Typology of Programming Languages

Exam Review

May 2025

Points repartition

- General MCQ Questions : 10pt
- Publication dates & authors : 10pt
- Argument passing : 10pt
- Open question : 10pt

Broadly, not guaranteed to be the exact same.

Who is the original inventor of Unix?

1 pt

- Bjarne Stroustrup
- Brian Kernighan
- Richard Stallman
- Ken Thompson

Who is the original inventor of Unix?

1 pt

- Bjarne Stroustrup
- Brian Kernighan
- Richard Stallman
- Ken Thompson

Answer: Ken Thompson.

Who invented the computer mouse?

1 pt

- Douglas Engelbart
- Charles Babbage
- Bill Gates
- Steve Jobs

Who invented the computer mouse?

1 pt

- Douglas Engelbart
- Charles Babbage
- Bill Gates
- Steve Jobs

Answer: Douglas Engelbart.

What does APL stand for?

1 pt

- Array Programming Language
- Array Processing Language
- A Programming Language
- Abstract Programming Language

What does APL stand for?

1 pt

- Array Programming Language
- Array Processing Language
- A Programming Language
- Abstract Programming Language

Answer: A Programming Language.

Which language was the first to have an optimizing compiler?

1 pt

- Fortran
- C
- COBOL
- Algol 60

Which language was the first to have an optimizing compiler?

1 pt

- Fortran
- C
- COBOL
- Algol 60

Answer: Fortran.

Which language introduced the concept of contract programming?

1 pt

- Ada
- Eiffel
- D
- C++

Which language introduced the concept of contract programming?

1 pt

- Ada
- Eiffel
- D
- C++

Answer: Eiffel.

Which argument passing mode corresponds to pass-by-name in Ada?

1 pt

- There is no pass-by-name in Ada.
- out
- in out
- in

Which argument passing mode corresponds to pass-by-name in Ada?

1 pt

- There is no pass-by-name in Ada.
- out
- in out
- in

Answer: no pass-by-name in Ada.

C void* pointers are an example of genericity by boxing

1 pt

True or false?

C void* pointers are an example of genericity by boxing

1 pt

True or false?

Answer: true.

C++ templates are compiled using monomorphization

1 pt

True or false?

C++ templates are compiled using monomorphization

1 pt

True or false?

Answer: true.

Which language does not monomorphize its generics?

1 pt

- C++
- Ada
- Rust
- OCaml

Which language does not monomorphize its generics?

1 pt

- C++
- Ada
- Rust
- OCaml

Answer: OCaml.

Rust macros are an example of metaprogramming

1 pt

True or false?

Rust macros are an example of metaprogramming

1 pt

True or false?

Answer: true.

Match the languages with their publication date

5 pts

- 1956
- 1958
- 1959
- 1972
- 1983
- 2009

- Ada
- ALGOL
- COBOL
- FORTRAN
- Go
- Prolog

Match the languages with their publication date

5 pts

- 1956
- 1958
- 1959
- 1972
- 1983
- 2009

- Ada
- ALGOL
- COBOL
- FORTRAN
- Go
- Prolog

Answer: FORTRAN - 1956, ALGOL - 1958, COBOL - 1959, Prolog - 1972, Ada - 1983, Go - 2009

Match the authors with their language

5 pts

- Alan Kay
- John Backus
- Niklaus Wirth
- John McCarthy
- Barbara Liskov

- Smalltalk
- FORTRAN
- Pascal
- Lisp
- CLU

Match the authors with their language

5 pts

- Alan Kay
- John Backus
- Niklaus Wirth
- John McCarthy
- Barbara Liskov

- Smalltalk
- FORTRAN
- Pascal
- Lisp
- CLU

Answer: Kay - Smalltalk, Backus - FORTRAN, Wirth - Pascal, McCarthy - Lisp, Liskov - CLU

Argument passing

10 pts

Considering the following piece of pseudo-code:

```
var t : integer
    foo : array [0..1] of integer;

procedure shoot_my(x : Mode integer);
begin
    foo[0] := 43;
    t := 0;
    x := x + 8;
end;

begin
    foo[0] := -1;
    foo[1] := 0;
    t := 1;
    shoot_my(foo[t]);
end;
```

Give the values of the variables **at the end of the execution of the program** depending on the argument passing mode used.

- Using pass-by-value:

Argument passing

10 pts

Considering the following piece of pseudo-code:

```
var t : integer
    foo : array [0..1] of integer;

procedure shoot_my(x : Mode integer);
begin
    foo[0] := 43;
    t := 0;
    x := x + 8;
end;

begin
    foo[0] := -1;
    foo[1] := 0;
    t := 1;
    shoot_my(foo[t]);
end;
```

Give the values of the variables **at the end of the execution of the program** depending on the argument passing mode used.

- Using pass-by-value:
  - `foo[0]` = 43, `foo[1]` = 0, `t` = 0

Argument passing

10 pts

Considering the following piece of pseudo-code:

```
var t : integer
    foo : array [0..1] of integer;

procedure shoot_my(x : Mode integer);
begin
    foo[0] := 43;
    t := 0;
    x := x + 8;
end;

begin
    foo[0] := -1;
    foo[1] := 0;
    t := 1;
    shoot_my(foo[t]);
end;
```

Give the values of the variables **at the end of the execution of the program** depending on the argument passing mode used.

- Using pass-by-value:
  - `foo[0]` = 43, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Algol W (the l-value in which we copy the result value is evaluated at function return):

Argument passing

10 pts

Considering the following piece of pseudo-code:

```
var t : integer
    foo : array [0..1] of integer;

procedure shoot_my(x : Mode integer);
begin
    foo[0] := 43;
    t := 0;
    x := x + 8;
end;

begin
    foo[0] := -1;
    foo[1] := 0;
    t := 1;
    shoot_my(foo[t]);
end;
```

Give the values of the variables **at the end of the execution of the program** depending on the argument passing mode used.

- Using pass-by-value:
  - `foo[0]` = 43, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Algol W (the l-value in which we copy the result value is evaluated at function return):
  - `foo[0]` = 8, `foo[1]` = 0, `t` = 0

Argument passing

10 pts

Considering the following piece of pseudo-code:

```
var t : integer
    foo : array [0..1] of integer;

procedure shoot_my(x : Mode integer);
begin
    foo[0] := 43;
    t := 0;
    x := x + 8;
end;

begin
    foo[0] := -1;
    foo[1] := 0;
    t := 1;
    shoot_my(foo[t]);
end;
```

Give the values of the variables **at the end of the execution of the program** depending on the argument passing mode used.

- Using pass-by-value:
  - `foo[0]` = 43, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Algol W (the l-value in which we copy the result value is evaluated at function return):
  - `foo[0]` = 8, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Ada (the l-value in which we copy the result value is evaluated at function call):

Argument passing

10 pts

Considering the following piece of pseudo-code:

```
var t : integer
    foo : array [0..1] of integer;

procedure shoot_my(x : Mode integer);
begin
    foo[0] := 43;
    t := 0;
    x := x + 8;
end;

begin
    foo[0] := -1;
    foo[1] := 0;
    t := 1;
    shoot_my(foo[t]);
end;
```

Give the values of the variables **at the end of the execution of the program** depending on the argument passing mode used.

- Using pass-by-value:
  - `foo[0]` = 43, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Algol W (the l-value in which we copy the result value is evaluated at function return):
  - `foo[0]` = 8, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Ada (the l-value in which we copy the result value is evaluated at function call):
  - `foo[0]` = 43, `foo[1]` = 8, `t` = 0

Argument passing

10 pts

Considering the following piece of pseudo-code:

```
var t : integer
    foo : array [0..1] of integer;

procedure shoot_my(x : Mode integer);
begin
    foo[0] := 43;
    t := 0;
    x := x + 8;
end;

begin
    foo[0] := -1;
    foo[1] := 0;
    t := 1;
    shoot_my(foo[t]);
end;
```

Give the values of the variables **at the end of the execution of the program** depending on the argument passing mode used.

- Using pass-by-value:
  - `foo[0]` = 43, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Algol W (the l-value in which we copy the result value is evaluated at function return):
  - `foo[0]` = 8, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Ada (the l-value in which we copy the result value is evaluated at function call):
  - `foo[0]` = 43, `foo[1]` = 8, `t` = 0
- Using pass-by-reference:

Argument passing

10 pts

Considering the following piece of pseudo-code:

```
var t : integer
    foo : array [0..1] of integer;

procedure shoot_my(x : Mode integer);
begin
    foo[0] := 43;
    t := 0;
    x := x + 8;
end;

begin
    foo[0] := -1;
    foo[1] := 0;
    t := 1;
    shoot_my(foo[t]);
end;
```

Give the values of the variables **at the end of the execution of the program** depending on the argument passing mode used.

- Using pass-by-value:
  - `foo[0]` = 43, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Algol W (the l-value in which we copy the result value is evaluated at function return):
  - `foo[0]` = 8, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Ada (the l-value in which we copy the result value is evaluated at function call):
  - `foo[0]` = 43, `foo[1]` = 8, `t` = 0
- Using pass-by-reference:
  - `foo[0]` = 43, `foo[1]` = 8, `t` = 0

Argument passing

10 pts

Considering the following piece of pseudo-code:

```
var t : integer
    foo : array [0..1] of integer;

procedure shoot_my(x : Mode integer);
begin
    foo[0] := 43;
    t := 0;
    x := x + 8;
end;

begin
    foo[0] := -1;
    foo[1] := 0;
    t := 1;
    shoot_my(foo[t]);
end;
```

Give the values of the variables **at the end of the execution of the program** depending on the argument passing mode used.

- Using pass-by-value:
  - `foo[0]` = 43, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Algol W (the l-value in which we copy the result value is evaluated at function return):
  - `foo[0]` = 8, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Ada (the l-value in which we copy the result value is evaluated at function call):
  - `foo[0]` = 43, `foo[1]` = 8, `t` = 0
- Using pass-by-reference:
  - `foo[0]` = 43, `foo[1]` = 8, `t` = 0
- Using pass-by-name:

Argument passing

10 pts

Considering the following piece of pseudo-code:

```
var t : integer
    foo : array [0..1] of integer;

procedure shoot_my(x : Mode integer);
begin
    foo[0] := 43;
    t := 0;
    x := x + 8;
end;

begin
    foo[0] := -1;
    foo[1] := 0;
    t := 1;
    shoot_my(foo[t]);
end;
```

Give the values of the variables **at the end of the execution of the program** depending on the argument passing mode used.

- Using pass-by-value:
  - `foo[0]` = 43, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Algol W (the l-value in which we copy the result value is evaluated at function return):
  - `foo[0]` = 8, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Ada (the l-value in which we copy the result value is evaluated at function call):
  - `foo[0]` = 43, `foo[1]` = 8, `t` = 0
- Using pass-by-reference:
  - `foo[0]` = 43, `foo[1]` = 8, `t` = 0
- Using pass-by-name:
  - `foo[0]` = 51, `foo[1]` = 0, `t` = 0

Argument passing

10 pts

Considering the following piece of pseudo-code:

```
var t : integer
    foo : array [0..1] of integer;

procedure shoot_my(x : Mode integer);
begin
    foo[0] := 43;
    t := 0;
    x := x + 8;
end;

begin
    foo[0] := -1;
    foo[1] := 0;
    t := 1;
    shoot_my(foo[t]);
end;
```

Give the values of the variables **at the end of the execution of the program** depending on the argument passing mode used.

- Using pass-by-value:
  - `foo[0]` = 43, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Algol W (the l-value in which we copy the result value is evaluated at function return):
  - `foo[0]` = 8, `foo[1]` = 0, `t` = 0
- Using pass-by-value-result, à la Ada (the l-value in which we copy the result value is evaluated at function call):
  - `foo[0]` = 43, `foo[1]` = 8, `t` = 0
- Using pass-by-reference:
  - `foo[0]` = 43, `foo[1]` = 8, `t` = 0
- Using pass-by-name:
  - `foo[0]` = 51, `foo[1]` = 0, `t` = 0

Open question

10 pts

We imagine to be implementing a text editor in either **Rust** or **Python**.

Present both language in a few words, compare their broad characteristics, and discuss the pros and cons of using either in the context of the software we wish to implement.