Faculty of Computers and Artificial Intelligence
Cairo University

Mid-term Exam

Program:	**Computer Science / Software Engineering**	**Date:**	**10/12/2020**
Course Name:	**Concepts of Programming Languages**	**Duration:**	**1 hour**
Course Code:	**CS317 / SCS317**	**Total Marks:**	**20 marks**
Instructor(s):	**Dr. Amin Allam**		

ID: .... **Full Name in Arabic:** ....

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 أي كتب أو ملازم أو أوراق داخل اللجنة والمخالفة تعتبر حالة غش .

Exam consists of 30 multiple-choice questions in 3 pages

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

```
<expr> -> <expr> * <term> | <term>
<term> -> <var> + <term> | <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 $\star$ 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] -> <expr>[2]*<term> 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

10 In an attribute grammar, the syntax rule <expr>[1] -> <expr>[2]*<term> 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

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

```
1 int r=8;
2 void F(int& t){t=5;}
3 int main()
4 {
5    int a=5, b=6; float c=1.2, d=2.9;
6    int x=a+r; 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 An interpreter interprets a statement inside a loop:
A once B twice C number of times equal to number of loop iterations
D same as compiler 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 values at load time B bind to type at compile time C bind to storage at load 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 `(false && g)`:
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 x,y,z,r;
 2 void main()
 3 {
 4    int x,y,z;
 5    while(x<10)
 6    {
 7       int x,y,w;
 8       if(x<5) {int x;}
 9    }
10 }
```

28 Assuming static scoping, the referencing environment of Line 8 does not contain:
A `x` 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 `x,y,z,r` of line 1 B `x,y,z` of line 4 C `x,y,z` of line 4 and `r` D `r` E `x,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