

Value categories Move Semantics

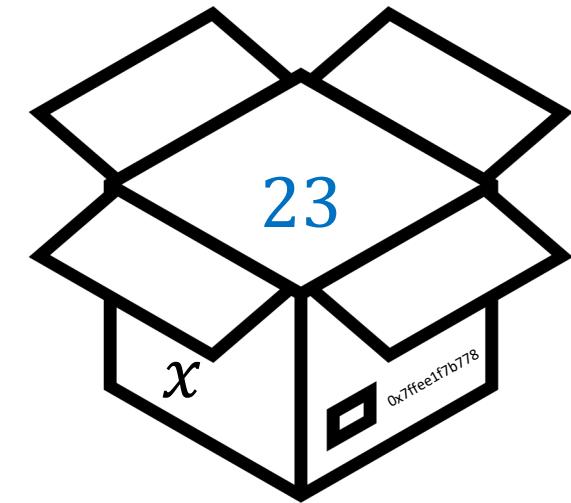
VGP131 – OOPII
Instructor: Ivaldo
Tributino



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

The image shows a code editor interface with a dark theme. On the left, there's a vertical toolbar with various icons. The main area displays a C++ file named 'main.cpp'. The code demonstrates the difference between lvalues and rvalues. It includes comments like 'lvalues and rvalues' and 'error: cannot take the address of an rvalue of type 'int''. A large blue speech bubble on the right side contains the explanatory text.

```
14
15     static int x = 23;
16
17 //A function that returns an lvalue.
18 int& getValue(){
19     return x;
20 }
21
22 void ref1(int& x){}
23
24 void ref2(const int& x){}
25
26 int main(){
27
28 //*****----- lvalues and rvalues -----
29 //      ----- lvalues and rvalues -----
30 //*****----- rvalues -----
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

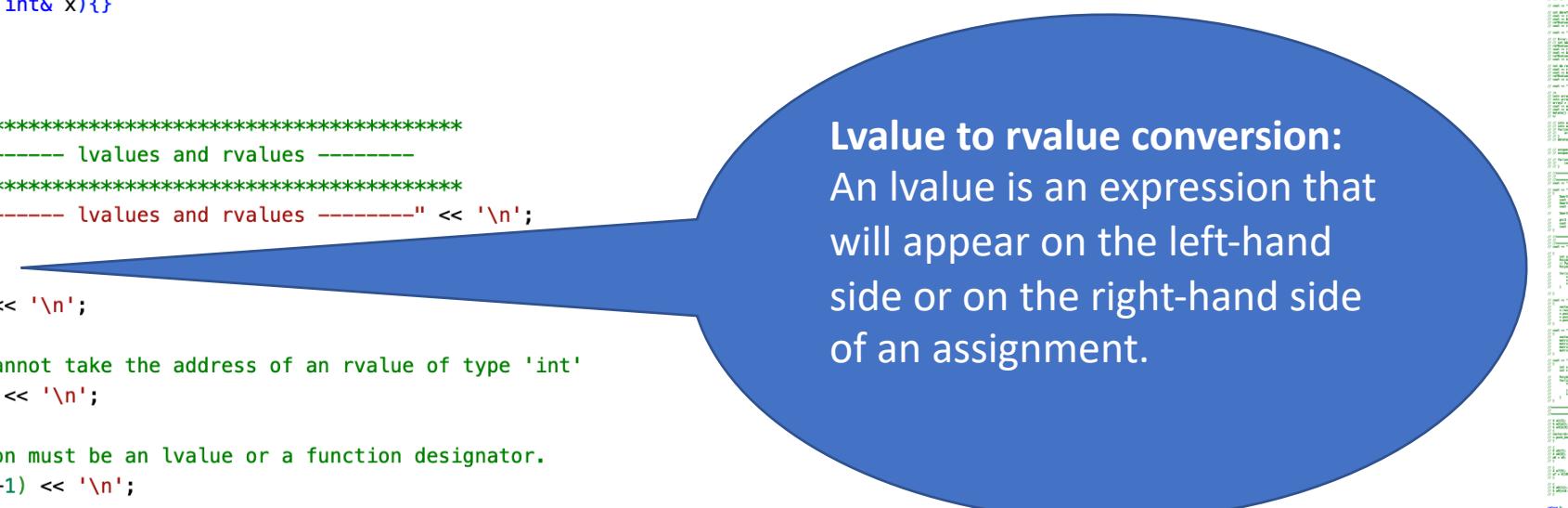
PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

bash + ×

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

Ln 47, Col 1 Spaces: 4 UTF-8 LF C++ Mac Live Share

LVALUE AND RVALUE



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.

```
14
15     static int x = 23;
16
17 //A function that returns an lvalue.
18 int& getValue(){
19     return x;
20 }
21
22 void ref1(int& x){}
23
24 void ref2(const int& x){}
25
26 int main(){
27
28 //*****----- lvalues and rvalues -----
29 //      ----- lvalues and rvalues -----
30 //*****----- lvalues and rvalues -----
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 << &x << '\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
```

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

bash + ×

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

Ln 47, Col 1 Spaces: 4 UTF-8 LF C++ Mac Live Share

LVALUE AND RVALUE

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. On the left, there's a vertical toolbar with various icons: a file icon, a magnifying glass, a person icon with '10K+', a play/pause icon, a square icon, a monitor icon, a gear icon, a GitHub icon, and a refresh/circular arrow icon.

The main area displays a C++ file named `main.cpp`. The code demonstrates the concept of lvalues and rvalues. It includes comments explaining the error of trying to take the address of an rvalue and the correct way to use the `&` operator to create a pointer to an lvalue.

```
24 void ret2(const int& x){}
25
26 int main(){
27
28 //***** lvalues and rvalues *****
29 //      ----- lvalues and rvalues -----
30 //***** lvalues and rvalues *****
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 << &x << '\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 = &x;
```

The terminal at the bottom shows the output of the code execution:

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

At the bottom, there are tabs for PROBLEMS (with 1), OUTPUT, TERMINAL (which is selected), and DEBUG CONSOLE. To the right of the terminal are icons for bash, a plus sign, a square, a trash can, an upward arrow, and an X.

LVALUE AND RVALUE

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 file named `main.cpp` with the following content:

```
35 // error: cannot take the address of an rvalue of type 'int'  
36 // cout << &x13 << '\n';  
37  
38 // Expression must be an lvalue or a function designator.  
39 // cout << &(x+1) << '\n';  
40  
41  
42 // Taking the memory address of x and setting it to y(pointer), using the & (ampersand) operator.  
43 int* y = &x;  
44 cout << y << '\n';  
45 cout << *y << '\n';  
46  
47 // Expression must be an lvalue or a function designator.  
48 // int* y = &x14;  
49  
50 cout << "- A function that returns an lvalue " << '\n';  
51  
52 cout << getLvalue() << '\n';  
53 cout << &getLvalue() << '\n';  
54  
55 getLvalue() = 100;  
56  
57 cout << getLvalue() << '\n';  
58 cout << &getLvalue() << '\n';  
59  
60 getLvalue()++; // Increment operator  
61 cout << getLvalue() << '\n';  
62  
63
```

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

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

A modal window is open in the bottom right corner, containing the following code:

```
static int x = 23;  
  
//A function that returns an lvalue.  
int& getLvalue(){  
    return x;  
}
```

The status bar at the bottom shows the file name `main.cpp`, line `Ln 68, Col 5`, and other settings like `Spaces: 4`, `UTF-8`, `LF`, `C++`, and `Mac`.

LVALUE AND RVALUE

The screenshot shows a C++ development environment with the following details:

- File Explorer:** Shows a single file named "main.cpp" with one error.
- Code Editor:** Displays the following C++ code:

```
66 cout << getValue() << '\n';
67 cout << &getValue() << '\n';
68
69 getValue()++; // Increment operator
70 cout << getValue() << '\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 }
```

A tooltip box highlights the error at line 77: "initial value of reference to non-const must be an lvalue C/C++(461)". It includes "View Problem" and "Quick Fix..." buttons.
- Terminal:** Shows the output of the program:

```
----- 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$
```
- Bottom Status Bar:** Shows the current file is "master*+", line 1, column 0, and the terminal is active.

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     initial value of reference to non-const must be an lvalue C/C++(461)
86
87     View Problem Quick Fix... (⌘.)
88
89
90
91
92
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

----- 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;  
}
```

LVALUE AND RVALUE

The screenshot shows a C++ development environment with a sidebar containing various icons. The main area displays a file named `main.cpp` with the following code:

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

A large blue callout bubble points from the line `int a = 10;` to the explanatory text in the center. The text reads:

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.

The terminal output shows the results of the code execution:

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

A code block in the bottom right corner shows the definition of the `passAllvalue` function:

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

The status bar at the bottom indicates the current file is `master*`, line 104, column 1, with 4 spaces, in UTF-8 encoding, for LF, C++, Mac, and Live Share.

LVALUE AND RVALUE

The screenshot shows a Visual Studio Code (VS Code) interface. On the left is a dark sidebar with various icons for file operations, search, and settings. The main area is a code editor with the following content:

```
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;           const int &ref
90 passAllvalue(a);    error: variable 'ref' declared const here
91 passAllvalue(10);
92 }
93
94
95 // error: variable 'r' View Problem Quick Fix... (⌘.)
96 const int& ref = 10;
97 cout << ref++ << endl;
98 }

99
100
101
102
103
104
105
106
```

A tooltip is displayed over the line `int a = 10;`, showing the text `const int &ref`. Below the code editor, the terminal window shows the output of the program:

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

The bottom status bar indicates the file is on line 31, column 14, with 4 spaces, in UTF-8 encoding, and the keyboard layout is set to Mac.

LVALUE AND RVALUE

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** On the left, showing files: main.cpp (1), smartPtr.h, and libraries.h.
- Code Editor:** The main area displays a C++ file named main.cpp. The code includes several cout statements and declarations. A tooltip is open over the line "cout << ref++ << '\n';".
 - Text in tooltip:** "const int &ref
expression must be a modifiable lvalue C/C++(137)"
 - Actions:** "View Problem" and "Quick Fix..." (with a gear icon).
- Terminal:** At the bottom, showing command-line output:

```
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$
```
- Bottom Status Bar:** Shows file name "master*+", line "Ln 81, Col 5", and encoding "UTF-8".

RVALUE REFERENCES

The screenshot shows a C++ development environment with the following details:

- File:** main.cpp
- Code Snippet:** A portion of main.cpp showing the use of rvalue references. It includes declarations like `const int& ref = 10;` and `int &&refRvalue = 101;`, and function calls like `passAllvalue(a);` and `passAllvalue(10);`.
- Output Terminal:** Shows the output of the program, which includes the string "- T&& (double ampersand)" followed by memory addresses (0x7ffee84bd76c, 101, 15) and the command `(base) Ivaldo:Week7 admin\$`.
- Callout:** A large blue callout bubble points from the text `int &&refRvalue = 101;` to the explanatory text below.
- Explanatory Text:** Inside the callout bubble, the text reads: "It can bind to an rvalue like a temporary without having to be const."
- Code Block:** A highlighted code block in the bottom right corner shows the definition of the `passRvalue` function:void passRvalue(int&& x){
 cout << ++x << endl;
}

RVALUE REFERENCES

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a tree view with a file named "main.cpp" and a folder named "Week7".
- Search Bar:** Contains the text "main.cpp > main()".
- Code Editor:** Displays the "main.cpp" file content. The code demonstrates rvalue references with three lambda functions: `f_Lvalue`, `f_Allvalue`, and `f_Rvalue`. The code outputs "square" and "Polygon was destructive" to the terminal.
- Terminal:** Shows the output of the program:

```
Constructor Invoked
square
square
Constructor Invoked
square
Polygon was destructive
Constructor Invoked
square
Polygon was destructive
Polygon was destructive
(base) Ivaldo:Week7 admin$
```
- Bottom Status Bar:** Shows the file path "master*+", line count "0 △ 0", and "Live Share".
- Bottom Right:** Includes icons for bash, terminal tabs, and file operations.

STD :: MOVE

The screenshot shows a Visual Studio Code (VS Code) interface. On the left is the sidebar with various icons. The main area displays a C++ file named `main.cpp`. The code demonstrates different ways to pass objects to a function and how move semantics are applied. The code is as follows:

```
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
```

The code uses four different lambda functions to demonstrate lvalue references, const lvalue references, rvalue references, and move semantics. The output window at the bottom shows the results of running the code, which includes output from the constructor and destructor of the `Polygon` class.

Bottom navigation bar:

- PROBLEMS
- OUTPUT
- TERMINAL
- DEBUG CONSOLE

Bottom right corner:

- bash
- + (New Terminal)
- (Close Terminal)
- ^ (Switch Terminal)
- × (Close All Terminals)

Bottom status bar:

- Ln 130, Col 1
- Spaces: 4
- UTF-8
- LF
- C++
- Mac

Bottom left status bar:

- master*+ (File Name)
- 0 △ 0 (File Status)
- Live Share (Icon)

STD :: MOVE

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** On the left sidebar, there is a tree view showing a single file named "main.cpp".
- Search:** A magnifying glass icon is present in the sidebar.
- 10K+:** A circular icon with the number "10K+" is in the sidebar.
- Icons:** A row of icons includes a triangle, a square, a rectangle, a list, a gear, a GitHub logo, and a refresh symbol.
- Code Editor:** The main area displays the following C++ code from "main.cpp":

```
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
156
```

The code uses `std::move` to move the value of variable `x` to `refRValue1`. The output of the program is shown in the Terminal tab:

```
- std::move
0x7fee250476c
0x10d700140
101
101
0x10d700140
0x10d700140
101
101
(base) Ivaldo:Week7 admin$
```

The terminal also shows a count of 101 occurrences of the string "101".

At the bottom, the status bar indicates: Ln 153, Col 1 | Spaces: 4 | UTF-8 | LF | C++ | Mac | Live Share

Two types of references^[2]

In C++ there are two types of references:

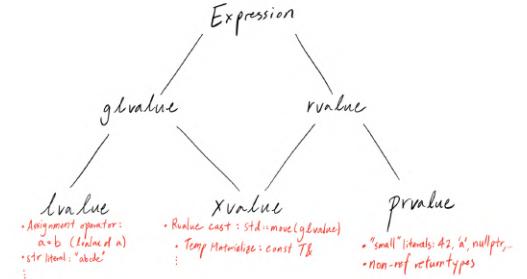
Ivalue reference:

- 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;

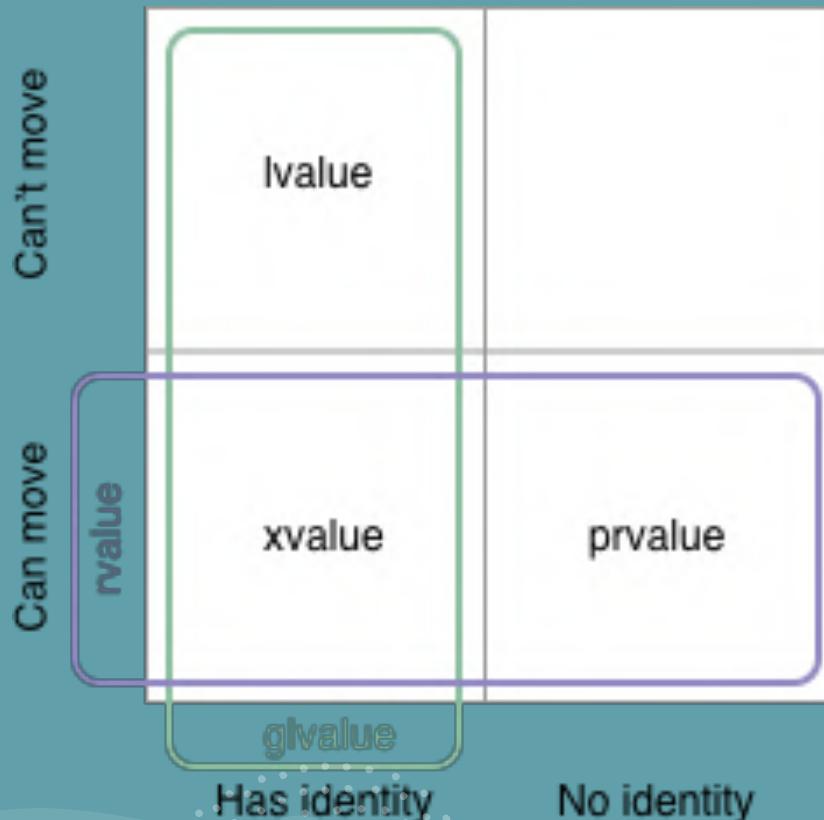
xvalue

- (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

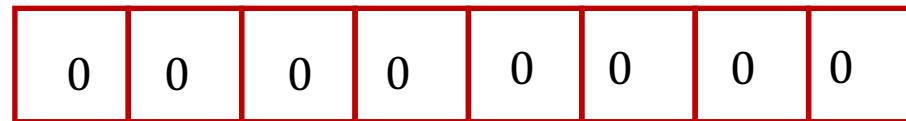
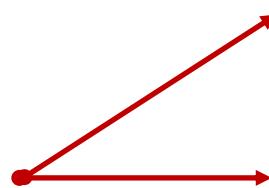


- The functions that accept rvalue reference parameters (including move constructors, move assignment operators, and regular member functions such as `std::vector::push_back`) are selected, by overload resolution, when called with rvalue arguments.
- Names of rvalue reference variables are lvalues and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

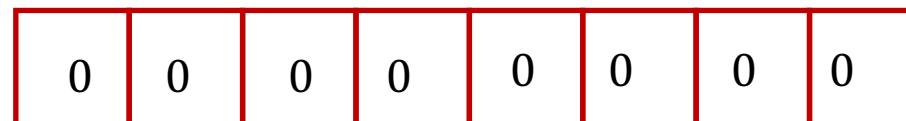


```
int* array1 = new int[8]{1,2,3,4,5,6,7,8};  
int* array2 = new int[8];  
array2 = array1; // Memory leak  
cout << array1 << '\n';  
cout << array2 << '\n';  
delete[] array1; // array2 dangling pointer
```

array1



array2



```
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



```
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

The screenshot shows a Visual Studio Code (VS Code) interface with a dark theme. On the left is a vertical sidebar with various icons: file, search, 10K+, play, folder, monitor, settings, GitHub, and refresh. The main area displays a C++ file named `main.cpp`. The code demonstrates the use of `std::unique_ptr` and move semantics. It includes examples of raw pointer assignments leading to memory leaks, and the correct use of `std::move` to transfer ownership between `unique_ptr` objects.

```
148
149
150     cout << "-std::unique_ptrs" << '\n';
151
152     /*
153     int* array1 = new int[8]{1,2,3,4,5,6,7,8};
154     int* array2 = new int[8];
155     array2 = array1; // Memory leak
156     cout << array1 << '\n';
157     cout << array2 << '\n';
158     delete[] array1; // array2 dangling pointer
159     */
160
161     int* array1 = new int[8]{1,2,3,4,5,6,7,8};
162     int* array2 = new int[8];
163     for(int i = 0; i<8; ++i){
164         array2[i] = array1[i];
165     }
166     delete[] array1;
167
168
169     unique_ptr<int[]> ptr1(new int[8]{1,2,3,4,5,6,7,8});
170     unique_ptr<int[]> ptr2 = std::move(ptr1); // memory resource is transferred to another unique_ptr
171
172     for(unsigned i=0; i<8;++i){
173         cout << ptr2[i] << "\n";
174     }
175
176
177
```

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

```
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 dereferencing 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

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows two files: `main.cpp` (1 file) and `smartPtr.h`.
- Sidebar:** Includes icons for search, 10K+, copy, paste, settings, GitHub, and refresh.
- Code Editor:** Displays the `main.cpp` file with code demonstrating move semantics. The code creates two `Polygon` objects, swaps them using `std::move`, and prints their names and addresses.
- Terminal:** Shows the output of the program:

```
- 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$
```

A blue arrow points from the terminal output to the line "Move Assignment operator invoked" in the code editor.
- Bottom Status Bar:** Shows the current file is `master*+`, line 200, column 1, with 4 spaces, in UTF-8 encoding, and the keyboard layout is set to 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);

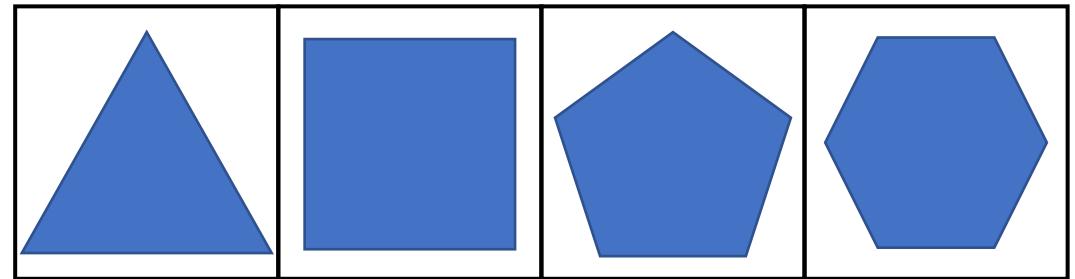
    int getSize();
    void setSize(unsigned size);
    ~PolyArray() = default;
};
```

POLYARRAY CLASS

`unique_ptr<Polygon[]>`



`new Polygon[4]`



COPY CONSTRUCTOR

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

- File Explorer:** Shows files: main.cpp, polyArray.h, polyArray.cpp, Polygon.cpp, smartPtr.h, and libraries.h.
- Code Editor:** The main.cpp file is open, showing code related to PolyArray and Polygon classes. A comment block for PolyArray starts at line 147. The code includes a loop from i=0 to size-1 printing shape names, side counts, and areas.
- Terminal:** The terminal window displays the following output:

```
----- PolyArray -----
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
```
- Bottom Status Bar:** Shows the file is master*, 0 changes, 0 errors, and 0 warnings. It also shows "Live Share".
- Bottom Right:** Includes icons for bash, terminal tabs, and file operations.

MOVE CONSTRUCTOR



The screenshot shows a code editor interface with several tabs at the top: main.cpp, polyArray.h, polyArray.cpp, Polygon.cpp, smartPtr.h, and libraries.h. The main.cpp tab is active, displaying C++ code. The code includes a move constructor for a PolyArray class and a loop that prints information about shapes. The terminal below shows the execution of the program, outputting five instances of the default constructor and then the move constructor followed by shape details.

```
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
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
(base) Ivaldo:Week7 admin$ 
```

Ln 164, Col 6 Spaces: 4 UTF-8 LF C++ Mac Live Share



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



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

```
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 }
```

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

```
- 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$
```

The bottom status bar indicates the current file is master*, the line number is 179, column 1, with 4 spaces, and the encoding is UTF-8.

PUSH BACK

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

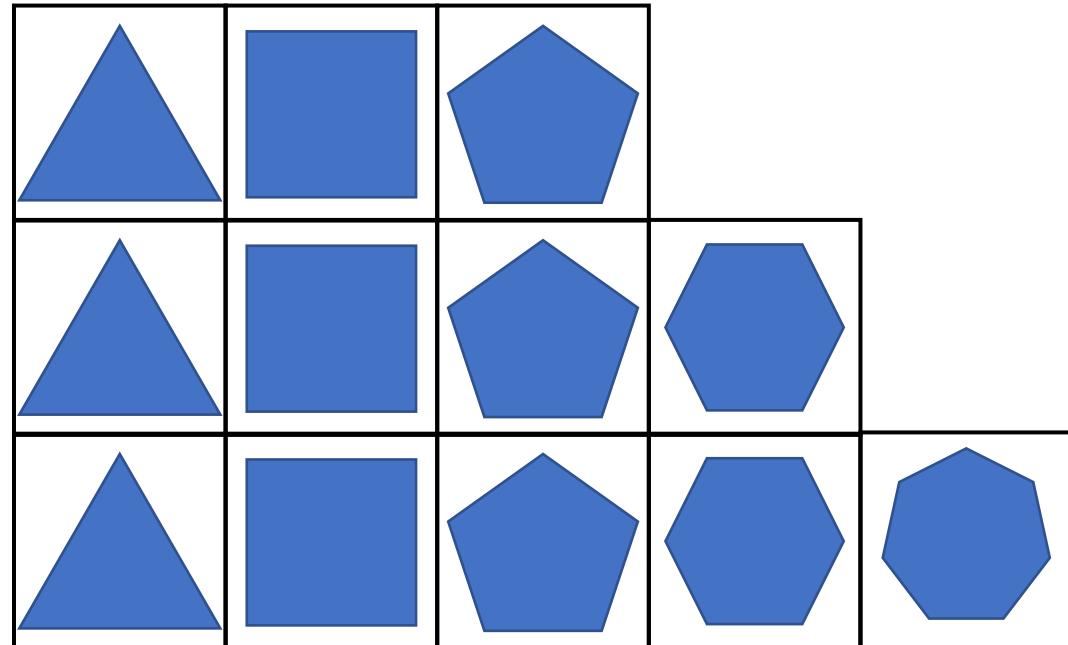
```
main.cpp > main()
170     v.push_back(Polygon(3));
171     v.push_back(Polygon(4));
172     v.push_back(Polygon(5));
173 }
174
175 cout << "- Push_back in vector<PolyArray>" << '\n';
176 {
177     vector<PolyArray> matrix;
178     matrix.reserve(3); //only allocation
179     matrix.push_back(PolyArray(3));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

The code demonstrates pushing three polygons (triangle, square, pentagon) into a vector and then pushing them into a larger matrix vector.

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
Move constructor in PolyArray Invoked
Polygon was destructive
(base) Ivaldo:Week7 admin$
```



MATRIX

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

- Top Bar:** Shows file tabs for main.cpp, smartPtr.h, classA.h, Source.cpp, polyArray.h, and polyArray.cpp.
- Sidebar:** Includes icons for file operations like copy, paste, search, and refresh, along with a "10K+" badge.
- Code Area:** Displays the main.cpp file content. The code creates a vector of PolyArray objects and prints them to the console.
- Terminal:** Shows the output of the program execution, which includes multiple instances of the constructor being called and the resulting matrix values.
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, TERMINAL (which is selected), and DEBUG CONSOLE. It also features a terminal icon with a "bash" label and various terminal control icons.
- Status Bar:** Shows the current file path as master*, line count as 224, column count as 9, and character count as 243 selected.

```
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 }
```

```
- PolyMatrix
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
(base) Ivaldo:Week7 admin$
```

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

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