LASALLE COLLEGE VANCOUVER

Value categories Move Semantics

VGP 131 - Object Oriented Programming in C++ II

Instructor: Ivaldo Tributino

May 16, 2022

ASSIGNMENT INSTRUCTION

- The exam must be submitted by Feb 20, 2022.
- Each problem presents its own score, the sum of all scores is 100.

Student's Number:
Student's Name:

(10 POINTS) PROBLEM 1

Create class **A** and add the following to it:

- A private member _number which is a pointer to an int.
- A public constructor that:
  1. Accepts an int as input argument
  2. Allocates an int on the heap and sets its value according to the input argument
  3. Points _number to the newly allocated int
  4. Prints to standard out what int value it has been initialized with
- A destructor that:
  1. Prints what value _number points to before it frees the associated memory
  2. Prints nullptr if _number does not point to an int

```
A a1{3} //Given this code:

//class A shall generate the following output:
```

class A constructor: 3
class A destructor: 3

(10 POINTS) PROBLEM 2

Add a copy constructor to class **A** so that it can handle the following code. Print out a line to indicate that the copy constructor has been used.

```
A a1{3};
A a2{a1};
A a3{A{4}};

//The output shall be:
```

class A constructor: 3
class A copy constructor: 3
class A constructor: 4
class A destructor: 4
class A destructor: 3
class A destructor: 3

Please note that the copy constructor was not used when creating a3 because of compiler optimization.

(15 POINTS) PROBLEM 3

Add a move constructor to class **A**. Print out a line to indicate that the move constructor has been used. The following code:

```
vector<A> v;
v.push_back(A{5});

//Should generate this output
```

class A constructor: 5
class A move constructor: 5
class A destructor: nullptr
class A destructor: 5

(10 POINTS) PROBLEM 4

Add a copy assignment operator to class **A**. Print out a line to indicate that the copy assignment operator has been used.

```
//The following code:
A a5{7};
A a6{8};
a6 = a5;
//Shall generate this output:
```

class A constructor: 7
class A constructor: 8
class A copy assignment operator: 7
class A destructor: 7
class A destructor: 7

(15 Points) Problem 5

Add a move assignment operator to class **A**. Print out a line to indicate that the move assignment operator has been used.

```
//The following code:
A a7{9};
a7 = A{10};
//Shall generate this output:
```

class A constructor: 9
class A constructor: 10
class A move assignment operator: 10
class A destructor: nullptr
class A destructor: 10

(10 Points) Problem 6

Use the move function to force move semantics to be used when creating a new instance of class **A**.

```
//The following code:
A a8{11};
A a9{std::move(a8)};
//Shall generate this output:
```

class A constructor: 11
class A move constructor: 11
class A destructor: 11
class A destructor: nullptr

(5 Points) Problem 7

Does the following code compile successfully?

```
template<typename T>
void print(T&& x){
    cout << x << endl;
```

```
}

int main(){

    int x = 10;
    print(10);
    print(x);
    print(std::move(x));

    return 0;
}
```

(10 POINTS) PROBLEM 8

Rvalue references are the glue that ties move semantics and perfect forwarding. Perfect forwarding what is it all about?

(5 POINTS) PROBLEM 9

What is the output of the following C++ program?

```
#include <utility>      // std::forward
#include <iostream>

void overloaded (const int& x) {std::cout << "[lvalue]" << endl;}
void overloaded (int&& x) {std::cout << "[rvalue]" << endl;}

template <class T> void fn (T&& x) {
  overloaded (x);
  overloaded (std::forward<T>(x));
}

int main () {

    int k;
    fn (k);
    fn (0);
    fn (std::move(k));

   return 0;
}
```

(10 POINTS) PROBLEM 10

What is the output of the following C++ program?

```
struct Y
{
  Y(){}
  Y(const Y &){ std::cout << "Y␣Copy␣constructor\n"; }
  Y(Y &&) noexcept { std::cout << "Y␣Move␣constructor\n"; }
};

struct X
{
  Y a_;
  Y b_;

  template<typename A, typename B>
  X(A && a, B && b) :a_{std::forward<A>(a)},b_{std::forward<B>(b)}
  {
    std::cout << "X␣Constructor\n";
  }
};

template<typename A, typename B>
X factory(A && a, B && b)
{
  return X(std::forward<A>(a), std::forward<B>(b));
};

int main(){
    Y y;
    X two = factory(y, Y());

    return 0;
}
```