1. The grammar for nonterminal type is as follows:

```
type ::= identifier
     | ^ type
     | [ int : int ] type
```

show the LR parsing deterministic finite automata that would be generated to recognize this nonterminal. Number your states, then show the sequence of shifts and reduces that would be followed in recognizing the input \([2 : 4] \uparrow \text{real}\). (There is extra space on the next page for the 2nd part of the question).

2. Imagine we reversed the order of the caret and the type in pointer types in the preceding grammar rule, as in

```
type : type ^
```

if this is the only change we make, show that the resulting grammar is now ambiguous.

3. What is the essential difference between LL and LR grammars? Why is the set of LR languages larger than the set of LL languages? Despite this, why might a designer want to insist that a language satisfy the LL property?

4. Show the code fragment that would be generated for the statement

```
while (i > 3) or (i < 7) do i := i + 1
```

assume variable i is found at location 8 relative to the frame pointer. For comparisons, you can just push the arguments on the stack, then emit a comp instruction, followed by the appropriate branch.
5. Show the code that would be generated for the following procedure. Assume c is a
8 by 8 matrix of 4 byte integer values, indexed from 1 to 8. Then, describe an example
optimization of each of the following types that could be performed on this code:
   A. Algebraic transformations
   B. Loop invariant code removal
   C. Reduction in strength
   D. Busy expressions
(You can leave the code in the form of assignments, ASTs, and branch statements).
You don’t need to rewrite the code after each optimization, just describe the effect.

```pascal
for i := 1 to 8 do
    for j := 1 to 8 do
        if i == j then
            c[i, j] := 1
        else
            c[i, j] := 0;
```