Name: ___________________________________

There are 7 questions, worth a total of 100 points. Attempt all 7 questions. Keep your answers brief and to the point.

You may refer to the following materials:

- Your primary compiler textbook – Cooper&Torczon
- Course lecture slides

You may use electronic copies of these materials but may not access any other information on the Internet, or other resources on your computer.

No other books or other materials, including old exams or homework, are allowed.

The exam booklet is printed on one side only. Use the blank pages for temporary scribbles, or for any answers that 'overflow' the original space. Please score out any working scribbles afterwards, so we do not mistake them for part of you answer!

If you run out of space in the exam booklet for your actual answers, grab an extra sheet from the invigilator (Nat at UW; me at MS) and staple it to this exam booklet.

All answers must be your own. (This examination is not a joint project)

Please wait to turn the page until we tell you to begin.

Overall Score ___________________

1 ________ / 15

2 ________ /10

3 ________ / 25

4 ________ / 15

5 ________ / 20

6 ________ / 5

7 ________ / 10

Question 1: Regular Expressions – 15 points

Write down a regular expression that recognize the strings below. Use only:

- Concatenation
- choice (|)
- repetition (* and + and ?)
- simple character classes like [a-d0-5] and [^a-h]
- abbreviations such as: decDigits = [0-9] if you prefer

A. 3 points. All the 3-digit number in the range 100 to 234 (inclusive).

1 [0-9] [0-9] | 2 [0-2] [0-9] | 2 3 [0-4]

B. 4 points. Currency values. Start with '$'. Insert a comma between each group of digits to the left of the decimal point. Include two digits after decimal point.

Eg: $1,234,567.89
Eg: $12.34

D = [0-9]

$ D D? D? (, D D D) * . D D

(Strictly, Regex meta-characters such as "." should be *escaped* as "\." but we ignore this tweak)

C. 8 points. Pascal defines real (floating-point) numeric constants as follows:

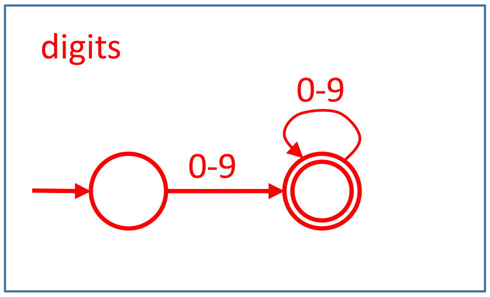
digit = [0-9]

digits = digit+

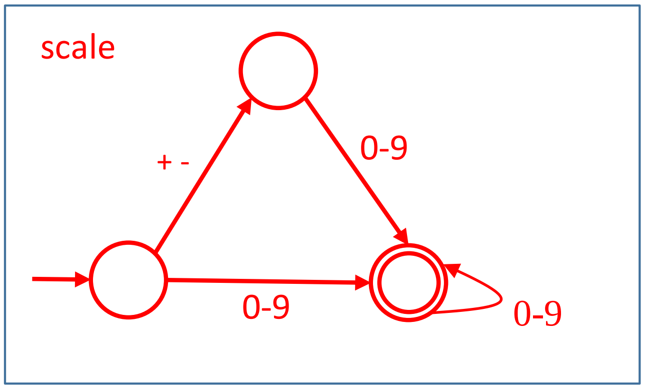
real = digits . digits | digits . digits E scale | digits E scale

scale = digits | (+ | -) digits

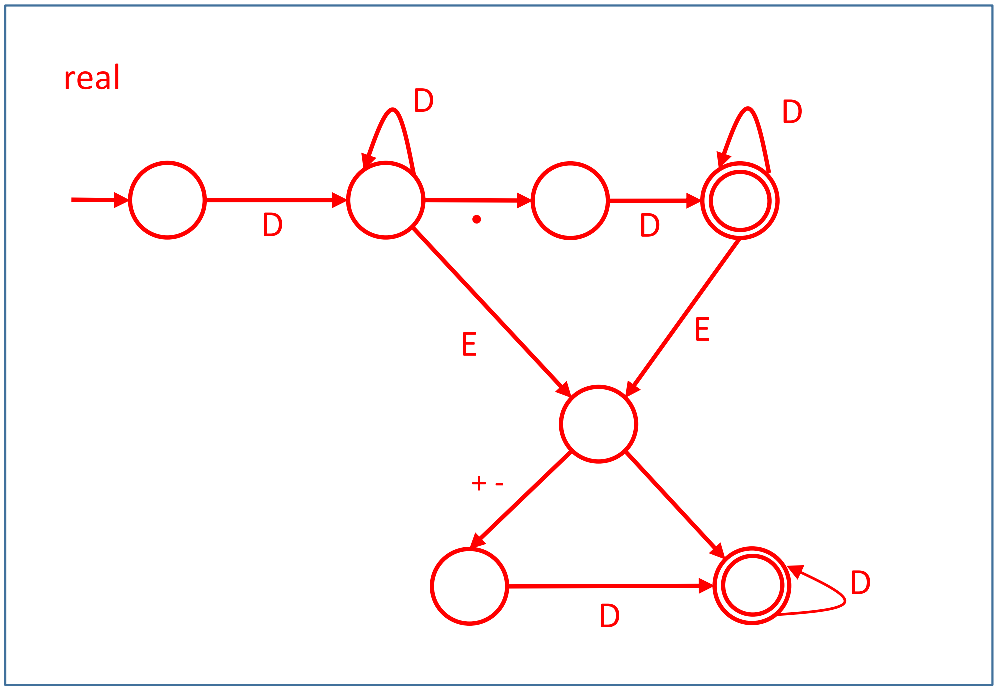
Draw a DFA that accepts Pascal's real constants. (No need to construct an NFA or use algorithms to convert into a DFA. Just draw the DFA)



Note that *digits* is defined to contain at least one decimal digit.



For compactness, denote D = [0-9]. The final answer is:



Question 2: Context Free Grammars – 10 points

A Give a single sentence describing each of the following terms, as they relate to a Context-Free Grammar.

1. Terminal

   A symbol in the grammar that appears only on the RHS of productions. A token that can appear in a sentence (of that grammar's language).

2. Non-Terminal

   A symbol that appears on the LHS of productions.

3. Start Symbol

   The Non-Terminal representing the start, or “goal symbol”, of the grammar.

4. Production (sometimes called a “rule”)

   Shows what a Non-Terminal (on LHS) can be expanded into (on RHS)

5. Top-Down Parse

   Analyzes a program by constructing a tree, with the Start symbol as the root, and working down towards the leaves.

6. Bottom-Up Parse

   Analyzes a program by constructing a tree, starting from the leaves, and working upwards toward the root.

7. Ambiguous grammar

   A grammar for which there is at least one sentence with two distinct parse trees.

8. Predictive parser

   Top-down parser that knows exactly which production to use next, by examining the next (lookahead) token from the input stream. Requires no back-tracking.

9. LR(0) grammar

   A grammar that can be parsed by a (bottom-up, shift-reduce) parser, without using any lookahead token.

10. Follow set

The Follow Set of a Non-Terminal is the set of Terminals that can legally follow that Non-Terminal's productions.

Question 3: LR Parser – 25 points

In this question, we will construct the LR(0) State Machine (or Discrete Finite Automaton) for the following grammar:

1. $S' \rightarrow S\ \$$ (recall that \$ represents end-of-file)
2. $S \rightarrow (\ L\ )$
3. $S \rightarrow x$
4. $L \rightarrow S$
5. $L \rightarrow L\ ,\ S$

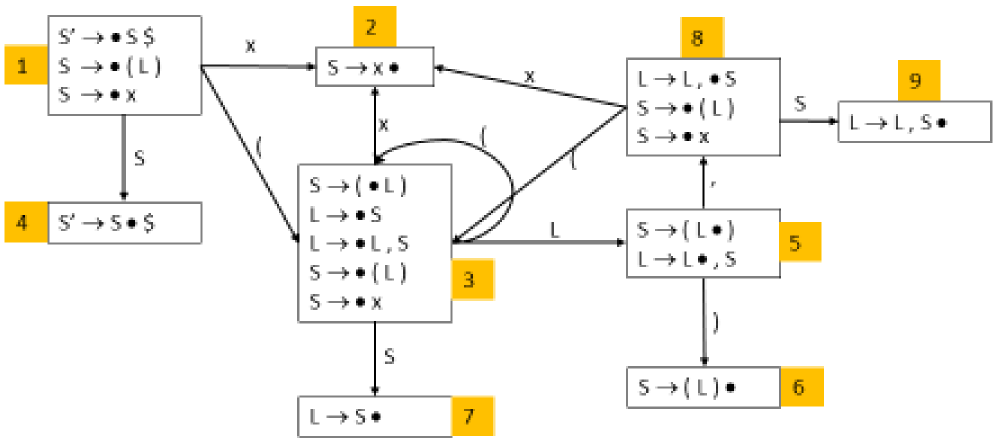
A. 2 points. List all the "dotted items" for production number 2: just the core items – no need to include the closure.

$S \rightarrow \bullet (\ L\ )$ $S \rightarrow (\bullet\ L\ )$ $S \rightarrow (\ L\ \bullet\ )$ $S \rightarrow (\ L\ )\ \bullet$

B. 3 points. Show the initial state for the DFA. Remember to show its start-item (or core), as well as its completion (or closure).

$$\boxed{\begin{array}{l} S' \rightarrow \bullet\ S\ \$ \\ S \rightarrow \bullet\ (\ L\ ) \\ S \rightarrow \bullet\ x \end{array}}$$

C. 15 points. Complete drawing the State Machine for this Grammar. (Recall, in lectures, we also called it the "Handles-DFA").



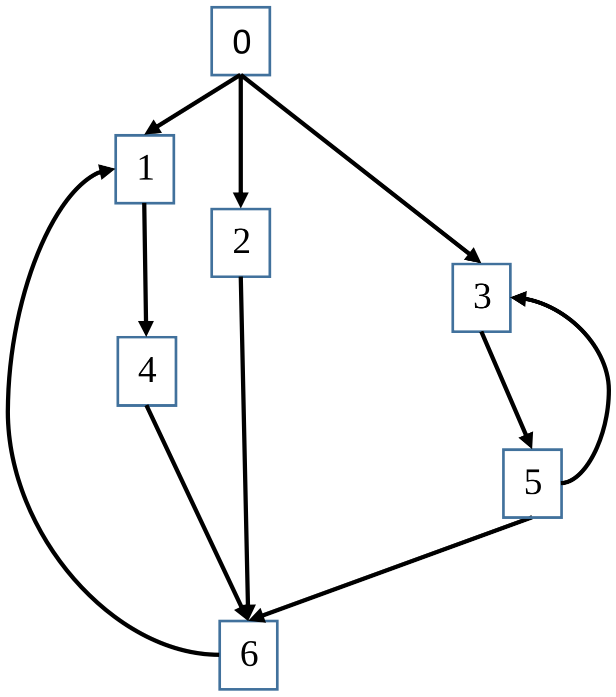
D. 2 points. Is the grammar LR(0)?

Yes

E. 3 points. Justify your answer to part D – why is the grammar LR(0) or why not?

There are no conflicts (Shift-Reduce or Reduce-Reduce) conflicts in the DFA.

Question 4: Flowgraph – 15 points



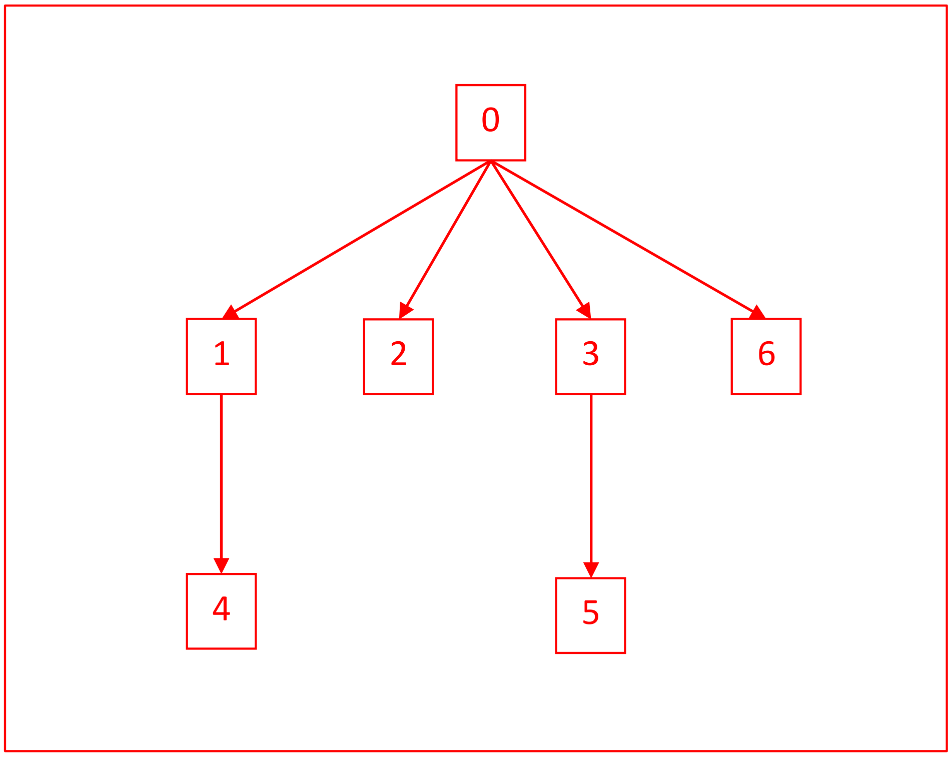
A. 10 points. For each node in the above flowgraph (or control-flow graph), list its Predecessors, Successors; the nodes that it dominates; the nodes that dominate it – by filling in the table below:

Node	Predecessors	Successors	Dominates	Dominated-By
0				
1				
2				
3				
4				
5				
6				

There is no need to show any algorithms or interim work – just fill in the table with your answers.

Node	Predecessors	Successors	Dominates	Dominated-By
0	-	1 2 3	0 1 2 3 4 5 6	0
1	0 6	4	1 4	0 1
2	0	6	2	0 2
3	0 5	5	3 5	0 3
4	1	6	4	0 1 4
5	3	3 6	5	0 3 5
6	2 4 5	1	6	0 6

B. 5 points. Draw the Dominator Tree for the above flowgraph.



Question 5: Stack Frames and Objects – 20 points

A. 15 points. Consider the following short program:

```
class Point {
   int x;
   int y;

   public int distSquaredTo(int tox, int toy) {
      int dx = tox - this.x;
      int dy = toy = this.y;
      int d  = dx * dx  +  dy * dy
      return d;
   }
}

class App {
   public static void main(String[] args) {
      Point p;
      p = new Point();
      p.x = 10;
      p.y = 20;
      p.distSquaredTo(100, 100);
   }
}
```

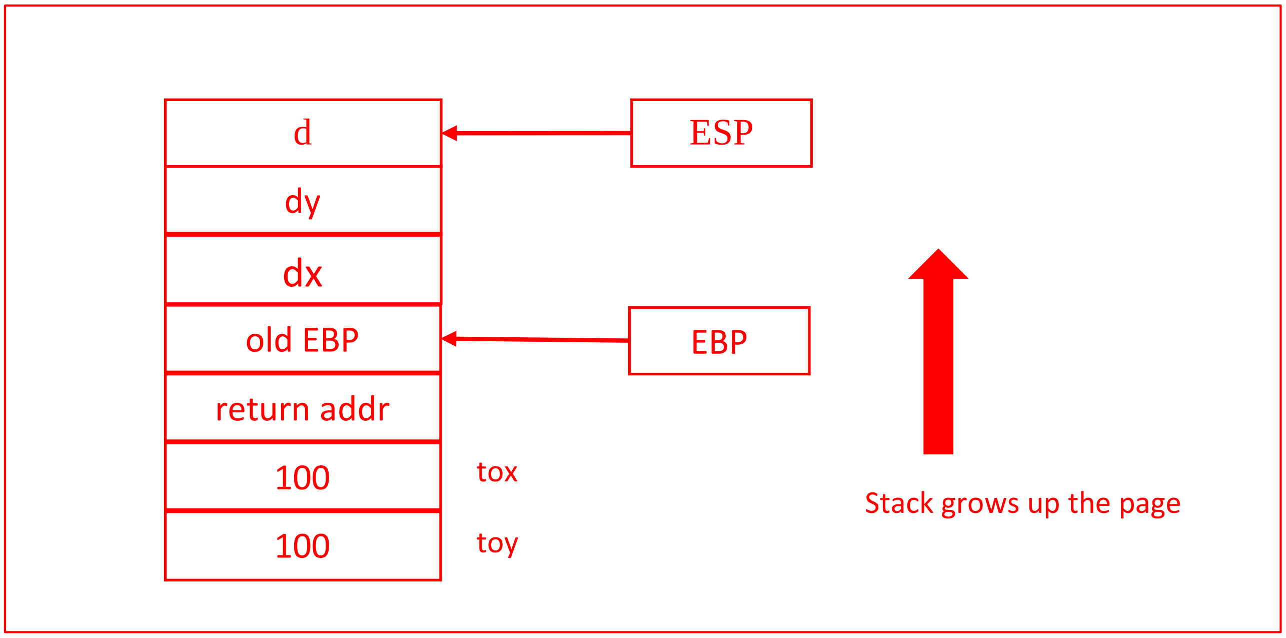
Draw a picture of the stack frame (also called the "activation record") at the point where the method **p.distSquaredTo** is about to execute its first statement. (ie, just after the prolog code in method **distSquaredTo** has executed). Assume we are compiling for the x86 chip using the *cdecl* calling convention that passes all arguments on the stack (ie, the calling convention we used in class lectures, and for the MiniJava project).

Your drawing should include:

- Position and values of the arguments supplied to the **distSquaredTo** method
- Position and values of all variables that are local to the **distSquaredTo** method
- Position of the return address from the **distSquaredTo** method
- To which stack-slot the EBP (Extended Base Pointer) register is pointing; and what that stack-slot contains (a description, not its numeric value)
- To which stack-slot the ESP (Extended Stack Pointer) register is pointing; and what that stack-slot contains

You may draw the stack as growing up the page, or down the page – whichever you find easier. It is important, however, to state which direction you chose!

(Note that the *cdecl* calling convention does not specify the order in which locals are stored in the stack frame. You may choose any order that works – just label which slots you chose for which local variables)



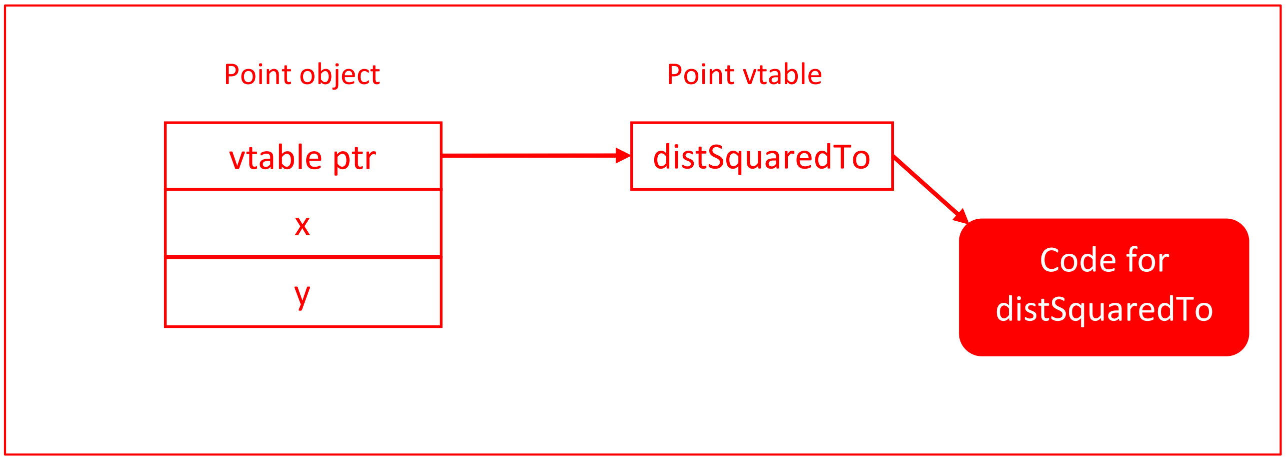
B. 1 point. How did you choose to pass the "this" pointer?

Via register ECX

(alternatively, might have passed "this" pointer as a hidden first argument, using *push toy, push tox, push this*)

C. 4 points. Draw a picture of the object **p** as it appears in memory. Your drawing should include:

- Position and size of each field within the object
- Position and size of the object's Vtable pointer
- Contents of the vtable (assume class **Point** has no implicit base class)



Note that because Point has no implicit base class, the only entry in its vtable is a pointer to the distSquaredTo function (so no entries for inherited virtual methods, such as *hashcode* or *toString*, for example)

Question 6: SSA Form – 5 points

A. 5 points. By adding a subscript to each variable in the following basic block, convert it to SSA form. Any variables that are “live-in” to the basic block should be assigned SSA number 0 (zero).

```
a     =     b     +     c

b     =     3

d     =     a     *     b

a     =     a     -     a

b     =     d

a     =     x
```

$a_1 = b_0 + c_0$

$b_1 = 3$

$d_1 = a_1 * b_1$

$a_2 = a_1 - a_1$

$b_2 = d_1$

$a_3 = x_0$

Question 7: Cache – 10 points

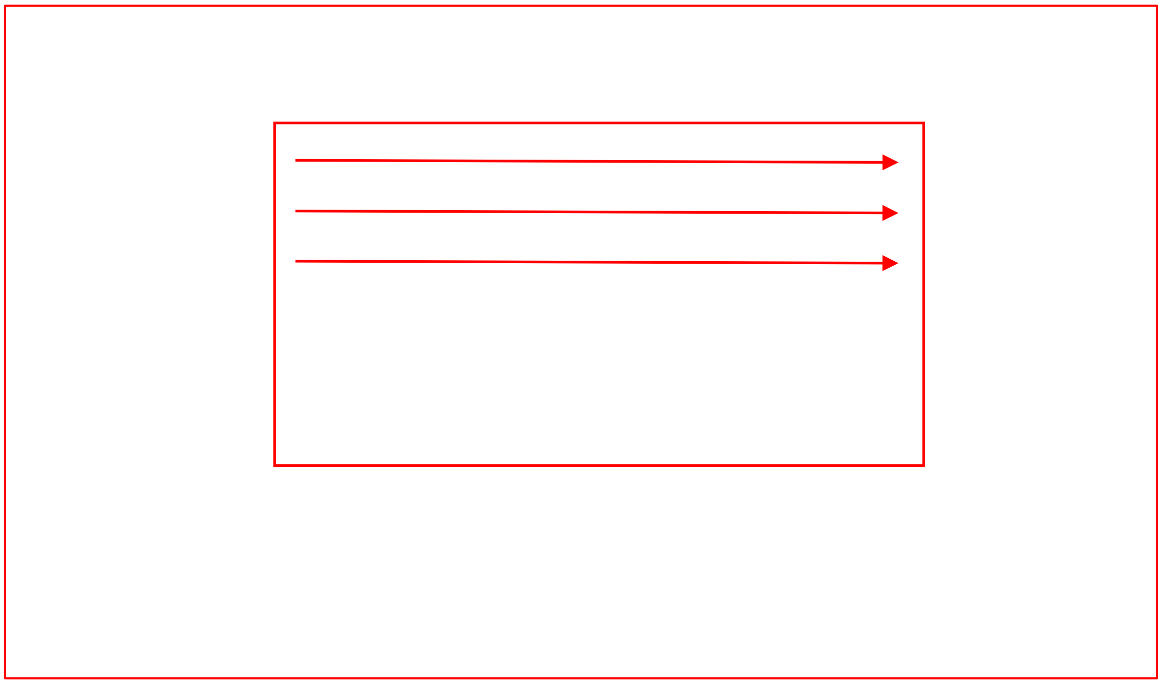
A. 5 points. The following loop is written in a language like C or C++ that supports 2-dimensional arrays. These arrays are stored, in memory, in Row-Major form: all the elements of the first row, followed by all the elements of the second row, etc.

Here is a simple program loop that sets every element of the 2-dimensional, integer array called "a", to hold the value 42.

```
int row;
int col;
int a[50, 100];
for (row = 0; row < 50; ++row) {
    for (col = 0; col < 100; ++col) {
        a[row, col] = 42;
    }
}
```

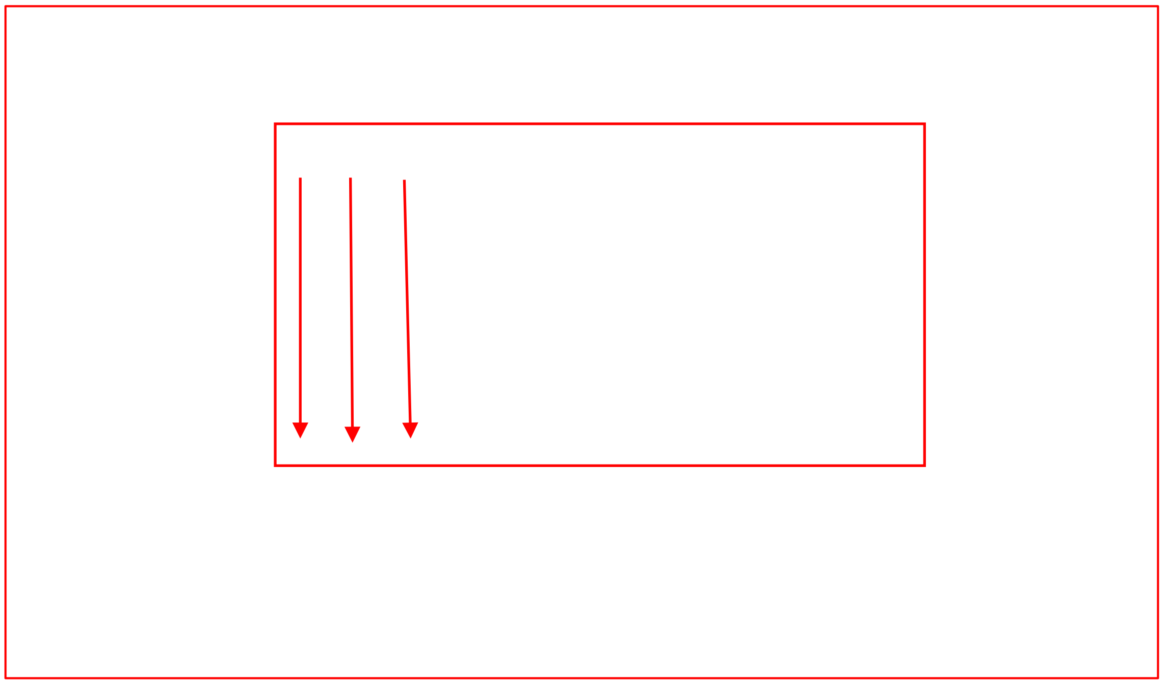
This program seems to run quickly. Why? (include a drawing of the array memory layout to help explain your answer)

The program accesses elements of the array 'a' in the order they are laid out in memory. So one read into a cache line satisfies many successive accesses.



B. 3 points. Write a modified version of this loop that will run much slower, and explain why it will run much slower.

```
int row;
int col;
int a[50, 100];
for (col = 0; col < 100; ++col) {
    for (row = 0; row < 50; ++row) {
        a[row, col] = 42;
    }
}
```



Successive writes to elements of the array will require reading a new cache line. The program does not benefit from caching.

C. 2 points. You have just mimicked the reverse of an optimization that a compiler might perform automatically, on your behalf. What is the name of this particular optimization?

Loop interchange

D. 0 points. Name that book, TV series or movie: why is "42" used so often in computer-science examples?

Hitchhiker's Guide to the Galaxy, by Douglas Adams