

Programming in C++ - Primer

Lesson 5 - Objects

Jakub 'Eremiell' Marek
<marekj14@fel.cvut.cz>

Silicon Hill C++ Academy

2013/11/18

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

1 Revision

2 OOP

3 Objects

4 Advanced Topics

Welcome!

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Passing value by:

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Passing value by:

- Value

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Passing value by:

- Value
- Reference

Passing value by:

- Value
- Reference

Pointers are:

Passing value by:

- Value
- Reference

Pointers are:

- Addresses of memory

Passing value by:

- Value
- Reference

Pointers are:

- Addresses of memory
- Variables like any other

Passing value by:

- Value
- Reference

Pointers are:

- Addresses of memory
- Variables like any other

Two kinds of pointers:

Passing value by:

- Value
- Reference

Pointers are:

- Addresses of memory
- Variables like any other

Two kinds of pointers:

- Pointers *

Passing value by:

- Value
- Reference

Pointers are:

- Addresses of memory
- Variables like any other

Two kinds of pointers:

- Pointers *
- References &

Pointers

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Two pointer related operators:

Two pointer related operators:

- Reference &

Two pointer related operators:

- Reference &
- Deference *

Two pointer related operators:

- Reference &
- Defererence *

Segfaults

Two pointer related operators:

- Reference &
- Deference *

Segfaults

Command line arguments

Two pointer related operators:

- Reference &
- Defererence *

Segfaults

Command line arguments

Two kinds of memory allocation:

Two pointer related operators:

- Reference &
- Defererence *

Segfaults

Command line arguments

Two kinds of memory allocation:

- Static

Two pointer related operators:

- Reference &
- Defererence *

Segfaults

Command line arguments

Two kinds of memory allocation:

- Static
- Dynamic

Two pointer related operators:

- Reference &
- Defererence *

Segfaults

Command line arguments

Two kinds of memory allocation:

- Static
- Dynamic

Dynamic allocation operators:

Two pointer related operators:

- Reference &
- Defererence *

Segfaults

Command line arguments

Two kinds of memory allocation:

- Static
- Dynamic

Dynamic allocation operators:

- new

Two pointer related operators:

- Reference &
- Deference *

Segfaults

Command line arguments

Two kinds of memory allocation:

- Static
- Dynamic

Dynamic allocation operators:

- new
- delete, delete[]

Containers and Structs

C++ Primer

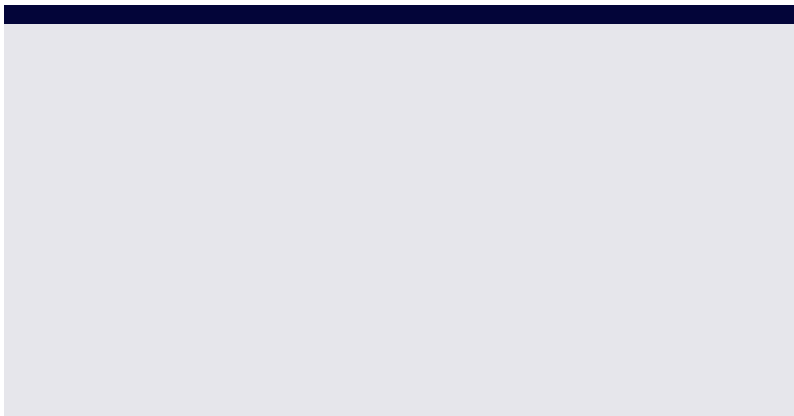
Jakub Marek

Revision

OOP

Objects

Advanced
Topics



Containers and Structs

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Containers

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Containers

■ Strings

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Containers

- Strings
- Vectors

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Containers

- Strings
- Vectors
- and many others...

Containers and Structs

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Containers

- Strings
- Vectors
- and many others...

Structs

Containers and Structs

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Containers

- Strings
- Vectors
- and many others...

Structs

- aggregate data types

Containers and Structs

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Containers

- Strings
- Vectors
- and many others...

Structs

- aggregate data types
- struct

Containers and Structs

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Containers

- Strings
- Vectors
- and many others...

Structs

- aggregate data types
- struct
- union

Programming Styles

C++ Primer

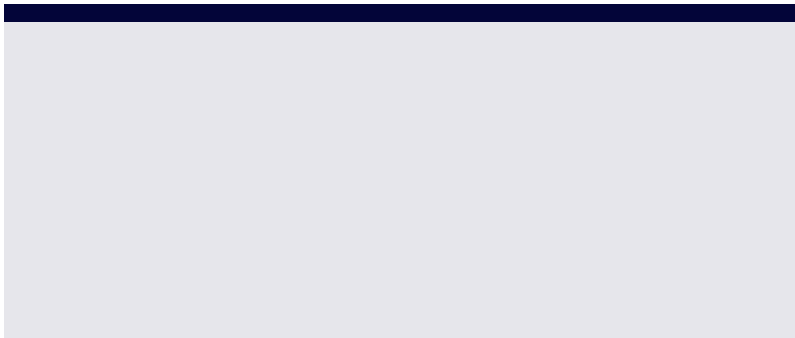
Jakub Marek

Revision

OOP

Objects

Advanced
Topics



Programming Styles

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

What we've been to before?

Programming Styles

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

What we've been to before?

- naive programming

Programming Styles

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

What we've been to before?

- naive programming
- procedural programming

Programming Styles

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

What we've been to before?

- naive programming
- procedural programming
- object oriented programming

Programming Styles

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

What we've been to before?

- naive programming
- procedural programming
- object oriented programming

other branches like functional programming

Programming Styles

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

What we've been to before?

- naive programming
- procedural programming
- object oriented programming

other branches like functional programming
the best of it usually integrated

Programming Styles

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

What we've been to before?

- naive programming
- procedural programming
- object oriented programming

other branches like functional programming
the best of it usually integrated
still new & unpredictable

Why OOP?

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Why OOP?

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

easier to maintain

Why OOP?

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

easier to maintain
things that belong together are together

Why OOP?

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

easier to maintain
things that belong together are together
tighter logic

Why OOP?

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

easier to maintain
things that belong together are together
tighter logic
considered best practice

Questions?

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Objects

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Objects

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

better structs

Objects

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

better structs
their own member functions = methods

Objects

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

better structs
their own member functions = methods
their own visibility layers

Objects

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

better structs
their own member functions = methods
their own visibility layers
inheritance & polymorphism

Access Specifiers

C++ Primer

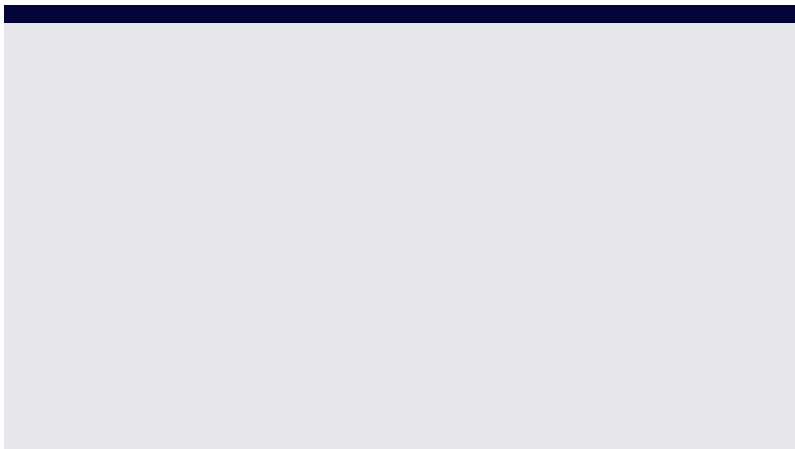
Jakub Marek

Revision

OOP

Objects

Advanced
Topics



Access Specifiers

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

new keywords

Access Specifiers

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

new keywords
important for data encapsulation

Access Specifiers

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

new keywords
important for data encapsulation
define member visibility

Access Specifiers

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

new keywords
important for data encapsulation
define member visibility

`public` everyone sees it

Access Specifiers

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

new keywords

important for data encapsulation

define member visibility

public everyone sees it

protected only instances of this class and it's derived classes
and friends

Access Specifiers

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

new keywords

important for data encapsulation

define member visibility

`public` everyone sees it

`protected` only instances of this class and it's derived classes
and friends

`private` only instances of this class and friends

Access Specifiers

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

new keywords

important for data encapsulation

define member visibility

`public` everyone sees it

`protected` only instances of this class and it's derived classes
and friends

`private` only instances of this class and friends

best practice:

Access Specifiers

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

new keywords

important for data encapsulation

define member visibility

public everyone sees it

protected only instances of this class and it's derived classes
and friends

private only instances of this class and friends

best practice:

start with public block, then protected, then private

Access Specifiers

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

new keywords

important for data encapsulation

define member visibility

public everyone sees it

protected only instances of this class and it's derived classes
and friends

private only instances of this class and friends

best practice:

start with public block, then protected, then private
(biggest audience first)

Encapsulation

C++ Primer

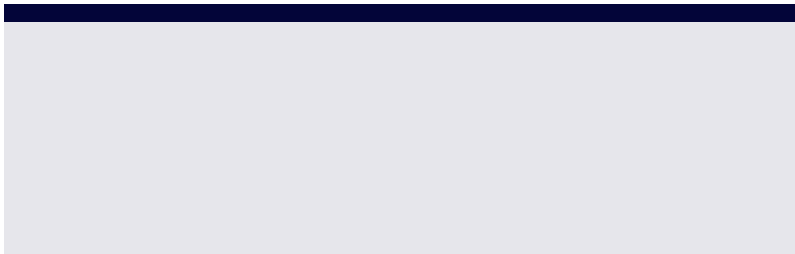
Jakub Marek

Revision

OOP

Objects

Advanced
Topics



Encapsulation

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

data should never be freely reachable by anyone

Encapsulation

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

data should never be freely reachable by anyone
because if they are, anyone can change them!

Encapsulation

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

data should never be freely reachable by anyone
because if they are, anyone can change them!
so we cannot guard the consistence of our objects

Encapsulation

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

data should never be freely reachable by anyone
because if they are, anyone can change them!
so we cannot guard the consistence of our objects
members are usually private or protected

Encapsulation

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

data should never be freely reachable by anyone
because if they are, anyone can change them!
so we cannot guard the consistence of our objects
members are usually private or protected
getter/setter methods in public part

Encapsulation

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

data should never be freely reachable by anyone
because if they are, anyone can change them!
so we cannot guard the consistence of our objects
members are usually private or protected
getter/setter methods in public part
guarding of the contents

Constructor/Destructor

C++ Primer

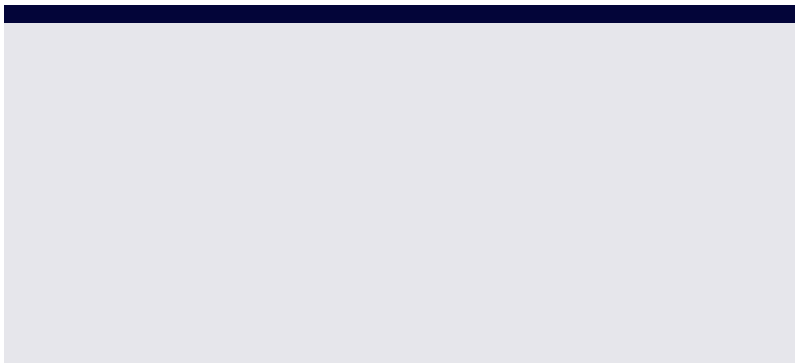
Jakub Marek

Revision

OOP

Objects

Advanced
Topics



Constructor/Destructor

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Constructor brings object to its initial state

Constructor/Destructor

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Constructor brings object to its initial state
Destructor frees all allocated memory and prepares object for deletion

Constructor/Destructor

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Constructor brings object to its initial state
Destructor frees all allocated memory and prepares object for deletion
often shorthand as ctor/dtor

Constructor/Destructor

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Constructor brings object to its initial state

Destructor frees all allocated memory and prepares object for deletion

often shorthand as ctor/dtor

named same as class

Constructor/Destructor

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Constructor brings object to its initial state

Destructor frees all allocated memory and prepares object for deletion

often shorthand as ctor/dtor

named same as class

never have return types (unlike any other functions)

Constructor/Destructor

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Constructor brings object to its initial state

Destructor frees all allocated memory and prepares object for deletion

often shorthand as ctor/dtor

named same as class

never have return types (unlike any other functions)

dtors start with ~

Constructor/Destructor

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Constructor brings object to its initial state

Destructor frees all allocated memory and prepares object for deletion

often shorthand as ctor/dtor

named same as class

never have return types (unlike any other functions)

dtors start with ~

if you write none, default ones will be provided

Constructor/Destructor

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Constructor brings object to its initial state
Destructor frees all allocated memory and prepares object for deletion
often shorthand as ctor/dtor
named same as class
never have return types (unlike any other functions)
dtors start with ~
if you write none, default ones will be provided
if you write any, default ones will be gone!

Access Operators

C++ Primer

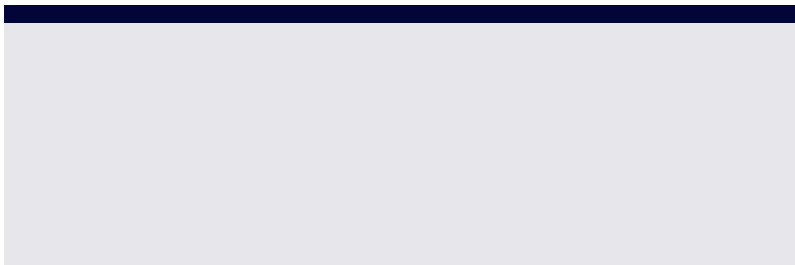
Jakub Marek

Revision

OOP

Objects

Advanced
Topics



Access Operators

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

```
class creates namespace
```

Access Operators

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

```
class creates namespace  
instance creates inner block
```

Access Operators

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

```
class creates namespace  
instance creates inner block  
  
:: operator class members
```

Access Operators

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

```
class creates namespace
instance creates inner block

:: operator class members
. operator instance members
```

Access Operators

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

```
class creates namespace
instance creates inner block

:: operator class members
. operator instance members
-> operator instance members via pointers
```

Declaration (Header file)

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

```
#ifndef HEADER_HPP
#define HEADER_HPP

#include <string>
#include <cassert>

class Hero {
public:
    Hero(std::string name, int maxHealth = 100) : name(name),
        maxHealth(maxHealth), health(maxHealth) {};
    ~Hero() {};
    int getHealth();
    int getMaxHealth();
    std::string getName();
    int harm(int);
    int heal(int);
protected:
    int health;
    int maxHealth;
    std::string name;
    bool checkAlive();
    void testInvariant();
};

#endif
```

Definition (Source file)

C++ Primer

Jakub Marek

Revision

OPP

Objects

Advanced
Topics

```
#include "header.hpp"

int Hero::getHealth() {
    return health;
}

int Hero::getMaxHealth() {
    return maxHealth;
}

std::string Hero::getName() {
    return name;
}

int Hero::harm(int i) {
    health -= i;
    if (health < 0) {
        health = 0;
    }
    checkAlive();
    testInvariant();
    return health;
}
```

Definition (Source file)

C++ Primer

Jakub Marek

Revision

OOO

Objects

Advanced
Topics

```
int Hero::heal(int i) {
    health += i;
    if (health > maxHealth) {
        health = maxHealth;
    }
    testInvariant();
    return health;
}

bool Hero::checkAlive() {
    return health > 0;
}

void Hero::testInvariant() {
    assert (health >= 0 && health <= maxHealth);
}
```


Questions?

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Advanced Topics

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics



Advanced Topics

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

self-testing code

Advanced Topics

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

self-testing code
methods are of two kinds:

Advanced Topics

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

self-testing code
methods are of two kinds:
observers and mutators

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

self-testing code
methods are of two kinds:
observers and mutators

- observers just observe

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

self-testing code
methods are of two kinds:
observers and mutators

- observers just observe
- mutators mutate the object and can break it

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

self-testing code
methods are of two kinds:
observers and mutators

- observers just observe
- mutators mutate the object and can break it

so they need to test invariant

self-testing code
methods are of two kinds:
observers and mutators

- observers just observe
- mutators mutate the object and can break it

so they need to test invariant
contracts

Makefiles

C++ Primer

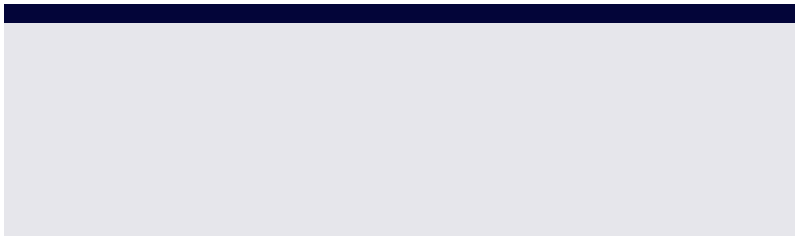
Jakub Marek

Revision

OOP

Objects

Advanced
Topics



so you don't have to compile in hand all the time

```
all:    clean
        g++ -ansi -pedantic -Wall -Wextra -O2 -omyprog myprog.cpp
        module1.cpp module2.cpp -I. -L. -lm
install:
        sudo cp myprog /usr/bin/
clean:
        rm -f *.o
        rm -f myprog
```

Questions?

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics

Break!

C++ Primer

Jakub Marek

Revision

OOP

Objects

Advanced
Topics