COMP6080

Transpilation

Presented by
Nic Barker

Definition

What is transpilation?

Definition

Compilation

source code -> machine code

Transpilation

source code -> source code

Definition

Compilation

C++ -> Machine code

Transpilation

C++ -> Javascript 

Transpilation

Why do we need it?

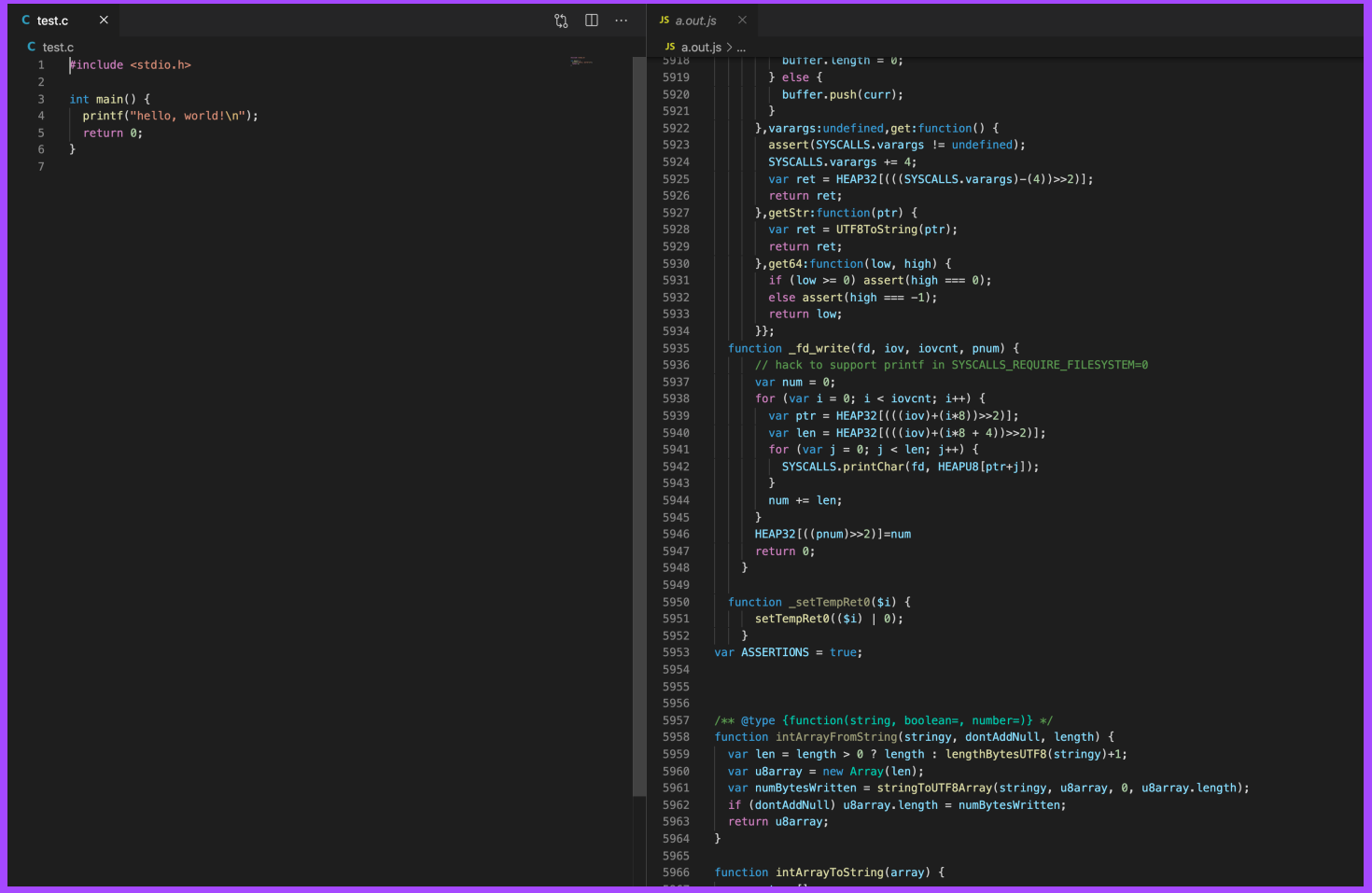
What are the use cases?

① Other language -> Javascript

Javascript is the only natively supported language in the browser**. This means that by default if you want to use a different language to write your frontend web apps, you need to transpile to JS.

**yes there's wasm now, it's still not totally well supported / tooled

Source (C++) -> Transpiled (Javascript)



test.c

```c
#include <stdio.h>

int main() {
  printf("hello, world!\n");
  return 0;
}

```

a.out.js

```
        buffer.length = 0;
      } else {
        buffer.push(curr);
      }
    },varargs:undefined,get:function() {
      assert(SYSCALLS.varargs != undefined);
      SYSCALLS.varargs += 4;
      var ret = HEAP32[(((SYSCALLS.varargs)-(4))>>2)];
      return ret;
    },getStr:function(ptr) {
      var ret = UTF8ToString(ptr);
      return ret;
    },get64:function(low, high) {
      if (low >= 0) assert(high === 0);
      else assert(high === -1);
      return low;
    }};
  function _fd_write(fd, iov, iovcnt, pnum) {
    // hack to support printf in SYSCALLS_REQUIRE_FILESYSTEM=0
    var num = 0;
    for (var i = 0; i < iovcnt; i++) {
      var ptr = HEAP32[(((iov)+(i*8))>>2)];
      var len = HEAP32[(((iov)+(i*8 + 4))>>2)];
      for (var j = 0; j < len; j++) {
        SYSCALLS.printChar(fd, HEAPU8[ptr+j]);
      }
      num += len;
    }
    HEAP32[((pnum)>>2)]=num
    return 0;
  }

  function _setTempRet0($i) {
    setTempRet0(($i) | 0);
  }
var ASSERTIONS = true;



/** @type {function(string, boolean=, number=)} */
function intArrayFromString(stringy, dontAddNull, length) {
  var len = length > 0 ? length : lengthBytesUTF8(stringy)+1;
  var u8array = new Array(len);
  var numBytesWritten = stringToUTF8Array(stringy, u8array, 0, u8array.length);
  if (dontAddNull) u8array.length = numBytesWritten;
  return u8array;
}

function intArrayToString(array) {
```

② New Javascript -> Old Javascript

New features become available in Javascript that aren't supported in old browsers. We can **transpile** our new Javascript (e.g. **native js Class**) to compatible older APIs, while still using new features when we write our code.

Source -> Transpiled (no class support)

```
class Test {
  classMethod() {
    console.log('test');
  }
}
```

```
"use strict";

function _instanceof(left, right) { if (right != null && typeof Symbol !== "undefined" && right[Symbol.hasInstance]) { return !!right[Symbol.hasInstance](left); } else { return left instanceof right; } }

function _classCallCheck(instance, Constructor) { if (!_instanceof(instance, Constructor)) { throw new TypeError("Cannot call a class as a function"); } }

function _defineProperties(target, props) { for (var i = 0; i < props.length; i++) { var descriptor = props[i]; descriptor.enumerable = descriptor.enumerable || false; descriptor.configurable = true; if ("value" in descriptor) descriptor.writable = true; Object.defineProperty(target, descriptor.key, descriptor); } }

function _createClass(Constructor, protoProps, staticProps) { if (protoProps) _defineProperties(Constructor.prototype, protoProps); if (staticProps) _defineProperties(Constructor, staticProps); return Constructor; }

var Test = /*#__PURE__*/function () {
  function Test() {
    _classCallCheck(this, Test);
  }

  _createClass(Test, [{
    key: "classMethod",
    value: function classMethod() {
      console.log('test');
    }
  }]);

  return Test;
}();
```

The why

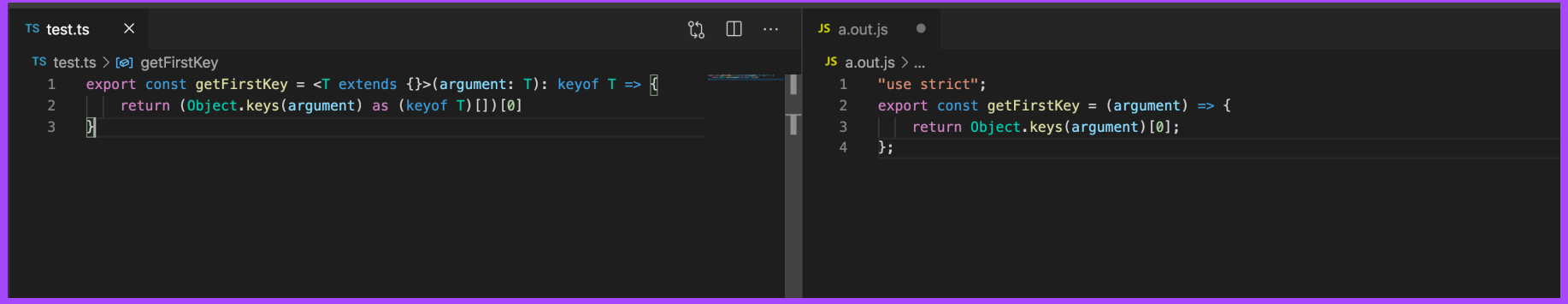
You can try compiling new -> old Javascript at

https://babeljs.io/repl

③ Almost Javascript -> Javascript

Flavours of Javascript have been created for syntactic sugar or **type safety.** For example, **Coffeescript** or **Typescript**. These are almost JS (much closer than C++) but still need to be **transpiled** to vanilla JS to run in the browser.

Typescript -> Javascript



Try the typescript compiler:

typescriptlang.org/play

JSX transformation

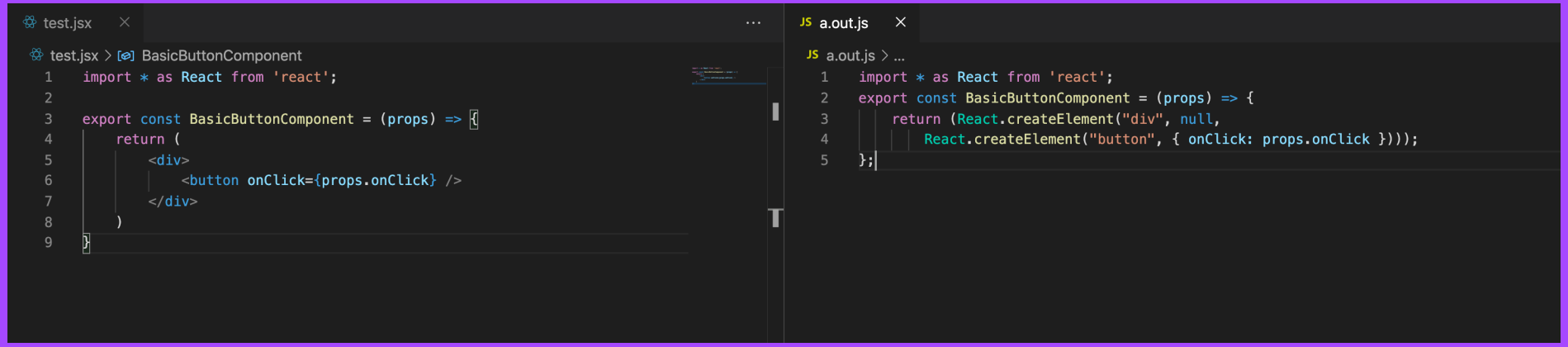
React is just Javascript. It doesn't use any string or HTML templating.

Under the hood when you write something like
**component = <button /> **

what you're actually writing is:

```
component = React.createElement("div", null,
React.createElement("button", null));
```

JSX -> Javascript



The why

Minification + Obfuscation

Javascript **Minification** and / or **Obfuscation** are both technically types of **transpilation.**

Minification / Obfuscation takes our existing code and condenses it, removing whitespace, removing comments, shortening variable names and in some cases even automatically refactoring control flow to save network transfer when downloading scripts.

This process also makes it more difficult for competitors or malicious actors to reverse engineer our source code.

Min / Obfs example: Google "Closure Compiler" in advanced mode can determine that the function **hello** and the **user** object are only used once, and removes them completely

over a large application, this can save a lot of bytes

Try it out:

https://closure-compiler.appspot.com/home

How does transpiling generally work?

Before we deploy our Javascript code to production, we run a **build** step using tooling like **webpack** or **rollup** or **gulp**.

The how

Tools like webpack are usually a chain of plugins that handle lots of different file types and transpilation steps. In webpack they are called **loaders** but have different names ("plugins", etc) in other tools.

Caveats

Transpiling isn't free. It takes time and will slow down your development / deployment experience if you aren't careful.

Heavy transpilation means the code executing in the browser may be difficult to debug. You will need good **source map** support to help you understand how the transpiled code maps back to the source code.