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;
}
```

AREA MATRIX

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

main.cpp > main()

```
40  //***********************************************
41  //           -------- Area Matrix --------
42  //***********************************************
43  cout << "-------- Area Matrix --------" << endl;
44
45  Polygon triangle(3);
46  Polygon square(4);
47  Polygon pentagon(5);
48  Polygon hexagon(6);
49  Polygon heptagon(7);
50
51  vector<Polygon> polys; // for the fastest implementation use pointers. vector<Polygon*> polys
52  polys.reserve(5);
53
54  polys.push_back(triangle);
55  polys.push_back(square);
56  polys.push_back(pentagon);
57  polys.push_back(hexagon);
58  polys.push_back(heptagon);
59
60  for(Polygon const & poly : polys){
61    for(int i =1; i<6; i++){
62      cout <<  poly(2*i) << " | ";
63    }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
-------- Area Matrix --------
Constructor Invoked
Constructor Invoked
Constructor Invoked
Constructor Invoked
Constructor Invoked
Copy Constructor Invoked
Copy Constructor Invoked
Copy Constructor Invoked
Copy Constructor Invoked
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 |
Polygon destroyed
```

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

master*+ ⊗ 0 ⚠ 0 Live Share Ln 227, Col 1 (4701 selected) Spaces: 2 UTF-8 LF C++ Mac

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

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

main.cpp > main()

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

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Copy Constructor Invoked
Copy Constructor Invoked
Copy Constructor Invoked
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
```

The object was called as if it were an ordinary function.

master*+ ⊗ 0 ⚠ 0 Live Share Ln 74, Col 1 Spaces: 2 UTF-8 LF C++ Mac

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

main.cpp | Polygon.cpp | Polygon.h | Makefile

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

bash

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

master*+ 0 0 Live Share Ln 69, Col 1 Spaces: 2 UTF-8 LF C++ Mac

CLASS PrintName

main.cpp | Polygon.cpp | Polygon.h | Makefile

main.cpp > main()

```
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  //       -------- Example: PrintName --------
65  //************************************************
66  cout << "-------- PrintName --------" << endl;
67
68  PrintName print;
69
70  // The for_each function applied a specific function to each member of a range:
71  for_each(polys.begin(), polys.end(), print);
72
73
74
75
76
77
78
79
80
81
```

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

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

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

output

master*+ 0 0 Live Share Ln 94, Col 3 Spaces: 2 UTF-8 LF C++ Mac

CLASS PrintName

main.cpp | Polygon.cpp | Polygon.h | Makefile

main.cpp > main()

```
absValue absObj;

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



//************************************************
//        -------- Example: 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);
}
```

The **for_each** algorithm looks like this:

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

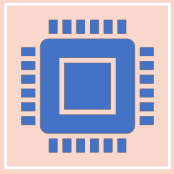
bash

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

master*+ 0 0 Live Share Ln 91, Col 3 Spaces: 2 UTF-8 LF C++ Mac

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

main.cpp | Polygon.cpp | Polygon.h | Makefile

main.cpp > ...

```
67
68  PrintName print;
69
70  // The for_each function applied a specific function to each member of a range:
71  for_each(polys.begin(), polys.end(), print);
72  cout << endl;
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  //************************************************
81  //       -------- Function Pointer --------
82  //************************************************
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
```

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

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

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

add3's address

master*+ | 0 ⚠ 0 | Live Share | Ln 188, Col 19 | Spaces: 2 | UTF-8 | LF | C++ | Mac

CALLING A FUNCTION USING A FUNCTION POINTER

main.cpp | Polygon.cpp | Polygon.h | Makefile

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  //************************************************
81  //       -------- Function Pointer --------
82  //************************************************
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
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

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

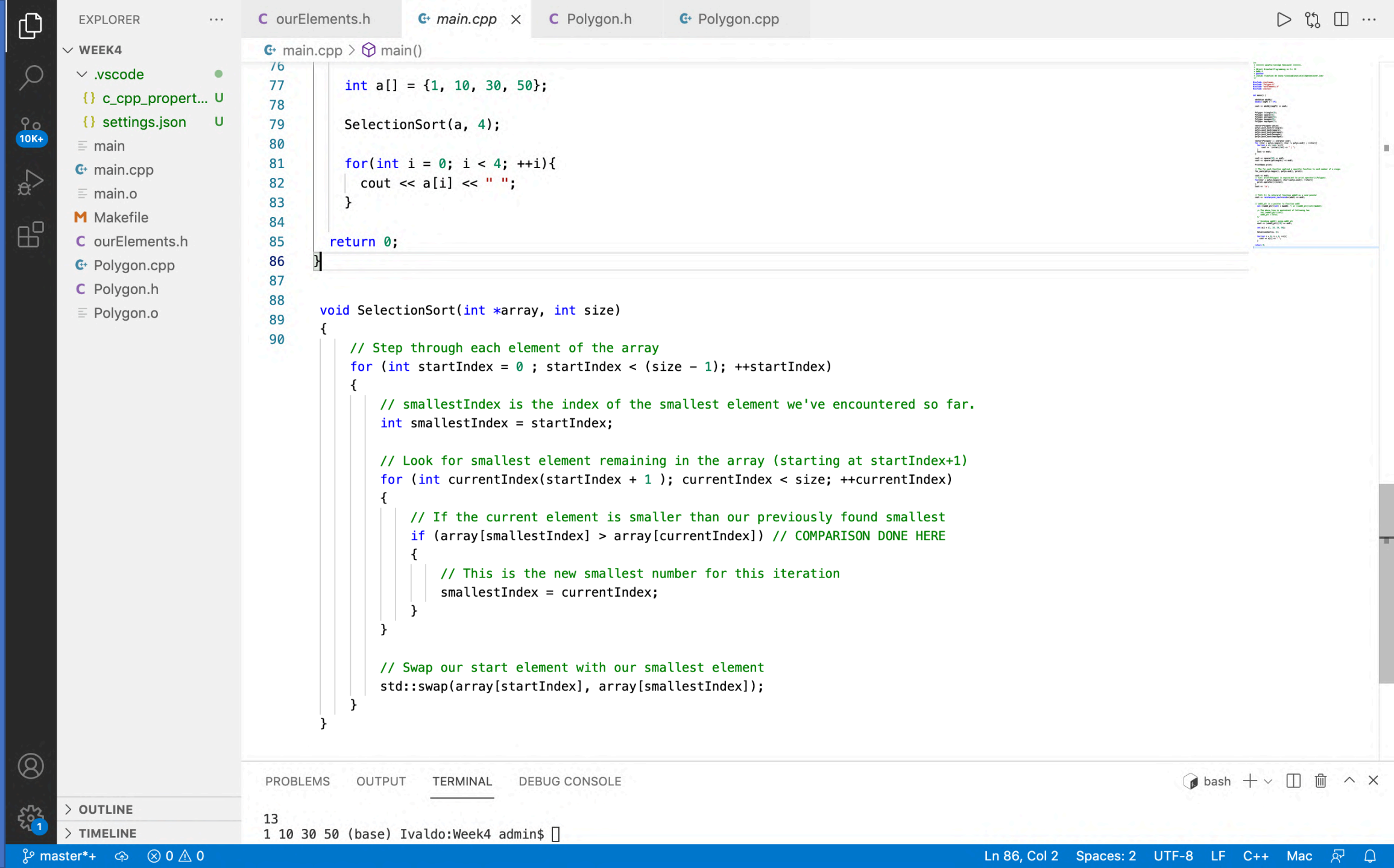
master*+ | 0 ⚠ 0 | Live Share | Ln 107, Col 1 | Spaces: 2 | UTF-8 | LF | C++ | Mac

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



Let’s replace that comparison with a functions 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

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

main.cpp > main()

```cpp
//************************************************
//       -------- Function Pointer --------
//************************************************
cout << "-------- Function Pointer --------" << endl;

// Tell C++ to interpret function addd3 as a void pointer
cout << reinterpret_cast<void*>(add3) << endl;

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

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

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

int a[] = {1, 14, 30, 52, 63};

SelectionSort(a, 4, ascendingZ5);

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

SelectionSort(a, 4, descendingZ5);
cout << '\n';
for(int i = 0; i < 4; ++i){
  cout << a[i] << " ";
}
cout << '\n';
```

```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]);
    }
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

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

master*+ 0 0 Live Share Ln 203, Col 1 (1559 selected) Spaces: 2 UTF-8 LF C++ Mac

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.

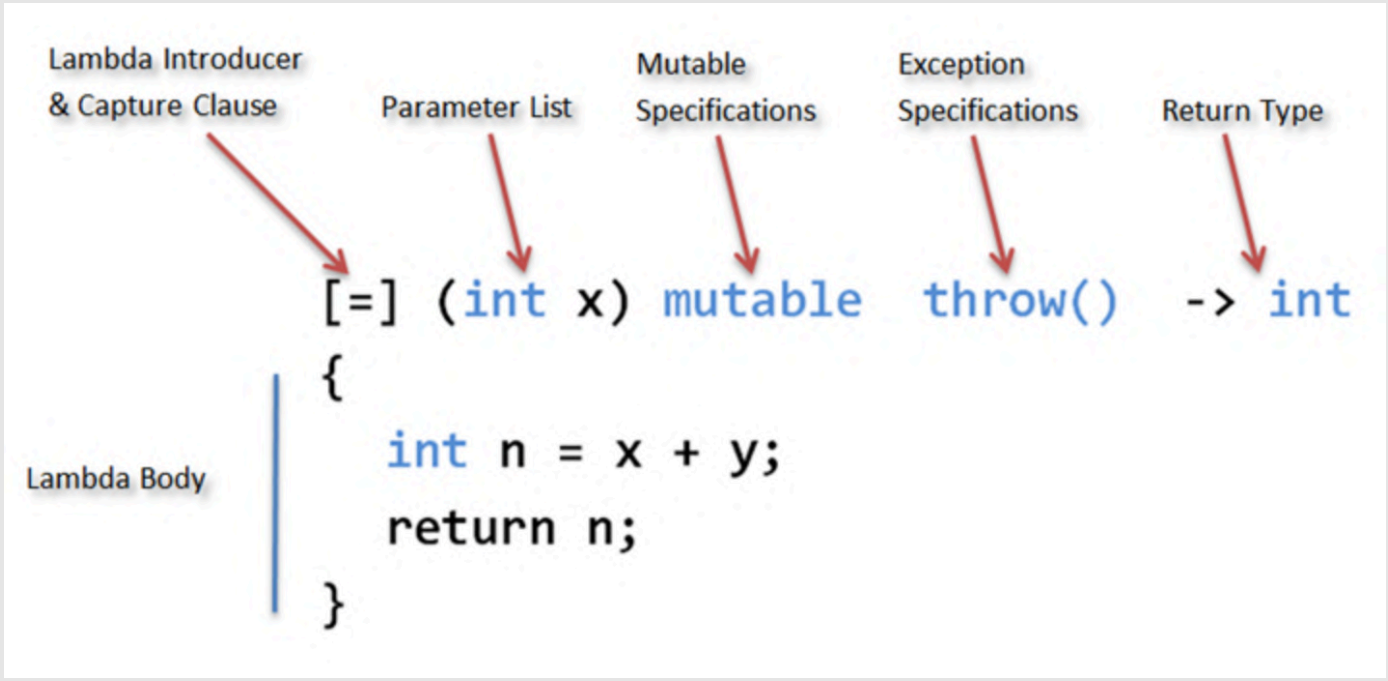
```
using BinaryPredicate = bool(*)(int, int);

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

main.cpp ● | Polygon.cpp | ourElements.h | libraries.h | Makefile

main.cpp > main()

```
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
139 //***********************************************
140 //        -------- Lambda Function --------
141 //***********************************************
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
159
160
161
162
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

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

master*+ ⊗ 0 ⚠ 0 Live Share Ln 163, Col 3 Spaces: 2 UTF-8 LF C++ Mac

Std::Function

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

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
```

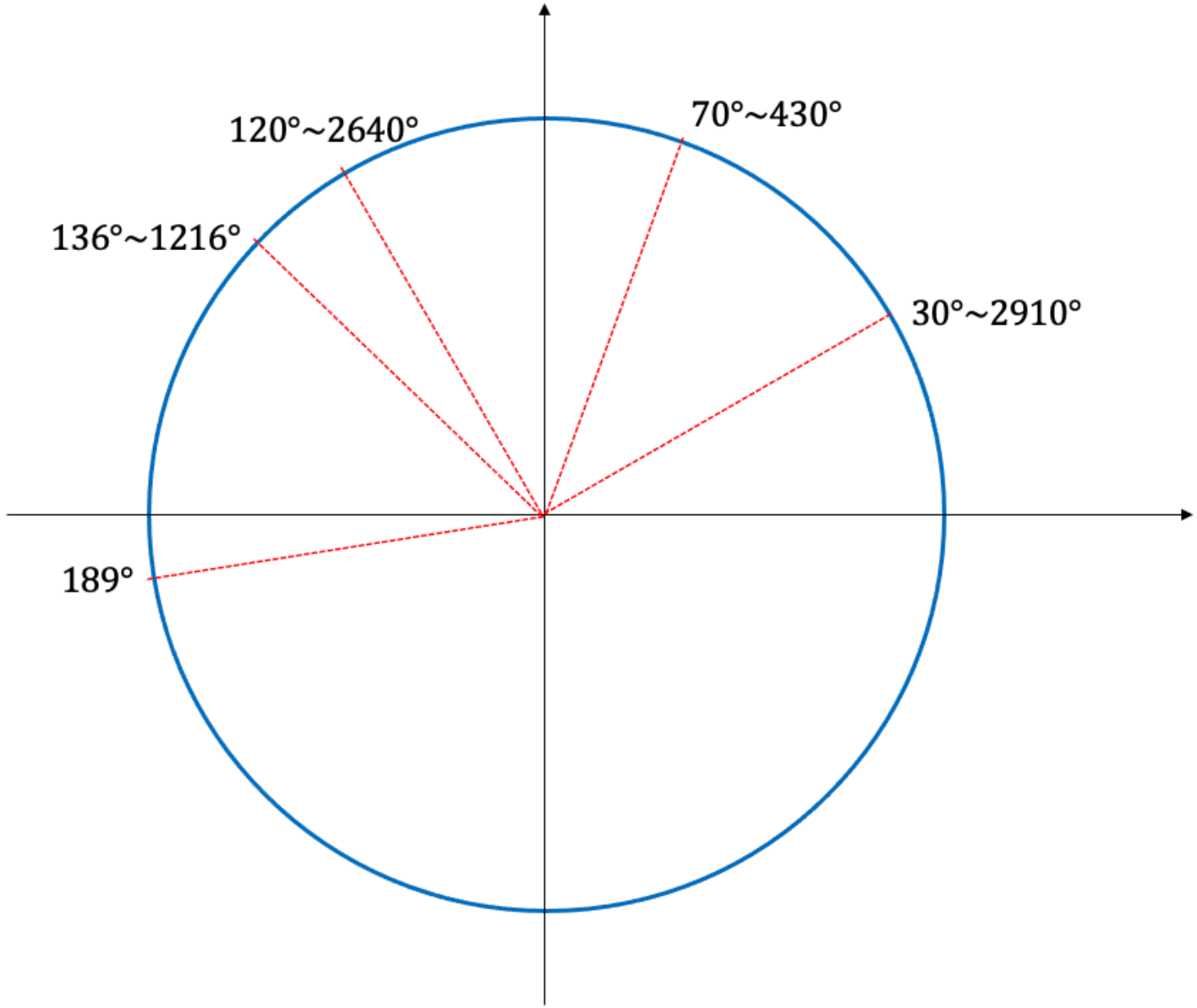
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

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

master*+ 0 0 Live Share Ln 180, Col 1 Spaces: 2 UTF-8 LF C++ Mac

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



{189, 2910, 2640, 1216, 430} ⟹ {2910 430 2640 1216 189}

{189, 2910, 2640, 1216, 430} ⟹ {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

main.cpp 1 | Polygon.cpp | ourElements.h | libraries.h | Makefile

main.cpp > main()

```
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  });
175  for(int i = 0; i < 5; ++i){
```

no suitable conversion function from "lambda []bool (int x, int y)->bool" to "BinaryPredicate" exists C/C++ (413)

View Problem Quick Fix... (⌘.)

There is a hidden parameter in the *operator()*

Lambda cannot be passed as a function pointer if it has a capture

PROBLEMS 1 | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

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

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

master*+ | 1 ⚠ 0 | Live Share | Ln 181, Col 1 | Spaces: 2 | UTF-8 | LF | C++ | Mac

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)
```

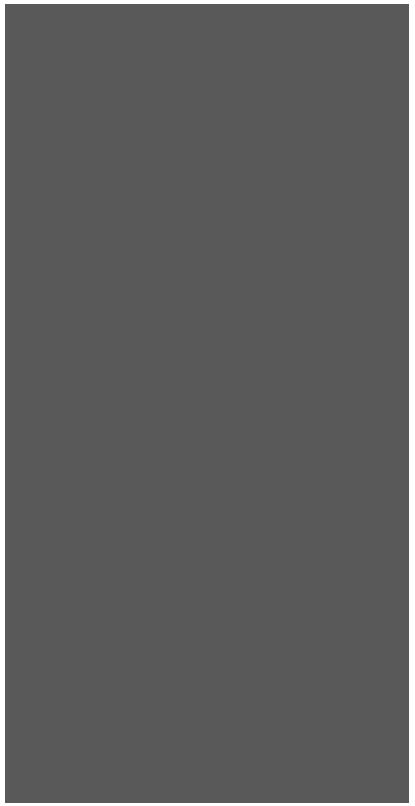
2. Use std::function

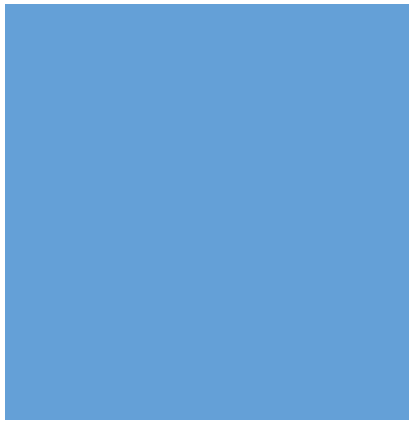
```
void SelectionSort(int *array, int size,
                   std::function<bool(int, int)> bp)
```

```
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

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

main.cpp > main()

```
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  });
175  for(int i = 0; i < 5; ++i){
176    cout << angles[i] << " ";
177  }
178
179
180
181
182
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

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

master*+ 0 0 Live Share Ln 181, Col 1 Spaces: 2 UTF-8 LF C++ Mac

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.learncpp.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