Informatics 3.
Lecture 1: Introduction

Kristóf Kovács

Budapest University of Technology and Economics

2024-02-13

Requirements to pass

- 2 written exams
  - April 12, May 24
  - each worth 50 points
  - individually need 40% to pass
  - grade boundaries: 40,55,70,85

Requirements to pass

- 2 written exams
  - April 12, May 24
  - each worth 50 points
  - individually need 40% to pass
  - grade boundaries: 40,55,70,85
- homework each week
  - 2 points each
  - 40% required to pass (8 points)
  - points above the requirement are added to the points of the written exam (12 points max)

Requirements to pass

- 2 written exams
  - April 12, May 24
  - each worth 50 points
  - individually need 40% to pass
  - grade boundaries: 40,55,70,85
- homework each week
  - 2 points each
  - 40% required to pass (8 points)
  - points above the requirement are added to the points of the written exam (12 points max)
- "pop quizzes"
  - at the beginning of the practicals starting from next week
  - 1 point each
  - 40% required to pass (4 points)
  - points above the requirement are added to the points of the written exam (6 points max)

Requirements to pass

- 2 written exams
    - April 12, May 24
    - each worth 50 points
    - individually need 40% to pass
    - grade boundaries: 40,55,70,85
- homework each week
    - 2 points each
    - 40% required to pass (8 points)
    - points above the requirement are added to the points of the written exam (12 points max)
- "pop quizzes"
    - at the beginning of the practicals starting from next week
    - 1 point each
    - 40% required to pass (4 points)
    - points above the requirement are added to the points of the written exam (6 points max)
- $W_1 + W_2 + \max(0, \min(H, 20) - 8) + \max(0, \min(Q, 10) - 4) + ??$

Questionnaire

Example

1. Give an algorithm that collects all even numbers from an input list of numbers.

Questionnaire

Example

1. Give an algorithm that collects all even numbers from an input list of numbers.
2. Construct a data structure that stores complex numbers.

Questionnaire

Example

1. Give an algorithm that collects all even numbers from an input list of numbers.
2. Construct a data structure that stores complex numbers.

Exercises

1. Give an algorithm that returns all numbers divisible by 7 that are smaller than 100 (and non-negative).

Questionnaire

Example

1. Give an algorithm that collects all even numbers from an input list of numbers.
2. Construct a data structure that stores complex numbers.

Exercises

1. Give an algorithm that returns all numbers divisible by 7 that are smaller than 100 (and non-negative).
2. Give an algorithm that returns all prime numbers smaller than 100.

Questionnaire

Example

1. Give an algorithm that collects all even numbers from an input list of numbers.
2. Construct a data structure that stores complex numbers.

Exercises

1. Give an algorithm that returns all numbers divisible by 7 that are smaller than 100 (and non-negative).
2. Give an algorithm that returns all prime numbers smaller than 100.
3. Give an algorithm that returns the first 100 prime numbers.

Questionnaire

Example

1. Give an algorithm that collects all even numbers from an input list of numbers.
2. Construct a data structure that stores complex numbers.

Exercises

1. Give an algorithm that returns all numbers divisible by 7 that are smaller than 100 (and non-negative).
2. Give an algorithm that returns all prime numbers smaller than 100.
3. Give an algorithm that returns the first 100 prime numbers.
4. Give an algorithm that collects all numbers divisible by 7 from an input list.

Questionnaire

Example

1. Give an algorithm that collects all even numbers from an input list of numbers.
2. Construct a data structure that stores complex numbers.

Exercises

1. Give an algorithm that returns all numbers divisible by 7 that are smaller than 100 (and non-negative).
2. Give an algorithm that returns all prime numbers smaller than 100.
3. Give an algorithm that returns the first 100 prime numbers.
4. Give an algorithm that collects all numbers divisible by 7 from an input list.
5. Give an algorithm that collects all prime numbers from an input list.

Questionnaire

Give an algorithm that...

1. receives an arbitrary number of real numbers and after each prints their average.

Questionnaire

Give an algorithm that...

1. receives an arbitrary number of real numbers and after each prints their average.
2. selects those that are relative prime from a list of (n, m) pairs.

Questionnaire

Give an algorithm that...

1. receives an arbitrary number of real numbers and after each prints their average.
2. selects those that are relative prime from a list of (n, m) pairs.
3. counts how many words contain the letter *c* (at least once) from a list of strings.

Questionnaire

Give an algorithm that...

1. receives an arbitrary number of real numbers and after each prints their average.
2. selects those that are relative prime from a list of (n, m) pairs.
3. counts how many words contain the letter *c* (at least once) from a list of strings.
4. finds the index of the letter *c* for each string in a given list, returns them as a list of (string, index) pairs.

Questionnaire

Give an algorithm that...

1. receives an arbitrary number of real numbers and after each prints their average.
2. selects those that are relative prime from a list of (n, m) pairs.
3. counts how many words contain the letter *c* (at least once) from a list of strings.
4. finds the index of the letter *c* for each string in a given list, returns them as a list of (string, index) pairs.
5. computes the product of two complex numbers (with *re* and *im* members).

Questionnaire

Give an algorithm that...

1. receives an arbitrary number of real numbers and after each prints their average.
2. selects those that are relative prime from a list of (n, m) pairs.
3. counts how many words contain the letter *c* (at least once) from a list of strings.
4. finds the index of the letter *c* for each string in a given list, returns them as a list of (string, index) pairs.
5. computes the product of two complex numbers (with *re* and *im* members).
6. computes the $*$ product of the input objects.

Questionnaire

Give an algorithm that...

1. receives an arbitrary number of real numbers and after each prints their average.
2. selects those that are relative prime from a list of (n, m) pairs.
3. counts how many words contain the letter *c* (at least once) from a list of strings.
4. finds the index of the letter *c* for each string in a given list, returns them as a list of (string, index) pairs.
5. computes the product of two complex numbers (with *re* and *im* members).
6. computes the $*$ product of the input objects.
7. receives a list of object pairs and computes the / quotient of each, while making sure to substitute the string *err* in case of an error.

Questionnaire

Construct (a) data structure(s) that...

1. can represent a square in a 2 dimensional space.

Questionnaire

Construct (a) data structure(s) that...

1. can represent a square in a 2 dimensional space.
2. can represent an arbitrary polygon in a 2 dimensional space.

Questionnaire

Construct (a) data structure(s) that...

1. can represent a square in a 2 dimensional space.
2. can represent an arbitrary polygon in a 2 dimensional space.
3. is a simple list, but it counts how many times a new element is inserted or an element is read.

Questionnaire

Construct (a) data structure(s) that...

1. can represent a square in a 2 dimensional space.
2. can represent an arbitrary polygon in a 2 dimensional space.
3. is a simple list, but it counts how many times a new element is inserted or an element is read.
4. can represent a circle, triangle, rectangle.

Questionnaire

Construct (a) data structure(s) that...

1. can represent a square in a 2 dimensional space.
2. can represent an arbitrary polygon in a 2 dimensional space.
3. is a simple list, but it counts how many times a new element is inserted or an element is read.
4. can represent a circle, triangle, rectangle.
5. can represent a circle, triangle, rectangle, each of which have their circumference/perimenter and their area methods and these can be called without knowing which object is stored in a given variable.

History

C

- created by Dennis Ritchie
- old (1972), but still used today
- strongly typed
- compiled language (not interpreted like python)
- efficient, very close to machine code
- very easy to obfuscate (not necessarily intentionally)

History

C

- created by Dennis Ritchie
- old (1972), but still used today
- strongly typed
- compiled language (not interpreted like python)
- efficient, very close to machine code
- very easy to obfuscate (not necessarily intentionally)

C++

- created by Bjarne Stroustrup
- old (1985), still widely used today
- you can think of it as an extension of C
- still being developed, latest standard is C++20 (2020)

C

We'll start with C

```
#include <stdio.h>
int main(void) {
    printf("Hello,␣World!\n");
    return 0;
}
```

C

We'll start with C

```
#include <stdio.h>
int main(void) {
    printf("Hello,␣World!\n");
    return 0;
}
```

Before we could execute this code we have to compile it.
Compilers:

- Windows: mingw, visual studio
- Linux: gcc/g++
- Mac: Xcode -> gcc/g++, cc

C

We'll start with C

```
#include <stdio.h>
int main(void) {
    printf("Hello,␣World!\n");
    return 0;
}
```

Before we could execute this code we have to compile it.
Compilers:

- Windows: mingw, visual studio
- Linux: gcc/g++
- Mac: Xcode -> gcc/g++, cc
- search on stackoverflow: *OS C compiler*

From python to C

- the type of variables have to be declared:

```c
int i = 0;
float f = 0.0;
```

From python to C

- the type of variables have to be declared:

```
int i = 0;
float f = 0.0;
```

- the entry point of a C program is the *main* function:

```
int main(void) {
```

From python to C

- the type of variables have to be declared:

```
int i = 0;
float f = 0.0;
```

- the entry point of a C program is the *main* function:

```
int main(void) {
```

- the *for* cycle doesn't iterate on a list:

```
int i;
for(i = 0; i < 10; i++) {
```

From python to C

- the type of variables have to be declared:

```
int i = 0;
float f = 0.0;
```

- the entry point of a C program is the *main* function:

```
int main(void) {
```

- the *for* cycle doesn't iterate on a list:

```
int i;
for(i = 0; i < 10; i++) {
```

- there are no complex native data structures (lists, dictionaries):

```
int t[10];
t[0] = 1;
```