# Technicalities

- `enum`

# Technicalities

- `enum`
- `union`

# Technicalities

- `enum`
- `union`
- `std::variant`, `std::any`, `std::optional`

# Technicalities

- `enum`
- `union`
- `std::variant`, `std::any`, `std::optional`
- `assert`, `static_assert`

# Technicalities

- `enum`
- `union`
- `std::variant`, `std::any`, `std::optional`
- `assert`, `static_assert`
- `std::endl` vs. `'\n'`

# Technicalities

- `enum`
- `union`
- `std::variant`, `std::any`, `std::optional`
- `assert`, `static_assert`
- `std::endl` vs. `'\n'`
- CLI arguments: `argc`, `argv`

# Enumerations

```
enum class Color { red = 0, blue, white, black};
//...
Color c = Color::red;
//...
```

# Enumerations

```
enum class Color { red = 0, blue, white, black};

std::array<int, 3> toArray(Color c = Color::red){
  switch(c){
  case Color::red:   return {255, 0, 0};
  case Color::blue:  return {0, 0, 255};
  case Color::white: return {255, 255, 255};
  case Color::black: return {0, 0, 0};
  }
}
```

# Unions

```
union U {
  int i;
  double d;
};
```

- Only one member can be active
- Size: maximum of the members

# Unions

```
union U {
  int i;
  double d;
};
U u;
u.i = 42; // int mode
std::cout << u.i; // ok
std::cout << u.d; // undefined behavior
```

- Only one member can be active
- Size: maximum of the members

# Unions

```
union U {
  int i;
  double d;
};
U u;
u.i = 42; // int mode
std::cout << u.i; // ok
u.d = 3.14; // double mode
std::cout << u.d; // ok
```

- Only one member can be active
- Size: maximum of the members

# Better Union: std::variant

```cpp
#include <variant>
std::variant<int, double> v;
```

- Only one member can be active
- Size: maximum of the members

# Better Union: std::variant

```cpp
#include <variant>
std::variant<int, double> v;
v = 42; // int mode
std::cout << std::get<int>(v); // ok
v = 3.14; // double mode
std::cout << std::get<double>(v); // ok
int i = std::get<int>(v); // exception
```

- Only one member can be active
- Size: maximum of the members

# Any

```cpp
#include <any>
std::any a;
```

- Dynamic memory

# Any

```cpp
#include <any>
std::any a;
a = 42;
a = std::string("hello");
a = 3.14;
a = std::vector<int>{1, 2, 3};
```

- Dynamic memory

# Any

```cpp
#include <any>
std::any a;
a = 42;
a = std::string("hello");
std::cout<<std::any_cast<std::string>(a); // ok
std::cout<<std::any_cast<int>(a); // exception
```

- Dynamic memory

# Optional

```cpp
#include <optional>
std::optional<int> o;
```

- Might or might not hold a value

# Optional

```cpp
#include <optional>
std::optional<int> o;
if(o) std::cout << *o;
o = 42;
if(o) std::cout << *o;
```

- Might or might not hold a value

# Optional

```cpp
#include <optional>
std::optional<int> foo() {
  if(errorCondition) return std::nullopt;
  return 42;
}
int main(){
  auto o = foo();
  if(o.has_value())
    std::cout << o.value();
  //std::cout << *o;
}
```

- Error-handling option

# Assertions

```
#include <cassert>

void f(int i) {
  if(i >= 0)
    throw std::invalid_argument("i<0!");
  // ...
}
```

# Assertions

```cpp
#include <cassert>

void f(int i) {
  assert(i >= 0);

  // ...
}
```

- Disabled if compiled with -DNDEBUG
- CMake: `-DCMAKE_BUILD_TYPE=Release`

# Assertions

```cpp
#include <cassert>
template<typename Container>
void f(const Container &v) {
  assert(std::is_sorted(v));

  // ...
}
```

- Disabled if compiled with -DNDEBUG
- CMake: `-DCMAKE_BUILD_TYPE=Release`

# Static Assertions

```
static_assert(sizeof(int) == 4,
              "int is not 4 bytes");
```

- Compile time check

# Static Assertions

```
template <typename T>
void f(T t) {
  static_assert(std::is_integral<T>::value,
                "T must be integral");
  // ...
}
```

- Useful for templates

# Flushing streams

```
std::cout << "Hello" << std::endl;
std::cout << "World" << '\n';
```

- `std::endl` flushes the buffer
- `'\n'` does not

# Inputs to main

```c
int main(int argc, char *argv[]) {




}
```

- `$ ./program arg1 arg2...`
- `argv[0]` $\rightarrow$ program name

# Inputs to main

```
using namespace std;
int main(int argc, char *argv[]) {
  for(int i = 0; i < argc; ++i)
    cout << format("{} received\n", argv[i]);




  // ...
}
```

- $ ./program arg1 arg2...
- argv[0] $\rightarrow$ program name

# Inputs to main

```cpp
using namespace std;
int main(int argc, char *argv[]) {
  if(argc < 2) {
    cerr <<"Usage:";
    cerr<<format("./{} filename",argv[0]);
    return 1;
  }
  ifstream file(argv[1]);
  // ...
}
```

- $ ./program arg1 arg2...
- argv[0] $\rightarrow$ program name