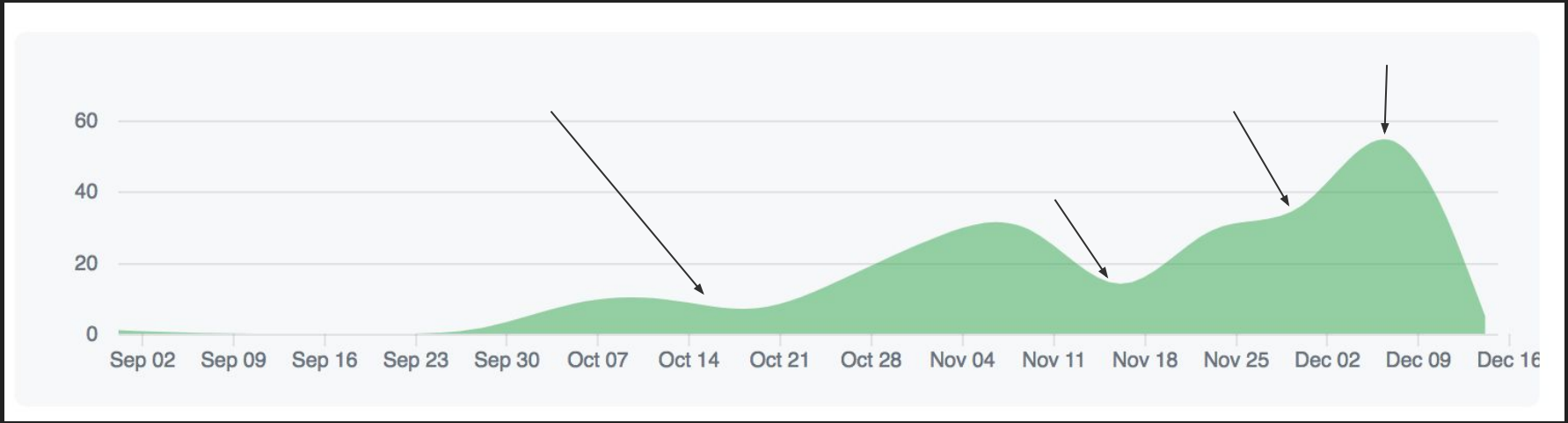
Grape.grp

Timmy Wu, Nick Krasnoff, Edward Yoo, James Kolsby

Timeline

Milestones	Time
LRM, Scanner done, elementary Parser	10/15
Parser, AST, SAST, “Hello World” *	11/18
Semantically checked types (edges, nodes)	11/25
Edge, Node, List typing in codegen.ml*	12/2
Graph type in codegen.ml	12/10
Writing C library, Linking C library *	12/11
List indexing, Dot notation, Overloading functions*	12/12



Design Philosophy

- Our Goals

  Execute graph algorithms

- Why Grape

  The primary motivation behind Grape is to enable the parsing and manipulation of graphs using simple syntax and inline initialization

C Graph vs Grape Graph

```c
#include <stdio.h>
#include <stdlib.h>

#include "list.h"
#include "graph.h"

int main() {
    struct Graph *g= init_graph();
    void *ptra = malloc(sizeof(int));
    *((int*)ptra) = 5;
    void *ptrb = malloc(sizeof(int));
    *((int*)ptrb) = 7;
    void *ptre = malloc(sizeof(int));
    *((int*)ptre) = 6;

    struct Node *a= init_node(ptra);
    struct Node *b= init_node(ptrb);
    struct Edge *e= init_edge(ptre);
    link_edge_from(e, a);
    link_edge_to(e, b);
    add_node(g, a);
    add_node(g, b);
    add_edge(g, e);
    struct List *d = get_outgoing(a);
    struct Node *result = d->head->to;
    printf("%d\n", *(int *)result->data);
    return 0;
}
```

```
fun Int main(){
    Node<Int> a = '5';
    Graph<Int, Int> graph = << a -6- '7' >>;
    List<Edge<Int> > nodeList = graph.outgoing(a);
    Edge<Int> edge= nodeList[0];
    Node<Int> toNode = edge.to;
    print(toNode.val);
    return 0;
}
```

A simple program that creates a graph with an Edge and two Nodes and gets the value of the neighbor of one of the Nodes

The Grape program is much simpler and more intuitive

Types

Edge: directed edges, can hold any data type

Node: Hold any data type, can have multiple edges outgoing to multiple nodes

List: Typed list, can hold any data type

Graph: Holds node and edge that respectively hold their own data.

List Manipulation

- Indexing
- Nested list with reference types

Nested List (with Int and Node):

```
fun Int main() {
    String hi = "hi";
    print(hi[0]);

    List<List<Int> > a = [
        [1,2,3,4,5],
        [1,2,3,4,5],
        [1,2,300,4,5],
        [1,2,3,4,5]];

    print(a[2][2]);

    List<List<Node<Int> > > b = [
        ['1','2','3','4','5'],
        ['1','2','3','4','5'],
        ['1','2','313','4','5'],
        ['1','2','3','4','5']];

    print(b[2][2].val);

    return 0;
}
```

Graph types

```
fun Int main() {

        Graph<Int, Int> a;

        a = <<'3' -3- '4'>>;

        return 0;

}
```

Node:

```
fun Int main() {
        Node<Int> a;
        a = '3';
        return 0;
}
```

Edge:

```
fun Int main() {
        Edge<Int> a = <<-3->>;

        return 0;
}
```

DEMO: Simulating a DFA