Конспект по уроку 14.02

Горячие клавиши

browser-like: chrome, vscode
browser: chrome, firefox
editors: chrome, word
code-editors: vscode, PyCharm
vscode: vscode

Windows
win + d - закрыть, открыть все окна
|win + e - проводник
win + r - быстрый промпт в консоль (например запуск ssh)
win + P - тип работы с вторым монитором(проектором)

Hot Keys

- ctrl + shift + n - новое окно (browser - like)
- ctrl + s - сохранить (editors)
- alt + tab - переключить окно (windows браузер)
- shift + arrows (-> <-) - выделение (editors)
- ctrl + arrows - указатель через слово (editors)
- ctrl + shift + arrows - выделить слово (editors)
- shift + alt + arrows up/down - скпорировать строку (code-editors)
- ctrl + delete - удалить слово после указателя (editors)
- ctrl + backspace - удалить слово перед указателя (editors)
- ctrl + / - разбить окно на два (vscode)
- ctrl + tab - запоминает последний использованный файл и переключается между ними (VSCode)
- ctrl + tab - переключить окно (Browser-like)
- alt + arrows (-> <-) - переключение между вкладками (переход к последнему изменению)
- alt + arrows - следующая/предыдущяя страница в истории окна (browsers)
- ctrl + w - закрыть окно (broswer-like)

- ctrl + b - закрыть/открыть левую панель (vscode)
- ctrl + shift + p - открыть панель быстрого доступа (vscode)
- ctrl + ` - открыть консоль (vscode)
- ctrl + l - выделить строку (vscode)
- shift + delete - удалить всю строку (vscode)
- ctrl + H - поиск + замена (editors) (в vscode есть regex)
- alt + click - несколько курсоров (vscode)
- ctrl + arrows up/down - пролистать вверх/вниз (vscode)

Для разных редакторов свои горячие клавиши. Справочник по сочетаниям в VSCode (ctrl + shift + t - открывает приборную панель со всеми функциями)

// код -> прекомпиляция -> ассемблер -> линковка -> exe
main.ii - расскрывает весь include <iostream>
include <bits/stdc++.h> - все возможные инклюды

Классы - между объектами имеется связь. Классы содержат поля и методы.
Структуры - передаёт тайно ссылку на бъект, на котором он находится.
*примеры с Citizen, Ray и Key
-> конструкторы (*cout = true) -> деструкторы (никогда ничего не принимает) (*isAlife = false)
Чтобы не повторялись поля (примеры с Ray и Key) используется принцип *инкапсуляции:*

- Public (объект может изменяться везде)
- Private (могут изменять только методы определённого класса; снаружи объект недоступен)

Объявление - компилятор знает, что в коде существует какая-то переменная; выделена помять под хранение переменной такого-то типа с таким-то именем (* объявление пустого класса Citizen)
Определение - информирует компилятор о том, что память под переменную нужно взять прямо в этом месте, где написано данное определение (* определяем функции класса Citizen)

Реализация функций класса снаружи класса:

```
void City::print() const {
    for (const Citizen* res : residents) {
        res->print();
    }
}
```

Фулл код с пары:

```
#include <iostream>
#include <vector>

class Citizen;

class City {
private:
    std::string title;
    std::vector<Citizen*> residents;
public:
    City(std::string title_);

    void print() const;

    void AddCitizen(Citizen* person);

    std::string GetTitle();
};

class Citizen {
private:
    bool is_alife;
    std::string name;
    City* home;
public:
```

```
    Citizen(std::string name_) : name(name_), is_alife(true), home(nullptr) {
        std::cout << "We created a person!\n";
    }
    //
    void print() const {
        if (is_alife) {
            std::cout << name;
            if (home) {
                std::cout << " is live in city " << home->GetTitle();
            }
        } else {
            std::cout << name << " is Dead";
        }
        std::cout << '\n';
    }

    void SetHome(City *new_home) {
        if (home == new_home) {
            std::cout << "No relocation\n";
        }
        home = new_home;
    }

    ~Citizen() {
        is_alife = false;
        std::cout << name << " is Dead\n";
    }
};

City::City(std::string title_) : title(title_) {
}

void City::print() const
{
    for (const Citizen* res : residents) {
        res->print();
```

```
    }
}

void City::AddCitizen(Citizen *person) {
    residents.push_back(person);
    person->SetHome(this);
}

std::string City::GetTitle() {
    return title;
}

int main() {
    Citizen Ray("Ray");
    Citizen Key("Key");
    std::cout << "City in town!\n";
    std::string t("Downtown");
    City Downtown(t);
    Downtown.AddCitizen(&Ray);
    Downtown.AddCitizen(&Key);
    Downtown.print();
}
```