Department I - C Plus Plus

Modern and Lucid C++ Advanced for Professional Programmers

Week 1 – C Plus Plus Recap

Prof. Peter Sommerlad / Thomas Corbat
Rapperswil, 21.02.2019
FS2019

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

IFS
INSTITUTE FOR
SOFTWARE

Basics Language Features

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

IFS
INSTITUTE FOR
SOFTWARE

- **Semantics of Values and (lvalue) References**

```
void foo(int p) {
  p = 23;
}
...
int i = 5;
foo(i);
```

```
void foo(int & p) {
  p = 23;
}
...
int i = 5;
foo(i);
```

- **What is the value of `i` after `foo(i)`?**

- **We will look at rvalue References too**

```
void foo(int && p) {
  ...
}
...

foo(5);
```

- **With the `const` keyword you have a powerful tool to improve your code**
- **Where can we place the `const` keyword?**
  - Types of Variables/Parameters/Return Types
  - Member Functions (`this` pointer)
  - Pointers

```
Spaceship vulture{};
vulture.load(Stuff{});

Spaceship const eagle{};
out << eagle.built();
```

```
struct Spaceship {
  void load(Stuff const &) {}
  Date built() const {}
};
```

- **`load()` modifies the Spaceship thus it must not be `const`**
- **`built()` only queries the date, does not modify Spaceship and therefore should be `const`**

- **Composite types (Classes)**

```
struct Telephone {
  void dial(PhoneNumber);
private:
  Log<Calls> call_log;
};
```

- **Enums**

```
enum class Gender {
  female, male, apache
  //50 more
};
```

- **Lambdas**

```
void catch_me_if_you_can() {
  Criminal abagnale{"Frank"s};
  auto hanratty = [abagnale] {
    offer_deal(abagnale);
  };
}
```

- **Functions**

```
Meal cookBreakfast(Kitchen & kitchen) {
  auto frying_pan = kitchen.getPan();
  kitchen.sink().wash(frying_pan);
  auto oven = kitchen.oven();
  oven.put(frying_pan);
  oven.turnOn(Oven::Temperature::hot);
  frying_pan.add(Egg{});
  frying_pan.add(Bacon{});
  return frying_pan.slightlyBurntFood();
}
```

- **Namespaces**
  - Named

```
namespace Labyrinth {
  Minotaur asterion{};
}
```

  - Global

```
//Global namespace
int main() {}
```

  - Anonymous

```
namespace {
}
```

  - Inline

```
namespace MyLib {
  inline namespace V1 {
    struct WillImprove {
    };
  }
}
```

- **Variables**
  - Local

```
void tour_de_rappi() {
  Restaurant baeren{};
}
```

  - Global

```
Climate warming{};
```

  - Member

```
class Ship {
  int number_of_leaks{};
};
```

  - Can all be `static`, what would each mean?

- **You can throw everying in C++**
- **`try/catch`**
  - but no `finally` – is that a problem?
- **Catch clauses tried from top to bottom**
- **Exception wildcard ellipsis (`...`)**
- **Good style to**
  - Throw by value
  - Catch by `const &`

```
void go_home_from_lecture() try {
  try {
    waitForBell();
  } catch (FellAsleepException const & e) {
    wakeUp();
    wonderWhyTheRoomIsDarkAndEverybodyIsGone();
  }
  packYourStuff();
  try {
    getUp();
  } catch (LegGotPinsAndNeedlesException const & e) {
    dieOfPain() || stayMotionless(TIME_TO_RECOVER);
  }
  leaveHSR();
  gotoTheStation();
  //...
} catch(...) {
  //Did not expect that. I don't know what it is.
  wonderAboutException();
  //Let somebody else care...
  throw;
}
```

- **Operators for primitive types are specified in the language (E.g +, -, /,...)**
  - Caveat: Only for expressions with operands of the same type

```
int intValue1 = 15;
int intValue2 = 24;
auto intIntSum = intValue1 + intValue2;

long longValue1 = 11l;
auto longIntSum = longValue1 + intValue1;

double doubleValue = 128.0;
auto doubleIntSum = doubleValue + intValue1;

unsigned unsignedValue = 99u;
auto unsignedIntSum = unsignedValue + intValue1;
```

- **Negative/positive overflow of unsigned integers is defined**
  - However, it might feature unexpected behavior

```
int zeroIndex = 0;
for (unsigned size = 5; size <= 10; size--) {
  if (zeroIndex <= size - 1) {
    std::cout << "access with 0 is ok for size " << size << '\n';
  } else {
    std::cout << "access with 0 is not ok for size " << size << '\n';
  }
}
```

- **Program output:**

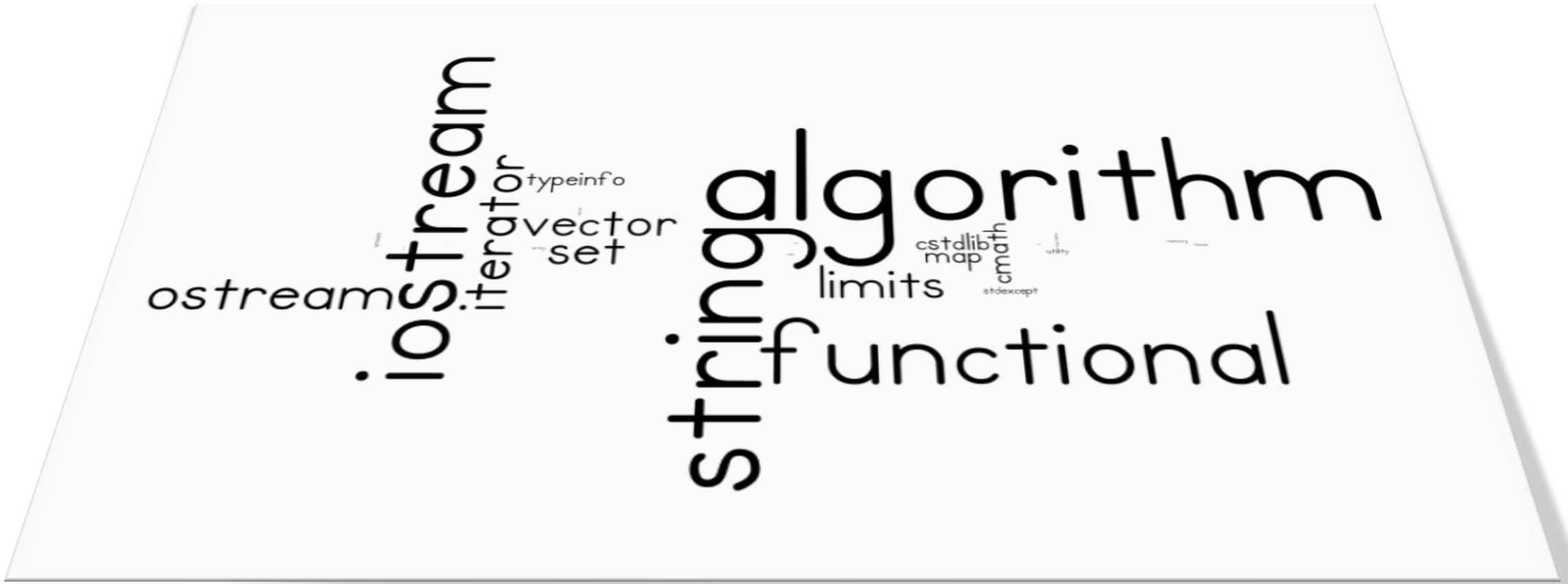
```
access with 0 is ok for size 5
access with 0 is ok for size 4
access with 0 is ok for size 3
access with 0 is ok for size 2
access with 0 is ok for size 1
access with 0 is ok for size 0
```

Discuss with your neighbor the keywords

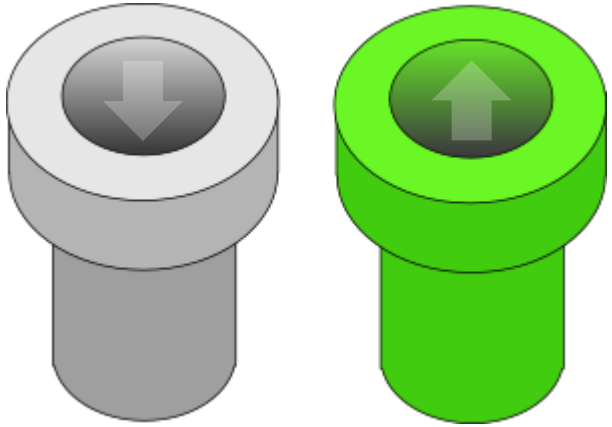
- `explicit`
- `inline`
- `using`
- `virtual`
- `mutable`
- `friend`
- `override`
- `final`

- **In which context can they be used?**
- **What do they mean?**
- **Do you know any other keywords?**

Standard Library


algorithm
functional
string
iostream
ostream
iterator
typeinfo
vector
set
limits
cstdlib
map
cmath
stdexcept

- **Input and output for programs**
  - `std::cin` and `std::cout` (only in `main` function)
- **Using and overloading input and output operators**
  - `operator >>`
  - `operator <<`

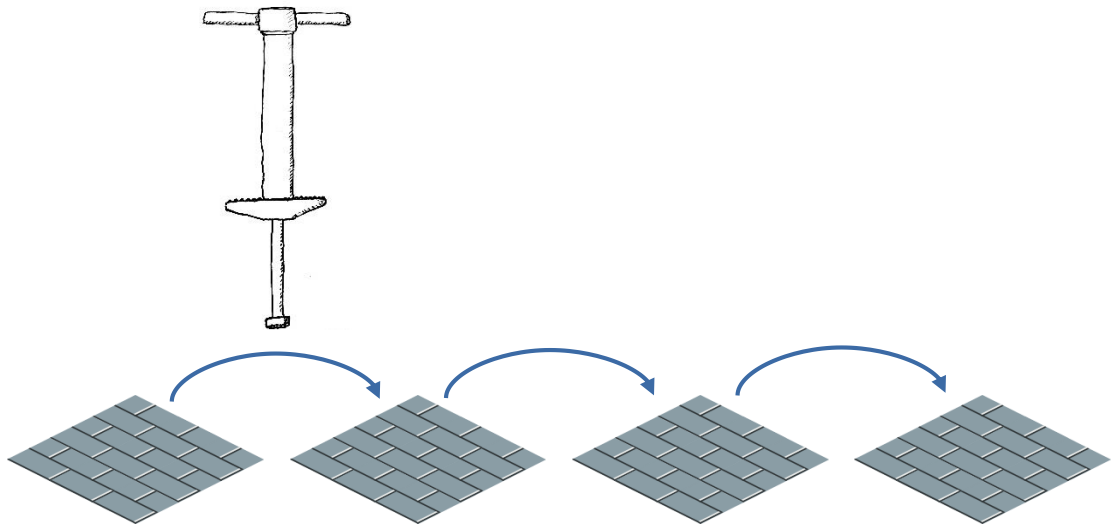


```
struct Order {
  unsigned amount;
  Product product;
};

std::ostream & operator<<(std::ostream & out, Order const & o) {
  out << o.amount << "x" << o.product;
  return out;
}
```

- **Iterators specify ranges**
- **Jump from element to element**

```
void iteration() {
  std::vector<Tile> tiles{...};
  auto pogo_stick = std::begin(tiles);
  pogo_stick++;
  auto tile = *pogo_stick;
}
```



- **Capabilities depend on the specific iterator type**
- **Categories:** Input iterator, Bidirectional iterator, Output iterator, Forward iterator, Random access iterator
- **We will implement our own iterators in this module**

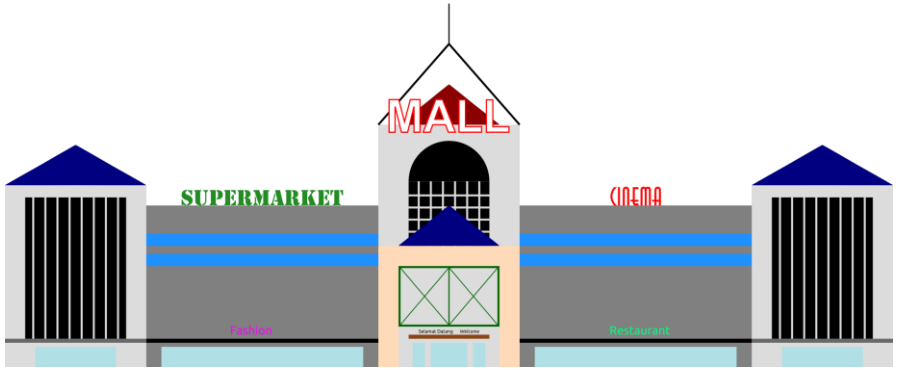
- **Benefits of using standard algorithms over hand-written loops**
  - Correctness
  - Easier to read and understand
  - Performance

```
bool find_with_loop(std::vector<int> const & values, int const v) {
  auto const end = std::end(values);
  for(auto it = std::begin(values); it != end; ++it) {
    if (*it == v) {
      return true;
    }
  }
  return false;
}
```

```
bool find_with_algorithm(std::vector<int> const & values, int const v) {
  auto const pos = std::find(std::begin(values), std::end(values), v);
  return pos != std::end(values);
}
```

- **Heap memory management in C++ should be handled with `std::shared_ptr` and `std::unique_ptr`**

```
void mall() {
  auto main_door = std::make_shared<Mall>();
  auto side_door = main_door;
  auto more_door = side_door;
  auto hodor = main_door;
}
```




```
void toolbox() {
  auto handle = std::make_unique<Hammer>();
  //You cannot have a hammer with two handles
  auto handle_too = handle;
}
```

- **We will look at legacy alternatives using `new` and `delete` in this module though**

Advanced Topics

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

IFS
INSTITUTE FOR
SOFTWARE

- **The most capable means of introducing coupling into your software**

```
class Parent {
  //Members of Parent
};

class Child : public Parent {
  //Members of Child
};
```

- **Can be used to "mix-in" functionality into your classes**
  - E.g. boost::operators
- **Multiple inheritance will be discussed in more detail**
  - Especially diamond hierarchies

- **To write compile-time polymorphic classes and functions**
  - With template argument deduction for template functions

```
template<typename T>
class Box {
  T content;
public:
  T const & peek() const {
    return content;
  }
  T & open() {
    return content;
  }
};
```

```
template<typename T>
Box<T> wrap(T const & t) {
  return Box<T>{t};
}


...
Present doll{};
Box<Present> gift = wrap(doll);
```

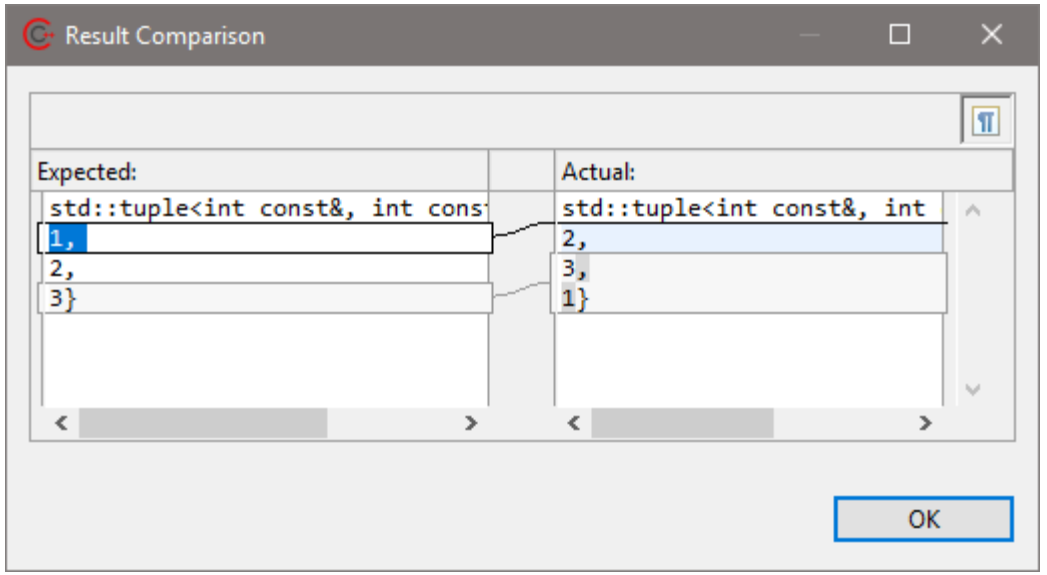
- **Variable templates will be introduced**
- **SFINAE**

- **Creating unit tests using CUTE**

```
void thisIsATest() {
  ASSERTM("start writing tests", false);
}
```

- **The CUTE plug-in has a simplified wizard, regarding libraries**
- **Also new the visualization of tuples**

```
void testTuplesAreDifferent() {
  int const a { 1 };
  int const b { 2 };
  int const c { 3 };
  ASSERT_EQUAL(std::tie(a, b, c), std::tie(b, c, a));
}
```



- **We might have a look at a different test approach later in this course for our container example**

- **Signed overflow (Unit C++20)**

```
void evil_inside() {
  int i = 0;
  while (i++ > 0);
}
```

- **Dangling references**

```
int & evil_outside() {
  int i = 0;
  return i;
}
```

- **Multiple side effects**

```
void whats_that() {
  int i{};
  i = i++ + ++i;
}
```

- **Accessing (possibly) unallocated memory**

```
void wish_me_luck() {
  std::vector<int> values{1, 2};
  int second = values[2];
  ...
}
```

- **Not UB but the emojis of programming**

```
bool statement = ...;
if (statement == true) {
  return false;
} else if (false == statement) {
  return true;
} else {
  return true;
}
```