Курс-интенсив

Программирование на С++

Стандартная библиотека
Часть 2

academy.rubius.com
sergey@prohanov.com
Сергей Проханов

std::optional

```
template< class T >
class optional;
```

```
std::optional<std::string> UI::FindUserNick()
{
   if (nick_available)
       return { mStrNickName };

   return std::nullopt;
}

std::optional<std::string> UserNick = UI->FindUserNick();

if (UserNick)
   Show(*UserNick);
```

std::optional

```
#include <string>
#include <iostream>
#include <optional>
std::optional<std::string> create(bool b) {
 if (b)
   return "Godzilla";
 return {};
}
auto create2(bool b) {
 return b ? std::optional<std::string>{"Godzilla"} : std::nullopt;
}
int main()
{
 std::cout << "create(false) returned"
           << create(false).value_or("empty") << '\n';
 if (auto str = create2(true)) {
   std::cout << "create2(true) returned " << *str << '\n';
 }
}
```

std::any

```
#include <any>

std::vector<std::any> items = { 42, "text"s, std::vector {1, 0}};
auto any = std::make_any<User>("user@company.org", "password");


if (any.has_value() && any.type() == typeid(User)) {
    const auto user = std::any_cast<User>(any);
    // do something...
}

try {
    const auto user = std::any_cast<User>(any);
    // do something...
} catch (const std::bad_any_cast &e) {
    std::cout << e.what() << "/n";
}
```

std::variant

```
std::variant<double, std::string, User> variant = "text";

const std::string text = std::get<std::string>(variant);
const double value = std::get<0>(variant);
```

```
try {
    const User user = std::get<User>(variant);
    // do something...
} catch (const std::bad_variant_access &e) {
    std::cout << e.what() << "/n";
}
```

std::variant

```
std::variant<int, std::string> v = 42;

switch (v.index()) {
    case 0:
     // …

    case 1:
     //...
}


if (auto* ptr = std::get_if<int>(&v)) {
    int value = *ptr;
} else if (auto *ptr = std::get_if<std::string>(&v)) {
    std::string value *ptr;
}
```

std::visit

```
struct Voice {
  template<typename T>
  void operator()(const T& animal)
  {
     animal.voice();
  }
};
```

```
using Animal = std::variant<Cat, Dog>;
```

```
std::vector<Animal> animals = {
    Cat(),
    Dog(),
};
```

```
for (const Animal& animal : animals) {
    std::visit(Voice(), animal);
}
```

std::pair

```
#include<utility>

namespace std {
template<
   class T1,
   class T2
> struct pair;
}
```

```
#include <utility>
#include <string>
#include <iostream>

int main () {
 std::pair <std::string,double> product1;
 std::pair <std::string,double> product2 ("tomatoes",2.30);
 std::pair <std::string,double> product3 (product2);

 product1 = std::make_pair(std::string("lightbulbs"),0.99);
 product2.first = "shoes";
 product2.second = 39.90;

return 0;
}
```

std::tuple

```
#include<tuple>
namespace std {
     template<class... Types >
     class tuple;
}
```

```
std::tuple <char, int, float> first;
std::tuple <int, char, float> second(10,'f',15.5);

first = std::make_tuple('a', 10, 15.5);

std::cout << "The initial values of tuple are : ";
std::cout << get<0>(first) << " " << get<1>(first);
std::cout << " " << get<2>(first) << endl;
std::get<0>(first) = 'b';
std::get<2>(first) =  20.5;
std::cout << tuple_size<decltype(first)>::value << "\n";
first.swap(second);

std::tie(int i_val, char ch_val, ignore) = second;

auto third = std::tuple_cat(first,second);
```

std::initializer_list

```
#include <initializer_list>

template< class T >
class initializer_list;
```

```
template <class T>
class MyVector
{
public:
explicit MyVector(std::initializer_list<T> il)
:size_(il.size()),data_(new T[size_])
{
 std::copy(std::begin(il), std::end(il), data_);
}
~MyVector()
{
 delete [] data_;
}
private:
size_t size_;
T *data_;
};

auto v = MyVector<std::string>{"Hello", "to", "you"};
```

Исключения

```
#include <cmath>
#include <iostream>
double TrueSqrt(double a)
{
        if (a < 0.0)
             throw "Negative number!";
        return sqrt(a);
}
int main()
{
        double a = -42.0;
        try
        {
             double d = TrueSqrt(a);
             std::cout << "The sqrt of " << a << " is " << d << '\n';
        }
        catch (const char* exception)
        {
             std::cerr << "Error: " << exception << std::endl;
        }
        return 0;
}
```

Обработка ошибок

```
#include <exception>
```

```
logic_error
       invalid_argument
       domain_error
       length_error
       out_of_range
       future_error(C++11)
   bad_optional_access(C++17)
   runtime_error
       range_error
       overflow_error
       underflow_error
       regex_error(C++11)
       nonexistent_local_time(C++20)
       ambiguous_local_time(C++20)
       tx_exception(TM TS)
       system_error(C++11)
           ios_base::failure(C++11)
           filesystem::filesystem_error(C++17)
   bad_typeid
   bad_cast
       bad_any_cast(C++17)
   bad_weak_ptr(C++11)
   bad_function_call(C++11)
   bad_alloc
       bad_array_new_length(C++11)
   bad_exception
   bad_variant_access(C++17)
```

Обработка ошибок

```
#include <iostream>
#include <cassert>

int main()
{
     assert(2 + 2 == 4);
     std::cout << "Execution continues past the first assert\n";

     static_assert(2 + 2 == 5,"definition not true");
     std::cout << "Execution continues past the second assert\n";

     return 0;
}
```

Умные указатели

```
include <memory>
```

```
std::unique_ptr

std::shared_ptr

std::weak_ptr
```

```
template<
   class T,
   class Deleter = std::default_delete<T>
> class unique_ptr;

template <
   class T,
   class Deleter
> class unique_ptr<T[], Deleter>;
```

```
template< class T > class shared_ptr;

template< class T > class weak_ptr;
```

std::unique_ptr

```
struct MyClass {
  MyClass(const char* s);
  void methodA();
};
void someMethod(MyClass* m);

void test() {
  unique_ptr<MyClass> ptr1(new MyClass("obj1"));
  ptr1->methodA();
  someMethod(ptr1.get());
  unique_ptr<MyClass> ptr2(std::move(ptr1));
  ptr1.reset(new MyClass("obj2"));
  ptr2.reset("obj3");
  ptr1.reset();
}
```

```
void methodA() {
  unique_ptr<int> buf(new int[256]);

  int result = fillBuf(buf))
  if(result == -1) {
     return;
  }
  printf("Result: %d", result);
}
```

std::shared_ptr

```
struct MyClass {
  MyClass(const char* s);
  void methodA();
};

void someMethod(MyClass* m);
auto ptr = make_shared<MyClass>("obj1");
ptr->methodA();
someMethod(ptr.get());
shared_ptr<MyClass> anotherPtr = ptr;
ptr.reset(new MyClass("obj2");
anotherPtr.reset();
```

std::weak_ptr

```
#include <iostream>
#include <memory>
std::weak_ptr<int> gw;
void observe()
{
   std::cout << "use_count == " << gw.use_count() << ": ";
   if (auto spt = gw.lock()) {
      std::cout << *spt << "\n";
   }
   else {
       std::cout << "gw is expired\n";
   }
}
int main()
{
   {
      auto sp = std::make_shared<int>(42);
      gw = sp;
      observe();
   }

   observe();
}
```

Немного математики

Комплексные числа

https://en.cppreference.com/w/cpp/numeric/complex

Рациональные числа

https://en.cppreference.com/w/cpp/numeric/ratio

Библиотека общих функций

https://en.cppreference.com/w/cpp/numeric/math

Специальные функции

https://en.cppreference.com/w/cpp/numeric/special_math

std::numeric_limits

https://en.cppreference.com/w/cpp/types/numeric_limits

```
#include <limits>

// плохо
std::cout << INT_MIN << "\n";
std::cout << DBL_EPSILON << "\n";

// хорошо
std::cout << std::numeric_limits<int>::min() << "\n";
std::cout << std::numeric_limits<double>::epsilon() << "\n";
```

Псевдослучайные числа

https://en.cppreference.com/w/cpp/numeric/random

```
template<typename T = double, REQUIRES(std::is_arithmetic_v<T>)>
T randomValue(
    T min = std::numeric_limits<T>::min(),
    T max = std::numeric_limits<T>::max()
)
{
    std::random_device device;
    std::default_random_engine engine(device());

    if constexpr (std::is_integral_v<T>) {
        return std::uniform_int_distribution<T>(min, max)(engine);
    } else  {
        return std::uniform_real_distribution<T>(min, max)(engine);
    }
}
```

std::chrono

```
template<
    std::intmax_t Num,
    std::intmax_t Denom = 1
> class ratio;
```

- clocks
- timepoints
- durations

```
template<
    class Rep,
    class Period = std::ratio<1>
> class duration;
```

```
template<
    class Clock,
    class Duration = typename
Clock::duration
> class time_point;
```

std::chrono

```cpp
using nanoseconds = duration<long long, nano>;
using microseconds  = duration<long long, micro>;
using milliseconds = duration<long long, milli>;
using seconds = duration<long long>;
using minutes = duration<int, ratio<60> >;
using hours = duration<int, ratio<3600> >;

//C++ 20
std::chrono::days
std::chrono::weeks
std::chrono::months
std::chrono::years


using nano = ratio<1, 1000000000>;
using micro = ratio<1, 1000000>;
using milli = ratio<1, 1000>;
```

std::chrono

```
#include <iostream>
#include <chrono>

constexpr auto year = 31556952ll; // seconds in average Gregorian year

using microfortnights = std::chrono::duration<float, std::ratio<14*24*60*60, 1000000>>;
using nanocenturies = std::chrono::duration<float, std::ratio<100*year, 1000000000>>;

std::chrono::seconds sec(1);
std::cout << "1 second is:\n";
std::cout << std::chrono::duration_cast<std::chrono::minutes>(sec).count()
             << " minutes\n";

std::cout << microfortnights(sec).count() << " microfortnights\n"
             << nanocenturies(sec).count() << " nanocenturies\n";
```

std::chrono

```
#include <iostream>
#include <iomanip>
#include <ctime>
#include <chrono>

int main()
{
   std::chrono::system_clock::time_point now = std::chrono::system_clock::now();
   std::time_t now_c = std::chrono::system_clock::to_time_t(now - std::chrono::hours(24));
   std::cout << "24 hours ago, the time was "
             << std::put_time(std::localtime(&now_c), "%F %T") << '\n';

   std::chrono::steady_clock::time_point start = std::chrono::steady_clock::now();
   std::cout << "Hello World\n";
   std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
   std::cout << "Printing took "
             << std::chrono::duration_cast<std::chrono::microseconds>(end - start).count()
             << "us.\n";
}
// high_resolution_clock::now()
```

Куда копать дальше?

C++11
СТАНДАРТНАЯ БИБЛИОТЕКА C++
Справочное руководство
ВТОРОЕ ИЗДАНИЕ
НИКОЛАИ М. ДЖОСАТТИС
ozon.ru

Яцек Галовиц
C++17 STL
СТАНДАРТНАЯ БИБЛИОТЕКА ШАБЛОНОВ
Packt>
ПИТЕР
ozon.ru

Курс-интенсив

Программирование на С++

Стандартная библиотека Часть 2

academy.rubius.com

sergey@prohanov.com

Сергей Проханов