Введение в классы

Hello world!

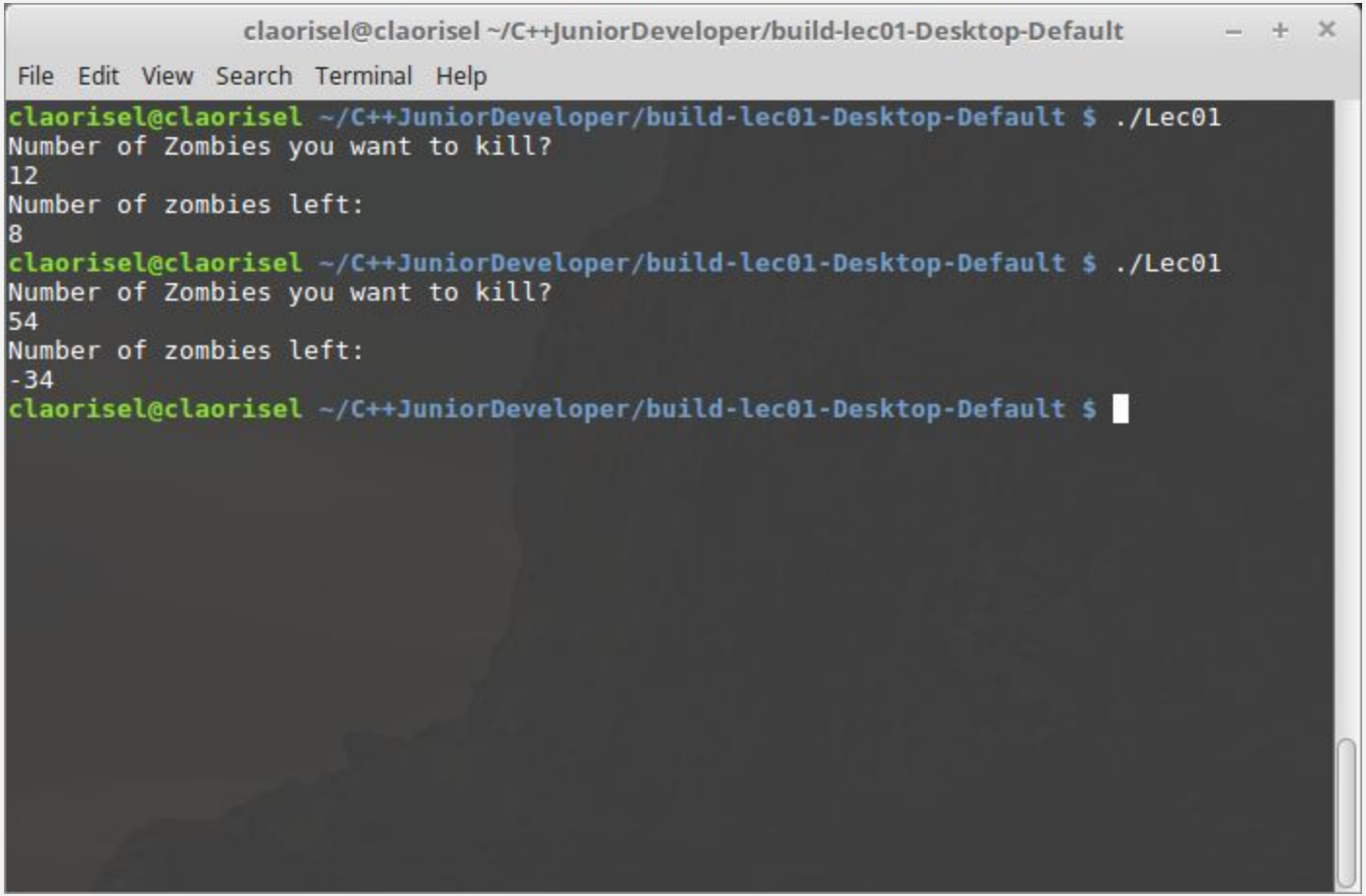
```
#include <iostream>
using namespace std;

int main(void) {
    cout << "Hello world!";
    return 0;
}
```

Типы данных

- bool
- int
- float
- double
- char
- string

Ввод/вывод

```
#include <iostream>
using namespace std;

int main(void) {

   int numZombies = 20;
   cout << "Number of Zombies you want to kill? " <<  endl;

   int numKilled;
   cin >> numKilled;

   cout << "Number of zombies left: " << endl;
   cout << numZombies - numKilled << endl;

   return 0;
}
```


claorisel@claorisel ~/C++JuniorDeveloper/build-lec01-Desktop-Default
File Edit View Search Terminal Help
claorisel@claorisel ~/C++JuniorDeveloper/build-lec01-Desktop-Default $ ./Lec01
Number of Zombies you want to kill?
12
Number of zombies left:
8
claorisel@claorisel ~/C++JuniorDeveloper/build-lec01-Desktop-Default $ ./Lec01
Number of Zombies you want to kill?
54
Number of zombies left:
-34
claorisel@claorisel ~/C++JuniorDeveloper/build-lec01-Desktop-Default $

Пользовательские типы данных

- Необходимо хранить учетные записи 43 000 студентов
- Каждая учетная запись должна содержать:
  - имя
  - фамилия
  - уникальный идентификатор
  - еще что-нибудь
- В каком виде хранить информацию?

class

```
class Student {
  public:
    string first_name;
    string last_name;
    string uniqname;
    int id;
    int status; //!< graduated or not
    int year_entry;
};
```

Способ объявления:
Student student1,
student2;

Доступ к полям класса

```
student1.first_name = “Kevin”;
student1.last_name = “Lee”;
student1.uniqname = “mrkavin”;
student1.id = 854345;
student1.status = UNGRAD;
student1.year_entry = 2013;
```

“dot operator”

Class vs Instance of class

- Класс является определением типа данных
- Имя типа совпадает с именем класса
- Класс описывает набор переменных экземпляра
- Класс реализует методы
- Экземпляр - конкретный объект в памяти
- Экземпляр - единственный и уникальный представитель класса

Class vs Instance of class

Класс

Экземпляры класса

Методы

```
class Student {
  public:
    string first_name;
    string last_name;
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    cout << kevin.first_name << " "
         << kevin.last_name << endl;
}
```

Методы

```
class Student {
  public:
    string first_name;
    string last_name;
    void printName() {
        cout << first_name << " "
             << last_name << endl;
    }
};

int main() {
    Student kevin = {"Kevin", "Lee"};
    kevin.printName();
}
```

Модификаторы доступа

- private
- public
- protected

Модификаторы доступа

```
class Student {
public:
   string first_name;
   string last_name;
   double gpa;
};

int main()
{
   Student helen = {"Helen", "Smith", 1.2};
   cout << helen.gpa << endl;
   helen.gpa = 4.0;
   cout << helen.gpa << endl;
   return 0;
}
```

```
claorisel@claorisel ~/C++JuniorDeveloper/build-lec01-Desktop-Default $ ./Lec01
1.2
4
```

Модификаторы доступа

```
using namespace std;

class Student {
private:
  string first_name;
  string last_name;
  double gpa;
};

int main()
{
  Student helen = {"Helen", "Smith", 1.2};
  helen.gpa = 4.0;
  cout << helen.gpa << endl;
  return 0;
}
```

```
claorisel@claorisel ~/C++JuniorDeveloper/lec01
File  Edit  View  Search  Terminal  Help
/home/claorisel/C++JuniorDeveloper/lec01/main.cpp:9:12: error: 'double Student::
gpa' is private
     double gpa;
            ^

/home/claorisel/C++JuniorDeveloper/lec01/main.cpp:15:11: error: within this cont
ext
     helen.gpa = 4.0;
           ^

/home/claorisel/C++JuniorDeveloper/lec01/main.cpp:9:12: error: 'double Student::
gpa' is private
     double gpa;
            ^

/home/claorisel/C++JuniorDeveloper/lec01/main.cpp:16:19: error: within t
ext
     cout << helen.gpa << endl;
                  ^

CMakeFiles/Lec01.dir/build.make:62: recipe for target 'CMakeFiles/Lec01.
.cpp.o' failed
make[2]: *** [CMakeFiles/Lec01.dir/main.cpp.o] Error 1
CMakeFiles/Makefile2:67: recipe for target 'CMakeFiles/Lec01.dir/all' fa
make[1]: *** [CMakeFiles/Lec01.dir/all] Error 2
Makefile:83: recipe for target 'all' failed
make: *** [all] Error 2
claorisel@claorisel ~/C++JuniorDeveloper/lec01 $
```

Конструктор

- имя конструктора совпадает с именем класса
- не имеет типа возвращаемого значение
- создает экземпляр класса
- инициализирует переменные объекты класса

Конструктор

```
class Student {
  public:
    Student(string first, string last) {
        first_name = first;
        last_name = last;
    }
  private:
    string first_name;
    string last_name;
};

int main() {
    Student kevin("Kevin", "Lee");
}
```

Конструктор устанавливает имя и фамилию студента.

Экземпляр класса kevin инициализируется с помощью “вызова функции”

Getters

```
class Student {
  public:
    Student(string first, string last) {
        first_name = first;
        last_name = last;
    }
    string getFirstName() {
        return first_name;
    }
  private:
    string first_name;
    string last_name;
};

int main() {
    Student kevin("Kevin", "Lee");
    cout << kevin.getFirstName() << endl;
}
```

Функция getFirstName() позволяет обратиться к закрытому(private) полю класса

Setters

```
class Student {
  public:
    Student(string first, string last) {
        first_name = first;
        last_name = last;
    }
    void getFirstName() {
        return first_name;
    }
    string setFirstName(string first) {
        first_name = first;
    }

  private:
    string first_name;
    string last_name;
};
```

```
int main() {
    Student kevin("Kevin", "Lee");
    kevin.setFirstName("NewName");
    cout << kevin.getFirstName() << endl;
}
```

Зачем нужны геттеры и сеттеры если можно сделать переменную public?

Header .h

```
#include <string>
using namespace std;

class Student {
public:
    Student(string first, string last);
    string getFirstName();
    void setFirstName(string first);
    void printName();
private:
    string first_name;
    string last_name;
};
```

Source .cpp

```cpp
#include <iostream>
#include "student.h"

using namespace std;

Student::Student(string first, string last) {

    first_name = first;
    last_name = last;
}

string Student::getFirstName() {
    return first_name;
}

void Student::setFirstName(string first) {
    first_name = first;
}

void Student::printName() {
    cout << first_name << " " << last_name << endl;
}
```

Собираем все вместе

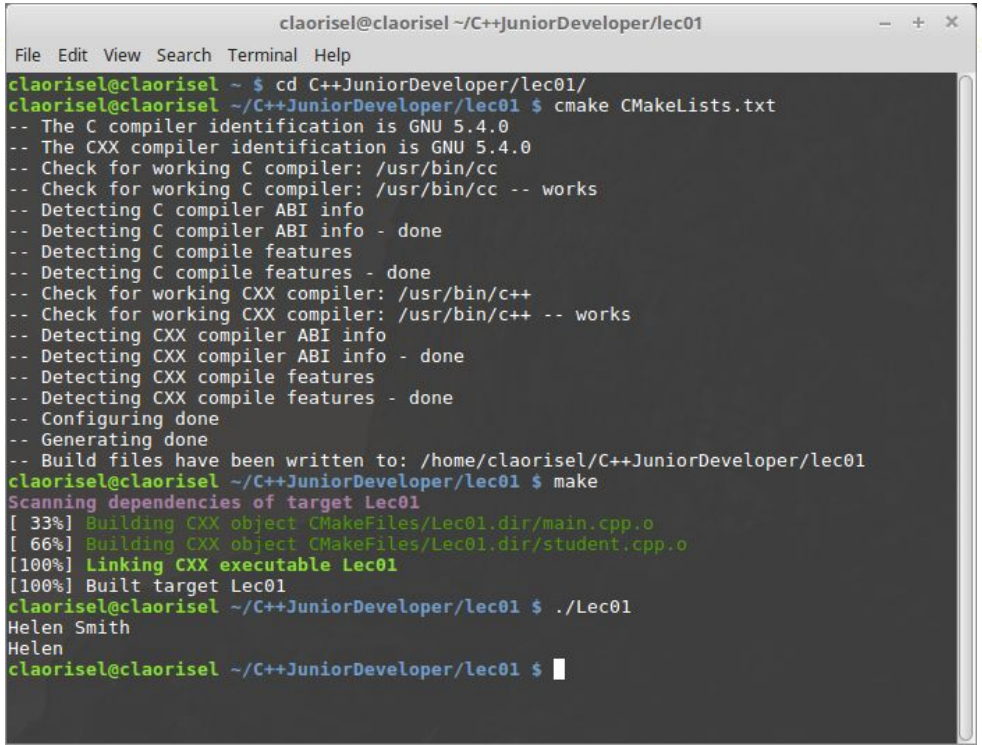
```cpp
#include <iostream>
#include "student.h"

using namespace std;

int main()
{
    Student helen("Helen", "Smith");
    helen.printName();
    cout << helen.getFirstName() << endl;
    return 0;
}
```

CMake

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)

project(Lec01)

set(SOURCE main.cpp student.cpp)

add_executable(${PROJECT_NAME} ${SOURCE})
```

CMake



Процесс сборки и запуска проекта

Lab

```
#include <iostream>
using namespace std;
class Critter
{
public:
   int hunger_;
   void getHungerLevel();
};
void Critter::getHungerLevel()
{
   cout << "My hunger level is " << hunger_ << endl;
}
int main()
{
   Critter critte11;

   return 0;
}
```

1. Конструктор / деструктор
2. get/set голод, уровень голода от 0 до 100
3. поле класса boredom - уровень готовности к активности
4. реализовать функции play(), eat(), passTime()

Создать тамагочи
Если пользователь не собирается завершить игру
Представить ему меню с вариантами выбора
Если пользователь хочет выслушать тамагочи
Предложить тамагочи что-то сообщить
Если пользователь хочет покормить тамагочи
Предложить тамагочи поесть
Если пользователь хочет поиграть с тамагочи
Предложить тамагочи поиграть