Compilers and Languages

Project Document

Presented By:	Ahmed Tarek Fahmy Elwakil	1114697
	Ahmed Hany Shorim	1112796
	Mustafa Mohamed Barakat	1112293
	Ahmed Hatem Zahran	1112689
Presented To:	Dr. Mona Farouk	
	Eng. Sarah Rashad	

Submitted: May 11, 2015

Project Overview

The project is an implementation of a simple programming language (mini C) using the LEX and YACC compiler generating package. The new language syntax is very similar to the C language and consists of the following:

- Variables and Constants declaration.
- Mathematical and logical expressions.
- Assignment statement.
- If-then-else statement, while loops, repeat-until loops, for loops, switch statement.
- Block structure (nested scopes where variables may be declared at the beginning of blocks).
- Scopes

After building the language, there is a designed program that takes our language file (.exe), your code file (.c) and the output text file of the generated code (.txt) which contains the Assembly code of the original C code.

Technologies Used

1. Cygwin (Unix Command Line Interface)
2. Flex (Lexical Analyzer)
3. Bison (Yacc Compatible Parser Generator)
4. Microsoft Visual Studio
5. Microsoft C#
6. Adobe Photoshop CC
7. Notepad++

Tokens List

Token Name	**Token Description**
ENDPROGRAM	A word written at the end of the file being compiled to be used as an end marker to the code
WHILE	“while” reserved word for while loops
UNTIL	“until” reserved word for repeat until loops
REPEAT	“repeat” reserved word for repeat until loops
BREAK	“break” reserved word for breaking switch cases
IF	“if” reserved word for if conditions
THEN	“then” reserved word for starting the if statement code block
ELSE	“else” reserved word for starting the else code block of the if statement
ENDIF	“endif” reserved word for ending the if statement
FOR	“for” reserved word for for loops
SWITCH	“switch” reserved word for switch case statements
CASE	“case” reserved word for the case statements inside the switch case statement
TYPE_INT	“int” reserved data type
TYPE_FLOAT	“float” reserved data type
TYPE_CHAR	“char” reserved char type
CONST	“const” reserved word for defining constants
VARIABLE	Variable names consisting of one character
FLOAT	Float numbers
INT	Integer numbers
CHAR	Character
INC	Incrementing variables “++”
GE	Greater than or equal
LE	Less than or equal
EQ	Equality condition
NE	Not equal

Production Rules

Production Rules	**Description**
Program:	
Function	The start of the parser
Function:	
Function stmt	Start the code with a statement
NULL	If there is nothing left in the code
Stmt:	
CONST TYPE_INT VARIABLE '=' expr ';'	Declaring and Initializing a constant integer with value
CONST TYPE_FLOAT VARIABLE '=' expr ';'	Declaring and Initializing a constant float with value
CONST TYPE_CHAR VARIABLE '=' expr ';'	Declaring and Initializing a constant char with value
TYPE_INT VARIABLE '=' expr ';'	Declaring and Declaring and Initializing an integer with value
TYPE_FLOAT VARIABLE '=' expr ';'	Declaring and Initializing a float with value
TYPE_CHAR VARIABLE '=' expr ';'	Declaring and Initializing a char with value
VARIABLE '=' expr ';'	Changing value of a variable ex: x=5;
WHILE '(' logicExpr ')' '{' bracket_stmt_list	While loop
IF '(' logicExpr ')' THEN stmt_list ENDIF	If…then.. statement
IF '(' logicExpr ')' THEN stmt_list ELSE stmt_list ENDIF	If…then….else statement
REPEAT '{' bracket_repeat_list	Repeat statement
SWITCH '(' VARIABLE ')' '{' case_stmts '}'	Switch case
FOR '(' loop_stmt1 ';' logicExpr ';' loop_stmt2 ')' '{' stmt_list '}'	For loop
'{'	Opening a scope
Stmt_list	For multiple statements
'}'	Closing a scope
ENDPROGRAM	Added by our GUI to indicate the end point of the program.
bracket_repeat_list:	
'}' UNTIL '(' logicExpr ')'	The last part of the repeat until statement
Stmt	For a statement inside the repeat until
stmt_list stmt	For multiple statements inside the repeat until
bracket_stmt_list:	
Stmt	For a statement inside a scope
stmt_list stmt	For multiple statements inside a scope
loop_stmt1:	
TYPE_INT VARIABLE '=' expr	Defining the for loop variable
loop_stmt2:	
VARIABLE '=' expr	The statement to be executed on the for loop variable at the end of each iteration
VARIABLE INC	Incrementing the for loop variable at the end of each iteration
stmt_list:	
Stmt	For a single statement inside the code file
stmt_list stmt	For multiple statements inside the code file

case_stmt:	
CASE INT ':' stmt_list BREAK ';'	Defining the case statement inside the switch case statement
case_stmts:	
case_stmt	For a single case statement inside the switch case statement
case_stmts case_stmt	For multiple case statements inside the switch case statement
logicExpr:	
expr '<' expr	Comparing smaller than between two expressions
expr '>' expr	Comparing bigger than between two expressions
expr GE expr	Comparing bigger than or equal between two expressions
expr LE expr	Comparing smaller than or equal between two expressions
expr NE expr	Checking if two expressions are not equal
expr EQ expr	Checking if two expressions are equal
'(' logicExpr ')'	Putting logical expressions between brackets
Expr:	
INT	Integer Value
FLOAT	Float Value
CHAR	Character Value
'-' expr %prec UMINUS	Negation
expr '+' expr	Evaluating sum between two expressions
expr '-' expr	Evaluating subtraction between two expressions
expr '*' expr	Evaluating product of two expressions
expr '/' expr	Evaluating division of two expressions
'(' expr ')'	An expression between two brackets

Quadruples List

Quadruple	**Description**
Push x	Pushes the value of x into the stack
Push 8	Pushes 8 into the stack
Pop x	Pops a value into the x
L001:	Label L001
jmpTrue L001	Jump to label L001 if the comparison done before this jump is true
jmpFalse L001	Jump to label L001 if the comparison done before this jump is false
Jmp L001	Jump to label L001
Neg x	Negate the value of x
Add	Adds the last 2 values pushed into the stack then pushes the result to the stack
Sub	subtracts the last 2 values pushed into the stack then pushes the result to the stack
Mul	Multiplies the last 2 values pushed into the stack then pushes the result to the stack
div	Divides the last 2 values pushed into the stack then pushes the result to the stack
compLT	Checks if the value/variable that was pushed into the stack before the last is less than the last value/variable pushed into the stack
compGT	Checks if the value/variable that was pushed into the stack before the last is greater than the last value/variable pushed into the stack
compGE	Checks if the value/variable that was pushed into the stack before the last is greater than or equal to the last value/variable pushed into the stack
compLE	Checks if the value/variable that was pushed into the stack before the last is less than or equal to the last value/variable pushed into the stack
compNE	Checks if the value/variable that was pushed into the stack before the last is not equal to the last value/variable pushed into the stack
compEQ	Checks if the value/variable that was pushed into the stack before the last is equal to the last value/variable pushed into the stack