# 440 Midterm 2012 fall

1

Given the following two classes:

```
public class A {
        void doSomething () {
          System.out.println("A");
        }
}
public class B extends A{
        void doSomething () {
        System.out.println("B”);
}
```

The method doSomething() is

1. Overloaded;
2. Overridden.

Answer: overidden

1. For the expression "1 + 3.7 " in java, the evaluation mechanism of the expression is
   (a) Overloading;
   (b) Overridden;
   (c) Generics;
   (d) Coercion

Answer: d

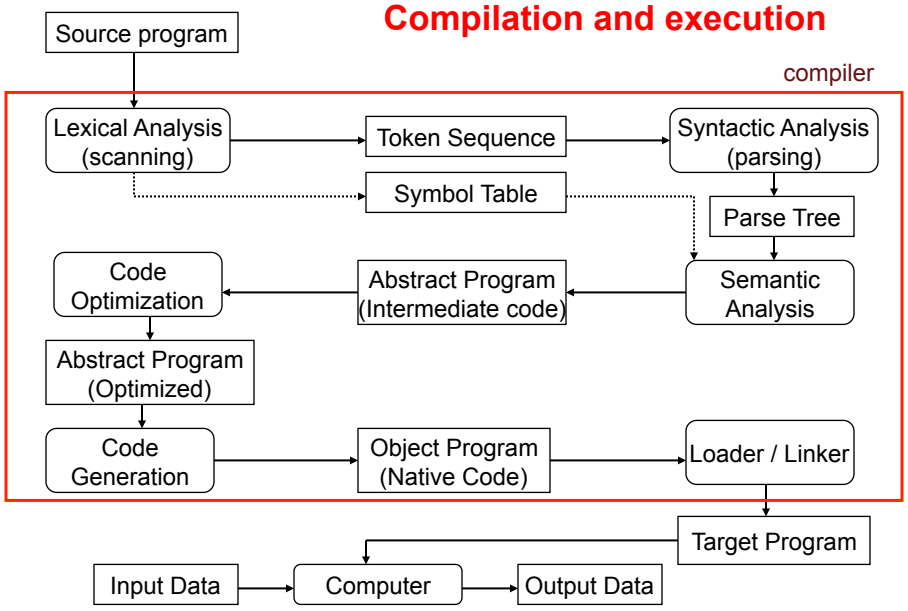
3

- Which of the following is not part of a compiler?
  - Scanner;
  - Parser;
  - Code generator;
  - Optimizer;
  - None of the above.

Answer: none of the above

4

## Compilation and execution




5

1. Given a Prolog rule A:-B,C. It is equivalent to

   (a) $\neg B \vee C \vee A$

   (b) $B \vee C \vee \neg A$

   (c) $B \vee C \vee A$

   (d) $B \wedge C \wedge \neg A$

   (e) $\neg B \wedge \neg C \wedge A$

   (f) None of the above

Answer: None of the above

6

1. Which of the following is not a valid lambda expression?

   (a) x

   (b) xx

   (c) λx.x

   (d) λx

- Answer: d
- Syntax for lambda expression
  - Variable
  - Application
  - Abstraction

7

8. What is the result of running (+ (* 1 2) (* 3 4)) in Scheme language?

(a) 14

(b) 10

(c) 11

(d) 24

(e) none of the above

8

1. Given the lambda expression **λ x . x y.** Derive the expression as far as possible using $\alpha$, β, and $\eta$-conversions. The final result will be:

   (a) x

   (b) y

   (c) **λ x . x y**

   (d) **λ x . x**

- Answer: c
- Notice that eta conversion is not applicable here
- It is λx.fx ➔ f

9

1. Given the expression **(λ x y . x y) (λ x . x y) (λ a b . a b)** in lambda calculus. Derive the expression as far as possible using $\alpha$, β, and $\eta$-conversions. The final result will be:

   (a) a

   (b) b

   (c) ab

   (d) λ b.y b

   (e) λx.x

   (f) none of the above

- Answer: f

10

- (2 points) Write the result of running the following program:
- (reduce concat '( ) '((1 2 3) (4 5) (6 7 8)))

11

1. What is the result of running **(map (lambda (x) 1) '(1 2 3) )** ?

   (a) 1

   (b) a list (1)

   (c) error

   (d) a list (1 1 1)

   (e) a list (1 2 3)

- Answer: d

12

1. What is the result of running **(reduce * 1 (map (lambda (x) (+ 2 x)) '(1 2 3)))** ?
   (a) 60
   (b) ( ) (the empty list)
   (c) (3 4 5)
   (d) (2 4 6)
   (e) 12
   (f) none of above

- Answer: a

13

1. Which of the following is the earliest OO programming language?
   (a) Java
   (b) C++
   (c) Smalltalk
   (d) Perl
   (e) Fortran
   (f) C

- Answer: c

14

1. Which of the following language is the **least** declarative?
   (a) Java
   (b) Prolog
   (c) SQL
   (d) XSLT
   (e) Scheme

- Answer: a

15

1. Church-Rosser theorem is an important theorem in
   (a) Hoare logic
   (b) Lambda Calculus
   (c) BNF
   (d) Logic programming
   (e) Horn Clause

- Answer: b

16

1. Which of the following formal system ***best*** describes the semantic foundation for Prolog?
   (a) BNF
   (b) Horn clause;
   (c) Predicate calculus;
   (d) Lambda calculus;
   (e) Hoare logic;
   (f) Propositional logic

- Answer:b

17

1. Map-reduce programming framework is based on
   (a) Logic programming
   (b) Functional programming
   (c) Database programming
   (d) Map programming
   (e) Google Map

- Answer: b

18

1. Map-reduce programming is best used for
   (a) Web page development
   (b) Logic programming
   (c) Object persistence
   (d) Parallel computing
   (e) Google map apps

- Answer: d

19

1. Hadoop is a Java implementation of
   (a) A relational database system
   (b) Map-reduce programming
   (c) Logic programming
   (d) Functional programming

- Answer: b

20

## True or false

- There is more than one way to reduce a lambda expression.
- One programming language can support more than one programming paradigm.
- One programming paradigm can be supported by more than one language.
- Abstract Data Type defines a data type using a set of values of the type.

21

## Data Type and Abstract Data Type

ADT

- Data type
  - Data values
  - Operations on the data
- Abstract
  - Focus on some details while ignore others.
    - Simplified description of objects
    - Emphasis significant information only
    - Suppress irrelevant information
- Abstract Data Type
  - Focus on operations, ignore the concrete data representation.

22

## ADT Example

ADT

Stack();
Stack push(Object item);
Stack pop();
Object peek();
boolean empty();

for all [s: Stack; i: Object]

Stack().empty() = true
s.push(i).empty()=false

Stack().pop()=error
s.push(i).pop()=s;

Stack().peek()=error
s.push(i).peek()=i;

(Note that this is not the Stack in Java)

- ADT defines a data type in terms of operations instead of the data
- The implementation of Stack data type can be an Array, or a Vector, or other data structure
- This is the idea of information hiding
  - Implementation is not relevant to the user of the data type
  - Implementation can be changed without affecting other parts of the system.
- The meaning of ADT can be specified in terms of the relationships between the operators, independent of the data representation.

23

  - Dynamic binding is the binding of the method name to the method definition dynamically at compilation time.

- answer: false.
- Dynamic binding happens at run time
- Recall the example

```
class Cat extends Animal {
   void talk() {
      System.out.println("Meow.");
   }
}

class Interrogator {
   static void makeItTalk(Animal subject) {
      subject.talk();
   }
}
```

24

- Given the following code fragment:

```
Vector v = new Vector();
v.add("test");
String a = (String) v.get(0);
Integer b = (Integer) v.get(0);
```

  – Whether the code will pass the compilation correctly?

- Answer: it will compile correctly.
- But it will have runtime error
- Type cast error
- Violate the strong typing rule

25

- (1 point) Rewrite the λ expression λx.xz λy.xy by adding parentheses around sub-expressions so that there is no ambiguity. For instance, λx.xy should be rewritten as λx.(xy)
  – λx.((xz) (λy.(xy)))
- (1 point) Find and circle all free variables in the following λ expression

  (λx. x z) λy. w λw. w y z x

  (λx. x z) (λy. (w (λw. (((w y) z) x))))
-

26

- (4 points) Prove **not (not true) = true** using λ calculus, supposing the encoding of **not, true**, and **false** are defined using λ expressions as below:
  - **not = λx.((x false) true)**
  - **true = λx.λy.x**
  - **false = λx.λy.y**

N=λx.xFT  T=λxy.x

F=λxy.y

**N(NT)➔N(TFT)➔N(F)➔FFT➔T**

**➔(TFT)FT➔FFT➔T**

**N(NT)➔(NT)FT➔(TFT)FT➔FFT➔T**

not (not true)

= λx.((x false) true) (not true)

= ((not true) false) true

= ((λx.((x false) true) true) false) true

= (((true false) true) false) true

= ((((λx.λy.x) false) true) false) true

= (((λy.false) true) false) true

= ((false) false) true

= ((λx.λy.y) false) true

= (λy.y) true

27

- (2 points) Prove **if false then x else y = y** using λ calculus, supposing **if a then b else c = a b c, true** and **false** are defined the same as in the previous question.

F=λxy.y

Fxy➔y

- (2 points) Write the result of running the following program:

(reduce concat '( ) '((1 2 3) (4 5) (6 7 8)))

28

```
a(X,X):-b(X,X).
a(X,Z):-b(X,Y), a(Y,Z).
b(a, b).   b(b, c).   b(b, d).   b(a, d)
| ?-a(a, X).
```

Answer: no.

29

```
a(X, [ ], [X]).
a(X, [Y|L], [X, Y|L]) :- X =< Y.
a(X, [Y|L], [Y|IL]) :-
        X > Y, a(X, L, IL).
| ?-a(2, [2, 3, 4], X)
```

Answer: [2,2,3,4]
It is an insertion.

30

```
static  max(S x, T y){
   return (x.compareTo(y) < 0 )? y:x;
}
```

Answer:

```
static <T extends Comparable<T> > T max(T x, T y){
 return (x.compareTo(y) < 0 )?y:x;
}
```

31

```
<source> e1
<employee>
   <firstName>Joe</firstName>
   <surName>Smith</surName>
</employee>
<employee>
   <firstName>Andrew</firstName>
   <surName>Wang</surName>
   <supervisor>
            <employee> e2
               <firstName>Steve</firstName>
               <surName>Miller</surName>
           </employee>
    </supervisor>
 </employee>
</source>
```

32

```
<xsl:stylesheet   xmlns:xsl="http://www.w3.org/
   1999/XSL/Transform" version="1.0">
<xsl:template match="employee/supervisor/
   employee">
   first name is <xsl:value-of select="firstName"/
   >
</xsl:template>
</xsl:stylesheet>
```

e1

Joe

Smith

Andrew

Wang

first name is Steve

33

```
<xsl:stylesheet   xmlns:xsl="http://
   www.w3.org/1999/XSL/Transform"
   version="1.0">
 <xsl:template match="employee">
   <xsl:apply-templates
   select="supervisor/employee/*"/>
 </xsl:template>

 <xsl:template match="surName">
  HI
 </xsl:template>
</xsl:stylesheet>
```

e1

Steve

HI

34

```
<xsl:stylesheet   xmlns:xsl="http://
   www.w3.org/1999/XSL/
   Transform" version="1.0">

<xsl:template match="employee">

Hello, Found you!

</xsl:template>

</xsl:stylesheet>
```

e1

Hello, Found you!

Hello, Found you!

35