
# High Level Programming

## Exercises on C++ Libraries

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

# Exercise

- What does the keyword "const" do?
  1. It allows to keep search time within a container nearly constant
  2. It prevents objects to be casted
  3. It allows to keep execution time of a member function nearly constant
  4. It prevents variables to be copied (i.e., assigned) to other variables
  5. It prevents objects from being mutated

# Solution

- What does the keyword "const" do?
  1. It allows to keep search time within a container nearly constant
  2. It prevents objects to be casted
  3. It allows to keep execution time of a member function nearly constant
  4. It prevents variables to be copied (i.e., assigned) to other variables
  5. It prevents objects from being mutated

# Exercise

- Which is the thing in common between passing parameters by address and by reference?
  - They both create a copy of the passed object
  - In both cases, deferencing the operator is necessary to access data
  - They both call a move constructor
  - In both cases it is possible to modify the data
  - They both free the resources of the passed object

# Solution

- Which is the thing in common between passing parameters by address and by reference?
  - They both create a copy of the passed object
  - In both cases, deferencing the operator is necessary to access data
  - They both call a move constructor
  - In both cases it is possible to modify the data
  - They both free the resources of the passed object

# Exercise

- Analyze the following program
- Which is the output generated?

```
#include <functional>
#include <iostream>
using namespace std;
int main() {
  int i = 3;
  int j = 5;
  function<int (void)>
    f = [&i, j] { return i + j; };
  i = 22;
  j = 44;
  cout << f() << endl;
}
```

# Solution

- Analyze the following program
- Which is the output generated?

```
#include <functional>
#include <iostream>
using namespace std;
int main() {
  int i = 3;
  int j = 5;
  function<int (void)>
    f = [&i, j] { return i + j; };
  i = 22;
  j = 44;
  cout << f() << endl;
}
```

```
27
```

# Exercise

- Analyze the following program
- Which is the final value of the variable b?

```
#include <iostream>

int a = 10;

int main() {
  int& b = a, a;
  a = 0;
  b++;
}
```

# Solution

- Analyze the following program
- Which is the final value of the variable b?

```
#include <iostream>

int a = 10;

int main() {
  int& b = a, a;
  a = 0;
  b++;
}
```

```
11
```

# Exercise

- Analyze the following program
- Which is the final value of the variable b?

```
#include <iostream>

int main() {
  int a = 10;
  int& b = a;
  auto f = [=](){return a+b;};

  a = 0;
  b+=f();
  return 1;
}
```

# Solution

- Analyze the following program
- Which is the final value of the variable b?

```
#include <iostream>

int main() {
  int a = 10;
  int& b = a;
  auto f = [=](){return a+b;};

  a = 0;
  b+=f();
  return 1;
}
```

```
20
```

# Exercise

- The RAII paradigm guarantees that ...
  - The resources are deleted before initialization and then new ones allocated
  - The resources are copied at object initialization so that objects can have independent lives
  - The resources are acquired at object initialization and released when its lifetime ends
  - The resources are allocated when a new operator is used
  - The resources are copied when objects are passed as parameters, to keep original data safe

# Solution

- The RAII paradigm guarantees that ...
  - The resources are deleted before initialization and then new ones allocated
  - The resources are copied at object initialization so that objects can have independent lives.
  - The resources are acquired at object initialization and released when its lifetime ends
  - The resources are allocated when a new operator is used
  - The resources are copied when objects are passed as parameters, to keep original data safe

# Exercise

- Analyze the following program
- When (and why) the default constructor and assignment operators are called

# Exercise

```
class Y {
private: int i;
public:
  Y () { ... }                          // Constructor
  ~Y() { ... }                          // Destructor
  Y (const Y &n) { ... }                // Copy Constructor
  Y &operator=(const Y &n) {            // Copy Assignement Operator
    ... return *this;
  }
  Y (Y&& n) noexcept { ... }            // Move Constructor
  Y &operator=(Y&&n) noexcept {         // Move Assignment Operator
    ... return *this;
  }
  void set(int n) {i = n;};
  int get () {return i;}
};
```

# Exercise

```
void f1(Y y) { y.set(5); }
void f2(Y &y) { int n = y.get(); }

   int main() {
1.   Y y1;
2.   Y y2=y1;
3.   f1(y1);
4.   f1(std::move(y1));
5.   return 0;
   }
```

# Exercise

```
void f1(Y y) { y.set(5); }
void f2(Y &y) { int n = y.get(); }

   int main() {
1.   Y y1;
2.   Y y2=y1;
3.   f1(y1);
4.   f1(std::move(y1));
5.   return 0;
   }
```

```
{1} [Constructor]
{2} [Copy Constructor]
{3} [Copy Constructor]
{f1}[Destructor]
{4} [Move Constructor]
{f1}[Destructor]
{5} [Destructor]
    [Destructor]
```

# Exercise

- Analyze the following program
- When (and why) the default constructor and assignment operators are called

# Exercise

```
class Y {
private: int i;
public:
  Y () { ... }                          // Constructor
  ~Y() { ... }                          // Destructor
  Y (const Y &n) { ... }                // Copy Constructor
  Y &operator=(const Y &n) {            // Copy Assignement Operator
    ... return *this;
  }
  Y (Y&& n) noexcept { ... }            // Move Constructor
  Y &operator=(Y&&n) noexcept {         // Move Assignment Operator
    ... return *this;
  }
  void set(int n) {i = n;};
  int get () {return i;}
};
```

# Exercise

```
void f1(Y y) { }
Y f2(Y &y) { Y ay; return ay; }
void f3(Y y1, Y &y2){ }

   int main() {
1.   Y y1, y2, y3;
2.   y1=y2;
3.   f3(y1, y3);
4.   Y y4 = f2(y1);
5.   return 0;
   }
```

# Exercise

```
void f1(Y y) { }
Y f2(Y &y) { Y ay; return ay; }
void f3(Y y1, Y &y2){ }

   int main() {
1.    Y y1, y2, y3;
2.    y1=y2;
3.    f3(y1, y3);
4.    Y y4 = f2(y1);
5.    return 0;
   }
```

```
{1} [Constructor]
    [Constructor]
    [Constructor]
{2} [Copy Assignment Op]
{3} [Copy Constructor]
{f3}[Destructor]
{f2}[Constructor]
{5} [Destructor]
    [Destructor]
    [Destructor]
    [Destructor]
```

# Exercise

- Analyze the following program
- When (and why) the default constructor and assignment operators are called

# Exercise

```
class Y {
private: int i;
public:
  Y () { ... }                          // Constructor
  ~Y() { ... }                          // Destructor
  Y (const Y &n) { ... }                // Copy Constructor
  Y &operator=(const Y &n) {            // Copy Assignement Operator
    ... return *this;
  }
  Y (Y&& n) noexcept { ... }            // Move Constructor
  Y &operator=(Y&&n) noexcept {         // Move Assignment Operator
    ... return *this;
  }
  void set(int n) {i = n;};
  int get () {return i;}
};
```

# Exercise

```
void f1(Y y) { }
void f2(Y &y) { }

   int main() {
1.    Y y1;
2.    f1(y1);
3.    f2(y1);
4.    Y *y2 = new Y;
5.    Y y3;
6.    y3 = (std::move(y1));
7.    return 0;
}
```

# Exercise

```
void f1(Y y) { }
void f2(Y &y) { }

   int main() {
1.    Y y1;
2.    f1(y1);
3.    f2(y1);
4.    Y *y2 = new Y;
5.    Y y3;
6.    y3 = (std::move(y1));
7.    return 0;
}
```

```
{1} [Constructor]
{2} [Copy Constructor]
{f1}[Destructor]
{4} [Constructor]
{5} [Constructor]
{6} [Move Assigment Op]
{7} [Destructor]
    [Destructor]
```

# Exercise

- Analyze the following program
- Which is the output generated?

```
#include <iostream>
#include <vector>
class A {
public:
  ~A() { std::cout << "*"; }
  A() { std::cout << "a"; }
  A(const A&) { std::cout << "&"; }
};
int main() {
  int i=3;
  std::vector<A> v(i);
}
```

# Solution

- Analyze the following program
- Which is the output generated?

```
#include <iostream>
#include <vector>
class A {
public:
  ~A() { std::cout << "*"; }
  A() { std::cout << "a"; }
  A(const A&) { std::cout << "&"; }
};
int main() {
  int i=3;
  std::vector<A> v(i);
}
```

```
a***
```