Intro. Move семантика, perfect forwarding, lifetime

Курс

- 15 лекций и семинаров
- 3 больших ДЗ, 15 маленьких + Бонус
- Маленькие ДЗ будут выкладываться каждую неделю. От 3 задач
- $O_{\text{итог}} = \text{Round}(0.6 \cdot O_{\text{большие ДЗ}} + 0.4 \cdot O_{\text{маленькие ДЗ}} + \text{Бонус})$

Курс. Организация

- https://t.me/cpp_advanced_hse_2021
- Чат прилинкован к каналу
- Больше практики, больше писать кода
- Репозиторий https://gitlab.com/danlark/cpp-advanced-hse
- Инструкция в SETUP.md этого репозитория
- Тесты почти всегда открытые. Проверяются CI gitlab (не Яндекс.Контест)
- Списывание идёт сразу в УО
- Опционально стримы с написанием кода C++

Курс. Задачи

- Маленькие задачи
  - Написать один-два файла, закрепить материал. Привыкнуть к боевому окружению
  - Не разбираются на семинарах
- Большие задачи
  - Модульные на 500-2000 строк вместе с тестированием, сложные концепты, дотошность в деталях, код ревью по желанию
  - Разбираются на семинарах
- Бонусные задачи
  - Любой развлекательный материал как CTF, сверхсложные задачи на креативность, полезная деятельность вокруг инфраструктуры курса
  - Не разбираются вообще

Курс. Цели

- После основ C++, хочется объяснить, что стоит за концептами, какое место сейчас у C++ в мире
- Дать много задач, чтобы набилась рука и не было боязно идти на стажировки
- Курс за основу взять ШАДовский с поправкой на знание первого курса C++. Перезачесть ШАДовский курс в будущем будет нельзя
- C++ всё ещё используется в тьме продуктов. 9 из 11 Google миллиардников на C++, большинство Яндекса на C++, геймдев на C++ и т.д.

Лектор

- Работаю в Google Data Pipelining Efficiency
- Был разок в комитете по C++, много пишу каждый день на C++
- Закончил ФКН, был в ШАД, в инфраструктуре Яндекс.Поиска
- Мои основные увлечения это перформанс приложений, распределённые системы, поисковые движки, АКОС, C++, Rust, алгоритмы.

move семантика

lvalues, rvalues

```
// lvalue is a location value (or left hand side value)
// rvalue is a right hand side value
int i = 1;   // i is an lvalue, 1 is rvalue (literal constant)
int* y = &i; // we can take addresses from lvalue as they have location
1 = i;       // 1 is an rvalue, cannot be on the left
y = &666;    // 666 is rvalue, no location
```

move семантика

lvalues, rvalues

```
int SetValue() {
  return 42;
}

int& SetGlobal() {
  static const int j;
  return j;
}

SetValue() = 1;  // SetValue is not an lvalue, it is temporary returned from function
SetGlobal() = 5; // ok, int& is lvalue reference, it has location

// lvalue reference
int& a_ref = a;
++a_ref;
```

move семантика

```
// On the right side we have a temporary thing, an rvalue that needs to be
// stored somewhere in an lvalue.
// On the left side we have the reference (an lvalue) that should point to an
// existing object. But being 10 a numeric constant, i.e. without a specific
// memory address, i.e. an rvalue, the expression clashes with the very spirit
// of the reference.
int& b_ref = 10; // not ok
```

move семантика

```
void foo(int& x)
{
}

foo(42);  // Nope!
// This works instead:
// int x = 42;
// foo(x);
// Because int& can modify things! 42 is not modifiable by itself
```

move семантика

```
void foo_const(const int& x)
{
}
foo_const(10); // Yes!
```

move семантика

```
// Why?
// Well, it is a mistake and not at the same time.

// the following...
const int& ref = 10;

// ... would translate to:
int __internal_unique_name = 10;
const int& ref = __internal_unique_name;

// Without this rule we wouldn't do the things like:
class T; // defined somewhere
T f();
void g(const T &x);

g(f());
```

move семантика

Pre C++11

```
void move_string(std::string& s, std::string& f) {
  s.swap(f);
  f.clear();
}
```

move семантика

Example in KALDI:

```
void LatticeSimpleDecoder::ProcessEmitting(DecodableInterface *decodable) {
  int32 frame = active_toks_.size() - 1; // frame is the frame-index
                                         // (zero-based) used to get likelihoods
                                         // from the decodable object.
  active_toks_.resize(active_toks_.size() + 1);
  prev_toks_.clear();
  cur_toks_.swap(prev_toks_);

  // Processes emitting arcs for one frame.  Propagates from
  // prev_toks_ to cur_toks_.
  BaseFloat cutoff = std::numeric_limits<BaseFloat>::infinity();
  for (unordered_map<StateId, Token*>::iterator iter = prev_toks_.begin();
       iter != prev_toks_.end();
       ++iter) {
    StateId state = iter->first;
    Token *tok = iter->second;
  // ...
```