Viz: a Beginner's Language

Matthew Duran - md3420 (Tester)
Jakob Deutsch - jgd2150 (System Architect)
Yanhao Li - yl4734 (Tester)
Nicholas Wu - nkw2115 (Language guru)
Jinsen Wu - jw4157 (Project Manager)

1 Introduction

Viz is a general-purpose programming language that allows the visualization of data structure and the highlight of each operation step. It is imperative, statically scoped, statically and weakly typed like C++ but with simpler features and more intuitive syntax. Our programming language supports the most basic primitive data types, operations, and control flows. On top of the basics, we also include features such as garbage collection, object-oriented and abstract data types like array, stack, and tree. For visualization, Viz will generate a styled HTML file for developers to examine each step of operation applied to data structures. Furthermore, viz will be a helpful tool for beginners through friendly syntax which will allow users to not get bogged down by computer science fundamentals, but rather focus on writing code that executes a given task.

1.1 Motivation

Data structures are the core of computer science. The mastery of data structures is the keystone for any developer on building optimized software. They are also crucial during computer science technical interviews in top companies. However, for many beginners, the concept and utilization of data structures are difficult to grasp. Our aim is to help beginner programmers better understand different data structures and their implementations. We chose to generate visualizations through HTML instead of output console because we believe a clear and intuitive illustration can not only provide accurate guides for programmers but also incentivize them to study.

2 Paradigms and Features

2.1 Primitive Data Types

string	An array of ASCII characters, wrapped with double or single quotation marks.
int	4 bytes, signed integer.
float	4 bytes, floating-point number
boolean	1 byte, true/false value

2.2 Built-in Complex Data Types

Each of these abstract data types (ADT) may only contain the same type within the structure. These five ADTs will be built in and generic, and we will use the following syntax to instantiate an object.

- ADT|data type|
- E.g. Queue|string|, Array|int|, treeNode|int|

Furthermore, since this is a beginner's language they will not need to worry about garbage collection and can do something like the example below without fear of memory management

- Array|Array|int||, a hash table, for example,

array	A collection of elements of same primitive data type
queue	A first-in-first-out collection of same-type elements.
stack	A last-in-first-out collection of same-type elements.
linkedNode	The smallest unit of a doubly linked list. It has a value field and two reference fields "prev" and "next" that link to the previous node and next node
treeNode	A tree data structure where each node has at most two children: left child and right child. The smallest unit of a binary tree. It has a value field and has at most two children: left child and right child.

2.3 Basic operators

Arithmetic	+,-,*,/,%
Assignment	=, +=, -=, *=, %=
Relational	==, !=, >=, <=, >, <
Logical	and, or, not

2.4 Control flow:

Conditional	If (...) {...} else if (...){...} else {...}
Loop	while(...) {...}
	for n in start..end {...} `// start and end are inclusive, increment by 1 in each iteration`
	for n in start..<end {...} `// end is excluded, increment by 1 in each iteration`
	for n in start..end step n {...} `// increment by n in each iteration`
	for n in [2, 7, 8] {...} `// iterate through element in an array`

2.5 Functions

```
function (arg_type arg1, …) -> return_type {
  …
```

```
 return ...
}
```

In addition to the data type listed above, *None* can be used as a return_type

2.6 Imperative, Statically Typed and Weakly Typed

```
function absoluteValue(int num) -> int {
  if (num >= 0) {
    return num;
  }
  return -num;
}

function add1(int x) -> int {
  int ans = x + 1;
  return ans;
}

function alphabet(int x) -> string {
  return x % 26 + “a”;
  // weakly typed, so this function will return a string contains a single
  // character
}
```

2.7 Conventions

int x = 1;
Semicolons are the line ending syntax
{ … } denote a block of code

// single line comment
/ *
*/ block comment

function main() will be the only file run, so if there is no main() then the file will be treated as a module which can be imported and used.

2.8 viz() function

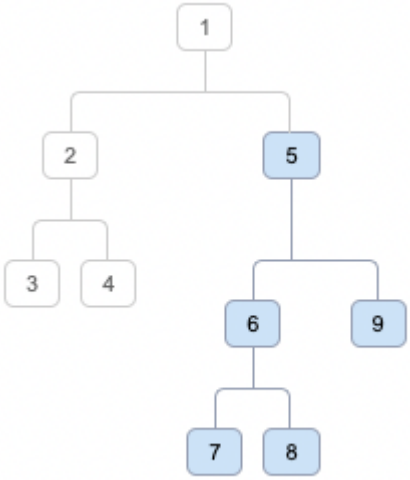
viz() will be a built in library function that will use the data structure object to display its contents. This will be done by writing the fields to an html file that will be generated, and which you could open, for either debugging or for a teaching tool for new students.

```
/*
each ADT defined as a part of the language will have a viz() method which
will display the inner workings of the data structure in an html file which
you can open up and render in a browser.
Any user defined objects will of course not have this implemented, unless
they did that themselves.
*/
We can support:
queue.viz() // each of these ADTs will have their own html format as part of
array.viz()  // viz standard library
stack.viz()
listHead.viz()

// example pictured below
treeNode|int| root = 1;
treeNode|int| left = 2;
treeNode|int| right = 5;
treeNode &cur = root; // created reference instead of new copy
cur.left = left; cur.right = right;

left.left = 3; left.right = 4; // indirectly assign left and right children with the
same type will automatically create treeNode with the same type

right.right = 9;
cur = right;
cur.left = 6;
cur = cur.left; // cur is now pointed to the left child of previous cur
cur.left = 7; cur.right = 8;
root.viz().highlight(right) // below styled html will be generated
```



3 Language In One Slide Program

first_program.viz

```
/*
 * only file that will be run is one that has main() defined
 */
function main(arg_type arg1, …) -> None {
   queue|String| hw = {“hello, ”}; // initialize Queue with a value
   hw.add(“world!”);
   hw.viz(); // html file with queue struct will have hello, world as elements
   for (int i = 0; i < n; i++) {
     print(hw[i]); // not fancy print to console like any other lang
   }
   return;
}
```