CSC 455: Structures of Programming Languages

Project 1 Due on Friday (02/17) Midnight to D2L

Policies:

1. Discussions on these questions are welcomed and encouraged. However, you should NOT ask any other person to write solution for you. You should write the names of the persons from whom you received help and cite the references used if any.
2. The total score for this project is 100. Late turn in will cause a 10% deduction on your grade for each late day.

Question 1: (10 points)

8. Prove that the following grammar is ambiguous:

<S> → <A>

<A> → <A> + <A> | <id>

<id> → a | b | c

Question 2: (30 points)

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.

In your answer, you will attach your source codes (.c files), screenshots of your code execution, and answer the following questions:

- Compare the time efficiency of the two programs. Which one is more efficiency? Why?
- Which of the two programs is likely to be more reliable? Why?

Question 3: (20 points)

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.

1. Syntax rule: <assign> → <var> = <expr>
   Semantic rule: <expr>.expected_type ← <var>.actual_type
2. Syntax rule: <expr> → <var>[2] + <var>[3]
   Semantic rule: <expr>.actual_type ←
   if (<var>[2].actual_type == int) and
   (<var>[3].actual_type == int)
   then int
   else real
   end if
   Predicate: <expr>.actual_type == <expr>.expected_type
3. Syntax rule: <expr> → <var>
   Semantic rule: <expr>.actual_type ← <var>.actual_type
   Predicate: <expr>.actual_type == <expr>.expected_type
4. Syntax rule: <var> → A | B | C
   Semantic rule: <var>.actual_type ← look-up (<var>.string)

Question 4 (20 points)

Assume the JavaScript program on the right 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?

```
var x;
function sub1() {
  document.write("x = " + x + "");
}
function sub2() {
  var x;
  x = 10;
  sub1();
}
x = 5;
sub2();
```

Question 5. (20 points)

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)**

```
void fun(void) {
  int a, b, c; /* definition 1 */
  . . .
  while (. . .) {
    int b, c, d; /*definition 2 */
    . . . <------------  1
    while (. . .) {
    int c, d, e; /* definition 3 */
    . . . <------------  2
    }
   . . . <-------------  3
   }
  . . . <----------------  4
}
```

Question 5