C++ Development

Lecture 3

Schedule

- References
- Inheritance
- More about the constructors

Questions about the homework ?

References

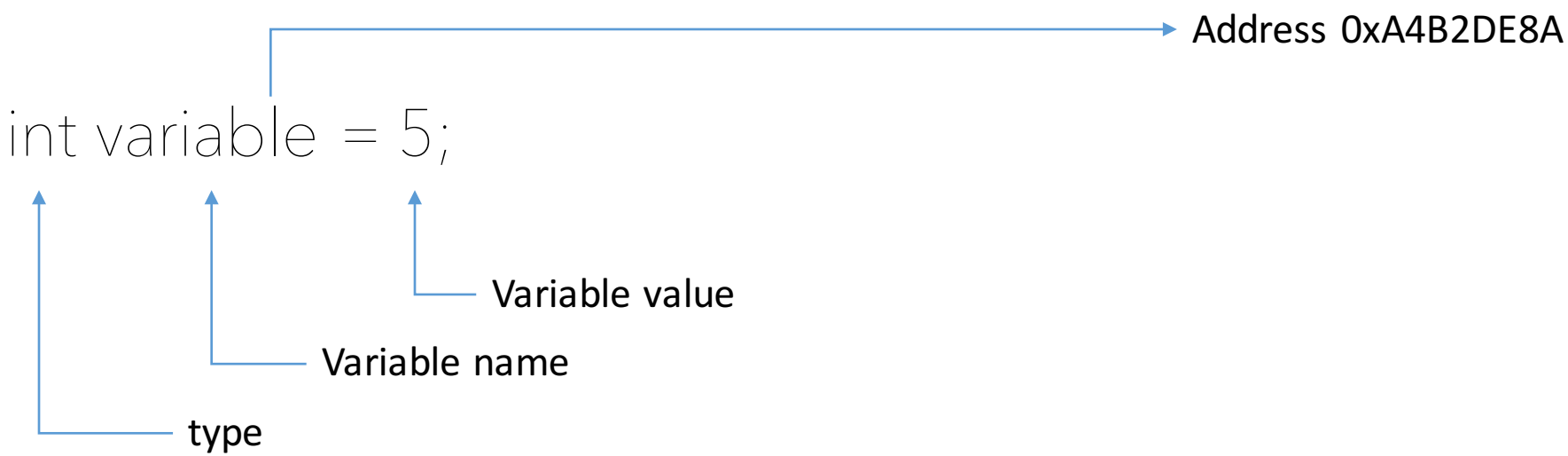
На български - псевдоними

First of all let’s talk about pointers

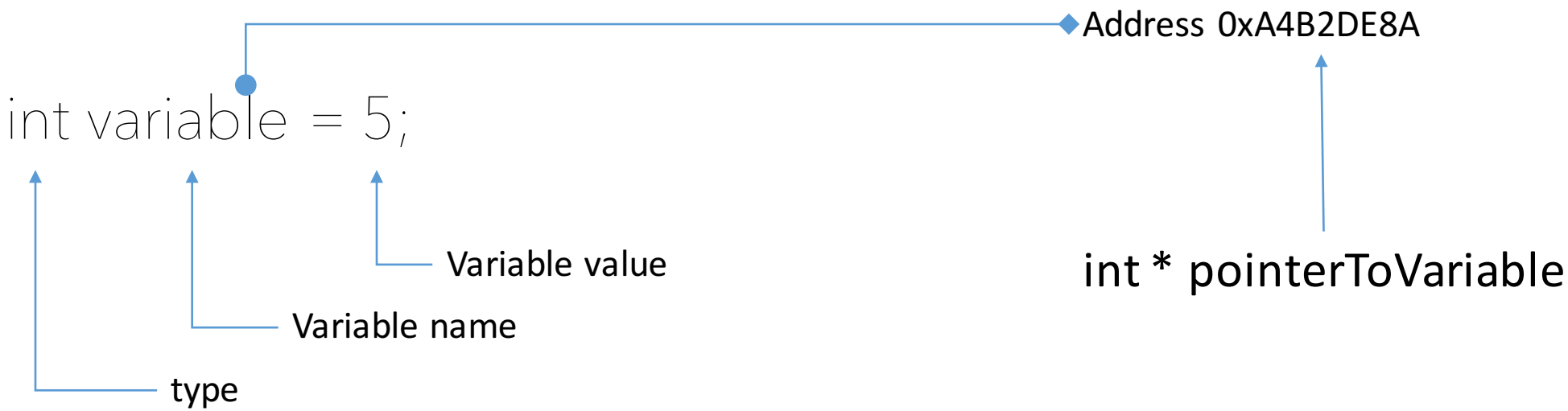
Pointer

- Something that point to somewhere. Like a road sign for the program.

Pointer


Address 0xA4B2DE8A
int variable = 5;
Variable value
Variable name
type

Pointer


Address 0xA4B2DE8A
int variable = 5;
Variable value
Variable name
type
int * pointerToVariable

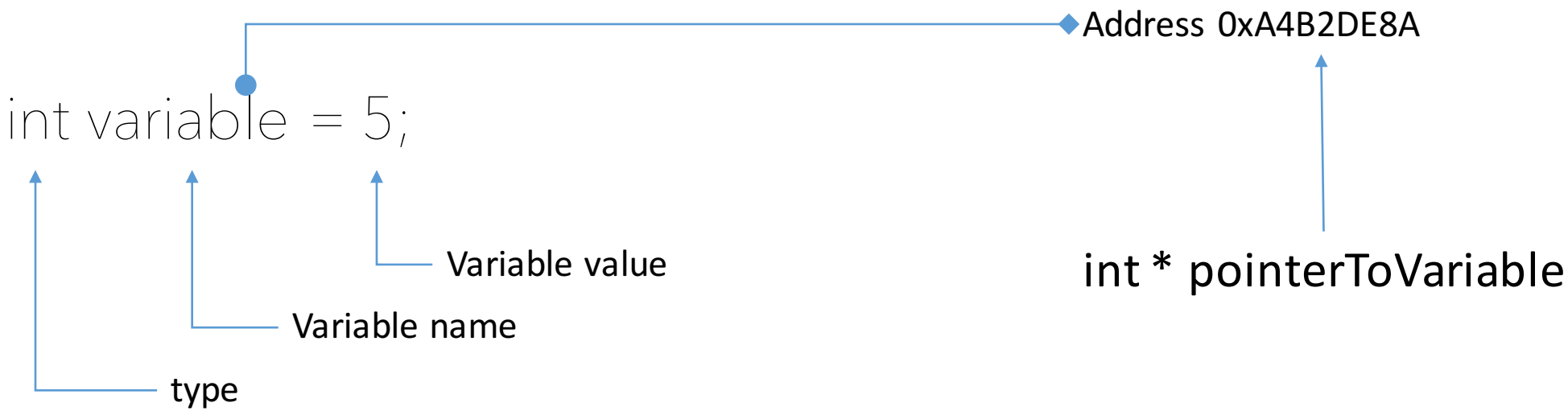
Pointer

```c
int main()
{
    int variable = 5;
    printf("%d, with address: %p\n", variable, &variable);
    int * pointerToVariable = &variable;
    printf("%d, pointing to address %p\n", *pointerToVariable, pointerToVariable);
    return 0;
}
```

To make it clear

- int variable = 5;
- if we call "variable" we get the value stored on the memory
- If we call "&variable" we get the address of the memory
- If we call have int * pointerToVariable = &variable :
  - If we call "pointerToVariable" we get the address of variable
  - If we call "*pointerToVariable" we get the value that is stored on the address of variable

Pointer


Address 0xA4B2DE8A
int variable = 5;
Variable value
Variable name
type
int * pointerToVariable

What about arrays

```c
int main()
{
    int array[] = {1,12,23,34,45,56,67,78,89,90};
    int * pointerToArray = array;
    printf("%d", pointerToArray[4]);
    return 0;
}
```

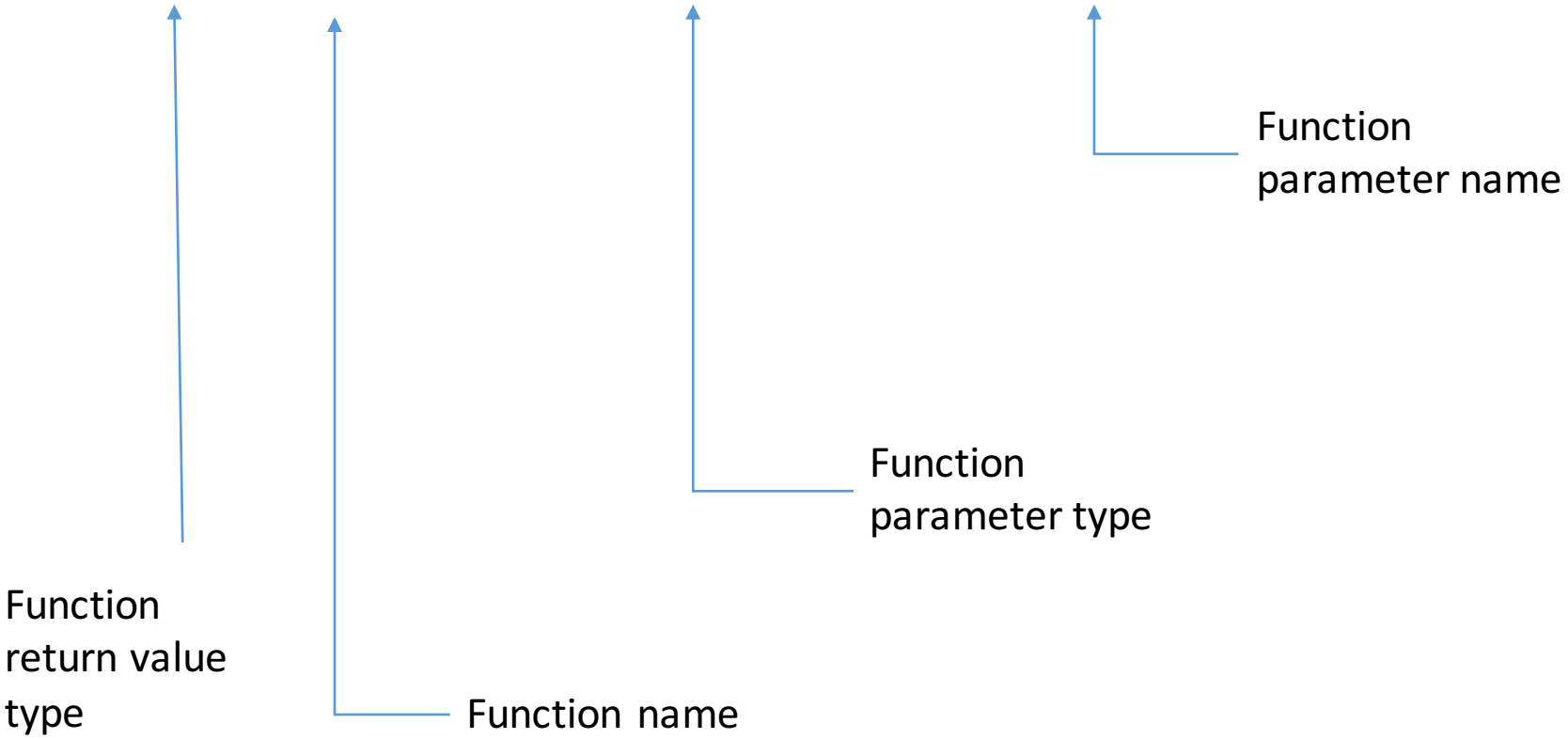
What about objects

```
class Human
{
    public:
    std::string name;
    int age;

};

int main()
{
    Human Pesho = Human();
    Pesho.name = "Pesho";
    Pesho.age = 21;

    Human * pointerToPesho = &Pesho;
    std::cout<<pointerToPesho->name<<" "<<pointerToPesho->age;
    return 0;
}
```

Lets talk about how we call functions and set parameters to them.

A simple function

- int getValue(parameter_t parameter);



What is happening when we do this

- int something = getName(aParameter);

Description

- The parameter is taken and copied to the stack memory.
- Then it is used and modified in the function and if changed this change is not affecting the parameter passed in the function. (with other words this change is safe)

When this case is useful ?

When this case is useful

- When passing simple value type (int, char etc.)
- When we need to compute simple data
- When we need to safely compute data without changing the passed parameter

But what to do when want to pass an object

- We can always pass the object as simple parameter, but is it correct ?

- The answer is not at all.
- It will work, you can test some stuff, but your program may run slowly

Code example

```
class rectangle
{
    public:
    double sideA;
    double sideB;
};

double calcPerimeter(rectangle &rect)
{
    return 2*rect.sideA + 2*rect.sideB;
}

int main()
{
    rectangle rect;
    rect.sideA = 5;
    rect.sideB = 8;
    std::cout << calcPerimeter(rect);
    return 0;
}
```

Lets change few lines

```
class rectangle
{
    public:
    double sideA;
    double sideB;
};

double calcPerimeter(rectangle &rect)
{
    rect.sideA += 1;
    return 2*rect.sideA + 2*rect.sideB;
}

int main()
{
    rectangle rect;
    rect.sideA = 5;
    rect.sideB = 8;
    std::cout << calcPerimeter(rect) << std::endl;
    std::cout << rect.sideA;
    return 0;
}
```

What is the problem ?

What is the problem

- When you change something inside the function, you change the whole object (variable).
- This may lead to misbehavior in the software

How to fix it

- ~~Carefully check every function for unintentional changes of the object~~

Const

- Adding the special word const in front of the variable will tell the program that you must not change the value of the variable.
- Example function
- Double calcPerimeter(const rectangle &rect);

Let's try

```
class rectangle
{
    public:
    double sideA;
    double sideB;
};

double calcPerimeter(const rectangle &rect)
{
    rect.sideA += 1;
    return 2*rect.sideA + 2*rect.sideB;
}

int main()
{
    rectangle rect;
    rect.sideA = 5;
    rect.sideB = 8;
    std::cout << calcPerimeter(rect) << std::endl;
    std::cout << rect.sideA;
    return 0;
}
```

): can’t build

```
 In function 'double calcPerimeter(const rectangle&)':
14:16: error: assignment of member 'rectangle::sideA' in read-only object
```

Const

- If there is a change in the source code where we have explicitly defined that this variable is const the compiler will tell us if we are having problem

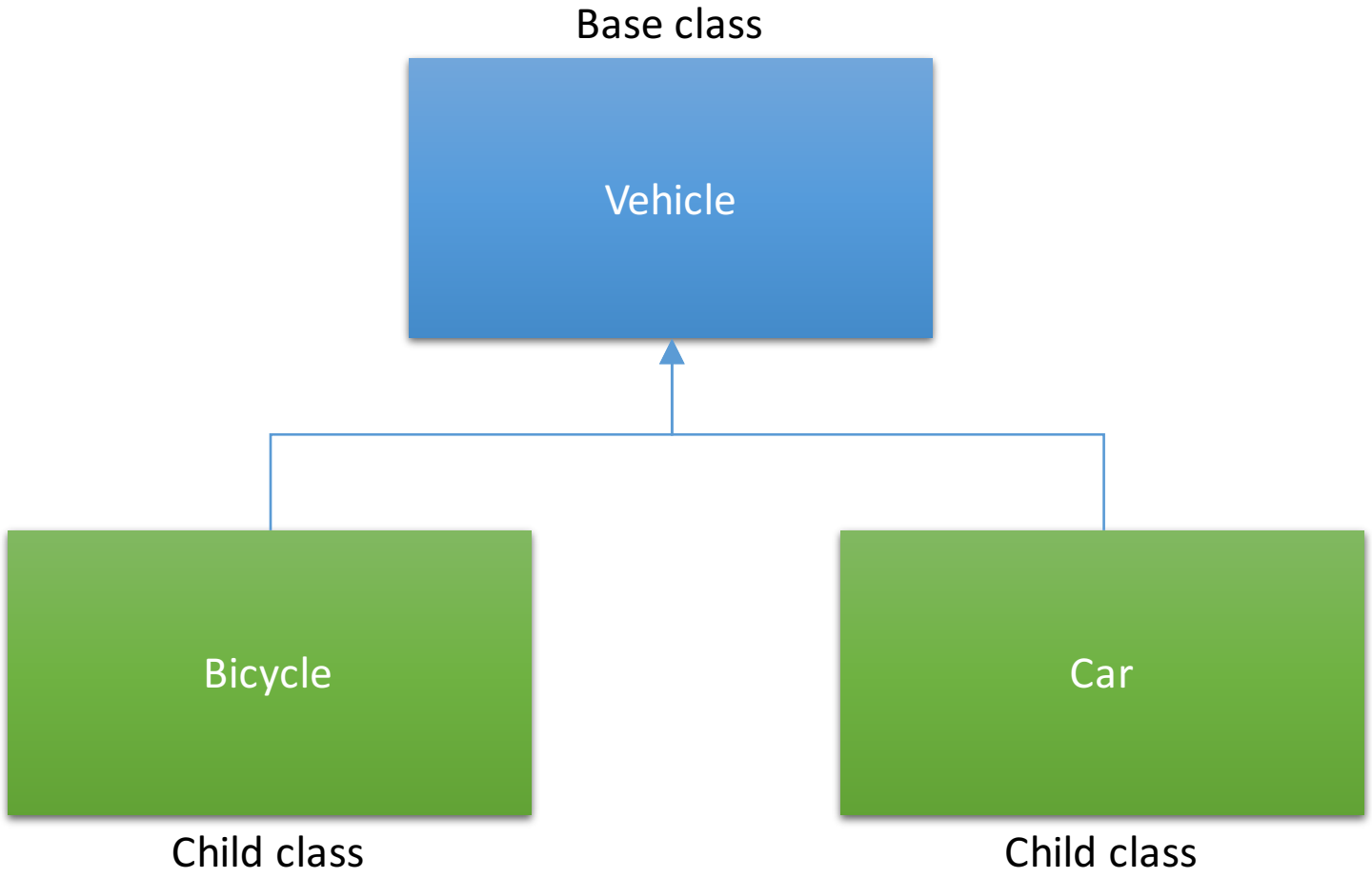
Task

- Create a class called cube and add external functions for calculating perimeter, surface and volume.

Inheriatance

Basics of inheritance

What is inheritance



Inheritance

- Inheritance is the process in which, a class can inherit functions and members from another class. That means that the inheriting class will have a base structure equal to the inherited class and this structure can be changed and improved.

Lets implement

The base class (parent class)

```
class Vehicle
{
    public:
    int numberOfUsers;
    float maxSpeed;
};
```

First derived class (child class)

```
class Bicycle: public Vehicle
{
    public:
    int numberOfGears;
};
```

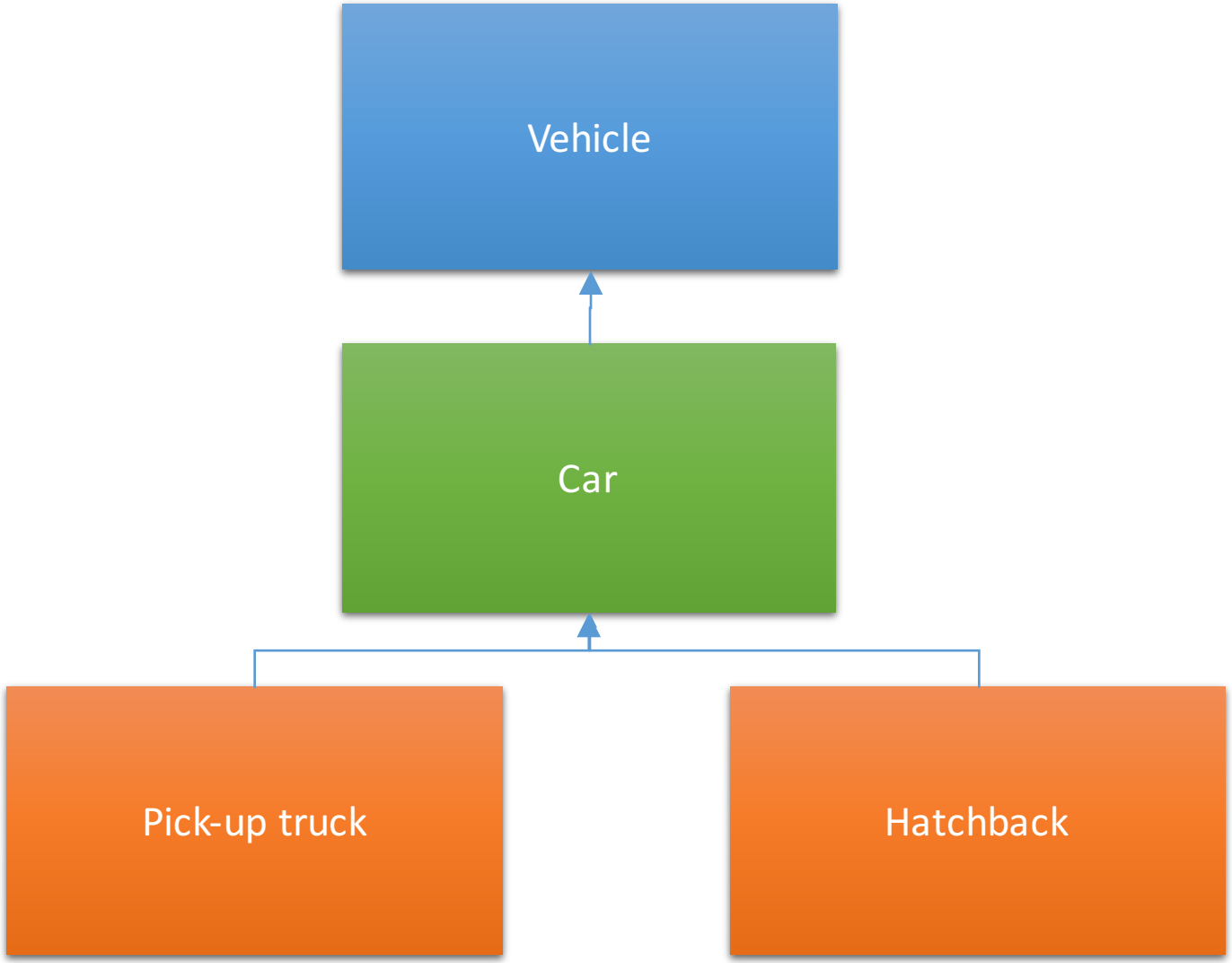
Second derived class (child class)

```
class Car: public Vehicle
{
    public:
    float horsePower;
    int numberOfSeats;
    float rimSize;
    std::string make;
    std::string model;
};
```

Main function

```
int main()
{
    Bicycle myBike;
    myBike.numberOfGears = 18;
    myBike.maxSpeed = 50;

    Car myCar;
    myCar.horsePower = 450;
    myCar.numberOfSeats = 2;
    myCar.maxSpeed = 380;

    std::cout << myBike.maxSpeed << std::endl;
    std::cout << myCar.maxSpeed << std::endl;
    return 0;
}
```

Multistage inheritance



Lets try to implement multistage inheritance

Why we do inheritance ?

Why we do inheritance

- Can separate a class into many smaller classes with more specific functions
- More functionalities can be reused when inheriting
- Functions with same name but different behavior can be implemented in different child classes
- For classes that have a lot in common we can implement same protocol for the classes, hence make them easier to be used.
- Child classes can extend functionalities of the base classes

What is inherited between the classes

- Everything but
  - Constructors
  - Destructor
  - Redefined operators
  - Friend functions and classes

Deeper in the class constructors

- class derived-class: access base-class

Class members inheritance

- We have been talking about public, private, protected

Class members inheritance

- Public – all inherited members and functions retain their access rights
- Protected – all inherited public members and functions access changed to protected, others retain their access rights
- Private – all inherited members and functions are changed to prived

Lets make an experiment

- Lets make class Boss, inherited by Manager, inherited by Worker.
- The members are bossSalary, managerSalary, workerSalary

Now lets extend even more

- Class wife
- Class FriendOfWife
- Class FriendOfWorker

Polymorphism

Polymorphism

- The functionality of OOP where we can redefine the functionality(implementation) of a function in derived classes. Hence functions with same name can do different things.

Example

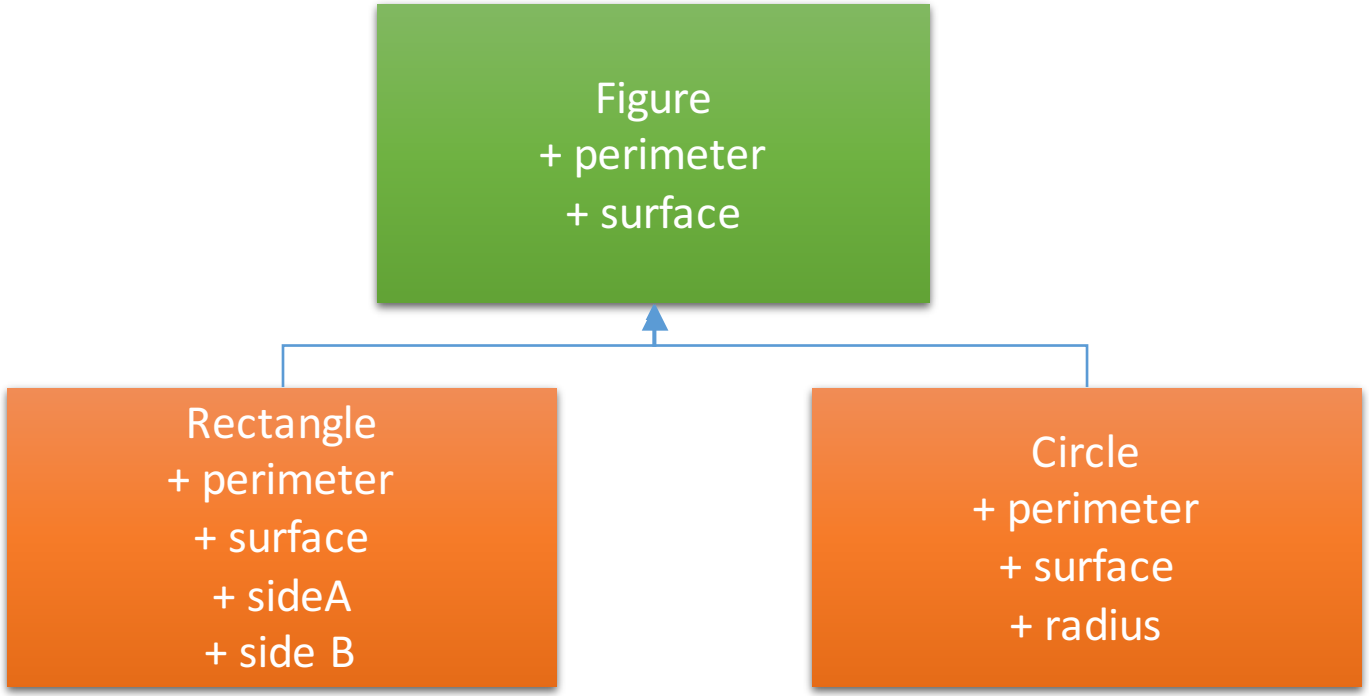


Figure
+ perimeter
+ surface
Rectangle
+ perimeter
+ surface
+ sideA
+ side B
Circle
+ perimeter
+ surface
+ radius

Implementation

Task

- Extend Rectangle with derived class called Square

Task: Extent knowledge from the last lecture

Initialization list constuctor

```
class Vehicle
{
    public:
    Vehicle(){};
    Vehicle(int _numberOfUsers, float _maxSpeed): numberOfUsers(_numberOfUsers), maxSpeed(_maxSpeed) {} ;
    int numberOfUsers;
    float maxSpeed;
};
```

Initialization list constructor

- ConstructorName(parameters): variable(value) {};

Preprocessor special words

#include ”headerName.h”

- Used for including different headers.
- Use ”headerName.h” for headers defined at specific path in the OS
- Use <headerName> for headers define by the OS and included in it’s core libraries

#define VAR_NAME VALUE

- Used for defining a constant into a program.
- It is changing the VAR_NAME with VALUE everywhere in the code

#ifdef (VAR_NAME)

- #ifdef is used to check if we have defined specific definition name
- If the condition is true the code below it is being compiled, else it is being skipped
- The block must end with #endif.

#ifndef (VAR_NAME)

- The same priciple as #ifdef, but it is getting in the block if a variable is not defined.

enum enum_name {value1, value2, value3}

- It is used to define certain constant variables.

Daily task

- Look at the homework