```
/*
        This file is part of solidity.

        solidity is free software: you can redistribute it and/or modify
        it under the terms of the GNU General Public License as published by
        the Free Software Foundation, either version 3 of the License, or
        (at your option) any later version.

        solidity is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
        GNU General Public License for more details.

        You should have received a copy of the GNU General Public License
        along with solidity.  If not, see <http://www.gnu.org/licenses/>.
*/
// SPDX-License-Identifier: GPL-3.0
/**
 * Unit tests for Solidity's ABI decoder.
 */

#include <functional>
#include <string>
#include <tuple>
#include <boost/test/unit_test.hpp>
#include <liblangutil/Exceptions.h>
#include <test/libsolidity/SolidityExecutionFramework.h>

#include <test/libsolidity/ABITestsCommon.h>

using namespace std;
using namespace std::placeholders;
using namespace solidity::test;

namespace solidity::frontend::test
{

BOOST_FIXTURE_TEST_SUITE(ABIDecoderTest, SolidityExecutionFramework)

BOOST_AUTO_TEST_CASE(value_types)
{
        string sourceCode = R"(
                contract C {
                        function f(uint a, uint16 b, uint24 c, int24 d, bytes3 x, bool e, C
g) public returns (uint) {
                                if (a != 1) return 1;
                                if (b != 2) return 2;
                                if (c != 3) return 3;
                                if (d != 4) return 4;
                                if (x != "abc") return 5;
                                if (e != true) return 6;
                                if (g != this) return 7;
                                return 20;
                        }
                }
        )";
        BOTH_ENCODERS(
                compileAndRun(sourceCode);
                ABI_CHECK(callContractFunction(
                        "f(uint256,uint16,uint24,int24,bytes3,bool,address)",
                        1, 2, 3, 4, string("abc"), true, m_contractAddress
                ), encodeArgs(u256(20)));
        )
}
```

```
BOOST_AUTO_TEST_CASE(decode_from_memory_simple)
{
	string sourceCode = R"(
		contract C {
			uint public _a;
			uint[] public _b;
			constructor(uint a, uint[] memory b) {
				_a = a;
				_b = b;
			}
		}
	)";
	BOTH_ENCODERS(
		compileAndRun(sourceCode, 0, "C", encodeArgs(
			7, 0x40,
			// b
			3, 0x21, 0x22, 0x23
		));
		ABI_CHECK(callContractFunction("_a()"), encodeArgs(7));
		ABI_CHECK(callContractFunction("_b(uint256)", 0), encodeArgs(0x21));
		ABI_CHECK(callContractFunction("_b(uint256)", 1), encodeArgs(0x22));
		ABI_CHECK(callContractFunction("_b(uint256)", 2), encodeArgs(0x23));
		ABI_CHECK(callContractFunction("_b(uint256)", 3), encodeArgs());
	)
}

BOOST_AUTO_TEST_CASE(decode_function_type)
{
	string sourceCode = R"(
		contract D {
			function () external returns (uint) public _a;
			constructor(function () external returns (uint) a) {
				_a = a;
			}
		}
		contract C {
			function f() public returns (uint) {
				return 3;
			}
			function g(function () external returns (uint) _f) public returns
(uint) {
				return _f();
			}
			// uses "decode from memory"
			function test1() public returns (uint) {
				D d = new D(this.f);
				return d._a()();
			}
			// uses "decode from calldata"
			function test2() public returns (uint) {
				return this.g(this.f);
			}
		}
	)";
	BOTH_ENCODERS(
		compileAndRun(sourceCode, 0, "C");
		ABI_CHECK(callContractFunction("test1()"), encodeArgs(3));
		ABI_CHECK(callContractFunction("test2()"), encodeArgs(3));
	)
}

BOOST_AUTO_TEST_CASE(decode_function_type_array)
{
	string sourceCode = R"(
```

```
                contract D {
                        function () external returns (uint)[] public _a;
                        constructor(function () external returns (uint)[] memory a) {
                                _a = a;
                        }
                }
                contract E {
                        function () external returns (uint)[3] public _a;
                        constructor(function () external returns (uint)[3] memory a) {
                                _a = a;
                        }
                }
                contract C {
                        function f1() public returns (uint) {
                                return 1;
                        }
                        function f2() public returns (uint) {
                                return 2;
                        }
                        function f3() public returns (uint) {
                                return 3;
                        }
                        function g(function () external returns (uint)[] memory _f, uint i)
public returns (uint) {
                                return _f[i]();
                        }
                        function h(function () external returns (uint)[3] memory _f, uint i)
public returns (uint) {
                                return _f[i]();
                        }
                        // uses "decode from memory"
                        function test1_dynamic() public returns (uint) {
                                function () external returns (uint)[] memory x = new
function() external returns (uint)[](4);
                                x[0] = this.f1;
                                x[1] = this.f2;
                                x[2] = this.f3;
                                D d = new D(x);
                                return d._a(2)();
                        }
                        function test1_static() public returns (uint) {
                                E e = new E([this.f1, this.f2, this.f3]);
                                return e._a(2)();
                        }
                        // uses "decode from calldata"
                        function test2_dynamic() public returns (uint) {
                                function () external returns (uint)[] memory x = new
function() external returns (uint)[](3);
                                x[0] = this.f1;
                                x[1] = this.f2;
                                x[2] = this.f3;
                                return this.g(x, 0);
                        }
                        function test2_static() public returns (uint) {
                                return this.h([this.f1, this.f2, this.f3], 0);
                        }
                }
        )";
        BOTH_ENCODERS(
                compileAndRun(sourceCode, 0, "C");
                ABI_CHECK(callContractFunction("test1_static()"), encodeArgs(3));
                ABI_CHECK(callContractFunction("test1_dynamic()"), encodeArgs(3));
                ABI_CHECK(callContractFunction("test2_static()"), encodeArgs(1));
                ABI_CHECK(callContractFunction("test2_dynamic()"), encodeArgs(1));
        )
```

```
}

BOOST_AUTO_TEST_CASE(decode_from_memory_complex)
{
	string sourceCode = R"(
		contract C {
			uint public _a;
			uint[] public _b;
			bytes[2] public _c;
			constructor(uint a, uint[] memory b, bytes[2] memory c) {
				_a = a;
				_b = b;
				_c = c;
			}
		}
	)";
	NEW_ENCODER(
		compileAndRun(sourceCode, 0, "C", encodeArgs(
			7, 0x60, 7 * 0x20,
			// b
			3, 0x21, 0x22, 0x23,
			// c
			0x40, 0x80,
			8, string("abcdefgh"),
			52, string("ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ")
		));
		ABI_CHECK(callContractFunction("_a()"), encodeArgs(7));
		ABI_CHECK(callContractFunction("_b(uint256)", 0), encodeArgs(0x21));
		ABI_CHECK(callContractFunction("_b(uint256)", 1), encodeArgs(0x22));
		ABI_CHECK(callContractFunction("_b(uint256)", 2), encodeArgs(0x23));
		ABI_CHECK(callContractFunction("_b(uint256)", 3), encodeArgs());
		ABI_CHECK(callContractFunction("_c(uint256)", 0), encodeArgs(0x20, 8,
string("abcdefgh")));
		ABI_CHECK(callContractFunction("_c(uint256)", 1), encodeArgs(0x20, 52,
string("ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ")));
		ABI_CHECK(callContractFunction("_c(uint256)", 2), encodeArgs());
	)
}

BOOST_AUTO_TEST_CASE(short_input_value_type)
{
	string sourceCode = R"(
		contract C {
			function f(uint a, uint b) public pure returns (uint) { return a; }
		}
	)";
	BOTH_ENCODERS(
		compileAndRun(sourceCode);
		ABI_CHECK(callContractFunction("f(uint256,uint256)", 1, 2), encodeArgs(1));
		ABI_CHECK(callContractFunctionNoEncoding("f(uint256,uint256)", bytes(64, 0)),
encodeArgs(0));
		ABI_CHECK(callContractFunctionNoEncoding("f(uint256,uint256)", bytes(63, 0)),
encodeArgs());
	)
}

BOOST_AUTO_TEST_CASE(short_input_array)
{
	string sourceCode = R"(
		contract C {
			function f(uint[] memory a) public pure returns (uint) { return 7; }
		}
	)";
	BOTH_ENCODERS(
		compileAndRun(sourceCode);
```

```
                ABI_CHECK(callContractFunctionNoEncoding("f(uint256[])", encodeArgs(0x20,
0)), encodeArgs(7));
                ABI_CHECK(callContractFunctionNoEncoding("f(uint256[])", encodeArgs(0x20,
1)), encodeArgs());
                ABI_CHECK(callContractFunctionNoEncoding("f(uint256[])", encodeArgs(0x20, 1)
+ bytes(31, 0)), encodeArgs());
                ABI_CHECK(callContractFunctionNoEncoding("f(uint256[])", encodeArgs(0x20, 1)
+ bytes(32, 0)), encodeArgs(7));
                ABI_CHECK(callContractFunctionNoEncoding("f(uint256[])", encodeArgs(0x20, 2,
5, 6)), encodeArgs(7));
        )
}

BOOST_AUTO_TEST_CASE(short_dynamic_input_array)
{
        string sourceCode = R"(
                contract C {
                        function f(bytes[1] memory a) public pure returns (uint) { return 7;
}
                }
        )";
        NEW_ENCODER(
                compileAndRun(sourceCode);
                ABI_CHECK(callContractFunctionNoEncoding("f(bytes[1])", encodeArgs(0x20)),
encodeArgs());
        )
}

BOOST_AUTO_TEST_CASE(short_input_bytes)
{
        string sourceCode = R"(
                contract C {
                        function e(bytes memory a) public pure returns (uint) { return 7; }
                        function f(bytes[] memory a) public pure returns (uint) { return 7; }
                }
        )";
        NEW_ENCODER(
                compileAndRun(sourceCode);
                ABI_CHECK(callContractFunctionNoEncoding("e(bytes)", encodeArgs(0x20, 7) +
bytes(5, 0)), encodeArgs());
                ABI_CHECK(callContractFunctionNoEncoding("e(bytes)", encodeArgs(0x20, 7) +
bytes(6, 0)), encodeArgs());
                ABI_CHECK(callContractFunctionNoEncoding("e(bytes)", encodeArgs(0x20, 7) +
bytes(7, 0)), encodeArgs(7));
                ABI_CHECK(callContractFunctionNoEncoding("e(bytes)", encodeArgs(0x20, 7) +
bytes(8, 0)), encodeArgs(7));
                ABI_CHECK(callContractFunctionNoEncoding("f(bytes[])", encodeArgs(0x20, 1,
0x20, 7) + bytes(5, 0)), encodeArgs());
                ABI_CHECK(callContractFunctionNoEncoding("f(bytes[])", encodeArgs(0x20, 1,
0x20, 7) + bytes(6, 0)), encodeArgs());
                ABI_CHECK(callContractFunctionNoEncoding("f(bytes[])", encodeArgs(0x20, 1,
0x20, 7) + bytes(7, 0)), encodeArgs(7));
                ABI_CHECK(callContractFunctionNoEncoding("f(bytes[])", encodeArgs(0x20, 1,
0x20, 7) + bytes(8, 0)), encodeArgs(7));
        )
}

BOOST_AUTO_TEST_CASE(validation_int_inside_arrays)
{
        string sourceCode = R"(
                contract C {
                        enum E { A, B }
                        function f(uint16[] memory a) public pure returns (uint r) { assembly
{ r := mload(add(a, 0x20)) } }
                        function g(int16[] memory a) public pure returns (uint r) { assembly
```

```
{ r := mload(add(a, 0x20)) } }
                        function h(E[] memory a) public pure returns (uint r) { assembly { r
:= mload(add(a, 0x20)) } }
                }
        )";
        NEW_ENCODER(
                compileAndRun(sourceCode);
                ABI_CHECK(callContractFunction("f(uint16[])", 0x20, 1, 7), encodeArgs(7));
                ABI_CHECK(callContractFunction("g(int16[])", 0x20, 1, 7), encodeArgs(7));
                ABI_CHECK(callContractFunction("f(uint16[])", 0x20, 1, u256("0xffff")),
encodeArgs(u256("0xffff")));
                ABI_CHECK(callContractFunction("g(int16[])", 0x20, 1, u256("0xffff")),
encodeArgs());
                ABI_CHECK(callContractFunction("f(uint16[])", 0x20, 1, u256("0x1ffff")),
encodeArgs());
                ABI_CHECK(callContractFunction("g(int16[])", 0x20, 1, u256("0x10fff")),
encodeArgs());
                ABI_CHECK(callContractFunction("h(uint8[])", 0x20, 1, 0),
encodeArgs(u256(0)));
                ABI_CHECK(callContractFunction("h(uint8[])", 0x20, 1, 1),
encodeArgs(u256(1)));
                ABI_CHECK(callContractFunction("h(uint8[])", 0x20, 1, 2), encodeArgs());
        )
}

BOOST_AUTO_TEST_CASE(validation_function_type)
{
        string sourceCode = R"(
                contract C {
                        function f(function () external) public pure returns (uint r) { r =
1; }
                        function g(function () external[] memory) public pure returns (uint
r) { r = 2; }
                        function h(function () external[] calldata) external pure returns
(uint r) { r = 3; }
                        function i(function () external[] calldata a) external pure returns
(uint r) { a[0]; r = 4; }
                }
        )";
        bool newDecoder = false;
        string validFun{"012345678901234567 89abcd"};
        string invalidFun{"012345678901234567 89abcdX"};
        BOTH_ENCODERS(
                compileAndRun(sourceCode);
                ABI_CHECK(callContractFunction("f(function)", validFun), encodeArgs(1));
                ABI_CHECK(callContractFunction("f(function)", invalidFun), newDecoder ?
bytes{} : encodeArgs(1));
                ABI_CHECK(callContractFunction("g(function[])", 0x20, 1, validFun),
encodeArgs(2));
                ABI_CHECK(callContractFunction("g(function[])", 0x20, 1, invalidFun),
newDecoder ? bytes{} : encodeArgs(2));
                ABI_CHECK(callContractFunction("h(function[])", 0x20, 1, validFun),
encodeArgs(3));
                // No failure because the data is not accessed.
                ABI_CHECK(callContractFunction("h(function[])", 0x20, 1, invalidFun),
encodeArgs(3));
                ABI_CHECK(callContractFunction("i(function[])", 0x20, 1, validFun),
encodeArgs(4));
                ABI_CHECK(callContractFunction("i(function[])", 0x20, 1, invalidFun),
newDecoder ? bytes{} : encodeArgs(4));
                newDecoder = true;
        )
}

BOOST_AUTO_TEST_CASE(struct_short)
```

```
{
	string sourceCode = R"(
		contract C {
			struct S { int a; uint b; bytes16 c; }
			function f(S memory s) public pure returns (S memory q) {
				q = s;
			}
		}
	)";
	NEW_ENCODER(
		compileAndRun(sourceCode, 0, "C");
		ABI_CHECK(
			callContractFunction("f((int256,uint256,bytes16))", 0xff010,
0xff0002, "abcd"),
			encodeArgs(0xff010, 0xff0002, "abcd")
		);
		ABI_CHECK(
			callContractFunctionNoEncoding("f((int256,uint256,bytes16))",
encodeArgs(0xff010, 0xff0002) + bytes(32, 0)),
			encodeArgs(0xff010, 0xff0002, 0)
		);
		ABI_CHECK(
			callContractFunctionNoEncoding("f((int256,uint256,bytes16))",
encodeArgs(0xff010, 0xff0002) + bytes(31, 0)),
			encodeArgs()
		);
	)
}

BOOST_AUTO_TEST_CASE(complex_struct)
{
	string sourceCode = R"(
		contract C {
			enum E {A, B, C}
			struct T { uint x; E e; uint8 y; }
			struct S { C c; T[] t;}
			function f(uint a, S[2] memory s1, S[] memory s2, uint b) public
returns
					(uint r1, C r2, uint r3, uint r4, C r5, uint r6, E
r7, uint8 r8) {
				r1 = a;
				r2 = s1[0].c;
				r3 = b;
				r4 = s2.length;
				r5 = s2[1].c;
				r6 = s2[1].t.length;
				r7 = s2[1].t[1].e;
				r8 = s2[1].t[1].y;
			}
		}
	)";
	NEW_ENCODER(
		compileAndRun(sourceCode, 0, "C");
		string sig = "f(uint256,(address,(uint256,uint8,uint8)[])[2],(address,
(uint256,uint8,uint8)[])[],uint256)";
		bytes args = encodeArgs(
			7, 0x80, 0x1e0, 8,
			// S[2] s1
			0x40,
			0x100,
			// S s1[0]
			m_contractAddress,
			0x40,
			// T s1[0].t
			1, // length
```

```
                        // s1[0].t[0]
                        0x11, 1, 0x12,
                        // S s1[1]
                        0, 0x40,
                        // T s1[1].t
                        0,
                        // S[] s2 (0x1e0)
                        2, // length
                        0x40, 0xa0,
                        // S s2[0]
                        0, 0x40, 0,
                        // S s2[1]
                        0x1234, 0x40,
                        // s2[1].t
                        3, // length
                        0, 0, 0,
                        0x21, 2, 0x22,
                        0, 0, 0
                );
                ABI_CHECK(callContractFunction(sig, args), encodeArgs(7, m_contractAddress,
8, 2, 0x1234, 3, 2, 0x22));
                // invalid enum value
                args.data()[0x20 * 28] = 3;
                ABI_CHECK(callContractFunction(sig, args), encodeArgs());
        )
}

BOOST_AUTO_TEST_SUITE_END()

} // end namespaces
```