**California University of PA**

**Dept. of Computer Science, Info Systems, and Engineering Technology**

**ACSC 455 Structures of Programming Languages**

**Spring 2023**

**= Project =**

**Project 1**

**Andrew Bissell**

**Date Submitted: 02/17/2023**

**Question 1:**

Prove that the following grammar is ambiguous.

<S> => <A>

<A> => <A> + <A> | <id>

<id> => a | b | c

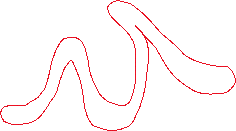
**Answer 1:**

<S> <S>

<A> <A>

<A> + <A> <A> + <A>

<id> <A> + <A> <A> + <A> <id>



<a> <id> <id> <id> <id> <c>

<b> <c> <a> <b>

a + b + c

**Question 2:**

Write a C program that does a large number of references to elements of two-dimensioned arrays, using only subscripting. Write a second program that does the same operations but uses pointers and pointer arithmetic for the storage-mapping function to do the array references.

**Question 2a:** Compare the time efficiency of the two programs. Which one is more efficient and why?

**Question 2b:** Which of the two programs is likely to be more reliable and why?

**Answer 2:**

|  |  |  |
| --- | --- | --- |
| Array Size | Subscripting Time (seconds) | Pointer Time (seconds) |
| 10 | 0.074 | 0.048 |
| 30 | 0.568 | 0.424 |
| 50 | 1.397 | 1.112 |
| 70 | 2.493 | 1.873 |
| 90 | 3.458 | 0.672 |
| 110 | 4.588 | 0.543 |
| 130 | 4.415 | 0.760 |
| 150 | 5.032 | 1.007 |
| 170 | 5.072 | 1.296 |
| 190 | 5.222 | 1.548 |
| 210 | 6.047 | 2.195 |
| 230 | 6.473 | 2.363 |
| 250 | 7.191 | 2.932 |

See attached programs!

**Answer 2a:**

Pointer is more efficient because the pointer is the variable doing all of the work instead of having two loops run through a large amount of memory.

**Answer 2b:**

The pointer would be more reliable if the user knows how to avoid hanging pointers and other such issues that can arise with the use of pointers. The pointer is more reliable because it can easily step through the array because of row major form and it will take up less space since the pointer is doing the indexing and pointing to each array slot as it moves through the array.

**Question 3:**

Attached is the BNF Example 3.6 in the book that requires (1) expression’s data type will be **int** only when both operands are **int**, and (2) the data types on both sides of the assignment operator “=” must be the same. Now change/add/remove the semantic rules/predicates **(i.e., you are not going to change the Syntax rules)** so that

1) If there are two operands on the right side of the assignment, i.e., syntax rule #2, then

a. Data types of the two operands must be the same, and

b. (either both sides have same data type) or (left side is real and both operands on right side are int). In other words, the following are legal: *int=int+int; real=real+real;* or *real=int+int;* But *int=real+real* is illegal.



**Answer 3:**

The following will be changed from the above rules to include the two operands.

2. Semantic rule: if(<var>[2].actual\_type == int) && <var>[3].actual\_type == int)

then <expr>.actual\_type <= Real || <expr>.actual\_type <= int

else

if(<var>[2].actual\_type == Real) && (<var>[3].actual\_type == Real)

then <expr>.actual\_type <= Real

**Question 4:**

Assume the JavaScript program below was interpreted using static-scoping rules. What value of x is displayed in function sub1? Under dynamic-scoping rules, what value of x is displayed in function sub1?

****

**Answer 4:**

Static Scoping x = 5

Dynamic Scoping x = 10

**Question 5:**

For each of the four marked points in this function, list each visible variable, along with the number of the definition statement that defines it. **(Hints: there is no function calls here, so static scope is used)**



**Answer 5:**

1. a def 1, b def 2, c def 2, d def 2, no e
2. a def 1, b def 2, c def 3, d def 3, e def 3
3. a def 1, b def 2, c def 2, d def 2, no e
4. a def 1, b def 1, c def 1, no d, no e