CSCI 2270
Data Structures and Algorithms
Lecture 25

Elizabeth White
elizabeth.white@colorado.edu
Office hours: ECCS 128
Wed 1-2pm
Fri 2-3pm

Administrivia

Today: Template functions

More big_number functions

Linked lists revisited

First doubly linked list stored integers

```
struct node {
        int data;
        node* next;
        node* prev;
};
```

Second one (for hw2) stored characters

```
struct node {
        char data;
        node* next;
        node* prev;
};
```

Given enough search and replace, we can make lists of anything

First doubly linked list stored integers

```
struct node {
        int data;
        node* next;
        node* prev;
};
void print_list(const node* head_ptr);
```

Second one (for hw2) stored characters

```
struct node {
        char data;
        node* next;
        node* prev;
};
void prchar_list(const node* head_ptr);
```

oops

Or we can use templates

Tell the linked list: you store some type of data,

And we'll tell you what it is later (!)

```
struct node {
        ItemType data;          // generic data type
        node* next;
        node* prev;
};
```

In the test program (the main()), we have to specify the type of data the list is storing inside <>…

```
node<int>* head_ptr1 = nullptr;
```

Good things about templates

Today: rewrite the DLL as a template class

Then you can have lists of strings, lists of integers, lists of big_numbers, etc… all using this code.

Big advantage: generality. Write one list, get all data types free.

Completely general? No. We need to be able to compare the data we're storing:

```
while (cursor != nullptr && cursor->next != nullptr
                         && cursor->next->data < payload)
```

Bad things about templates

Can’t compile without a main program

Give lots more distressing error messages for same number of errors

Not always implemented efficiently under the hood