Модуль 2

- Комментарии и автодокументирование
- Введение в функции
- Заголовочные файлы
- Базовая работа с файлами
- Директивы препроцессора
- Отладка в Microsoft Visual Studio

Комментарии

1. **Комментарии игнорируется компилятором**
2. **Однострочные комментарии**

   Однострочные комментарии начинаются с `//`.

   ```
   int index { 0 }; // counter initialization
   ```

3. **Многострочные комментарии**

   Многострочные комментарии начинаются с `/*` и заканчиваются на `*/`.

   ```
   /*
   This program
   prints a message
   to the console
   */
   ```

   Такие комментарии, обычно, используют для документирования кода, например, функций.

4. **С помощью комментариев можно пояснять код или устранять ошибки.**

Комментарии

Правила хорошего тона

Использовать или не использовать комментарии?

Автодокументирование

DOXYGEN

- система документирования исходных текстов, которая поддерживает C++, Си, Objective-C, Python, Java, IDL, PHP, C#, Фортран, VHDL и, частично, D.

Автодокументирование

```
/**
 * @brief Create queries to perform operations on symbols
 *
 * @param[in] conns              A list of connections used by the query to make requests
 * @param[in] filter_symbol      A flag to enable/disable symbol filtering
 * @param[in] filter_user_group  A flag to enable/disable user group filtering
 * @return                       A symbol query instance
 */
virtual std::unique_ptr<i_symbol_query>
create_symbol_query(app::ilist_cref<net::connection> conns,
                    const bool filter_symbol     = false,
                    const bool filter_user_group = false) const = 0;
```

Введение в функции

Функция определяет действия, которые выполняет программа. Функции позволяют выделить набор инструкций и назначить ему имя. А затем многократно по присвоенному имени вызывать в различных частях программы. По сути функция - это именованный блок кода.

```
тип имя_функции(параметры)
{
    инструкции
}
```

Возвращаемое значение

```
int sum(const int x, const int y)
{
    return x + y;
}
```

Введение в функции

```
#include <iostream>

void hello();

int main()
{
    hello();
    hello();
}

void hello()
{
    std::cout << "hello" << std::endl;
}
```

Заголовочные файлы

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

Заголовочные файлы

.h

```
#ifndef ADD_H // директивы препроцессора, ADD_H - обычно название файла
#define ADD_H

int add(int x, int y);

#endif
```

.cpp

```
int add(int x, int y)
{
    return x + y;
}
```

Заголовочные файлы

- Не определяйте функции в заголовочных файлах.
- Один файл - одна задача
- Название .cpp и .h должны совпадать
- По возможности подключайте .h файлы в .cpp.

Работа с файлами

Зачем нам файлы?

Работа с файлами

Классы для работы с файлами:

- **Ifstream;**
- **ofstream;**
- **fstream;**

Последовательность работы с файлом:

```
ofstream outf("SomeText.txt"); // открытие файла на запись
if (!outf) {                   // проверка на открытие файла
...
}
outf << "test" << endl;        // работа с файлом (здесь запись)
outf.close();                  // закрытие файла!
```

Работа с файлами

Константа	Описание
ios_base::in	открыть файл для чтения
ios_base::out	открыть файл для записи
ios_base::ate	при открытии переместить указатель в конец файла
ios_base::app	открыть файл для записи в конец файла
ios_base::trunc	удалить содержимое файла, если он существует
ios_base::binary	открытие файла в двоичном режиме

Комбинирование флагов `ios_base::out | ios_base::trunc`

Буферизация записи в файл

- ➔ Запись не происходит сразу после “**<<**”;
- ➔ Запись происходит по каким-то рандомным алгоритмам внутри класса;
- ➔ Можно использовать метод “**std::flush**”;
- ➔ **std::endl; -** также приводит к записи в файл. Используйте всегда **“\n”** для перехода на новую строку там, где это возможно.

Буферизация записи в файл

Рандомный доступ к файлам

- ➔ **seekg() // для ввода**
- ➔ **seekp() // для вывода**

Пример:

```
inf.seekg(15, ios::cur); // перемещаемся вперед на 15 байт относительно текущего местоположения файлового указателя
inf.seekg(0, ios::beg); // перемещаемся в начало файла
```

Флаги ios, которые принимают функции seekg() и seekp() в качестве второго параметра:

beg — смещение относительно начала файла (по умолчанию);

cur — смещение относительно текущего местоположения файлового указателя;

end — смещение относительно конца файла.

Директивы препроцессора

Директивы — это специальные команды, которые начинаются с символа # и НЕ заканчиваются точкой с запятой.

- #include <filename> ("filename")
- #define, #undef

  #define SIZE 5, #define SUM_5_5 5+5, #define true false)))))
- #ifdef, #ifndef, #endif
- #pragma once

Отладка в Microsoft Visual Studio

- ➔ Breakpoint
- ➔ Stack
- ➔ View local variables
- ➔ Пошаговая отладка
- ➔ Profiler
- ➔ Переключение на место определения/описания функции