page 1 page 2 page 3 Total / 32

*Please print clearly :*

**Name :** SOLUTION

**Login :** @ucsc.edu

***No books ; No calculator ; No computer ; No email ; No internet ; No notes ; No phone. Neatness counts ! Do your scratch work elsewhere and enter only your final answer into the spaces provided.***

1. Define a lexical grammar using **flex**. use the function **yylval_token** where appropriate. **[4✔]**
   (a) An **IDENT** uses exactly the same syntax as C.
   (b) A **NUMBER** uses the same syntax as a C decimal, octal, or hexadecimal number.
   (c) Single-character tokens needed by the grammar of the next question are returned.
   (d) Comments and white space are discarded. Comments start with a double slash (//) and continue to the end of a line. White space consists of spaces, tabs, and newlines.
   (e) Call the function **error** for any invalid input character.

```
IDENT [A-Za-z_][A-Za-z_0-9]*
NUMBER [1-9][0-9]*|0[0-7]*|0[Xx][0-9a-fA-F]+
%%
{IDENT}          { return yylval_token (IDENT); }
{NUMBER}         { return yylval_token (NUMBER); }
"//".*           { }
[ \t\n]+         { }
";"              { return yylval_token (';'); }
"+"              { return yylval_token ('+'); }
"-"              { return yylval_token ('-'); }
"("              { return yylval_token ('('); }
"/"              { return yylval_token ('/'); }
"("              { return yylval_token ('('); }
")"              { return yylval_token (')'); }
","              { return yylval_token (','); }
.                { error(); }
%%
```

2. Define a grammar using **bison**. Assume the functions **adopt1** and **adopt2** from the project. Build the abstract syntax tree. **[6✔]**
   (a) A program is a sequence of zero or more exprs separated by semi-colons.
   (b) An expr is an **IDENT** ; or a **NUMBER** ; or two exprs connected by one of the operators **'+'**, **'-'**, **'*'**, **'/'**, using the same precedence and associativity as C ; or an expr in parentheses ; or a call.
   (c) A call is an **IDENT** followed by a pair of parentheses which contain zero or more exprs separated by commas.

```
%token NUMBER IDENT
%left '+' '-'
%left '*' '/'
%start program
%%
program  : sequence              { parse_root = $1; }
         | root                  { parse_root = $1; }
         ;
root     :                       { $$ = new_root(); }
         ;
sequence : sequence ';' expr     { free ($2); $$ = adopt ($1, $3); }
         | root expr             { $$ = adopt1 ($1, $2); }
         ;
expr     : IDENT                 { $$ = $1; }
         | NUMBER                { $$ = $1; }
         | expr '+' expr         { $$ = adopt2 ($2, $1, $3); }
         | expr '-' expr         { $$ = adopt2 ($2, $1, $3); }
         | expr '*' expr         { $$ = adopt2 ($2, $1, $3); }
         | expr '/' expr         { $$ = adopt2 ($2, $1, $3); }
         | '(' expr ')'          { free($1); free($2); $$ = $2; }
         | call                  { $$ = $1; }
         ;
call     : IDENT '(' ')'         { free($3); $$ = adopt1 ($2, $1);
         | callargs ')'          { free($2); $$ = $1; }
         ;
callargs : IDENT '(' expr        { $$ = adopt2 ($2, $1, $3); }
         | callargs ',' expr     { free($2); $$ = adopt1 ($1, $3); }
         ;
```

3. Write a grammar using `bison` which will recognize a stream of balanced parentheses. The only input tokens to this program are left and right parentheses. All other characters are screened out by the scanner. Examples of balanced parentheses are shown here. In other words, every left parenthesis must be matched up later in the stream with a corresponding right parenthesis. **[2✓]**

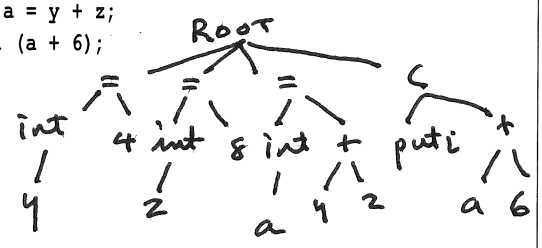
```
(((((()))))
()()()()()
((()))((()))
(())((()))()()
```

```
%%
p : '(' p ')' p
  |
  ;
%%
```

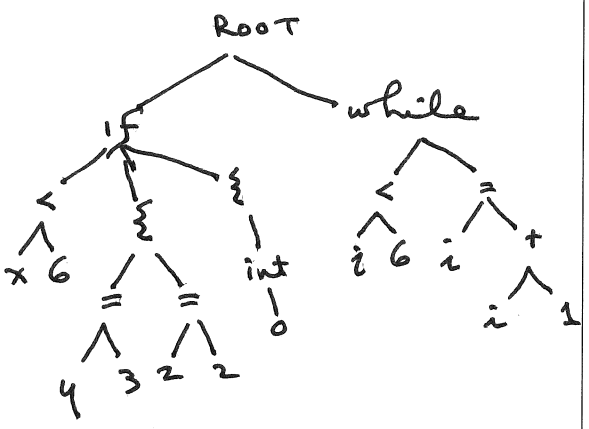
```
%start p
```

4. According to the specifications of project 2, draw abstract syntax trees for the following. Show the root at the top and draw lines between parent and child, but not between siblings. **[4✓]**

```
int y = 4;
int z = 8;
int a = y + z;
puti (a + 6);
```

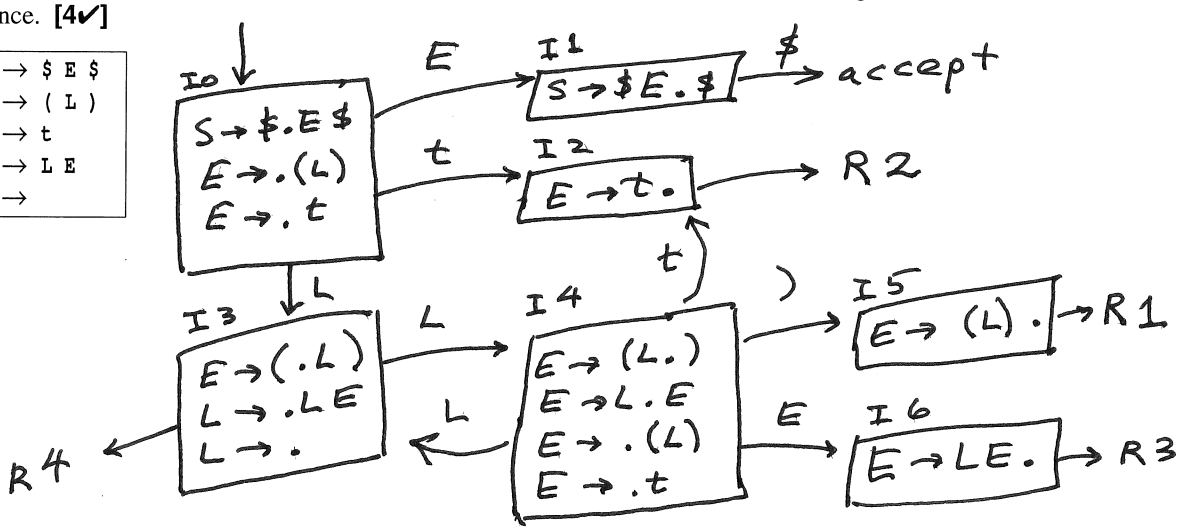



```
if (x < 6) { y = 3; z = 2; } else { int o; }
while (i < 6) i = i + 1;
```




5. Given the grammar presented here, and using the style of the handout, use LR(0) analysis to construct the characteristic finite state machine (CSFM), sets of items and transition diagram, showing shifts, reductions, and acceptance. **[4✓]**

| | |
|---|---|
| 0. | S → $ E $ |
| 1. | E → ( L ) |
| 2. | E → t |
| 3. | L → L E |
| 4. | L → |



Multiple choice. To the ***left*** of each question, write the letter that indicates your answer. Write **Z** if you don't want to risk a wrong answer. Wrong answers are worth negative points. **[12✔]**

| | | | |
|---|---|---|---|
| number of correct answers | | × 1 = | $= a$ |
| number of wrong answers | | × ½ = | $= b$ |
| number of missing answers | | × 0 = | 0 |
| column total $c = \max(a-b, 0)$ | 12 | | $= c$ |

1. What grammar analysis method is used by `bison`?
   (A) LALR(1)
   (B) LR(0)
   (C) recursive descent
   (D) regular

A

2. Which of the following is ambiguous?
   (A) $E \rightarrow E + E$
   (B) $E \rightarrow E + T$
   (C) $E \rightarrow T + E$
   (D) $E \rightarrow T + T$

A

3. How many 64-bit registers are available with the x86_64 architecture?
   (A) 8
   (B) 12
   (C) 16
   (D) 32

C

4. For a grammar $G = \langle V_N, V_T, P, S \rangle$, the set $P$ contains rules of the form $(A \rightarrow \beta)$, where
   (A) $A \in V_N$ and $\beta \in (V_N \cap V_T)^*$
   (B) $A \in V_N$ and $\beta \in (V_N \cup V_T)^*$
   (C) $A \in V_T$ and $\beta \in (V_N \cap V_T)^+$
   (D) $A \in V_T$ and $\beta \in (V_N \cup V_T)^+$

B

5. We should shift if the precedence of the rule on the stack is [x] than the lookahead symbol, or if the precedences are the same, and they are [y] associative.
   (A) [x] = higher, [y] = left
   (B) [x] = higher, [y] = right
   (C) [x] = lower, [y] = left
   (D) [x] = lower, [y] = right

D

6. What parsing action pops the right side of a rule off the stack, passes it to a semantic action, then pushes the left side of that rule (along with `$$`) onto the stack?
   (A) accept
   (B) error
   (C) reduce
   (D) shift

C

7. Which item was added during a closure operation on a state in a CFSM?
   (A) $E \rightarrow \bullet\, E + T$
   (B) $E \rightarrow E \bullet + T$
   (C) $E \rightarrow E + \bullet\, T$
   (D) $E \rightarrow E + T \bullet$

A

8. What input is acceptable to the following grammar?
   $A \rightarrow x\ A\ y$
   $A \rightarrow$
   (A) xxxxxxxxyyyyyyyy
   (B) xxxxyyyyxxxxyyyy
   (C) xyxyxyxyxyxyxyxy
   (D) yyyyyyyyxxxxxxxx

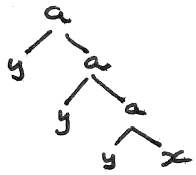
A

9. Thompson's construction:
   (A) constructs an NFA from a regular expression.
   (B) converts an NFA into a DFA.
   (C) minimizes a DFA.
   (D) produces a CFSM from a context-free grammar.

A

10. Which `bison` grammar matches any number of `'y'`s followed by exactly one `'x'`? Which `bison` grammar matches any number of `'y'`s followed by exactly one `'x'`?
    (A) `a : 'x' a | 'y' ;`
    (B) `a : 'y' a | 'x' ;`
    (C) `a : a 'x' | 'y' ;`
    (D) `a : a 'y' | 'x' ;`

B




11. Which function can change the size of the heap?
    (A) `brk`(2)
    (B) `free`(3)
    (C) `heap`(1)
    (D) `malloc`(3)

A

12. Which statement in a `flex` action will generate the message "warning: return makes integer from pointer without a cast".
    (A) `"a" { return "="; }`
    (B) `"a" { return '='; }`
    (C) `"a" { return *yytext; }`
    (D) `"a" { return 2015; }`

A