

FINAL EXAM

- This test contains 9 questions with a total of 80 points
- 1-3 5 points each
- 4-7 10 points each
- 8 10 points
- 9 15 points
- You have 1 hour and 50 minutes to complete this exam.
- You may use **one** two-sided 8.5" x 11" sheet of notes (or any combination of paper that does not exceed 187 square inches of notes). You must turn in your notes with the test.
- You may **not** use your text, or any other reference material.
- Do not turn this page or turn over the exam until instructed to do so.
- Turn off your cell phones.

Exam Reminders

- **If a question is unclear or you are not sure what I am asking, talk to me during the exam.**
- If I find a mistake during the exam, I will write it on the board (so if you see me writing on the board, look at what I am writing).
- Use additional sheets of paper if you need more room.
- **Don't spend too much time on short answer questions.** If you don't know the answer right away, move on and come back.
- There are some questions that will require more time to think and organize your thoughts. If you find yourself stuck on one of these questions, save it until you have answered all the questions you can answer quickly. If you spend too much time on one question you may get frustrated and that may impair your ability to answer the other questions.
- Don't turn your test in early. If you have time left over, use it to review your answers. Students who turn tests in early often make trivial mistakes that they would catch if they went back over their answers.
- Students that get the highest grades usually go over their answers several times.
- The space below a questions usually indicates how long of an answer I am expected. Sometimes I leave extra room so the next question will start on the next page.
- If your answer does not include anything you learned in this class, maybe you misunderstood the question.

- 1) (5 points) In gpl all statements belong to a statement block. How many statement blocks are created for the following gpl code? List the statement(s) in each block. Number the blocks in terms of creation order. That is: indicate which block is the first block and the statements it contains. Then which block is the second block and the statements it contains and so on. Assume a,b,i are declared integers.

```
on space
{
  if (a < b)
    for (i = 0; i < 10; i += 1)
    {
      print("i = " + i);
    }
}
```

- Block 1: event block for “on space”
contains the if statement
- Block 2: the body block for the if statement in block 1
contains the for statement
- Block 3: the initialization block for the for statement
contains the assignment statement $i = 0$
- Block 4: the increment block for the for statement
contains the assignment statement $i += 1$
- Block 5: the body block for the for statement
contains the print statement

- 2) (5 points) Draw the complete structure your program would create for the following code. Include the assignment statement object.

```
nums[2 * k] = 42;
```

- 3) (5 points) How could local variables be added to gpl event blocks? That is, variables local to the entire event block. Hint: don't panic, I'm looking for an overview not a complete implementation. Think about how to hack gpl to make it work. Don't think of the much more difficult problem of allowing local variables in any block.

The easiest way would be to associate a symbol table with each event block. When a variable is declared inside of an event block it would be inserted into the event block's symbol table.

When a variable is encountered in the event block, first the local symbol table would be searched. If it was not found in the local symbol table the global symbol table would be searched. This could be done by the Variable object. All we would have to do is to insert a pointer to the local symbol table inside of the Variable object. We could store the current local symbol table in a global variable so it would be easily accessible when creating a new Variable object.

4) (10 points) Demonstrate that the following grammar is ambiguous. “ID”, “.”, and “->” are terminal tokens.

$S \rightarrow S . S$
 $S \rightarrow S \rightarrow S$
 $S \rightarrow ID$

Two leftmost or two rightmost derivations for the same input demonstrate the grammar is ambiguous.
The following are two leftmost derivations for the same input.

$S \Rightarrow S.S \Rightarrow S \rightarrow S.S \Rightarrow ID \rightarrow S.S \Rightarrow ID \rightarrow ID.S \Rightarrow ID \rightarrow ID.ID$
 $S \Rightarrow S \rightarrow S \Rightarrow ID \rightarrow S \Rightarrow ID \rightarrow S.S \Rightarrow ID \rightarrow ID.S \Rightarrow ID \rightarrow ID.ID$

- 5) (10 points) LALR parsers can handle both left and right recursion. However, left recursion is more efficient. Remove the **right recursion** from the following grammar. Hint: First you have to figure out how to modify the rule to remove left recursion so it removes right recursion then apply your new rule.

$S \rightarrow abS$
 $S \rightarrow cdS$
 $S \rightarrow e$
 $S \rightarrow f$

The rule for removing left recursion is:

Replace all productions with the following form:

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$$

With the productions:

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_m A'$$
$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_n A' \mid \epsilon$$

This can be converted to a rule for removing right recursion just by swapping the order of the A's and the α 's:

Replace all productions with the following form:

$$A \rightarrow \alpha_1 A \mid \alpha_2 A \mid \dots \mid \alpha_n A \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$$

With the productions:

$$A \rightarrow A' \beta_1 \mid A' \beta_2 \mid \dots \mid A' \beta_m$$
$$A' \rightarrow A' \alpha_1 \mid A' \alpha_2 \mid \dots \mid A' \alpha_n \mid \epsilon$$

Applying this new rule to the given grammar, we get:

$S \rightarrow S'e$
 $S \rightarrow S'f$
 $S' \rightarrow S'ab$
 $S' \rightarrow S'cd$
 $S' \rightarrow \epsilon$

6) (10 points) Consider the language that contains n a's followed by $2*n$ b's followed by $3*n$ c's where n is greater than zero:

```
abbccc
aabbbbcccccc
aaabbbbbccccccccc
...
```

It is not possible to create a context free grammar for this language. However, it is possible to use bison to write a parser that can recognize if the input string is in the language or not in the language. Write bison pseudo-code for this parser (the pseudo-code must contain the grammar).

The trick is to use an action to check if there are an allowable number of a's b's and c's.

```
program:
  a_string b_string c_string
  {
    if ($1*2 == $2 && $1*3 == $3)
      cout << "yes" << endl;
    else cout << "no" << endl;
  }
;

a_string:
  T_A
  {
    $$ = 1;
  }
  |
  T_A a_string
  {
    $$ = 1 + $2;
  }
;

b_string:
  similar to a_string

c_string:
  similar to a_string
```

Alternative: you could also use global variables

7) (10 points) Using the given grammar and parse table, fill in the Stack/Input/Action table with the moves an LR parser would take for the given input. The table contains enough rows. You may not need all the rows.

- (1) $E \rightarrow E + T$
- (2) $E \rightarrow T$
- (3) $T \rightarrow T * F$
- (4) $T \rightarrow F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow id$

State	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Stack	Input	Action
0	id + id) * (id + id) \$	s5
0 id 5	+ id) * (id + id) \$	r6
0 F 3	+ id) * (id + id) \$	r4
0 T 2	+ id) * (id + id) \$	r2
0 E 1	+ id) * (id + id) \$	s6
0 E 1 + 6	id) * (id + id) \$	s5
0 E 1 + 6 id 5) * (id + id) \$	r6
0 E 1 + 6 F 3) * (id + id) \$	r4
0 E 1 + 6 T 9) * (id + id) \$	r1
0 E 1) * (id + id) \$	error

8) (10 points) Show that the following grammar is LALR but not SLR. When building the collection of items, always consider the grammar symbols in this order: {S A B d f g h}. You may not need all the states in the given tables. Only fill in the part of the SLR table necessary to demonstrate the grammar is not SLR (you must explain your answer). Fill in the complete LALR(1) action and goto tables.

- (0) $S' \rightarrow S$
- (1) $S \rightarrow Ad$
- (2) $S \rightarrow Bh$
- (3) $S \rightarrow gBd$
- (4) $S \rightarrow gAh$
- (5) $A \rightarrow f$
- (6) $B \rightarrow f$

State	d	f	g	h	\$	S	A	B
0								
1								
2								
3								
4	r5/r6			r5/r6				
5								
6								
7								
8								
9								
10								
11								
12								

First show the grammar is not SLR(1)
 Calculate the set of LR(0) items (below).
 Fill in the table.

Follow(S) = {\$}
 Follow(A) = {d,h}
 Follow(B) = {d,h}

For states [4,d] and [4,h] we reduce with rule 5 and 6 on Follow(A) and Follow(B)

Thus we have the two reduce/reduce conflicts

Thus this grammar is NOT SLR

I0: $S' \rightarrow \cdot S$
 $S \rightarrow \cdot Ad$
 $S \rightarrow \cdot Bh$
 $S \rightarrow \cdot gBd$
 $S \rightarrow \cdot gAh$
 $A \rightarrow \cdot f$
 $B \rightarrow \cdot f$

I3: $S \rightarrow B \cdot h$

I7: $S \rightarrow Bh \cdot$

I4: $A \rightarrow f \cdot$
 $B \rightarrow f \cdot$

I8: $S \rightarrow gA \cdot h$

I5: $S \rightarrow g \cdot Bd$
 $S \rightarrow g \cdot Ah$
 $A \rightarrow \cdot f$
 $B \rightarrow \cdot f$

I9: $S \rightarrow gB \cdot d$

I10: $S \rightarrow gAh \cdot$

I1: $S' \rightarrow S \cdot$

I11: $S \rightarrow gBd \cdot$

I2: $S \rightarrow A \cdot d$

I6: $S \rightarrow Ad \cdot$

9) (15 points) Show all the items, fill in table.

- (0) $S' \rightarrow S$
- (1) $S \rightarrow Ad$
- (2) $S \rightarrow Bh$
- (3) $S \rightarrow gBd$
- (4) $S \rightarrow gAh$
- (5) $A \rightarrow f$
- (6) $B \rightarrow f$

State	d	f	g	h	\$	S	A	B
0		s4	s5			1	2	3
1					acc			
2	s6							
3				s7				
4	r5			r6				
5		s10					8	9
6					r1			
7					r2			
8				s11				
9	s12							
10	r6			r5				
11					r4			
12					r3			

No conflicts in table.
Grammar is LALR

- I0: $S' \rightarrow \cdot S, \$$
- $S \rightarrow \cdot Ad, \$$
- $S \rightarrow \cdot Bh, \$$
- $S \rightarrow \cdot gBd, \$$
- $S \rightarrow \cdot gAh, \$$
- $A \rightarrow \cdot f, d$
- $B \rightarrow \cdot f, h$

I1: $S' \rightarrow S \cdot, \$$

I2: $S \rightarrow A \cdot d, \$$

I3: $S \rightarrow B \cdot h, \$$

I4: $A \rightarrow f \cdot, d$
 $B \rightarrow f \cdot, h$

I5: $S \rightarrow g \cdot Bd, \$$
 $S \rightarrow g \cdot Ah, \$$
 $A \rightarrow \cdot f, h$
 $B \rightarrow \cdot f, d$

I6: $S \rightarrow Ad \cdot, \$$

I7: $S \rightarrow Bh \cdot, \$$

I8: $S \rightarrow gA \cdot h, \$$

I9: $S \rightarrow gB \cdot d, \$$

I10: $A \rightarrow f \cdot, h$
 $B \rightarrow f \cdot, d$

I11: $S \rightarrow gAh \cdot, \$$

I12: $S \rightarrow gBd \cdot, \$$