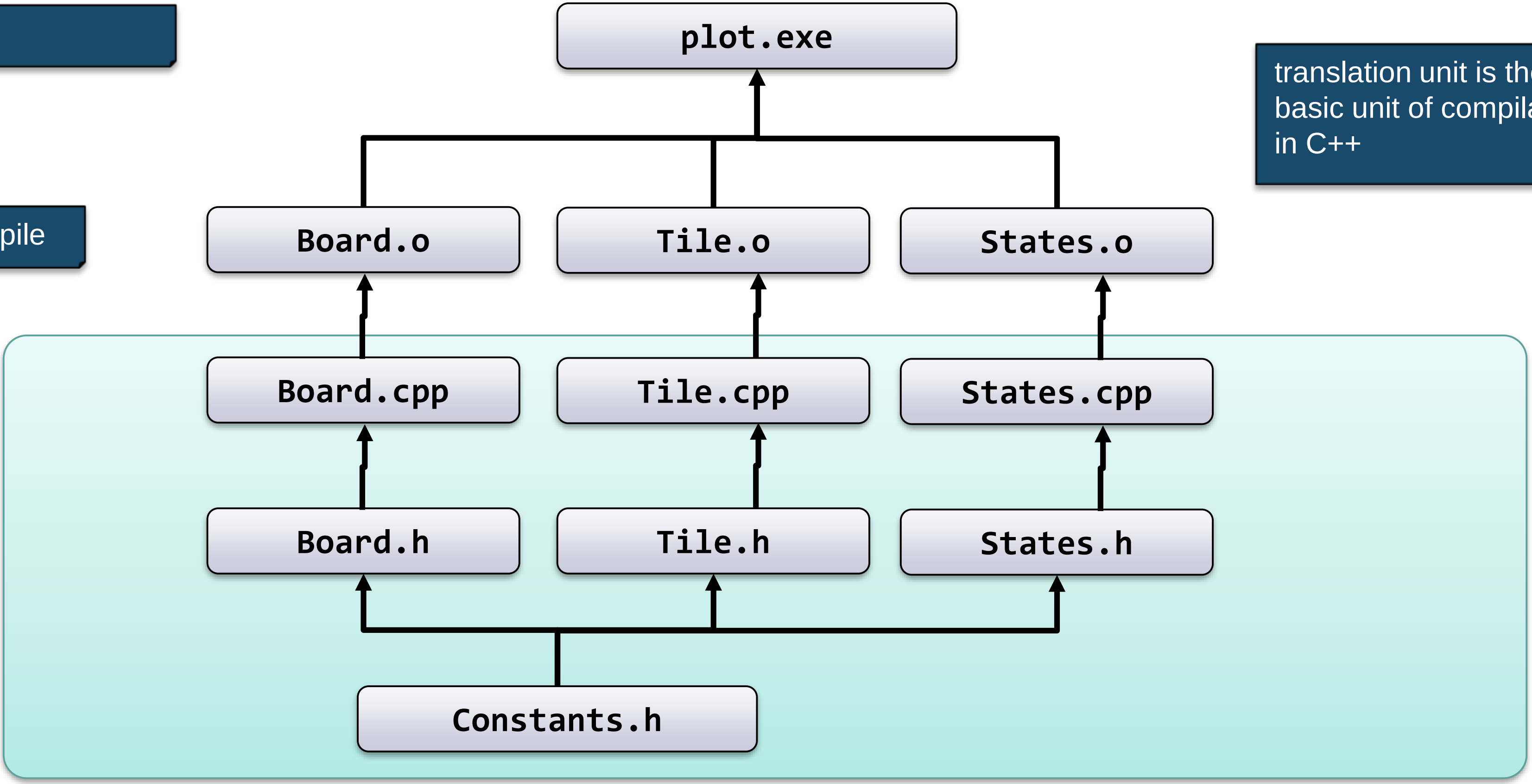
C++ Projects

Link

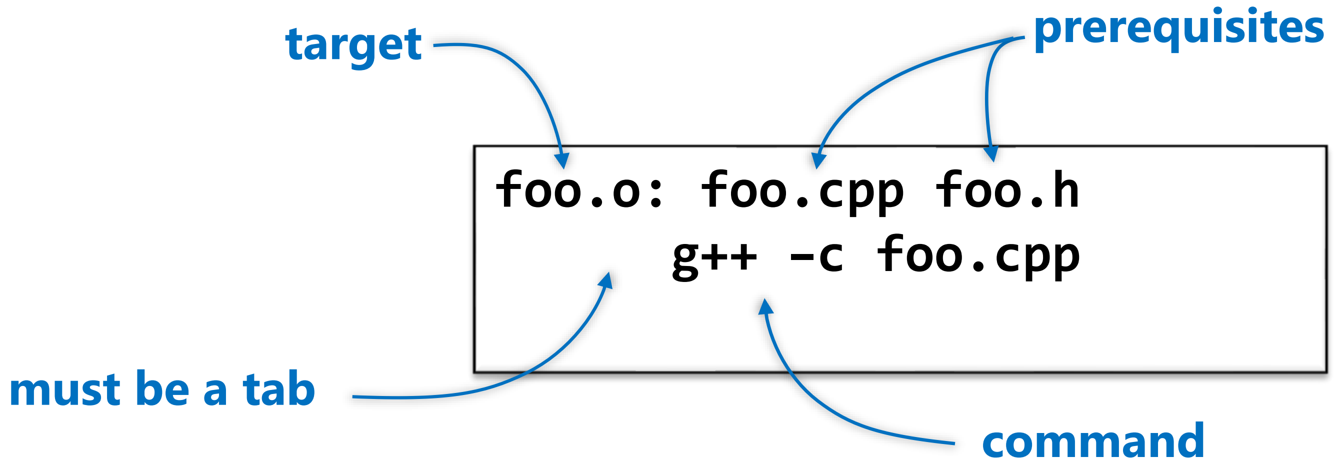
Compile



translation unit is the basic unit of compilation in C++

Makefile (1)

- A Makefile specifies how *out of date* files must be processed.
- The executable `make` reads the Makefile and invokes the rules needed to re-build the target.
- Full documentation: <u>here</u>



```
foo.o: foo.cpp foo.h
	g++ -c foo.cpp
```

Makefile (2)

target

$< first element of the prerequisite list

```
foo.o: foo.cpp foo.h
      g++ -c $<

a.exe: $(OBJ)
      @g++  $@ $^
```

$( ) variables

@: don’t echo

the target

$< all elements of the pre-requisite list

022

Class

Class: OOP Foundation (1)

```
class Shape {









};
```

Class: blueprint of an object

Class: OOP Foundation (2)

```
class Shape {
 private:
  int color_;









};
```

Objects of a class: member (private, protected or public)

Class: OOP Foundation (3)

```
class Shape {
 private:
  int color_;
 public:
  Shape(int color) {
    color_ = color;
  }











};
```

This special function which returns nothing is a constructor: it creates and initialize an object

Functions of a class are used to initialize, change, obtain information about the object

Class: OOP Foundation (4)

```
class Shape {
 private:
  int color_;
 public:
  Shape(int color) {
    color_ = color;
  }
  ~Shape() = default;









};
```

This special function which returns nothing is a destructor:
it releases the resources of an object. In most cases, the default implementation is what you want

Class: OOP Foundation (5)

```
class Shape {
 private:
  int color_;
 public:
  Shape(int color) {
    color_ = color;
  }
  ~Shape() = default;

  inline
  int get_color() const { return color_;}

  void draw() const {
    std::cout << "draw a shape" << std::endl;
  }

  void set_color(int color) { color_ = color; }
};
```

Other functions, called methods, can either access the members (accessor) or change the members (mutators)

Class: OOP Foundation (5 bis)

```
class Shape {
 private:
  int color_;
 public:
  Shape(int color);
  int get_color() const;
  void set_color(int color);
};

Shape::Shape(int color) {
    color_ = color;
}

int Shape::get_color() const {
 return color_;
}

void Shape::set_color(int color) {
 color_ = color;
}
```

Class Declaration

Class Definition

Object Creation

```
int main() {
  Shape my_shape(QColor::red);

}
```

Type of the object: either a basic type (int, double) or a class type

Name of the object

Using an object

```
int main() {
  Shape my_shape(QColor::red);
  std::cout << “Color is “ << my_shape.get_color();
}
```

Apply a method on the object

Uniform Initialization (1)

```
class Rect {
 private:
  int width_;
  int height_;







}


int main() {
  Rect my_rect{2, 3};
}
```

Aggregate-initialization

Uniform Initialization (2)

```
class Rect {
 private:
  int width_;
  int height_;
 public:
  Rect(int w, int h) : width_{w}, height_{h} {}




}


int main() {
  Rect my_rect{2, 3};
}
```

Regular Constructor alternative to:

```
Rect my_rect(2,3);
```

Uniform Initialization (3)

```
class Rect {
 private:
  int width_;
  int height_;
 public:
  Rect(int w, int h) : width_{w}, height_{h} {}
  Rect(const std::initializer_list<int>& args) {
    width_  = *(args.begin());
    height_ = *(args.begin() + 1);
  }
}


int main() {
  Rect my_rect{2, 3};
}
```

Initializer List

Class: Summary

- Suggested Reading
  - CPP how to program 8th edition, Sections 3.1 to 3.5
- Summary
  - Class declaration, definition, instantiation
  - Class constructor & destructor
  - Class accessors and mutators

Constructor / Destructor

Assignment #2

- Write a small program which reads a player report text file in CSV format (coma separated value), sorts players based the score (a floating point in the last field) and pretty print the results.

data.txt

```
Smith,Linda,2615.93
Romero,Georgia,863.93
Davenport,Darin,1990.52
Rubio,Alfonso,2815.77
Wong,Otis,1181.31
Faulkner,Enrique,1321.13
Nolan,Marianne,455.36
Hanna,Thelma,812.47
Irwin,Mara,2638.90
Hartman,Rosalie,17301.72
```

```
shell> sorted_names data.txt
+------+----------+-----------+-----------+
| Rank |    Score | Last Name | 1st Name  |
+------+----------+-----------+-----------+
|    1 | 17301.72 | Hartman   | Rosalie   |
|    2 |  2815.77 | Rubio     | Alfonso   |
|    3 |  2638.90 | Irwin     | Mara      |
|    4 |  2615.93 | Smith     | Linda     |
|    5 |  1990.52 | Davenport | Darin     |
|    6 |  1321.13 | Faulkner  | Enrique   |
|    7 |  1181.31 | Wong      | Otis      |
|    8 |   863.93 | Romero    | Georgia   |
|    9 |   812.47 | Hanna     | Thelma    |
|   10 |   455.36 | Nolan     | Marianne  |
+------+----------+-----------+-----------+
```

Possible Solution (1a)

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

Naming convention: all structure members are post-fixed with _

A `struct` is a `class` with only public members and methods

```
class Player {
public:
...
}
```

Possible Solution (1b)

```
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<Player> players;
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    Player player(line);
    players.push_back(player);
  }
  std::sort(players.begin(), players.end(),
            [](const Player &a, const Player &b) -> bool  {
      return b.score_ < a.score_;
    });
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```

Possible Solution (2)

```cpp
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<Player> players;
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    Player player(line);
    players.push_back(player);
  }
  std::sort(players.begin(), players.end(),
            [](const Player &a, const Player &b) -> bool  {
        return b.score_ < a.score_;
    });
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```

`Player player(line)`

Create a player structure from a given line on the *stack*

Possible Solution (2)

```
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<Player> players;
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    Player player(line);
    players.push_back(player);
  }
  std::sort(players.begin(), players.end(),
            [](const Player &a, const Player &b) -> bool  {
      return b.score_ < a.score_;
    });
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```

```
[](,,) -> t {
...
}
```

Signature of a lambda function

Possible Solution (3)

```cpp
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<Player> players;
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    Player player(line);
    players.push_back(player);
  }
  std::sort(players.begin(), players.end(),
            [](const Player &a, const Player &b) -> bool  {
      return b.score_ < a.score_;
    });
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```

```cpp
for(auto &v : vs) {
  ...
}
```

Range based loop

Possible Solution (4)

```cpp
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<Player> players;
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    Player player(line);
    players.push_back(player);
  }
  std::sort(players.begin(), players.end(),
            [](const Player &a, const Player &b) -> bool  {
      return b.score_ < a.score_;
    });
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```

`print_table_entry()`

Function only applicable to Player object, i.e. a method.

Constructor / Destructor

```cpp
struct Player {
  string last_name_;
  vector<string> names_;
  double score_;
  Player(const string &line);
  ~Player() = default;
  ...
```

Constructor
1) Memory acquisition
2) Initialize elements

Destructor
1) Destroy elements
2) Release memory

Object Life-Time

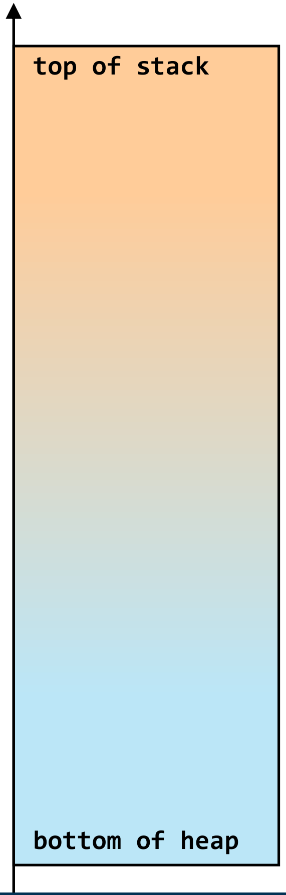
Questions

```cpp
int main(int argc, char *argv[]) {
  string file_name{argv[1]};
  vector<Player> players;
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    Player player(line);
    players.push_back(player);
  }
  std::sort(players.begin(), players.end(),
            [](const Player &a, const Player &b) -> bool  {
      return b.score_ < a.score_;
    });
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```

Write a simplified `struct` for string and vector as found in the STL library?

Where are the players stored in memory ? heap or stack?

Memory Layout (1)

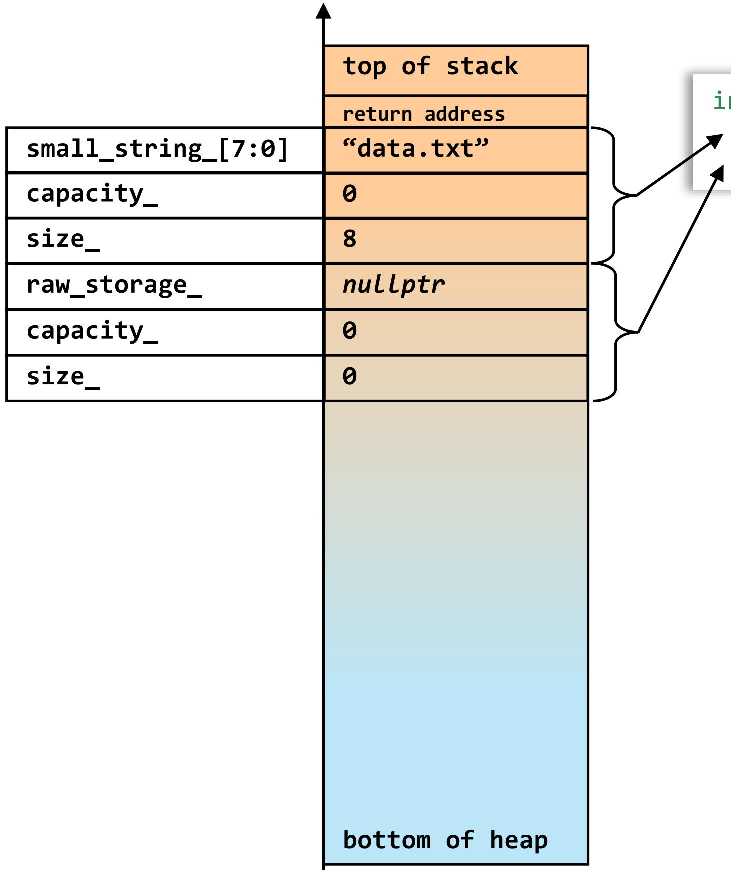


```
struct Player {
  string last_name_;
  vector<string> names_;
  double score_;
};
```

```
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_;
  };
};
```

Memory Layout (2)



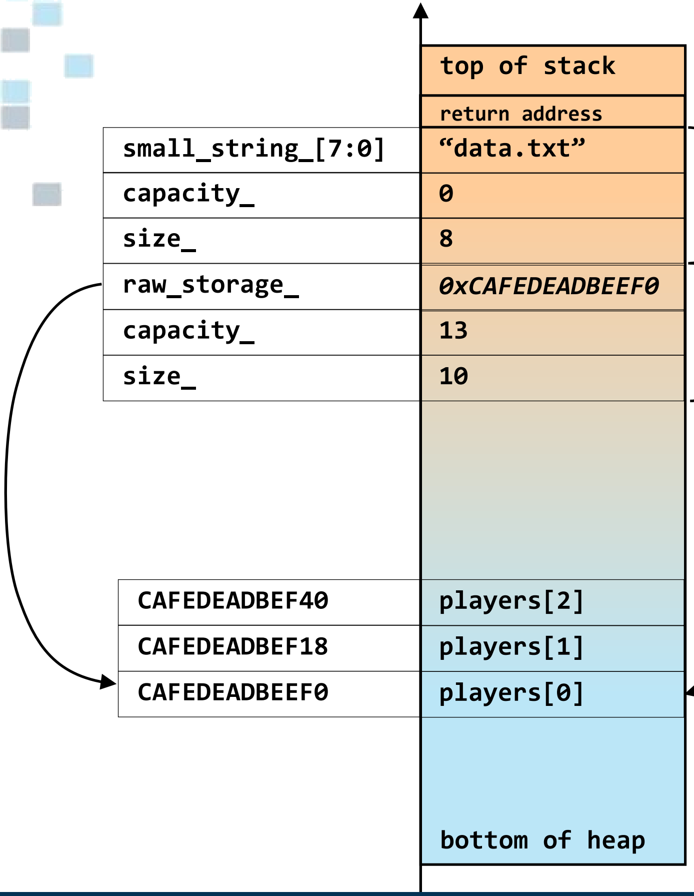
```
int main(int argc, char *argv[]) {
  string file_name(argv[1]);
  vector<Player> players;
```

```
struct Player {
  string last_name_;
  vector<string> names_;
  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_;
  };
};
```

Memory Layout (3)



```
int main(int argc, char *argv[]) {
  string file_name(argv[1]);
  vector<Player> players;
  ...
  while (std::getline(fin, line)) {
    Player player(line);
    players.push_back(player);
  }
```

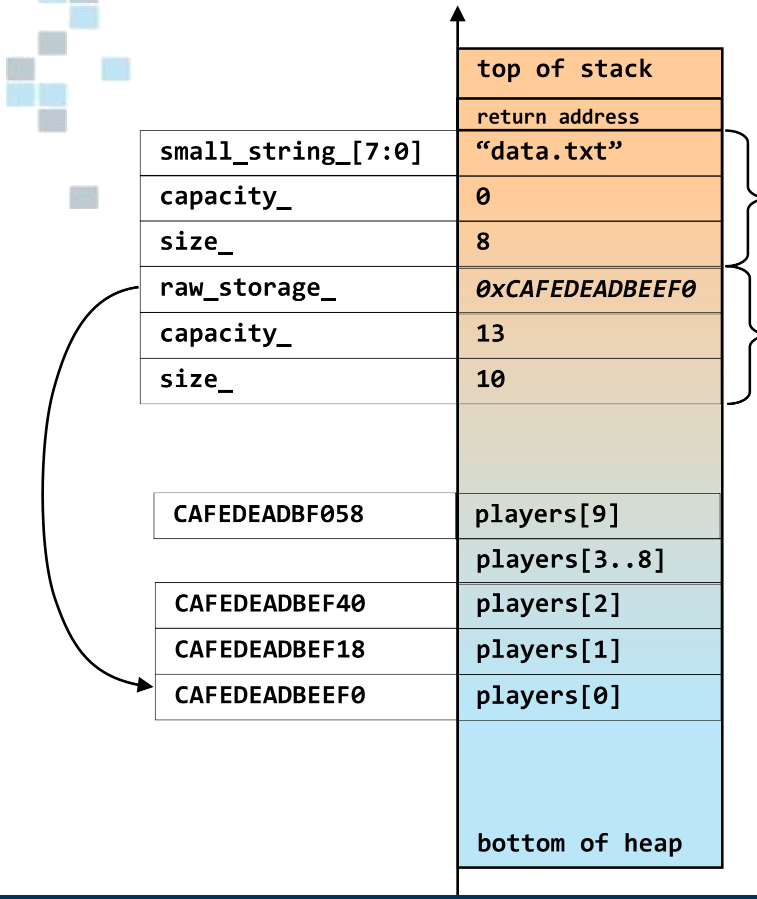
sizeof(player[i]) is 40

```
struct Player {
  string last_name_;
  vector<string> names_;
  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_;
  };
};
```

Memory Layout (4)



```
int main(int argc, char *argv[]) {
  string file_name(argv[1]);
  vector<Player> players;
  ...
  while (std::getline(fin, line)) {
    Player player(line);
    players.push_back(player);
  }
  ...

}
```

```
struct Player {
  string last_name_;
  vector<string> names_;
  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_;
  };
};
```

Call vector destructor on players then string destructor on file_name.

1) Call Player destructor for each player[0..9]
2) Free players raw_storage
3) Free file_name resources (not needed here)
4) Return to caller

Experiment with Destructor (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;

  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    Player player(line);
    players.push_back(player);
  }
  // sort removed
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```

What will be the output ?

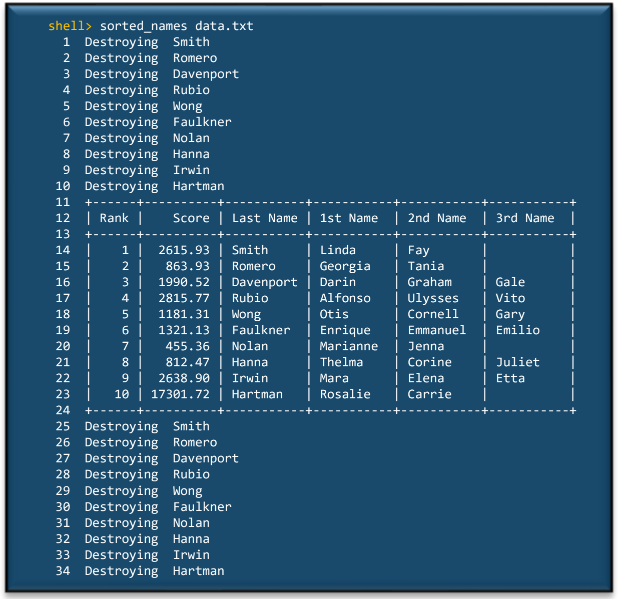
Experiment with Destructor (2)

```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;

  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    Player player(line);
    players.push_back(player);
  }
  // sort removed
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```

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

Experiment with Destructor (3)

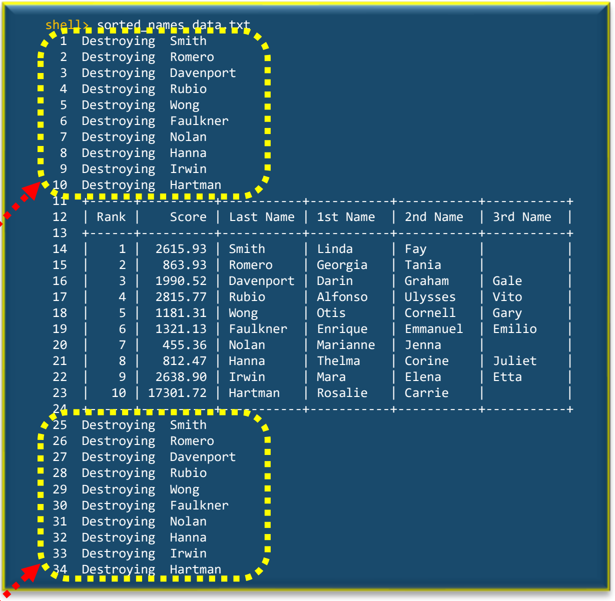
```cpp
struct Player {
  string last_name_;
  vector<string> names_;
  double score_;
  Player(const string &line) { ... }
  ~Player() {
    std::cout << "Destroying " << last_name_ << std::endl;
  }
};
int main(int argc, char *argv[]) {
  string file_name(argv[1]);
  vector<Player> players;
  players.reserve(100);
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    Player player(line);
    players.push_back(player);
  }
  // sort removed
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```



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

Experiment with Destructor (4)

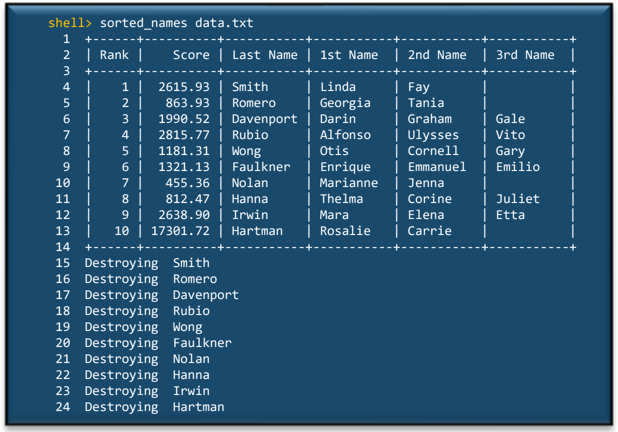
```cpp
struct Player {
  string last_name_;
  vector<string> names_;
  double score_;
  Player(const string &line) { ... }
  ~Player() {
    std::cout << "Destroying " << last_name_ << std::endl;
  }
};
int main(int argc, char *argv[]) {
  string file_name(argv[1]);
  vector<Player> players;
  players.reserve(100);
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    Player player(line);
    players.push_back(player);
  }
  // sort removed
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```



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

Experiment with Destructor (5)

```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);
  }
  // sort removed
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```



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

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 ?

POLYTECH°
NICE-SOPHIA

to be continued...

Sort & Map Helper function

Assignment #2b

- Same as #2, but with detection of duplicate player in the input file.

data.txt

```
Smith,Linda,2615.93
Romero,Georgia,863.93
Davenport,Darin,1990.52
Rubio,Alfonso,2815.77
Wong,Otis,1181.31
Faulkner,Enrique,1321.13
Nolan,Marianne,455.36
Davenport,Darin,1990.520
Hanna,Thelma,812.47
Irwin,Mara,2638.90
Hartman,Rosalie,17301.72
```

```
shell> sorted_names data.txt
+------+----------+-----------+-----------+
| Rank |    Score | Last Name | 1st Name  |
+------+----------+-----------+-----------+
|    1 | 17301.72 | Hartman   | Rosalie   |
|    2 |  2815.77 | Rubio     | Alfonso   |
|    3 |  2638.90 | Irwin     | Mara      |
|    4 |  2615.93 | Smith     | Linda     |
|    5 |  1990.52 | Davenport | Darin     |
|    6 |  1321.13 | Faulkner  | Enrique   |
|    7 |  1181.31 | Wong      | Otis      |
|    8 |   863.93 | Romero    | Georgia   |
|    9 |   812.47 | Hanna     | Thelma    |
|   10 |   455.36 | Nolan     | Marianne  |
+------+----------+-----------+-----------+
```

Sort: Helper Function

```
std::sort(players.begin(), players.end(),
   [](const Player &a, const Player &b) -> bool  {
     return a.score_ > b.score_;
   }
 );
```

```
class Player {
...

  static bool compare(const Player &a, const Player &b) {
    return a.score_ > b.score_;
  }
};
```

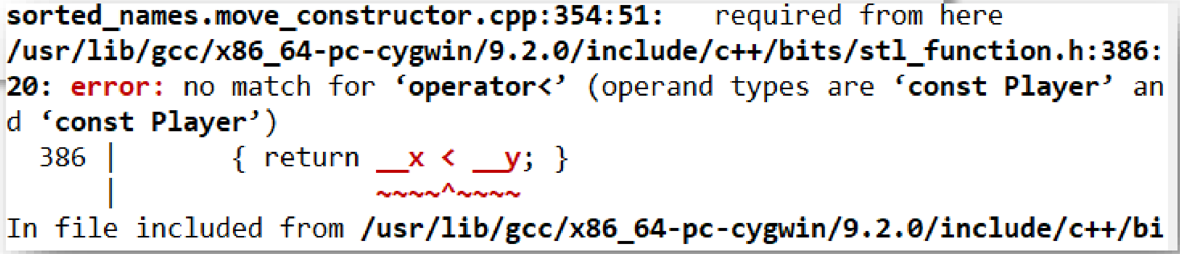
```
std::sort(players.begin(), players.end(), Player::compare);
```

You want to replace a lambda function by an explicit function ?
How?
Where to store the function?

Map: Helper Function (1)

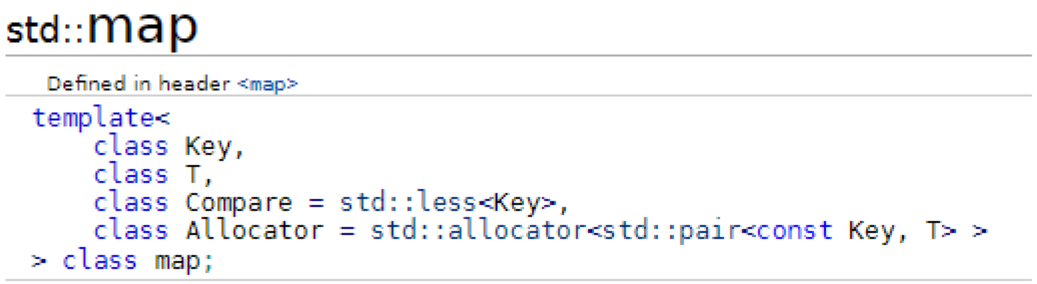
```
std::map<Player, int> map_of_players;
...

while(...) {
  Player player(line);
  auto iter = map_of_players.find(player);
  if (iter == map_of_players.end()) {
    map_of_players.insert({players, lineno});
  } else {
    std::cout << "INFO: duplicate at line " << lineno << std::endl;
  }
  ++lineno
}
```

Problem: you want to detect duplicate players in the input files.
How?



```
sorted_names.move_constructor.cpp:354:51:   required from here
/usr/lib/gcc/x86_64-pc-cygwin/9.2.0/include/c++/bits/stl_function.h:386:
20: error: no match for ‘operator<’ (operand types are ‘const Player’ an
d ‘const Player’)
  386 |       { return __x < __y; }
      |                ~~~~^~~~~
In file included from /usr/lib/gcc/x86_64-pc-cygwin/9.2.0/include/c++/bi
```

Map: Helper Function (2)



std::map

Defined in header <map>

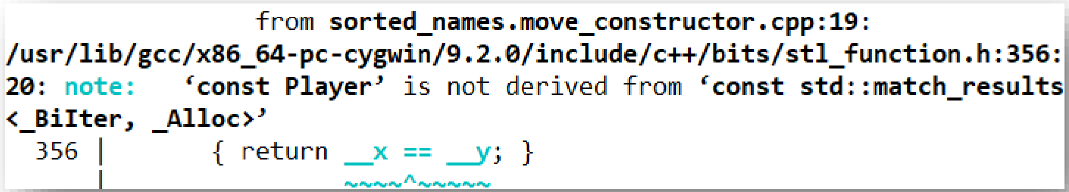
```
template<
    class Key,
    class T,
    class Compare = std::less<Key>,
    class Allocator = std::allocator<std::pair<const Key, T> >
> class map;
```

```
class Player {
...

  static bool compare(const Player &a, const Player &b) {
    return a.score_ > b.score_;
  }

  friend bool operator<(const Player &a, const Player &b) {
    return a.isless(b);
  }
};
```

Unordered Map: Helper Function (1)

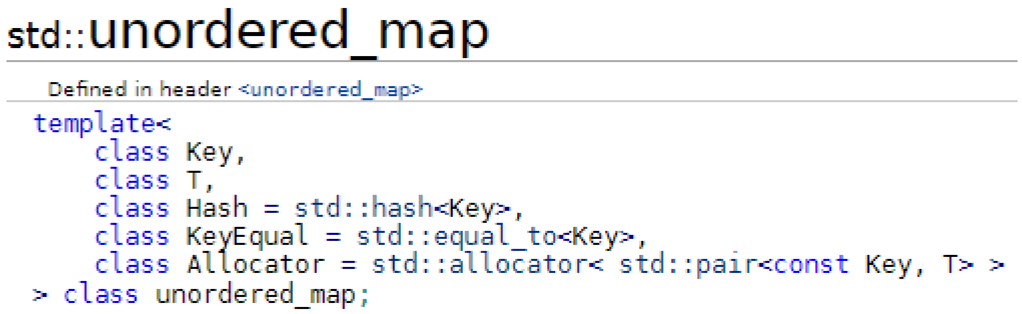
```
std::unordered_map<Player, int> map_of_players;
...

while(...) {
  Player player(line);
  auto iter = map_of_players.find(player);
  if (iter == map_of_players.end()) {
    map_of_players.insert({players, lineno});
  } else {
    std::cout << "INFO: duplicate at line " << lineno << std::endl;
  }
  ++lineno
}
```

Problem: you want to detect duplicate players in the input files.
How?



```
                 from sorted_names.move_constructor.cpp:19:
/usr/lib/gcc/x86_64-pc-cygwin/9.2.0/include/c++/bits/stl_function.h:356:
20: note:   'const Player' is not derived from 'const std::match_results
<_BiIter, _Alloc>'
  356 |       { return __x == __y; }
      |                ~~~~^~~~~~
```

Unordered Map: Helper Function (2)



std::unordered_map

Defined in header <unordered_map>

```
template<
    class Key,
    class T,
    class Hash = std::hash<Key>,
    class KeyEqual = std::equal_to<Key>,
    class Allocator = std::allocator< std::pair<const Key, T> >
> class unordered_map;
```

```
class Player {
...

  friend bool operator==(const Player &a, const Player &b) {
    return a.isequal(b);
  }
};
```

You now have to define the isequal() method

Unordered Map: Helper Function (3)

```
std::unordered_map<Player, int, Player::Hash> map_of_players;
```

```
class Player {
...
   struct Hash {
    size_t operator()(const Player &player) const {
      return player.hash();
    }
  };
...
};
```

You now have to define the hash() method

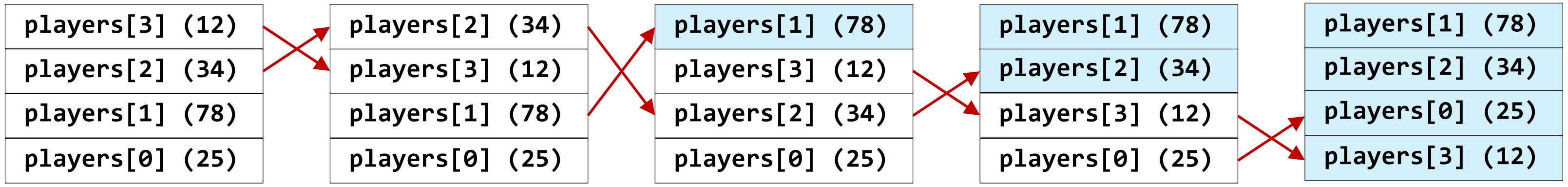
Pattern: function object:
A object which acts like a function

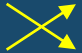
Another Experiment (2)

```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
```

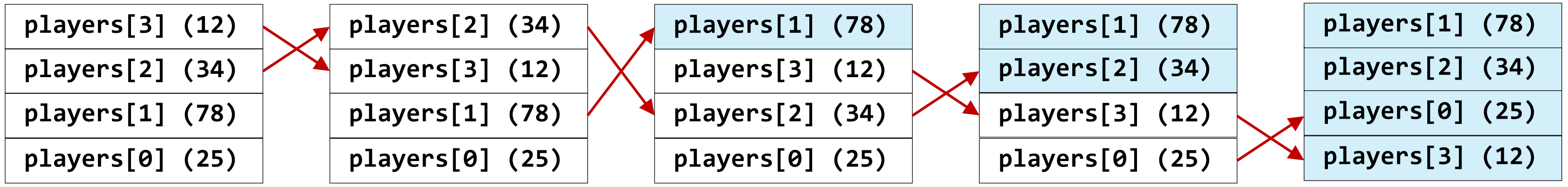
Sort (1)



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;
}
```

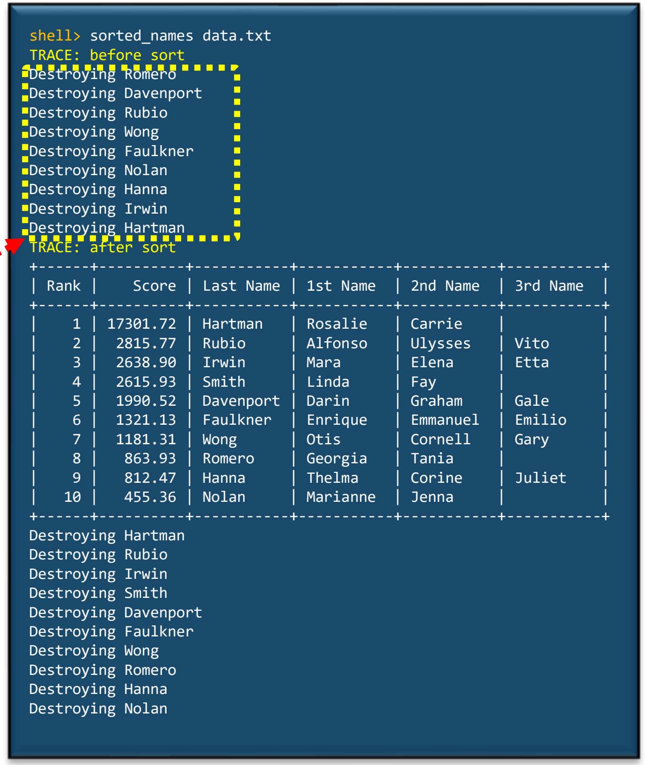
Sort (3)

```cpp
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)

```cpp
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
```

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;
```

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;`

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 perfect_forwarding.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)

```
#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);
```

Lvalue vs. Rvalue [2]

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

```
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 lvalues

emplace_back()   [1]

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

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

Using &&
Forward Reference

Using special ...
Variadic Template

Using std::forward
Cast to original type

022

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;
  ...
}
```

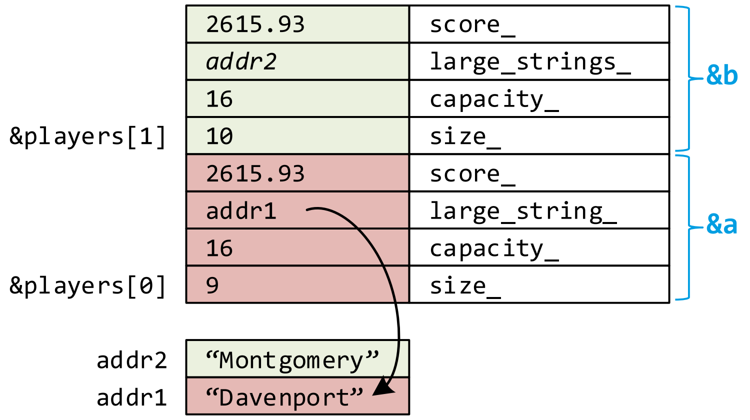
Doing initialization twice:
(1) default
(2) copy

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

Efficiency ? v1 or v2

Object Copy vs. Object Move

Basic Swap: Full Copy (1)



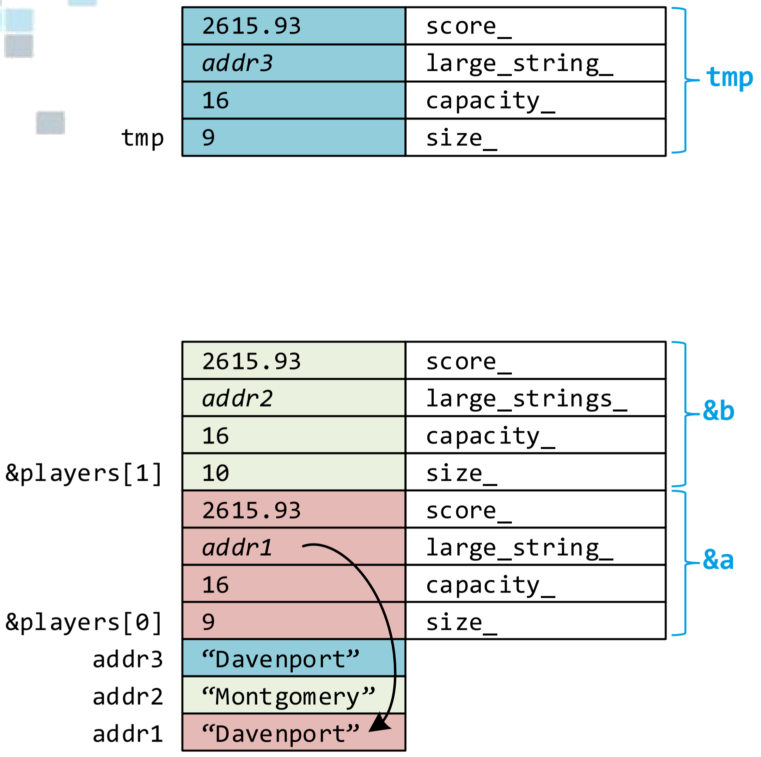
```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = a;
  a = b;
  b = tmp;
}
```

```
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*

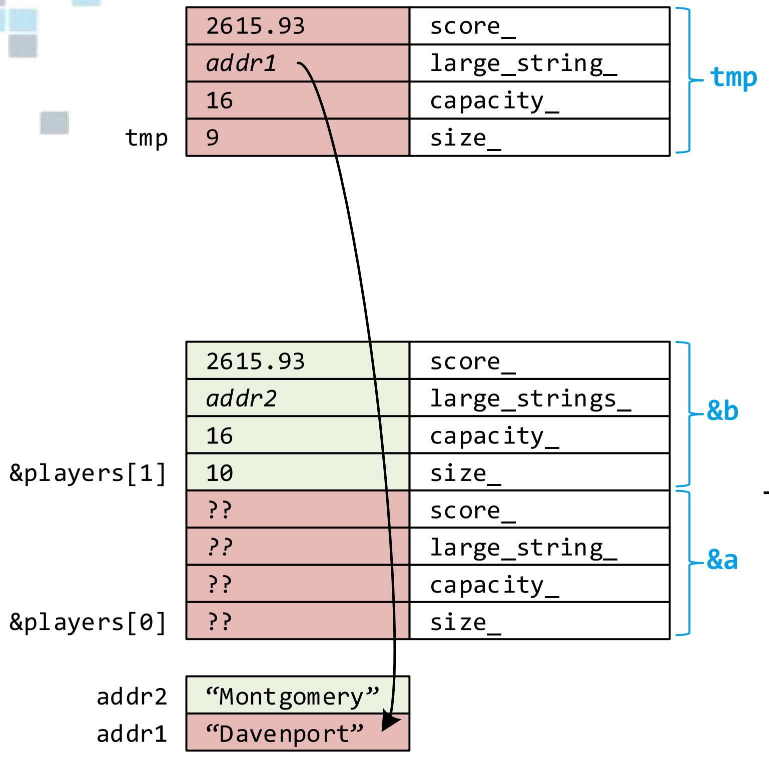
```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = a;
  a = b;
  b = tmp;
}
```

```
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*

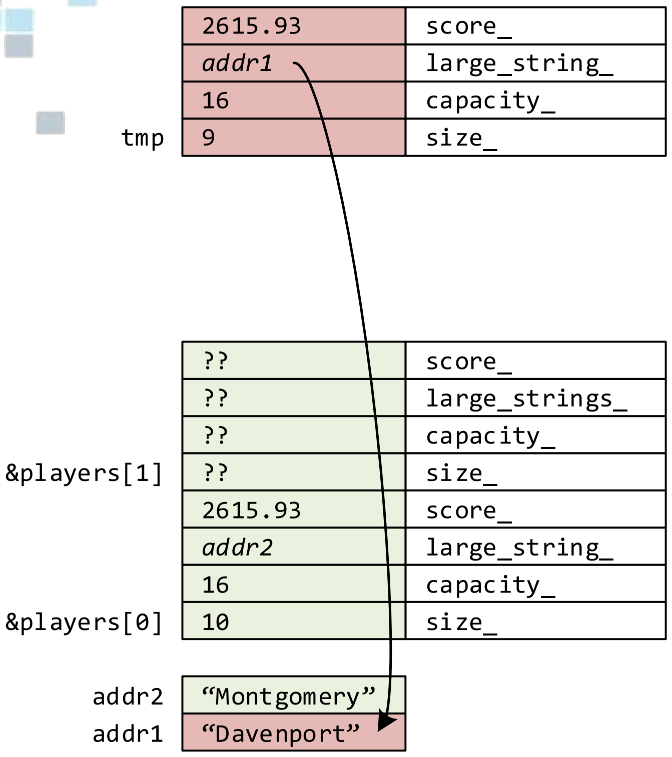
```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = std::move(a);
  a = b;
  b = tmp;
}
```

```
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*

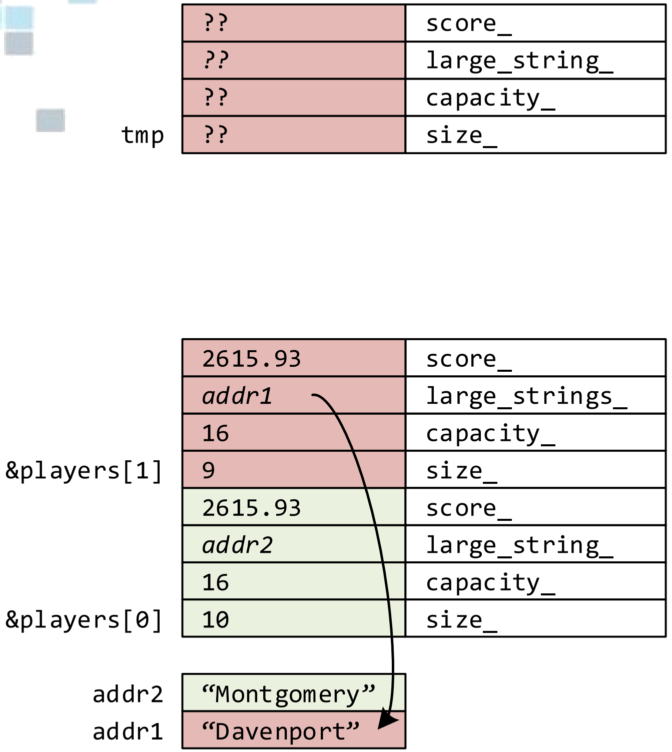
```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = std::move(a);
→a = std::move(b);
  b = std::move(tmp);
}
```

```
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)



Shallow copy of *tmp* into *b*

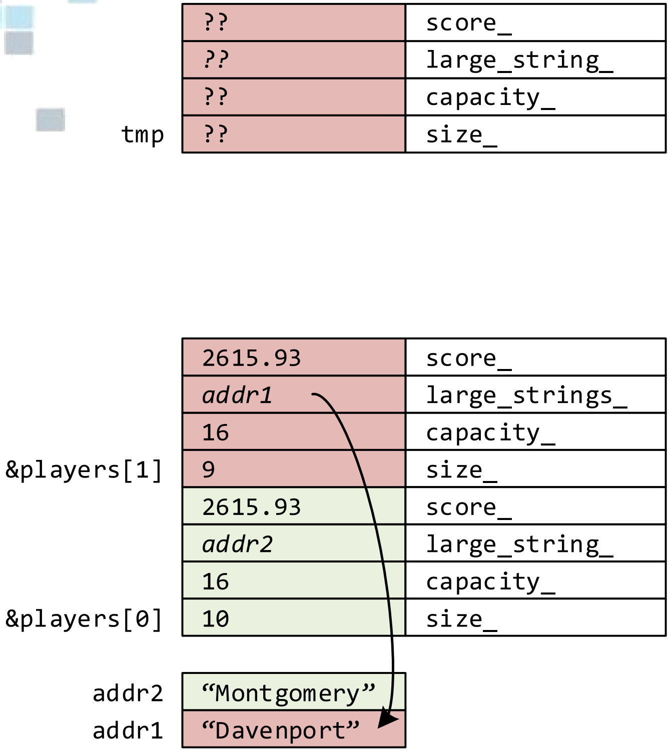
```
template<typename tpl_t>
void swap(tpl_t &a, tpl_t &b) {
  tpl_t tmp = std::move(a);
  a = std::move(b);
→b = std::move(tmp);
}
```

```
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 (4)



Destruction of *tmp* is immediate

```
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_;
  };
};
```

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   => not always generated !!
  - 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

```cpp
// 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;
```

```cpp
// 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;
  };
```