Faculty of Engineering

"System Programming"

Presented to:

Dr.Mohamed El-Khouly

Presented By:

Veronia Osama Ragaey

Compiler

Brief definition:

A compiler is a program that reads a program written in one language –the source language and translates it into an equivalent program in another language-the target language. The compiler reports to its user the presence of errors in the source program and it consist of three main stages: Lexical analyzer, Parsing and Code generation.

Lexical analyzer:

Is the first phase of compiler also known as scanner.

It converts the input program into a sequence of Tokens.

A token is a sequence of characters that can be treated as a unit in the grammar of the programming languages and a token may be keyword, operator, identifier, or delimiter.

Parse tree:

A parse tree is an ordered tree that represents the structure of a set of tokens.

The parser uses tokens and grammar to construct parse tree.

Code generation

• As each statement (included in the grammar) is recognized, a code-generation routine is called to create the corresponds object code.

• Code generator scans the statements from the parse tree and generate the code according to the routines of each grammar rule.

Grammar of the Project

<Program> :: = <statement> | <program> <statement>

<statement> ::= <assignstmt> | <ifstmt>

<assignstmt> ::= <id> = <expr> ; | <id> = <id> ; | <id> = <int> ; |
int <id> = <expr> ; | int <id> = <id> ; |
int <id> = <int> ;

<ifstmt> ::= if (<condition>) { <assign> } ;

<assign> ::= <id> = <expr> | <id> = <id> | <id> = <int>

<expr> ::= <id> + <id> | <id> + <int> | <int> + <id> | <int> + <int> |
<id> * <id> | <id> * <int> |<int> * <id> | <int> * <int>

<condition> ::= <id> == <id> | <id> == <int>

<id> ::= a | b | c | …… | z | A | B | C | …… | Z

<int> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Explanation of the Program

Firstly, we have two small function the first one is:

```
void del(string arr[], int rep) -> This function is for
deleting the array to reallocate in it again
```

The second one:

```
string loop(string arr) -> This function is for taking a
string a number in digits and return number in words as a
string. Ex: 7 to be SEVEN
```

Secondly, we enter the Lexical analyzer part this part consists of the file that we read our program from it `(myfile.open("C:\\Users\\Compu fast\\Desktop\\prog.txt"); )` as the lexical analyzer takes the program as lexemes and through the pattern it came out as a token so we have our dictionary `(mix.open("C:\\Users\\Compu fast\\Desktop\\mix.txt");)` that contains all the words approved by our compiler to go through them and the compiler produced a table with all the tokens with repetition`(lexical_analysis.open("C:\\Users\\Compufast\\Desktop\\lexical_analysis .txt");)` and the it produce the symbol table without any repetition `(symbol.open("C:\\Users\\Compu fast\\Desktop\\symbol.txt");)` if the lexical analyzer didn't found any error the compiler will pass this stage and go to the parser but if it found any error an error message will show up

Thirdly, if we passed from the lexical analyzer safely without any errors we will go through the parser stage the parser take the output from the lexical analyzer and take the tokens in an array till it find a semicolon **";** "so we will check if the content of the array approved by the grammar if it is correct, we will pass to the code generation phase if not an error will show up

Finally, the code generation phase it the phase that comes up after checking on the spelling by the lexical analyzer and checking on the syntax by the parser and format the parse tree code generation phase take every valid statement which consist from valid tokens and ask if any token from the tokens in the statement is in the accumulator "Register A" and if it is in the accumulator the assembly will be formed like in figure 9 and if not, it will be formed like figure 10

After the program has been compiled it comes to the assembler to get the object program which will be located in the memory we have made the assembler in the midterm project so we used it as to make a whole complete tiny compiler and we will show some samples from our project.