CPS2000 - Compiler Theory and Practice
Course Assignment (Part II) 2020/2021

Department of Computer Science, University of Malta
Sandro Spina

April 27, 2021

Instructions

- This is the description for Part II of the assignment.
- This is **an individual** assignment. This assignment carries **50%** of the final CPS2000 grade.
- The submission deadline is Saturday **18th June 2021**, same as for for Part I. A soft-copy of the report and all related files (including code) must be uploaded to the VLE by midnight of the indicated deadline. Hard copies **are not** required to be handed in. Source and executable files must be archived into a single .zip file before uploading to the VLE. It is the student's responsibility to ensure that the uploaded .zip file is valid. The PDF report (separate from that for Part I) must be submitted through the Turnitin submission system on the VLE.
- A report describing how you designed and/or implemented the different tasks of the assignment **is required**. Tasks (1-4) for which no such information is provided in the report will **not be** assessed.
- You are welcome to share ideas and suggestions. However, under no circumstances should code be shared among students. Please remember that plagiarism will not be tolerated; the final submission must be entirely your work.
- You are to allocate approximately **40** hours for this assignment.

Description

Over the years, *TeaLang* has grown in popularity and several developers started requesting additional features to improve the language and further increase its adoption amongst software houses. The highly anticipated new compiler is now due to be released around Q2 2021. *Tea2Lang* is expected to be a solid improvement over *TeaLang* and adds support for arrays, structs, the 'auto' type specifier and function overloading.

In this assignment you - the developer of the new *Tea2Lang* compiler - will be tasked with extending *TeaLang* (note that the specification of *TeaLang* can be found in Part I) into *Tea2Lang*. This assignment (Part II) is composed of three major components: i) modifications to the original *TeaLang* language specification, ii) extension of the Lexer/Parser and visitor classes for *TeaLang* to handle *Tea2Lang* features iii) a report detailing how you designed and implemented the different tasks. The following is a syntactically correct *Tea2Lang* program:

```
auto XGreaterY(int toCompare[]) {
 let ans:auto = false;
 if (toCompare[0] > toCompare[1])
 {
    ans = true;
 }
 return ans;
}

auto XGreaterY(int x, int y) {
  if (x>y) { return true; } else { return false; }
}

float Average(float toAverage[], int count) {
  let total:float = 0.0;
  for (int i = 0; i<count; i = i+1)
  {
    total = total + toAverage[i];
  }
  return total / count;
}

let arr1[2]:float;
let arr2[4]:float = { 2.4, 2.8, 10.4, 12.1 };

arr1[0] = 22.4;
arr1[1] = 6.25;
print arr1[1];                              //6.25
print XGreaterY(arr1);                      //true
print Average(arr2);                        //6.92
print XGreaterY(2,3);                       //false
```

Task Breakdown

The assignment is broken down into four tasks. Below is a description of each task with the assigned mark.

Task 1 - char type and Arrays

In this first task you are to extend *TeaLang* by adding another primitive type, 'char', to represent individual characters and also include support for arrays. An array in *Tea2Lang* consists of a series of elements of the **same** type in contiguous memory that can be individually accessed using the indexing operator [ ] over the array identifier. By default arrays are left uninitialised when declared, i.e. element values are not individually set. The programmer has the option of initialising the elements in the array upon declaration. Check out the code fragments above for examples of how this can be done. Deviations from the syntax used in this code fragment are allowed as long as you support the required functionality. The following subtasks are included:

- Update the EBNF grammar with new (or modified) rules to support arrays and 'char' type.
- Update the Lexer to generate the required new tokens.
- Update the Parser to handle these new tokens.
- Update the XML visitor to include 'char' and arrays.
- Update the Semantic Analysis visitor to support the new type and the use of arrays.
- Update the Execution visitor to support evaluation of expression using 'char' and arrays.

[Marks: 30%]

Task 2 - The 'auto' type

Tea2Lang supports the 'auto' type specifier both for variables and functions. Check out the code fragments above for examples. For variables it specifies that the type of the variable that is being declared will be automatically deduced from its initialiser. For functions it specifies that the return type will be deduced from its return statements. Note the 'auto' type specifier cannot be used to specify the type of the formal parameters in function declarations. The following subtasks are included:

- Update the EBNF grammar with new (or modified) rules to support 'auto'.
- Update the Lexer to generate the required new token.
- Update the Parser to handle the new token.
- Update the XML visitor to include 'auto' types.
- Update the Semantic Analysis visitor to support resolution of 'auto' variables and functions.
- Update the Execution visitor to support evaluation of expressions using 'auto'. Changes, if any, are minimal here since most of the work is done at the semantic analysis phase.

[Marks: 20%]

Task 3 - Structs

Structs are user-defined data types in *Tea2Lang*. Unlike arrays, they allow programmers to combine data items of different primary data types under a single name. Each element in a struct is referred to as a member. In addition to data members (i.e. a normal *Tea2Lang* variable declaration), structures in *Tea2Lang* also support member functions. The structure is defined with the keyword 'tlstruct' and all its members are public. The code below demonstrates how structs in *Tea2Lang* are used:

```
tlstruct Vector {
  let x:float = 0.0;
  let y:float = 0.0;
  let z:float = 0.0;

  int Scale(float s) {
    x = x * s;
    y = y * s;
    z = z * s;
    return 0;    //Because functions always return something
  }

  int Translate(float tx, float ty, float tz) {
    x = x + tx;
    y = y + ty;
    z = z + tz;
    return 0;    //Language does not support void
  }
}

Vector Add(Vector v1, Vector v2) {
  let v3:Vector;
  v3.x = v1.x + v2.x;
  v3.y = v1.y + v2.y;
  v3.z = v1.z + v2.z;
  return v3;
}

let v1:Vector:
v1.x = 1.0;
v1.y = 2.0;
v1.z = 2.0;

let v2:Vector;
v2.x = 2.0;
v2.y = 1.2;
v2.z = 0.0;
```

```
let v3:Vector = Add (v1, v2);

print v3.x;    // 3.0
print v3.y;    // 3.2
print v3.z;    // 2.0

v3.translate(1.0, 1.0, 1.0);

let v4:Vector = Add (v1, v3);

print v3.x;    // 5.0
print v3.y;    // 6.2
print v3.z;    // 5.0
```

The following subtasks are included:

- Update the EBNF grammar with new (or modified) rules to support structures.
- Update the Lexer to generate the required new tokens.
- Update the Parser to handle these new tokens.
- Update the XML visitor to include 'auto' types.
- Update the Semantic Analysis visitor to support additional semantic checks for structures.
- Update the Execution visitor to support evaluation of programs using 'tlstruct'.

[Marks: 30%]

Task 4 - Function overloading

In *Tea2Lang*, two functions can have the same name if the number and/or type of the arguments passed is different. This feature is referred to as function overloading. As far as the Lexer and Parser are concerned, no changes need to be carried out, however, the semantic analysis and execution visitors will now need to cater for this new feature. An example of function overloading is seen in the code examples (page 2). The function XGreaterY takes as argument both an array of int and two integers types. The following subtasks are included:

- Update the Semantic Analysis visitor to support function overloading.
- Update the Execution visitor to support evaluation of programs which include function overloading. Changes are minimal here since most of the work is done at the semantic analysis phase.

[Marks: 20%]

Report

In addition to the source and class files, you are to write and submit a report (separate from the Assignment (Part 1) report). Remember that Tasks 1 to 4 for which no information is provided in the report will not be assessed. In your report you should include any sample *Tea2Lang* programs you developed for testing the outcome of your compiler. State what you are testing for, insert the program AST (pictorial or otherwise) and the outcome of your test.