Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A11

Language Specification

Lab Professor / Lab Session:

[Paulo Sousa / 302]

Team:

[Kshitij Goyani] - Id: [041075243] / [Ishan Vani] - Id: [041078083]

Language Name [Cappy]

|  |  |
| --- | --- |
| **Part**  **1** | **Language User Reference** |

* 1. **User Manual**

**Element 1: Name / Extension**

* The language name is “Cappy”
* File extension is “.cy”

**Element 2 – Comments**

* Single-Line Comments (//): These start with // and continue until the end of the line. Anything after the // on the same line is considered a comment.
* Multi-Line Comments (/\*content\*/): These start with /\* and end with \*/. Anything between these delimiters is considered a comment, and it can span multiple lines.

**Element 3 – Keywords**

break default func interface select

case map struct else goto

package switch const if range

type continue for import return

var true false make new

**Element 4 – Datatypes**

* **Integers:**
* int:

Bytes: Depends on the machine (either 4 or 8 on 32-bit or 64-bit architectures, respectively)

Range: Corresponds to either int32 or int64

* uint:

Bytes: Same as int

Range: Corresponds to either uint32 or uint64

* int8:

Bytes: 1

Range: -128 to 127

* uint8 (also known as byte):

Bytes: 1

Range: 0 to 255

* int16:

Bytes: 2

Range: -32,768 to 32,767

* uint16:

Bytes: 2

Range: 0 to 65,535

* int32:

Bytes: 4

Range: -2,147,483,648 to 2,147,483,647

* uint32:

Bytes: 4

Range: 0 to 4,294,967,295

* int64:

Bytes: 8

Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

* uint64:

Bytes: 8

Range: 0 to 18,446,744,073,709,551,615

* Real Numbers (Floating Points):
* float32:

Bytes: 4

Approximate Range: 1.18E-38 to 3.4E+38

* float64:

Bytes: 8

Approximate Range: 2.23E-308 to 1.80E+308

* Strings:
* string: Strings in Go are sequences of bytes with UTF-8 encoded Unicode code points.

Bytes: Varies based on the length of the string. Strings in Go are dynamic; their size can grow as needed.

**Element 5 – Variables**

1. **Integers:**

To hold numbers with no decimal points, we use types like int, int8, int16, int32, and int64 (or their unsigned counterparts uint, uint8, uint16, uint32, uint64).

**Example :-**

var myInt int = 10

var myInt8 int8 = 20

1. **Floating Point Numbers:**

To hold numbers with decimal points, we use the types float32 and float64.

**Example:-**

var myFloat32 float32 = 10.5

var myFloat64 float64 = 20.55

1. **Strings:**

To hold text, we use the string type.

**Example:-**

var myString string = "Hello World!"

**Element 6 – Methods / Functions**

1. **Define a Type:**

type Circle struct {

    Radius float64

}

* This code defines a new type named Circle. It is a struct with a single field Radius of type float64, representing the radius of the circle.

1. **Define a Method on that Type:**

func (c Circle) Area() float64 {

    return 3.14159 \* c.Radius \* c.Radius

}

* This defines a method named Area on the Circle type. The method calculates and returns the area of the circle. The receiver (c Circle) indicates that this method is associated with the Circle type, and within the method, the instance of the Circle on which the method is called can be accessed using the variable c.

1. **Pointer Receiver Method:**

func (c \*Circle) SetRadius(newRadius float64) {

    c.Radius = newRadius

}

* This method, named SetRadius, is defined with a pointer receiver \*Circle. This means it receives a pointer to an instance of the Circle type. The method updates the Radius field of the circle instance with a new value. Using a pointer receiver allows the method to modify the original instance it's called on, rather than working on a copy.

**Element 7 - Commands**

**Assignment / Attribution:**

* varName = value;
* For instance, age = 25.

**Type Casting**:

* Yes, Cappy will support type casting using a to<Type> function.
* For instance, floatAge = toFloat(age)

**Math Operations:**

* Cappy will support standard arithmetic (+, -, \*, /), with proper precedence rules.

**String Concatenation:**

* Strings can be merged using the + operator.
* For instance, greeting = "Hello, " + "Cappy!"

**Selection:**

1. **If-style Logic:**

if (condition) {

    // code block

} else {

    // code block

}

1. **Switch Case**

switch (varName) {

    case value1:

        // code

    case value2:

        // code

    default:

        // code

}

1. **Boolean Operations:**

* Cappy will use || for OR, && for AND, ! for NOT. Standard comparison operators (<, >, ==, !=, <=, >=) will be used for conditions.

**Interaction:**

1. **For-style Loop:**

for( i = 0; i < 10; i++ ) {

    //...

}

1. **While Loop:**

While ( condition ) {

//...

}

**Input:**

1. **Keyboard Input**

UserInput = read(“Enter Your Name: ”)

**Output:**

1. **Displaying Output**

print "Hello, Cappy!"

print age

**Functions:**

1. **Definition**

function functionName(param1: Type1, param2: Type2) -> ReturnType {

//...

    return result

}

1. **Calling**

result = functionName(value1, value2)

**Element 7 – Proper elements**

Certainly, list comprehensions are a very powerful feature in Python that allow concise expression of sequences. Incorporating such a feature in your new language, Cappy, could provide similar expressive and concise ways to create and transform collections. Below is a detailed explanation of how this feature could be implemented and used in Cappy.

**Feature Name: List Comprehensions**

**Description:**

List Comprehensions in Cappy would allow developers to concisely build lists using a single line of code, utilizing a more readable and declarative syntax. This feature would be particularly useful for transforming, filtering, and creating lists in a more efficient and expressive manner.

**Syntax Proposal:**

In Cappy, a possible syntax for list comprehensions could be inspired by Python, with some modifications to fit Cappy’s syntax style. Here’s a proposed syntax:

newList = [expression for item in collection if condition];

* expression is what each item in the new list will be.
* item is the variable name assigned to each item in the collection.
* collection is the iterable being looped over.
* condition is an optional filter to include only items that satisfy a certain condition.

**Example Usage:**

* Create a list of squares from another list of numbers:

originalList = [1, 2, 3, 4, 5];

squares = [x \* x for x in originalList];

// Result: squares = [1, 4, 9, 16, 25]

* Filtering a list to get only even numbers:

originalList = [1, 2, 3, 4, 5];

evens = [x for x in originalList if x % 2 == 0];

Result: evens = [2, 4]

* Combining two lists into pairs:

list1 = [1, 2, 3];

list2 = ['a', 'b', 'c'];

combined = [(x, y) for x in list1 for y in list2];

Result: combined = [(1, 'a'), (1, 'b'), (1, 'c'), (2, 'a'), (2, 'b'), (2, 'c'), (3, 'a'), (3, 'b'), (3, 'c')]

|  |  |
| --- | --- |
| **Part**  **2** | **Language Comparison** |

**Comparing with C language**

**Differences**

|  |  |  |
| --- | --- | --- |
|  | 1. Memory Management:     - C: Manual memory management using functions like malloc(), free(), etc.     - cappy: Automatic memory management using garbage collection.    2. Concurrency:     - C: Concurrency is achieved using threads (e.g., pthreads).     - cappy: Built-in support for concurrency using goroutines and channels.    3. Object-Oriented Programming:     - C: Doesn't support object-oriented programming natively. However, structs and function pointers can be used to emulate it.     - cappy: Supports a form of object-oriented programming without classes. It uses interfaces and struct types.    4. Error Handling:     - C: Uses error codes and errno.     - cappy: Uses multiple return values, one of which can be an error.    5. Compilation:     - C: Compiles to machine code.     - cappy: Compiles to machine code. Also has a powerful standard library which is statically linked to the binary, making cappy binaries self-contained.    6. Standard Library:     - C: Has a modest standard library focusing on core functionalities.     - cappy: Comes with a rich standard library, especially for web services, networking, etc.    7. Pointer Arithmetic:     - C: Supports pointer arithmetic.     - cappy: Has pointers but doesn't allow pointer arithmetic. |  |

**Advantages / Disadvantages (in comparison with C)**

|  |  |  |
| --- | --- | --- |
|  | 1. Memory Management:     - C: Manual memory management using functions like malloc(), free(), etc.     - cappy: Automatic memory management using garbage collection.    2. Concurrency:     - C: Concurrency is achieved using threads (e.g., pthreads).     - cappy: Built-in support for concurrency using goroutines and channels.    3. Object-Oriented Programming:     - C: Doesn't support object-oriented programming natively. However, structs and function pointers can be used to emulate it.     - cappy: Supports a form of object-oriented programming without classes. It uses interfaces and struct types.    4. Error Handling:     - C: Uses error codes and errno.     - cappy: Uses multiple return values, one of which can be an error.    5. Compilation:     - C: Compiles to machine code.     - cappy: Compiles to machine code. Also has a powerful standard library which is statically linked to the binary, making cappy binaries self-contained.    6. Standard Library:     - C: Has a modest standard library focusing on core functionalities.     - cappy: Comes with a rich standard library, especially for web services, networking, etc.    7. Pointer Arithmetic:     - C: Supports pointer arithmetic.     - cappy: Has pointers but doesn't allow pointer arithmetic. |  |

**Comparing with another language**

**Language Name: Java**

**Differences**

|  |  |  |
| --- | --- | --- |
|  | 1. Memory Management:     - cappy: Automatic memory management using garbage collection.     - Java: Automatic memory management using garbage collection.    2. Concurrency:     - cappy: Built-in support for concurrency using cappy routines and channels.     - Java: Supports concurrency using threads and has built-in libraries for more advanced concurrency patterns.    3. Object-Oriented Programming:     - cappy: Supports a form of object-oriented programming without classes. It uses interfaces and struct types.     - Java: Fully supports object-oriented programming with classes, inheritance, and polymorphism.    4. Error Handling:     - cappy: Uses multiple return values, one of which can be an error.     - Java: Uses a try-catch mechanism for exception handling.    5. Compilation:     - cappy: Compiles to machine code. Also has a powerful standard library which is statically linked to the binary, making cappy binaries self-contained.     - Java: Compiles to bytecode, which runs on the Java Virtual Machine (JVM). |  |

**Advantages / Disadvantages (in comparison with this second language)**

|  |  |  |
| --- | --- | --- |
|  | 1. Memory Management:     - cappy: Automatic memory management using garbage collection.     - Java: Automatic memory management using garbage collection.    2. Concurrency:     - cappy: Built-in support for concurrency using cappy routines and channels.     - Java: Supports concurrency using threads and has built-in libraries for more advanced concurrency patterns.    3. Object-Oriented Programming:     - cappy: Supports a form of object-oriented programming without classes. It uses interfaces and struct types.     - Java: Fully supports object-oriented programming with classes, inheritance, and polymorphism.    4. Error Handling:     - cappy: Uses multiple return values, one of which can be an error.     - Java: Uses a try-catch mechanism for exception handling.    5. Compilation:     - cappy: Compiles to machine code. Also has a powerful standard library which is statically linked to the binary, making cappy binaries self-contained.     - Java: Compiles to bytecode, which runs on the Java Virtual Machine (JVM). |  |

|  |  |
| --- | --- |
| **Part**  **3** | **Architectural Questions** |

**Advantages**

* Cappy seeks to combine Go's concurrency and simplicity benefits with a few special features designed specifically for contemporary web development. Our aim is to produce a language that is easy to learn for new developers while providing them with powerful tools. We also intend to include native support for distributed computing tasks, making it simpler for programmers to create scalable apps without relying on third-party frameworks or tools.

**Strategy: C Implementation**

* Cappy will be implemented using C for its underlying operations. Here are a few initial thoughts:

* + Data Types: Just like Go, we'll have int, float64, and string as primary datatypes. We'll also introduce a webRequest type that makes handling HTTP requests intuitive.
  + Functions: Functions will be similar to Go, but with some added syntactic sugar for asynchronous operations, allowing easy non-blocking I/O.
  + Libraries: Given C's vast ecosystem, we'll provide a mechanism to easily bind and use C libraries in Cappy, extending its capabilities.

**Identifying Language Elements:**

* Tokenization: Break the .cy source code into tokens using space, newline, and special symbols as delimiters.
* Detect Commands: We'll have specific keywords like printToConsole for writing to the console. This will be detected during the parsing phase.
* Handle Literals and Variables: Anything between quotes (e.g., "this gets printed") will be treated as a string literal. Anything without quotes and not recognized as a keyword will be treated as a variable.
* Grammar and Syntax Analysis: We'll define a grammar for our language and use it to validate the syntax of the code.
* Command Detection: Cappy will recognize commands like show for console output.

**Identifying Scope:**

* Marking Code Blocks: Blocks start with { and should be indented until they end with }.
* Loop Constructs: Code within loop's {} braces gets executed as part of the loop. For instance:

loop 5 times {

    show "Printing 5 times."

}

**References**

ChatGPT:- <https://chat.openai.com>

Algonquin College

Fall, 2023