

Function Pointers, Functors, lambdas, std::function.

VGP130 – OOP II

Instructor: Ivaldo Tributino

Area Matrix

| | | Area | | | |
|----------------|-------------------------|---------------------|-------------------------|-------------------------|-------------------------|
| Length\Polygon | Triangle | Square | Pentagon | Hexagon | Heptagon |
| 2 cm | 1.73205 cm ² | 4 cm ² | 6.88191 cm ² | 10.3923 cm ² | 14.5356 cm ² |
| 4 cm | 6.9282 cm ² | 16 cm ² | 27.5276 cm ² | 41.5692 cm ² | 58.1426 cm ² |
| 6 cm | 15.5885 cm ² | 36 cm ² | 61.9372 cm ² | 93.5307 cm ² | 130.821 cm ² |
| 8 cm | 27.7128 cm ² | 64 cm ² | 110.111 cm ² | 166.277 cm ² | 232.57 cm ² |
| 10 cm | 43.3013 cm ² | 100 cm ² | 172.048 cm ² | 259.808 cm ² | 363.391 cm ² |

```
Polygon :: Polygon(double length, int numberSides){
```

```
    . . .
    cout << "Constructor Invoked" << endl;
```

```
}
```

POLYGON CLASS

```
class Polygon {  
  
    private: // Private members:  
  
        // Data Members (underscore indicates a private member variable)  
        double length_;  
        unsigned int numberSides_;
```

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


function call Operator () Overloading

- Objects can be treated as if they were a function.
- We can create objects that overloads the *operator()*.

```
returnType operator symbol (arguments) {  
    ... ... ...  
}
```

```
// function to overload the operator  
double Polygon :: operator()(float length){  
    double perimeter = numberSides_*length;  
    double apothem = (length)/(2*tan(PI/numberSides_));  
    return perimeter*apothem/2;  
}
```

AS A FUNCTION

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

```
main.cpp:46: Polygon square(4);
main.cpp:47: Polygon pentagon(5);
main.cpp:48: Polygon hexagon(6);
main.cpp:49: Polygon heptagon(7);
main.cpp:50:
main.cpp:51: vector<Polygon> polys; // for the fastest implementation use pointers. vector<Polygon*> polys
main.cpp:52: polys.reserve(5);
main.cpp:53:
main.cpp:54: polys.push_back(triangle);
main.cpp:55: polys.push_back(square);
main.cpp:56: polys.push_back(pentagon);
main.cpp:57: polys.push_back(hexagon);
main.cpp:58: polys.push_back(heptagon);
main.cpp:59:
main.cpp:60: for(Polygon const & poly : polys){
main.cpp:61:     for(int i =1; i<6; i++){
main.cpp:62:         cout << poly(2*i) << " | ";
main.cpp:63:     }
main.cpp:64:     cout << endl;
main.cpp:65: }
main.cpp:66:
main.cpp:67: cout << "- object behaves like a function" << endl;
main.cpp:68:
main.cpp:69: cout << triangle(20/pow(3,0.25)) << endl;
main.cpp:70: cout << square(10) << endl;
main.cpp:71: cout << hexagon(20/pow(3,0.25)) << endl;
main.cpp:72:
```

The code uses objects of the Polygon class as if they were functions, printing their values to the console. A callout bubble points from the line `cout << "- object behaves like a function" << endl;` to a blue box containing the text: "The object was called as if it were an ordinary function." Below the code editor, the terminal output shows the results of the program execution.

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Copy Constructor Invoked
1.73205 | 6.9282 | 15.5885 | 27.7128 | 43.3013 |
4 | 16 | 36 | 64 | 100 |
6.88191 | 27.5276 | 61.9372 | 110.111 | 172.048 |
10.3923 | 41.5692 | 93.5307 | 166.277 | 259.808 |
14.5356 | 58.1426 | 130.821 | 232.57 | 363.391 |
- object behaves like a function
100
100
600
```

Ln 74, Col 1 Spaces: 2 UTF-8 LF C++ Mac Live Share

Functors (Function Objects)

Sometimes we can use a **function object** when an ordinary function won't work. The STL often uses function objects and provides several function objects that are very helpful. [1]

Functors are objects that can be treated as though they are a function or function pointer. Functors are most commonly used along with STLs in a scenario like following (in the next slides). [2]

```
Class X {  
public:  
    // define "function call" operator  
    return-value operator() (arguments) const;  
    ...  
};
```

Object that behaves as a function

- For example, we could define a struct names **absValue** that encapsulates the operation of converting a value of type **double** to its absolute value:

```
Class X {  
public:  
    // define "function call" operator  
    return-value operator() (arguments) const;  
    ...  
};  
struct absValue  
{  
    double operator()(double f) {  
        | return f > 0 ? f : -f;  
    }  
};
```

FUNCTION OBJECT

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

- File Tabs:** main.cpp, Polygon.cpp, Polygon.h, Makefile.
- Sidebar Icons:** Copy, Paste, Find, Undo, Redo, Open, Save, Settings, GitHub, Refresh.
- Code Area:** The main.cpp file contains C++ code demonstrating function objects. It includes definitions for triangle, square, hexagon, and absValue functions, and shows how they can be used like regular objects.
- Terminal Area:** The terminal shows the output of the absValue function, which correctly prints the absolute value of -PI (approximately -3.14159) instead of the negative value (-3.14159).
- Bottom Bar:** PROBLEMS, OUTPUT, TERMINAL (selected), DEBUG CONSOLE.
- Right Panel:** A large panel on the right displays the build log and terminal history.

```
main.cpp > main()
37     cout << (*iter)(2*i) << " | ";
38 }
39 cout << endl;
40 }
41
42 cout << "- object behaves like a function" << endl;
43
44 cout << triangle(20/pow(3,0.25)) << endl;
45 cout << square(10) << endl;
46 cout << hexagon(20/pow(3,0.25)) << endl;
47
48
49
50 //*****
51 // ----- Example: absValue -----
52 //*****
53 cout << "----- absValue -----" << endl;
54
55 absValue absObj;
56
57 cout << -PI << endl;
58 cout << absObj(-PI) << endl;
59 cout << absObj(PI) << endl;
60
61
62
63
64
65
66
67
68
69
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

----- absValue -----
-3.14159
3.14159
3.14159
(base) Ivaldo:Week4 admin\$

bash + ×

Ln 69, Col 1 Spaces: 2 UTF-8 LF C++ Mac

CLASS PrintName

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

```
class PrintName {
public:
    void operator()(const Polygon & elem){
        cout << elem.shapeName() << " ";
    }
};

absValue absObj;

cout << -PI << endl;
cout << absObj(-PI) << endl;
cout << absObj(PI) << endl;

//***** Example: PrintName *****
//----- PrintName -----
cout << "----- PrintName -----" << endl;

PrintName print;

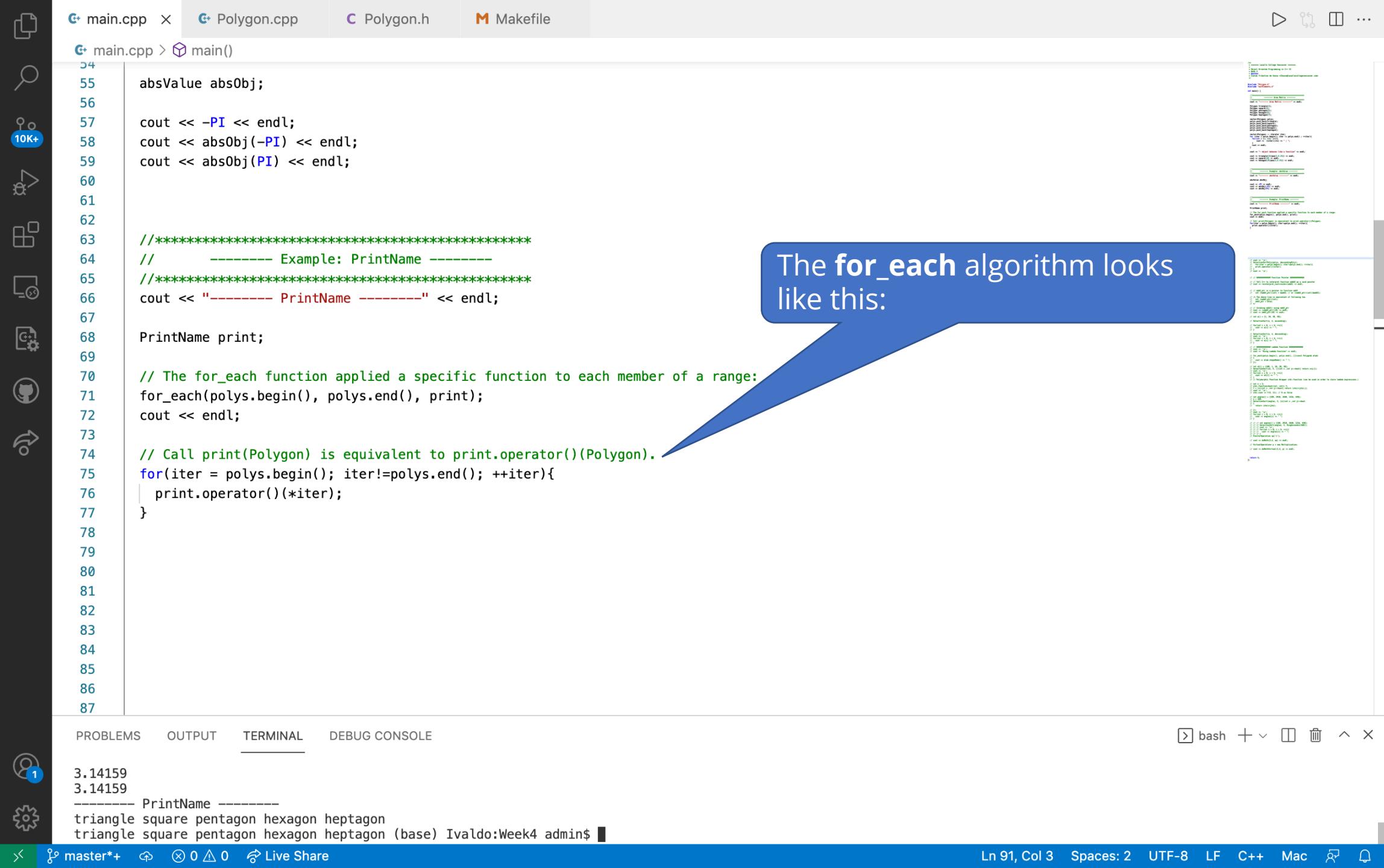
// The for_each function applied a specific function to each member of a range:
for_each(polys.begin(), polys.end(), print);
```

A callout arrow points from the text "The for_each function applied a specific function to each member of a range:" to the `for_each` line in the code.

The bottom right corner of the editor has a blue button labeled "output". Below the editor, the terminal output shows:

```
-3.14159
3.14159
3.14159
----- PrintName -----
triangle square pentagon hexagon heptagon (base) Ivaldo:Week4 admin$
```

The bottom status bar indicates: Ln 94, Col 3 Spaces: 2 UTF-8 LF C++ Mac



The **for_each** algorithm looks like this:

```
absValue absObj;
cout << -PI << endl;
cout << absObj(-PI) << endl;
cout << absObj(PI) << endl;

//***** Example: PrintName *****
//----- PrintName -----
cout << "----- PrintName -----" << endl;

PrintName print;

// The for_each function applied a specific function to each member of a range:
for_each(polys.begin(), polys.end(), print);
cout << endl;

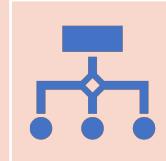
// Call print(Polygon) is equivalent to print.operator()(Polygon).
for( iter = polys.begin(); iter!=polys.end(); ++iter){
    print.operator()(*iter);
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

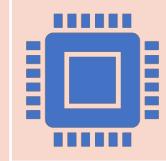
bash + ×

3.14159
3.14159
----- PrintName -----
triangle square pentagon hexagon heptagon
triangle square pentagon hexagon heptagon (base) Ivaldo:Week4 admin\$

Function Pointers



You learned that a pointer is a variable that holds the address of another variable. Function pointers are similar, except that instead of pointing to variables, they point to functions!



Much like variables, functions live at an assigned address in memory.

FUNCTION POINTERS

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

```
67     PrintName print;
68
69     // The for_each function applied a specific function to each member of a range:
70     for_each(polys.begin(), polys.end(), print);
71     cout << endl;
72
73
74     // Call print(Polygon) is equivalent to print.operator()(Polygon).
75     for(iter = polys.begin(); iter!=polys.end(); ++iter){
76         print.operator()(*iter);
77     }
78     cout << '\n';
79
80     //***** Function Pointer *****
81     // ----- Function Pointer -----
82     //***** Function Pointer *****
83     cout << "----- Function Pointer -----" << endl;
84
85     // Tell C++ to interpret function addd3 as a void pointer
86     cout << reinterpret_cast<void*>(add3) << endl;
87
88
89
90
91
92
93
94
95
96
97
98
99
```

A blue callout box highlights the code from line 81 to line 83, which outputs "----- Function Pointer -----". To the right of this box is another callout box containing the following C++ code:

```
int add3(int number){
    return 3 + number;
}
```

At the bottom of the screen, the terminal output shows:

```
triangle square pentagon hexagon heptagon
triangle square pentagon hexagon heptagon
----- Function Pointer -----
0x1012c4d90
(base) Ivaldo:Week4 admin$
```

A large blue callout box points from the terminal output to the address "0x1012c4d90", which is labeled "add3's address".

CALLING A FUNCTION USING A FUNCTION POINTER

The screenshot shows a code editor interface with a dark theme. On the left is a vertical toolbar with various icons. The main area displays a C++ file named `main.cpp`. The code demonstrates how to call a function using a function pointer. A specific section of the code is highlighted with a blue rounded rectangle.

```
main.cpp > main()
73
74 // Call print(Polygon) is equivalent to print.operator()(Polygon).
75 for(iter = polys.begin(); iter!=polys.end(); ++iter){
76     print.operator()(*iter);
77 }
78 cout << '\n';
79
80 //***** Function Pointer *****
81 // ----- Function Pointer -----
82 //***** Function Pointer *****
83 cout << "----- Function Pointer -----" << endl;
84
85 // Tell C++ to interpret function addd3 as a void pointer
86 cout << reinterpret_cast<void*>(add3) << endl;
87
88 // add3_ptr is a pointer to function add3
89 int (*add3_ptr)(int) = &add3; // or (*add3_ptr)(int)(&add3);
90
91 /* The above line is equivalent of following two
92    int (*add3_ptr)(int);
93    add3_ptr = &fun;
94 */
95
96 // Invoking add3() using add3_ptr
97 cout << (*add3_ptr)(10) << endl;
98 cout << add3_ptr(10) << endl;
99
100
101
102
103
104
105
```

The highlighted section of the code is:

```
88 // add3_ptr is a pointer to function add3
89 int (*add3_ptr)(int) = &add3; // or (*add3_ptr)(int)(&add3);

/* The above line is equivalent of following two
92    int (*add3_ptr)(int);
93    add3_ptr = &fun;
94 */

// Invoking add3() using add3_ptr
97 cout << (*add3_ptr)(10) << endl;
98 cout << add3_ptr(10) << endl;
```

Below the code editor, there are tabs for PROBLEMS, OUTPUT, TERMINAL, and DEBUG CONSOLE. The TERMINAL tab is selected, showing the output of the program:

```
----- Function Pointer -----
0x10e2f3cd0
13
13
(base) Ivaldo:Week4 admin$
```

The bottom status bar shows the file name `master*`, line count `0 △ 0`, and other system information like `Ln 107, Col 1`, `Spaces: 2`, `UTF-8`, `LF`, `C++`, and `Mac`.

Passing functions as arguments to other functions

- One of the most useful things to do with function pointers is pass a function as an argument to another function. Functions used as arguments to another function are sometimes called **callback functions**.
- Consider a case where you are writing a function to perform a task (such as sorting an array), but you want the user to be able to define how a particular part of that task will be performed (such as whether the array is sorted in ascending or descending order). So our go is change the way the algorithm sorts without affecting the rest of the sorting code. (see [4])

SelectionSort

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows a project structure under **WEEK4** containing files: **.vscode**, **c_cpp_property.h**, **settings.json**, **main**, **main.cpp**, **main.o**, **Makefile**, **ourElements.h**, **Polygon.cpp**, **Polygon.h**, and **Polygon.o**. A **10K+** icon is present.
- FILES**: Shows the **main.cpp** file open, displaying the SelectionSort algorithm.
- PROBLEMS**: Shows 13 issues.
- OUTPUT**: Shows the terminal output: `1 10 30 50 (base) Ivaldo:Week4 admin$`.
- TERMINAL**: Shows the terminal tab.
- DEBUG CONSOLE**: Shows the debug console tab.
- BASH**: Shows the bash tab.
- STATUS BAR**: Shows the status bar with `Ln 86, Col 2 Spaces: 2 UTF-8 LF C++ Mac`.

```
int a[] = {1, 10, 30, 50};

SelectionSort(a, 4);

for(int i = 0; i < 4; ++i){
    cout << a[i] << " ";
}

return 0;

void SelectionSort(int *array, int size)
{
    // Step through each element of the array
    for (int startIndex = 0 ; startIndex < (size - 1); ++startIndex)
    {
        // smallestIndex is the index of the smallest element we've encountered so far.
        int smallestIndex = startIndex;

        // Look for smallest element remaining in the array (starting at startIndex+1)
        for (int currentIndex(startIndex + 1); currentIndex < size; ++currentIndex)
        {
            // If the current element is smaller than our previously found smallest
            if (array[smallestIndex] > array[currentIndex]) // COMPARISON DONE HERE
            {
                // This is the new smallest number for this iteration
                smallestIndex = currentIndex;
            }
        }

        // Swap our start element with our smallest element
        std::swap(array[startIndex], array[smallestIndex]);
    }
}
```

Let's replace that comparison with a function to do the comparison.

```
//function that sorts in ascending order in cyclic group of order 5
bool ascendingZ5(int x, int y) {
    return x%5 > y%5;
}

//function that sorts in descending order in cyclic group of order 5
bool descendingZ5(int x, int y) {
    return x%5 < y%5;
}
```

Instead of using our own hard-coded comparison function, we'll allow the caller to provide their own sorting function! This is done via a function pointer.

```
void selectionSort(int *array, int size, bool (*comparisonFcn)(int, int))
```

SelectionSort WITH FUNCTION POINTER PARAMETER

The image shows a code editor interface with two tabs open: `main.cpp` and `SelectionSort.cpp`.

main.cpp:

```
103 //*****
104 // ----- Function Pointer -----
105 //*****
106 cout << "----- Function Pointer -----" << endl;
107
108 // Tell C++ to interpret function addd3 as a void pointer
109 cout << reinterpret_cast<void*>(add3) << endl;
110
111 // add3_ptr is a pointer to function add3
112 int (*add3_ptr)(int) = &add3; // or (*add3_ptr)(int)(&add3);
113
114 /* The above line is equivalent of following two
115    int (*add3_ptr)(int);
116    add3_ptr = &fun;
117 */
118
119 // Invoking add3() using add3_ptr
120 cout << (*add3_ptr)(10) << endl;
121 cout << add3_ptr(10) << endl;
122
123
124 int a[] = {1, 14, 30, 52, 63};
125
126 SelectionSort(a, 4, ascendingZ5);
127
128 for(int i = 0; i < 4; ++i){
129     cout << a[i] << " ";
130 }
131
132 SelectionSort(a, 4, descendingZ5);
133 cout << '\n';
134 for(int i = 0; i < 4; ++i){
135     cout << a[i] << " ";
136 }
137 cout << '\n';
138
```

SelectionSort.cpp:

```
void SelectionSort(int *array, int size, bool (*comparisonFcn)(int, int))
{
    // Step through each element of the array
    for (int startIndex = 0 ; startIndex < (size - 1); ++startIndex)
    {
        // smallestIndex is the index of the smallest element we've encountered so far.
        int smallestIndex = startIndex;

        // Look for smallest element remaining in the array (starting at startIndex+1)
        for (int currentIndex(startIndex + 1); currentIndex < size; ++currentIndex)
        {
            // If the current element is smaller than our previously found smallest
            if (comparisonFcn(array[smallestIndex], array[currentIndex])) // COMPARISON DONE HERE
            {
                // This is the new smallest number for this iteration
                smallestIndex = currentIndex;
            }
        }

        // Swap our start element with our smallest element
        std::swap(array[startIndex], array[smallestIndex]);
    }
}
```

The terminal output shows the sorted arrays:

```
30 1 52 14
14 52 1 30
(base) Ivaldo:Week4 admin$
```

Making function pointers prettier with type aliases.

- In C++11, the **using** keyword when used for **type alias** is identical to **typedef**.
- **using** keyword can bring member functions into scope.

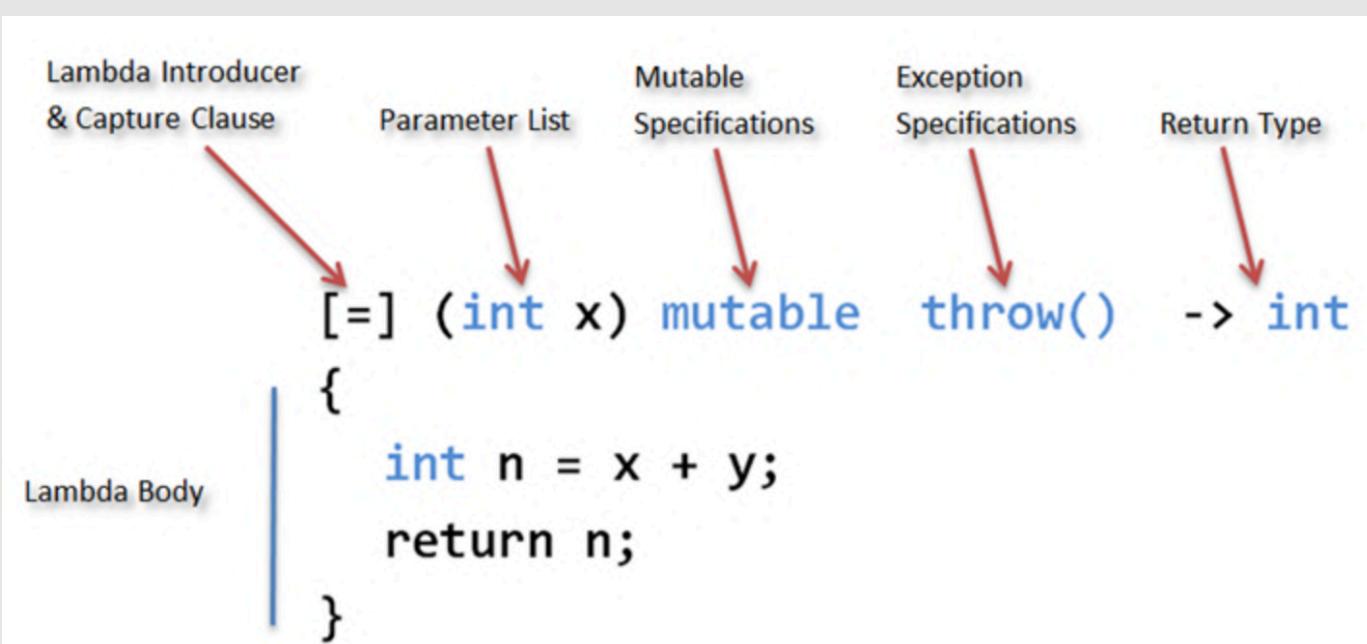
```
using BinaryPredicate = bool(*)(int, int);

void SelectionSort(int *array, int size, BinaryPredicate bp)
{
    // Step through each element of the array
    for (int startIndex = 0 ; startIndex < (size - 1); ++startIndex)
    {
        // smallestIndex is the index of the smallest element we've encountered so far.
        int smallestIndex = startIndex;

        // Look for smallest element remaining in the array (starting at startIndex+1)
        for (int currentIndex(startIndex + 1); currentIndex < size; ++currentIndex)
        {
            // If the current element is smaller than our previously found smallest
            if (bp(array[smallestIndex], array[currentIndex])) // COMPARISON DONE HERE
            {
                // This is the new smallest number for this iteration
                smallestIndex = currentIndex;
            }
        }

        // Swap our start element with our smallest element
        std::swap(array[startIndex], array[smallestIndex]);
    }
}
```

Understanding Lambda



- Is a convenient way of defining an anonymous function object (a *closure*) right at the location where it's invoked or passed as an argument to a function.
- Typically lambdas are used to encapsulate a few lines of code that are passed to functions. [6]

Lambda is just a convenient way to write a functor.

We should always think about **lambda** as a functor when coding in C++. Therefore, lambda is often used for writing callback functions. Consequently, they are very useful when dealing with STL algorithms.

```
class PrintName {  
public:  
    void operator()(const Polygon & elem){  
        cout << elem.shapeName() << " ";  
    }  
};
```

```
PrintName print;  
  
for_each(polys.begin(), polys.end(), print);
```

```
for_each(polys.begin(), polys.end(), [](const Polygon& elem)  
{  
    cout << elem.shapeName() << " ";  
});
```

Lambdas

The screenshot shows a code editor interface with a dark theme. On the left, there's a vertical toolbar with various icons: file, search, refresh, etc. The main area displays a C++ file named `main.cpp`. The code includes several lambda expressions and standard library functions like `SelectionSort`.

```
main.cpp:129 cout << a[i] << " ";
main.cpp:130 }
main.cpp:131
main.cpp:132 SelectionSort(a, 4, descendingZ5);
main.cpp:133 cout << '\n';
main.cpp:134 for(int i = 0; i < 4; ++i){
main.cpp:135 | cout << a[i] << " ";
main.cpp:136 }
main.cpp:137 cout << '\n';
main.cpp:138
main.cpp:139 //*****
main.cpp:140 // ----- Lambda Function -----
main.cpp:141 //*****
main.cpp:142 cout << "----- Lambda Function -----" << endl;
main.cpp:143
main.cpp:144 for_each(polys.begin(), polys.end(), [](const Polygon& elem)
main.cpp:145 {
main.cpp:146 | cout << elem.shapeName() << " ";
main.cpp:147 });
main.cpp:148 cout << '\n';
main.cpp:149
main.cpp:150 int a1[] = {101, 17, 56, 18, 99};
main.cpp:151 SelectionSort(a1, 5, [](int x ,int y)->bool{ return x%8<y%8;});
main.cpp:152
main.cpp:153 for(int i = 0; i < 5; ++i){
main.cpp:154 | cout << a1[i] << " ";
main.cpp:155 }
main.cpp:156 cout << '\n';
main.cpp:157
main.cpp:158
main.cpp:159
main.cpp:160
main.cpp:161
main.cpp:162
```

The code editor has two sections of code highlighted with blue rounded rectangles. The first section starts at line 144 and ends at line 149. The second section starts at line 150 and ends at line 156. Both sections contain lambda expressions.

At the bottom, the terminal window shows the output of the program:

```
----- Lambda Function -----
triangle square pentagon hexagon heptagon
101 99 18 17 56
(base) Ivaldo:Week4 admin$
```

Below the terminal, the status bar shows the current file is `master*`, the line and column are `Ln 163, Col 3`, and the encoding is `UTF-8`.

Std::Function



The screenshot shows a code editor interface with several tabs at the top: main.cpp, Polygon.cpp, ourElements.h, libraries.h, and Makefile. The main.cpp tab is active, displaying C++ code. A sidebar on the left contains various icons for file operations like copy, paste, search, and refresh. The code editor has a status bar at the bottom with tabs for PROBLEMS, OUTPUT, TERMINAL, and DEBUG CONSOLE, and a terminal section showing command-line output.

```
main.cpp > main()
142     cout << "----- Lambda Function -----" << endl;
143
144     for_each(polys.begin(), polys.end(), [](const Polygon& elem)
145     {
146         cout << elem.shapeName() << " ";
147     });
148     cout << '\n';
149
150     int a1[] = {101, 17, 56, 18, 99};
151     SelectionSort(a1, 5, [](int x ,int y)->bool{ return x%8<y%8;});
152
153     for(int i = 0; i < 5; ++i){
154         cout << a1[i] << " ";
155     }
156     cout << '\n';
157
158     cout << "----- std::function -----" << endl;
159     // Polymorphic Function Wrapper std::function (can be used in order to store lambda expressions.)
160
161     int n = 3;
162     std::function<bool(int, int)> f;
163     f = [n](int x ,int y)->bool{ return (x%n)>(y%n);};
164     std::cout << f(9, 11) << endl; // 0 so false
165
166
167
168
169
170
171
172
173
174
```

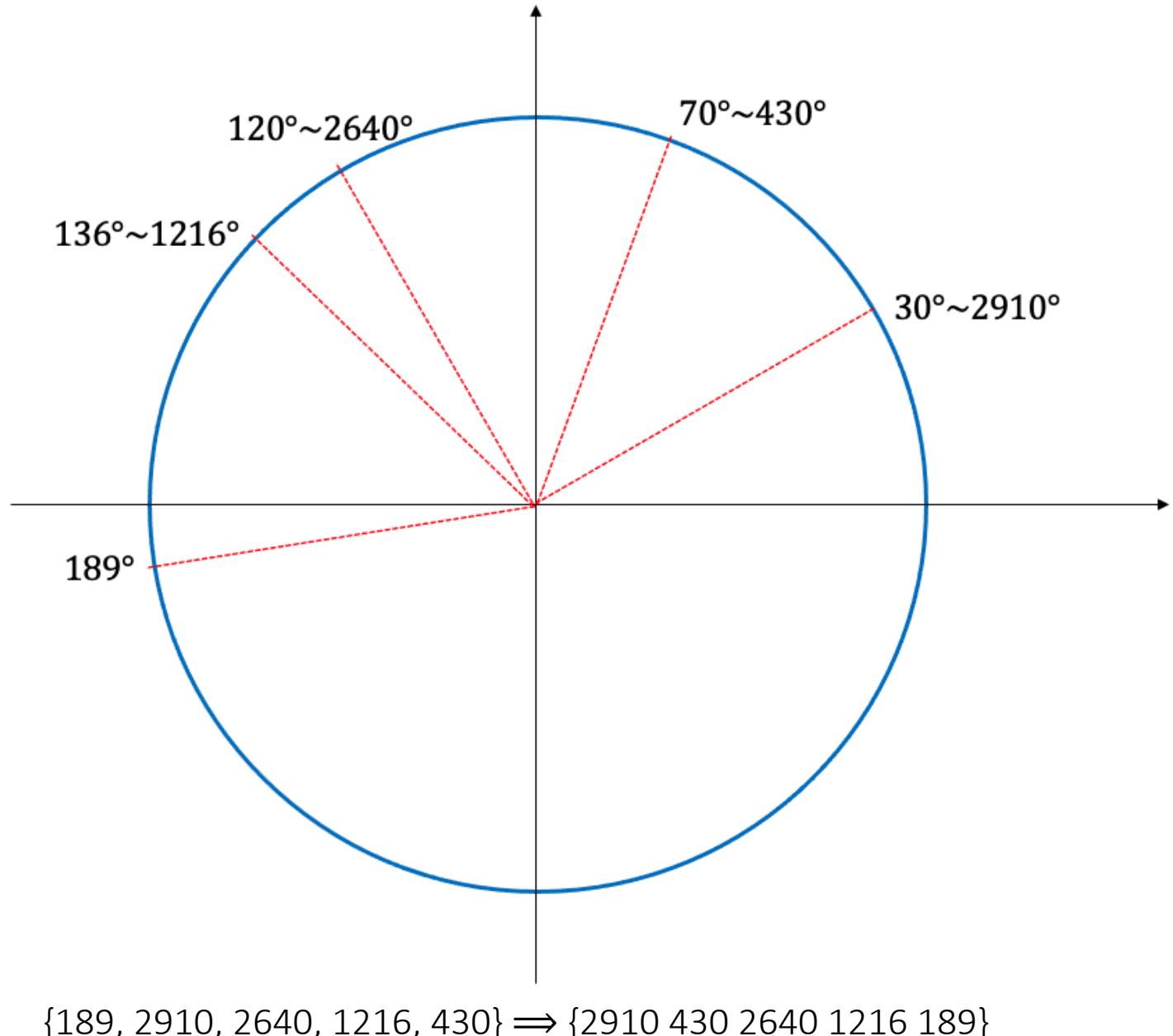
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash + ×

```
----- Lambda Function -----
triangle square pentagon hexagon heptagon
101 99 18 17 56
----- std::function -----
0
(base) Ivaldo:Week4 admin$
```

<div data-bbox="104

Problem: Put the angles in ascending order taking into account their locations on the unit circle.



$\{189, 2910, 2640, 1216, 430\} \Rightarrow \{2910 \ 430 \ 2640 \ 1216 \ 189\}$

```
struct RingAscendin
{
    int quotient;
    RingAscendin(int q) : quotient(q){}

    bool operator()(int x, int y){
        return (x%quotient) > (y%quotient);
    }
};
```



```
int n = 360;
[n](int x ,int y)->bool{ return (x%n)>(y%n);}
```

Lambdas

The screenshot shows a C++ code editor with several tabs at the top: main.cpp 1, Polygon.cpp, ourElements.h, libraries.h, and Makefile. The main.cpp tab is active, displaying the following code:

```
144     for_each(polys.begin(), polys.end(), [](const Polygon& elem)
145     {
146         cout << elem.shapeName() << " ";
147     });
148     cout << '\n';
149
150     int a1[] = {101, 17, 56, 18, 99};
151     SelectionSort(a1, 5, [](int x, int y)->bool{ return x%8<y%8;});
152
153     for(int i = 0; i < 5; ++i){
154         cout << a1[i] << " ";
155     }
156     cout << '\n';
157
158     cout << "----- std::function -----" << endl;
159 // Polymorphic Function Wrapper std::function (can be used in order to store lambda expressions.)
160
161     int n = 3;
162     std::function<bool(int, int)> f;
163     f = [n](int x, int y)->bool{ return (x%n)>(y%n);};
164     std::cout << f(9, 11) << endl; // 0 so false
165
166     cout << "----- Problem -----";
167
168     int angles[] = {189, 2910
169     n = 360;
170     SelectionSort(angles, 5, [n](int x, int y)->bool
171     {
172         return (x%n)>(y%n);
173     });
174     for(int i = 0; i < 5; ++i){
```

A tooltip appears over the line `[n](int x, int y)->bool`, stating: "no suitable conversion function from "lambda []bool (int x, int y)->bool" to "BinaryPredicate" exists C/C++ (413)". Below the tooltip, there are "View Problem" and "Quick Fix... (⌘.)" buttons.

A red arrow points from the text "Lambda cannot be passed as a function pointer if it has a capture" to the tooltip.

On the left sidebar, there are icons for file, search, refresh, and other development tools. The status bar at the bottom shows: PROBLEMS 1, OUTPUT, TERMINAL, DEBUG CONSOLE, bash, and file navigation icons.

At the bottom, the terminal output shows:

```
argument
void SelectionSort(int *array, int size, BinaryPredicate bp)

1 error generated.
make: *** [main.o] Error 1
(base) Ivaldo:Week4 admin$
```

The status bar at the bottom right indicates: Ln 181, Col 1, Spaces: 2, UTF-8, LF, C++, Mac, and a few other icons.

Pass lambdas as arguments of functions:

There are two ways to pass lambdas as arguments of functions:

1. The STL way, with template

```
template<typename T>
void SelectionSort(int *array, int size, T bp)
```

2. Use std::function

```
void SelectionSort(int *array, int size,
                   std::function<bool(int, int)> bp)
```

```
template<typename T>
void SelectionSort(int *array, int size, T bp)
{
    // Step through each element of the array
    for (int startIndex = 0 ; startIndex < (size - 1); ++startIndex)
    {
        // smallestIndex is the index of the smallest element we've encountered so far.
        int smallestIndex = startIndex;

        // Look for smallest element remaining in the array (starting at startIndex+1)
        for (int currentIndex(startIndex + 1) ; currentIndex < size; ++currentIndex)
        {
            // If the current element is smaller than our previously found smallest
            if (bp(array[smallestIndex], array[currentIndex])) // COMPARISON DONE HERE
            {
                // This is the new smallest number for this iteration
                smallestIndex = currentIndex;
            }
        }

        // Swap our start element with our smallest element
        std::swap(array[startIndex], array[smallestIndex]);
    }
}
```

Lambdas

The screenshot shows a code editor interface with a dark theme. On the left is a vertical toolbar with various icons: file, search, refresh, 10K+, copy, paste, settings, GitHub, and a refresh/circular arrow icon. The main area displays a C++ file named `main.cpp`. The code uses lambdas and `std::function` to sort arrays and print results. The terminal below shows the execution of the program, outputting sorted lists of numbers and a message about polymorphic function wrappers.

```
main.cpp > main()
150  int a1[] = {101, 17, 56, 10, 55}, 
151      SelectionSort(a1, 5, [](int x ,int y)->bool{ return x%8<y%8;});
152
153  for(int i = 0; i < 5; ++i){
154      cout << a1[i] << " ";
155  }
156  cout << '\n';
157
158  cout << "----- std::function -----" << endl;
159 // Polymorphic Function Wrapper std::function (can be used in order to store lambda expressions.)
160
161  int n = 3;
162  std::function<bool(int, int)> f;
163  f = [n](int x ,int y)->bool{ return (x%n)>(y%n);};
164  std::cout << f(9, 11) << endl; // 0 so false
165
166  cout << "----- Problem -----" << endl;
167
168  int angles[] = {189, 2910, 2640, 1216, 430};
169  n = 360;
170  SelectionSort(angles, 5, [n](int x ,int y)->bool
171  {
172      return (x%n)>(y%n);
173  });
174  for(int i = 0; i < 5; ++i){
175      cout << angles[i] << " ";
176  }
177
178
179
180
181
182
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

triangle square pentagon hexagon heptagon
101 99 18 17 56
----- std::function -----
0
----- Problem -----
2910 430 2640 1216 189 (base) Ivaldo:Week4 admin\$

bash + × ^ ×

Ln 181, Col 1 Spaces: 2 UTF-8 LF C++ Mac Live Share

Bibliography

1. Geeksforgeeks (2022/01/24) Functors in C++,
<https://www.geeksforgeeks.org/functors-in-cpp/>
2. Bogotobogo (2022/01/24) C++ TUTORIAL - FUNCTORS,
<https://www.bogotobogo.com/cplusplus/functors.php>
3. Cppreference (2022/01/24) std::function,
<https://en.cppreference.com/w/cpp/utility/functional/function>
4. Cppreference (2022/01/24) std::for_each,
https://en.cppreference.com/w/cpp/algorithm/for_each
5. LEARN C++ (2022/01/24) Function Pointers,
<https://www.learncplusplus.com/cpp-tutorial/function-pointers/>
6. Towards Data Science (2022/01/24) C++ Basics: Understanding Lambda, <https://towardsdatascience.com/c-basics-understanding-lambda-7df00705fa48>
7. Microsoft (2022/01/24) C++ Lambda expressions in C++,
<https://docs.microsoft.com/en-us/cpp/cpp/lambda-expressions-in-cpp?view=msvc-160>
8. Wikipedia (2022/01/24) Quotient ring,
https://en.wikipedia.org/wiki/Quotient_ring

Appendix

Libraries.h

Polygon.h

Polygon.cpp

ourElements.h

main.cpp