NAME: _____ ID: _____

CS444 grammar lesson nov 14th:

Prompt for this page:
show tokens (numbered, one line per each listing basic type (keyword, identifier, constant, string literal, operator, other separator), Token_type and lexeme, C grammar derivation and abstract syntax tree for simplified nov12.c which is: int main(int argc, char *argv[], char *envp[]){ int k; k = 0; while (envp[k] != 0) { k++; } return 0; }

Answer: lexical analysis (tokens), C grammar derivation, abstract syntax tree
1. Lexical Analysis (Tokens): 44 tokens according to standard C lexical info

#	Basic Type	Token_type	Lexeme
1	Keyword	INT	int
2	Identifier	IDENTIFIER	main
3	Other separator	LPAREN	(
4	Keyword	INT	int
5	Identifier	IDENTIFIER	argc
6	Other separator	COMMA	,
7	Keyword	CHAR	char
8	Operator	ASTERISK (or STAR)	*
9	Identifier	IDENTIFIER	argv
10	Other separator	LBRACKET	[
11	Other separator	RBRACKET	]
12	Other separator	COMMA	,
13	Keyword	CHAR	char
14	Operator	ASTERISK (or STAR)	*
15	Identifier	IDENTIFIER	envp
16	Other separator	LBRACKET	[
17	Other separator	RBRACKET	]
18	Other separator	RPAREN	)
19	Other separator	LBRACE	{
20	Keyword	INT	int
21	Identifier	IDENTIFIER	k
22	Other separator	SEMICOLON	;
23	Identifier	IDENTIFIER	k
24	Operator	ASSIGN	=
25	Constant	INT_LITERAL	0
26	Other separator	SEMICOLON	;
27	Keyword	WHILE	while
28	Other separator	LPAREN	(
29	Identifier	IDENTIFIER	envp
30	Other separator	LBRACKET	[
31	Identifier	IDENTIFIER	k
32	Other separator	RBRACKET	]
33	Operator	NEQ	!=
34	Constant	INT_LITERAL	0
35	Other separator	RPAREN	)
36	Other separator	LBRACE	{
37	Identifier	IDENTIFIER	k
38	Operator	INC	++
39	Other separator	SEMICOLON	;
40	Other separator	RBRACE	}
41	Keyword	RETURN	return
42	Constant	INT_LITERAL	0
43	Other separator	SEMICOLON	;
44	Other separator	RBRACE	}

2. C Grammar Derivation (Leftmost)
Condensed leftmost derivation from <translation-unit> to the terminal tokens:

```
<translation-unit>
 => <external-declaration>
 => <function-definition>
 => <declaration-specifiers> <declarator> <compound-statement>
 => int <direct-declarator> <compound-statement>
 => int <direct-declarator> ( <parameter-type-list> ) <compound-statement>
 => int main ( <parameter-list> ) <compound-statement>
 => int main ( <parameter-declaration> , <parameter-declaration> , <parameter-
declaration> ) <compound-statement>
 => int main ( int argc , char *argv[] , char *envp[] ) { <block-item-list> }
 => int main ( int argc , char *argv[] , char *envp[] ) { <declaration>
<statement> <statement> <statement> }

 --- Expanding <declaration> (int k;) ---
 => int <init-declarator-list> ;
 => int k ;

 --- Expanding 1st <statement> (k = 0;) ---
 => <expression-statement>
 => <assignment-expression> ;
 => <unary-expression> = <assignment-expression> ;
 => k = 0 ;

 --- Expanding 2nd <statement> (while (envp[k] != 0) { k++; }) ---
 => <iteration-statement>
 => while ( <expression> ) <statement>
 => while ( <equality-expression> ) <compound-statement>
 => while ( <postfix-expression> != <relational-expression> ) { <block-item-
list> }
 => while ( envp [ k ] != 0 ) { <expression-statement> }
 => while ( envp [ k ] != 0 ) { <postfix-expression> ++ ; }
 => while ( envp [ k ] != 0 ) { k ++ ; }

 --- Expanding 3rd <statement> (return 0;) ---
 => <jump-statement>
 => return <expression> ;
 => return 0 ;

3. Abstract Syntax Tree (AST)
Unlike a parse tree, AST strips away syntactic punctuation (;, (, ), {, }) and
represents the hierarchical structure of program's execution and declarations.

TranslationUnit
 └── FunctionDecl: main (return_type: int)
      ├── Parameters
      │    ├── ParamDecl: argc (type: int)
      │    ├── ParamDecl: argv (type: char*[])
      │    └── ParamDecl: envp (type: char*[])
      └── CompoundStmt (Function Body)
           ├── VarDecl: k (type: int)
           ├── BinaryOperator: '=' (Assignment)
           │    ├── DeclRefExpr: k
           │    └── IntegerLiteral: 0
           ├── WhileStmt
           │    ├── Condition: BinaryOperator '!='
           │    │    ├── ArraySubscriptExpr
           │    │    │    ├── DeclRefExpr: envp
           │    │    │    └── DeclRefExpr: k
           │    │    └── IntegerLiteral: 0
           │    └── Body: CompoundStmt
           │         └── UnaryOperator: '++' (Postfix)
           │              └── DeclRefExpr: k
           └── ReturnStmt
                └── IntegerLiteral: 0
```