# Codegen #5
27.03.19

# STL

- template<typename T> std::vector<T>
  - Random access
- template<typename T> std::map<T>
  - Hash
  - Access to
- #include <algorithms>
  - template<typename T> T **max**(const T& a, const T& b)
  - template<typename T, class Compare> T **max**(const T& a, const T& b, Compare **comp**)
  - template<typename InputIt, class UnaryPredicate> InputIt **find_if**(InputIt first, InputIt last, UnaryPredicate q)

# Lambdas & autos

- auto – language keyword
  - Выводит тип переменной во время компиляции
- Lambdas:

```
auto lmbd = [](auto a, auto b) {
    return a.first.first >= b.first.first && a.first.second <= b.first.second;
};
```

Lambda – сферическая функция в вакууме. Может быть заменена конструкцией:

```
struct noname{    return_type operator() (A a, B b)    { ... } }
```

**! Но** вы могли заметить пару квадратных скобок, их тоже можно использовать

**[=]** - захват переменных по значению **[]** - не захватывать ничего

**[&]** - захват переменных по ссылке **[a, b, &c]** – захват a, b – по значению, c – по ссылке

# Использование в коде

```
enum class Type {    METHOD, OBJECT   };

struct EnvElement{
    Symbol self;   Symbol name;    int offset;    Type  type;
    EnvElement(Symbol _self,Symbol _name, int _offset, Type _type) :
      self(_self),name(_name),offset(_offset),type(_type) {};
};
typedef std::vector<EnvElement>* Environment;
...
cgen(ostream&, Symbol self, Environment var, Environment met)
...
    auto pred = [ = ](EnvElement a){return a.name == name;};
    auto off_var = std::find_if(var->rbegin(), var->rend(), pred);
```

# Дополнительные MIPS-инструкции

- $s – source, $d – destination, TGT – target label(or address), imm – immediate
- blt $s1, $s2, TGT - branch if less than
- bgt $s1, $s2, TGT - branch if greater than
- bne $s, $s2, TGT - branch if not equal
- jalr $s -jump register and link to **ra** (ra = PC + 4; goto s;)
- la $d, TGT - load label address

# Статические переменные

- expr_is_const
  - Необходим для арифметики – признак того, что обьект константный или находится в куче:

$$\frac{so, S_1, E \vdash e_1 : Int(i_1), S_2 \quad v_1 = Int(-i_1)}{so, S_1, E \vdash \tilde{}e_1 : v_1, S_2} \quad [\text{Neg}]$$

В таком случае нам приидется создавать копию

# create_label

- При генерации условных бранчей нам каждый раз необходима уникальная метка, при вызове этой функции каждый раз будет возвращаться новое число

```
static int create_label()
{
    static int label = 0;
    return label++;
}
```

Однако у такого подхода есть предел!  +-32767
(look at <limits>)

# Кто такие emit

- Это обертка над записью в поток
- Emit_ ### (const char * reg … ostream& s)
  - emit_fetch_int – получение значения (load) из обьекта типа int (offset = DEFAULT_OBJFIELDS)
  - emit_push – запись регистра на стек (с увеличением sp)
  - emit_protobj_**ref** – выводит имя прототипа обьекта
  - emit_label_**def** – выводит определение нового label
- Все ref – это имена тех или иных обьектов, def – это определения(обычно в data-сегменте)

# Аллокация перемнных

- init_alloc_temp – обнуление выделенной памяти (при входе в процедуру)
- alloc_temp – выделение памяти на стеке для локальной переменной
- calc_temp – рекурентный вариант для обхода AST и подсчета максимального количества переменных

# Идеальное решение

- Cool-manual.pdf + 13.4 – Operational rules
- По описанию смотрим, пишем маленький ассемблер:
  - Для загрузки переменных из стека или из кучи (ищем в окружении оффсеты)
  - Создаем лейблы, прыгаем
  - Не забываем про окружение
- Если тест не проходит
  - Смотрим на его исходник (*.cl) – если этот тип еще не реализовали
  - Пробуем запустить его на spim – может станет видно