ST0244 Programming Languages
4. Object-Oriented Programming

Andrés Sicard-Ramírez

Universidad EAFIT

Semester 2023-2

Preliminaries

Conventions

- The number and page numbers assigned to chapters, examples, exercises, figures, quotes, sections and theorems on these slides correspond to the numbers assigned in the textbook [Lee 2017].
- The source code examples are in course's repository.

Introduction

Concept	Count	%
Inheritance	71	81%
Object	69	78%
Class	62	71%
Encapsulation	55	63%
Method	50	57%
Message Passing	49	56%
Polymorphism	47	53%
Abstraction	45	51%

‘The quarks of object-oriented development’ [Armstrong 2006].

Introduction

Quark's definitions [Armstrong 2006]

(i) '**Inheritance:** A mechanism that allows the data and behavior of one class to be included in or used as the basis for another class.'

(ii) '**Object:** Individual, identifiable item, either real or abstract, which contains data about itself and descriptions of its manipulations of the data.'

(iii) '**Class:** A description of the organization and similar objects.'

(iv) '**Encapsulation:** A technique for designing classes and objects that restricts access to the data and behavior by defining a limited set of messages that an object of that class can receive.'

(continued on next slide)

Introduction

Quark's definitions (continuation)

(v) '**Method:** A way to access, set or manipulate object's information.'

(vi) '**Message passing:** The process by which an object sends data to another object or asks the other object to invoke a method.'

(vii) '**Polymorphism** is defined as: the ability of different classes to respond to the same message and each implement the method appropriately.'

(viii) '**Abstraction:** The act of creating classes to simplify aspects of reality using distinctions inherent to the problem.'

Introduction

Programming languages

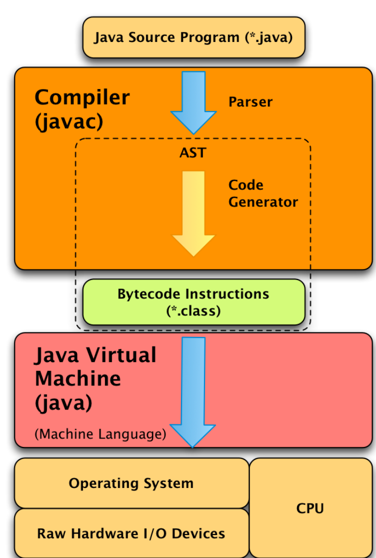
OOP languages include C++, Java, Python, Ruby, Scala and Smalltalk.

The Java Environment

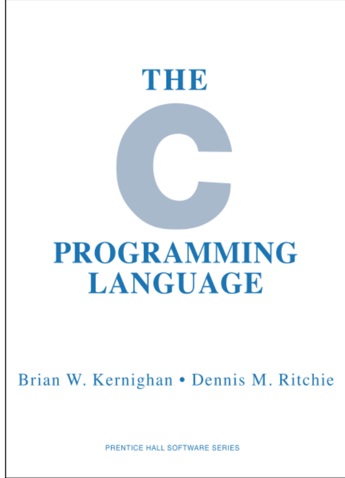
Tools

- Java compiler (command `javac` on Linux): Source code to bytecode.
- Java Virtual Machine (JVM) (command `java` on Linux): Executes the bytecode.

(Fig. 4.4)



About the ‘hello, world’ example



‘The first program to write is the same for all languages: Print the words `hello, world`.’ [1978, §1.1]

The Java Environment

Example

Compile and run the `hw/HelloWorld.java` program by running the followings commands:

```
$ javac HelloWorld.java
$ java HelloWorld
```

The Java Environment

Example

Compiling the `oop/Warning.java` program using the `-Xlint` option:

```
$ javac -Xlint Warning.java
Warning.java:4: warning: [cast] redundant cast to String
      String s = (String)"hello, word!";
                 ^
1 warning
```

The Java Environment

Example

Compiling the oop/Warning.java program using the -Xlint and -Werror options:

```
$ javac -Xlint -Werror Warning.java
Warning.java:4: warning: [cast] redundant cast to String
    String s = (String)"hello, word";
               ^
error: warnings found and -Werror specified
1 error
1 warning
```

The Java Environment

Example

More examples using `-Xlint` keys:

https://docs.oracle.com/en/java/javase/20/docs/specs/man/javac.html#examples-of-using--xlint-keys .

The Java Environment

Example

More examples using `-Xlint` keys:

https://docs.oracle.com/en/java/javase/20/docs/specs/man/javac.html#examples-of-using--xlint-keys .

Remark

For getting information about the extra options and `-Xlint` keys use the following commands, respectively:

```
$ javac --help -X
$ javac --help-lint
```

The C++ Environment

The C++ environment

Fig. 4.5:

https://kentdlee.github.io/PL/build/html/oop.html#the-c-environment

The C++ Environment

Example

Compile and run the `hw/hello-world.cc` program by running the followings commands:

```
$ g++ -o hello-world hello-world.cc
$ ./hello-world
```

The C++ Environment

Example

Compiling the `oop/warning.cc` program using the `-Wall` option:

```
$ g++ -Wall -o warning warning.cc
warnings.cc: In function ‘int main(int, char**)’:
warnings.cc:10:11: warning: ‘i’ is used uninitialized
in this function [-Wuninitialized]
cout << i << endl;
        ^
```

The C++ Environment

Example

Compiling the `oop/warning.cc` program using the `-Wall` and `-Werror` options:

```
$ g++ -Wall -Werror -o warning warning.cc
warnings.cc: In function ‘int main(int, char**)’:
warnings.cc:10:11: warning: ‘i’ is used uninitialized
in this function [-Wuninitialized]
cout << i << endl;
        ^
cc1plus: all warnings being treated as errors
```

The C++ Macro Processor

Description

The C++ **macro processor** is a program that processes **directives**, which give instructions (e.g. for including files, for conditional compilation, for macro definition and expansion, among other) to the compiler to preprocess the source code before the compilation starts.

The C++ Macro Processor

Description

The C++ **macro processor** is a program that processes **directives**, which give instructions (e.g. for including files, for conditional compilation, for macro definition and expansion, among other) to the compiler to preprocess the source code before the compilation starts.

Example

From the line

```
#include <iostream>
```

in `hw/hello-world.cc`, the macro processor includes the `iostream` library.

The C++ Macro Processor

Remark

The C++ macro processor is called cpp.

Example

Using cpp (or equivalently g++ -E):

```
$ cpp hello-world.cc > xxx.cc
```

See the `xxx.cc` file in your favourite editor.

How many lines has the `xxx.cc` file?

```
$ wc -l xxx.cc
28647 xxx.cc
```

The make Tool

Description

> *'The make tool is a program that can be used to compile programs that are composed of modules and utilise separate compilation.' (p. 120)*

The make Tool

Description

> *'The make tool is a program that can be used to compile programs that are composed of modules and utilise separate compilation.' (p. 120)*

Rules

A makefile consists of a set of 'rules' with the following shape:

```
target ... : prerequisites ...
        recipe
        ...
```

The make Tool

Example (make rule)

```
foo : foo.cc
        g++ -o foo foo.cc
```

The make Tool

Make default rule for C++

The default rule for C++ is something like:

```
file : file.cc
     $(CXX) $(CPPFLAGS) $(CXXFLAGS) -o $@ $<
```

The make Tool

Example

See `oop/Makefile` file.

Classes and Objects

Description

*'Object-Oriented programming is all about creating objects. Objects have **state information**, sometimes just called **state**, and **methods** that operate on that state, sometimes altering the state. If we alter the state of an object we call it a **mutable** object. If we cannot alter the object's state once it is created, the object is called **immutable**. A **class** defines the state information maintained by an object and the methods that operate on that state.' (p. 127)*

Classes and Objects

Exercise

The GNU Smalltalk tutorial* shows the implementation of toy home-finance accounting system with three classes: `Account`, `Savings` and `Cheking`.

Implement the system in C++ and Java.

*In `https://www.gnu.org/software/smalltalk/manual/html_node/Tutorial.html`.

Classes and Objects

Example (class for vectors in C++)

From file `oop/vector.cc`:

- Private and public members

*Figure from [Stroustrup 2019, p. 24].

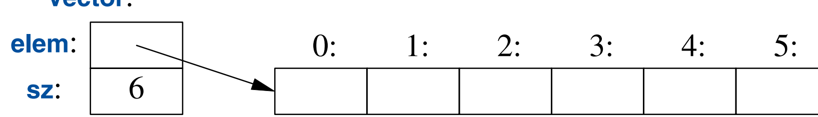
Classes and Objects

Example (class for vectors in C++)

From file `oop/vector.cc`:

- Private and public members
- Constructor

  Graphical representation of `Vector v(6)`:*



(continued on next slide)

*Figure from [Stroustrup 2019, p. 24].

Classes and Objects

Example (continuation)

- Initialisation

  Graphical representation of `Vector v(6)`:*

*Figure from [Stroustrup 2019, p. 52].

Classes and Objects

Example (continuation)

- Initialisation

  Graphical representation of `Vector v(6)`:*

Vector:						
elem:	→	0:	1:	2:	3:	4:	5:
sz: 6		0	0	0	0	0	0

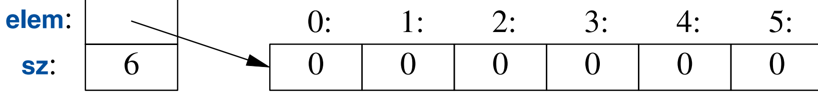
- Destructor

*Figure from [Stroustrup 2019, p. 52].

Example (continuation)

- Initialisation

  Graphical representation of `Vector v(6)`:*



- Destructor
- Use of `const`.

*Figure from [Stroustrup 2019, p. 52].

Inheritance and Polymorphism

Definition

*'**Inheritance** is a relationship among classes wherein one class shares the structure and/or behavior defined in one (**single inheritance**) or more (**multiple inheritance**) other classes. We call the class from which another class inherits its **superclass**... Similarly, we call a class that inherits from one or more classes a **subclass**.' (Booch, Maksimchuk, Engle, Young, Conallen and Houston 2007, p. 100)*

Inheritance and Polymorphism

Definition

*‘**Inheritance** is a relationship among classes wherein one class shares the structure and/or behavior defined in one (**single inheritance**) or more (**multiple inheritance**) other classes. We call the class from which another class inherits its **superclass**. . . Similarly, we call a class that inherits from one or more classes a **subclass**.’ (Booch, Maksimchuk, Engle, Young, Conallen and Houston 2007, p. 100)*

*‘**Inheritance** is: a mechanism that allows the data and behavior of one class to be included in or used as the basis for another class.’ (Armstrong 2006, p. 124)*

Inheritance and Polymorphism

Definition

*'**Inheritance** is a relationship among classes wherein one class shares the structure and/or behavior defined in one (**single inheritance**) or more (**multiple inheritance**) other classes. We call the class from which another class inherits its **superclass**...Similarly, we call a class that inherits from one or more classes a **subclass**.' (Booch, Maksimchuk, Engle, Young, Conallen and Houston 2007, p. 100)*

*'**Inheritance** is: a mechanism that allows the data and behavior of one class to be included in or used as the basis for another class.' (Armstrong 2006, p. 124)*

*'**Inheritance** is the mechanism we employ to re-use code in software we are currently writing.' (p. 131)*

Inheritance and Polymorphism

Definition

*‘**Polymorphism** is a concept in type theory wherein a name may denote instances of many different classes as long as they are related by some common superclass. . . With polymorphism, an operation can be implemented differently by the classes in the hierarchy. In this manner, a subclass can extend the capabilities of its superclass or override the parent’s operation.’ (Booch, Maksimchuk, Engle, Young, Conallen and Houston 2007, p. 102)*

Inheritance and Polymorphism

Definition

‘***Polymorphism*** *is a concept in type theory wherein a name may denote instances of many different classes as long as they are related by some common superclass. . . With polymorphism, an operation can be implemented differently by the classes in the hierarchy. In this manner, a subclass can extend the capabilities of its superclass or override the parent’s operation.’ (Booch, Maksimchuk, Engle, Young, Conallen and Houston 2007, p. 102)*

‘***Polymorphism*** *is defined as: the ability of different classes to respond to the same message and each implement the method appropriately.’ (Armstrong 2006, p. 126)*

Inheritance and Polymorphism

Definition

*'**Polymorphism** is a concept in type theory wherein a name may denote instances of many different classes as long as they are related by some common superclass. . . With polymorphism, an operation can be implemented differently by the classes in the hierarchy. In this manner, a subclass can extend the capabilities of its superclass or override the parent's operation.' (Booch, Maksimchuk, Engle, Young, Conallen and Houston 2007, p. 102)*

*'**Polymorphism** is defined as: the ability of different classes to respond to the same message and each implement the method appropriately.' (Armstrong 2006, p. 126)*

*'**Polymorphism** is the mechanism we employ to customize the behavior of code we have already written.' (p. 131)*

Inheritance and Polymorphism

Remark

The previous polymorphism is called **run-time polymorphism** or **subtype polymorphism**.

Inheritance and Polymorphism

Remark

The previous polymorphism is called **run-time polymorphism** or **subtype polymorphism**.

Example (C++)

See the `oop/polymorphim.cc` file.

Namespaces

Description

In programming languages **namespaces** are context for identifiers. They help to uniquely identify the names of variables, functions, classes, etc.

Namespaces

Example (C++)

The line

```
using namespace std;
```

in `hw/hello-world.cc` opens the `std` (standard) namespace. If we remove this line, we should replace the line

```
cout << "Hello␣World!" << endl;
```

by the line

```
std::cout << "Hello␣World!" << std::endl;
```

where '`::`' is a scope qualifier.

Namespaces

Example (Java)

Namespaces in Java are handle by packages (named collection of classes).

- From the line

  ```
  import java.io.File;
  ```

  we can write `File` instead of `java.io.File`.

- From line

  ```
  import java.io.*;
  ```

  we don't need qualified names when using the classes in `java.io`.

Namespaces

Remark

‘The safest way to program is to not open up namespaces or merge them together. But, that is also inconvenient since the whole name must be written each time. What is correct for your program depends on the program being written.’ (p. 122)

Linking

Linking libraries

Libraries are common in programming languages. For using the libraries them must be linked into your program.

Linking

Static linking versus dynamic linking*

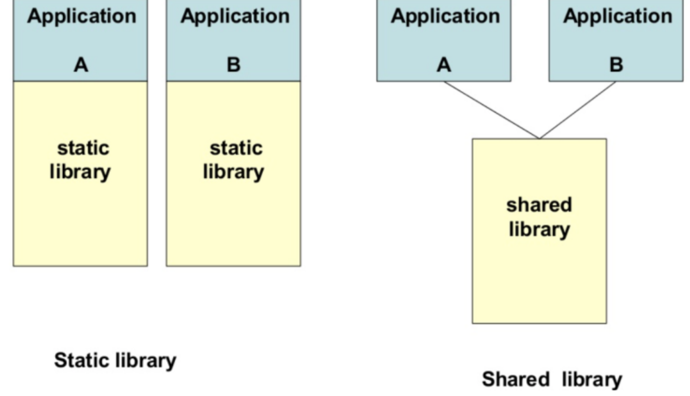


*Figure from https://achindrabhatnagar.wordpress.com/2018/09/08/static-and-dynamic-linking-c-code/.

Linking*

STATIC LINKING	DYNAMIC LINKING
Process of copying all library modules used in the program into the final executable image	Process of loading the external shared libraries into the program and then bind those shared libraries dynamically to the program
Last step of compilation	Occurs at run time
Statistically linked files are larger in size	Dynamically linked files are smaller in size
Static linking takes constant load time	Dynamic linking takes less load time
There will be no compatibility issues with static linking	There will be compatibility issues with dynamic linking

Visit www.PEDIAA.com

*Figure from https://pediaa.com/what-is-the-difference-between-static-and-dynamic-linking/.

Linking

Example

C++ uses dynamic linking by default but allows static linking.

Java only uses dynamic linking.

Linking

Example

Statically linking the `hw/hello-word.cc`:

```
$ g++ -c hello-world.cc
$ g++ --static -o hw-static hello-world.o
```

Dynamically linking the `oop/hello-word.cc`:

```
$ g++ -o hw-dynamic hello-world.cc
```

Comparing sizes:

```
$ ls -l hw-* | awk '{print $5, $9}'
17K hw-dynamic
2,4M hw-static
```

The Main Function

Example

See the files `oop/cli.cc` and `oop/CLI.java`.

I/O Streams

Reading

To read Section 4.6 "I/O Streams".

Garbage Collection

Features

- The GC removes automatically objects (variables, data structures, functions, or methods) from the heap (i.e. dynamically created) when are no longer needed.
- Trade-off between programmer control and automatically memory management.
- The GC avoids memory leaks.
- The GC impacts the run-time performance of a system.
- Languages with GC require a run-time system (i.e. virtual machine) for executing the programs.
- The GC runs in a thread.

Garbage Collection

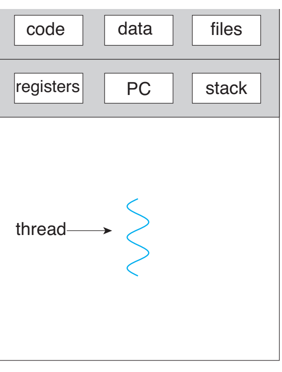
Example

Java, Haskell and Python have garbage collection. C and C++ haven't.

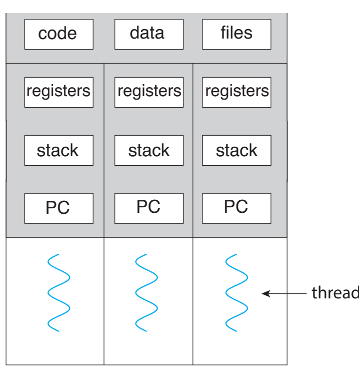
Threading

Description

'A **thread** is a basic unit of CPU utilization; it comprises a thread ID, a program counter (PC), a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.' [Silberschatz, Galvin and Gagne 2018, p. 160 and Fig. 4.1]



single-threaded process



multithreaded process

Threading

Example

A text processor with three threads.*

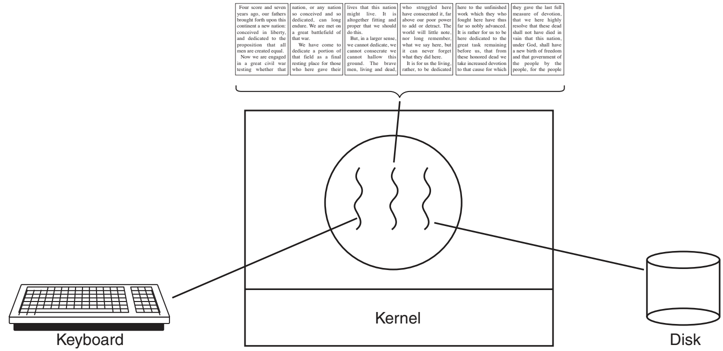


*Figure from [Tanenbaum and Bos 2014, Fig. 2.7].

Pointers and References

Description

*‘**Pointers** are the address of data in the memory of the computer. Pointers can be used in expressions to create new pointers using pointer arithmetic. In a programming language a pointer can point anywhere. A **reference** is much more controlled. References are somewhat like pointers except that they cannot be used in arithmetic expressions. They also don’t directly point to locations in memory. When a reference is dereferenced using a dot, the run-time system does the lookup in a reference table.*
This difference between references and pointers means that we can safely rely on every reference pointing to a real object where we don’t necessarily know if a pointer is pointing to space that might be safely freed or not since the pointer might be the result of some pointer arithmetic. References are safe for garbage collection. Pointers are not.’ (p. 130)

Pointers and References

Pointers and references in C++

See the `oop/pointers-and-references.cc` file.

Pointers and References

Exercise

¿What does print the following program? Why?

```
void f(int a, int& b)
{
  a = a + b;
  b = a - b;
  a = a - b;
}

void g(int* a, int b)
{
  *a = *a + b;
  b  = *a - b;
  *a = *a - b;
}
```

```
int main()
{
  int x = 1, y = 2;

  f(x,y);
  cout << "x:␣" << x;
  cout << "␣y:␣" << y << endl;

  g(&x,y);
  cout << "x:␣" << x;
  cout << "␣y:␣" << y << endl;
}
```

Interfaces and Implementations

Description

*'Each class must have two parts: an interface and an implementation... The **interface** of a class is the one place where we assert all of the assumptions that a client may make about any instances of the class; the **implementation** encapsulates details about which no client may make assumptions.' (Booch, Maksimchuk, Engle, Young, Conallen and Houston 2007, p. 51)*

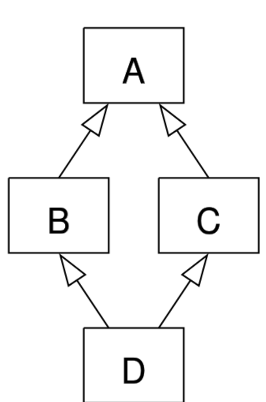
Interfaces and Implementations

Example

- Java's interfaces are a set of method declarations without implementation.
- C++ has no interfaces but we can think in the header files as the "interfaces" of the classes.

Multiple Inheritance

*‘The **diamond problem** is an ambiguity that arises when two classes `B` and `C` inherit from `A`, and class `D` inherits from both `B` and `C`. If there is a method in `A` that `B` and `C` have overridden, and `D` does not override it, then which version of the method does `D` inherit: that of `B`, or that of `C`?’*



Description and figure from Wikipedia (2023-09-05).

Multiple Inheritance

Example

- C++ supports multiple inheritance.
  See the `oop/multiple-inheritance.cc` file.
- Java does not support multiple inheritance on classes but it is supported on interfaces.
  See the `oop/java-multiple-inheritance` directory.

Function Overloading

Definition

Function overloading is a feature of some programming languages where it is possible to define multiple functions with the same name but different parameters.

Function Overloading

Definition

Function overloading is a feature of some programming languages where it is possible to define multiple functions with the same name but different parameters.

Example (C++)

See the `oop/function-overloading.cc` file.

Function Overloading

Definition

Function overloading is a feature of some programming languages where it is possible to define multiple functions with the same name but different parameters.

Example (C++)

See the `oop/function-overloading.cc` file.

Remark

> *'When a function is overloaded, each function of the same name should implement the same semantics.' (Stroustrup 2019, p. 5)*

Function Overloading

Question

Let's suppose we have defined the following overload functions in C++:

```
void fn(int, double);
void fn(double, int);
```

What happens if we write/call the function `fn(0,0)`?

C++ Templates

Definition

'A ***template*** *is a class or a function that we parameterize with a set of types or values.'*
(Stroustrup 2019, p. 79)

C++ Templates

Definition

*'A **template** is a class or a function that we parameterize with a set of types or values.' (Stroustrup 2019, p. 79)*

Use

'We use templates to represent ideas that are best understood as something general from which we can generate specific types and functions by specifying arguments.' (Stroustrup 2019, p. 79)

C++ Templates

Definition

*'A **template** is a class or a function that we parameterize with a set of types or values.'* *(Stroustrup 2019, p. 79)*

Use

'We use templates to represent ideas that are best understood as something general from which we can generate specific types and functions by specifying arguments.' *(Stroustrup 2019, p. 79)*

Example

See directory `oop/templates`.

Exception Handling

Definition

*‘**Exception handling** is the process of responding to the occurrence of exceptions—anomalous or exceptional conditions requiring special processing—during the execution of a program. In general, an exception breaks the normal flow of execution and executes a pre-registered **exception handler**.’ (Wikipedia 2023-09-07)*

Exception Handling

Exception handling using programming languages constructs

Various programming languages (e.g. C++, Java, Haskell, Python and Standard ML) have throw and try-catch statements for handling exceptions.

Exception Handling

Exception handling using programming languages constructs

Various programming languages (e.g. C++, Java, Haskell, Python and Standard ML) have throw and try-catch statements for handling exceptions.

Example (C++)

See the `oop/exceptions.cc` file.

Exception Handling

Definition

*'A **signal** is an asynchronous notification sent to a process or to a specific thread within the same process in order to notify it of an event that occurred.' (Wikipedia 2019-10-02)*

Exception Handling

Definition

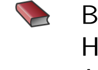
*'A **signal** is an asynchronous notification sent to a process or to a specific thread within the same process in order to notify it of an event that occurred.' (Wikipedia 2019-10-02)*

Example (signal handling in C++)

See the `oop/signal-handling.cc` file.

References

Armstrong, Deborah J. (2006). The Quarks of Object-Oriented Development. Communications of the ACM 49.2, pp. 123–128. DOI: 10.1145/1113034.1113040 (cit. on pp. 3, 4, 33–38).


Booch, Grady, Maksimchuk, Robert A., Engle, Michael W., Young, Bobbi J., Conallen, Jim and Houston, Kelli A. [1991] (2007). Object-Oriented Analysis and Design with Applications. 3rd ed. Addison-Wesley (cit. on pp. 33–38, 59).

Lee, Kent D. [2014] (2017). Foundations of Programming Languages. 2nd ed. Undergraduate Topics in Computer Science. Springer (cit. on p. 2).

Silberschatz, Abraham, Galvin, Peter Baer and Gagne, Greg [2002] (2018). Operating System Concepts. 10th ed. Wiley (cit. on p. 54).

Stroustrup, Bjarne [2013] (2019). A Tour of C++. 2nd ed. C++ In-Depth Series. Third printing. Addison-Wesley (cit. on pp. 28–32, 63–65, 67–69).

Tanenbaum, Andrew S. and Bos, Herbert [1992] (2014). Modern Operating Systems. 4th ed. Pearson (cit. on p. 55).