Day03

- Lab: Improvement and Pitfalls
- Object Lifetime Part2
- References
- More on classes

Map: Full Solution

```
int main(int argc, char *argv[]) {
  string file_name(argv[1]);

  map<Player, uint32_t, Player::Compare > map_of_players;

  std::ifstream fin(file_name, std::ios::in);
  string line;

  // read player information from file

  uint32_t lineno = 0;
  while (std::getline(fin, line)) {
    ++lineno;
    auto result = map_of_players.emplace(std::make_pair(line, lineno));
    if (not result.second) {
      auto previous_lineno = (*(result.first)).second;
      std::cout << "Error, duplicate at line " << lineno
                << ", previously found at line " << previous_lineno << std::endl;
    }
  }
  ...
```

emplace returns a pair (iterator, Boolean)

Map iterator data is a pair (K, V)

Beware that element is created before detection

```
shell> map.exe data_duplicate.txt
Destroying  Noylan
Error, duplicate at line 8,
previously found at line 4
```

Beware of std::map

```
for(;;) {
  std::cout << "size of map is "
            << map_id2values.size()
            << std::endl;
  std::cout << "query> ";

  std::cin >> qin;

  if(qin == "end" || qin == "END") {
    break;
  }
  if(map_id2values[qin]) {
    std::cout << "value[" << qin << "] = "
              << map_id2values[qin]
              << std::endl;
  } else {
    std::cout << "This ID does not exists"
              << std::endl;
  }
}
```

```
size of map is 9995
query> 98fd6d660451
value[98fd6d660451] = 5033.93

size of map is 9995
query> 84fd1c80659c
value[84fd1c80659c] = 863.93

size of map is 9995
query> wrongid
This ID does not exist

size of map is 9996
query> END
```

Beware of std::map

std::map<Key,T,Compare,Allocator>::operator[]

```
T& operator[]( const Key& key );      (1)
T& operator[]( Key&& key );           (2)   (since C++11)
```

Returns a reference to the value that is mapped to a key equivalent to key, performing an insertion if such key does not already exist.

1) Inserts `value_type(key, T())` if the key does not exist. This function is equivalent to
`return insert(std::make_pair(key, T())).first->second;`
- key_type must meet the requirements of *CopyConstructible*.
- mapped_type must meet the requirements of *CopyConstructible* and *DefaultConstructible*.

(until C++11)

If an insertion is performed, the mapped value is value-initialized (default-constructed for class types, zero-initialized otherwise) and a reference to it is returned.

Another Experiment (1)

```
struct Player {
  string last_name_;
  vector<string> names_;
  double score_;
  Player(const string &line) { ... }
  ~Player() {
    std::cout << "Destroying " << last_name_ << std::endl;
  }
};
int main(int argc, char *argv[]) {
  string file_name(argv[1]);
  vector<Player> players;
  players.reserve(100);
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    players.emplace_back(line);
  }
  std::cout << "TRACE: before sort" << std::endl;
  std::sort(players.begin(), players.end(),
    [](const Player &a, const Player &b) -> bool  {
      return a.score_ > b.score_;
    });
  std::cout << "TRACE: after sort" << std::endl;
  int idx = 0;
  print_table_header();
  ...
}
```

What will be the output ?

Another Experiment (1)

```
struct Player {
  string last_name_;
  vector<string> names_;
  double score_;
  Player(const string &line) { ... }
  ~Player() {
    std::cout << "Destroying " << last_name_ << std::endl;
  }
};
int main(int argc, char *argv[]) {
  string file_name(argv[1]);
  vector<Player> players;
  players.reserve(100);
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    players.emplace_back(line);
  }
  std::cout << "TRACE: before sort" << std::endl;
  std::sort(players.begin(), players.end(),
    [](const Player &a, const Player &b) -> bool  {
      return a.score_ > b.score_;
    });
  std::cout << "TRACE: after sort" << std::endl;
  int idx = 0;
  print_table_header();
  ...
}
```

```
shell> sorted_names data.txt
TRACE: before sort

TRACE: after sort
+------+----------+-----------+-----------+-----------+-----------+
| Rank |    Score | Last Name | 1st Name  | 2nd Name  | 3rd Name  |
+------+----------+-----------+-----------+-----------+-----------+
|    1 | 17301.72 | Hartman   | Rosalie   | Carrie    |           |
|    2 |  2815.77 | Rubio     | Alfonso   | Ulysses   | Vito      |
|    3 |  2638.90 | Irwin     | Mara      | Elena     | Etta      |
|    4 |  2615.93 | Smith     | Linda     | Fay       |           |
|    5 |  1990.52 | Davenport | Darin     | Graham    | Gale      |
|    6 |  1321.13 | Faulkner  | Enrique   | Emmanuel  | Emilio    |
|    7 |  1181.31 | Wong      | Otis      | Cornell   | Gary      |
|    8 |   863.93 | Romero    | Georgia   | Tania     |           |
|    9 |   812.47 | Hanna     | Thelma    | Corine    | Juliet    |
|   10 |   455.36 | Nolan     | Marianne  | Jenna     |           |
+------+----------+-----------+-----------+-----------+-----------+
Destroying  Smith
Destroying  Romero
Destroying  Davenport
Destroying  Rubio
Destroying  Wong
Destroying  Faulkner
Destroying  Nolan
Destroying  Hanna
Destroying  Irwin
Destroying  Hartman
```

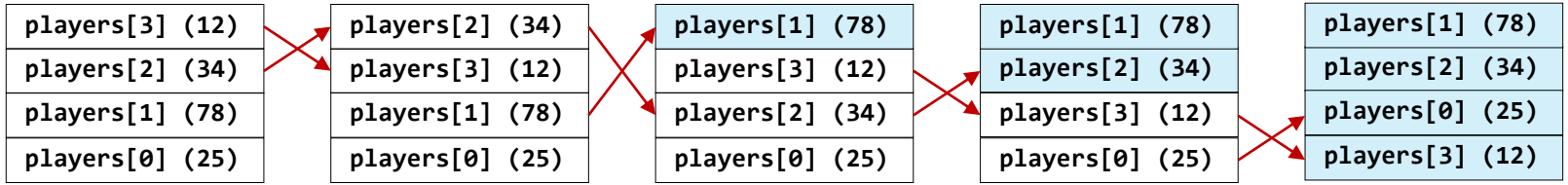
What will be the output ?

Another Experiment (2)

```cpp
struct Player {
  string last_name_;
  vector<string> names_;
  double score_;
  Player(const string &line) { ... }
  ~Player() {
    std::cout << "Destroying " << last_name_ << std::endl;
  }
};
int main(int argc, char *argv[]) {
  string file_name(argv[1]);
  vector<Player> players;
  players.reserve(100);
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    players.emplace_back(line);
  }
  std::cout << "TRACE: before sort" << std::endl;
  std::sort(players.begin(), players.end(),
    [](const Player &a, const Player &b) -> bool  {
      return a.score_ > b.score_;
    });
  std::cout << "TRACE: after sort" << std::endl;
  int idx = 0;
  print_table_header();
  ...
}
```

```
shell> sorted_names data.txt
TRACE: before sort
Destroying Romero
Destroying Davenport
Destroying Rubio
Destroying Wong
Destroying Faulkner
Destroying Nolan
Destroying Hanna
Destroying Irwin
Destroying Hartman
TRACE: after sort
+------+----------+-----------+-----------+-----------+-----------+
| Rank |    Score | Last Name | 1st Name  | 2nd Name  | 3rd Name  |
+------+----------+-----------+-----------+-----------+-----------+
|    1 | 17301.72 | Hartman   | Rosalie   | Carrie    |           |
|    2 |  2815.77 | Rubio     | Alfonso   | Ulysses   | Vito      |
|    3 |  2638.90 | Irwin     | Mara      | Elena     | Etta      |
|    4 |  2615.93 | Smith     | Linda     | Fay       |           |
|    5 |  1990.52 | Davenport | Darin     | Graham    | Gale      |
|    6 |  1321.13 | Faulkner  | Enrique   | Emmanuel  | Emilio    |
|    7 |  1181.31 | Wong      | Otis      | Cornell   | Gary      |
|    8 |   863.93 | Romero    | Georgia   | Tania     |           |
|    9 |   812.47 | Hanna     | Thelma    | Corine    | Juliet    |
|   10 |   455.36 | Nolan     | Marianne  | Jenna     |           |
+------+----------+-----------+-----------+-----------+-----------+
Destroying Hartman
Destroying Rubio
Destroying Irwin
Destroying Smith
Destroying Davenport
Destroying Faulkner
Destroying Wong
Destroying Romero
Destroying Hanna
Destroying Nolan
```

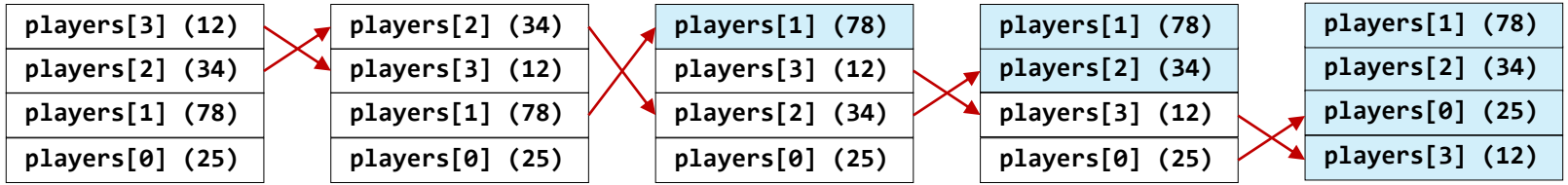
Sort (1)



What is  ?

Implementation ?

Sort (2)




```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = a;
  a = b;
  b = tmp;
}
```

Sort (3)

```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = a;
  a = b;
  b = tmp;
}
```

```
shell> sorted_names data.txt
TRACE: before sort
Destroying Romero
Destroying Davenport
Destroying Rubio
Destroying Wong
Destroying Faulkner
Destroying Nolan
Destroying Hanna
Destroying Irwin
Destroying Hartman
TRACE: after sort
+------+----------+-----------+-----------+-----------+-----------+
| Rank |    Score | Last Name | 1st Name  | 2nd Name  | 3rd Name  |
+------+----------+-----------+-----------+-----------+-----------+
|    1 | 17301.72 | Hartman   | Rosalie   | Carrie    |           |
|    2 |  2815.77 | Rubio     | Alfonso   | Ulysses   | Vito      |
|    3 |  2638.90 | Irwin     | Mara      | Elena     | Etta      |
|    4 |  2615.93 | Smith     | Linda     | Fay       |           |
|    5 |  1990.52 | Davenport | Darin     | Graham    | Gale      |
|    6 |  1321.13 | Faulkner  | Enrique   | Emmanuel  | Emilio    |
|    7 |  1181.31 | Wong      | Otis      | Cornell   | Gary      |
|    8 |   863.93 | Romero    | Georgia   | Tania     |           |
|    9 |   812.47 | Hanna     | Thelma    | Corine    | Juliet    |
|   10 |   455.36 | Nolan     | Marianne  | Jenna     |           |
+------+----------+-----------+-----------+-----------+-----------+
Destroying Hartman
Destroying Rubio
Destroying Irwin
Destroying Smith
Destroying Davenport
Destroying Faulkner
Destroying Wong
Destroying Romero
Destroying Hanna
Destroying Nolan
```

Sort (4)

```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = a;
  a = b;
  b = tmp;
}
```

```
shell> sorted_names data.txt
TRACE: before sort
Destroying Romero
Destroying Davenport
Destroying Rubio
Destroying Wong
Destroying Faulkner
Destroying Nolan
Destroying Hanna
Destroying Irwin
Destroying Hartman
TRACE: after sort
+------+----------+-----------+-----------+-----------+-----------+
| Rank |    Score | Last Name | 1st Name  | 2nd Name  | 3rd Name  |
+------+----------+-----------+-----------+-----------+-----------+
|    1 | 17301.72 | Hartman   | Rosalie   | Carrie    |           |
|    2 |  2815.77 | Rubio     | Alfonso   | Ulysses   | Vito      |
|    3 |  2638.90 | Irwin     | Mara      | Elena     | Etta      |
|    4 |  2615.93 | Smith     | Linda     | Fay       |           |
|    5 |  1990.52 | Davenport | Darin     | Graham    | Gale      |
|    6 |  1321.13 | Faulkner  | Enrique   | Emmanuel  | Emilio    |
|    7 |  1181.31 | Wong      | Otis      | Cornell   | Gary      |
|    8 |   863.93 | Romero    | Georgia   | Tania     |           |
|    9 |   812.47 | Hanna     | Thelma    | Corine    | Juliet    |
|   10 |   455.36 | Nolan     | Marianne  | Jenna     |           |
+------+----------+-----------+-----------+-----------+-----------+
Destroying Hartman
Destroying Rubio
Destroying Irwin
Destroying Smith
Destroying Davenport
Destroying Faulkner
Destroying Wong
Destroying Romero
Destroying Hanna
Destroying Nolan
```

C++ vs. Other Languages

```python
def main():
    players = []

    p1 = Player('Turing, Alan, 100')
    p2 = Player('Richie, Dennis, 200')
    p3 = Player('Knuth, Donald, 300')

    players.append(p1)
    players.append(p2)
    players.append(p3)

    p2.add10percent()

    do_pretty_print(players)


#
```

```cpp
main(int argc, char *argv[]) {
  vector<Player> players;

  Player p1{"Turing, Alan, 100"};
  Player p2{"Richie, Dennis, 200"};
  Player p3{"Knuth, Donald, 300"};

  players.push_back(p1);
  players.push_back(p2);
  players.push_back(p3);

  p2.add10percent();

  do_pretty_print(players);

  return 0;
}
```

```
11:46 cygwin> python3 player_a_b.py
+------+----------+------------+------------+
| Rank | Score    | Last Name  | First Name |
+------+----------+------------+------------+
|    1 |   100.00 | Turing     | Alan       |
|    2 |   220.00 | Richie     | Dennis     |
|    3 |   300.00 | Knuth      | Donald     |
+------+----------+------------+------------+
```

```
11:46 cygwin> ./player_a_b.exe
+------+----------+------------+------------+
| Rank |    Score | Last Name  | First Name |
+------+----------+------------+------------+
|    1 |   100.00 | Turing     | Alan       |
|    2 |   200.00 | Richie     | Dennis     |
|    3 |   300.00 | Knuth      | Donald     |
+------+----------+------------+------------+
```

Can you explain the differences?

Reference

Using References (1)

&a ≈ pointer with automatic dereference

```
template<typename tpl_t>
void swap(tpl_t& a, tpl_t& b) {
  tpl_t tmp = a;
  a = b;
  b = tmp;
}
```

```
template<typename tpl_t>
void swap(tpl_t* pa, tpl_t* pb) {
  tpl_t tmp = *pa;
  *pa = *pb;
  *pb = tmp;
}
```

Using References (2)

&a ≈ pointer with automatic dereference and pointer can not change

```
template<typename tpl_t>
void swap(tpl_t& a, tpl_t& b) {
  tpl_t tmp = a;
  a = b;
  b = tmp;
}
```

```
template<typename tpl_t>
void swap(tpl_t* const pa, tpl_t* const pb) {
  tpl_t tmp = *pa;
  *pa = *pb;
  *pb = tmp;
}
```

Using References (3)

&a ≈ pointer with automatic dereference
and pointer can not change
and implicit cast

```
template<typename tpl_t>
void swap(tpl_t& a, tpl_t& b) {
  tpl_t tmp = a;
  a = b;
  b = tmp;
}
int main(...) {
  int i = 1;
  int j = 2;
  swap<int>(i, j);
  ...
```

```
template<typename tpl_t>
void swap(tpl_t* const pa, tpl_t* const pb) {
  tpl_t tmp = *pa;
  *pa = *pb;
  *pb = tmp;
}
int main(...) {
  int i = 1;
  int j = 2;
  swap<int>(&i, &j);
  ...
```

Reference for Input Parameters

```
funct(const type_t& obj) {
...

}
```

`const T& obj` ⇔ object shall not be modified by the function

```
funct(type_t& obj) {
...

}
```

`T& obj` ⇔ object can be modified, stay alert...

```
funct(type_t obj) {
...

}
```

`T obj` ⇔ object is copied, rarely needed.

What is missing ?

Returning Reference (1)

```
struct String {
  std::string s_;

  explicit String(const std::string& s) : s_{s} {}

  ~String() = default;

  String(const String& s) = default;

  friend std::ostream& operator<<(std::ostream& os, const String& obj)
  {
    os << obj.s_;
    return os;
  }
};


String s("Hello World")
cout << s << endl;
```

Returned reference: same as input parameter

Returning Reference (2)

```
struct String {
  std::string s_;
  ...
  friend std::ostream& operator<<(std::ostream& os, const String& obj) {
    os << obj.c.s_;
    return os;
  }
  // remove leading white spaces, in-place
  String& ltrim() {
    std::size_t idx = s_.find_first_not_of(" ");
    if (idx != std::string::npos) {
      s_ = s_.substr(idx);
    }
    return *this;
  }
};

String s("   abc")
cout << s << endl;
cout << s.ltrim() << endl;
```

What is missing?

Note the *this

Returning Reference (3)

```
struct String {
  std::string s_;
  ...
  // remove leading white spaces, in-place
  String& ltrim() {
    std::size_t idx = s_.find_first_not_of(" ");
    if (idx != std::string::npos) {
      s_ = s_.substr(idx);
    }
    return *this;
  }
  // remove tailing white spaces, in-place
  String& rtrim() {
    ...
    return *this;
 }
};

String s("   abc  ")
cout << s << endl;
cout << s.ltrim().rtrim() << endl;
```

fluent interface

Typical Example of an *adapter* design pattern:
Class with the interface you want instead of given interface

Reference: Summary

- Suggested Reading
  - CPP how to program 8th edition, Sections 6.14
  - cours_cpp.pdf, pages 19 to 28
- Summary
  - Reference ~ *const ptr with automatic dereference
  - In function call: automatic cast of a variable into a reference
- Good Practice
  - No need to use reference on primitive types
  - Use const reference parameter passing
  - Beware of code returning a reference
    - `int &operator[](int idx);` vs `int operator[](int idx) const;`

Reference: Improvement (1)

```
int main() {
  std::vector<Player> ps;
  // lines removed reading players
  std::cout << "Before search = " << ps << '\n';

  // search for player with max score
  Player& candidate = ps[0];
  for(auto& value: ps) {
    if (value > candidate) {
      candidate = value;
    }
  }
  std::cout << "Candidate = " << candidate << std::endl;
  std::cout << "After search = " << ps << std::endl;
  return 0;
}
```

```
Before search  = {
{ "Dwalin    ", 5 },
{ "Balin     ", 7 },
{ "Kili      ", 4 },
{ "Fili      ", 2 },
{ "Dori      ", 8 },
{ "Nori      ", 6 },
{ "Ori       ", 1 },
{ "Oin       ", 9 },
{ "Gloin     ", 0 },
{ "Bifur     ", 3 },
{ "Bofur     ", 7 },
{ "Bombur    ", 2 }
}
```

```
Candidate =  { "Oin       ", 9 },
```

```
After search  = {
{ "Oin       ", 9 },
{ "Balin     ", 7 },
{ "Kili      ", 4 },
{ "Fili      ", 2 },
{ "Dori      ", 8 },
{ "Nori      ", 6 },
{ "Ori       ", 1 },
{ "Oin       ", 9 },
{ "Gloin     ", 0 },
{ "Bifur     ", 3 },
{ "Bofur     ", 7 },
{ "Bombur    ", 2 }
}
```

Can you explain?

Reference: Improvement (2)

```
int main() {
std::vector<Player> ps;
// lines removed reading players
std::cout << "Before search = " << ps << '\n';

// search for max value
size_t current_idx = 0;
size_t best_idx = 0;

for(auto& aplayer: ps) {
  if (aplayer > ps) {
    best_idx = current_idx;
  }
  ++current_idx;
}
auto& candidate = ps[best_idx];

std::cout << "Candidate = " << candidate << std::endl;
std::cout << "After search = " << ps << std::endl;
return 0;
}
```

Code Review

```
auto iterator = std::ranges::max_elements(ps, Player::lt)
auto& candidate = *iterator;
```

More on Reference (1)

```
#include <iostream>
#include <string>

using namespace std;

class Q {
  int num_;
  int den_;

 public:
  Q(int num, int den) : num_{num}, den_{den} {}

  void print(string sep) const {
    cout << "Q = " << num_
         << sep << den_
         << std::endl;
  }
};

int main() {

  Q myq{3,4};
  myq.print(" / ");
}
```

```
> g++ -O3 -std=c++14 test019.cpp
> a.exe
3 / 4
```

More on Reference (2)

```
#include <iostream>
#include <string>

using namespace std;

class Q {
  int num_;
  int den_;

 public:
  Q(int num, int den) : num_{num}, den_{den} {}

  void print(string sep) const {
    cout << "Q = " << num_ << sep << den_ << std::endl;
  }
};

int main() {

  Q myq{3,4};
  myq.print(" / ");
}
```

What about using a reference here? (to avoid copy)

More on Reference (3)

```cpp
#include <iostream>
#include <string>

using namespace std;

class Q {
  int num_;
  int den_;

 public:
  Q(int num, int den) : num_{num}, den_{den} {}

 void print(string& sep) const {
    cout << "Q = " << num_
         << sep << den_
         << std::endl;
  }
};

int main() {

  Q myq{3,4};
  myq.print(" / ");
}
```

```
> g++ -O3 -std=c++14 perfect_forwarding.cpp
 error: cannot bind non-const lvalue reference of
type ‘std::string& {aka
std::basic_string<char>&}’ to an rvalue of type
‘std::string {aka std::basic_string<char>}’
   myq.print(“ / ");
               ^
In file included from /usr/lib/gcc/x86_64-pc-
cygwin/7.3.0/include/c++/string:52:0,
/usr/lib/gcc/x86_64-pc-
cygwin/7.3.0/include/c++/bits/basic_string.h:3535
:7: note:   after user-defined conversion:
std::basic_string<_CharT, _Traits,
_Alloc>::basic_string(const _CharT*, const
_Alloc&) [with _CharT = char; _Traits =
std::char_traits<char>; _Alloc =
std::allocator<char>]
       basic_string(const _CharT* __s, const
_Alloc& __a = _Alloc());
```

Lvalue vs. Rvalue [1]

- Rvalue
  - Temporary objects.
  - Objects without names.
  - Objects which have no address.

In blue, only Rvalue

```
int n = 5;
string a = string("Rvalue");
string b = a + itos(n);

my_function(a + itos(n));
```

```
void my_function(const string& value);
```

```
void my_function(string& value);
```

Lvalue vs. Rvalue [2]

- Lvalue: Can be assigned to
  - Can appear on the RHS

```
int n = 5;
string a = string("Rvalue");
string b = a + itos(n);
```

In red, Lvalue

```
const int p = 5;

int &r = 5;
```

p: Lvalue or Rvalue
Lvalue

r: Possible?
No, A non-const lvalue reference will only bind to non-const rvalues

emplace_back() [1a]

```
std::ifstream fin(file_name, std::ios::in);
string line;
while (std::getline(fin, line)) {
  players.emplace_back(line);
}
```

emplace_back() [1b]

```
class Q {
  int num_;
  int den_;
 public:
  Q(int num, int den) : num_{num}, den_{den} {}
  void print(const string &sep) const {
    cout << "Q = " << num_ << sep
                   << den_ << std::endl;
  }
};

int main() {
  vector<Q> qs;
  for(int num = 3; int den = 4; num < 6; ++num; ++den) {
    qs.emplace_back(num, den);
  }
  ...
}
```

```
template<typename T>
class vector {
   ...
 public:
  void emplace_back(int num, int den) {
    T q(num, den);
    ...
  }
};
```

Not generic!!!

emplace_back() [2]

```
class Q {
  int num_;
  int den_;
 public:
  Q(int num, int den) : num_{num}, den_{den} {}

  void print(const string &sep) const {
    cout << "Q = " << num_ << sep
                   << den_ << std::endl;
  }
};

template<typename T>
class vector {
  ...
 public:
  template<typename... Args>
  void emplace_back(Args && ... args) {
    T obj(std::forward<Args>(args)...);
    ...
  }
};
```

Using &&
Forward Reference

Using special ...
Variadic Template

Using std::forward
Cast to original type

Lambda Capture by Reference

```
bool sort_ascending = false;
auto mycompare2 = [&sort_ascending](
                        const Player &a,
                        const Player &b) -> bool  {
      return sort_ascending ^ (a.score_ > b.score_);
    };
sort_ascending = true;
std::sort(players.begin(), players.end(), mycompare2);
```

```
Name = Balin      Score = 2
Name = Dwalin     Score = 4
Name = Bifur      Score = 5
Name = Thorin     Score = 10
Name = Bofur      Score = 15
Name = Bombur     Score = 20
Name = Fili       Score = 23
Name = Kili       Score = 25
```

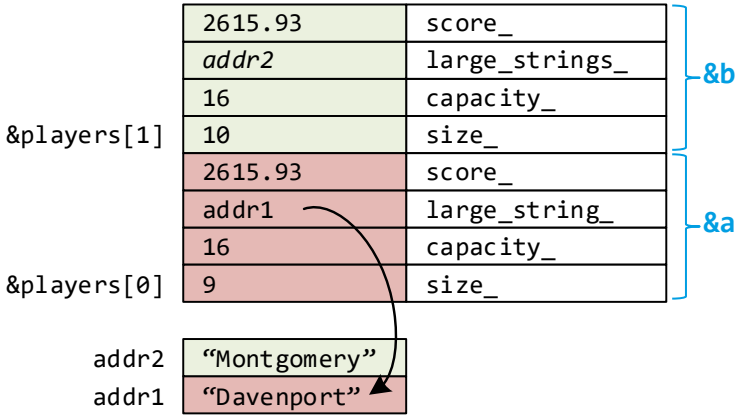
Capture by reference

- Occurs when the lambda is invoked
- Captured value by reference must still be in scope when lambda is invoked

POLYTECH
NICE-SOPHIA

Object Copy vs. Object Move

Basic Swap: Full Copy (1)



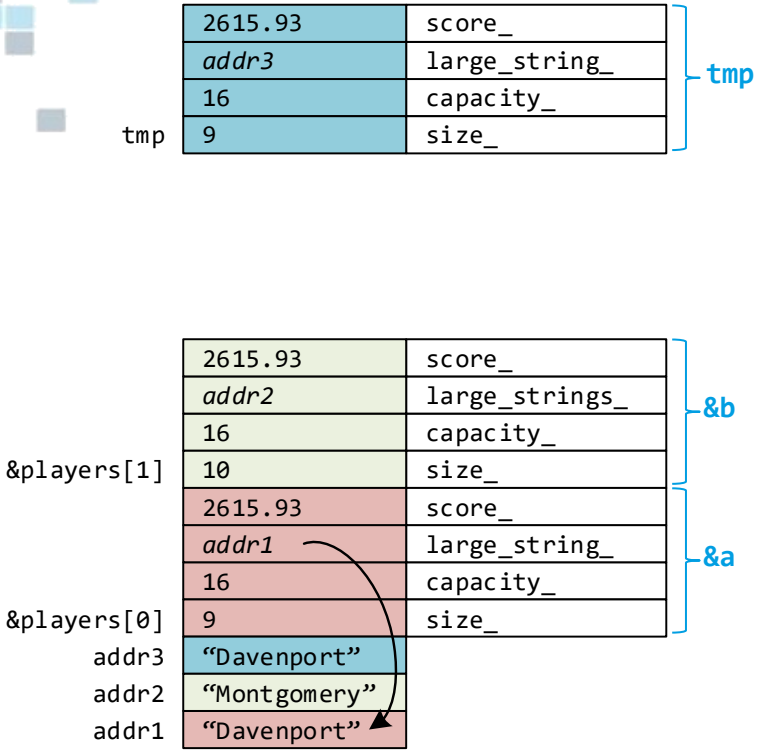

```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = a;
  a = b;
  b = tmp;
}
```

```
struct Player {
  string last_name_;
  double score_;
};
```

```
template<typename tpl_t>
class vector {
  int size_;
  int capacity_;
  tpl_t *raw_storage_;
};
```

```
class string {
  int size_;
  int capacity_;
  union {
   char small_string_[8];
   char *large_string_;
  };
};
```

Basic Swap: Full Copy (2)



tmp Player created on the stack, full copy of *a*

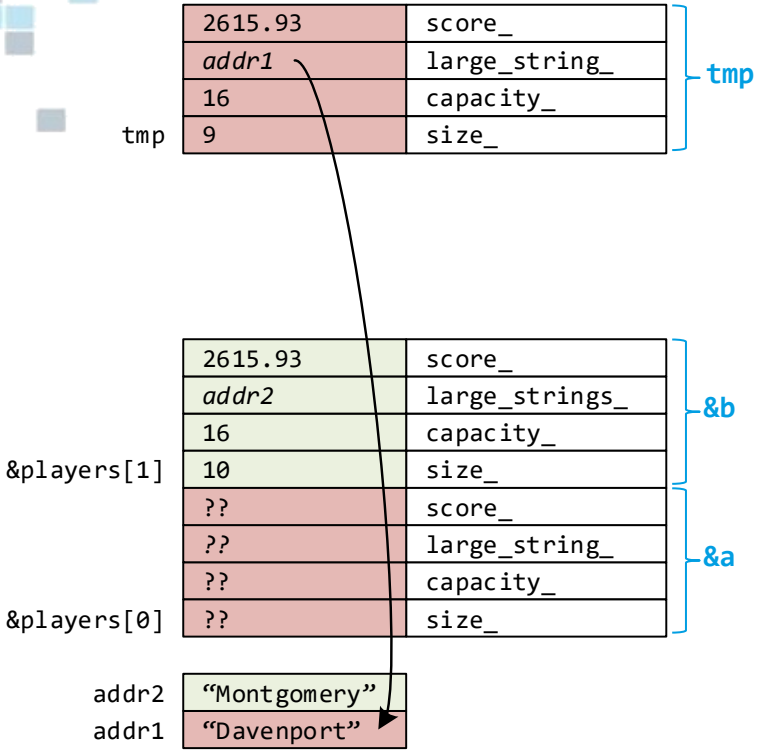
```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = a;
  a = b;
  b = tmp;
}
```

```
struct Player {
  string last_name_;
  double score_;
};
```

```
template<typename tpl_t>
class vector {
  int size_;
  int capacity_;
  tpl_t *raw_storage_;
};
```

```
class string {
  int size_;
  int capacity_;
  union {
   char small_string_[8];
   char *large_string_;
  };
};
```

Optimized Swap: Shallow Copy (1)



tmp Player created on the stack, shallow copy of *a*

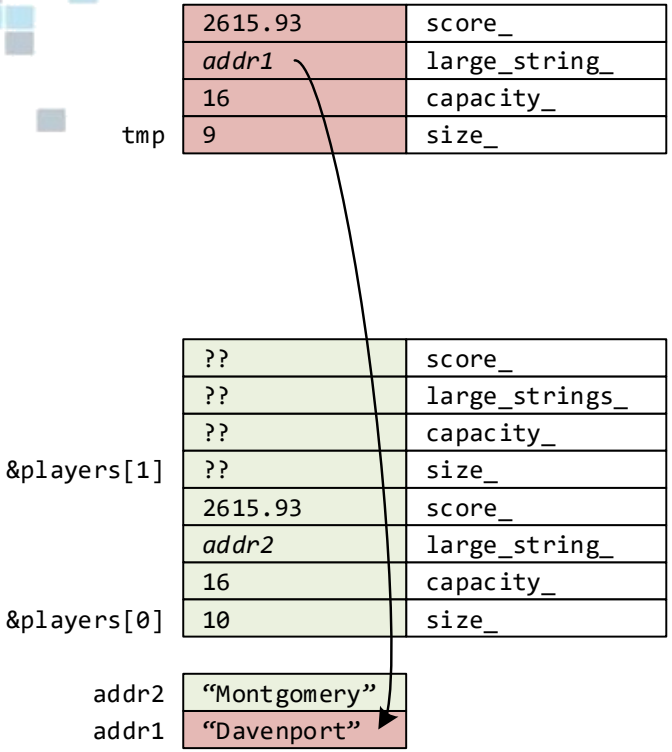
```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = std::move(a);
  a = b;
  b = tmp;
}
```

```
struct Player {
  string last_name_;
  double score_;
};
```

```
template<typename tpl_t>
class vector {
  int size_;
  int capacity_;
  tpl_t *raw_storage_;
};
```

```
class string {
  int size_;
  int capacity_;
  union {
   char small_string_[8];
   char *large_string_;
  };
};
```

Optimized Swap: Shallow Copy (2)



Shallow copy of *b* into *a*

```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = std::move(a);
  a = std::move(b);
  b = std::move(tmp);
}
```

```
struct Player {
  string last_name_;
  double score_;
};
```

```
template<typename tpl_t>
class vector {
  int size_;
  int capacity_;
  tpl_t *raw_storage_;
};
```

```
class string {
  int size_;
  int capacity_;
  union {
   char small_string_[8];
   char *large_string_;
  };
};
```

Optimized Swap: Shallow Copy (3)

	??	score_
	??	large_string_
	??	capacity_
tmp	??	size_

Shallow copy of *tmp* into *b*

```
struct Player {
  string last_name_;
  double score_;
};
```

```
template<typename tpl_t>
class vector {
  int size_;
  int capacity_;
  tpl_t *raw_storage_;
};
```

	2615.93	score_
	addr1	large_strings_
	16	capacity_
&players[1]	9	size_
	2615.93	score_
	addr2	large_string_
	16	capacity_
&players[0]	10	size_

addr2	"Montgomery"
addr1	"Davenport"

```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = std::move(a);
  a = std::move(b);
  b = std::move(tmp);
}
```

```
class string {
  int size_;
  int capacity_;
  union {
   char small_string_[8];
   char *large_string_;
  };
};
```

Optimized Swap: Shallow Copy (4)

	??	score_
	??	large_string_
	??	capacity_
tmp	??	size_

Destruction of *tmp* is immediate

```
struct Player {
  string last_name_;
  double score_;
};
```

```
template<typename tpl_t>
class vector {
  int size_;
  int capacity_;
  tpl_t *raw_storage_;
};
```

	2615.93	score_
	addr1	large_strings_
	16	capacity_
&players[1]	9	size_
	2615.93	score_
	addr2	large_string_
	16	capacity_
&players[0]	10	size_

addr2	"Montgomery"
addr1	"Davenport"

```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = std::move(a);
  a = std::move(b);
  b = std::move(tmp);
}
```

```
class string {
  int size_;
  int capacity_;
  union {
   char small_string_[8];
   char *large_string_;
  };
};
```

Object Move, Copy & Destroy (1)

- The compiler generate implicit move, copy and destroy functions for you.
  - Unless you are allocating raw memory with new, the compiler generated functions are better optimized
- Generated functions are
  - Default constructor (unless non default is provided)
  - Copy constructor
  - Move constructor
  - Copy assignment
  - Move assignment
  - Destructor

Object Move, Copy & Destroy (2)

- The compiler will create move, copy and destroy functions for you.
  - Unless you are allocating raw memory with new, the compiler generated functions are better optimized

```
// Default Constructor
// => Player a;
Player();

// Copy Constructor
// => Player b{a};
// => Player b = a;
Player(const Player &player);

// Move Constructor
// => Player c{std::move(b)};
// => Player c = std::move(b);
Player(Player &&player) noexcept;

// Destructor
~Player() noexcept;

// Copy Assignment
// => Player d;
//    d = c;
Player &operator=(const Player &player);

// Move Assignment
// => Player e;
//    e = std::move(d);
Player &operator=(Player &&player) noexcept;
```

Object Move, Copy & Destroy (3)

```
// Source Code

class Player {
  string id_;
  int score_;
};
```

```
// Generated Code

  class Player {
   private:
    string id_;
    int score_;
   public:
    Player() = default;
    ~Player() noexcept = default;
    Player(const Player &) = default;
    Player &operator=(const Player &) = default;
    Player(Player &&) noexcept = default;
    Player& operator=(const Player &&) noexcept = default;
  };
```

POLYTECH
NICE-SOPHIA

Class Member Initialization

Member Initialization (1)

```
class Spline {
 private:
  vector<double> xs_;
  vector<double> as_;
  vector<double> bs_;
  vector<double> cs_;
  vector<double> ds_;
  size_t  dim_;
 public:
  Spline(const vector<double> &xs, const vector<double> &ys) {

    ...

    Eigen::VectorXd x = ma.fullPivHouseholderQr().solve(b)
    for (size_t i = 0; i < dim_; ++i) {
      auto bi = 3 * i;
      as_.push_back(x[bi + 0]);
      bs_.push_back(x[bi + 1]);
      cs_.push_back(x[bi + 2]);
    }
  }

  ...

};
```

How as_, bs_ and cs_ are initialized ?

Member Initialization (2)

```
class Spline {
 private:
  vector<double> xs_;
  vector<double> as_;
  vector<double> bs_;
  vector<double> cs_;
  vector<double> ds_;
  size_t  dim_;
 public:
  Spline(const vector<double> &xs, const vector<double> &ys) {

    ...

    Eigen::VectorXd x = ma.fullPivHouseholderQr().solve(b)
    for (size_t i = 0; i < dim_; ++i) {
      auto bi = 3 * i;
      as_.push_back(x[bi + 0]);
      bs_.push_back(x[bi + 1]);
      cs_.push_back(x[bi + 2]);
    }
  }

  ...

};
```

Class members are initialized before the body of the constructor

Member Initialization (3)

```
class Spline {
 private:
  vector<double> xs_;
  vector<double> as_;
  vector<double> bs_;
  vector<double> cs_;
  vector<double> ds_;
  size_t  dim_;
 public:
  Spline(const vector<double> &xs, const vector<double> &ys) : xs_{xs}, ds_{ys} {

    ...

    Eigen::VectorXd x = ma.fullPivHouseholderQr().solve(b)
    for (size_t i = 0; i < dim_; ++i) {
      auto bi = 3 * i;
      as_.push_back(x[bi + 0]);
      bs_.push_back(x[bi + 1]);
      cs_.push_back(x[bi + 2]);
    }
  }

  ...

};
```

Non default initialization can be specified in a *member initialization list*

Member Initialization (4)

```
// Spline constructor (v1)
Spline(const vector<double> &xs, const vector<double> &ys) {
  xs_ = xs;
  ds_ = ys;
  ...
}
```

Doing initialization twice:
(1) default
(2) copy

```
// Spline constructor (v2)
Spline(const vector<double> &xs, const vector<double> &ys) : xs_{xs}, ds_{ys} {
  ...
}
```

Efficiency ? v1 or v2