VALUES AND REFERENCES

tbd

Florian Warg, Max Staff

April 19, 2017

WHAT ARE VALUES?

- values are expressions in a normal form
- cannot be reduced or evaluated any further

```
1 + 2 // not a value (can be reduced)
3     // value (in normal form)
```

- variables can hold values
- value is independent from its location

```
int x = 2, y = 4, z = 4;
x + y == x + z // 2 + 4 == 2 + 4
```

LVALUES

- lvalues are expressions which can be used on the left side of an assignment operation
- i.e. an lvalue has a memory address

```
int x = 1; // x holds value 1
x = 2;     // x is lvalue
&x;        // might give us the lvalue 0xFA02E1
2 = x;     // compiler error: 2 is not an lvalue
```

RVALUES

- rvalues are non-lvalues or temporary lvalues
- can only be used on the right side of an assignment operation
- temporary variables and literals are rvalues
- non-temporary variables hold lvalues and rvalues

```
int x = 2; // non-temporary variable x
&x;        // address of x is its lvalue
x;         // 2 is rvalue of x
```

REFERENCES

- a reference is an alias for an existing value
- there are references for lvalues and rvalues
- references are not necessarily objects
- i.e. there are no pointers, arrays or references to references
- give access to a variable without copying its value
- allow modification of local variables in other scopes

LVALUE REFERENCES

- alias which refers to an lvalue

```
1 int x = 42;
2 int& lx = x; // create lvalue reference to x
3 ++lx;        // use alias instead of x
4 cout << x;   // what is printed here?
```

- functions taking lvalue references can modify local variables

```
1 void add2(int& ref) { ref += 2; }
2 int x = 10;
3 add2(x); // ref is initialized with x
4 cout << x;
```

RVALUE REFERENCES

- alias which refers to an rvalue

```
1 string&& sr = "Hello";  // create temporary
2 cout << sr;
```

- used to implement move semantics and perfect forwarding
- move temporaries into function instead of copying their values

```
1 void sinkStr(string&& tmp) { cout << tmp; }
2 sinkStr("Hello World!");
```

POINTERS

- pointers are data types that hold addresses as their values
- can be dereferenced: interpret data at address as value
- can be used for pointer arithmetics

```
1  int x = 42;
2  int* px = &x; // px holds lvalue (address) of x
3  cout << px;   // print address
4  *px = 43;     // set content of variable at &x
5  cout << *px   // print 43 (value of x)
```

POINTER ARITHMETICS