Appendix C: TinyCL (Tiny C-Like Language) Reference

This appendix provides a comprehensive reference for the TinyCL language, including its complete syntax, semantics, built-in features, and example programs.

C.1 Language Overview

TinyCL (Tiny C-Like Language) is a modern, fully-featured programming language that demonstrates advanced language implementation techniques. It includes:

- **Variables and Constants**: `var` and `const` declarations
- **Functions**: User-defined functions with parameters and return values
- **Arrays**: Dynamic arrays with indexing
- **Full Expression System**: Arithmetic, logical, and comparison operators with proper precedence
- **Control Flow**: If-else statements and while loops
- **Data Types**: Numbers, strings, characters, booleans, and arrays
- **Comments**: Single-line comments with #

C.2 Complete Syntax Reference

C.2.1 Program Structure

A TinyCL program consists of a sequence of statements:

```
ebnf Program ::= Statements Statements ::= Statement*
```

C.2.2 Statements

TinyCL supports the following types of statements:

```
ebnf Statement ::= FunctionDecl | VariableDecl | ConstantDecl
| IfStatement | WhileStatement | PrintStatement |
ReturnStatement | AssignmentStatement | ExpressionStatement |
Block | Comment
```

Variable Declaration

A variable declaration creates a new variable and initializes it:

```ebnf
VariableDecl ::= "var" Identifier "=" Expression ";"
```

Example: `var x = 42; var name = "Alice"; var numbers = [1, 2, 3];`

Constant Declaration

A constant declaration creates an immutable value:

```ebnf
ConstantDecl ::= "const" Identifier "=" Expression ";"
```

Example: `const PI = 3; const MAX_SIZE = 100;`

Function Declaration

A function declaration defines a reusable block of code:

```ebnf
FunctionDecl ::= "func" Identifier "(" Parameters? ")" Block
Parameters ::= Identifier ("," Identifier)*
```

Example: ``` func add(a, b) { return a + b; }

func factorial(n) { if (n <= 1) { return 1; } else { return n * factorial(n - 1); } } ```

Assignment Statement

An assignment statement assigns a new value to an existing variable:

```ebnf
AssignmentStatement ::= Identifier "=" Expression ";"
```

Example: `x = 42; name = "Bob"; numbers[0] = 10;`

If Statement

An if statement conditionally executes code based on a condition:

```ebnf
IfStatement ::= "if" "(" Expression ")" Block ("else" Block)?
```

Example: ``` if (x > 0) { print("Positive"); } else { print("Non-positive"); }

if (x > 0 && x < 10) { print("Single digit positive"); } ```

While Statement

A while statement repeatedly executes code as long as a condition is true:

```
ebnf WhileStatement ::= "while" "(" Expression ")" Block
```

Example: `while (x > 0) { print(x); x = x - 1; }`

Print Statement

A print statement outputs a value:

```
ebnf PrintStatement ::= "print" "(" Expression ")" ";"
```

Example: `print("Hello, world!"); print(42); print("Result: " + result);`

Return Statement

A return statement exits a function and optionally returns a value:

```
ebnf ReturnStatement ::= "return" Expression? ";"
```

Example: `return 42; return x + y; return; # Return without a value`

Block

A block groups multiple statements together:

```
ebnf Block ::= "{" Statements? "}"
```

Example: `{ var x = 1; var y = 2; print(x + y); }`

Comment

A comment is a line of text that is ignored by the parser:

```
ebnf Comment ::= "#" [^\n]*
```

Example: ```

This is a comment

```

### C.2.3 Complete Expression System

TinyCL has a comprehensive expression system with proper operator precedence:

```
ebnf Expression ::= LogicalOr LogicalOr ::= LogicalAnd ("||" LogicalAnd)* LogicalAnd ::= Equality ("&&" Equality)* Equality ::= Comparison (("!=" | "==") Comparison)* Comparison ::= Term (("<=" | ">=" | "<" | ">") Term)* Term ::= Factor (("+" | "-") Factor)* Factor ::= Unary (("*" | "/") Unary)* Unary ::= ("!" | "-")? Postfix Postfix ::= Primary ("[" Expression "]")* Primary ::= "(" Expression ")" | Identifier "(" Arguments? ")" | "[" Arguments? "]" | Identifier | Number | String | Character | "true" | "false"
```

#### Arithmetic Operators

```
2 + 3 * 4 # Result: 14 (proper precedence) (2 + 3) * 4 # Result: 20 (parentheses override precedence) 10 / 2 - 1 # Result: 4
```

Logical Operators

```
true && false # Result: false true || false # Result: true ! true # Result: false x > 0 && x < 10 # Compound condition
```

Comparison Operators

```
x == 42 # Equal to x != 0 # Not equal to x < 10 # Less than x > 5 # Greater than x <= 100 # Less than or equal to x >= 1 # Greater than or equal to
```
```

Array Operations

```
var arr = [1, 2, 3]; # Array literal var first = arr[0]; # Array access arr[1] = 42; # Array assignment var mixed = [1, "hello", true]; # Mixed types
```

Function Calls

```
var result = add(10, 20); var fact = factorial(5); print("Hello");
```

C.2.4 Data Types and Literals

TinyCL supports multiple data types:

Numbers

`ebnf Number ::= [0-9]+` Example: `42`, `0`, `123`

Strings

`ebnf String ::= '"' StringChar* '"' StringChar ::= [printable characters] | EscapeSequence EscapeSequence ::= '\' ('"' | '\' | 'n' | 'r' | 't' | '0' | 'b' | 'f' | 'v' | 'l')` Example: `"Hello, world!"`, `"Line 1\nLine 2"`, `"Quote: \"Hello\""`

Characters

`ebnf Character ::= "'" CharChar "'" CharChar ::= [printable character] | EscapeSequence` Example: `'A'`, `'1'`, `'\n'`

Booleans

```
true false
```

Arrays

```
[1, 2, 3] ["hello", "world"] [1, "mixed", true] [] # Empty array
```

C.2.5 Identifiers

Identifiers are used for variable, constant, and function names:

```
ebnf Identifier ::= [a-zA-Z_][a-zA-Z0-9_]*
```

Example: `x counter first_name calculateTotal MAX_SIZE`

C.3 Standard Library

TinyCL has a minimal standard library with the following built-in functionality:

C.3.1 Input/Output

- `print(expression)`: Print the value of an expression.

Example: `print("Hello, world!"); print(42); print("The answer is " + 42);`

C.3.2 Arithmetic Operations

TinyCL supports the following arithmetic operations:

- Addition: `a + b`
- Subtraction: `a - b`
- Multiplication: `a * b`
- Division: `a / b`

Example: `let x = 2 + 3; # x = 5 let y = x * 4; # y = 20 let z = y / 2; # z = 10 let w = z - 1; # w = 9`

C.3.3 String Operations

TinyCL supports string concatenation using the + operator:

Example: `let name = "Alice"; let greeting = "Hello, " + name + "!"; # greeting = "Hello, Alice!"`

C.3.4 Comparison Operations

TinyCL supports the following comparison operations:

- Equal to: `a == b`
- Not equal to: `a != b`
- Less than: `a < b`
- Greater than: `a > b`
- Less than or equal to: `a <= b`
- Greater than or equal to: `a >= b`

Example: ``` if (x == 42) { print("x is 42"); }

if (y != 0) { print("y is not 0"); }

if (z < 10) { print("z is less than 10"); } ```

C.4 Example Programs

Here are some example TinyCL programs to demonstrate the language's features:

C.4.1 Hello, World!

```

# Hello, World! program

print("Hello, World!"); ```

### C.4.2 Factorial with Functions

```

Calculate factorial using recursion

func factorial(n) { if (n <= 1) { return 1; } else { return n * factorial(n - 1); } }

var n = 5; var result = factorial(n); print("Factorial of " + n + " is " + result); ```

C.4.3 Fibonacci Sequence

```
# Calculate Fibonacci numbers

var n = 10; var a = 0; var b = 1; var i = 0;

print("Fibonacci sequence:"); print(a); print(b);

while (i < n - 2) { var c = a + b; print(c); a = b; b = c; i = i + 1; } ```

### C.4.4 Array Processing

```
Working with arrays

var numbers = [5, 2, 8, 1, 9]; var sum = 0; var i = 0;

Calculate sum

while (i < 5) { sum = sum + numbers[i]; i = i + 1; }

print("Sum: " + sum);

Find maximum

var max = numbers[0]; i = 1; while (i < 5) { if (numbers[i] > max) { max = numbers[i]; } i = i + 1; }

print("Maximum: " + max); ```

C.4.5 FizzBuzz

```
```

FizzBuzz program

let i = 1;

while (i <= 100) { if (i % 15 == 0) { print("FizzBuzz"); } else { if (i % 3 == 0) { print("Fizz"); } else { if (i % 5 == 0) { print("Buzz"); } else { print(i); } } } i = i + 1; } ```

C.4.6 Prime Numbers

```

# Print prime numbers up to n

let n = 100; let i = 2;

while (i <= n) { let is_prime = 1; let j = 2;

```
while (j < i) {
    if (i % j == 0) {
        is_prime = 0;
    }
    j = j + 1;
}

if (is_prime == 1) {
    print(i);
}

i = i + 1;
```

} ```

## C.5 Language Features Summary

TinyCL is a comprehensive programming language with the following implemented features:
```

✅ Implemented Features

1. **Complete Data Types**: Numbers, strings, characters, booleans, and arrays
2. **User-Defined Functions**: Function declarations with parameters and return values
3. **Arrays and Indexing**: Dynamic arrays with element access and assignment
4. **Full Expression System**: Arithmetic, logical, and comparison operators with proper precedence
5. **Control Flow**: If-else statements and while loops
6. **Variable Management**: Variable and constant declarations
7. **Comments**: Single-line comments with #
8. **Built-in I/O**: Print statement for output

📋 Current Limitations

1. **Limited I/O**: Only supports output via `print` statement (no input capabilities)
2. **No Exception Handling**: No try-catch blocks or error handling mechanisms
3. **No Modules**: No support for importing code from other files
4. **Integer-Only Numbers**: No floating-point number support
5. **Minimal Standard Library**: Only basic built-in functions

🚀 Possible Future Extensions

1. **Floating-Point Numbers**: Add support for decimal numbers
2. **Input Functions**: Add `input()` or `read()` functions
3. **Exception Handling**: Add try-catch blocks for error handling
4. **Module System**: Add `import` statements for code reuse
5. **Object-Oriented Features**: Add classes and objects
6. **Standard Library**: Expand with string manipulation, math functions, etc.
7. **File I/O**: Add file reading and writing capabilities

Despite the current limitations, TinyCL is a fully functional programming language capable of expressing complex algorithms and computations.

Summary

TinyCL is a simple but complete programming language with variables, control structures, and basic I/O. Its syntax is inspired by popular programming languages like JavaScript and Python, making it easy to learn and use.

This appendix has provided a comprehensive reference for TinyCL, including its syntax, semantics, standard library, and example programs. With this information, you should be able to write and understand TinyCL programs.