ALGEBRA

DATA STRUCTURES AND ALGORITHMS

Lecture 01

Learning outcome 1

ABOUT THE MODULE

ALGEBRA

Teachers

- Lectures:
  - Doc. dr. sc. Goran Đambić
  - goran.djambic@racunarstvo.hr
  - Consultations: anytime, with prior e-mail or Teams communication
- Workshops:
  - Daniel Bele, p.m. comp.ing

About the module

- Module goals:
  - Learn about basic abstract data types and algorithms
  - Learn to use concrete implementations in C++
  - Reach junior programming skills in C++
- Module is worth **6 ECTS** points (roughly 30 hours / point)
  - 30 hours of lectures (15 weeks x 2 hours)
  - 30 hours of workshops (15 weeks x 2 hours)
  - 120 hours of individual work (15 weeks x 8 hours)
    - **The actual amount of individual work depends on skills acquired in Programming**

Signature

- In order to obtain the right to a signature, it is necessary to participate in class at the percentage rate prescribed by the Book of Regulations on studies and studying.

Lectures and practical classes participation	
At least 50 % of physical presence in lectures	At least 60 % of physical presence in practical classes

- Whoever fails to obtain a signature will have to enroll in the same course the following year, to pay the enrollment and does not have the right to take the exam.

Points and grades

- There are **100 points** available to collect:
  - Homeworks: max **6 points**
  - Schoolworks: max **6 points**
  - Two midterms: max **88 points**
  - No oral exam

$$\sum = 100$$

- Grades:
  - 92.01 – 100.00 points: excellent (5)
  - 75.01 – 92.00 points: very good (4)
  - 58.01 – 75.00 points: good (3)
  - 50.01 – 58.00 points: sufficient (2)
- In every learning outcome you must collect at least 50% of points

Learning outcomes

LO set	LO	MINIMUM LEVEL	DESIRED LEVEL
S1	LO1	Determine and argue the time complexity a priori and a posteriori for a given algorithm derived in a programming language.	Create more complex software solutions using multi-file projects and user-defined data types.
S1	LO2	Construct a solution using linear data structures (list, linked list, stack, queue) and associated algorithms.	Construct a more complex solution using linear data structures (list, linked list, stack, queue) and associated algorithms.
S2	LO3	Construct a solution using hierarchical data structures (tree, heap, priority queue) and associated algorithms.	Construct a more complex solution using hierarchical data structures (tree, heap, priority queue) and associated algorithms.
S2	LO4	Construct a solution using tree-based dictionaries and associated algorithms.	Construct a more complex solution using tree-based dictionaries and associated algorithms.
S3	LO5	Describe sorting and search algorithms and construct a solution based on sorting and search algorithms.	Describe sorting and search algorithms and construct a more complex solution based on sorting and search algorithms.
S4	LO6	Create a solution using addressing techniques and argue their time complexity.	Create a more complex solution using addressing techniques and argue their time complexity.

ALGEBRA

Learning outcomes and assessments

	Midterm 1	Midterm 2	Homeworks	Schoolworks	MAX
LO1	20		2	2	24
LO2	20		2	2	24
LO3		12		1	13
LO4		12	2		14
LO5		12		1	13
LO6		12			12
TOTAL	40	48	6	6	100

ALGEBRA

Exams

- There will be two midterm exams:
  - Midterm exam 1 (MI1) covers LOs 1 and 2
    - After week 8
  - Midterm exam 2 (MI2) covers LOs 3, 4, 5 and 6
    - After week 15
- In MI2, student can also retake any/all LOs from Midterm 1
- In any other exam term student can retake any/all LOs

GitHub

- Every student should have his own GitHub account because it will be used to submit homeworks
- Your task until next week: open a GitHub account

Academic standard of conduct

- In written and oral communication it is necessary to follow the rules of business communication appropriate for the academic level.
- It is necessary to abide by the strictly defined deadlines for task submissions (homework, seminar papers, projects, etc…).
  - Every task, homework, project etc…, submitted after the defined deadline will not be evaluated nor graded.
- Only those students who can confirm their attendance, will be considered as present.
  - Signing other students, or registering their card is not allowed and may be subject to disciplinary action. The teacher will delete the student's attendance if he / she determines that the student is registered and is not present at the class.

Rules of conduct during classes

- One has to come to class on time.
- Each student should disinfect their hands before accessing the workplace.
- Upon entering the classroom, student registers for classes with a card and then sits in an accessible place for work.
- Compliance with epidemiological measures is mandatory: currently this means wearing a mask in a way that it covers mouth and nose all the time. A student who violates that will be removed from class and reported to the Disciplinary Board.
  - If and when epidemiological measures change, we will adjust the rules.
- Disruption of class and inactive class participation is not allowed.
  - Continuous breaking of this rule is sanctioned by reporting students to the Disciplinary Board.

Programming vs Data Structures and Algorithms

PUBLIC AND PRIVATE MEMBERS

Structures and classes (1/2)

- Data types can be categorized:
  - Built-in types (`int`, `void`, `double`, `char`, `wchar_t`, `long`, ...)
  - User-defined types (structures and classes)
- Structure members are public by default, class members are private by default
  - Members are most often variables (called *fields*) and functions (called methods)

```
struct Rectangle {
    int a;
    int b;
};
```

```
class Rectangle {
    int a;
    int b;
};
```

Structures and classes (2/2)

- We can explicitly define which members are public and which are private

```
struct Rectangle {
public:
    int a;
private:
    int b;
};
```

```
class Rectangle {
public:
    int a;
private:
    int b;
};
```

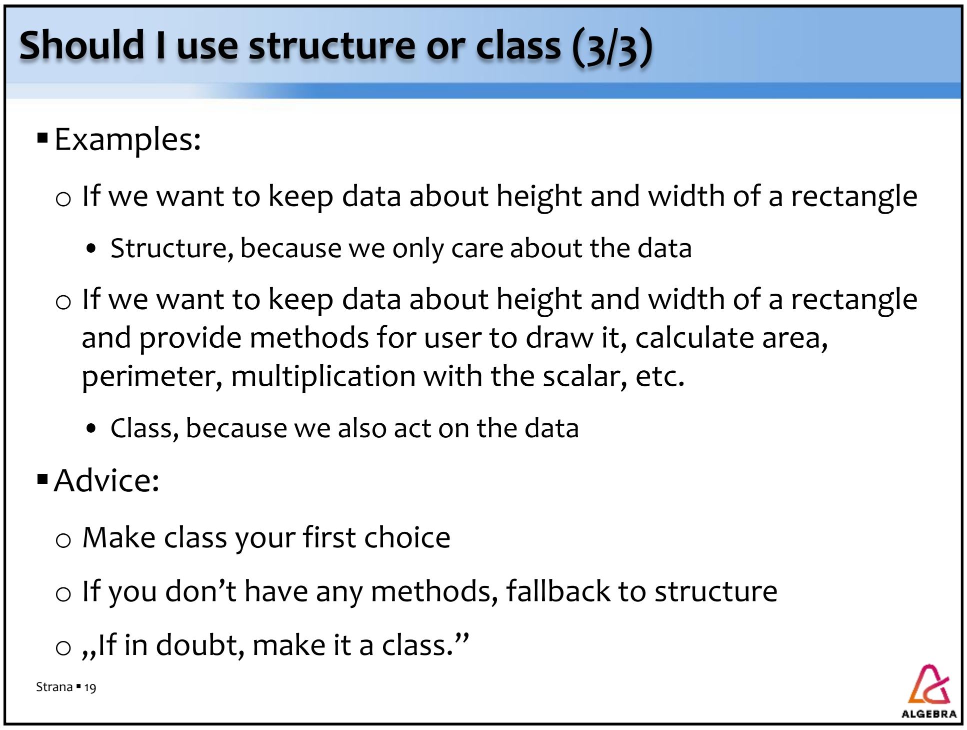
- Private members can be accessed and used only from methods on the same class
- Public members can be accessed and use from anywhere (from main, other functions etc.)

Should I use structure or class (1/3)

- Besides default member visibility, structures and classes don’t have any other differences
  - As far as the compiler is concerned
- There are differences in semantics
  - What it means to people
  - If we need a new user-defined data type whose main purpose will be holding data without many methods => we choose structure
  - For everything else => we choose class

Should I use structure or class (2/3)

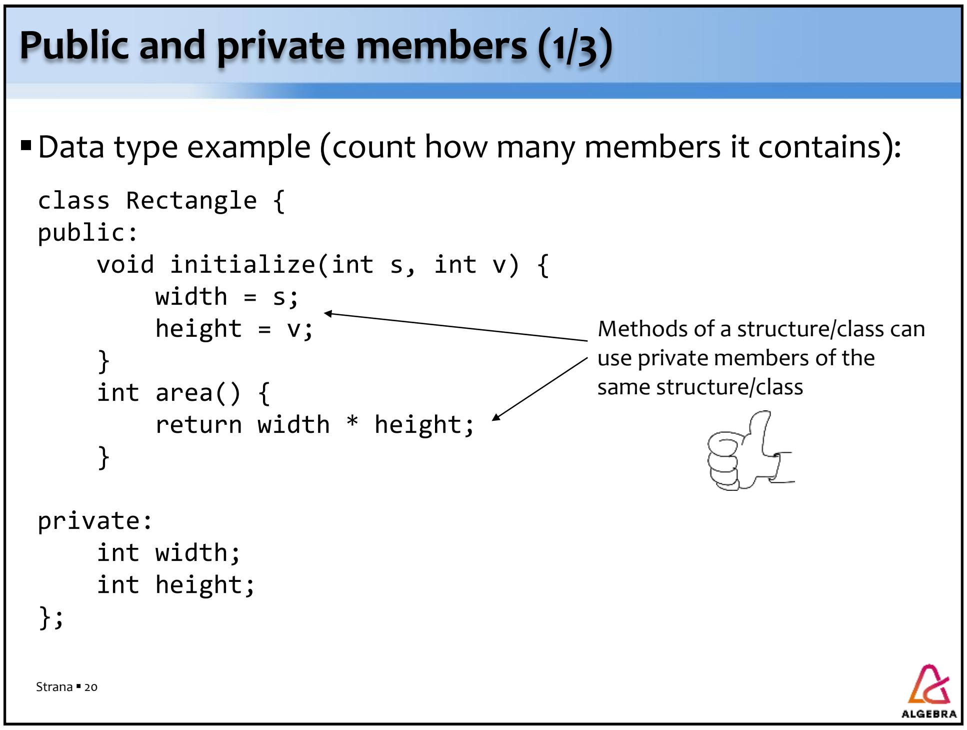
- Google C++ Style Guide:
  - google.github.io/styleguide/cppguide.html
  - „structs should be used for passive objects that carry data, and … lack any functionality other than access/setting the data members. The accessing/setting of fields is done by directly accessing the fields rather than through method invocations. Methods should not provide behavior but should only be used to set up the data members, e.g., constructor, destructor, Initialize(), Reset(), Validate().”
  - „If more functionality is required, a class is more appropriate.”
  - „If in doubt, make it a class.”



Should I use structure or class (3/3)

- Examples:
  - If we want to keep data about height and width of a rectangle
    - Structure, because we only care about the data
  - If we want to keep data about height and width of a rectangle and provide methods for user to draw it, calculate area, perimeter, multiplication with the scalar, etc.
    - Class, because we also act on the data
- Advice:
  - Make class your first choice
  - If you don't have any methods, fallback to structure
  - „If in doubt, make it a class."

Strana ▪ 19



Public and private members (1/3)

- Data type example (count how many members it contains):

```
class Rectangle {
public:
    void initialize(int s, int v) {
        width = s;
        height = v;
    }
    int area() {
        return width * height;
    }

private:
    int width;
    int height;
};
```

Methods of a structure/class can use private members of the same structure/class

Strana ▪ 20

Public and private members (2/3)

- Example of usage:

main can use public members

```
int main() {
    Rectangle p;
    p.initialize(10, 5);
    cout << p.area() << endl;
    ~~cout << p.width << endl;~~
    ~~cout << p.height << endl;~~

    return 0;
}
```

main cannot use private members

- Neither main nor other functions outside a structure/class cannot use private members of that structure/class

Public and private members (3/3)

- Public and private members help implement OOP concept called encapsulation:
  - Public members define an interface that is used by outside world
  - Private members server to hold object state
  - Much more details: OOP
- We will use the following procedure:
  - Public members will be: all methods that will be used from main
  - Private members will be: all variables (always) and those functions that won't be used from main

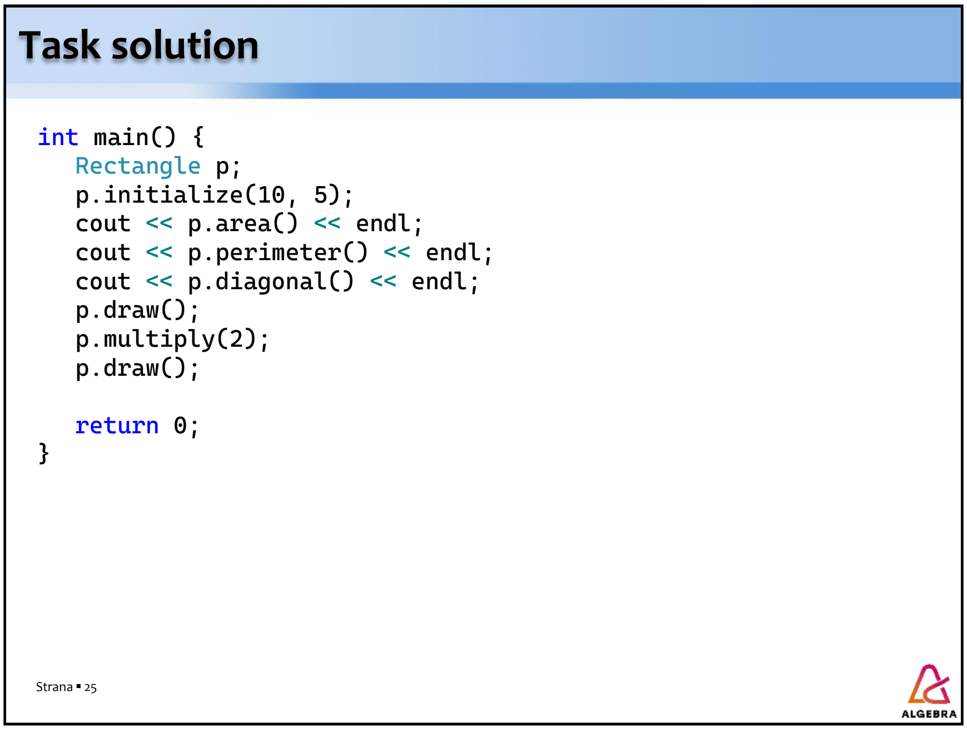
Two short questions

```
class Person {
    string name;
    void initialize(string n, string ln) {
        name = n;
        last_name = ln;
    }
    void display() {
        cout << name << " " << last_name << endl;
    }
    string last_name;
};
```

1. Which members can be used from the `main`-a?
2. Which members should be private and which public?
3. Lest rearrange…

Task

- Lest solve the following task: define a data type capable of holding information about height and width of a rectangle. Enable multiplication with scalar, calculation of area, perimeter and diagonal. Enable also displaying the rectangle with stars. Demonstrate all operations.
  - Questions that guide us to the answer:
    - Should we choose a structure or a class?
    - Which members will be private and which public?
    - If width and height are private, how can we set them from the main?



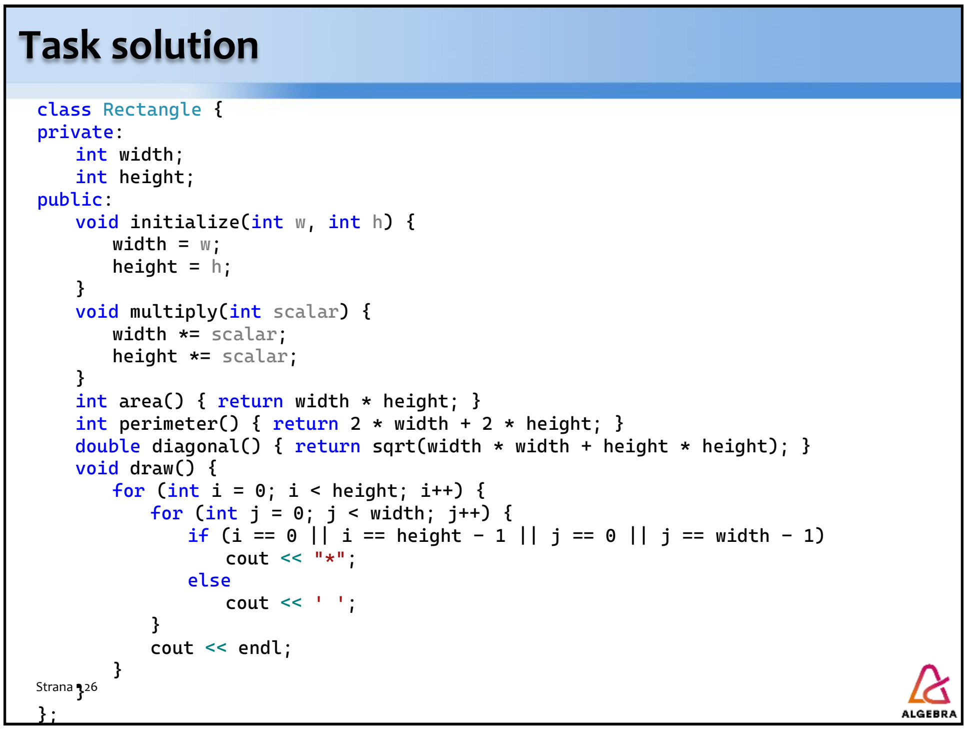
Task solution

```
int main() {
   Rectangle p;
   p.initialize(10, 5);
   cout << p.area() << endl;
   cout << p.perimeter() << endl;
   cout << p.diagonal() << endl;
   p.draw();
   p.multiply(2);
   p.draw();

   return 0;
}
```

Strana ▪ 25

ALGEBRA



Task solution

```
class Rectangle {
private:
    int width;
    int height;
public:
    void initialize(int w, int h) {
        width = w;
        height = h;
    }
    void multiply(int scalar) {
        width *= scalar;
        height *= scalar;
    }
    int area() { return width * height; }
    int perimeter() { return 2 * width + 2 * height; }
    double diagonal() { return sqrt(width * width + height * height); }
    void draw() {
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                if (i == 0 || i == height - 1 || j == 0 || j == width - 1)
                    cout << "*";
                else
                    cout << ' ';
            }
            cout << endl;
        }
    }
};
```

Strana ▪ 26

ALGEBRA

PROJECT ORGANIZATION

Problem #1

- Previous solution works, but it is hard to maintain
  - In reality, there can be hundreds of structures and classes
  - Our .cpp file would become huuuuge
- C++ project can contain multiple files:
  - Header files (.h) contain declarations and prototypes
  - Implementation files (.cpp) contain, well, implementations
- Only.cpp get compiled in the following way:
  - Instead of #include, compiler first copies the entire .h file (preprocessing)
  - Each.cpp file is then individually compiled (.cpp => .obj)
  - Finally, all .obj files get linked into.exe

Solution to problem #1 (1/4)

- In order to better organize our code, we will split the class into two files:
  - Rectangle.h will contain a declaration of a class
  - Rectangle.cpp will contain implementations of methods
- Lets add new files to the project

Solution to problem #1 (2/4)

- .h will contain the declaration of the class
  - We can include additional headers as necessary
  - We write only method prototypes:

```
class Rectangle {
private:
   int width;
   int height;
public:
   void initialize(int w, int h);
   void multiply(int scalar);
   int area();
   int perimeter();
   double diagonal();
   void draw();
};
```

Solution to problem #1 (3/4)

- .cpp will contain implementations of all methods
  - We must write class name in front of every method
  - We must include our .h file
  - We can include additional headers as necessary

```
#include <iostream>
#include "Rectangle.h"
using namespace std;

void Rectangle::initialize(int w, int h) {
   width = w;
   height = h;
}
void Rectangle::multiply(int scalar) {
   width *= scalar;
   height *= scalar;
}
int Rectangle::area() { return width * height; }
```

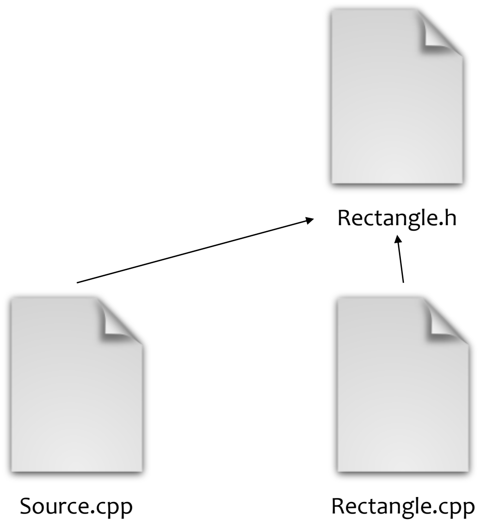
Strana ▪ 31

Solution to problem #1 (4/4)

- After the reorganization, we must make correct includes:
  - Rectangle.cpp will include Rectangle.h
  - Source.cpp will also include Rectangle.h
    - But NOT Rectangle.cpp
    - We never include .cpp files
- Rules:
  - When we include standard headers, we use angle brackets
    - For example: `#include <string>`
  - When we include our headers, we use double quotation marls
    - For example: `#include "Rectangle.h"`

Strana ▪ 32

Include example



Problem #2

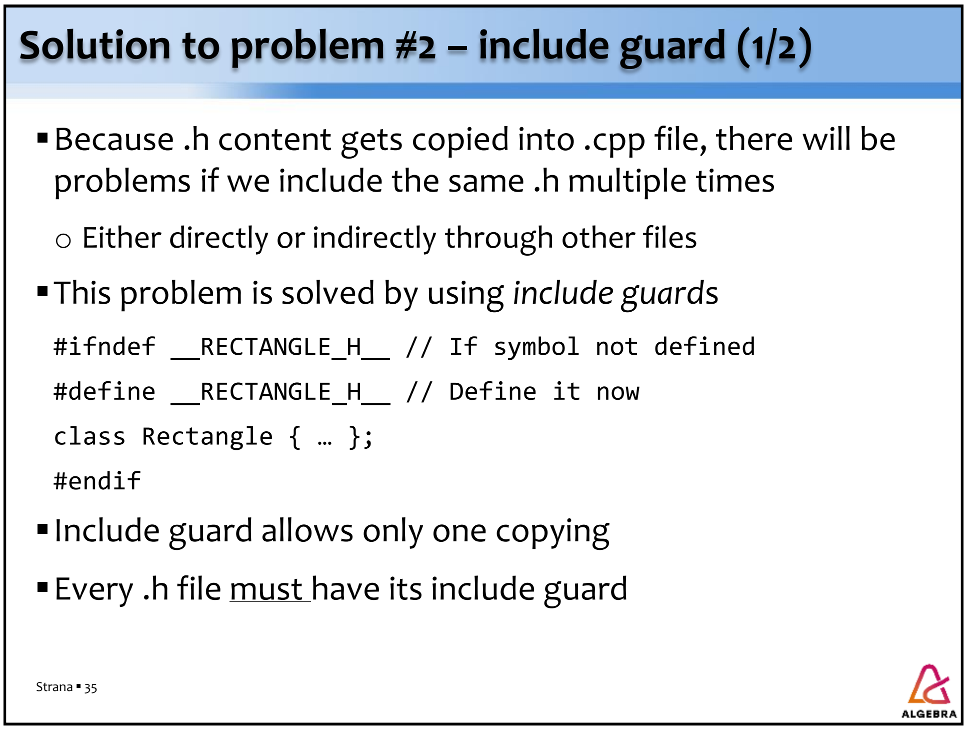
- Lets change the beginning of Source.cpp to this:

```
#include <string>
#include <string>
#include "Rectangle.h"
```

- Can we compile this?
- Now, lets change it to this:

```
#include <string>
#include "Rectangle.h"
#include "Rectangle.h"
```

- Can we compile this?
  - What’s the error saying?



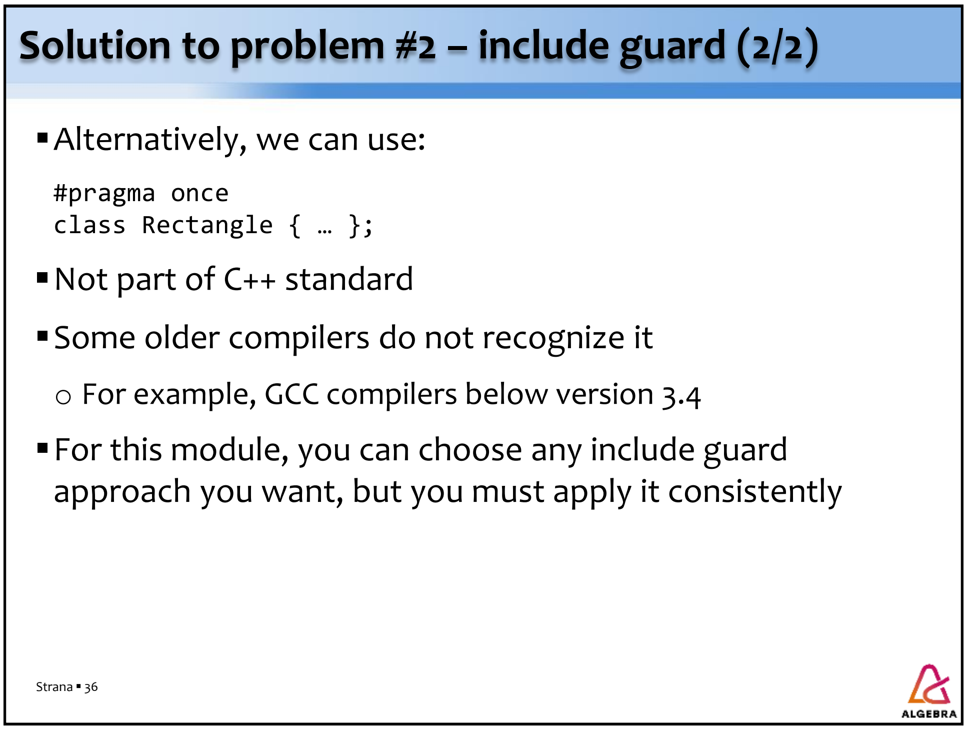
Solution to problem #2 – include guard (1/2)

- Because .h content gets copied into .cpp file, there will be problems if we include the same .h multiple times
  - Either directly or indirectly through other files
- This problem is solved by using *include guards*

```
#ifndef __RECTANGLE_H__ // If symbol not defined
#define __RECTANGLE_H__ // Define it now
class Rectangle { … };
#endif
```

- Include guard allows only one copying
- Every .h file must have its include guard

Strana ▪ 35



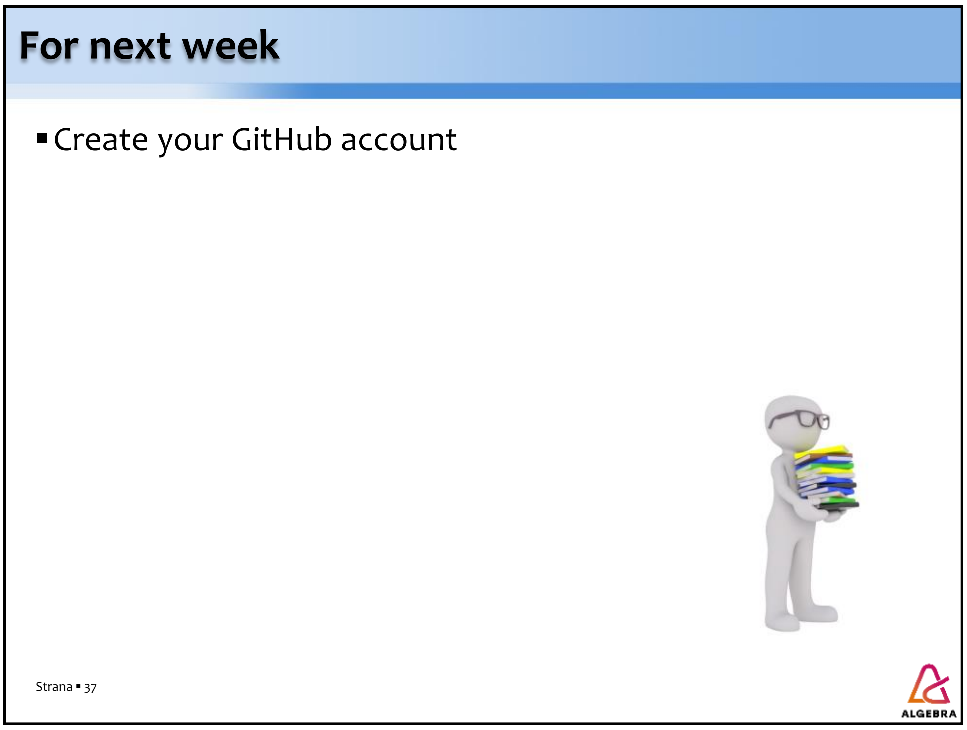
Solution to problem #2 – include guard (2/2)

- Alternatively, we can use:

```
#pragma once
class Rectangle { … };
```

- Not part of C++ standard
- Some older compilers do not recognize it
  - For example, GCC compilers below version 3.4
- For this module, you can choose any include guard approach you want, but you must apply it consistently

Strana ▪ 36



For next week

- Create your GitHub account

Strana ▪ 37