Mohamed Hani Aljinni 100061418

Key Algorithms

The C4 code identifies and tokenizes input primarily using the next() function. The function reads input sequentially using the p pointer for the position of characters. Each time it is invoked, it reads one character at a time and the sequence is tokenized. The function categorizes the input into tokens according to the pattern defined by the logic of the function.

Identifiers are known by a letter or underscore but are not allowed to start with a number. Furthermore, the dollar sign is not recognized which is standard in GCC. The identifiers are then hashed and stored in the symbol table and compared with existing hashes which must be unique. Numbers are digits handled in decimal, hexadecimal or octal. Hexadecimals are known by an ‘X’ or ‘x’ symbol before the digits. Strings and characters are know by being enclosed in double or single quotes and escape characters are processed (only \n or \0 for this datatype) . Comments are preceded by ‘#’ or ‘//’ and terminated by the null terminator or a new line and a single ‘/’ is the division operator. The token tk is then stored in id table along with its value. The function also deals with operators and gives them tokens.

The expr() function is responsible for the parsing process of expressions. This function defines the order of precedence and associativity, and is called recursively to construct the AST. The parsed expressions are emitted as assembly instructions that is used by the virtual machine (VM). The stmt() function is responsible for parsing statements. These include if-else statements and while loop. This compiler does not deal with do-while or for loops. Conditions are evaluated by the expr() functions and then the integer value is checked. Jump statements are used with addresses saved for code blocks, and the start of the condition of the loop. Jumps are done for if

statements when evaluated to true to avoid the else block, and are used to go back to the start of loop or break from it. Control flow is also managed by the VM using the BZ (Zero branch) or BNZ (Non-zero branch) flag. The parsed output is a sequence of instructions stored in the e array, which serves as the intermediate representation for the VM to handle the operations and flow of the program.

The VM executes the compiled instructions generated during parsing and simulates a simple stack-based architecture. The VM uses three primary registers which are the PC (Program Counter): Points to the next instruction to execute and is also known as the return address, SP (Stack Pointer): Points to the top of the stack, and BP (Base Pointer): Used for managing function call frames and accessing local variables. The VM executes instructions in a loop in main() and fetches and decodes each instruction.

When a function is called, the stack frame grows according to the pool size, the SP is moved to the top by the growth offset, the old BP is pushed to the stack, the BP is moved to the SP location and the PC is updated while the local variables are pushed onto the stack. There is also the function prologue with its ENT instruction and the function epilogue with its LEV instruction. Memory is simply allocated using malloc with the pool size but is never freed until the program ends. The allocation is for the symbols, emitted code, the data, and the stack. The deallocation process or garbage collection is missing here.