Faculty of Computers and Artificial Intelligence
Cairo University

Final Exam

Program:	**Computer Science / Software Engineering**	**Date:**	**16/3/2021**
Course Name:	**Concepts of Programming Languages**	**Duration:**	**2 hours**
Course Code:	**CS317 / SCS317**	**Total Marks:**	**60 marks**
Instructor(s):	**Dr. Amin Allam**		

تعليمات هامة:

• حيازة التليفون المحمول مفتوحا داخل لجنة الأمتحان يعتبر حالة غش تستوجب العقاب وإذا كان ضرورى الدخول بالمحمول فيوضع مغلقا فى الحقائب .
• لا يسمح بدخول سماعة الأذن أو البلوتوث .
• لايسمح بدخول اي كتب أو ملازم أو أوراق داخل اللجنة والمخالفة تعتبر حالة غش .

- **Exam consists of 60 multiple-choice questions in 7 pages. Each question weights 1 mark.**
- **Record in the bubble sheet exactly ONE answer for each question .**

Qa ⇒ For questions 1 to 7, consider the following BNF grammar:

```
<expr> -> <term> % <expr> | <term>
<term> -> <term> # <var> | <var>
<var> -> x | y | z
```

1 The above BNF grammar is:
A unambiguous B left associative C right associative D orthogonal E semantics

2 The % operator in the above BNF grammar is:
A unambiguous B left associative C right associative D orthogonal E semantics

3 The % operator in x#y%z according to the above BNF grammar is applied to:
A y and z B x and y C x and z D x#y and z E cannot be determined

4 The # operator in x#y%z according to the above BNF grammar is applied to:
A y and z B x and y C x and z D x and y%z E cannot be determined

5 In the above BNF grammar, y is:
A right associative B terminal C nonterminal D metasymbol E low precedence

6 The highest precedence operator in the above BNF grammar is:
A -> B % C # D | E several operators have equal highest precedence

7 The second line of the above BNF grammar can be replaced in an EBNF grammar with <term> -> <var> { # <var> } which is equivalent except for:
A orthogonality B precedence C associativity D efficiency E terminality

Qb 8 In an EBNF grammar having the rule <expr> -> <term>{(+|-)[*](-|*)<term>}, <expr> can be expanded to:
A <term>**<term> B <term>-+<term> C <term>--<term>
D <term>++<term> E <term>+<term>*<term>

9 In an attribute grammar, the syntax rule <expr>[1] -> <term>%<expr>[2] and the attribute computation function <expr>[1].type <- <term>.type mean that:

A % operator can be applied to all types
B types of LHS and RHS of % operator must match
C there is an array of two expressions
D type of LHS expression is assigned to type of RHS term
E all expression types in the program are the same

10 In an attribute grammar, the syntax rule <expr>[1] -><term>%<expr>[2] and the predicate <expr>[2].type==<term>.type mean that:

A % operator can be applied to all types
B types of LHS and RHS of % operator must match
C there is an array of two expressions
D type of LHS expression is assigned to type of RHS term
E all expression types in the program are the same

Qc ⇒ For questions 11 to 19, consider the following C++ program:

```
1 int t=8;
2 void F(int& r){r=5;}
3 int main()
4 {
5     int a=5, b=6; float c=1.2, d=2.9;
6     int x=a+t; float y=c+d; float z=a+c;
7     int* s=new int; F(*s); delete s;
8     return 0;
9 }
```

11 The following variable is static:

A r B t C x D s E no static variable exists

12 The following line is related to orthogonality:

A 2 B 5 C 6 D 7 E no such line exists

13 The following line is related to dynamic type binding:

A 1 B 2 C 6 D 7 E no such line exists

14 The following variable is heap-dynamic:

A r B t C s D the unnamed variable created by new E no such variable exists

15 The following variable is an alias to another variable:

A r B t C y D the unnamed variable created by new E no such variable exists

16 Line 2 checks for types at:

A compile time B load time C run time D exception time E no check

17 The following line contains coersion:

A 1 B 2 C 6 D 7 E no such line exists

18 The following line causes a side effect:

A 1 B 2 C 5 D 6 E no such line exists

19 The following line contains static value binding:

A 1 B 2 C 5 D 1 and 5 E none of the previous choices

Qd

20 One keyword with two different meanings mainly reduces:

A readability B writability C reliability D efficiency E generality

21 A compiler compiles a statement inside a loop:

A once B twice C number of times equal to number of loop iterations

D same as interpreter E zero times

22 Abstraction mainly improves:

A readability B writability C reliability D efficiency E portability

23 A static variable differs from other variables because it must:

A bind to storage at load time B bind to values at load time C bind to type at compile time

D bind to values at run time E bind to values at compile time

24 If a language supports short-circuit evaluation for `&&` (logical AND), consider `(g && false)`:

A g is evaluated B g is not evaluated C g may be evaluated D error E exception

25 If a language supports short-circuit evaluation for `&&` (logical AND), consider `(true && g)`:

A g is evaluated B g is not evaluated C g may be evaluated D error E exception

26 Type checking achieves its maximum reliability if it is done:

A at compile time B at load time C at run time D when the related function is called

E immediately before an error occurs

27 An `int*` variable and an `int**` variable are:

A compatible for name type equivalence rules B compatible for structure type equivalence rules

C all previous choices D never compatible E implicitly converted for name type equivalence

Qe ⇒ For questions 28 to 30, consider the following C++ program and assume static scoping:

```
1 int t,y,z,r;
2 void main()
3 {
4     int t,y,z;
5     while(a<10)
6     {
7         int t,y,w;
8         if(t<5) {int t;}
9     }
10 }
```

28 Assuming static scoping, the referencing environment of Line 8 does not contain:
A t of line 4 B y of line 7 C r D w E none of the previous choices

29 Assuming static scoping, the referencing environment of Line 5 consists exactly of:
A t,y,z,r of line 1 B t,y,z of line 4 C t,y,z of line 4 and r D r E t,y,z of line 1

30 Assuming static scoping, the variable z of line 4 with respect to the block from line 6 to line 9 is:
A local B nonlocal C global D invisible E not related

Qf ⇒ For questions 31 to 33, consider the following program and assume dynamic scoping:

```
0 void Sub2()
1 {
2    int w, x;
3    // Line 3
4 }
5 void Sub1()
6 {
7    int x, y;
8    // Line 8
9    Sub2();
10 }
11 void main()
12 {
13    int y, z;
14    // Line 14
15    Sub1();
16 }
```

31 Assuming dynamic scoping, the referencing environment of Line 14 contains:
A w of line 2 B x of line 2 C x of line 7 D y of line 7 E y of line 13

32 Assuming dynamic scoping, the referencing environment of Line 8 contains:
A w of line 2 B x of line 2 C y of line 13 D z of line 13 E none of the previous choices

33 Assuming dynamic scoping, the referencing environment of Line 3 does not contain:
A w of line 2 B x of line 2 C y of line 7 D y of line 13 E z of line 13

Qg ⇒ For questions 34 to 41, consider the following program:

```
1 void Fun(in int a, out int b, in-out int c)
2 {
3      // Line 3
4      a=7; b=8; c=9;
5      // Line 5
6      a=a; b=b; c=c;
7 }
8
9 void main()
10 {
11     int x=2, y=3, z=4;
12     Fun(x, y, z);
13     // Line 13
14 }
```

34 When execution reaches line 3, the value of a is:
A 0   B 2   C 7   D undefined   E none of the previous choices

35 When execution reaches line 3, the value of b is:
A 0   B 3   C 8   D undefined   E none of the previous choices

36 When execution reaches line 3, the value of c is:
A 0   B 4   C 9   D undefined   E none of the previous choices

37 When execution reaches line 5, assuming pass by value-result, the value of z is:
A 0   B 4   C 9   D undefined   E none of the previous choices

38 When execution reaches line 5, assuming pass by reference, the value of z is:
A 0   B 4   C 9   D undefined   E none of the previous choices

39 When execution reaches line 13, the value of x is:
A 0   B 2   C 7   D undefined   E none of the previous choices

40 When execution reaches line 13, the value of y is:
A 0   B 3   C 8   D undefined   E none of the previous choices

41 When execution reaches line 13, the value of z is:
A 0   B 4   C 9   D undefined   E none of the previous choices

Qh ⇒ For questions 42 to 49, consider the following **Prolog** declarative program:

```
female(shelly). female(mary). female(ann).
male(bill). male(jake). male(tom).
father(bill, jake). father(bill, shelly). father(jake, ann).
mother(mary, jake). mother(mary, shelly). mother(shelly, tom).
parent(X, Y) :- father(X, Y).
parent(X, Y) :- mother(X, Y).
gpm(X, Z) :- parent(X, Y), parent(Y, Z), male(Z).
```

42 The output of the query: `female(tom)` is:
A `yes` B `no` C `female` D `male` E none of the previous choices

43 The output of the query: `male(alex)` is:
A `yes` B `no` C `female` D `male` E none of the previous choices

44 The output of the query: `jake` is:
A `yes` B `no` C `female` D `male` E none of the previous choices

45 The output of the query: `parent(bill, Z)` is:
A `Z=bill` B `Z=jake` C `Z=shelly` D `Z=jake` and `Z=shelly`
E none of the previous choices

46 The output of the query: `parent(Z, shelly)` is:
A `Z=mary` B `Z=bill` C `Z=bill` and `Z=mary` D `no`
E none of the previous choices

47 The output of the query: `gpm(X, Z)` is:
A `X=bill, Z=tom` B `X=bill, Z=tom` and `X=bill, Z=ann`
C `X=bill, Z=tom` and `X=mary, Z=tom` D `no` E none of the previous choices

48 The output of the query: `gpm(X, Z), female(X)` is:
A `X=bill, Z=tom` B `X=mary, Z=tom`
C `X=mary, Z=tom` and `X=mary, Z=ann` D `no` E none of the previous choices

49 The output of the query: `gpm(X, Z), female(Z)` is:
A `X=bill, Z=ann` B `X=mary, Z=ann`
C `X=bill, Z=ann` and `X=mary, Z=ann` D `no` E none of the previous choices

Qi

50 The following item does **not** belong to activation records:
A return variable B local variables C global variables D parameter variables
E none of the previous choices

51 Compiler knows where to resume control after a function call terminates by storing the location in:
A local variable B global variable C automatic variable D parameter variable
E activation record instance

52 Recursion can be simulated using a:
A stack B queue C priority queue D tree E none of the previous choices

53 A variable x local to a recursive function F () may have **at most** the following number of different copies stored in the run-time stack at the same time:
A 0 B 1 C 2 D 3 E none of the previous choices

54 If several subprograms execute simultaneously on different processors, this is called:
A physical concurrency B logical concurrency C generic programming
D cooperation synchronization E competition synchronization

55 If task A must wait for task B to complete some activity before task A continues, this is called:
A physical concurrency B logical concurrency C generic programming
D cooperation synchronization E competition synchronization

56 Monitors and semaphores differ in:
A synchronization time B the problems they target C the number of deadlocks
D location of synchronization responsibilities E none of the previous choices

57 The following technique has the **minimum** type-safety:
A variable parameters B function pointers C inheritance D static polymorphism
E dynamic polymorphism

58 Creating objects of the following classes is semantically meaningless:
A abstract classes B concrete classes C solid classes D static classes E dynamic classes

59 The following technique attempts to achieve generality by code generation:
A static inheritance B dynamic inheritance C static polymorphism D dynamic polymorphism
E templates

60 The decorator design pattern can simulate:
A static inheritance B dynamic inheritance C static polymorphism D dynamic polymorphism
E templates