Strux
Data Structure Visualization Language

Sophie Stadler, Millie Yang, Joshua Bartlett, Fredrick Kofi Tam
Strux

- Strux is a general-purpose language that allows for the visualization of data structures.
- Using a familiar Java-like syntax, it helps programmers become familiar with the different ways data is stored.
- Easy to use: data structures are built in.
Motivation

Learning data structures is difficult

- It can be hard to relate code to data structure concepts
- Textbook drawings of data structures are static
  - Difficult to see how they relate to code after each operation

Solution: Visualizations

Visualizations

- ASCII art rendering of a stack, queue, linked list, binary tree, or array
- Dynamic
- Help programmers:
  - Become familiar with the key features of each structure
  - Expose the data their objects currently contain
Features

Standard Language Elements
- Strictly typed
- Operators
  - Increment and decrement
- Control flow
  - if/elif/else
  - Loops
  - Statements
- Program entry point: `main()`
  - Must return `int`

Strux Specialties
- Built-in data structures
  - Array
  - Tree
  - Stack
  - Queue
  - LinkedList
- `.show()`
  - Data structures are printed in ASCII format
- Quick Sort arrays: step-by-step visualization
- Unified function calls for data structures
  - `enqueue/push → add()`
  - `dequeue/pop → remove()`
  - `quicksort()/tree can take in num[], string[], or int[]`
Software Technologies Used

GitHub
LLVM
OCaml
Ubuntu
Atom
C
DigitalOcean
Running Strux

$ make
$ ./linkStrux.sh
$ ./testall.sh
Running Strux

linkStrux.sh

- Stack, Queue, LinkedList, Tree, showQuickSort written in C
- linkStrux.sh converts C to bytecode

testall.sh

- MicroC test script
- Runs all .strux files in tests/ directory and compares output to .out or .err file
Test

- **Edge Cases**
  - Visualization of large numbers or large data structures
  - Calling methods on empty data structures
  - Handling null exceptions
- **Incompatible Types**
  - Initialization, typos
- **Static vs. dynamic scoping**
- **Pass by value vs. by reference**
- **Duplicate initialization**
- **Calling methods that do not exist**
Test - Unit and Integration Tests

Unit Test

```c
int main() {
    int i = 2;
    print(i--);
    return 0;
}
```

Integration Test

```c
int main() {
    print(gcd(81, 153));
    return 0;
}
```

```c
int gcd(int n1, int n2) {
    int gcd = 1;

    for (int i = 1; i <= n1 and i <= n2; i++) {
        if (n1 % i == 0 and n2 % i == 0) {
            gcd = i;
        }
    }
    return gcd;
}
```
Test - Incompatible Types/Error Checks

Fail Test:
int main() {
    LinkedList::string l = new LinkedList::string();
    l.add(3);
    return 0;
}

Terminates Gracefully:
Fatal error: exception Failure("illegal actual add argument found int expected string in 3")
```cpp
int main()
{
    Stack::int s = new Stack::int(2, 3, 4);
    s.show();
    s.remove();
    s.show();
    s.add(5);
    s.show();
    return 0;
}
```

Console Output:
```
+----- <- Top
| 4 |
+-----
| 3 |
+-----
| 2 |
+-----
| 3 |
+-----
| 2 |
+-----
| 5 |
+-----
| 3 |
+-----
| 2 |
+-----
```
int main() {
    Queue::string struxers = new Queue::string("Josh", "Kofi", "Millie", "Sophie");
    struxers.show();
    struxers.add("Edwards");
    struxers.show();
    struxers.remove();
    struxers.show();
}
int main() {
    LinkedList::num ll = new LinkedList::num(8.6, 6.0);
    ll.show();

    ll.delete(1);
    ll.show();
}

+----------+  +----------+  +------+      +----------+  +------+
| 8.600000 |->| 6.000000 |->| NULL |      | 8.600000 |->| NULL |
+----------+  +----------+  +------+      +----------+  +------+

0             1             <- Index      0             <- Index
int main() {
    int[4] myArr = [0, 1, 2, 3];
    myArr.show();

    myArr[2] = 8;
    myArr.show();
}

[0, 1, 2, 3]
[0, 1, 8, 3]
```c
int main() {
    int[9] myArr = [1, 4, 3, 6, 7, 2, 99, 23, 37];
    myArr.quickSort();
    myArr.show();

    num[9] a = [1.2, 4.4, 3.5, 6.5, 7.5, 2.3, 23.9, 99.5, 37.9];
    a.quickSort();
    a.show();
}
```

```
[1, 2, 3, 4, 6, 7, 23, 37, 99]
[1.200000, 2.300000, 3.500000, 4.400000, 6.500000, 7.500000, 23.900000, 37.900000, 99.500000]
```
int main() {
    int[4] myArr = [3, 4, 1, 6];
    myArr.showQuickSort();
}

At this step:
current array: [3 4 1 6 ]
numbers swapped: 1,4
array after swap: [3 1 4 6 ]
pivot is 4

At this step:
current array: [3 1 4 6 ]
pivot swapped: 3,1
array after swap: [1 3 4 6 ]
pivot is 3

At this step:
current array: [1 3 4 6 ]
pivot is 1

QuickSort complete! Final Result: [1 3 4 6 ]
int main() {
    BSTree::int tree = new BSTree::int();
    tree.add(1);
    tree.add(2);
    tree.add(3);
    tree.add(0);
    tree.add(-4);
    tree.add(-2);
    tree.add(-3);
    tree.show();
    return 0;
}
Challenges

macOS High Sierra

- On the new macOS, some previous architectural systems are not being supported
  - LLVM.Bitreader would throw a bitreader error
- Because we need to translate our C code into bitcode, we decided to develop on an Ubuntu instance hosted on DigitalOcean to introduce a level of abstraction and make it easier for members on the team with High Sierra
- Older macOS systems still work perfectly with Strux code

Limitations of ASCII

- Very large data structures appear distorted when show() is called due to the way they are printed
- For LinkedList and Queue, we introduce compact visualizations when structures are too long

LinkedList

```
+----+  +----+  +------+
| 0 |->| 1 |->| NULL |
```

vs.

```
[0]->[1]->[NULL]
```

```
+----+  +----+  +------+
| 0 | 1 |
```

Queue

```
+------------+
| 0 | 1 | 2 |
```

vs.

```
Head->[0][1][2]<-Tail
```

```
------------+
Head  Tail
```
NOW TIME FOR THE DEMO!