Day 02

- Overview of C++ Classes
  - Constructor/Destructor
  - Object Lifetime
- STL Algorithms
  - sort
- STL Containers
  - Dictionaries

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

1) Access the members (accessor) or
2) Change the state of the object (mutators or manipulators)

Class: OOP Foundation (5 bis)

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

Class Declaration
shape.h

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

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

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

Class Definition
shape.cpp

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 {
 public:
  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

Uniform Initialization in Practice 1

```
#include <iostream>
#include <cstdint>

int main() {
  int a{123456};
  int16_t v1{123456};
  int16_t v2 = {123456};
  int16_t v3 = 123456;
  int total = a + v1 + v2 + v3;

  std::cout
     << "Total: "
     << total
     << std::endl;
  return 0;
}
```

What is the output of this program ?

Uniform Initialization in Practice 2

```cpp
#include <iostream>
#include <cstdint>

int main() {
  int a{123456};
  int16_t v1{123456};
  int16_t v2 = {123456};
  int16_t v3 = 123456;
  int total = a + v1 + v2 + v3;

  std::cout
     << "Total: "
     << total
     << std::endl;
  return 0;
}
```

Narrowing is an error

```
16:04 cygwin> g++ -std=c++14 brace_init.cpp
brace_init.cpp: In function ‘int main()’:
brace_init.cpp:6:20: error: narrowing conversion of ‘123456’ from
‘int’ to ‘int16_t’ {aka ‘short int’} [-Wnarrowing]
    6 |   int16_t v1{123456};
      |                     ^
brace_init.cpp:7:23: error: narrowing conversion of ‘123456’ from
‘int’ to ‘int16_t’ {aka ‘short int’} [-Wnarrowing]
    7 |   int16_t v2 = {123456};
      |                       ^
brace_init.cpp:8:16: warning: overflow in conversion from ‘int’ to
 ‘int16_t’ {aka ‘short int’} changes value from ‘123456’ to ‘-761
’ [-Woverflow]
    8 |   int16_t v3 = 123456;
      |                ^~~~~~
```

Free Operators

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

  void print(std::ostream &strm) const {
    strm << "WxH = " << width_ << "x"
         << height_;
  }
  friend std::ostream &operator<<(
      std::ostream &strm,
      const Rect &rect)
  {
     rect.print(strm);
     return strm;
  }
};

int main() {
  Rect my_rect{2, 3};
  std::cout << my_rect << '\n';
}
```

Free Operator: it is a free function, not part of the class but defined within the class.
Overload some of the C++ operators (++, --, <<, ...)

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
  - Uniform initialization
  - Free Operators

POLYTECH
NICE-SOPHIA

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.exe 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)

```
1  struct Player {
2    string last_name_;
3    string names_;
4    double score_;
5    ...
6  };
```

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

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

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

Constructor / Destructor

```
1  struct Player {
2    string last_name_;
3    string names_;
4    double score_;
5    Player(const string& line);
6    ~Player() = default;
8    ...
9
```

Constructor
1) Memory acquisition
2) Initialize elements

Destructor
1) Destroy elements
2) Release memory

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 a.score_ > b.score_;
    });
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```

Possible Solution (2)

```
1   int main(int argc, char *argv[]) {
2     string file_name{argv[1]};
3     vector<Player> players;
4     std::ifstream fin(file_name, std::ios::in);
5     string line;
6     while (std::getline(fin, line)) {
7       Player player(line);
8       players.push_back(player);
9     }
10    std::sort(players.begin(), players.end(),
11              [](const Player &a, const Player &b) -> bool  {
12        return a.score_ > b.score_;
13      });
14    int idx = 0;
15    print_table_header();
16    for (auto &player : players) {
17      player.print_table_entry(++idx);
18    }
19    print_table_footer();
20  }
```

`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 a.score_ > b.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)

```
1   int main(int argc, char *argv[]) {
2     string file_name{argv[1]};
3     vector<Player> players;
4     std::ifstream fin(file_name, std::ios::in);
5     string line;
6     while (std::getline(fin, line)) {
7       Player player(line);
8       players.push_back(player);
9     }
10    std::sort(players.begin(), players.end(),
11              [](const Player &a, const Player &b) -> bool  {
12        return a.score_ > b.score_;
13      });
14    int idx = 0;
15    print_table_header();
16    for (auto &player : players) {
17      player.print_table_entry(++idx);
18    }
19    print_table_footer();
20  }
```

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

Range based loop

Possible Solution (4)

```
1   int main(int argc, char *argv[]) {
2     string file_name{argv[1]};
3     vector<Player> players;
4     std::ifstream fin(file_name, std::ios::in);
5     string line;
6     while (std::getline(fin, line)) {
7       Player player(line);
8       players.push_back(player);
9     }
10    std::sort(players.begin(), players.end(),
11              [](const Player &a, const Player &b) -> bool  {
12        return a.score_ > b.score_;
13      });
14    int idx = 0;
15    print_table_header();
16    for (auto &player : players) {
17      player.print_table_entry(++idx);
18    }
19    print_table_footer();
20  }
```

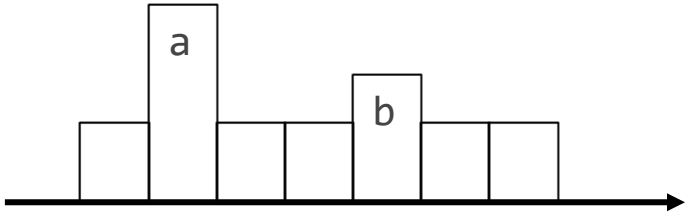
`print_table_entry()`

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

Sort and Lambda Function

```
std::vector<int> s{5, 7, 4, 2, 8, 6, 1, 9, 0, 3};
std::cout << "Before sort s = " << s << '\n';

std::ranges::sort(s);

std::cout << "After sort  s = " << s << std::endl;
```

```
wsl shell > ./test.exe
Before sort s = {5,7,4,2,8,6,1,9,0,3}
After sort  s = {0,1,2,3,4,5,6,7,8,9}
```



sort()

Default: sort in increasing order.
Rely on swap of 2 elements in the array.

IF (a > b) THEN swap(a, b)

Sort: Helper Function (1)

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

```
std::ranges::sort(players);
```

```
sorted_names.cpp:141:20: error: no match for call to
:vector<Player>&)'
  141 |    std::ranges::sort(players);
      |    ~~~~~~~~~~~~~~~~^~~~~~~~~
```

Do you expect sort to work?

Without specifying how Player objects are ordered, no way!

Sort: Helper Function (2)

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

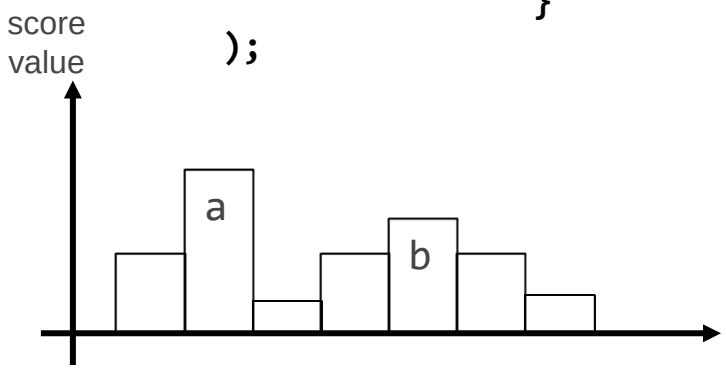
```
class Player {
...

  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?

Lambda Functions

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



When lambda returns `true`, the element *a* and *b* are correctly ordered, so NOT swapped.

`return a>b;` // for descending sort
`return a<b;` // for ascending sort

Lambda Functions

```
auto mycompare = [](const Player &a, const Player &b) -> bool
  {
    return a.score_ > b.score_;
  };


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

Question:
We want to sort by increasing or decreasing order based on a parameter?

Lambda Functions [2]

```
bool sort_ascending = false;
auto mycompare1 = [](const bool sort_ascending,
                     const Player &a,
                     const Player &b) -> bool  {
      return sort_ascending xor (a.score_ > b.score_);
    };

std::sort(players.begin(), players.end(), mycompare1);
```

```
/usr/lib/gcc/x86_64-pc-cygwin/10/include/c++/bits/predefined_ops.h:194:23
 error: no match for call to '(do_test2(int, char**)::<lambda(bool, const
Player&, const Player&)>) (Player&, Player&)'
  194 |   { return bool(_M_comp(*__it, __val)); }
      |                 ~~~~~~~^~~~~~~~~~~~~~
```

Lambda Capture [1]

Capture by value

```
bool sort_ascending = false;
auto mycompare2 = [sort_ascending](
                        const Player &a,
                        const Player &b) -> bool  {
      return sort_ascending xor (a.score_ > b.score_);
    };

std::sort(players.begin(), players.end(), mycompare2);
```

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

Lambda Capture [2]

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

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

```
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
```

What will be the results ?

Lambda Capture [3]

```
bool sort_ascending = true;
auto mycompare2 = [sort_ascending](
                        const Player &a,
                        const Player &b) -> bool  {
      return sort_ascending xor (a.score_ > b.score_);
    };

std::sort(players.begin(), players.end(), mycompare2);
```

1) Evaluation of the lambda expression

2) Invoking the lambda

```
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 value

- Occurs when the lambda is evaluated, not when it is used
- Captured values are stored in the lambda function
- Captured values are considered const
- Captured values are used during in the invocation of the lamdba

Update C++20 [1]

```
bool sort_ascending = true;
auto mycompare2 = [sort_ascending](
                        const Player &a,
                        const Player &b) -> bool  {
      return sort_ascending xor (a.score_ > b.score_);
    };

std::ranges::sort(players, mycompare2);
```

1) With ranges, no need to have begin(), end()

Sort: Helper Function (3)

```
#include <compare>

class Player {
...
auto operator<=>(const Player &other) const {
    if (score_ == other.score_) {
      return std::strong_ordering::equal;
    } else if (score_ > other.score_) {
      return std::strong_ordering::less;
    } else {
      return std::strong_ordering::greater;
    }
  }
...
};
```

Using < = >
The spaceship operator
The compiler will create all the compare function for you: <, <=, >, >=, ==, !=

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

or
  std::ranges::sort(players, std::less{});
```

POLYTECH
NICE-SOPHIA

Map and Unordered Map

Assignment #2b

- How to detect duplicated players in the input file, to filter them out?

data15.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
Noyland, Marianne, 1455.36
Hanna,Thelma,812.47
Irwin,Mara,2638.90
Hartman , Rosalie, 17301.72
Noyland,Marianne,1455.360
Glass,Dianne,3432.48
Miles,Jarrod,1984.21
Macdonald,Dion,1851.34
Hendricks,Patty,1353.11
Barron,Morgan,1283.09
```

```
shell> sorted_names --byscore data15.txt
+------+----------+------------+------------+
| Rank |    Score | Last Name  | First Name |
+------+----------+------------+------------+
|    1 | 17301.72 | Hartman    | Rosalie    |
|    2 |  3432.48 | Glass      | Dianne     |
|    3 |  2815.77 | Rubio      | Alfonso    |
|    4 |  2638.90 | Irwin      | Mara       |
|    5 |  2615.93 | Smith      | Linda      |
|    6 |  1990.52 | Davenport  | Darin      |
|    7 |  1984.21 | Miles      | Jarrod     |
|    8 |  1851.34 | Macdonald  | Dion       |
|    9 |  1455.36 | Noyland    | Marianne   |
|   10 |  1353.11 | Hendricks  | Patty      |
|   11 |  1321.13 | Faulkner   | Enrique    |
|   12 |  1283.09 | Barron     | Morgan     |
|   13 |  1181.31 | Wong       | Otis       |
|   14 |   863.93 | Romero     | Georgia    |
|   15 |   812.47 | Hanna      | Thelma     |
+------+----------+------------+------------+
```

Detection of Duplicate Player

```
1   int main(int argc, char *argv[]) {
2     string file_name{argv[1]};
3     vector<Player> players;
4     std::ifstream fin(file_name, std::ios::in);
5     string line;
6     while (std::getline(fin, line)) {
7       Player player(line);
8       players.push_back(player);
9     }
10    std::sort(players.begin(), players.end(),
11              [](const Player &a, const Player &b) -> bool  {
12        return a.score_ > b.score_;
13      });
14    int idx = 0;
15    print_table_header();
16    for (auto &player : players) {
17      player.print_table_entry(++idx);
18    }
19    print_table_footer();
20  }
```

Which line must be changed?

Map: Helper Function (1)

```
std::map<Player, uint32_t> 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 (2a)

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 {
...

  bool isless(const Player &other) {
    return score_ < other.score_;
  }

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

Map: Helper Function (1)

```
std::map<Player, uint32_t> map_of_players;
...

while(...) {
  Player player(line);
  auto iter = map_of_players.find(player);
  if (iter == map_of_players.end()) {
    map_of_players.insert({player, 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 (2b)

```
std::map<Player, uint32_t> map_of_players;
...

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

Are we happy with this implementation?

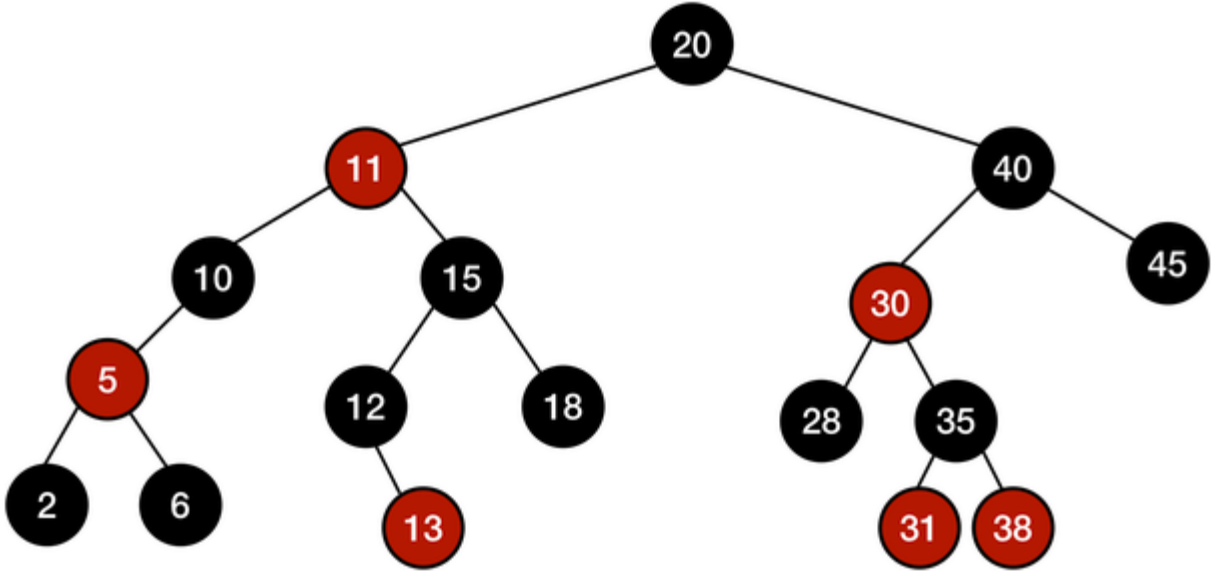
Map: Helper Function (2c)

```
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[std::move(player)] = lineno;
  } else {
    std::cout << "INFO: duplicate at line " << lineno << std::endl;
  }
  ++lineno
}
```

Are we happy with this implementation?

More on this topic later...

Map: Why < is needed?



https://en.wikipedia.org/wiki/Red%E2%80%93black_tree

Unordered Map: Helper Function (1a)

```
while(...) {
  Player player(line);



    std::cout << "INFO: duplicate at line " << lineno << std::endl;
  }
  ++lineno
}
```

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

```
                 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 (1b)

```
std::unordered_map<Player, uint32_t> 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
}
```

What is missing?

```
                 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 (1c)

```
std::unordered_map<Player, uint32_t> map_of_players;
...

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

What is missing, part 2?

```
sorted_names.move_constructor.cpp:346:31: error: use of deleted function 'std:
:unordered_map<_Key, _Tp, _Hash, _Pred, _Alloc>::unordered_map() [with _Key =
Player; _Tp = int; _Hash = std::hash<Player>; _Pred = std::equal_to<Player>; _
Alloc = std::allocator<std::pair<const Player, int> >]'
  346 |   unordered_map<Player, int > map_of_players;
      |                               ^~~~~~~~~~~~~~
```

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)

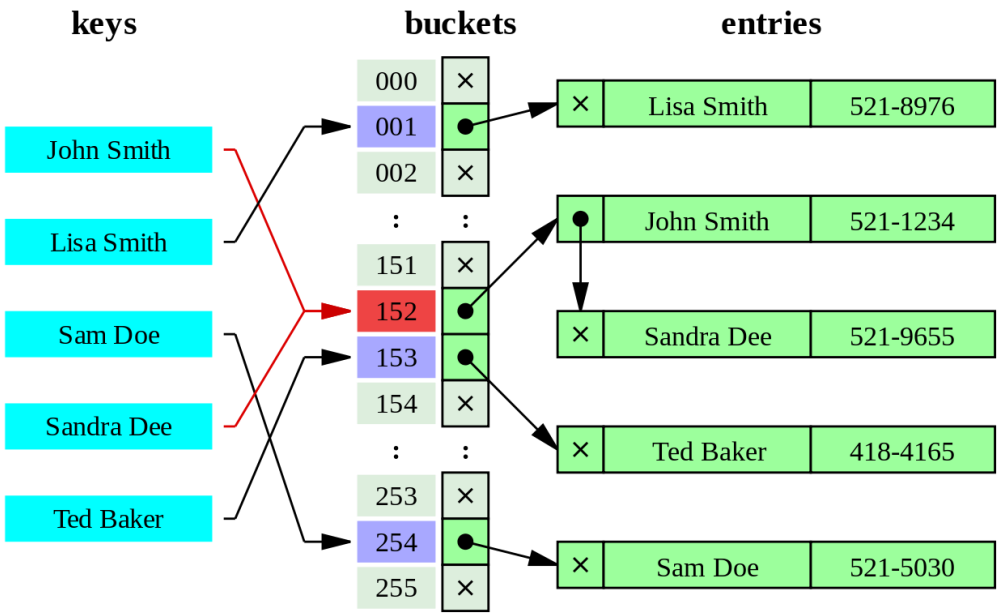
```
std::unordered_map<Player, int, Player::Hash> map_of_players;
```

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

You now have to define the hash() method

C++ Idiom: function object
An object which acts like a function

UMap: Why == and Hash are needed?



https://en.wikipedia.org/wiki/Hash_table

Code Optimization (1/2)

```
400    1706 *********************
500    2392 ******************************
600    3130 ****************************************
700    3684 ************************************************
0      3948 ****************************************************
```

What is the most efficient way to display the stars?

```
for (const auto bcount : buckets) {
  size_t total = ...
  for(size_t i = 0; i < total; ++i) {
    std::cout << "*";
  }
  ...
}
```

```
for (const auto bcount : buckets) {
  size_t total = ...

  std::cout << std::string(total, '*');

  ...
}
```

Code Optimization (2/2)

```
const string stars(MAX_STAR, '*');
const string_view sv{stars};

for (const auto bcount : buckets) {
  size_t total = ...
  std::cout << sv.substr(0, total);
  ...
}
```

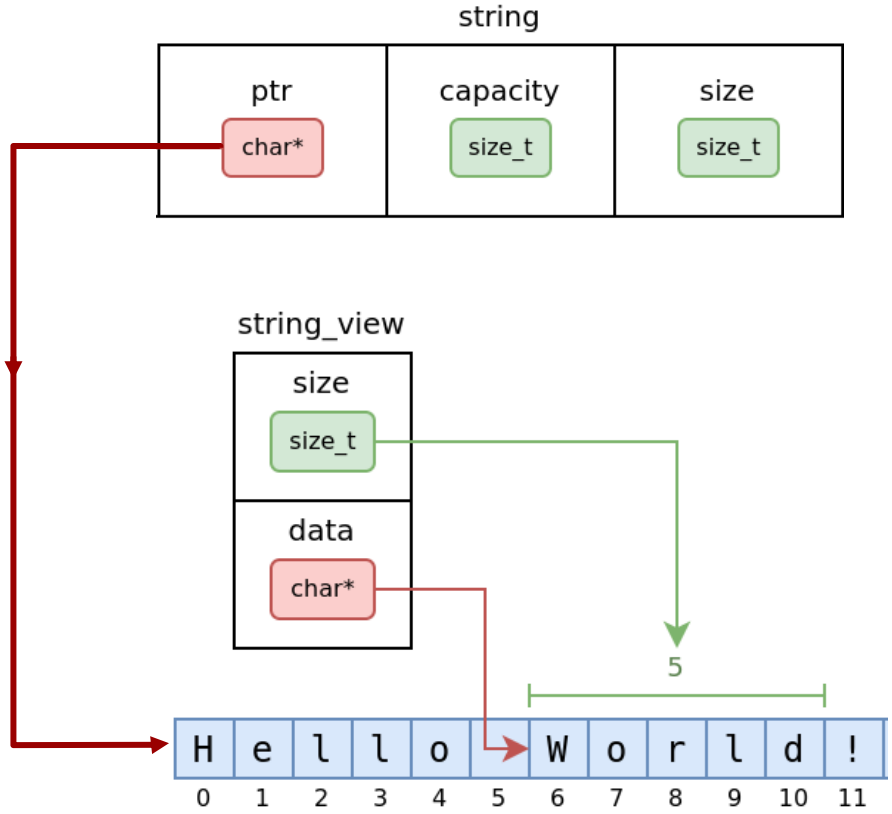
std::string
- created once
- expensive operation

std::string_view
- created multiple times
- cheap operation

```
for (const auto bcount : buckets) {
  size_t total = ...
  for(size_t i = 0; i < total; ++i) {
    std::cout << "*";
  }
  ...
}
```

```
for (const auto bcount : buckets) {
  size_t total = ...

  std::cout << std::string(total, '*');

  ...
}
```

std::string_view [C++17]



POLYTECH
NICE-SOPHIA

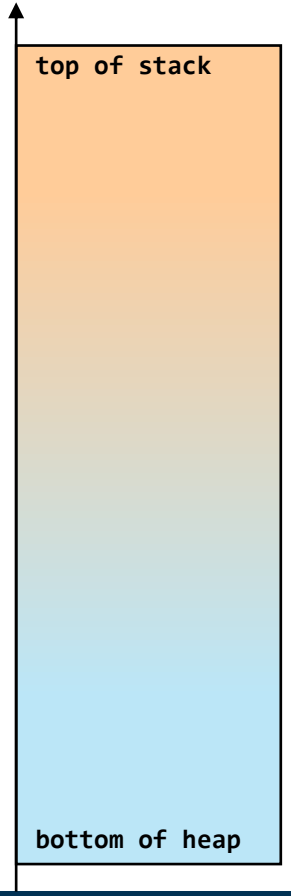
Object Life-Time

Memory: Heap or Stack

```
1   int main(int argc, char *argv[]) {
2     string file_name{argv[1]};
3     vector<Player> players;
4     std::ifstream fin(file_name, std::ios::in);
5     string line;
6     while (std::getline(fin, line)) {
7       Player player(line);
8       players.push_back(player);
9     }
10    std::sort(players.begin(), players.end(),
11              [](const Player &a, const Player &b) -> bool  {
12        return a.score_ > b.score_;
13      });
14    int idx = 0;
15    print_table_header();
16    for (auto &player : players) {
17      player.print_table_entry(++idx);
18    }
19    print_table_footer();
20  }
```

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

Memory Layout (1)

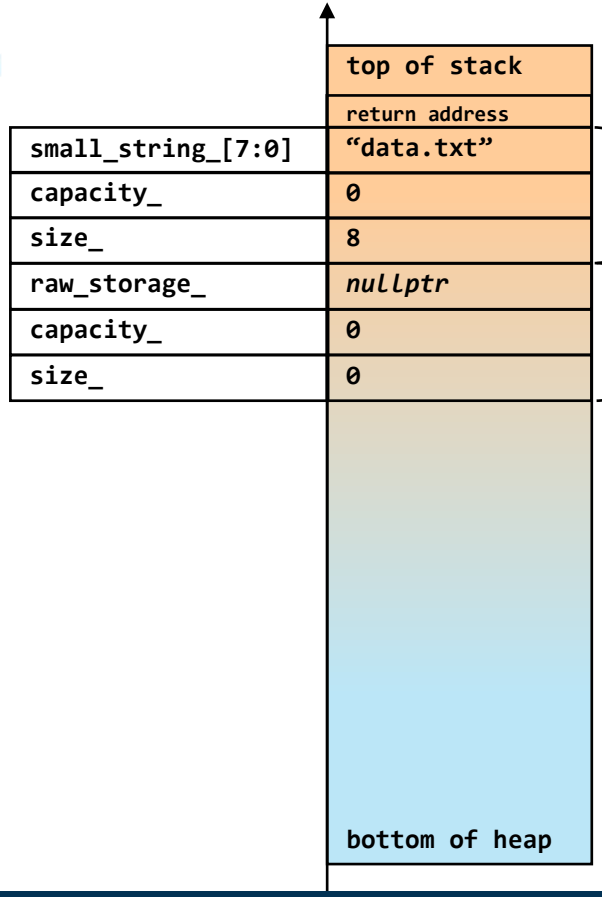



```
struct Player {
  string last_name_;
  vector<string> names_;
  double score_;
};
```

```
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)



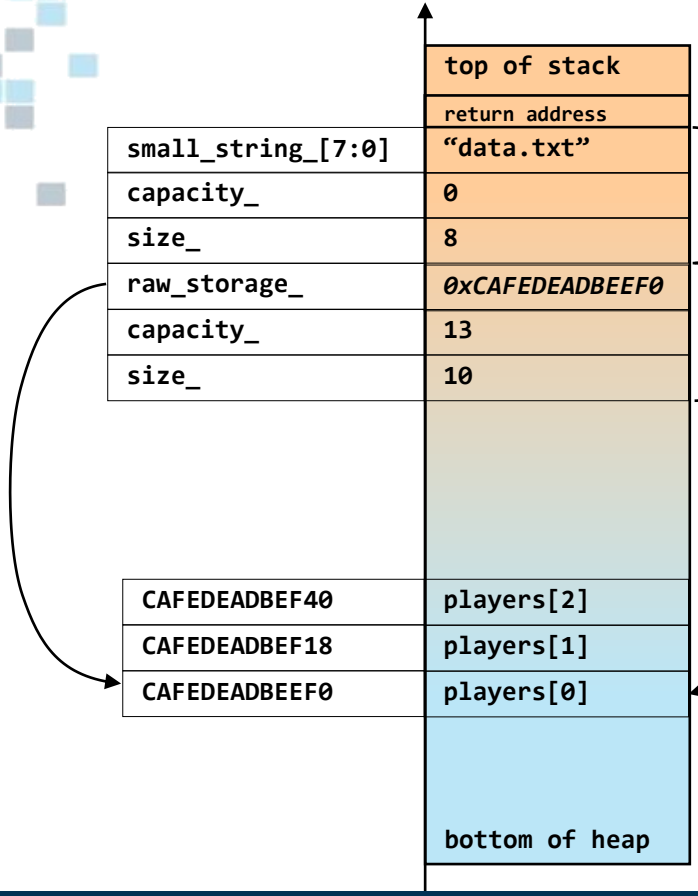

```
int main(int argc, char *argv[]) {
  string file_name(argv[1]);
  vector<Player> players;
```

```
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);
  }
```

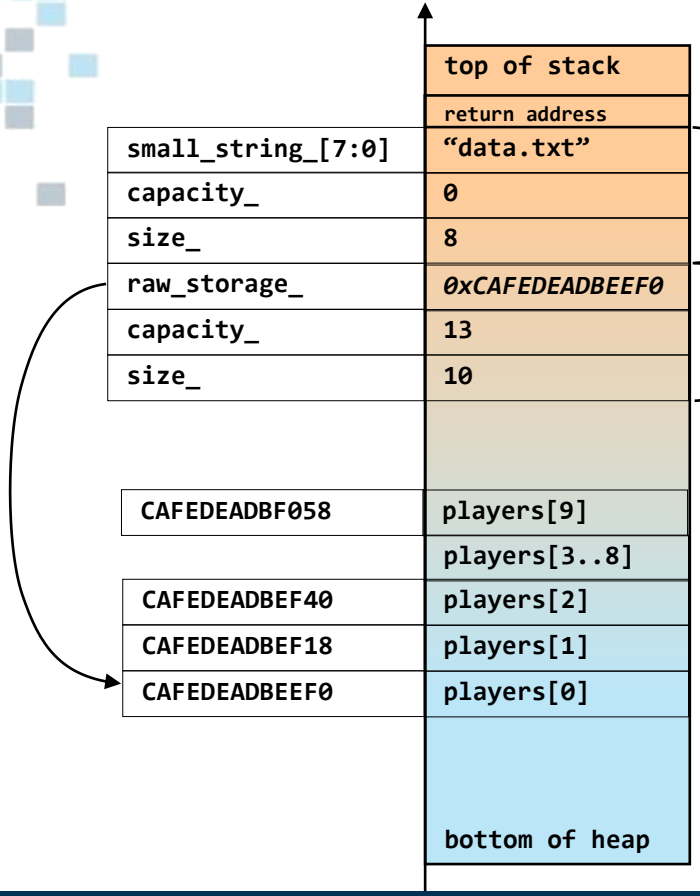
sizeof(player[i]) is 40

```
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 (2a)

```
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

...

+------+----------+-----------+-----------+-----------+-----------+
| 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    |           |
+------+----------+-----------+-----------+-----------+-----------+

...

shell>
```

Experiment with Destructor (2b)

```
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
 1  Destroying  Smith
 2  Destroying  Smith
 3  Destroying  Romero
 4  Destroying  Smith
 5  Destroying  Romero
 6  Destroying  Davenport
 7  Destroying  Rubio
 8  Destroying  Smith
 9  Destroying  Romero
10  Destroying  Davenport
11  Destroying  Rubio
12  Destroying  Wong
13  Destroying  Faulkner
14  Destroying  Nolan
15  Destroying  Hanna
16  Destroying  Smith
17  Destroying  Romero
18  Destroying  Davenport
19  Destroying  Rubio
20  Destroying  Wong
21  Destroying  Faulkner
22  Destroying  Nolan
23  Destroying  Hanna
24  Destroying  Irwin
25  Destroying  Hartman
26  +------+----------+-----------+-----------+-----------+-----------+
27  | Rank |    Score | Last Name | 1st Name  | 2nd Name  | 3rd Name  |
28  +------+----------+-----------+-----------+-----------+-----------+
29  |    1 |  2615.93 | Smith     | Linda     | Fay       |           |
30  |    2 |   863.93 | Romero    | Georgia   | Tania     |           |
31  |    3 |  1990.52 | Davenport | Darin     | Graham    | Gale      |
32  |    4 |  2815.77 | Rubio     | Alfonso   | Ulysses   | Vito      |
33  |    5 |  1181.31 | Wong      | Otis      | Cornell   | Gary      |
34  |    6 |  1321.13 | Faulkner  | Enrique   | Emmanuel  | Emilio    |
35  |    7 |   455.36 | Nolan     | Marianne  | Jenna     |           |
36  |    8 |   812.47 | Hanna     | Thelma    | Corine    | Juliet    |
37  |    9 |  2638.90 | Irwin     | Mara      | Elena     | Etta      |
38  |   10 | 17301.72 | Hartman   | Rosalie   | Carrie    |           |
39  +------+----------+-----------+-----------+-----------+-----------+
40  Destroying  Smith
41  Destroying  Romero
42  Destroying  Davenport
43  Destroying  Rubio
44  Destroying  Wong
45  Destroying  Faulkner
46  Destroying  Nolan
47  Destroying  Hanna
48  Destroying  Irwin
49  Destroying  Hartman
```

Experiment with Destructor (3)

```
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
 1  Destroying  Smith
 2  Destroying  Romero
 3  Destroying  Davenport
 4  Destroying  Rubio
 5  Destroying  Wong
 6  Destroying  Faulkner
 7  Destroying  Nolan
 8  Destroying  Hanna
 9  Destroying  Irwin
10  Destroying  Hartman
11  +------+----------+-----------+-----------+-----------+-----------+
12  | Rank |    Score | Last Name | 1st Name  | 2nd Name  | 3rd Name  |
13  +------+----------+-----------+-----------+-----------+-----------+
14  |    1 |  2615.93 | Smith     | Linda     | Fay       |           |
15  |    2 |   863.93 | Romero    | Georgia   | Tania     |           |
16  |    3 |  1990.52 | Davenport | Darin     | Graham    | Gale      |
17  |    4 |  2815.77 | Rubio     | Alfonso   | Ulysses   | Vito      |
18  |    5 |  1181.31 | Wong      | Otis      | Cornell   | Gary      |
19  |    6 |  1321.13 | Faulkner  | Enrique   | Emmanuel  | Emilio    |
20  |    7 |   455.36 | Nolan     | Marianne  | Jenna     |           |
21  |    8 |   812.47 | Hanna     | Thelma    | Corine    | Juliet    |
22  |    9 |  2638.90 | Irwin     | Mara      | Elena     | Etta      |
23  |   10 | 17301.72 | Hartman   | Rosalie   | Carrie    |           |
24  +------+----------+-----------+-----------+-----------+-----------+
25  Destroying  Smith
26  Destroying  Romero
27  Destroying  Davenport
28  Destroying  Rubio
29  Destroying  Wong
30  Destroying  Faulkner
31  Destroying  Nolan
32  Destroying  Hanna
33  Destroying  Irwin
34  Destroying  Hartman
```

Experiment with Destructor (4)

```
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
 1  Destroying  Smith
 2  Destroying  Romero
 3  Destroying  Davenport
 4  Destroying  Rubio
 5  Destroying  Wong
 6  Destroying  Faulkner
 7  Destroying  Nolan
 8  Destroying  Hanna
 9  Destroying  Irwin
10  Destroying  Hartman
11
12  | Rank |     Score | Last Name | 1st Name  | 2nd Name  | 3rd Name  |
13  +------+----------+-----------+-----------+-----------+-----------+
14  |    1 |   2615.93 | Smith     | Linda     | Fay       |           |
15  |    2 |    863.93 | Romero    | Georgia   | Tania     |           |
16  |    3 |   1990.52 | Davenport | Darin     | Graham    | Gale      |
17  |    4 |   2815.77 | Rubio     | Alfonso   | Ulysses   | Vito      |
18  |    5 |   1181.31 | Wong      | Otis      | Cornell   | Gary      |
19  |    6 |   1321.13 | Faulkner  | Enrique   | Emmanuel  | Emilio    |
20  |    7 |    455.36 | Nolan     | Marianne  | Jenna     |           |
21  |    8 |    812.47 | Hanna     | Thelma    | Corine    | Juliet    |
22  |    9 |   2638.90 | Irwin     | Mara      | Elena     | Etta      |
23  |   10 |  17301.72 | Hartman   | Rosalie   | Carrie    |           |
24  +------+----------+-----------+-----------+-----------+-----------+
25  Destroying  Smith
26  Destroying  Romero
27  Destroying  Davenport
28  Destroying  Rubio
29  Destroying  Wong
30  Destroying  Faulkner
31  Destroying  Nolan
32  Destroying  Hanna
33  Destroying  Irwin
34  Destroying  Hartman
```

Experiment with Destructor (5)

```
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
```