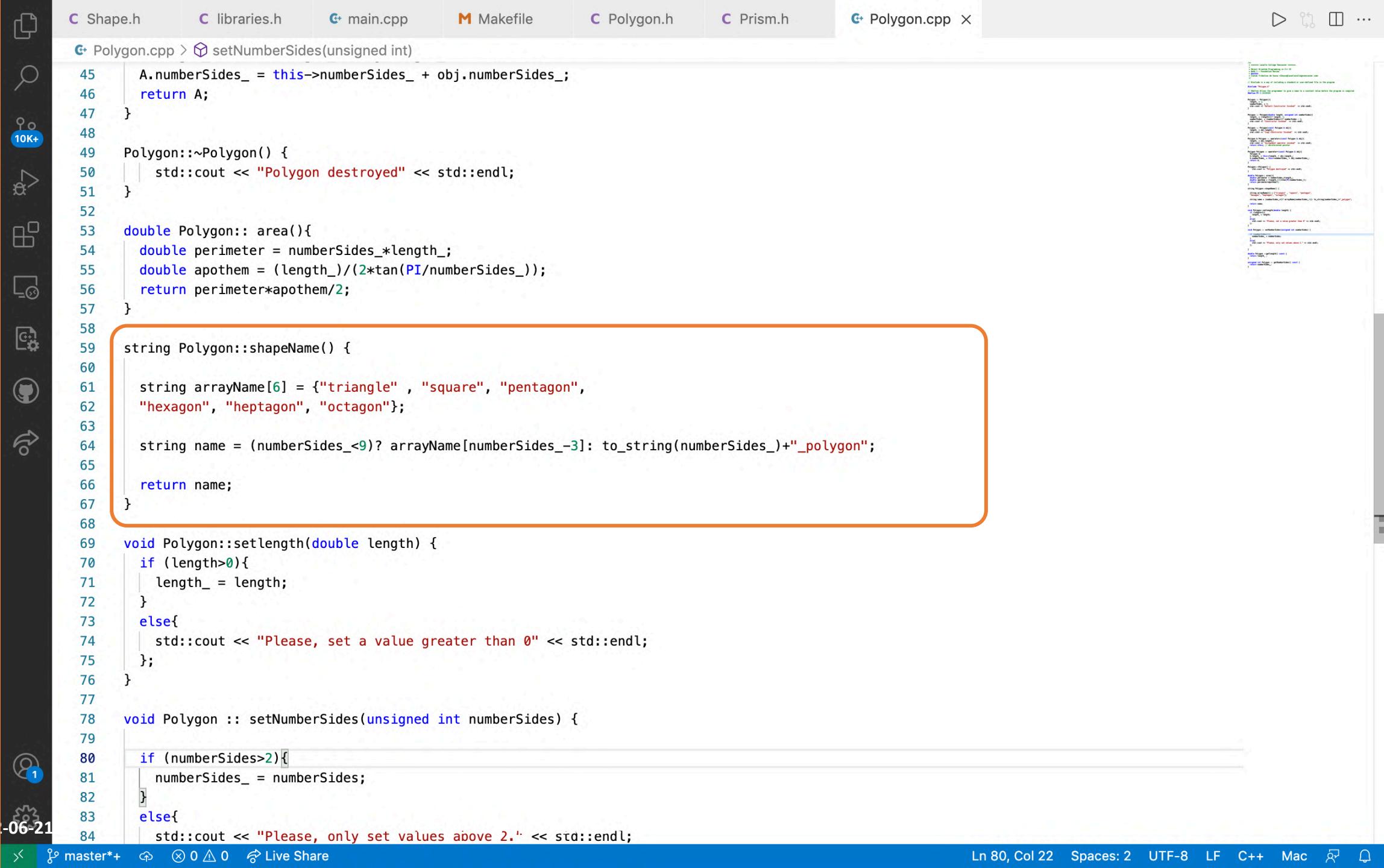
The screenshot shows a code editor with the following details:

- File Tabs:** Shape.h, libraries.h, main.cpp, Makefile, Polygon.h, Prism.h, Polygon.cpp.
- Sidebar Icons:** Copy, Paste, Find, Refresh, 10K+, Open, Save, Settings, GitHub, Sync.
- Code Editor Content:**

```
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 }
```
- Code Editor Bottom:** PROBLEMS, OUTPUT, TERMINAL, DEBUG CONSOLE, bash, +, ×.
- Terminal Output:**

```
----- Inheritance -----
Constructor Invoked
Shape2D
square
Polygon destroyed
(base) Ivaldo:Week1 admin$
```
- Bottom Status Bar:** master*+, 0 △ 0, Live Share, Ln 45, Col 3, Spaces: 2, UTF-8, LF, C++, Mac, ⌘, ⌘, ⌘.

Overriding



The screenshot shows a code editor interface with several tabs at the top: Shape.h, libraries.h, main.cpp, Makefile, Polygon.h, Prism.h, and Polygon.cpp (the active tab). On the left, there's a vertical toolbar with icons for copy, paste, search, and other file operations, along with a '10K+' badge. The main pane displays the following C++ code:

```
Shape.h          libraries.h        main.cpp      Makefile      Polygon.h      Prism.h      Polygon.cpp ×

C+ 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;
85     }
86 }
```

A specific section of the `shapeName()` method is highlighted with an orange rounded rectangle, starting from line 59 and ending at line 68. This highlights the overridden behavior where the polygon's name is determined based on its number of sides.

-06-21



Live Share

Overloading

The screenshot shows a code editor interface with several tabs at the top: Shape.h, libraries.h, main.cpp, Makefile, Polygon.h (active), Prism.h, and Polygon.cpp. The left sidebar contains icons for file operations like copy, paste, search, and refresh, along with a '10K+' badge. The main pane displays the following C++ code for the Polygon class:

```
#include "Shape.h"

class Polygon : public Shape2D {
private: // Private members:
    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

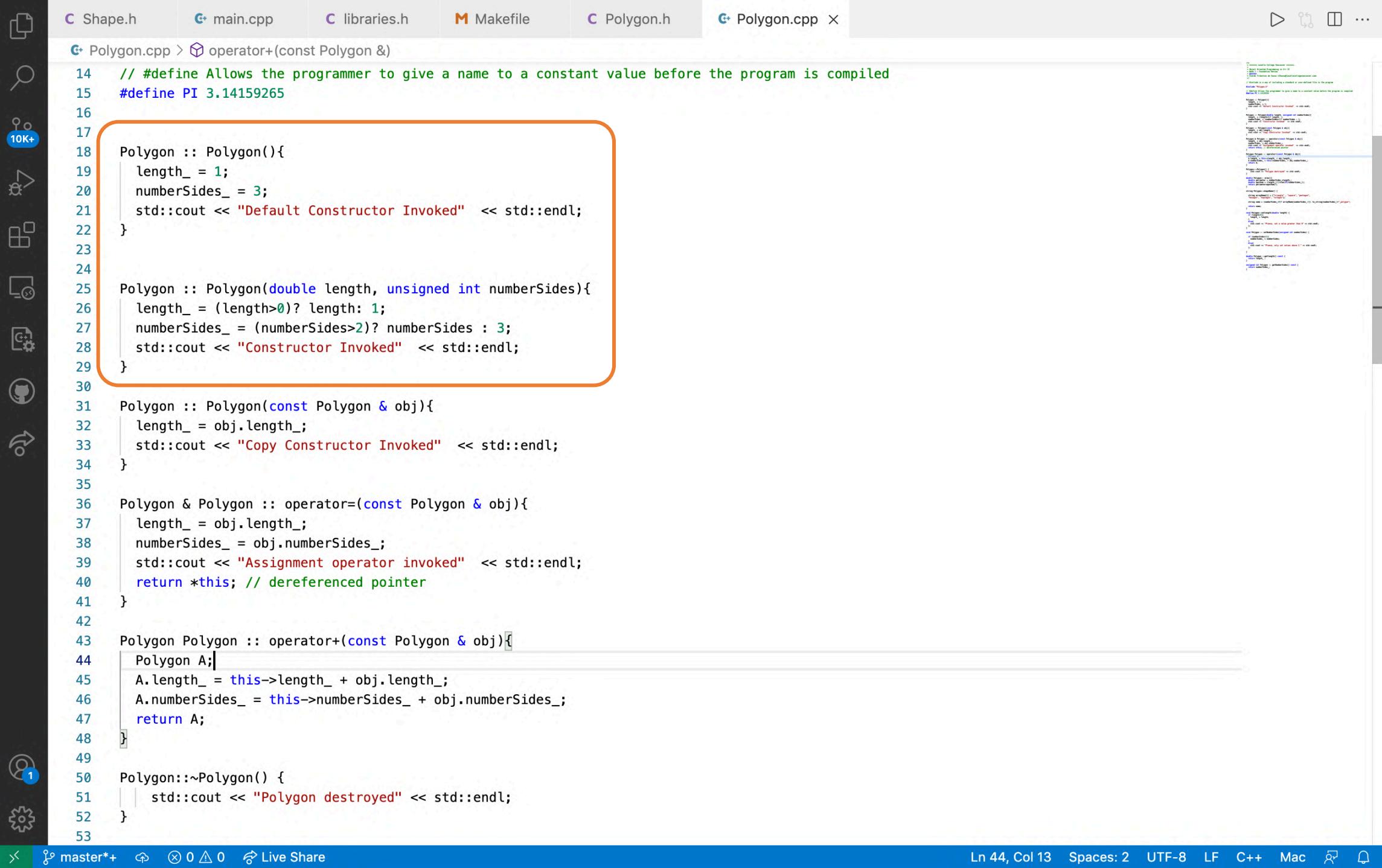
    /**
     * Assignment operator for setting two Polygon equal to one another.
     * @param obj Polygon to copy into the current Polygon.
     * @return The current image for assignment chaining.
     */
    Polygon & operator=(const Polygon & obj); // Custom assignment operator;

    Polygon operator+(const Polygon & obj); // Operators + Overloading

    /**
     * Destructor: frees all memory associated with a given Polygon object.
     */
}
```

The code includes detailed comments for each constructor, the assignment operator, and the addition operator, explaining their functionality and parameters.

Overloading



The screenshot shows a code editor interface with several tabs at the top: Shape.h, main.cpp, libraries.h, Makefile, Polygon.h, and Polygon.cpp (the active tab). On the left, there's a vertical toolbar with icons for copy, search, and other file operations, one of which has a '10K+' badge. The main area displays the following C++ code:

```
Shape.h          main.cpp          libraries.h          Makefile          Polygon.h          Polygon.cpp X

C+ Polygon.cpp > operator+(const Polygon &)

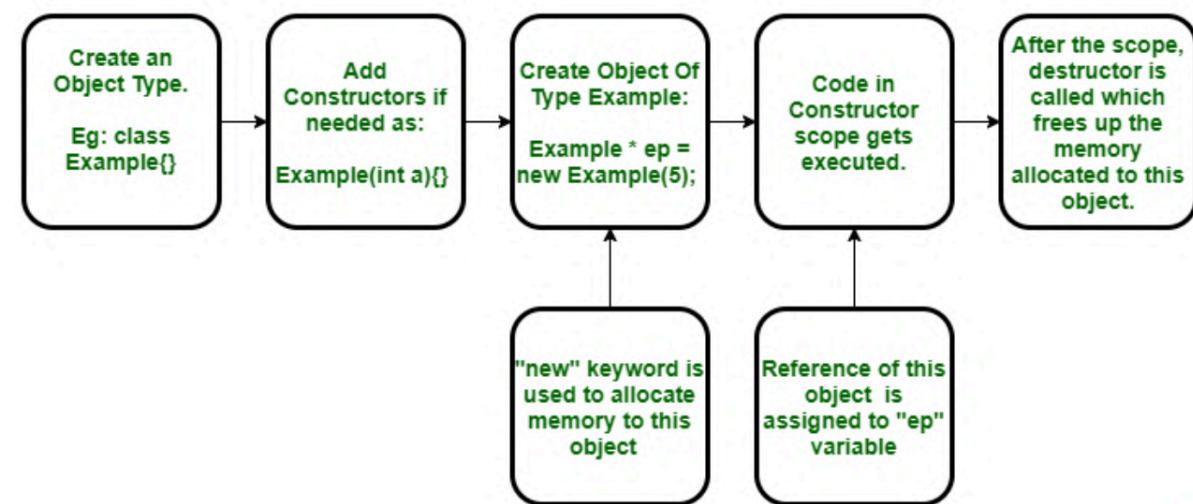
14 // #define Allows the programmer to give a name to a constant value before the program is compiled
15 #define PI 3.14159265
16
17 Polygon :: Polygon(){
18     length_ = 1;
19     numberSides_ = 3;
20     std::cout << "Default Constructor Invoked" << std::endl;
21 }
22
23
24
25 Polygon :: Polygon(double 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     numberSides_ = obj.numberSides_;
39     std::cout << "Assignment operator invoked" << std::endl;
40     return *this; // dereferenced pointer
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
```

A red box highlights the first constructor definition (Line 17). The code uses standard C++ features like namespaces, classes, and templates. It demonstrates the use of #define for constants, constructors for initializing objects, and assignment operators for updating object states.

Lifecycle of a C++ object.

Key Concepts

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



DE

Class Constructor

Custom Default Constructor	Custom Constructor
<p>Specifies the states of the object when the object is constructed.</p> <p>It is defined by creating:</p> <ul style="list-style-type: none">• A member function with the same name of the class;• The function take zero parameters;• The function does not have a return type. <p>Polygon :: Polygon();</p>	<p>Arguments will be provided.</p> <p>Polygon :: Polygon(double length, int numberSides);</p>

Polygon.h

```
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.
54 */
```

Polygon.cpp

```
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 Polygon :: Polygon(){
18     length_ = 1;
19     numberSides_ = 3;
20     std::cout << "Default Constructor Invoked" << std::endl;
21 }
22
23
24 Polygon :: Polygon(double length, unsigned int numberSides){
25     length_ = (length>0)? length: 1;
26     numberSides_ = (numberSides>2)? numberSides : 3;
27     std::cout << "Constructor Invoked" << std::endl;
28 }
29
30 Polygon :: Polygon(const Polygon & obj){
31     length_ = obj.length_;
32     std::cout << "Copy Constructor Invoked" << std::endl;
33 }
34
35 Polygon & Polygon :: operator=(const Polygon & obj){
36     length_ = obj.length_;
37     std::cout << "Assignment operator invoked" << std::endl;
38     return *this; // dereferenced pointer
39 }
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;
```

Constructor

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files like Shape.h, libraries.h, main.cpp, Makefile, Polygon.h, Prism.h, and Polygon.cpp.
- Sidebar:** Includes icons for file operations, search, and a "10K+" badge.
- Code Editor:** Displays the main.cpp file with code demonstrating inheritance and constructors. The code includes sections for Inheritance and Constructor, and defines two Polygon objects (poly1 and poly2) with different side counts (3 and 5).
- Terminal:** Shows the output of the program execution, displaying the printed values and destructor messages.
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, TERMINAL, and DEBUG CONSOLE, along with a terminal icon and various command-line options.

```
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 }
```

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

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

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files like Shape.h, libraries.h, main.cpp, Makefile, Polygon.h, Prism.h, and Polygon.cpp.
- Sidebar:** Includes icons for file operations, search, and a gear (settings).
- Code Editor:** Displays the `Polygon.cpp` file with the following content:

```
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;
}
```

A red box highlights the assignment operator implementation from line 36 to 41.
- Terminal:** Shows the output of the program:

```
----- 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$
```
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, TERMINAL, DEBUG CONSOLE, and a terminal icon with `bash`.

Copy Constructor & Copy assignment Operator

The screenshot shows a code editor interface with several tabs at the top: Shape.h, libraries.h, main.cpp (selected), Makefile, Polygon.h, Prism.h, and Polygon.cpp. The main.cpp tab displays the following code:

```
main.cpp > main()
73 {
74     Polygon poly1;
75     cout << poly1.getNumberSides() << endl;
76 }
77 {
78     Polygon poly2(2,5);
79     cout << poly2.getNumberSides() << endl;
80 }
81
82
83
84 //***** --- Copy Constructor & Copy assignment Operator --- *****
85 // --- Copy Constructor & Copy assignment Operator ---
86 //***** --- Copy Constructor & Copy assignment Operator --- *****
87 cout << "--- Copy Constructor & Copy assignment Operator ---" << endl;
88
89 {
90     Polygon poly1;
91     Polygon poly2(2,5);
92     test1(poly2);
93     Polygon poly3 = test2();
94     poly1 = poly3;
95 }
96
97
98
99
100
```

A red box highlights the following code block:

```
void test1(Polygon b){}

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

The terminal output below shows the execution results:

```
--- 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$
```

The bottom status bar indicates the date as 2022-06-21 and the file as master*+.

Class Destructor

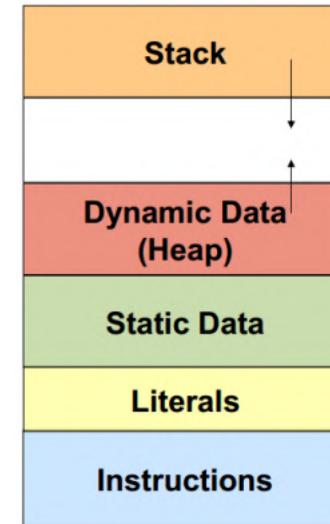
If you don't have any destructors defined, one will be provided automatically.

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

The screenshot shows a code editor interface with the following details:

- File Tabs:** Shape.h, libraries.h, main.cpp, Makefile, Polygon.h, Prism.h, Polygon.cpp (active tab).
- Sidebar Icons:** Document (1), Search, 10K+, Copy, Paste, Find, Settings, GitHub, Refresh.
- Code Area:** The `Polygon.cpp` file contains the following code:

```
operator=(const Polygon &)
}
}

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;
}

double Polygon:: area(){
    double perimeter = numberSides_*length_;
    double apothem = (length_)/(2*tan(PI/numberSides_));
    return perimeter*apothem/2;
}

string Polygon::shapeName() {
    string arrayName[6] = {"triangle", "square", "pentagon",
    "hexagon", "heptagon", "octagon"};
    string name = (numberSides_<9)? arrayName[numberSides_-3]: to_string(numberSides_)+"_polygon";
    return name;
}
```

A block of code from line 50 to line 69 is highlighted with an orange border.
- Terminal Output:** Shows the execution of the program, outputting the names of the shapes created and destroyed.

```
--- 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$
```
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, TERMINAL, DEBUG CONSOLE, and a terminal icon with bash.
- Status Bar:** Shows the current file is master*, line 38, column 35, with spaces: 2, UTF-8 encoding, and Mac OS X system information.

Class Destructor

The screenshot shows a C++ development environment with several tabs open: Shape.h, libraries.h, main.cpp, Makefile, Polygon.h, and Polygon.cpp. The main.cpp tab is active, displaying the following code:

```
main() {
    ...
    Polygon poly3 = test2();
    poly1 = poly3;
}

//***** ----- Class Destructor -----
//***** ----- Class Destructor -----
cout << "----- Class Destructor -----" << endl;

{
    Polygon poly(10,11);
    cout << poly.area() << endl;
}

{
    Polygon* p1 = new Polygon();
    Polygon* p2 = new Polygon(3,6);
    cout << p1->shapeName() << endl;
    cout << p2->shapeName() << endl;

    delete p1;
}

return 0;
}
```

A callout bubble points to the line `Polygon* p1 = new Polygon();` with the text:

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.

The terminal output window shows the following logs:

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

The status bar at the bottom indicates the file is master*, line 90, column 31, with spaces: 2, UTF-8 encoding, and Mac OS X interface.

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

The screenshot shows a C++ development environment with several tabs at the top: Shape.h, libraries.h, main.cpp, Makefile, Polygon.h, and Polygon.cpp. The main.cpp tab is active, displaying the following code:

```
100 //***** Variable storage *****
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 }
```

A red box highlights the following four function definitions:

```
void test1(Polygon b){}
Polygon test2(){
    Polygon square(2,4);
    return square;
}

void test3(Polygon& b){}

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

The bottom left corner shows a status bar with the date '06-21' and a terminal prompt '(base) Ivaldo:Week1 admin\$'. The bottom right corner shows the status bar with 'Ln 109, Col 12' and other terminal settings.

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