VGP131 – OOPII

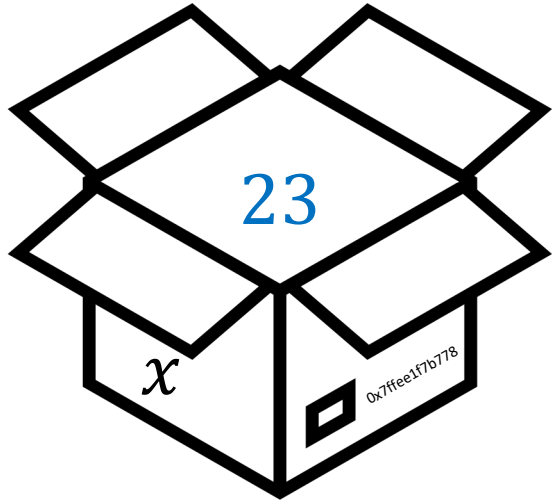
Instructor: Ivaldo Tributino

Value categories Move Semantics

Lvalues and rvalues

The terms *“right-hand mode”* and *“left-hand mode”* to describe the semantics of particular expressions. When an expression was evaluated in *left-hand mode*, it yielded an address, and when it was evaluated in *right-hand mode*, it yielded a “rule for the computation of a value”. [3]

$$x = x + 1$$



In C++, *lvalue* is something that points to a specific memory location. On the other hand, a *rvalue* is something that doesn't point anywhere. In general, rvalues are temporary and short lived, while lvalues live a longer life since they exist as variables. It's also fun to think of lvalues as *containers* and rvalues as *things contained in the containers*.[1]

LVALUE AND RVALUE

main.cpp 2 ●

main.cpp > main()

```cpp
14
15  static int x = 23;
16
17  //A function that returns an lvalue.
18  int& getLvalue(){
19    return x;
20  }
21
22  void ref1(int& x){}
23
24  void ref2(const int& x){}
25
26  int main(){
27
28      //************************************************
29      //       -------- lvalues and rvalues --------
30      //************************************************
31      cout << "-------- lvalues and rvalues --------" << '\n';
32
33      x = x + 1;
34      cout << &x << '\n';
35
36      // error: cannot take the address of an rvalue of type 'int'
37      cout << &13 << '\n';
38
39      // Expression must be an lvalue or a function designator.
40      cout << &(x+1) << '\n';
41
42
43
44
45
46
47
48
```

lvalue is something that points to a specific memory location. On the other hand, a *rvalue* is something that doesn't point anywhere

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

bash

```
-------- lvalues and rvalues --------
0x10c00f130
(base) Ivaldo:Week7 admin$
```

master*+ ⊗ 2 ⚠ 0 Live Share Ln 47, Col 1 Spaces: 4 UTF-8 LF C++ Mac

LVALUE AND RVALUE

main.cpp 2 ●

main.cpp > main()

```
14
15  static int x = 23;
16
17  //A function that returns an lvalue.
18  int& getLvalue(){
19    return x;
20  }
21
22  void ref1(int& x){}
23
24  void ref2(const int& x){}
25
26  int main(){
27
28      //************************************************
29      //       -------- lvalues and rvalues --------
30      //************************************************
31      cout << "-------- lvalues and rvalues --------" << '\n';
32
33      x = x + 1;
34      cout << &x << '\n';
35
36      // error: cannot take the address of an rvalue of type 'int'
37      cout << &13 << '\n';
38
39      // Expression must be an lvalue or a function designator.
40      cout << &(x+1) << '\n';
41
42
43
44
45
46
47
48
```

Lvalue to rvalue conversion:
An lvalue is an expression that will appear on the left-hand side or on the right-hand side of an assignment.

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

bash

```
-------- lvalues and rvalues --------
0x10c00f130
(base) Ivaldo:Week7 admin$
```

master*+ ⊗ 2 ⚠ 0 Live Share Ln 47, Col 1 Spaces: 4 UTF-8 LF C++ Mac

LVALUE AND RVALUE

main.cpp 1

main.cpp > main()

```
24  void ref2(const int& x){}
25
26  int main(){
27
28      //***********************************************
29      //       -------- lvalues and rvalues --------
30      //***********************************************
31      cout << "-------- lvalues and rvalues --------" << '\n';
32
33      x = x + 1;
34      cout << &x << '\n';
35
36      // error: cannot take the address of an rvalue of type 'int'
37      // cout << &13 << '\n';
38
39      // Expression must be an lvalue or a function designator.
40      // cout << &(x+1) << '\n';
41
42
43      // Taking the memory address of x and setting it to y(pointer), using the & (ampersand) operator.
44      int* y = &x;
45      cout << y << '\n';
46      cout << *y << '\n';
47
48      // Expression must be an lvalue or a function designator.
49      int* y = &14;
50
51
52
53
54
55
56
57
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

bash

```
-------- lvalues and rvalues --------
0x10471e138
0x10471e138
24
(base) Ivaldo:Week7 admin$
```

master*+ ⊗ 1 ⚠ 0 Live Share Ln 53, Col 1 Spaces: 4 UTF-8 LF C++ Mac

LVALUE AND RVALUE

main.cpp

main.cpp > main()

```
35
36      // error: cannot take the address of an rvalue of type 'int'
37      // cout << &13 << '\n';
38
39      // Expression must be an lvalue or a function designator.
40      // cout << &(x+1) << '\n';
41
42
43      // Taking the memory address of x and setting it to y(pointer), using the & (ampersand) operator.
44      int* y = &x;
45      cout << y << '\n';
46      cout << *y << '\n';
47
48      // Expression must be an lvalue or a function designator.
49      // int* y = &14;
50
51      cout << "- A function that returns an lvalue "<< '\n';
52
53      cout << getLvalue() << '\n';
54      cout << &getLvalue() << '\n';
55
56      getLvalue() = 100;
57
58      cout << getLvalue() << '\n';
59      cout << &getLvalue() << '\n';
60
61      getLvalue()++;  // Increment operator
62      cout << getLvalue() << '\n';
63
```

```
static int x = 23;

//A function that returns an lvalue.
int& getLvalue(){
  return x;
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
-------- lvalues and rvalues --------
0x103c52138
0x103c52138
24
- A function that returns an lvalue
24
0x103c52138
100
0x103c52138
101
(base) Ivaldo:Week7 admin$
```

master*+ 0 0 Live Share Ln 68, Col 5 Spaces: 4 UTF-8 LF C++ Mac

LVALUE AND RVALUE

main.cpp 1 ●

main.cpp > main()

```
66    cout << getLvalue() << '\n';
67    cout << &getLvalue() << '\n';
68
69    getLvalue()++;  // Increment operator
70    cout << getLvalue() << '\n';
71
72    cout << "- Assigned an integer directly to my reference" << '\n';
73    {
74    int& ref = x;
75    cout << &ref << endl;
76    // error: initial value of reference to non-const must be an lvalue
77    int& ref = 10;
78    }
79
80
81
82
83
84
85
86
87
88
89
90
91
92
```

initial value of reference to non-const must be an lvalue C/C++(461)

View Problem Quick Fix... (⌘.)

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE bash

```
-------- lvalues and rvalues --------
0x100f79138
0x100f79138
24
- A function that returns an lvalue
24
0x100f79138
100
0x100f79138
101
- Assigned an integer directly to my reference
0x100f79138
(base) Ivaldo:Week7 admin$
```

master*+ ⊗ 1 ⚠ 0 Live Share Ln 86, Col 1 Spaces: 4 UTF-8 LF C++ Mac

LVALUE AND RVALUE

```
main.cpp 1 ●
main.cpp > main()
67      cout << &getLvalue() << '\n';
68
69      getLvalue()++;  // Increment operator
70      cout << getLvalue() << '\n';
71
72      cout << "- Assigned an integer directly to my reference" << '\n';
73      {
74      int& ref = x;
75      cout << &ref << endl;
76      // error: initial value of reference to non-const must be an lvalue
77      // int& ref = 10;
78      }
79
80      cout << "--- Lvalue ---" << endl;
81      int a = 10;
82      passLvalue(a);
83      passLvalue(10);
84
85
86
87
88
89
90
91
92
```

initial value of reference to non-const must be an lvalue C/C++(461)

View Problem Quick Fix... (⌘.)

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE bash

```
-------- lvalues and rvalues --------
0x1033d8138
0x1033d8138
24
- A function that returns an lvalue
24
0x1033d8138
100
0x1033d8138
101
- Assigned an integer directly to my reference
0x1033d8138
--- Lvalue ---
10
(base) Ivaldo:Week7 admin$
```

```
void passLvalue(int& x){
    cout << x << endl;
}
```

master*+ ⊗ 1 ⚠ 0 Live Share Ln 85, Col 1 Spaces: 4 UTF-8 LF C++ Mac

LVALUE AND RVALUE

main.cpp

main.cpp > main()

```
71
72      cout << "- Assigned an integer directly to my reference" << '\n';
73      {
74      int& ref = x;
75      cout << &ref << endl;
76      // error: initial value of reference to non-const must be an lvalue
77      // int& ref = 10;
78      }
79
80      cout << "--- Lvalue ---" << endl;
81      int a = 10;
82      passLvalue(a);
83      // passLvalue(10);
84
85      cout << "--- Lvalue & Rvalue ---" << endl;
86      {
87      const int& ref = 10;
88      cout << ref << endl;
89      int a = 10;
90      passAllvalue(a);
91      passAllvalue(10);
92      }
93
94
95
96
97
98
99
100
101
102
```

Here, 10 is an rvalue, so it *appears* that &ref is an address of the 10. No, that is wrong. &ref is an address of *the temporary object* of type int, which is a hidden variable.

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
--- Lvalue ---
10
--- Lvalue & Rvalue ---
10
10
10
(base) Ivaldo:Week7 admin$
```

```
void passAllvalue(const int& x){
    cout << x << endl;
}
```

master*+ 0 0 Live Share Ln 104, Col 1 Spaces: 4 UTF-8 LF C++ Mac

LVALUE AND RVALUE

main.cpp 1 ●

main.cpp > passAllvalue(const int &)

```
74    int& ref = x;
75    cout << &ref << endl;
76    // error: initial value of reference to non-const must be an lvalue
77    // int& ref = 10;
78    }
79
80    cout << "--- Lvalue ---" << endl;
81    int a = 10;
82    passLvalue(a);
83    // passLvalue(10);
84
85    cout << "--- Lvalue & Rvalue ---" << endl;
86    {
87    const int& ref = 10;
88    cout << ref << endl;
89    int a = 10;
90    passAllvalue(a);
91    passAllvalue(10);
92    }
93
94    {
95    // error: variable 'r
96    const int& ref = 10;
97    cout << ref++ << endl;
98    }
99
100
101
102
103
104
105
```

```
const int &ref

error: variable 'ref' declared const here

expression must be a modifiable lvalue C/C++(137)

View Problem    Quick Fix... (⌘.)
```

PROBLEMS 1 | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
--- Lvalue ---
10
--- Lvalue & Rvalue ---
10
10
10
(base) Ivaldo:Week7 admin$
```

master*+ ⊗ 1 ⚠ 0 Live Share | Ln 31, Col 14 | Spaces: 4 | UTF-8 | LF | C++ | Mac

LVALUE AND RVALUE

main.cpp 1 | smartPtr.h | libraries.h

main.cpp > main()

```
47
48      cout << "- A function that returns an lvalue "<< '\n';
49
50      cout << getLvalue() << '\n';
51      cout << &getLvalue() << '\n';
52
53      getLvalue() = 100;
54
55      cout << getLvalue() << '\n';
56      cout << &getLvalue() << '\n';
57
58      getLvalue()++;
59      cout << getLvalue() << '\n';
60
61      cout << "- Assigned an integer directly to my reference" << '\n';
62
63      // error: initial value of reference to non-const must be an lvalue
64      // int& yref = 101; // non-const lvalue reference to type 'int' cannot bind to a temporary of type 'int'
65      // ref1(101); //candidate function not viable: expects an l-value for 1st argument
66
67      const int& ref = 101;
68      cout << ref << '\n';
69      cout << &ref << '\n';
70      ref2(101);
71
72      // error: variable 'ref' declared const here
73      cout << ref++ << '\n';
74
75
76
77
78
79
```

```
const int &ref

expression must be a modifiable lvalue C/C++(137)

View Problem    Quick Fix... (⌘.)
```

PROBLEMS 1 | OUTPUT | TERMINAL | DEBUG CONSOLE

```
main.cpp:67:16: note: variable 'ref' declared const here
    const int& ref = 101;
    ~~~~~~~~~~~^~~~~~~~~
1 error generated.
make: *** [main.o] Error 1
(base) Ivaldo:Week7 admin$
```

bash
bash

master*+ | 1 ⚠ 0 | Live Share | Ln 81, Col 5 | Spaces: 4 | UTF-8 | LF | C++ | Mac

RVALUE REFERENCES

main.cpp

```
87  const int& ref = 10;
88  cout << ref << endl;
89  int a = 10;
90  passAllvalue(a);
91  passAllvalue(10);
92  }
93
94  // {
95  // // error: variable 'ref' declared const here
96  // const int& ref = 10;
97  // cout << ref++ << endl;
98  // }
99
100 cout << "- T&& (double ampersand)" << '\n';
101
102 int &&refRvalue = 101;
103 cout << refRvalue << '\n';
104 cout << &refRvalue << '\n';
105 refRvalue++;
106 cout << refRvalue << '\n';
107
108 passRvalue(14);
109
110
111
112
113
114
115
116
117
118
```

It can bind to an rvalue like a temporary without having to be const.

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
- T&& (double ampersand)
101
0x7ffee84bd76c
102
15
(base) Ivaldo:Week7 admin$
```

```
void passRvalue(int&& x){
    cout << ++x << endl;
}
```

master*+ ⊗ 0 ⚠ 0 Live Share Ln 119, Col 1 Spaces: 4 UTF-8 LF C++ Mac

RVALUE REFERENCES

main.cpp

main.cpp > main()

```
106     cout << refRvalue << '\n';
107
108     passRvalue(14);
109
110     {
111
112     Polygon poly(4);
113
114     auto f_Lvalue = [](Polygon& p){ return p.shapeName();};
115
116     auto f_Allvalue = [](const Polygon& p){ return p.shapeName();};
117
118     auto f_Rvalue = [](Polygon&& p){ return p.shapeName();};
119
120     cout << f_Lvalue(poly) << endl;
121
122     cout << f_Allvalue(poly) << endl;
123     cout << f_Allvalue(Polygon(4)) << endl;
124
125     cout << f_Rvalue(Polygon(4)) << endl;
126     }
127
128
129
130
131
132
133
134
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
Constructor Invoked
square
square
Constructor Invoked
square
Polygon was destructive
Constructor Invoked
square
Polygon was destructive
Polygon was destructive
(base) Ivaldo:Week7 admin$
```

master*+ 0 0 Live Share Ln 130, Col 1 Spaces: 4 UTF-8 LF C++ Mac

STD :: MOVE

main.cpp

main.cpp > main()

```
106     cout << refRvalue << '\n';
107
108     passRvalue(14);
109
110     {
111
112     Polygon poly(4);
113
114     auto f_Lvalue = [](Polygon& p){ return p.shapeName();};
115
116     auto f_Allvalue = [](const Polygon& p){ return p.shapeName();};
117
118     auto f_Rvalue = [](Polygon&& p){ return p.shapeName();};
119
120     cout << f_Lvalue(poly) << endl;
121
122     cout << f_Allvalue(poly) << endl;
123     cout << f_Allvalue(Polygon(4)) << endl;
124
125     cout << f_Rvalue(Polygon(4)) << endl;
126     cout << f_Rvalue(std::move(poly)) << endl;
127     }
128
129
130
131
132
133
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
Constructor Invoked
square
square
Constructor Invoked
square
Polygon was destructive
Constructor Invoked
square
Polygon was destructive
square
Polygon was destructive
(base) Ivaldo:Week7 admin$
```

master*+ 0 0 Live Share Ln 130, Col 1 Spaces: 4 UTF-8 LF C++ Mac

STD :: MOVE

main.cpp

main.cpp > main()

```
127     }
128
129     cout << "- std::move" << '\n';
130
131     // Error: An rValue reference cannot be pointed to a lValue.
132     // int &&refRvalue = x;
133     refRvalue = x;
134
135     cout << &refRvalue << '\n';
136     cout << &x << '\n';
137
138     cout << refRvalue << '\n';
139     cout << x << '\n';
140
141     int && refRvalue1 = std::move(x);
142
143     cout << &refRvalue1 << '\n';
144     cout << &x << '\n';
145
146     cout << refRvalue1 << '\n';
147     cout << x << '\n';
148
149
150
151
152
153
154
155
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
- std::move
0x7ffee250476c
0x10d700140
101
101
0x10d700140
0x10d700140
101
101
(base) Ivaldo:Week7 admin$
```

master*+ 0 0 Live Share Ln 153, Col 1 Spaces: 4 UTF-8 LF C++ Mac

Two types of references[2]

In C++ there are two types of references:

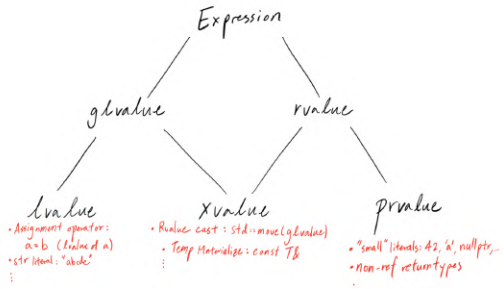
lvalue reference:

- An lvalue is an expression that will appear on the left-hand side or on the right-hand side of an assignment.
- Simply, a variable or object that has a name and memory address.
- It uses one ampersand (&).

rvalue reference:

- An rvalue is an expression that will appear only on the right-hand side of an assignment.
- A variable or object has only a memory address (temporary objects).
- It uses two ampersands (&&).

Value categories



lvalue

- (Expressions with identity but not movable from) is a glvalue (“generalized” lvalue) that is not an xvalue;

xvalue

- (Expressions with identity that are moveable from) (an “eXpiring” value) is a glvalue that denotes an object whose resources can be reused;

Rvalue

- (Expressions without identity that are moveable from)

Move Semantics

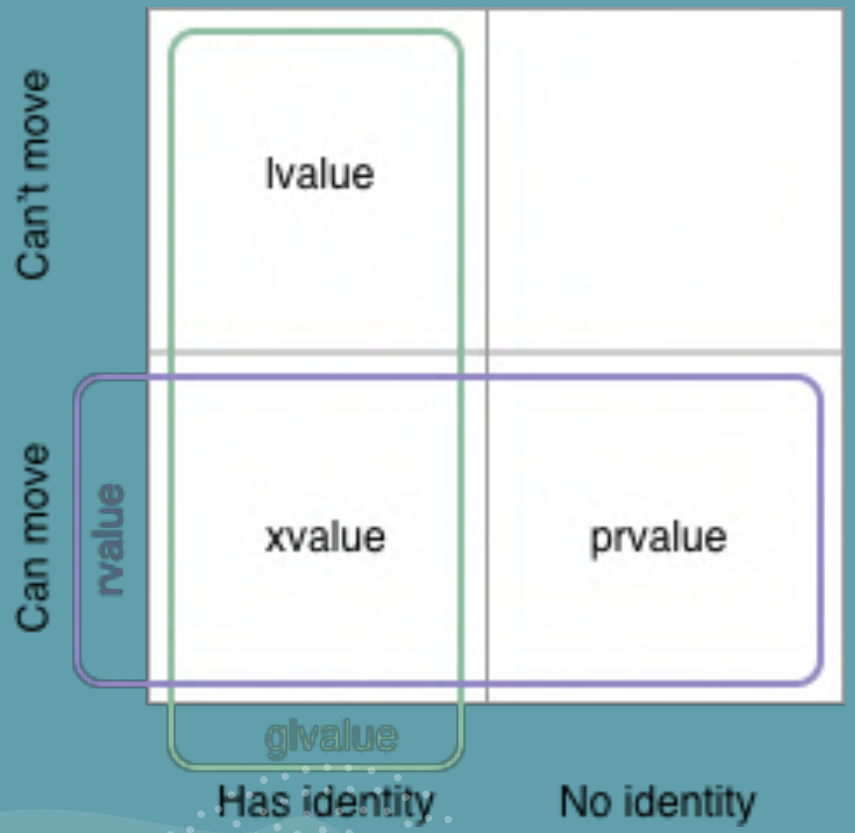
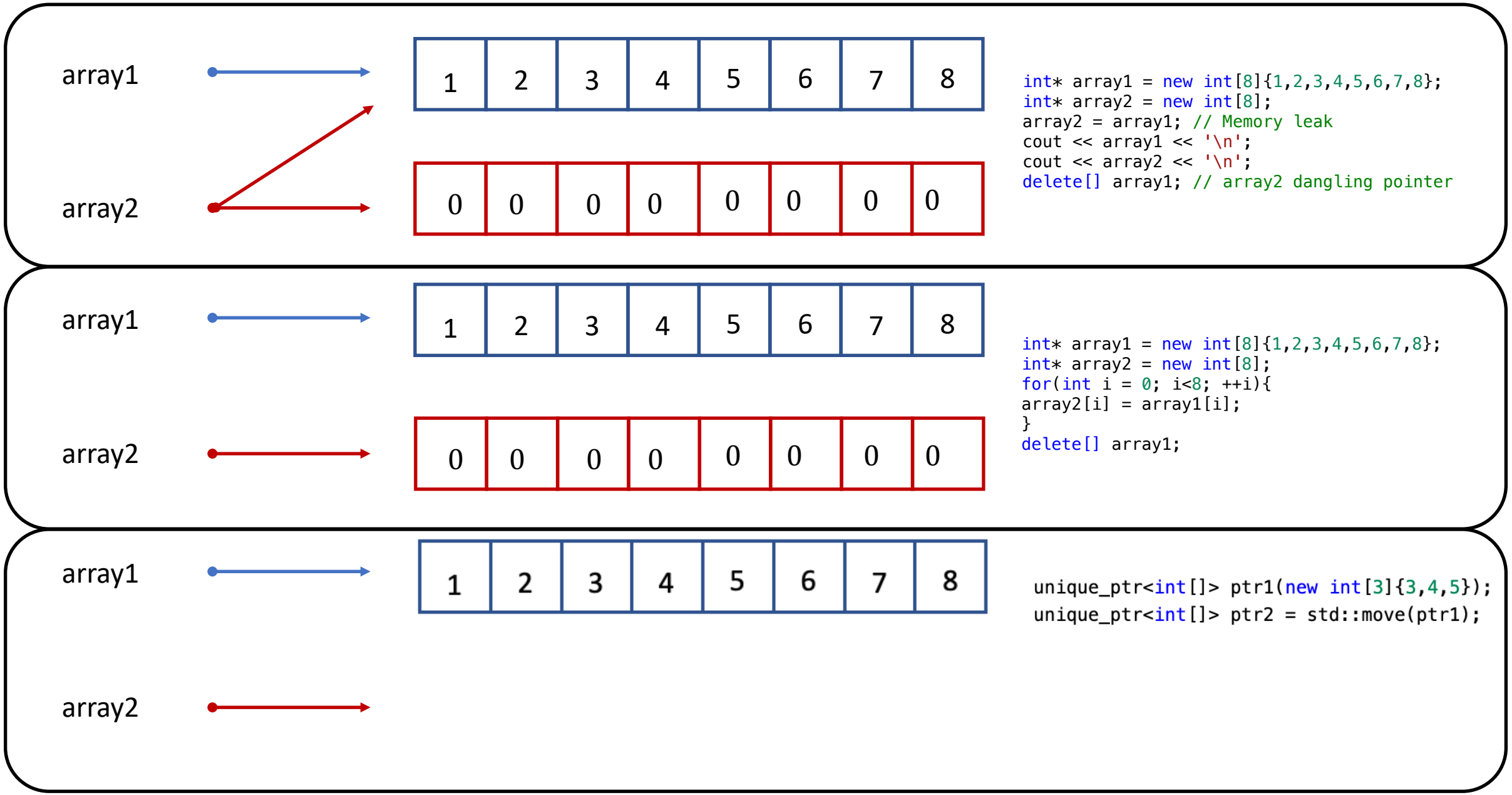


Figure from: https://accu.org/journals/overload/27/150/knatten_2641/

- The functions that accept rvalue reference parameters (including move constructors, move assignment operators, and regular member functions such as std::vector::push_back) are selected, by overload resolution, when called with rvalue arguments.

- Names of rvalue reference variables are lvalues and have to be converted to xvalues to be bound to the function overloads that accept rvalue reference parameters, which is why move constructors and move assignment operators typically use std::move:


array1
array2
1 2 3 4 5 6 7 8
0 0 0 0 0 0 0 0
int* array1 = new int[8]{1,2,3,4,5,6,7,8};
int* array2 = new int[8];
array2 = array1; // Memory leak
cout << array1 << '\n';
cout << array2 << '\n';
delete[] array1; // array2 dangling pointer
array1
array2
1 2 3 4 5 6 7 8
0 0 0 0 0 0 0 0
int* array1 = new int[8]{1,2,3,4,5,6,7,8};
int* array2 = new int[8];
for(int i = 0; i<8; ++i){
array2[i] = array1[i];
}
delete[] array1;
array1
array2
1 2 3 4 5 6 7 8
unique_ptr<int[]> ptr1(new int[3]{3,4,5});
unique_ptr<int[]> ptr2 = std::move(ptr1);

rvalue references

Design goals of C++ is to avoid memory allocations. In effect, we move objects instead of copying them.

That's what rvalue references and move semantics are for! Move semantics allows you to avoid unnecessary copies when working with temporary objects that are about to evaporate, and whose resources can safely be taken from that temporary object and used by another.

STD :: MOVE

main.cpp

main.cpp > main()

```cpp

    cout << "-std::unique_ptrs" << '\n';

    /*
    int* array1 = new int[8]{1,2,3,4,5,6,7,8};
    int* array2 = new int[8];
    array2 = array1; // Memory leak
    cout << array1 << '\n';
    cout << array2 << '\n';
    delete[] array1; // array2 dangling pointer
    */

    int* array1 = new int[8]{1,2,3,4,5,6,7,8};
    int* array2 = new int[8];
    for(int i = 0; i<8; ++i){
        array2[i] = array1[i];
    }
    delete[] array1;


    unique_ptr<int[]> ptr1(new int[8]{1,2,3,4,5,6,7,8});
    unique_ptr<int[]> ptr2 = std::move(ptr1); // memory resource is transferred to another unique_ptr

    for(unsigned i=0; i<8;++i){
        cout << ptr2[i] << "\n";
    }


```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
-std::unique_ptrs
1
2
3
4
5
6
7
8
(base) Ivaldo:Week7 admin$
```

master*+ 0 0 Live Share Ln 176, Col 5 Spaces: 4 UTF-8 LF C++ Mac

OUR SMART POINTER

```
template <class T>
class SmartPtr {

private:

    T* ptr;

public:

    // Constructor
    explicit SmartPtr(T* p= nullptr); // controls unwanted implicit type conversions.

    // Destructor
    ~SmartPtr();

    // Move Constructor
    SmartPtr(SmartPtr<T>&& obj) noexcept;

    // Move Assignment operator
    SmartPtr & operator=(SmartPtr<T> && obj) noexcept;

    // Overloading dereferncing operator
    T& operator*();

    // Overloading arrow operator
    T* operator->();

};
```

```
template <class T>
SmartPtr<T> :: SmartPtr(SmartPtr<T>&& obj) noexcept
{
    ptr = obj.ptr;
    obj.ptr = nullptr;
    cout << "Move Constructor Invoked" << endl;
}

template <class T>
SmartPtr<T> & SmartPtr<T> :: operator=(SmartPtr<T>&& obj) noexcept
{
    if (this != &obj) // beware of self-assignment
    {
        delete ptr; // release the old resource
        ptr = obj.ptr; // acquire the new resource
        obj.ptr = nullptr;
}
    cout << "Move Assignment operator invoked" << endl;
    return *this;
}
```

OUR SMART POINTER

main.cpp ● smartPtr.h

main.cpp > main()

```
174     }
175
176
177     //************************************************
178     //          ------ Move Semantics ------
179     //************************************************
180     cout << "-------- Move Semantics --------" << '\n';
181
182     cout << "- Our Smart Pointer" << '\n';
183     {
184         SmartPtr<Polygon> ptr1(new Polygon(5));
185         SmartPtr<Polygon> ptr2 = std :: move(ptr1);
186         cout << ptr2->shapeName() << '\n';
187         cout << &(*ptr1) << '\n';
188
189         SmartPtr<Polygon> ptr3(new Polygon(6));
190         ptr3 = std::move(ptr2);
191         cout << ptr3->shapeName() << '\n';
192         cout << &(*ptr2) << '\n';
193     }
194
195
196
197
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
- Our Smart Pointer
Constructor Invoked
Pointer Constructor Invoked
Move Constructor Invoked
pentagon
0x0
Constructor Invoked
Pointer Constructor Invoked
Polygon was destructive
Move Assignment operator invoked
pentagon
0x0
Polygon was destructive
Pointer destroyed
Pointer destroyed
Pointer destroyed
(base) Ivaldo:Week7 admin$
```

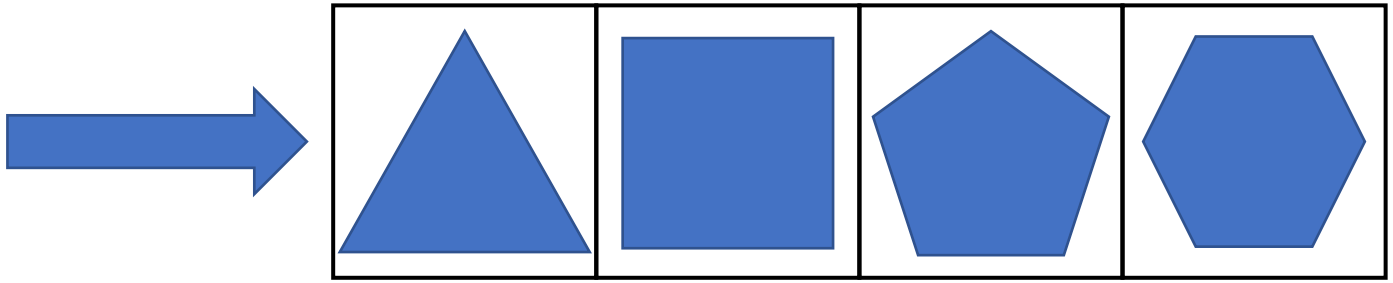
master*+ ⊗ 0 ⚠ 0 Live Share Ln 200, Col 1 Spaces: 4 UTF-8 LF C++ Mac

POLYARRAY CLASS

```
class PolyArray
{
private:
    // Polygon* _data;
    unique_ptr<Polygon[]> _data;
    int _size;

public:

    PolyArray ();

    PolyArray (int n);

    // copy constructor
    PolyArray (const PolyArray& other);

    // move constructor
    PolyArray (PolyArray&& other);

    // move assignment operator
    PolyArray& operator=(PolyArray&& other);

    // Overloading operator[]
    Polygon& operator[](int index);

    int getSize();

    void setSize(unsigned size);

    ~PolyArray() = default;

};
```

POLYARRAY CLASS

`unique_ptr<Polygon[]>`



COPY CONSTRUCTOR

main.cpp | polyArray.h | polyArray.cpp | Polygon.cpp | smartPtr.h | libraries.h

main.cpp > main()

```
147  //************************************************
148  //          ------ PolyArray ------
149  //************************************************
150  cout << "------- PolyArray ---------" << '\n';
151
152  {
153      int size = 5;
154      PolyArray array1(size);
155      PolyArray array2 = array1;
156      // PolyArray array2 = std :: move(array1);
157
158      for(int i=0; i<size; ++i){
159          cout <<"Shape Name: " << array2[i].shapeName() << ';';
160          cout <<" Number of sides: " << array2[i].getNumberSides()<< ';';
161          cout <<" Area: " << array2[i](4)<< '\n';
162      }
163
164  }
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
------- PolyArray ---------
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Copy constructor in PolyArray Invoked
Shape Name: triangle; Number of sides: 3; Area: 6.9282
Shape Name: square; Number of sides: 4; Area: 16
Shape Name: pentagon; Number of sides: 5; Area: 27.5276
Shape Name: hexagon; Number of sides: 6; Area: 41.5692
Shape Name: heptagon; Number of sides: 7; Area: 58.1426
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
```

master*+ | 0 ⚠ 0 | Live Share | Ln 163, Col 1 | Spaces: 4 | UTF-8 | LF | C++ | Mac

MOVE CONSTRUCTOR

main.cpp | polyArray.h | polyArray.cpp | Polygon.cpp | smartPtr.h | libraries.h

main.cpp > main()

```
142         ptr3 = std::move(ptr2);
143         cout << &(*ptr3) << '\n';
144 
145     }
146 
147     //************************************************
148     //          ------ PolyArray ------
149     //************************************************
150     cout << "------- PolyArray ---------" << '\n';
151 
152     {
153         int size = 5;
154         PolyArray array1(size);
155         // PolyArray array2 = array1;
156         PolyArray array2 = std :: move(array1);
157 
158         for(int i=0; i<size; ++i){
159             cout <<"Shape Name: " << array2[i].shapeName() << ';';
160             cout <<" Number of sides: " << array2[i].getNumberSides()<< ';';
161             cout <<" Area: " << array2[i](4)<< '\n';
162         }
163 
164     }
165 
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
------- PolyArray ---------
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Move constructor in PolyArray Invoked
Shape Name: triangle; Number of sides: 3; Area: 6.9282
Shape Name: square; Number of sides: 4; Area: 16
Shape Name: pentagon; Number of sides: 5; Area: 27.5276
Shape Name: hexagon; Number of sides: 6; Area: 41.5692
Shape Name: heptagon; Number of sides: 7; Area: 58.1426
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
(base) Ivaldo:Week7 admin$ 
```

master*+ | 0 ⚠ 0 | Live Share | Ln 164, Col 6 | Spaces: 4 | UTF-8 | LF | C++ | Mac

Push_back

- The C++11 compiler recognizes that the argument of push_back is an rvalue, that is, a temporary object that doesn't have a name and to which the address-of operator cannot be applied. When the C++11 compiler sees an rvalue, it calls the overload push_back(T &&value) taking an rvalue reference. This overload creates the CTeam object and moves it into the table container, as the following debug trace proves. The C++11 compiler chooses the move constructor instead of the copy constructor.[5]

PUSH BACK

main.cpp ● | polyArray.h | polyArray.cpp | Polygon.cpp | smartPtr.h | libraries.h

main.cpp > main()

```
156         PolyArray array2 = std :: move(array1);
157
158         for(int i=0; i<size; ++i){
159             cout <<"Shape Name: " << array2[i].shapeName() << ';';
160             cout <<" Number of sides: " << array2[i].getNumberSides()<< ';';
161             cout <<" Area: " << array2[i](4)<< '\n';
162         }
163
164     }
165
166     cout << "- Push_back in vector<Polygon>" << '\n';
167     {
168         vector<Polygon> v;
169         v.reserve(3); //only allocation
170         v.push_back(Polygon(3));
171         v.push_back(Polygon(4));
172         v.push_back(Polygon(5));
173     }
174
175
176
177
178
179
180
181
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
- Push_back in vector<Polygon>
Constructor Invoked
Copy Constructor Invoked
Polygon was destructive
Constructor Invoked
Copy Constructor Invoked
Polygon was destructive
Constructor Invoked
Copy Constructor Invoked
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
(base) Ivaldo:Week7 admin$
```

master*+ | 0 ⚠ 0 | Live Share | Ln 179, Col 1 | Spaces: 4 | UTF-8 | LF | C++ | Mac

PUSH BACK

main.cpp | polyArray.h | polyArray.cpp | Polygon.cpp | smartPtr.h | libraries.h

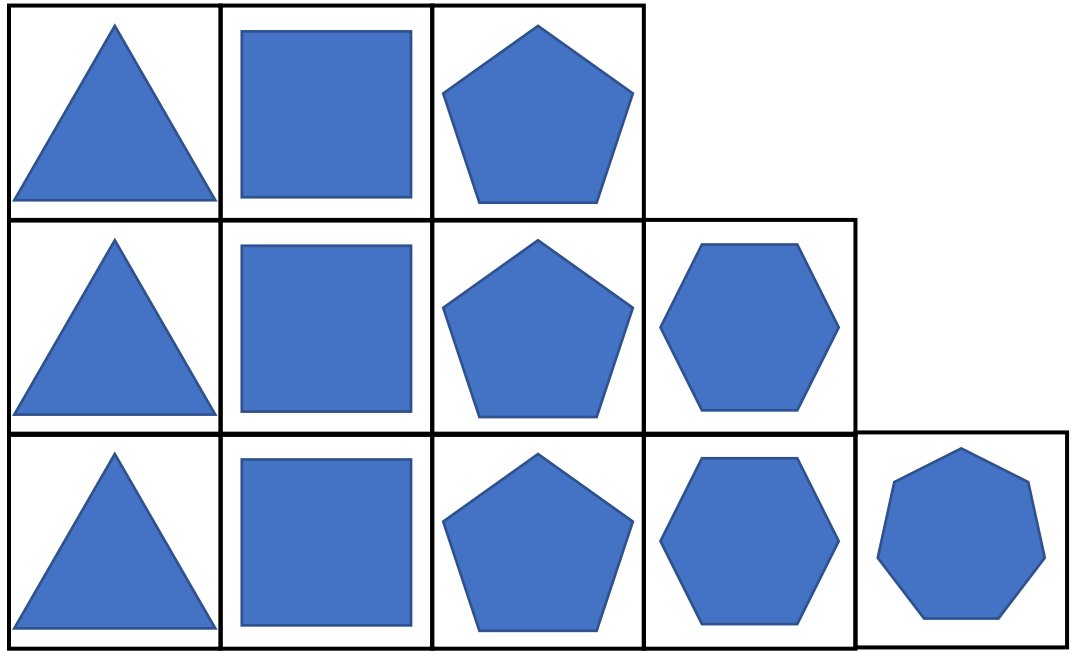
main.cpp > main()

```
170         v.push_back(Polygon(3));
171         v.push_back(Polygon(4));
172         v.push_back(Polygon(5));
173     }
174
175     cout << "- Push_back in vector<PolyArray>" << '\n';
176     {
177         vector<PolyArray> matrix;
178         matrix.reserve(3); //only allocation
179         matrix.push_back(PolyArray(3));
180         matrix.push_back(PolyArray(4));
181         matrix.push_back(PolyArray(5));
182     }
183
184
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
- Push_back in vector<PolyArray>
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Move constructor in PolyArray Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Move constructor in PolyArray Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Move constructor in PolyArray Invoked
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
(base) Ivaldo:Week7 admin$
```



master*+ 0 0 Live Share Ln 188, Col 5 Spaces: 4 UTF-8 LF C++ Mac

MATRIX

main.cpp | smartPtr.h | classA.h | Source.cpp | polyArray.h | polyArray.cpp

main.cpp > main()

```
176     {
177         vector<PolyArray> matrix;
178         matrix.reserve(3); //only allocation
179         matrix.push_back(PolyArray(3));
180         matrix.push_back(PolyArray(4));
181         matrix.push_back(PolyArray(5));
182     }
183
184     cout << "- PolyMatrix " << '\n';
185     {
186         int column = 5;
187         int row = 6;
188
189         PolyArray arrayPoly(row);
190         for(int i=0; i<row; ++i){
191             for(int j=0; j<column; ++j){
192                 cout<< arrayPoly[i](j+1) << '|';
193             }
194             cout << '\n';
195         }
196     }
197
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
- PolyMatrix
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
Default Constructor Invoked
0.433013|1.73205|3.89711|6.9282|10.8253|
1|4|9|16|25|
1.72048|6.88191|15.4843|27.5276|43.0119|
2.59808|10.3923|23.3827|41.5692|64.9519|
3.63391|14.5356|32.7052|58.1426|90.8478|
4.82843|19.3137|43.4558|77.2548|120.711|
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
Polygon was destructive
(base) Ivaldo:Week7 admin$
```

master*+ | 0 ⚠ 0 | Live Share | Ln 224, Col 9 (243 selected) | Spaces: 4 | UTF-8 | LF | C++ | Mac

Bibliography

1. Twitter(2022/02/10) Understanding the meaning of lvalues and rvalues in C++, https://www.internalpointers.com/post/understanding-meaning-lvalues-and-rvalues-c
2. Geeksforgeeks (2022/02/10) std::move in Utility in C++ | Move Semantics, Move Constructors and Move Assignment Operators, https://www.geeksforgeeks.org/stdmove-in-utility-in-c-move-semantics-move-constructors-and-move-assignment-operators/
3. Oneraynyday (2022/02/10) Value-Categories, https://oneraynyday.github.io/dev/2020/07/03/Value-Categories/
4. Stackoverflow (2022/02/10) What is move semantics?, https://stackoverflow.com/questions/3106110/what-is-move-semantics
5. Embeddeduse (2022/02/10) Performance Gains Through C++11 Move Semantics, https://embeddeduse.com/2016/05/25/performance-gains-through-cpp11-move-semantics/

Appendix

- Libraries.h
- smartPtr.h
- Polygon.h
- Polygon.cpp
- PolyArray.h
- PolyArray.cpp
- main.cpp