JSPP

Morph C++ into Javascript

Quiz

What programming language is it?

```
jParser.prototype.parse = function (structure) {
    // f, 1, 2 means f(1, 2)
    if (structure instanceof Function) {
        return structure.apply(this,
          Array.prototype.slice.call(arguments, 1));
    }

    // ['string', 256] means structure['string'](256)
    if (structure instanceof Array) {
        var key = structure[0];
        if (!(key in this.structure)) {
            throw new Error("Missing structure for `" + key + "`");
        }
        return this.parse.apply(this,
          [this.structure[key]].concat(structure.slice(1)));
    }
```

```
static Handle<Code> ComputeCallDebugPrepareStepIn(int argc, Code::Kind kind) {
  Isolate* isolate = Isolate::Current();
  return isolate->stub_cache()->ComputeCallDebugPrepareStepIn(argc, kind);
}


static v8::Handle<v8::Context> GetDebugEventContext(Isolate* isolate) {
  Handle<Context> context = isolate->debug()->debugger_entry()->GetContext();
  // Isolate::context() may have been NULL when "script collected" event
  // occured.
  if (context.is_null()) return v8::Local<v8::Context>();
  Handle<Context> global_context(context->global_context());
  return v8::Utils::ToLocal(global_context);
}
```

```
var Utils = {
  _["mapInplace"] = function (var array, var func) {
    for (var i = 0; i < array["length"]; ++i) {
      array[i] = func(i, array[i]);
    }
    return undefined;
  }
};
```

```
var a = {"a", "b", "c"};
std::cout << a << std::endl;
// [a, b, c]

Utils["mapInplace"](a, function (var key, var value) {
  return "(" + key + ":" + value + ")";
});
std::cout << a << std::endl;
// [(0:a), (1:b), (2:c)]
```

```
#include "../src/javascript_start.h"

var Utils = {
  _["mapInplace"] = function (var array, var func) {
    for (var i = 0; i < array["length"]; ++i) {
      array[i] = func(i, array[i]);
    }
    return undefined;
  }
};
```

```
var a = {"a", "b", "c"};
std::cout << a << std::endl;
// [a, b, c]

Utils["mapInplace"](a, function (var key, var value) {
  return "(" + key + ":" + value + ")";
});
std::cout << a << std::endl;
// [(0:a), (1:b), (2:c)]


#include "../src/javascript_end.h"
```

Plan

- JSON
  - Primitives
  - Arrays
  - Objects
- Lambda Functions
- Prototypal Inheritance

JSON

```
var json = {
    "number": 42,
    "string": "vjeux",
    "array": [1, 2, "three"],
    "nested": {
        "first": true
    }
};
```

Javascript Primitives

- undefined, null
- true, false
- Numbers
- Strings

```
var undefined,
bool   = true,
number = 4.2,
string = "JSConf";
```

C++ Primitives

```
struct Value {
  Value() {}                  // undefined
  Value(double n) {}          // number
  Value(int n) {}             // number
  Value(bool b) {}            // boolean
  Value(const char* str) {} // string
};
```

C++ Primitives

```
struct Value {
  Value() {}                  // undefined
  Value(double n) {}          // number
  Value(int n) {}             // number
  Value(bool b) {}            // boolean
  Value(const char* str) {} // string
};
```

```
Value v;
Value v(4.2);
Value v = "JSConf";
```

C++ Primitives

```
struct Value {
  Value() {}                  // undefined
  Value(double n) {}          // number
  Value(int n) {}             // number
  Value(bool b) {}            // boolean
  Value(const char* str) {} // string
};
typedef Value var;


Value v;
Value v(4.2);
Value v = "JSConf";
```

C++ Primitives

```
struct Value {
  Value() {}                    // undefined
  Value(double n) {}            // number
  Value(int n) {}               // number
  Value(bool b) {}              // boolean
  Value(const char* str) {} // string
};
typedef Value var;


var v;

var v = "JSConf";
```

C++ Primitives

```
struct Value {
  Value() {}                  // undefined
  Value(double n) {}          // number
  Value(int n) {}             // number
  Value(bool b) {}            // boolean
  Value(const char* str) {} // string
};
typedef Value var;


var undefined,
    boolean = true,
    number  = 4.2,
    string  = "JSConf";
```

C++ Primitives

```
struct Value {
  Value() {}                  // undefined
  Value(double n) {}          // number
  Value(int n) {}             // number
  Value(bool b) {}            // boolean
  Value(const char* str) {} // string
};
typedef Value var;

int main() {
    var undefined,
        boolean = true,
        number  = 4.2,
        string  = "JSConf";
}
```

Trial and Error

```
var array = A("JSConf", true, 4.2);
```

Trial and Error

```
var array = A("JSConf", true, 4.2);
```

```
var array = $[_, "JSConf", true, 4.2];
```

Trial and Error

```
var array = A("JSConf", true, 4.2);
```

```
var array = $[_, "JSConf", true, 4.2];
```

```
var array = {"JSConf", true, 4.2};
```

C++11 Initializer List

```
struct Value {
  Value(std::initializer_list<Value> list);
};
```

Fun Fact

```
js> {a: 1}
1
```

Trial and Error

```
var json = O(
  "place", "JSConf",
  "talk", O(
    "length", 4.2,
    "pwn", true
  )
);
```

Trial and Error

```
var json = O(
  "place", "JSConf",
  "talk", O(
    "length", 4.2,
    "pwn", true
    )
 );
```

```
var json = ($<
  "place" | "JSConf" +
  "talk" | ($<
    "length" | 4.2 +
    "pwn" | true
  >$)
>$);
```

Trial and Error

```
var json = O(
  "place", "JSConf",
  "talk", O(
    "length", 4.2,
    "pwn", true
  )
);
```

```
var json = ($<
  "place" | "JSConf" +
  "talk" | ($<
    "length" | 4.2 +
    "pwn" | true
  >$)
>$);
```

```
var json = {
  _["place"] = "JSConf",
  _["talk"] = {
    _["length"] = 4.2,
    _["pwn"] = true
  }
};
```

Precedence

```
1 ::
2 [] ++ -- () . ->
3 ++ -- + - ! ~ * & (type) sizeof new delete
4 .* ->*
5 * / %
6 + -
7 << >>
8 < <= > >=
9 == !=
10 &
11 ^
12 |
13 &&
14 ||
15 ?:
16 = += -= *= /= %= <<= >>= &= ^= |=
17 throw
18 ,
```

Implementation

```
{ _["key"] = "value" }
```

```
Underscore _;
```

Implementation

```
{ _["key"] = "value" }
```

```
{ _["key"] = "value" }
```

```
Underscore _;

struct Underscore {
  KeyValue& operator[](Value k);
};
```

Implementation

```
{ _["key"] = "value" }
```

```
{ _["key"] = "value" }
```

```
{ _["key"] = "value" }
{ _["key"] = "value" }
```

```
Underscore _;

struct Underscore {
  KeyValue& operator[](Value k);
};
```

```
struct KeyValue {
  KeyValue(Value k);
  KeyValue& operator=(Value v);
};
```

Implementation

```
{ _["key"] = "value" }
```

```
{ _["key"] = "value" }
```

```
{ _["key"] = "value" }
```

```
{ _["key"] = "value" }
```

```
{ _["key"] = "value" }
```

```
Underscore _;

struct Underscore {
  KeyValue& operator[](Value k);
};

struct KeyValue {
  KeyValue(Value k);
  KeyValue& operator=(Value v);
};

struct Value {
  Value(KeyValue& kv);
};
```

Result

```
var json = {
    _["number"] = 42,
    _["string"] = "vjeux",
    _["array"] = {1, 2, "three"},
    _["nested"] = {
        _["first"] = true,
        _["second"] = undefined
    }
};
```

```
var json = {
    number: 42,
    string: "vjeux",
    array: [1, 2, "three"],
    nested: {
        first: true,
        second: undefined
    }
};
```

Lambda Functions

C++11 Lambda

```
auto adder = [] (int x) {
  return [=] (int y) -> int {
    return x + y;
  };
};
auto add5 = adder(5);
// add5(1) == 6
```

```
var adder = function (x) {
  return function (y) {
    return x + y;
  };
};
var add5 = adder(5);
// add5(1) == 6
```

Macro :(

```
function (y) {
```

```
[=] (Value y) -> Value {
```

Macro :(

```
function (y) {
```

```
[=] (Value y) -> Value {
```

```
#define function (...)
        [=] (##__VA_ARGS__) -> Value
```

Macro :(

```
function (y) {
```

```
[=] (Value y) -> Value {
```

```
#define function (...)
        [=] (##__VA_ARGS__) -> Value
```

```
function (var y) {
```

```
[=] (var y) -> Value {
```

Store Me

```
struct Value {
  Value(std::function<Value ()> f);
  Value(std::function<Value (Value)> f);
  Value(std::function<Value (Value, Value)> f);
};
```

Call Me

```
Value Value::operator()(Value a) {
  if (n == 0) return f0();
  if (n == 1) return f1(a);
  if (n == 2) return f2(a, undefined);
}
```

Call Me

```
Value Value::operator()() {
  if (n == 0) return f0();
  if (n == 1) return f1(undefined);
  if (n == 2) return f2(undefined, undefined);
}
```

```
Value Value::operator()(Value a) {
  if (n == 0) return f0();
  if (n == 1) return f1(a);
  if (n == 2) return f2(a, undefined);
}


Value Value::operator()(Value a, Value b) {
  if (n == 0) return f0();
  if (n == 1) return f1(a);
  if (n == 2) return f2(a, b);
}
```

Prototypal Inheritance

Prototypal Inheritance

```
Value& Value::operator[](Value key) {
  if (this->map.contains(key) {
    return this->map[key];
  }
```

Prototypal Inheritance

```
Value& Value::operator[](Value key) {
  if (this->map.contains(key) {
    return this->map[key];
  }

  if (this->map.contains("__proto__")) {
    return this->map["__proto__"][key];
  }
```

Prototypal Inheritance

```
Value& Value::operator[](Value key) {
  if (this->map.contains(key) {
    return this->map[key];
  }

  if (this->map.contains("__proto__")) {
    return this->map["__proto__"][key];
  }

  return undefined;
}
```

Object.Create

```
var Object_Create = function (var parent) {
  return {
    _["__proto__"] = parent
  };
};
```

new

```
new (Point)(10, 20);
```

```
var new = function (var ctor) {
  return function () {
    var obj = { _["__proto__"] = ctor["prototype"] };
    ctor["apply"](obj, arguments);
    return obj;
  };
};
```

And more ...

Closure

```
var container = function (var data) {
  var secret = data;

  return {
    _["set"] = function (var x) {
      secret |= x;
      return undefined;
    },
    _["get"] = function () { return secret; }
  };
};

var a = container("secret-a");
var b = container("secret-b");

a["set"]("override-a");

std::cout << a["get"](); // override-a
std::cout << b["get"](); // secret-b
```

```
var container = function (data) {
  var secret = data;

  return {
    set: function (x) {
      secret = x;

    },
    get: function () { return secret; }
  };
};

var a = container("secret-a");
var b = container("secret-b");

a.set("override-a");

console.log(a.get()); // override-a
console.log(b.get()); // secret-b
```

This

```
var f = function (var x, var y) {
  std::cout << "this: " << this;
  this["x"] = x;
  this["y"] = y;
  return undefined;
};

// New creates a new object this
var a = new (f)(1, 2); // this: [function 40d0]

// Unbound call
var c = f(5, 6); // this: undefined

// Bound call
var obj = {42};
obj["f"] = f;
var d = obj["f"](1, 2); // this: [42]

// Call & Apply
var e = f["call"](obj, 1, 2); // this: [42]
```

```
var f = function (x, y) {
  console.log("this:", this);
  this["x"] = x;
  this["y"] = y;

};

// New creates a new object this
var a = new f(1, 2); // this: [object]

// Unbound call
var c = f(5, 6); // this: global object

// Bound call
var obj = [42];
obj["f"] = f;
var d = obj["f"](1, 2); // this: [42]

// Call & Apply
var e = f["call"](obj, 1, 2); // this: [42]=
```

```
#define function(...) [=] (var this, var arguments, ##__VA_ARGS__) -> Value
```

Operators

```
var repeat = function (var str, var times) {
  var ret = "";
  for (var i = 0; i < times; ++i) {
    ret += str + i;
  }
  return ret;
};

std::cout << repeat(" js++", 3) << std::endl;
// " js++0 js++1 js++2"
```

```
var repeat = function (str, times) {
  var ret = "";
  for (var i = 0; i < times; ++i) {
    ret += str + i;
  }
  return ret;
};

console.log(repeat(" js++", 3));
// " js++0 js++1 js++2"
```

Iteration

```
var object = {
    _["a"] = 1,
    _["b"] = 2,
    _["c"] = 3
};

for (var i in object) {
    std::cout << i << " - " << object[i];
}
// a - 1
// b - 2
// c - 3
```

```
var object = {
    "a": 1,
    "b": 2,
    "c": 3
};

for (var i in object) {
    console.log(i, object[i]);
}
// a - 1
// b - 2
// c - 3
```

#define in :

Exceptions

```
var go_die = function () {
  throw "Exception!";
};
try {
  go_die();
} catch (e) {
  std::cout << "Error: " << e;
}
// Error: Exception!
```

```
var go_die = function () {
  throw "Exception!";
};
try {
  go_die();
} catch (e) {
  console.log("Error:", e);
}
// Error: Exception!
```

```
#define throw throw _=
#define catch(e) catch(var e)
```

And less ...

Differences

- No `eval`
- No implicit `return undefined;`
- Different syntax for Array and Object initialization

- Only boxed version of primitive types
- C++ primitive types must sometimes be explicitly casted into Value
- No variable hoisting
- Control structures are blocks
- Cannot redeclare variables in the same scope
- No automatic global without `var`
- Function arguments must be preceded by `var`
- `return;` is not valid
- `new` requires parenthesis around the constructor function

Differences

- No dot notation `.` for object property access
- The empty object notation `{}` is treated as `undefined`

- Use `|=` instead of `=` to modify a closure reference
- `in`, `===` and `!==` renamed in `of`, `is` and `isnt`
- `typeof`, `delete` are functions instead of operators
- `switch case` construction with integers only
- Implementation dependent limit for number of arguments
- No `break label;` form
- No automatic semi-column `;` insertion
- No named functions
- No string literal with simple quote `'...'`
- No short regex notation `/.../`
- No `>>>`, `>>>=`, `void` operators

Differences

No `with` :(

Differences

No `with` :(

Yes `goto`!

Conclusion

- Is it useful?
  - I don't know.
- Is it fun?
  - Yes, certainly!

 github.com/**vjeux**/jspp

 blog.**vjeux**.com

 twitter.com/**vjeux**