# Smart Pointers

Victor Eijkhout, Harika Gurram,
Je'aime Powell, Charley Dey

Fall 2018

# Creating a shared pointer

Allocation and pointer in one:

```
shared_ptr<Obj> X =
    make_shared<Obj>( /* constructor args */ );
  // or:
auto X = make_shared<Obj>( /* args */ );
  // or:
auto X = shared_ptr<Obj>( new Obj(/* args */) );

X->method_or_member;
```

# Simple example

**Code:**

```
class HasX {
private:
  double x;
public:
  HasX( double x) : x(x) {};
  auto &val() { return x; };
};

int main() {
  auto X = make_shared<HasX>(5);
  cout << X->val() << endl;
  X->val() = 6;
  cout << X->val() << endl;
```

**Output**
**[pointer] pointx:**

```
5
6
```

# Linked lists

The prototypical example use of pointers is in linked lists. Let a class Node be given:

```
class Node {
private:
  int datavalue{0};
  shared_ptr<Node> tail_ptr{nullptr};
public:
  Node() {}
  Node(int value) { datavalue = value; };
  void set_tail( shared_ptr<Node> tail ) {
    tail_ptr = tail; };
```

```
void print() {
  cout << datavalue;
  if (has_next()) {
cout << ", "; tail_ptr->print();
  }
};
```

# List usage

Example use:

**Code:**

```
auto
  first = make_shared<Node>(23),
  second = make_shared<Node>(45);
first->set_tail(second);
cout << "List length: "
     << first->list_length() << endl;
first->print();
```

**Output**
**[tree] simple:**

```
List length: 2
23, 45
```

# Linked lists and recursion

Many operations on linked lists can be done recursively:

```
int Node::list_length() {
  if (!has_next()) return 1;
  else return 1+tail_ptr->list_length();
};
```

# Exercise 1

Write a recursive append method that appends a node to the end of a list:

**Code:**

```
auto
  first = make_shared<Node>(23),
  second = make_shared<Node>(45),
  third = make_shared<Node>(32);
first->append(second);
first->append(third);
first->print();
```

**Output**
**[tree] append:**

```
23, 45, 32
```

# Exercise 2

Write a recursive `insert` method that inserts a node in a list, such that the list stays sorted:

**Code:**

```
auto
  first = make_shared<Node>(23),
  second = make_shared<Node>(45),
  third = make_shared<Node>(32);
first->insert(second);
first->insert(third);
first->print();
```

**Output**
**[tree] insert:**

```
23, 32, 45
```