

Константин Игоревич Вадимов, Syntacore

Лаборатория RISC-V со среды

От C к C++

LRU - cache - last recent used



Реализация на C

Тема: как писать список + тесты, реализация + тесты, грайпер + тесты

Действия: „Свой собственный проект“

Стиль: два проекта на модуль: header + src file

! Реально важно: понимание переносимости абстракций (не std список например)

Реализация на C++

- Улучшения компилятора в C
- Одна из главных особенностей: объединение данных и методов их обработки
- Другой тон: писать лучше this (конечу 2-го семестра)
- Обобщение данных и методов: template <typename T> (в C: макросы)
- Прикладная задача: понимание комбинаторной сложности
- Бенчмарк: quick bench
- В cpp quicksort быстрее: компилятор оптимизирует (т.к. qsort не предкомпилирован: он универсален).
- Также подход к std позволяет писать кучу обобщенных контейнеров (cpp reference)
- В LRU не надо писать ни hashtable, ни list
- Создавать модификаторы максимально близко к т. использованию
- Семантика значений: better ceda как int

Д/З: grabbed ARC/20/LFU/LIRS с изданием конем (знает Sygynze)

Сингори C++ 20

Срагиспун 4-th edition

human C++ primer

лекция 3.

Объявление и определение

`int x` - определение (и объявление!)

`struct S;` - объявление

`extern void foo();` - определение

`extern S *ps;` - объявление

`int bar();` - объявление

`struct T {int x;};` - определение

`extern int y=0;` - определение

Ключевые слова

`static` - ограничивает область действия

`extern` - linkage specifier (указывает на определение)

Примеры:

`static` в ф-ии - модальное пер-ое, чтобы избежать дублирования

`static` в struct - для всех объектов пер-ое

Утверждение: для каждого объекта, он должен быть определен только один раз во всем программном модуле, в котором используется.

`inline` ф-ия - она на все экз. трансляции - может быть только в header или в ф-ии

`static` ф-ия - по умолчанию на каждую экз. Трансляция - вызывает ошибку при повторении

`static inline` \equiv `static` & `inline` вместе

метод внутри класса \equiv `inline` ф-ия (в header)

два класса - ODR violation

module linkage - чтобы избежать дублирования в header-е - нужно использовать `include!`

Можно пока не рассуждать!

Область видимости

function scope, block scope

lifetime - все наперед. Времени, когда переключатся баггана

Пример: `int a = a;` - scope нарушен, а lifetime - нет!

Правильные ссылки и указатели - beware!

Временный объект живёт до конца полного выражения.

Переопределение

C даёт гарантии по именован! Нельзя в asm = или ф-ии (из-за правил calling conv)

C++ - нет, т.к. перепрограммирование

Модули, скомпилированные разными компиляторами, могут не совпадать!

Поэтому C будет жить вечно

C++ = C - гарантия жизни

