CSC 461 Programming Languages
FINAL EXAM
FALL 2024

(update 1)

Please complete the FINAL QUIZ by

12/16 : Monday - Final Exam Due 11:59 PM

Please answer the questions provided in this exam clearly and to your best ability. If you have questions on any ambiguities on what is being asked, please contact me (email) as soon as possible for a clarification.

By submitting the Quiz, YOU affirm that the work product you turn in on this Exam is yours alone and you have not received help or advice from others. You understand that violation of this statement not only violates Dakota State University student code of conduct, it also degrades the reputation of the university and this program. By doing so it harms all the other students in this course and across the university.

You understand that use of AI generated material is considered plagiarism and will result in a failing grade for this exam.

****You should record your answers using the FINAL Quiz on D2L****

Q1-Q3: Formal Parameters can be **conceptually** classified as **IN** or **OUT** or **INOUT**.

Given the limited information one can get from looking at just this code and, assuming the code is being used correctly, what can you deduce from the following code in terms of classifying them?

```
Function Foo ( A, B , C )
begin

    A := B + C;   // add B to C and assign to A
    print(B)      // print out the value of B
    print(C)      // print out the value of C
    return B;     // return the value of B as the function value

end;
```

How would you **BEST** classify **A B C** given this limited information

Q1 (3 pts) : Variable **A**

a) IN b) OUT c) INOUT

Q2(3 pts) : Variable **B**

b) IN b) OUT c) INOUT

Q3 (3 pts) : Variable **C**

c) IN b) OUT c) INOUT

Q4 (3 pts) : The passing of the Actual parameters to the Formal parameters raises several issues.

Though it is conceptually easy to use text replacement of the formal with the actual parameter when working with PASS-BY-NAME, the implementation can be complex. What is actually passed from the actual to the formal parameter

a) Code to determine the address of the actual

b) Text of the actual to be substituted for the formal

c) A reference to the location of the actual

d) The value of the actual

Q5 (3 pts) : Some language X, requires all the Formal parameters to functions be **in-mode only** as a function call should only be used as part of a mathematical expression. Why would language X make such a strict requirement for functions?

a) So that the formal parameters cannot cause side effects
b) Because a function should return a value to the expression and should be used in it
c) So that a stack-dynamic variable is created

Q6 (3 pts) : A Closure is often used when passing a Function as a parameter. What is a Closure?

a) A Halverson skip
b) A pairing of the function with its referencing environment
c) A method to skip over a level of scoping
d) A static to dynamic link pairing

Q7 (3 pts) : As discussed in class, The C-like assignment expression **x = 3;** has a side effect. What is it?

a) x is assigned the value 3
b) the expression evaluates to 3
c) it evokes the Halverson illusion
d) The Name binding for x is modified

Q8 (3 pts) : An Unambiguous grammar will always

a) have meaningful variable names
b) be well commented to help understand the code
c) have only one distinct parse tree
d) have a well-balanced parse tree

Q9 (3pts) : The inclusion of the optional "else" with the "if" statement helps the user with reliability by enforcing __

a) Mutual exclusion
b) The order of precedence
c) Type checking
d) Short-circuiting

(VOIDED) (will only have one answer on Quiz so all get the points)
Q10 (3 pts) : The Pointer type is designed to be used for two distinct purposes. One is to allow Pass-by-Reference. The other is to ___

a) make code more readable.
b) make code more reliable.
c) access anonymous variables.
d) provide a form of indirect addressing.

Q11 (3 pts) : In what type of static-scoping languages are closures NOT useful?

a) Those that do not allow nested subprograms.
b) Imperative
c) Those that allow nested subprograms.
d) General-purpose

<u>Q12-15:</u> For the following code, show the output for the 4 output statements (**#1..#4**) under the parameter passing method stated.

Assume static scoping

```
PROGRAM EX1;
int i;        // global
int A[3];  // global {array starts at 1}

        PROCEDURE P1( int x, int y)
        BEGIN
                y := 2;
                i := 3;
        END;

BEGIN  //main
        A[1]:= 6;  A[2]:= 12;   A[3]:= 11;
        i := 2;
        P1(A[i], i);          // first call
       PRINT(i)          // #1 prints value of i
       PRINT_A(A);    // #2  assume function that prints all the values found in the array A
        i := 1;
       P1(i, A[i]);        // second call
       PRINT(i)         // #3  prints value of i
       PRINT_A(A);  // #4  assume function that prints all the values found in the array A
END.
```

Assume x is passed by **Name** and y is passed by **Reference**.

Q12 (2 pts) :
#1 :

a) 3
b) 4
c) 2
d) None of the others

Q13 (2 pts) :
#2:

a) 6 12 12
b) 6 12 11
c) 2 11 12
d) None of the others

Q14 (2 Pts) :
#3:

a) 3
b) 4
c) 2
d) None of the others

Q15 (2 pts) :
#4:

a) 2 12 11
b) 6 12 11
c) 11 11 11 11
e) None of the other answers

Q16 (3 pts) : What does this evaluate to in Lisp?

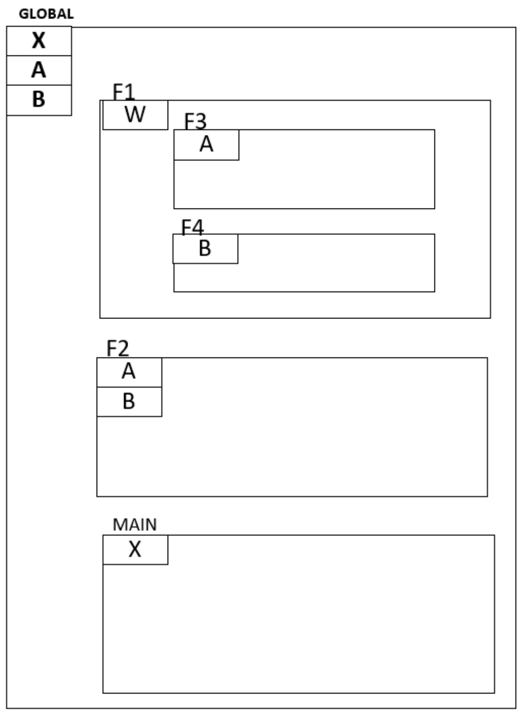
(car (cdr (cdr ‘((Happy New Year) Toys ((Merry X-Mas) to you) Dec25 Break yea!))))

a) (Dec25 Break yea!))
b) (Merry X-Mas)
c) Merry X-Mas to you
d) ((Merry X-MAS) TO YOU)
e) None of there

Q17 (3 pts) : How many members are in the following Lisp list

((a b (c (d f)) c (((y (f))))))

a) 1
b) 2
c) 8
d) 3
e) None of the others



(Fig 1) Scope Diagram for Program P.

Assume program P always starts in MAIN !

Q18 (3 pts) In Fig 1, using **STATIC** Scoping, what is the referencing environment if in MAIN?

a) {Global :(A and B) , MAIN:X}
b) {MAIN: X, F1:W, F3:A, F4:B}
c) {Global :A , MAIN: X, F1:W, F4:B}
d) {Global :(X and A) , F1:W, F4:B}
e) Can't tell by looking at fig 1.

Q19 (3 pts) in Fig 1, assuming the program starts executing in MAIN and then calls F1, which then calls F3. Using **STATIC** Scoping, what is the referencing environment if in F3?

a) {Global :(X and B) , F1:W, F3:A}
b) {MAIN: X, F1:W, F3:A, F4:B}
c) {Global :A , MAIN: X, F1:W, F4:B}
d) {Global :(X and B) , F1:W, F4:B}
e) Can't tell by looking at fig 1.

Q20 (3 pts) in Fig 1, assuming the program starts executing in MAIN and then calls F1, which then calls F2. Using **DYNAMIC** Scoping, what is the referencing environment if in F4?

a) {Global :(X and A and B) , F1:W, F4:B}
b) {MAIN: X, F1:W, F3:A, F4:B}
c) {Global :A , MAIN: X, F1:W, F4:B}
d) {MAIN:X, F1:W, F2: (A and B)}
e) Can't tell by looking at fig 1.

Q21 (3 pts) in Fig 1, assuming the program starts executing in MAIN and then calls F2, which then calls F1 which then calls F3 that then calls F4. Using **STATIC** Scoping, what variable are **"ALIVE"** if in F4?

f) {Global :(X and A and B) , F1:W, F4:B}
g) {Global :(X and A and B) , MAIN:X, F2(A and B) ,F1:W, F3: A, F4:B}
h) {Global :A , MAIN: X, F1:W, F4:B}
i) {Global :(X and A) , F1:W, F4:B}
j) Can't tell by looking at fig 1.

Q22 (3 pts) (updated) in Fig 1, assuming the program starts executing in MAIN and then calls F1, which then calls F3 that then calls F4. Using **DYNAMIC** Scoping, what variable are **"ALIVE"** if in F4?

a) {Global :(X and A and B) , F1:W, F4:B}
b) {Global :(X and A and B) , MAIN:X, F1:W, F3: A, F4:B}
c) {MAIN: X, F1:W, F3:A, F4:B}
d) {Global :(X and A) , F1:W, F4:B}
e) Can't tell by looking at fig 1.

Q23 (3 pts) Variable Descriptors need NOT stay around for the life of the variable if

a) The variable storage binding is stack-dynamic
b) The variable descriptor is used during compile time
c) None of the variable descriptor's attributes are dynamic
d) The variable is stored on the Heap

Q24 (3 pts) Many languages set the lower bound for an array range to 1 (one) because

a) Most languages use row-major data storage for arrays
b) It is more readable and writable for users
c) No language can use a negative number as an index value
d) It makes indexing more efficient

Q25 (3 pts) References, unlike Pointers will

a) never be used with anonymous variables on the heap
b) never need to be dereferenced by the programmer
c) often allow their memory cell's values to be manipulated by the programmer
d) actually reference other things

Q26 (3 pts) A Multiple Selection statement

a) will always enforce mutual exclusion
b) cannot use compound statements
c) can be written to enforce mutual exclusion
d) ALL of these

Q27 (3 pts) Our pseudo-code language we developed at the start of the semester has the MOVE operation (+0). Given the **limited number of statements our design allowed (only 20),** what unimplemented operation did we also use it for?

a) Jump
b) Increment
c) Invert
d) Assignment

Q28 (3 pts) Early programming languages often only implemented Static variable storge binding which

a) Limited the number of characters a variable name could have
b) Did not allow for recursion
c) Made things run less efficiently
d) All of the others

Q [29-33]

Given the following Extended BNF , which strings are a **valid** or i**nvalid** "LINE"?

- * is zero or more
- + is one or more

$$<run> \rightarrow \{0|1\}$$

$$<squeeze> \rightarrow \{010\}^{+} \mid \{101\}^{*}$$

$$<\mathbf{LINE}> \rightarrow <squeeze> <run> <squeeze>$$

(*NOTE: This is the **LINE** to check*)

Valid/Invalid "LINE" (V or I)

Q29 (2 pts) 1011010

Q30 (2 pts) 1010100101

Q31 (2 pts) 01001010

Q32 (2 pts) 1101

Q33 (2 pts) 0

Q34 (3 pts) A rightmost, leftmost, or neither right nor left most approach can be used in the derivation of a sentence in an ambiguous grammar. If each of these is used on the same 1 sentence of such a language, how many possible different parse trees could be involved?

a) 1
b) 2
c) 3
d) More than 3 are possible

Q35 (3 pts) The usefulness of a Language Recognizer is

a) To generate useful syntactically correct code for a given language (like a button to be pushed)
b) To determine if a statement is syntactically correct for a given language
c) Provide a set of rules for syntactically correct statements to help users understand the language
d) None of the others

Q36 (3 pts) The Scope and Lifetime of a variable

a) if a Global STATIC variable, will be from the start of execution to the end of execution of the program
b) if a STACK-DYNAMIC variable, will be from the function invocation to the end of the function
c) both answer a) and b)
d) Neither answer a) or b)

Q37 (3 pts) Are you glad the semester is over ?

a) Yes