РАЗБОР ДОРЕШКИ 4

Compose и Composer

- память под массив указателей на функций
- порядок вызова функций, вырожденные случаи
- семантика значения

Память под массив функций

Variable Length Array - это нестандартное расширение, позаимствовано из Си (C99)

```
double Compose(size_t n, ...) {
    FuncT funcs[n];
    .....
}
```

даже если мы себе это позволим, то

```
Composer::Composer(size_t n, ...) {
    FuncT funcs[n];
    .....
    this->funcs_ = funcs;
    // указатель на массив ГДЕ? который живёт СКОЛЬКО?
}
```

Семантика значения

Composer можно копировать и передавать по значению

```
Composer make() { return Composer(2, sqrt, sqrt); }

double run(Composer c, double v) { return c(v); }

int main() {
    Composer c1 = make();
    Composer c2 = c1;
    c1 = c2;

    double x = run(c1, 123);
    double y = c1(123);
    double z = c2(123);
    assert(x == y && y == z);
}
```

Минималистичная реализация

```
class Composer {
    FuncT* funcs_;
    size_t n_;
public:
    Composer(size_t n, ...) {
        funcs_ = new FuncT[n];
        .....
    }
    ~Composer() {
        delete[] funcs_;
    }
    double operator()(double v) const { ..... }
};
```

Что произойдёт при копировании/присваивании?

- компилятор создаёт конструктор копирования и оператор присваивания по умолчанию - копирование полей как есть
- будет скопирован указатель, но не массив
- деструктор грохнет общий для всех копий массив!

Что делать?

- написать конструктор копирования и присваивание правильно
- вместо голого указателя на массив использовать умный - который умеет копировать или разделять общие данные
- ЗАПРЕТИТЬ семантику значения. Передавать объекты Composer по указателю/ссылке.

Makefile

- Смешивание исходников на двух языках
  - Заголовочный файл на общем подмножестве Си и C++
  - Линковка
- Сборка с зависимостями

Общий заголовочный файл

- компилятор Си декорирует имя функции как `_sum`
- C++ - как `__Z3sumii` - если ему не сказать, что это Си-совместимое имя (`extern "C"`)

Способы сказать:

- sum.h - чисто сишный заголовочный файл, обрамить его включение
- sum.h - сишно-плюсовый файл, специальным образом написанный

Адаптация к сишному заголовочнику

```
extern "C" {
#include "sum.h"
}
```

Допиливание самого файла

```
#ifdef __cplusplus
extern "C" {
#endif

int sum(int, int);

#ifdef __cplusplus
}
#endif
```

```
#ifdef __cplusplus
#define EXTERNC extern "C"
#else
#define EXTERNC
#endif

EXTERNC int sum(int, int);
```

Зависимости

Смысл указания зависимостей - это инкрементная сборка:

Цель надо построить, если

- этот файл отсутствует
- он более старый, чем файлы, от которых он зависит
- (ну и далее рекурсивно)

```
all:
    gcc sum.c test.cpp -o task2
```

```
all: test

test:
    gcc sum.c test.cpp -o task2
```

```
all: task2

task2:
    gcc sum.c test.cpp -o task2
```

```
all: task2

task2: sum.c test.cpp
    gcc sum.c test.cpp -o task2
```

```
all: task2

task2: sum.c test.cpp sum.h
    gcc sum.c test.cpp -o task2
```

Тут я сделал две ошибки. Найдём их?

```
all: task2

task2: sum.o test.o
    g++ $^ -o $@
    # или, что то же самое,
    g++ sum.o test.o -o task2

sum.o: sum.c sum.h
    g++ $< -o $@
    # или, что то же самое,
    g++ sum.c -o sum.o

test.o: test.cpp
    g++ $< -o $@
```

Тут я сделал две ошибки. Найдём их?

```
all: task2

task2: sum.o test.o
    g++ $^ -o $@
    # или, что то же самое,
    g++ sum.o test.o -o task2

sum.o: sum.c sum.h
    g++ $< -o $@
    # или, что то же самое,
    g++ sum.c -o sum.o

test.o: test.cpp
    g++ $< -o $@
```

- sum.c - скомпилировал как C++

Тут я сделал две ошибки. Найдём их?

```
all: task2

task2: sum.o test.o
    g++ $^ -o $@
    # или, что то же самое,
    g++ sum.o test.o -o task2

sum.o: sum.c sum.h
    g++ $< -o $@
    # или, что то же самое,
    g++ sum.c -o sum.o

test.o: test.cpp
    g++ $< -o $@
```

- sum.c - скомпилировал как C++
- test.o - не указал зависимость от sum.h

Enum

- обычный `enum` и `enum class`
- определения внутри и вне `namespace wdu`
- ошибки

enum / enum class

```
enum X { X1, X2, X3 };

X x = X1;

int i = x;
X y = static_cast<X>(i);
```

```
enum class X { X1, X2, X3 };

X x = X::X1;

int i = static_cast<int>(x);
X y = static_cast<X>(i);
```

определения внутри и вне namespace

```
namespace wdu {

enum WeekDay { MONDAY, ..... };

WeekDay first();
WeekDay next(WeekDay w);

}  // namespace wdu
```

```
wdu::WeekDay wdu::first() { ..... }
wdu::WeekDay wdu::next(WeekDay w) { ..... }
```

```
namespace wdu {

WeekDay first() { ..... }
WeekDay next(WeekDay w) { ..... }

}  // namespace wdu
```

Ошибки

- MONDAY..SUNDAY = 0..6, а не 1..7

```
enum WeekDay { MONDAY, TUESDAY, ....., SUNDAY };
```

```
enum WeekDay { SUNDAY, MONDAY, TUESDAY, ....., SATURDAY
```

- не исчерпывающий switch / if

```
const char* GetDayOfWeekName(WeekDay w) {
    switch(w) {
        case WeekDay::MONDAY: return "monday";
        .....
        case WeekDay::SUNDAY: return "sunday";
    }
    // сюда мы не должны попасть
}
```

- не исчерпывающий switch / if

```
const char* GetDayOfWeekName(WeekDay w) {
    switch(w) {
        case WeekDay::MONDAY: return "monday";
        .....
        case WeekDay::SUNDAY: return "sunday";
    }
    // сюда мы не должны попасть
}
```

- а если вместо WeekDay тип w будет int?

- не исчерпывающий switch / if

```
const char* GetDayOfWeekName(WeekDay w) {
    switch(w) {
        case WeekDay::MONDAY: return "monday";
        .....
        case WeekDay::SUNDAY: return "sunday";
    }
    // сюда мы не должны попасть
}
```

- а если вместо WeekDay тип w будет int?
- будет warning `control may reach end of non-void function [-Wreturn-type]`

- не исчерпывающий switch / if

```
const char* GetDayOfWeekName(WeekDay w) {
    switch(w) {
        case WeekDay::MONDAY: return "monday";
        .....
        case WeekDay::SUNDAY: return "sunday";
    }
    // сюда мы не должны попасть
}
```

- а если вместо WeekDay тип w будет int?
- будет warning `control may reach end of non-void function [-Wreturn-type]`
- treat warnings as errors `-Werror`

Вычисления

Вычисления

- арифметика по модулю 7 на $[1..7]$
  - это просто $[0..6] + 1$, а не $[0..7]$ по модулю 8

Вычисления

- арифметика по модулю 7 на [1..7]
  - это просто [0..6] + 1, а не [0..7] по модулю 8
- учёт високосных (или невисокосных) лет
  - сколько максимум дней в феврале?

Вычисления

- арифметика по модулю 7 на [1..7]
  - это просто [0..6] + 1, а не [0..7] по модулю 8
- учёт високосных (или невисокосных) лет
  - сколько максимум дней в феврале?
- таблица (бегущих сумм) дней в (прошедших) месяцах года
  - с января по декабрь, для невисокосного и невисокосного
    - `{+0, +31, +28, +31, +30, …, +30}`
    - `{+0, +31, +29, +31, +30, …, +30}`
  - с марта по февраль
    - `{+0, +31, +30, …, +31}`

StringView

- Это яркий пример класса с семантикой значения
- Никаких собственных массивов! Это просто полуинтервал над внешним массивом символов.
- Дефолтные конструктор копирования и оператор присваивания
- Почти все функции не меняют объект - нужно объявить их как const
- Внимательность к диапазонам индексов
- Можно выражать одни функции через другие
- И извините, но детская ошибка - sizeof вместо strlen.

Конструкторы и присваивания

```
class StringView {
    const char* s_ = nullptr;
    size_t n_ = 0;
public:
    // конструктор без параметров - пустая подстрока
    StringView() = default;

    // конструкторы с 1 аргументом участвуют в преобразо
    StringView(std::string s);  // смотрите, здесь ошибк
    StringView(const char* s);

    StringView(const char* s, size_t n);

    // конструктор копирования и оператор присваивания -
};
```

Присваиваем:

```
StringView sv;
sv = "hello";  // = static_cast<StringView>("hello")
sv = std::string("hello");  // почему так делать не след
sv = StringView("hello", 4);
```

Мы всё это получили нахаляву!

Внимательность

Рассмотрим `substr(size_t left, size_t right = npos)`

- очевидное требование: left ≤ right
- если size < right или right < left, это
  - нарушение контракта - неопределённое поведение - возвращаем мусор
  - диагностируемая ошибка - кидаем исключение
  - исправляемая ошибка - упорядочиваем индексы, или возвращаем пустую строку
- пример исправляемой не-ошибки: size < right
  - в частности, right = npos

- `haystack.find(needle)`
  - haystack.size() ≥ needle.size()
  - случаи, когда одна или обе строки пустые?
- `removePrefix(size_t delta)`
  - delta ≤ size

- `findOneOf(StringView charset)`
  - случай, когда charset пустой?
  - поиск можно выразить через `charset.find(s_[i])`