



# CS 11 C++ track: lecture 5

---

- Today:
  - Member initialization lists
  - Linked lists
  - `friend` functions



# Member initialization lists (1)

---

```
class Foo {  
    private:  
        int x, y;  
    public:  
        Foo(int nx, int ny) {  
            x = nx; y = ny;  
        }  
        // other methods...  
};
```



# Member initialization lists (2)

---

```
class Bar {  
    private:  
        int w, z;  
        Foo f;  
    public:  
        Bar(int nw, int nz)  
            : w(nw), z(nz), f(nw, nz) {}  
  
    // other methods...  
};
```



## Member initialization lists (3)

---

- Initialization list not essential for primitive types...
- ...but needed to pass arguments to constructors of member objects



# Member initialization lists (4)

---

```
class Bar {
    private:
        int w, z;
        Foo *f; // now a pointer!
    public:
        Bar(int nw, int nz) {
            w = nw; z = nz;
            f = new Foo(nw, nz);
        }
};
```



# Member initialization lists (5)

---

```
class Bar {
    private:
        int w, z;
        Foo f;
    public:
        Bar(int nw, int nz) {
            w = nw; z = nz;
            Foo f2(nw, nz);
            f = f2; // inefficient; copies f2
        } // f2 deallocated here
};
```



# Linked lists (1)

---

```
struct node {  
    int data;  
    node *next;  
    node(int d, node *n)  
        : data(d), next(n) { }  
};
```



## Linked lists (2)

---

- Create a linked list:

```
Node *prev = 0; // null pointer
Node *head = 0;
for (int i = 10; i > 0; i--) {
    head = new Node(i, prev);
    prev = head;
}
```



## Linked lists (3)

---

- Create a linked list:

```
// Simpler:
```

```
Node *head = 0;
```

```
for (int i = 10; i > 0; i--) {
```

```
    head = new Node(i, head);
```

```
}
```



# Linked lists (4)

---

- Print a linked list:

```
for (Node *n = head; n; n = n->next) {  
    cout << n->data << endl;  
}
```

- iterate until reach a null pointer (`n == 0`)



## Linked lists (5)

---

- Free a linked list:

```
// This WON'T work:
```

```
for (Node *n = head; n; n = n->next) {  
    delete n;  
}
```

- Why won't it work?



# Linked lists (6)

---

- Free a linked list:

```
// This WILL work:
```

```
Node *n = head;
```

```
while (n != 0) {
```

```
    Node *next = n->next;
```

```
    delete n;
```

```
    n = next;
```

```
}
```



# friend functions (1)

---

- Want to be able to say:

```
Matrix m(2, 3);
```

```
cout << m << endl;
```

- but this really means:

```
cout.operator<<(m); // etc.
```

- No such method exists in cout's class (`ostream`)!
- Can't add methods to library classes



## friend functions (2)

---

- Can define overloaded function:

```
ostream& operator<<(ostream& os,  
    const Matrix& m);
```

- return type is for chaining
- `os` is `cout`
- `m` is the `Matrix` to output



## friend functions (3)

---

- Problem: not a member fn of **Matrix**
  - so can't access private **Matrix** data
  - could rely on public accessor fns
  - or could make it a **friend** of the **Matrix** class



# friend functions (4)

---

```
class Matrix {  
    // ...  
    friend ostream& operator<<(  
        ostream& os, const Matrix& m);  
    // ...  
};
```



## friend functions (5)

---

```
// Definition of friend function:
ostream& operator<<(
    ostream& os, const Matrix& m)
{
    // can use private Matrix data:
    os << "Matrix(" << m.rows
        << ", " << m.cols << ")";
    return os;
}
```



# friend functions (6)

---

- Usage:

```
Matrix m(2, 3);
```

```
cout << m << endl;
```

```
// Prints: "Matrix(2, 3)".
```

```
// Could define to print elems e.g.
```

```
// "Matrix(2, 3):{{0,0,0},{0,0,0}}"
```



# Next week

---

- Default function arguments
- `friend` classes
- Introduction to exception handling