1. Print your name in the space labeled **NAME**.
2. Print **CMSC 4023** in the space labeled **SUBJECT**.
3. Print the test number and version, **T3/V1**, in the space labeled **TEST NO.**
4. Print the date, **12-12-2018**, in the space labeled **DATE**.
5. Print your **CRN** number, **12105**, in the space labeled **PERIOD**.
6. This is a closed-book examination. No reference materials are permitted. No notes are permitted.
7. You may use your personal calculator on this test. You are prohibited from loaning your calculator or borrowing a calculator from another person enrolled in this course.
8. You may not consult your neighbors, colleagues, or fellow students to answer the questions on this test.
9. Cellular phones are prohibited. The possessor of a cellular phone will receive a ***zero*** (***0***) if the phone rings or is visible during the test.
10. Mark the best selection that satisfies the question. If selection **b** is better that selections **a** and **d**, then mark selection **b**. Mark only ***one*** selection.
11. Darken your selections completely. Make a heavy black mark that completely fills your selection.
12. Answer all **50** questions.
13. Record your answers on SCANTRON form **882-E (It is green!)**
14. When you have completed the test, place your scantron, face up, between pages 2 and 3 of your questionnaire and submit both the questionnaire and your scantron to your instructor.

1. (Ch. 1, p. 22) Which of the following is not a fundamental feature of an object-oriented programming language according to Sebesta?

    a. data abstraction
    b. inheritance
    c. dynamic method binding
    d. interface encapsulation

2. (Ch. 1, p. 19) What has been the strongest influence on programming language design over the past 50 years?

    a. imperative programming methodology
    b. the von Neumann architecture
    c. BNF specification of programming language
    d. the exponential growth of memory

3. (Ch. 1, p. 8) Which of the following is not a programming language evaluation criteria according to Sebesta?

    a. syntax design
    b. expressivity
    c. restricted aliasing
    d. type design

4. (Ch. 1, p. 27) Which of the following is **NOT** a phase of compilation?

    a. lexical analysis
    b. semantic analysis
    c. syntax analysis
    d. static analysis

5. (Ch. 1, p. 2-3.) Which of the following is NOT a reason for studying concepts of programming languages according to Sebesta?

    a. Increased capacity to express ideas.
    b. Improved background for choosing appropriate languages.
    c. Increased ability to design new languages.
    d. Increased ability to learn new languages.

6. (Ch. 3, p. 121-122) Given the grammar in the figure below, select the leftmost derivation of the string *a*+*b***c*.

$e \rightarrow t$
$e \rightarrow e+t$
$e \rightarrow e-t$
$t \rightarrow f$
$t \rightarrow t*f$
$t \rightarrow t/f$
$t \rightarrow t\%f$
$f \rightarrow (e)$
$f \rightarrow \mathbf{id}$

Figure 6.

a.

Step	Sentential Form	Explanation
1	*e*	
2	*e*+*t*	
3	*e*+*t***f*	
4	*t*+*f****id(c)**	
5	*f*+**id(b)*****id(c)**	
6	**id(a)*****id(b)**+**Id(c)**	

b.

Step	Sentential Form	Explanation
1	*e*	
2	*e*+*t*	
3	*e*+*t***f*	
4	*t*+*f****id(c)**	
5	*t*+**id(b)*****id(c)**	
6	*f*+**id(b)*****id(c)**	
7	**id(a)+id(b)*id(c)**	

c.

Step	Sentential Form	Explanation
1	*e*	
2	*e*+*t*	
3	*e*+*t***f*	
4	*t*+*t***f*	
5	*f*+*t***f*	
6	**id(a)**+*t***f*	
7	**id(a)**+*f***f*	
8	**id(a)+id(b)***f*	
9	**id(a)+id(b)*id(c)**	

d.	**Step**	**Sentential Form**	**Explanation**
	1	*e*	
	2	*e*+*t*	
	3	*t*+*t*	
	4	*f*+*t*	
	5	**id(a)**+*t*	
	6	**id(a)**+*t***f*	
	7	**id(a)**+*f***f*	
	8	**id(a)+id(b)****f*	
	9	**id(a)+id(b)*id(c)**	

7. (Ch. 3, p 120) The first language to employ BNF was

   a. Pascal
   b. LISP
   c. FORTRAN
   d. Algol60

8. (Ch. 3. Lecture Notes) Given the grammar in the figure below, select the rightmost derivation of the string *a*+*b***c*.

$$
\begin{aligned}
e &\rightarrow t \\
e &\rightarrow e+t \\
e &\rightarrow e-t \\
t &\rightarrow f \\
t &\rightarrow t*f \\
t &\rightarrow t/f \\
t &\rightarrow t\%f \\
f &\rightarrow (e) \\
f &\rightarrow \mathbf{id}
\end{aligned}
$$

Figure 8.

a.	**Step**	**Sentential Form**	**Explanation**
	1	*e*	
	2	e+t	
	3	e+t*f	
	4	t+f***id**(c)	
	5	f+**id**(b)***id**(c)	
	6	**id**(a)***id**(b)+**id**(c)	

b.	Step	Sentential Form	Explanation
	1	*e*	
	2	*e*+*t*	
	3	*t*+*t*	
	4	*f*+*t*	
	5	**id(a)**+*t*	
	6	**id(a)**+*t***f*	
	7	**id(a)**+*f***f*	
	8	**id(a)+id(b)****f*	
	9	**id(a)+id(b)*id(c)**	

c.	Step	Sentential Form	Explanation
	1	*e*	
	2	*e*+*t*	
	3	*e*+*t***f*	
	4	*t*+*t***f*	
	5	*f*+*t***f*	
	6	**id(a)**+*t***f*	
	7	**id(a)**+*f***f*	
	8	**id(a)+id(b)****f*	
	9	**id(a)+id(b)*id(c)**	

d.	Step	Sentential Form	Explanation
	1	*e*	
	2	*e*+*t*	
	3	*e*+*t***f*	
	4	*e*+*t****id(c)**	
	5	*e*+*f****id(c)**	
	6	*e*+**id(b)*id(c)**	
	7	*t*+**id(b)*id(c)**	
	8	*f*+**id(b)*id(c)**	
	9	**id(a)+id(b)*id(c)**	

9. (Project p01 notes) What sequence of tokens is recognized by the regular expressions shown in the figure below given the input string **+123.1**?

```
[+|-]?[0-9]+\.[0-9]*([E|e][+|-]?[0-9]+)?
[+|-]?[0-9]+
\.[0-9]+
“.”
[+|-]
[0-9]+
```

Figure 9.

   a. 3 tokens as follows **+ 123 .1**
   b. 1 token as follows **+123.1**
   c. 4 tokens as follows **+ 123 . 1**
   d. 2 tokens as follows **+123 .1**

10. (Ch. 3, p. 125 - 128) Which selection correctly orders the precedence of operators in the grammar in the figure below? Operators are ordered from left to right, highest to lowest.

$e \rightarrow t$
$e \rightarrow e+t$
$e \rightarrow e-t$
$t \rightarrow f$
$t \rightarrow t*f$
$t \rightarrow t/f$
$t \rightarrow t\%f$
$f \rightarrow (e)$
$f \rightarrow \mathbf{id}$

Figure 10.

   a. **() * / % + -**
   b. **+ * - / % ()**
   c. **() * + / - %**
   d. **+ - * / % ()**

11. (Ch. 4, p. 193) Which of the following grammars is pairwise disjoint?

Id	LHS		RHS
1	*A*	→	**a***B*
2	*A*	→	**b***A***b**
3	*A*	→	*B***b**
4	*B*	→	**c***B*
5	*B*	→	**d**

Figure 11 a.

Id	LHS		RHS
1	*A*	→	**a***B*
2	*A*	→	*BA***b**
3	*B*	→	**a***B*
4	*B*	→	**b**

Figure 11 b.

Id	LHS		RHS
1	*A*	→	*B***a**
2	*A*	→	**b***A***b**
3	*A*	→	**b***B*
4	*B*	→	**c***B*
5	*B*	→	**d**

Figure 11 c.

Id	LHS		RHS
1	*A*	→	*B***a**
2	*A*	→	*BA***b**
3	*B*	→	**a***B*
4	*B*	→	**b**

Figure 11 d.

12. (Ch. 4, p. 196) Consider the grammar shown in Figure 12.1, a sentential form in the grammar shown in Figure 12.2, and a parse tree of the sentence in Figure 12.3. How many phrases are shown in the parse tree?

e → *t*
e → *e*+*t*
t → *f*
t → *t***f*
f → (*e*)
f → **ID**

Figure 12.1

e+*t****ID**

Figure 12.2

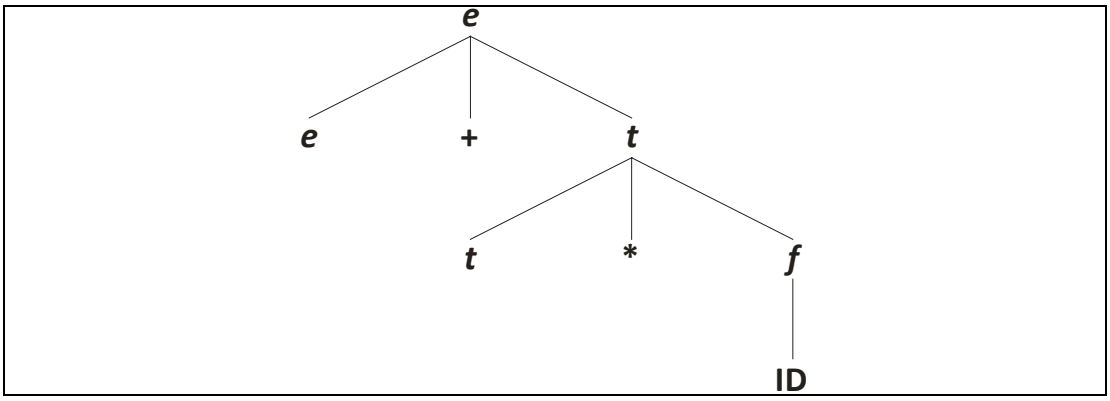

Figure 12.3

a. **1**
b. **2**
c. **3**
d. **4**

13. (Ch. 4, p. 193) Find the FIRST sets for the grammar shown below.

$$
\begin{aligned}
E &\rightarrow TE' \\
E' &\rightarrow +TE' \\
E' &\rightarrow \varepsilon \\
T &\rightarrow FT' \\
T' &\rightarrow *FT' \\
T' &\rightarrow \varepsilon \\
F &\rightarrow (E) \\
F &\rightarrow \mathbf{id}
\end{aligned}
$$

Figure 13

Nonterminal	FIRST set
E	{*T*}
E'	{+,ε}
T	{*F*}
T'	{*,ε}
F	{(,**id**}

Figure 13 a.

Nonterminal	FIRST set
E	{(,**id**}
E'	{+,ε}
T	{(,**id**}
T'	{*,ε}
F	{(,**id**}

Figure 13 b.

Nonterminal	FIRST set
E	{*E'*}
E'	{*E'*}
T	{*T'*}
T'	{*T'*}
F	{),**id**}

Figure 13 c.

Nonterminal	FIRST set
E	{(,**id**}
E'	{+}
T	{(,**id**}
T'	{*}
F	{(,**id**}

Figure 13 d.

14. (Ch. 4, p. 191 - 194) Which of the following is a limitation of the LL grammar class?

a. must be implemented by employing a recursive descent parser
b. left recursion must be eliminated
c. lexical analysis must be implemented using finite automata
d. must be implemented using a pushdown automaton

15. (Ch. 4, p. 193) Find the FOLLOW sets for the grammar shown in Figure 15 below.

Compute FOLLOW(A) for all nonterminals A, by applying the following rules until nothing can be added to any FOLLOW set.

1. Place \$ is FOLLOW(S), where S is the start symbol and \$ is the input right endmarker.
2. If there is a production $A \rightarrow \alpha B \beta$, then everything in FIRST(β) except for $\in$, the empty string, is placed in FOLLOW(B).
3. If there is a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$ where FIRST(β) contains $\in$ (i.e., $\beta \overset{*}{\Rightarrow} \in$), then everything in FOLLOW(A) is in FOLLOW(B).

E → *T*
E → *E* **or** *T*
T → *F*
T → *T* **and** *F*
F → **(***E***)**
F → **not** *F*
F → **true**
F → **false**

Figure 15

Nonterminal	FOLLOW set
E	{**and**,**or**,**)**,**$**}
T	{**and**,**or**,**)**,**$**}
F	{**and**,**or**,**)**,**$**}

Figure 15 a.

Nonterminal	FOLLOW set
E	{**(**,**not**,**true**,**false**}
T	{**(**,**not**,**true**,**false**}
F	{**(**,**not**,**true**,**false**}

Figure 15 b.

Nonterminal	FOLLOW set
E	{**or**,**)**,**$**}
T	{**and**,**or**,**)**,**$**}
F	{**and**,**or**,**)**,**$**}

Figure 15 c.

Nonterminal	FOLLOW set
E	{**)**,**$**}
T	{**or**,**)**,**$**}
F	{**and**,**or**,**)**,**$**}

Figure 15 d.

16. (Ch. 5. p 222) Select the correct storage type for variable *m*.

```
#include <iostream>
#include <iomanip>
using namespace std;
int max(int A[],int N)
{    int m=A[0];
     for (int i=1;i<N;i++) if (A[i]>m) m=A[i];
     return m;
}
int main()
{    int A[]={17,2,-3,11,5,-7,-13};
     cout << max(A,7) << endl;
     return 0;
}
```

Figure 16 Program for Question 16

a. Static variable
b. Stack-Dynamic variable
c. Explicit Heap-Dynamic variable
d. Implicit Heap-Dynamic variable

17. (Ch. 5. p. 233) What is printed by the program in Figure 17 assuming dynamic scope?

```
#include <iostream>
using namespace std;
int a,b;
void print(void){cout << "a=" << a << " b=" << b;}
int p(int& a){int p=2;a=0;b=1;return p;}
void q(void){int b=4;a=3;print();}
int main(){a=p(a);q();return 0;}
```

Figure 17 C++ Program for Question 17

a. a=1 b=3
b. a=2 b=4
c. a=3 b=1
d. a=3 b=4

18. (Ch. 5 p. 225) What is printed by the program in Figure 18 assuming static scope?

```
#include <iostream>
using namespace std;
int a,b;
void print(void){cout << "a=" << a << " b=" << b;}
int p(int& a){int p=2;a=0;b=1;return p;}
void q(void){int b=4;a=3;print();}
int main(){a=p(a);q();return 0;}
```

Figure 18 C++ Program for Question 18

a. a=3 b=4
b. a=2 b=4
c. a=1 b=3
d. a=3 b=1

19. (Ch. 5. p 222) What is the lifetime of variable *p* declared on line 6 of Figure 19?

```
 1.  #include <iostream>
 2.  using namespace std;
 3.  int a;
 4.  int b;
 5.  int p(int& a)
 6.  {   int p=2;
 7.      a=0;
 8.      b=1;
 9.      return p;
10.  }
11.  void q(void)
12.  {   static int b=4;
13.      a=3;
14.  }
15.  int main()
16.  {   a=p(a); q();
17.      return 0;
18.  }
```

Figure 19 Program for Question 19

a. Storage for variable *p* is allocated when function *main* is called and its storage is reclaimed when function *main* returns.
b. Storage for variable *p* is allocated during translation and reclaimed when the file containing executable form of the program in Figure 19 is deleted.
c. Storage for variable *p* is allocated when function *p* is called and its storage is reclaimed when function *p* returns.
d. Storage for variable *p* is allocated at load time and reclaimed when the program in Figure 19 returns control to the operating system.

20. (Ch. 5. p. 277) Assume that each element of array *A* occupies *e* bytes, array *A* has *r* rows whose indexes are 1, 2, …, *r* and *c* columns whose indexes are 1, 2, …, *c*. Array *A* is allocated in row-major order. Without subscripts *A* is the address of the first byte allocated. Which of the following expressions gives the address of the first byte of element *A***[***i***][***j***]**?

    a. $\boldsymbol{A + ((i - 1) \cdot c + j - 1) \cdot e}$
    b. $\boldsymbol{A + i \cdot c \cdot e + j \cdot e}$
    c. $\boldsymbol{A + ((j - 1) \cdot c + i - 1) \cdot e}$
    d. $\boldsymbol{A + j \cdot c \cdot e + i \cdot e}$

21. (Ch. 6. p. 288) Which set of C++ declarations most closely represents the Ada declarations given in the diagram below?

```
type Shape is (Circle,Triangle,Rectangle);
type Colors is (Red,Green,Blue);
type Figure (Form:Shape) is
    record
        Filled:Boolean;
        Color:Colors;
        case Form is
            when Circle =>
                Diameter:Float;
            when Triangle =>
                Left_Side:Integer;
                Right_Side:Integer;
                Angle:Float;
            when Rectangle =>
                Side_1:Integer;
                Side_2:Integer;
        end case;
    end record;
```

Figure 21 Declarations for Question 21

```
enum Shape {Circle,Triangle,Rectangle};
enum Colors {Red,Green,Blue};
struct Type_Triangle {
   int Left_Side,Right_Side;
   float Angle;
};
struct Type_Rectangle {
   int Side_1,Side_2;
};
struct Type_Circle {
   float Diameter;
};
struct Type_Form {
   Type_Circle C;
   Type_Triangle T;
   Type_Rectangle R;
};
struct Figure {
   bool Filled;
   Colors Color;
   Shape Form;
   Type_Form F;
};
```

Figure 21 a

```
enum Shape {Circle,Triangle,Rectangle};
enum Colors {Red,Green,Blue};
struct Type_Triangle {
   int Left_Side,Right_Side;
   float Angle;
};
struct Type_Rectangle {
   int Side_1,Side_2;
};
struct Type_Circle {
   float Diameter;
};
union Type_Form {
   Type_Circle C;
   Type_Triangle T;
   Type_Rectangle R;
};
struct Figure {
   bool Filled;
   Colors Color;
   Shape Form;
   Type_Form F;
};
```

Figure 21 b

```
enum Shape {Circle,Triangle,Rectangle};
enum Colors {Red,Green,Blue};
union Type_Form {
   int Side_1,Side_2;
   float Diameter;
   int Left_Side,Right_Side; float Angle;
};
struct Figure {
   bool Filled;
   Colors Color;
   Shape Form;
   Type_Form F;
};
```

Figure 21 c

```
enum Shape {Circle,Triangle,Rectangle};
enum Colors {Red,Green,Blue};
struct Type_Triangle {
   int Left_Side,Right_Side;
   float Angle;
};
struct Type_Rectangle {
   int Side_1,Side_2;
};
struct Type_Circle {
   float Diameter;
};
struct Type_Form {
   Type_Circle C;
   Type_Triangle T;
   Type_Rectangle R;
};
union Figure {
   bool Filled;
   Colors Color;
   Shape Form;
   Type_Form F;
};
```

Figure 21 d

22. (Ch. 6. p. 293) Which code fragment produces a dangling pointer?

```
#include <iostream>
using namespace std;
int* f(void)
{    int d;
     return &d;
}
int main()
{    int* p=f();
     return 0;
}
```

Figure 22 a

```
#include <iostream>
using namespace std;
int* f(void){return new int;}
int main()
{    int* p=f();
     int  q;
     p=&q;
     return 0;
}
```

Figure 22 b

```
#include <iostream>
using namespace std;
int main()
{    int* p=new int;
     int* q=p;
     delete q;
     return 0;
}
```

Figure 22 c

```
#include <iostream>
using namespace std;
int* f(void){return new int;}
int main()
{    int* p=f();
     p=new int;
     return 0;
}
```

Figure 22 d

23. (Ch. 6. p. 306) Which feature of C++ is primarily responsible for characterizing the language as NOT strongly typed?

    a. mixed-mode coercion
    b. user-defined operator overloading
    c. polymorphic pointers
    d. **union** types

24. (Ch. 6. p. 259) Which of the following declarations does NOT define an ordinal type?

    a. **enum** *color* **{***red*, *green*, *blue***};**
    b. **type** *day* **is ordinal (***Mon*,*Tue*,*Wed*,*Thu*,*Fri*,*Sat*,*Sun***);**
    c. **subtype** *Index* **is** *Integer* **range 1..100;**
    d. **char**

25. (Ch. 7. p. 324) According to Sebesta, what operator usually associates to the right?

    a. exponentiation operator ******
    b. unary minus **-**
    c. assignment **=**
    d. prefix increment **++**

26. (Ch. 7. p. 332) Which of the following code fragments contains an example of coercion?

```
#include <string>
using namespace std;
int main()
{    string s="tomat";
     char c='o';
     s=s+c;
     return 0;
}
```

Figure 26 a

```
#include <iostream>
using namespace std;
int main()
{    double x=1.0;
     double y=2.0;
     int i=(int)(x+y);
     return 0;
}
```

Figure 26 b

```
#include <iostream>
using namespace std;
int main()
{    char A=0x20;
     A = A << 1 | 0x01;
     cout << A << endl;
     return 0;
}
```

Figure 26 c

```
#include <iostream>
using namespace std;
int main()
{    double x=1.0;
     double y=x+1;
     return 0;
}
```

Figure 26 d

27. (Ch. 7. p. 337) Identify the order of evaluation for the expression given in the Figure below? Please note that expressions are evaluated in the order in which they appear from left to right. Expressions are separated by commas in the selections below.

```
int a=0; int b=5;
a>=0||b<5?a=5:b=0;
```

Figure 27 Expression for Question 27

a. *a*>=0, *b*<5, *a*=5, *b*=0
b. *a*>=0, *a*=5
c. *a*>=0, *b*<5, *a*=5,
d. *a*>=0, *b*<5, *b*=0

28. (Ch. 7. p. 321) Assume the following rules of associativity and precedence for expressions.

Operators	Precedence	Associativity
*, /, not	highest	left to right
+, - & mod		left to right
- (unary)		right to left
=, /=, <, <=, >=, >		left to right
and		left to right
or, xor	lowest	left to right

Show the order of evaluation of the expression in the Figure below by parenthesizing all sub-expressions and placing a superscript on the right parenthesis to indicate the order. For example, for the expression

a + *b* * *c* + *d*

the order of evaluation would be represented as

((*a* + (*b* * *c***)**1**)**2** + *d***)**3**

(*a* – *b*) / *c* & (*d* * *e* / *a* – *f*)

Figure 28 Expression for Question 28

a. **(((*a*-*b*)1/*c*)5&(((*d***e*)2/*a*)3-*f*)4)6**
b. **(((*a*-(*b*/*c*)1)2&(((*d***e*)3/*a*)4-*f*)5)6**
c. **(((*a*-*b*)4/*c*)5&(((*d***e*)1/*a*)2-*f*)3)6**
d. **(((*a*-*b*)1/*c*)2&(((*d***e*)3/*a*)4-*f*)5)6**

29. (Ch. 7. p. 321) Mark the selections that satisfy the relation given in Figure 29.1 for the grammar of Figure 29.2. Function $p(op)$ returns the precedence of the argument op. Argument op is an operator in the grammar given in Figure 29.2. The operators given in the grammar of Figure 29.2 include +, -, unary -, *, /, and ^. The operator, unary-, appears in production 8.

$\boldsymbol{p(op_1) \geq p(op_2) \geq \cdots \geq p(op_n) \mid op_i \in \{+, -, *, /, \wedge, unary-\}}$

Figure 29.1

1	*expression*	→	*term*
2	*expression*	→	*term addop expression*
3	*term*	→	*factor*
4	*term*	→	*factor mulop term*
5	*factor*	→	*power*
6	*factor*	→	*power powop factor*
7	*power*	→	**(** *expression* **)**
8	*power*	→	**-** *power*
9	*power*	→	**id**
10	*addop*	→	**+**
11	*addop*	→	**-**
12	*mulop*	→	*****
13	*mulop*	→	**/**
14	*powop*	→	**^**

Figure 29.2

a. $p(+) \geq p(unary-) \geq p(*)$
b. $p(+) \geq p(-) \geq p(*) \geq p(\wedge)$
c. $p(unary -) \geq p(-) \geq p(\wedge)$
d. $p(\wedge) \geq p(*) \geq p(/) \geq p(-)$

30. (Ch. 8. p. 359) What programming language feature illustrated in the code fragment below is prohibited in C#?

```
switch (value) {
    case -1:
        Negatives++;
        break;
    case 0:
        Zeros++;
        goto case 1;
    case 1:
        Positives++;
    default:
        Console.WriteLine("Default\n");
}
```

Figure 30 Code Fragment for Question 30

a. implicit execution of more than one selectable segment
b. unary assignment operators ++ and --.
c. goto statements
d. negative case values.

31. (Ch. 8. p. 373) What programming language contains the grammar given below for a *for-statement*?

for-statement	→	**for** *loop-variable* **in** *object statement*
for-statement	→	**for** *loop-variable* **in** *object statement* **else** *statement*

a. CLU.
b. Ada
c. Perl
d. Python

32. (Ch. 8. p. 368) What is the value of variable Count after the Ada code fragment given below exits the loop?

```
Count: Float:=1.35;
for Count in 1..10 loop
    Sum:=Sum+Count;
end loop;
```

Figure 32 Program for Question 32

a. 10.35
b. The code fragment fails to compile because Ada does not permit a floating point variable to be a loop control variable.
c. 1.35
d. 11

33. (Ch. 8. p. 354) In what language is the following example prohibited?

```
if (sum==0)
    if (count==0)
        result=0;
    else
        result=1;
```

Figure 33 Code Fragment for Question 33

a. Perl
b. Java
c. C#
d. C++

34. (Ch. 8. p. 353) What is printed when the program below is executed?

```
#include <iostream>
using namespace std;
int main()
{   int sum=0,result;
    if (sum=1) result=0; else result=1;
    cout << "sum=" << sum << " result=" << result;
    cout << endl;
    return 0;
}
```

Figure 34 Program for Question 34

a. sum=0 result=0
b. sum=0 result=1
c. sum=1 result=0
d. sum=1 result=1

35. (Ch. 8. p. 383) What is the relationship between integer variables *m1*, *m2*, and *m3* after exiting the loop containing guarded command proposed by Dkjkstra shown below?

```
do  m1 > m2 -> t:=m1; m1:=m2; m2:=t;
[]  m2 > m3 -> t:=m2; m2:=m3; m3:=t;
od
```

Figure 35 Program for Question 35

a. $m1 \leq m2 \leq m3$
b. $m3 \leq m2 \leq m1$
c. $m1 < m2 < m3$
d. $m3 < m2 < m1$

36. (Ch. 8. p. 388) The multiple-selection statement given in Figure 36.1 is translated to a multiple-selection statement in Figure 36.2. What is the language employed in Figure 36.2?

```
if ((k == 1) || (k == 2)) j = 2 * k – 1
if ((k == 3) || (k == 5)) j = 3 * k + 1
if (k == 4) j = 4 * k – 1
if ((k == 6) || (k == 7) || (k == 8)) j = k - 2
```

Figure 36.1 Multiple-selection statement for Question 36

```
case k is
    when 1..2=> j=2*k-1;
    when 3| 5=> j=3*k+1;
    when 4=> j=4*k-1;
    when 6..8=> j=k-2;
end case;
```

Figure 36.2 Multiple-selection statement for Question 36

a. Ada
b. Ruby
c. Fortran 95
d. Python

37. (Ch. 7. p. 332) Which of the following code fragments contains an example of coercion?

```
#include <string>
using namespace std;
int main()
{   string s="tomat";
    char c='o';
    s=s+c;
    return 0;
}
```

Figure 37 a

```
#include <iostream>
using namespace std;
int main()
{   double x=1.0;
    double y=2.0;
    int i=(int)(x+y);
    return 0;
}
```

Figure 37 b

```
#include <iostream>
using namespace std;
int main()
{   char A=0x20;
    A = A << 1 | 0x01;
    cout << A << endl;
    return 0;
}
```

Figure 37 c

```
#include <iostream>
using namespace std;
int main()
{   double x=1.0;
    double y=x+1;
    return 0;
}
```

Figure 37 d

38. (Ch. 9 p 427-429) What characteristic of nested subprograms makes this feature undesirable in modern programming languages?
    a. Many programming languages do not allow nested subprograms because it is believed that it will lead to ambiguity within the program as well as error compilations. If someone were to declare a subroutine that needed to access variables for two subprograms and they declare those variables in the global scope, this would be unsafe because there already may be a variable of the same name there. In this case, global variables would be a hindrance to readability.
    b. Nested subprograms have the potential to cause additional problems over those that are not nested. Nested subprograms introduce complexity with parameters involving scope such as whether parameters from one subprogram can be passed to another and which version of a variable has precedence. Many contemporary languages use more of an object oriented approach to abstraction which makes it desirable for each part of a program to do one and only one thing. Nested subprograms tend to widen that scope so abstraction is lost and multiple operations are happening at once. Often side effect – unwanted ones for the most part – tend to creep into any more complex section of code. Not only does complexity increase but with nested subprograms, often, readability diminishes. With any increase in chance of error, any unwanted side effect, or decrease in readability the efficiency of the program and of those working with decreases.
    c. Nested subprograms are possibly not included in contemporary languages for two possible reasons. The first being that nested subprograms can lead to ambiguity in the programs. The second reason is that nested subprograms can cause serious compilation errors.
    d. Nested subprograms present a challenge for non-local references when subprograms can be parameters because of the ambiguity of which locality is to be placed in the static chain. Omitting nested subprograms eliminates the need for a static chain making subprogram invocation and return simpler and faster.

39. (Ch. 9 p 407-414) Determine the parameter passing method for the code shown in Figure 39.1 given that the values of local variables after the last call to function *swap* and before exiting function *main* is as shown in Figure 39.2.

```
void swap(int a, int b)
{    int temp
     temp=a;
     a=b;
     b=temp;
}
int main()
{    int value=2,list[5]={1,3,5,7,9};
     swap(value,list[0]);
     swap(list[0],list[1]);
     swap(value,list[value]);
     return 0;
}
```

Figure 39.1 Code for Question 37.

value=2 list={3,2,1,7,9}

Figure 39.2 Variable Values for Question 39.

a. Pass-by-value
b. Pass-by-reference
c. Pass-by value-result.
d. Pass-by-result

40. (Ch. 9 p. 412) Assume that all values in the FORTRAN program shown in Figure 40 are passed strictly by reference to enhance performance. What is printed by the program?

```
PROGRAM MAIN
    IMPLICIT NONE
    INTEGER :: I
    I = 5
    CALL S(1,I)
    I = I + 1
    WRITE(*,*) I
    CONTAINS
        SUBROUTINE S(A,B)
            INTEGER, INTENT (INOUT) :: A, B
            A = 2
            B = 3
        END SUBROUTINE  S
END PROGRAM MAIN
```

Figure 40 Code for Question 40.

a. 2
b. 3
c. 5.
d. 6

41. (Ch. 9. p 433) What is printed by the program shown below?

```
#include <iostream>
using namespace std;
template <class T>T mx(T u,T v){return u>v?u:v;}
#define Mx(u,v)((u)>(v))?(u):(v)
int main()
{  int a(1),b(0);
   int c=Mx(a++,b);
   cout << " a=" << a;
   cout << " b=" << b;
   int d=mx(a--,b);
   cout << " a=" << a;
   cout << " b=" << b;
   cout << endl;
   return 0;
}
```

Figure 41 Code for Question 41.

a. a=3 b=0 a=2 b=0
b. a=2 b=0 a=1 b=0
c. a=2 b=0 a=2 b=0
d. a=1 b=0 a=0 b=0

42. (Ch. 9, p 428-429) Assuming that deep binding is used to establish the referencing environment, what is printed by the program shown below?

```
program q40;
 var x:integer;
 procedure p1;
  var x:integer;
  procedure p2; begin{p2} writeln('x=',x) end{p2};
  procedure p3;
   var x:integer;
  begin x:=3; p4(p2); end{p3};
  procedure p4(procedure px);
   var x:integer;
  begin x:=4;
   px;
  end{p4};
 begin{p1} x:=1;
  p3
 end{p1};
begin
 x:=2;
 p1
end.
```

Figure 42 Code for Question 42.

a. x=1
b. x=2
c. x=3
d. x=4

43. (Ch. 10, p 466) Select the Figure showing the stack with all activation record instances, including static and dynamic chains, when execution reaches position 1 in the following skeletal program. Assume *Bigsub* is at level 1?

```
procedure Bigsub is
    procedure A is
        procedure B is
            begin -- B
            …   <------------------------------1
            end; -- B
        procedure C is
            begin -- C
            …
            B;
            …
            end; -- C
        begin -- A
        …
        C;
        …
        end; -- A
    begin -- Bigsub
    …
    A;
    …
    end; -- Bigsub
```

Figure 43 Code for Question 43.

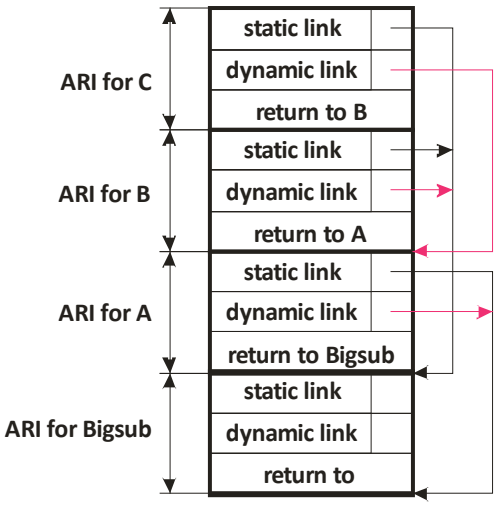

Figure 43 a

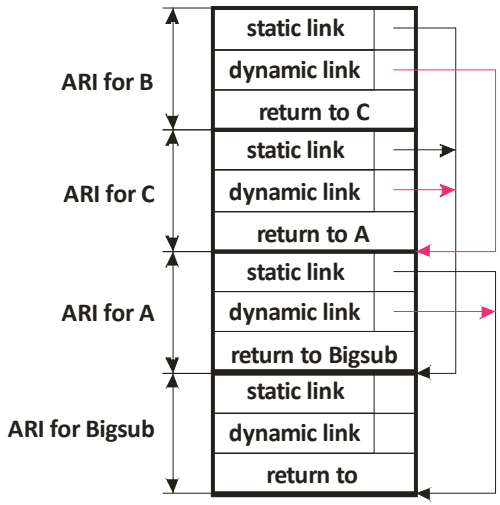

Figure 43 b

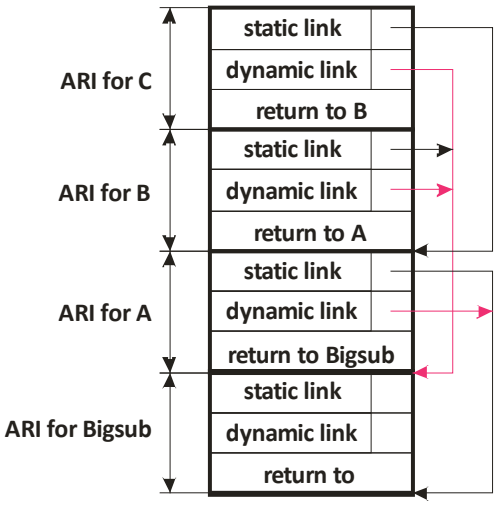

Figure 43 c

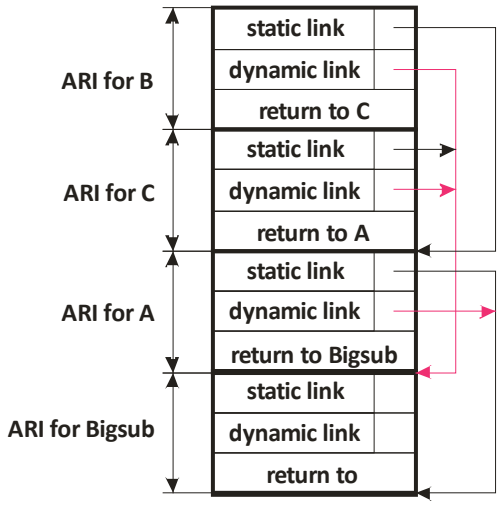

Figure 43 d

44. (Ch. 10) What is the maximum number of activation records allocated to execute the program shown in the Figure below?

```
program p;
    var a:integer;
    function f(n:integer):integer;
    begin{f} if n<=0 then f:=1 else f:=n*f(n-1) end{f};
begin{p}
    a:=f(5)
end{p}.
```

Figure 33 Code for Question 33.

a. 5
b. 2
c. 7
d. 6

45. (Ch. 10) Mark the selection that identifies all the characteristics required to implement FORTRAN activation records.

a. storage for local variables, storage for the return address
b. storage for the return address, static links
c. static links, dynamic links
d. storage for local variables, static links

46. (Ch. 10) Mark the selection that identifies all the characteristics required to implement LISP activation records.

    a. static links, storage for local variables, storage for the return address
    b. dynamic links, storage for local variables, storage for the return address
    c. static links, dynamic links, storage for the return address
    d. static links, dynamic links, storage for local variables

47. (Ch. 10) Mark the selection that identifies all the characteristics required to implement Ada activation records.

    a. static links, storage for local variables, storage for the return address, storage for the return value.
    b. static links, dynamic links, storage for the return address, storage for the return value.
    c. static links, dynamic links, storage for local variables, storage for the return value.
    d. static links, dynamic links, storage for local variables, storage for the return address

48. (Ch. 10) What dangers are avoided in Java and C# by having implicit garbage collection, relative to C++?

    a. The running environments of Java and C# take care of the destructing process. The risk of user can create the dangling pointers when explicitly calling the destructor of an object is avoided.
    b. The biggest danger that is being avoided by having implicit garbage collecting is memory leaks. It is very easy to create a memory leak in C++. Anytime you use the key word **new** resources are being allocated. If the previous statement isn't followed by **delete** at some point, then the allocated memory is never freed and will result in a memory leak. In C# and java, this is done automatically when the resource isn't being used anymore.
    c. The main danger is dangling pointers, causing runtime errors during execution. Another danger is getting incorrect results, it is possible that the pointer being used could be pointing to an incorrect value that user did not know about, giving incorrect results as output as the program runs.
    d. The running environments of Java and C# manage destructors. The risk of creating dangling pointers when explicitly calling the destructor of an object is avoided. Implicit garbage collection avoids memory leaks that careless C++ programmers usually forget about.

49. (Ch. 11) What detail of C++ classes fail to satisfy the requirements of an ideal abstract data type?

    a. The internal structure of the class is visible to the client program.
    b. Pointers must be employed to implement classes requiring aggregate data.
    c. Unnecessary details of the type are hidden from units outside the enclosure that use the type.
    d. The class definition includes only the data representation needed for the type.

50. (Ch. 9) What shortcoming is implemented in the program shown in the figure below?

```
#include <iostream>
#include <string>
using namespace std;
struct StackException {
    StackException(const char* m)
    {   cout << endl << "I am the Stack and I am " << m << "." << endl; }
};
class Stack {
    struct Element {
        Element* prev; string value;
        Element(Element* p,string v):prev(p),value(v){}
    };
    Element* tos;
    void Kill(Element* e){  if (e) {delete e;}}
public:
    Stack():tos(0){}
    ~Stack(){Kill(tos);}
    bool IsFull(void){return false;}
    bool IsEmpty(void){return tos==0;}
    void Push(string v)
    {   if (IsFull()) throw StackException("full");
        Element* e=new Element(tos,v); tos=e;
    }
    string Pop(void)
    {if (IsEmpty()) throw StackException("empty");
    Element* e=tos; string v=e->value; tos=e->prev;
    delete e;
    return v;
    }
};
int main()
{   Stack S;
    string v[]={"I","like","eels","except","as","meals","and","how","they","feels."};
    for (int a=0;a<10;a++) S.Push(v[a]);
    for (int a=0;a< 5;a++) cout << endl << S.Pop();
    cout << endl;
    return 0;
}
```

Figure 50. Program for question 50.

a. Dangling pointer.
b. Invalid memory reference.
c. None.
d. Memory leak.

This page is reserved for computations.