Final Examination
Concepts of Programming Languages
2023-2024

January 9, 2023, 9:00—11:30 (extra time: 12:00)

You are not allowed to make use of digital devices, books, slides, self-written notes, etc.

There are 10 questions: 5 multiple-choice questions and 5 open questions. Some questions have a small number of sub-questions. Each question is worth 1 point. The minimal number of points for a question is 0 points (i.e. a question never gives negative points). The grade is computed as $\frac{points}{10} \times 9 + 1$.

Hint: do not spend too much time on a single question ($\sim$15 min each).

Question 1. Given the following grammar where $\langle \text{expr} \rangle$ is the starting symbol.

$$\begin{aligned}
\langle \text{expr} \rangle &::= \langle \text{var} \rangle \mid (\langle \text{expr} \rangle + \langle \text{expr} \rangle) \mid (\langle \text{expr} \rangle \times \langle \text{expr} \rangle) \\
\langle \text{var} \rangle &::= \langle \text{char} \rangle \langle \text{var'} \rangle \\
\langle \text{var'} \rangle &::= \langle \text{char} \rangle \langle \text{var'} \rangle \mid \langle \text{num} \rangle \langle \text{var'} \rangle \mid \epsilon \\
\langle \text{char} \rangle &::= a \mid b \mid c \mid \cdots \mid z \\
\langle \text{num} \rangle &::= 0 \mid 1 \mid 2 \mid \cdots \mid 9
\end{aligned}$$

Which of the following statements are true?
(Zero or more answers)

A. This grammar is ambiguous.

B. The language generated by this grammar only contains a finite amount of strings.

C. The string "$(a + b) \times a$" is in the language generated by this grammar.

D. The string "$((var1 + var2) \times (var3 \times var4))$" is in the language generated by this grammar.

Question 2. Explain operator overloading using an example.
(Open question)

Question 3. Consider the program below. We assume call-by-value parameter passing.

```
 1. int arr[1000];
 2.
 3. int fib(int n) {
 4.     if(n == 1 || n == 2) {
 5.         return n-1;
 6.     }
 7.     if (arr[n] != -1) {
 8.         return arr[n];
 9.     }
10.     arr[n] = fib(n-1) + fib(n-2);
11.     return arr[n];
12. }
13.
14. void main() {
15.     for (int i = 0; i < 1000; i++) {
16.         arr[i] = -1;
17.     }
18.     fib(10);
19. }
```

Which of the following statements is true?
(Zero or more answers)

A. The procedure `fib` is a pure function with function signature **int** $\rightarrow$ **int**.

B. For values $a \leq 1000$, the maximum size of the call stack needed by `fib(a)` is approximately proportional to the value of a.

C. For some values $a \leq 1000$, when calling `fib(a)` twice in a row for the same a, `fib` will behave differently the second call.

D. The procedure `fib` has no side effects.

Question 4. Consider the pseudo-code below.

```
1. int y = 5
2. procedure P(int x) {
3.     x = x + 5
4.     print(x + y)
5. }
6. P(y)
7. P(y)
```

1. What does the program print if we assume call-by-value?
2. What does the program print if we assume call-by-name? (Assume `y` is always treated as a global variable).
3. For call-by-name, does the behavior change if we assume `y` is treated as a local variable in procedure `P` (i.e. its value gets pushed to the stack and restored when `P` is finished)? If so, what is the output in this case?

Question 5. Consider the following two classes.

```
class Square {
    int h, w;
    Square(int x) {
        this.h = x;
        this.w = x;
    }
    int getHeight() {return this.h;}
    int getWidth() {return this.w;}
    ⋮
}
```

```
class Rectangle {
    int h, w;
    Rectangle(int x, int y) {
        this.h = x;
        this.w = y;
    }
    int getHeight() {return this.h;}
    int getWidth() {return this.w;}
    ⋮
}
```

1. According to the Liskov substitution principle, which of the following statements is correct?
   (One answer)

   A. Rectangle can be a subtype of Square.

   B. Square can be a subtype of Rectangle.

   C. Either direction of subtyping is allowed.

   D. Neither can be a subtype of each other, they should instead both be subtypes of an abstract class Shape.

2. Explain your reasoning.

Question 6. Consider the following program.

```
 1. int *y = null
 2. void foo() {
 3.     int *x = new(5);
 4.     y = x;
 5.     print(*x);
 6.     y = new(10);
 7.     x = 42;
 8.     print(*x);
 9.     delete(y);
10. }
```

1. What does this program print when the function `foo()` is called?

2. Give an example of a memory leak in this program.

3. Give an example of a dangling pointer in this program.

Question 7. What statements about exceptions are true?
(Zero or more answers)

A. An exception which is thrown but never caught causes the program to crash.

B. After an exception is caught, the program can always resume its normal execution.

C. Exceptions are more useful than completely crashing the program only because exceptions can contain specific information about what caused the initial error.

D. Numerical errors such as division by zero must always raise an exception or cause the program to crash.

Question 8. Given the following grammar for the lambda calculus:

$$\langle \text{expr} \rangle ::= \langle \text{var} \rangle \mid \lambda \langle \text{var} \rangle \ \langle \text{expr} \rangle \mid \langle \text{expr} \rangle \ \langle \text{expr} \rangle \mid (\langle \text{expr} \rangle)$$
$$\langle \text{var} \rangle ::= a \mid b \mid c \mid \cdots \mid z$$

Ambiguity is resolved in the standard way: application associates to the left, and abstraction precedes application (that is, abstraction binds more strongly than application). Abstractions or applications inside matching parentheses precede those outside.

Draw the abstract syntax tree of the following expressions.

1. $\lambda x(\lambda y\ y)$
2. $\lambda x(\lambda y\ x)\ y$
3. $(\lambda y(x\ y))(\lambda x\ x)$

Question 9. For the following lambda expressions, underline all free variables.

1. $x\ (\lambda x\ x)(\lambda x\ x)\ x$
2. $((\lambda x(\lambda y(\lambda z\ x)))\ y)$
3. $\lambda x((\lambda x\ x)\ x\ (\lambda z\ x))\ z$

Question 10. Consider the simply typed lambda calculus, with its typing rules shown below:

$$\frac{}{\Gamma, x : T, \Gamma' \vdash x : T}\ (1) \quad \text{where } x \text{ does not occur in } \Gamma'$$

$$\frac{\Gamma \vdash E_1 : (T \to T') \quad \Gamma \vdash E_2 : T}{\Gamma \vdash (E_1\ E_2) : T'}\ (2) \qquad \frac{\Gamma, x : T \vdash E : T'}{\Gamma \vdash (\lambda x^T E) : (T \to T')}\ (3)$$

Which of the following judgments can be derived in this type system? I.e. which of the following expressions are correctly typed?
(Zero or more answers)

A. $\vdash \lambda x^{\alpha}\ (x\ x) : \alpha \to \alpha$

B. $x : \alpha, f : (\alpha \to \beta) \vdash (f\ x) : \beta$

C. $\vdash \lambda x^{\alpha}((\lambda y^{\beta} y)\ x) : \alpha \to \beta$

D. $\vdash \lambda y^{(\alpha \to \alpha)} y : (\alpha \to \alpha) \to (\alpha \to \alpha)$