ETH zürich

Exercise Session W10

Computer Science (CSE & CBB & Statistics) – AS 23

Overview

Today's Agenda

Feedback on **code** expert
Important Update regarding Feedback
Objectives
Classes and Operator Overloading
Exercise "Tribool"
Iterators
Outro

n.ethz.ch/~agavranovic

1. Feedback on **code** expert

General things regarding **code** expert

General things regarding **code** expert

- All of the submissions have increased a lot in quality!

General things regarding **code** expert

- All of the submissions have increased a lot in quality!
- Code looks a lot more legible and structured

General things regarding **code** expert

- All of the submissions have increased a lot in quality!
- Code looks a lot more legible and structured
- You all should study the master solution to E6:T1 "Perpetual Calendar"
  - not because most of the submissions were bad (they were very good)
  - but because it's a good exercise in breaking down a difficult problem into (many) more easier ones
  - and it contains a lot of handy ways of writing code

2. Important Update regarding Feedback

Important Update regarding Feedback

Due to time constraints and a massive correction backlog, I have to drastically reduce the feedback I'm giving for coding exercises on **code** expert . This means the following for you:

Important Update regarding Feedback

Due to time constraints and a massive correction backlog, I have to drastically reduce the feedback I'm giving for coding exercises on **code** expert . This means the following for you:

- Unless you specifically ask for feedback (as a comment at the very top of the submitted code), I will not give you (detailed) feedback

Important Update regarding Feedback

Due to time constraints and a massive correction backlog, I have to drastically reduce the feedback I'm giving for coding exercises on **code** expert . This means the following for you:

- Unless you specifically ask for feedback (as a comment at the very top of the submitted code), I will not give you (detailed) feedback
  - Something like `// FEEDBACK PLEASE` in the first couple lines is enough
  - You're still encouraged to submit solutions and questions
  - Text tasks will almost always still get full feedback

Important Update regarding Feedback

Due to time constraints and a massive correction backlog, I have to drastically reduce the feedback I'm giving for coding exercises on **code** expert . This means the following for you:

- Unless you specifically ask for feedback (as a comment at the very top of the submitted code), I will not give you (detailed) feedback
  - Something like `// FEEDBACK PLEASE` in the first couple lines is enough
  - You're still encouraged to submit solutions and questions
  - Text tasks will almost always still get full feedback
- The TA and Autograder points will still be awarded like before
- No feedback indicates a good submission anyway
- Some unsolicited feedback might still be provided if deemed necessary

Questions?

3. Objectives

Objectives

- ☐ be able to define own classes
- ☐ be able to overload operators for defined classes
- ☐ be able to use iterators

4. Classes and Operator Overloading

Differentiating between functions

Differentiating between functions

It is possible for two functions to have the same name, as long as the compiler has another way to differentiate between them. The only possible criteria for distinguishing functions are:

Differentiating between functions

It is possible for two functions to have the same name, as long as the compiler has another way to differentiate between them. The only possible criteria for distinguishing functions are:

- Names of the functions
- Numbers of function arguments
- Types of function arguments

Putting the *Fun* in *Function* I

Will this produce a compiler error?

```
int fun1(const int a){
    // ...
}

int fun1(const int a, const int b){
    // ...
}
```

Putting the *Fun* in *Function* I

Will this produce a compiler error?

```
int fun1(const int a){
    // ...
}

int fun1(const int a, const int b){
    // ...
}
```

Answer: No, because

Putting the *Fun* in *Function* I

Will this produce a compiler error?

```
int fun1(const int a){
    // ...
}

int fun1(const int a, const int b){
    // ...
}
```

Answer: No, because the two functions have a different numbers of arguments (1 vs 2)

Putting the *Fun* in *Function* II

Will this produce a compiler error?

```
int fun2(const int a){
    // ...
}

int fun2(const float a){
    // ...
}
```

Putting the *Fun* in *Function* II

Will this produce a compiler error?

```
int fun2(const int a){
    // ...
}

int fun2(const float a){
    // ...
}
```

Answer: No, because

Putting the *Fun* in *Function* II

Will this produce a compiler error?

```
int fun2(const int a){
    // ...
}

int fun2(const float a){
    // ...
}
```

Answer: No, because the two functions have a different parameter types (`int` vs `float`)

Putting the *Fun* in *Function* III

Will this produce a compiler error?

```
int fun3(const int a){
    // ...
}

int fun3(const int b){
    // ...
}
```

Putting the *Fun* in *Function* III

Will this produce a compiler error?

```
int fun3(const int a){
    // ...
}

int fun3(const int b){
    // ...
}
```

Answer: Yes, because

Putting the *Fun* in *Function* III

Will this produce a compiler error?

```
int fun3(const int a){
    // ...
}

int fun3(const int b){
    // ...
}
```

Answer: Yes, because the two functions don't have different numbers or types of arguments

Putting the *Fun* in *Function* III

Will this produce a compiler error?

```
int fun3(const int a){
    // ...
}

int fun3(const int b){
    // ...
}
```

Answer: Yes, because the two functions don't have different numbers or types of arguments

Notice: The names of the function parameters are irrelevant to the compiler!

Putting the *Fun* in *Function* IV

Will this produce a compiler error?

```
int fun4(const int a){
    // ...
}

double fun4(const int a){
    // ...
}
```

Putting the *Fun* in *Function* IV

Will this produce a compiler error?

```
int fun4(const int a){
    // ...
}

double fun4(const int a){
    // ...
}
```

Answer: Yes, because

Putting the *Fun* in *Function* IV

Will this produce a compiler error?

```
int fun4(const int a){
    // ...
}

double fun4(const int a){
    // ...
}
```

Answer: Yes, because the two functions don't have different numbers or types of arguments

Putting the *Fun* in *Function* IV

Will this produce a compiler error?

```
int fun4(const int a){
    // ...
}

double fun4(const int a){
    // ...
}
```

Answer: Yes, because the two functions don't have different numbers or types of arguments

Notice: The return types of the functions are irrelevant to the compiler!

Putting the *Fun* in *Function* V

Will this produce a compiler error?

```
int fun5(const int a){
    // ...
}

int fun6(const int a){
    // ...
}
```

Putting the *Fun* in *Function* V

Will this produce a compiler error?

```
int fun5(const int a){
    // ...
}

int fun6(const int a){
    // ...
}
```

Answer: No, because

Putting the *Fun* in *Function* V

Will this produce a compiler error?

```
int fun5(const int a){
    // ...
}

int fun6(const int a){
    // ...
}
```

Answer: No, because the two functions carry different names

Just my Type

```
void out(const int i){
  std::cout << i << " (int)\n";
}
void out(const double i){
  std::cout << i << " (double)\n";
}

int main(){
  out(3.5);
  out(2);
  out(2.0);
  out(0);
  out(0.0);
  return 0;
}
```

What's the output going to be?

Just my Type

```
void out(const int i){
  std::cout << i << " (int)\n";
}
void out(const double i){
  std::cout << i << " (double)\n";
}

int main(){
  out(3.5);
  out(2);
  out(2.0);
  out(0);
  out(0.0);
  return 0;
}
```

What's the output going to be?

- `3.5 (double)`

Just my Type

```
void out(const int i){
  std::cout << i << " (int)\n";
}
void out(const double i){
  std::cout << i << " (double)\n";
}

int main(){
  out(3.5);
  out(2);
  out(2.0);
  out(0);
  out(0.0);
  return 0;
}
```

What's the output going to be?

- `3.5 (double)`
- `2 (int)`

Just my Type

```
void out(const int i){
  std::cout << i << " (int)\n";
}
void out(const double i){
  std::cout << i << " (double)\n";
}

int main(){
  out(3.5);
  out(2);
  out(2.0);
  out(0);
  out(0.0);
  return 0;
}
```

What's the output going to be?

- 3.5 (double)
- 2 (int)
- 2 (double)

Just my Type

```
void out(const int i){
  std::cout << i << " (int)\n";
}
void out(const double i){
  std::cout << i << " (double)\n";
}

int main(){
  out(3.5);
  out(2);
  out(2.0);
  out(0);
  out(0.0);
  return 0;
}
```

What's the output going to be?

- 3.5 (double)
- 2 (int)
- 2 (double)
- 0 (int)

Just my Type

```
void out(const int i){
  std::cout << i << " (int)\n";
}
void out(const double i){
  std::cout << i << " (double)\n";
}

int main(){
  out(3.5);
  out(2);
  out(2.0);
  out(0);
  out(0.0);
  return 0;
}
```

What's the output going to be?

- 3.5 (double)
- 2 (int)
- 2 (double)
- 0 (int)
- 0 (double)

Questions?

5. Exercise "Tribool"

Tribool as a Logic Object

NOT(A)

A	¬A
F	T
U	U
T	F

AND(A,B)

A ∧ B		B		
		F	U	T
A	F	F	F	F
	U	F	U	U
	T	F	U	T

OR(A,B)

A ∨ B		B		
		F	U	T
A	F	F	U	T
	U	U	U	T
	T	T	T	T

F = FALSE, U = UNKNOWN, T = TRUE

Tribool as a Logic Object

NOT(A)

A	¬A
F	T
U	U
T	F

AND(A,B)

A ∧ B		B		
		F	U	T
A	F	F	F	F
	U	F	U	U
	T	F	U	T

OR(A,B)

A ∨ B		B		
		F	U	T
A	F	F	U	T
	U	U	U	T
	T	T	T	T

F = FALSE, U = UNKNOWN, T = TRUE

operator overloading

A && B

- How could we implement this in C++?
- What operations and values do we need?

Exercise "Tribool"

```
class Tribool {
private:
    // 0 means false, 1 means unknown, 2 means true.
    unsigned int value;  // INV: value in {0, 1, 2}.
public:
    // ...
};
```

Exercise "Tribool"

```
class Tribool {
private:
    // ...
public:
    // Constructor 1 (passing a numerical value)
    // PRE: value in {0, 1, 2}.
    // POST: tribool false if value was 0, unknown if 1, and true if 2.
    Tribool(unsigned int value_int);
    // TODO: add the definition in tribool.cpp

    // Constructor 2 (passing a string value)
    // PRE: value in {"true", "false", "unknown"}.
    // POST: tribool false, true or unknown according to the input.
    // TODO: add declaration here and the definition in tribool.cpp
    // ...
};
```

Exercise "Tribool"

```
class Tribool {
private:
    // ...
public:
    // ...
    // Member function string()
    // POST: Return the value as string
    // TODO: add declaration here and the definition in tribool.cpp

    // Operator && overloading
    // POST: returns this AND other
    // TODO: add declaration here and the definition in tribool.cpp
};
```

Exercise "Tribool"

Where do we even start?

1. First (`int`) Constructor
2. Second (`std::string`) Constructor
3. Implement `string()` method
4. Implement logical AND as an operator

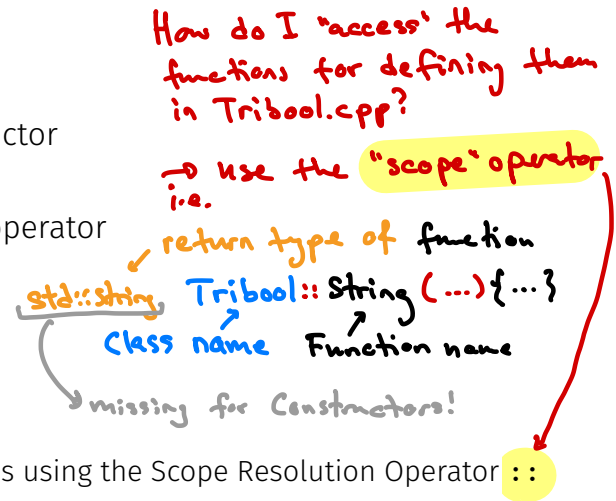
Exercise "Tribool"

Where do we even start?

1. First (`int`) Constructor
2. Second (`std::string`) Constructor
3. Implement `string()` method
4. Implement logical AND as an operator

Where to put all this?

- Declarations into `Tribool.h`
- Definitions into `Tribool.cpp`
  - Using Out-of-Class definitions using the Scope Resolution Operator `::`

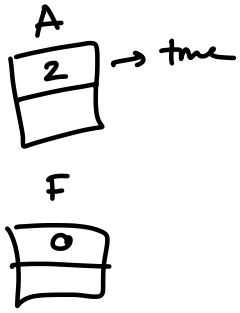


Let’s Code (together)!

- Open "Tribool" on **code** expert

Let’s Code (together)!

```
bool B = false;
Tribool A(2);
Tribool F(false);
```



- Open "Tribool" on **code** expert
- We’re doing a live coding session

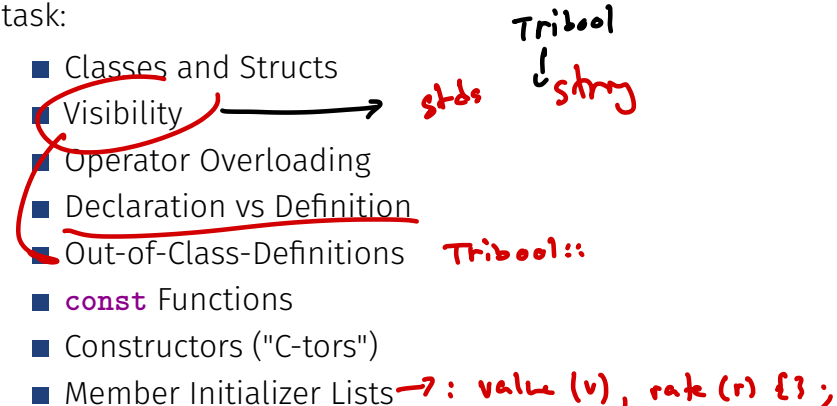
Exercise "Tribool" Concepts

We encountered the following concepts and keywords while solving this task:

Exercise "Tribool" Concepts

We encountered the following concepts and keywords while solving this task:

- Classes and Structs
- Visibility
- Operator Overloading
- Declaration vs Definition
- Out-of-Class-Definitions
- `const` Functions
- Constructors ("C-tors")
- Member Initializer Lists
- …



Questions?

6. Iterators

What even are Iterators?

- Iterators are used iterate (or move) through elements in a Container

[1] https://en.cppreference.com/w/cpp/container

What even are Iterators?

- Iterators are used iterate (or move) through elements in a Container
- What are Containers then?
  - Containers are objects that are used to store collections of elements
  - Some common C++containers include

[1] `https://en.cppreference.com/w/cpp/container`

What even are Iterators?

- Iterators are used iterate (or move) through elements in a Container
- What are Containers then?
  - Containers are objects that are used to store collections of elements
  - Some common C++containers include
    - `std::vector`
    - `std::set`
    - `std::list`

[1] `https://en.cppreference.com/w/cpp/container`

What even are Iterators?

- Iterators are used iterate (or move) through elements in a Container
- What are Containers then?
  - Containers are objects that are used to store collections of elements
  - Some common C++containers include
    - `std::vector`
    - `std::set`
    - `std::list`
  - A complete list of the containers of the C++-standard library can be found here[1]

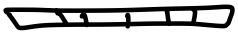
[1] `https://en.cppreference.com/w/cpp/container`

Using Iterators on Containers

Very easy and by design always the same!

Given: a container named `c`

std::vector<int> c = {...};



[2] PTE: Past-the-End

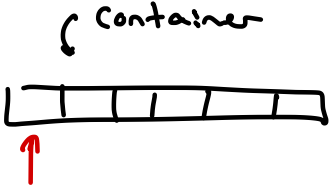
Using Iterators on Containers

Very easy and by design always the same!

Given: a container named `C`

- `it = C.begin()`

auto it = C.begin();



[2] PTE: Past-the-End

Using Iterators on Containers

Very easy and by design always the same!

Given: a container named `C`

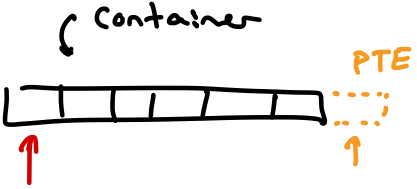
- `it = C.begin()`
  Iterator pointing to first element
- `it = C.end()`

[2] PTE: Past-the-End

Using Iterators on Containers

Very easy and by design always the same!

Given: a container named `C`

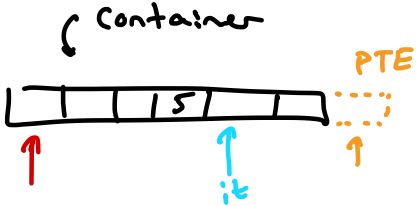


- `it = C.begin()`
  Iterator pointing to first element
- `it = C.end()`
  Iterator pointing to first element *past the end*[2]
- `*it`

[2] PTE: Past-the-End

Using Iterators on Containers

Very easy and by design always the same!
Given: a container named `C`



- `it = C.begin()`
  Iterator pointing to first element
- `it = C.end()`
  Iterator pointing to first element *past the end*[2]
- `*it` 5
  Access (and maybe modify) current element
- `++it`

[2] PTE: Past-the-End

Using Iterators on Containers

Very easy and by design always the same!

Given: a container named `C`

- `it = C.begin()`
  Iterator pointing to first element
- `it = C.end()`
  Iterator pointing to first element *past the end*[2]
- `*it`
  Access (and maybe modify) current element
- `++it`
  Advance iterator by one element

[2] PTE: Past-the-End

Exercise "Find Max"

Exercise "Find Max"

```
// PRE:  i < j <= v.size()
// POST: Returns the greatest element of all elements
//       with indices between i and j (excluding j)
unsigned int find_max(const std::vector<unsigned int>& v,
                      unsigned int i,
                      unsigned int j){
  unsigned int max_value = 0;

  for (; i < j; ++i) {
    if (max_value < v.at(i)) {
      max_value = v.at(i);
    }
  }

  return max_value;
}
```

Exercise "Find Max"

Exercise "Find Max"

- Open "Find Max" on **code** expert

Exercise "Find Max"

- Open "Find Max" on **code** expert
- Think about how you would approach the problem with pen and paper

Exercise "Find Max"

- Open "Find Max" on **code** expert
- Think about how you would approach the problem with pen and paper
- Implement a solution (optionally in groups)

Exercise "Find Max" (Solution)

Exercise "Find Max" (Solution)

```
// PRE:  (begin < end) && (begin and end must be valid iterators)
// POST: Return the greatest element in the range [begin, end)
unsigned int find_max(std::vector<unsigned int>::iterator begin,
                      std::vector<unsigned int>::iterator end) {
  unsigned int max_value = 0;

  for(; begin != end; ++begin) {
    if (max_value < *begin) {
      max_value = *begin;
    }
  }

  return max_value;
}
```

Questions?

The `algorithm` Library

- Surely somebody smarter already implemented all the common algorithms for us, right?

The `algorithm` Library

- Surely somebody smarter already implemented all the common algorithms for us, right?
- Yes! The `algorithm` library

The `algorithm` Library

std::max(...)

- Surely somebody smarter already implemented all the common algorithms for us, right?
- Yes! The `algorithm` library
- These functions are designed to work with various containers like vectors, arrays, lists, etc., and help in performing tasks efficiently without the need to write the algorithms from scratch each time

The `algorithm` Library

- Surely somebody smarter already implemented all the common algorithms for us, right?
- Yes! The `algorithm` library
- These functions are designed to work with various containers like vectors, arrays, lists, etc., and help in performing tasks efficiently without the need to write the algorithms from scratch each time
- Don't forget to `#include <algorithm>`

Exercise "The algorithm Library"

Exercise "The algorithm Library"

- Open "The algorithm Library" on **code** expert

Exercise "The algorithm Library"

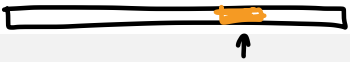
- Open "The algorithm Library" on **code** expert
- Think about how you would approach the problem

Exercise "The algorithm Library"

- Open "The algorithm Library" on **code** expert
- Think about how you would approach the problem
- Implement a solution (optionally in groups)

Exercise "The algorithm Library" (Solution)

Exercise "The algorithm Library" (Solution)



```
// ...

int largest_element = *std::max_element(vec.begin(), vec.end());

// ...

std::sort(vec.begin(), vec.end());

// ...
```

Questions?

7. Outro

General Questions?

Till next time!

Cheers!