Grape (.grp)

Team members:
Edward Yoo, UNI: hy2506
James Kolsby, UNI: jrk2181
Michael Makris, UNI: mm3443
Nick Krasnoff, UNI: nsk2144
Timmy Wu, UNI: pw2440

Description:

Grape is a language that is designed primarily to solve problems in the graph theory domain. It should allow the programmer to develop code that implements graph algorithms like Dijkstra's, Prim's and Kruskal's.

At the foundation of the language are edges. An edge is a container that points to two nodes, linking them together. They can be either directed or undirected

```
"hello" -> "world"
```

Nodes can be any data type, and are not a type of their own. Everything is a node except edges.

```
a = "hello"
b = 4
```

Edges also contain a pointer to their own object, for instance an integer to represent the cost of the edge

```
a -2- b
```

Graphs are described as a list of edges.

```
friends = [
	james -1- "Tim" -3- "Nick",
	"Michael" -2- "Ed" --
]
```

Types

Integer	int	10
Float	float	3.14
String	str	"Hello"
Boolean	bool	true/false
Null		null
List	list	[1 2 3]
Dictionary	dict	{foo:1, bar:2}
Edge	edge	(node -- node)
		(node -> node)
Graph	list	[a -1- b -2- c]

Operators

Assignment	=
Graph slice	in
Comparison	< > ==
Arithmetic	+ - * / ^

edge assignment:

Undirected: () -- (),
Directed: () -> ()

Precedence

Graph templating

Node wildcard	`*`
Node with type	`str`
Edge wildcard	`--`
Named node	`a`
Named typed node	`a:str`
Edge with nodes	

Control flow

Scope	{}
Loop	for … {}
Conditional	if … {}

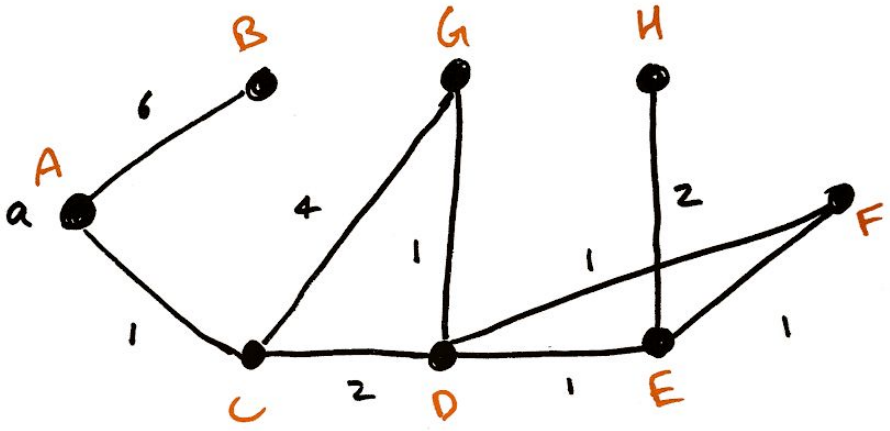
Functions

fun <name> <arg> <arg> {}

I/O

print "Hello world"

An example program (shown on the next page) that a user might write would be Dijkstra's Algorithm on the following graph, starting from vertex A



DIJKSTRA'S ALGORITHM

```
fun minDistance graph {
    minValue = INF
    min // Empty node

    // Slice graph with template
    for `a` in graph {
        if (a.dist < minValue &&
            a.visited == false) {

            minValue = a.dist
            min = a
        }
    }

    return min
}

fun dijkstra graph start {

    for `a` in graph {
        a.dist = INF
        a.prev = null
    }

    // Every node in graph
    for `*` in graph {

        current = minDistance graph

        for `current -a- b` in graph {

            newDist = current.dist + a

            if newDist < b.dist {
                b.dist = newDist
                b.prev = current
            }
        }

        current.visited = true
    }
}

dijkstra myGraph a

// Initialize empty dicts
a,b,c,d,e,f,g,h:{}

myGraph = [
    a -6- b,
    a -1- c -2- d -1- e -1- f,
    c -4- g -1- d,
    d -1- f,
    e -2- h
]
```