Programming Paradigms

Integration project

Pickle Cannon Programming Language

Done by:

Karolis Butkus s2700603

Group 14

Table of Contents

[Summary 3](#_Toc75774568)

[Problems and solutions 4](#_Toc75774569)

[Detailed language description 5](#_Toc75774570)

[Description of software 6](#_Toc75774571)

[Test plan and results 7](#_Toc75774572)

[Conclusions 8](#_Toc75774573)

[Appendices 9](#_Toc75774574)

[Grammar specification 9](#_Toc75774575)

[Extended test program 10](#_Toc75774576)

# Summary

*Pickle Cannon* is a simplistic programming language mostly intended for simple mathematical or logical calculations. The main features of the language are discussed below.

Firstly, language supports three data types in total. Two basic types - integers and booleans and one compound type – array. Arrays can be only one-dimensional and store the values of one of the basic types. Language is strongly typed, thus, before each new declaration of the variable its type must be specified. Each declaration of the variable does not require a programmer to specify an initial value and it is assigned by default if it was not specified. *Pickle Cannon* also supports local and nested scopes which allow a programmer to re-declare variables with the same name in the newly opened scope.

Secondly, language supports simple mathematical and logical expressions. Addition, subtraction, negation, multiplication, soft-division and comparisons are all possible arithmetic operations that can be applied to integers. Logical negation, logical AND, logical OR and equality/inequality are all possible logical operations that can be applied to booleans.

Thirdly, language supports program control flow constructs. These two constructs are *if* and *while* statements. *if* construct may consist only of *if* statement or of *if-else* statement. *While* cycle will be executed until the condition is met and does not support any cycle-ending commands like a *break* or *continue*.

Fourthly, language supports simplistic concurrency mechanisms. *Pickle Cannon* allows a programmer to spawn and join threads using fork/join construct. Also, language syntax allows declaring shared variables that can be accessed across multiple threads. Moreover, language has one global lock which can be used to make changes to a shared object in a concurrently safe manner.

Lastly, language supports procedures. All procedures are declared before the main body, which in *Pickle Cannon* language starts with the *cannon* keyword. All procedures in the *Pickle Cannon* language are called *pickles*, thus the name of the language. Even though procedures must be declared before the main body, they still can call other procedures even if they are declared below them. As it may already be clear language supports only procedures, so it is not possible to return value to the caller.

These are all main features supported by the *Pickle Cannon* language, which are discussed in greater detail in the *Detailed language description* section.

# Problems and solutions

During the project, there were 5 main encountered problems. All of them are discussed below.

## Concurrency

The first encountered problem was the management of shared data. To make type checking simpler, the definition of shared variables was embedded into the syntax of the language, so that all shared variables must be declared with the keyword ‘shared’. But then another problem arose – what to do with the re-declaration of a shared variable inside the forked thread, how long should it live and so on. Thus, the restriction was imposed that shared variables can be declared only in the global outer scope of the main body. Another encountered problem was the synchronization of the threads in the Sprockell. The main question was how to start and stop threads from executing. This problem was solved by acquiring the maximum number of concurrently executing threads during the elaboration phase and then allocating one memory unit in shared memory space for each thread synchronization. This way each thread could know if others are still executing, and also if they are waiting this space was used to pass the number of their next instruction (to start a thread).

## Memory management

Due to the fact that *Pickle Cannon* language supports arrays memory management complications were encountered. Arrays can take up a varying amount of space, thus storing the whole array in the registers was not the option. To solve this the taken approach was to push all array values from the end to the start on the stack so that another procedure could then just pop the values and get them in order. However, this means that during the execution of the program involving arrays, memory can get quite filled up.

## Register allocation

One major difference from laboratory exercises with ILOC is the fact that register number is limited, thus careful register management is needed. One of the major difficulties is the fact only 6 registers are available for general use, and due to the fact that *Pickle Cannon* language supports procedures one register was needed to store ARP, thus only 5 general use registers were left. So to make sure that all calculations were possible each expression calculation after using the needed registers would free them as early as possible so that the registers limit would not be reached. The most demanding operations were soft-division and array storing. For example, a division operation would use up 4 registers as it would need 2 registers for expressions, 1 register for result accumulator and 1 for general values (such as storing comparison values or offsets). That is why after this operation is done 3 registers must be freed immediately (1 register must remain to store the value) to allow other operations to execute normally.

## Arrays

There were two encountered problems with the arrays. The first problem was array storing. As the array is a compound type it does not have any predefined size as it depends on the number of stored values. So to make the compilation and type checking process easier, language enforces the user to declare the size of each array (even in the procedure parameter definition) that must remain the size during the whole execution. Also, due to time limitations and trying to keep code cleaner and more understandable, multi-dimensional arrays were omitted. The second encountered problem was the run-time errors of accessing array values out of bounds. Due to the fact that the exception handling mechanism was not implemented, the taken approach is similar to the C language - it is to inform the programmer to carefully access the array values, he is intending to.

## Procedures

The last encountered problem was concerning procedures. Due to the fact that procedures can call other procedures, it meant that procedure call type check could be executed only after all other elaboration steps have been done. The approach was to store all procedure calls in the list, and at the end of the elaboration phase check if they try to access the exiting procedures with correct parameters (language does not support nested procedures so all procedures are visible from the global scope).

# Detailed language description

## Basic types

## Arrays

## Assignments

## Expressions

### Integer expressions

Boolean expressions

## Local, nested scopes

## Control flow constructs: *if* and *while*

## Concurrency

## Procedures

# Description of software

# Test plan and results

# Conclusions

# Appendices

## Grammar specification

## Extended test program