

FINAL EXAM

- This test contains 9 questions with a total of 85 points
 - 1-4 5 points each
 - 5-8 10 points each
 - 9 25 point
- You have 1 hour and 50 minutes to complete this exam.
- You may use **one** two-sided (or two one sided) 8.5" x 11" sheet 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.

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) List the three types of analysis performed by a compiler. Give an example of the type of error that can be found by each.

Linear/Lexical	Error: identifiers/symbols not in the language (@_)
Hierarchical/Syntactic	Error: input that does not match a grammar rule (foo{"hello"})
Semantic:	Error: type errors (i = 42; i = "hello world")

Note: examples assume the language is C-like. Of course, most of this is legal in Perl.

- 2) (5 points) Explain the mechanism used in the gpl interpreter for making animation blocks use the Game_object that should be animated (the actual parameter) instead of the Game_object declared in the forward statement (the formal parameter).

```
forward animation bounce(rectangle cur_rec);

rectangle my_rec(animation_block = bounce);

animation bounce(rectangle cur_rec)
{
    cur_rec.x = 42;
}
```

In other word, how is it that my_rec.x is changed to 42 instead of cur_rect.x.

The code in the animation block is parsed like normal code (like that in initialization and event handler blocks). However, before the statements in the animation block are executed, the symbol for cur_rec is modified so it points to the Game_object my_rect. Thus when the code is executed it is as if the code were:

```
my_rect.x = 42
```

This is achieved by associating a pointer to cur_rect with the block of code (Animation_block) and passing a pointer to the actual Game_object (my_rect) to the Animation_block::execute(Game_object *actual)

3) (5 points) Explain everything that happens in the action for the following gpl production. Describe the mechanism which eventually executes this block. If you don't know the entire mechanism, describe what you do know.

on_block: T_ON keystroke statement_block

A pointer to a Statement_block is associated with statement_block (\$3). A keystroke (e.g. SPACE, AKEY, etc) is associated with keystroke (\$2).

Consider the following example:

```
on space
{
    x = 42;
    y = 86;
}
```

When the space key is pressed, the code (x = 42 and y = 86) must be executed. In other words, the block of code (\$3) that contains these statements must be told to execute.

To achieve this, the block (\$3) is passed to the Event_manager along with the key (SPACE). The Event_manager stores this <key, statement_block> pair.

When the key is pressed (SPACE in this example), GLUT calls a callback function in class Window. This function tells the Event_manager to execute all the statement_blocks associated with key (SPACE).

The Event_manager looks up the key and executes all statement_blocks associated with it.

4) (5 points) Describe the language defined by the following grammar. Is this grammar ambiguous?

$E \rightarrow EEO \mid id$

$O \rightarrow + \mid - \mid * \mid /$

Post-fix expressions with the binary operators +, -, *, /

This grammar is not ambiguous.

5) (10 points) Create a context free grammar (CFG) for the following language. If it is not possible to create a CFG explain why.

$a^n b^+ c^n$ n a's followed by one or more b's followed by n c's where $n > 0$

Example string: "abc" "abbc" "aabcc" "aabbcc"

$S \rightarrow aSc \mid aBc$

$B \rightarrow bB \mid b$

6) (10 points) Consider the following productions for an else statement:

if_statement:

```
T_IF T_LPAREN expression T_RPAREN if_block  
| T_IF T_LPAREN expression T_RPAREN if_block T_ELSE if_block
```

When this rule is parsed using the LALR algorithm there is a problem. What is that problem? Describe what it is about these production that causes this problem? It may help to describe exactly what happens when input is parsed.

These productions result in a shift reduce error. Consider the following input:

```
if (a)  
    if (b)  
        x();  
else y();
```

The problem is that the else clause could belong to either if statement. When the parser gets to the end of the first statement block (the “x();”) it can't tell if it should reduce on the first production or shift the else onto the stack so it can later reduce on the second production.

Bison and YACC resolve this problem by always performing a shift when there is a shift/reduce conflict. In this example the shift is what we want because an else clause is always bound to the closest if.

7) (10 points) Eliminate the left-recursion and then left factor the following grammar

$$\begin{aligned} S &\rightarrow ScB \mid B \\ B &\rightarrow e \mid efg \mid efCg \\ C &\rightarrow SdC \mid S \end{aligned}$$

eliminate left-recursion

$$\begin{aligned} S &\rightarrow BS' \\ S' &\rightarrow cBS' \mid \varepsilon \\ B &\rightarrow e \mid efg \mid efCg \\ C &\rightarrow SdC \mid S \end{aligned}$$

left factor

$$\begin{aligned} S &\rightarrow BS' \\ S' &\rightarrow cBS' \mid \varepsilon \\ B &\rightarrow eB' \\ B' &\rightarrow \varepsilon \mid fB'' \\ B'' &\rightarrow g \mid Cg \\ C &\rightarrow SC' \\ C' &\rightarrow dC \mid \varepsilon \end{aligned}$$

8) (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) \$	s4
0 (4	id * id) (id + id) \$	s5
0 (4 id 5	* id) (id + id) \$	r6 F --> id
0 (4 F 3	* id) (id + id) \$	r4 T --> F
0 (4 T 2	* id) (id + id) \$	s7
0 (4 T 2 * 7	id) (id + id) \$	s5
0 (4 T 2 * 7 id 5) (id + id) \$	r6 F --> id
0 (4 T 2 * 7 F 10) (id + id) \$	r3 T --> T * F
0 (4 T 2) (id + id) \$	r2 E --> T
0 (4 E 8) (id + id) \$	s11
0 (4 E 8) 11	(id + id) \$	error

9) (25 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 a b c d}. You may not need all the states in the given tables. Only fill in the part of the SLR tables 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) $S \rightarrow Aa$
- (2) $S \rightarrow bAc$
- (3) $S \rightarrow Bc$
- (4) $S \rightarrow bBa$
- (5) $A \rightarrow d$
- (6) $B \rightarrow d$

State	a	b	c	d	\$	S	A	B
0								
1								
2								
3								
4								
5	r5/r6		r5/r6					
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) = {a,c}
- Follow(B) = {a,c}

For states [5,a], [5,c], [10,a] and [10,c] 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: $S' \rightarrow . S$
- $S \rightarrow . Aa$
- $S \rightarrow . bAc$
- $S \rightarrow . Bc$
- $S \rightarrow . bBa$
- $A \rightarrow . d$
- $B \rightarrow . d$

- I4: $S \rightarrow b . Ac$
- $S \rightarrow b . Ba$
- $A \rightarrow . d$
- $B \rightarrow . d$
- I5: $A \rightarrow d .$
- $B \rightarrow d .$

- I8: $S \rightarrow bA . c$
- I9: $S \rightarrow bB . a$
- I10: $S \rightarrow bAc .$
- I11: $S \rightarrow bBa .$

I1: $S' \rightarrow S .$

I6: $S \rightarrow Aa .$

I2: $S \rightarrow Aa .$

I7: $S \rightarrow Bc .$

I3: $S \rightarrow Bc .$

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

- (7) $S' \rightarrow S$
- (8) $S \rightarrow Aa$
- (9) $S \rightarrow bAc$
- (10) $S \rightarrow Bc$
- (11) $S \rightarrow bBa$
- (12) $A \rightarrow d$
- (13) $B \rightarrow d$

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

No conflicts in table.
Grammar is LALR

I0: $S' \rightarrow \cdot S, \$$
 $S \rightarrow \cdot Aa, \$$
 $S \rightarrow \cdot BAc, \$$
 $S \rightarrow \cdot Bc, \$$
 $S \rightarrow \cdot bBa, \$$
 $A \rightarrow \cdot d, a$
 $B \rightarrow \cdot d, c$

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

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

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

I4: $S \rightarrow b \cdot Ac, \$$
 $S \rightarrow b \cdot Ba, \$$
 $A \rightarrow \cdot d, c$
 $B \rightarrow \cdot d, a$

I5: $A \rightarrow d \cdot, a$
 $B \rightarrow d \cdot, c$

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

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

I8: $S \rightarrow bA \cdot C, \$$

I9: $S \rightarrow bB \cdot a, \$$

I10: $A \rightarrow d \cdot, c$
 $B \rightarrow d \cdot, a$

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

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