Casting

- C programs allow unrestricted casting from one type to another
  - Some casts are conversions
    - E.g., between different numeric types
  - Some casts restrict or reveal information
    - E.g. between pointers to structs with more or fewer fields
    - void* is the implicit "supertype" of all pointers, akin to Object in Java
  - Some casts just reinterpret the bits
    - E.g. between an int and a pointer

"Generic" code

- One use for casting to write one piece of code that's generic across many possible client types
- E.g., a List of things, where we don't want to restrict what kind of things we can store
  - In Java: use Object as "universal" type, cast arguments to Object (implicitly) when put in and cast back to real type (explicitly) when take out
    - Except that primitive types aren't Objects ☹
  - In C: long, or void*, or unions, or ...
  - In C++: templates

Example

```
struct Link {
  void* data;
  Link* next;
};
Link* addFirst(Link* list, void* data) { ... }
...
Link* myList = NULL;
myList = addFirst(myList, "a string");
char* firstElem = (char*) myList->data; //cast
```


A taste of templates

```
template <class T> struct Link {
  T data;
  Link<T>* next;
};
template <class T>
 Link<T>* addFirst(Link<T>* list, T data) {...}
...
Link<const char*>* myList = NULL;
myList = addFirst(myList, "a string");
const char* firstElem = myList->data; //no cast
```


Multiple source files

- Bigger programs need to be broken up into multiple files
  - How does one file get access to things defined in other files?
- In Java:
  - User just writes .java source files
  - Compiler automatically looks in other .class files to see what they publicly export
- In C:
  - User needs to write both .c source files and .h *header files*

Header files

- Header files (redundantly) declare *public* functions and types that will be accessed by other .c files
  - Anything not declared is implicitly private to the .c file
- Each .c file #include's the .h files of the things it accesses
  - That way it sees the declarations of those things
- Anything not declared in .h files can't be accessed by other .c files (unless they cheat)

Example

- In link.h:

```
struct Link; // hide its body; allow Link* only
Link* addFirst(Link* list, void* data);
        // no {...} body! a prototype
... // other functions here
```

- In link.c:

```
#include "link.h"   // to verify consistency
... // full definitions of struct Link, addFirst, etc.
```

- In client.c:

```
#include "link.h"   // gain access to public decls
... // uses of Link*, calls of addFirst, etc.
```


Input/output library functions

- printf has many ways of producing formatted output
  - cout is C++ alternative that many prefer
- scanf is way to get input from stdin
  - cin is C++ alternative
  - note: pass *pointers* as arguments
- look up fopen, fread, fwrite, fclose to do file I/O

More useful features

- "const" can be put before a type to make that thing read-only
  - E.g. "const char*" is a pointer to a character (or character array) that can be read but not modified
- Enums are a nice way to declare a bunch of named integer constants and a integral type
  - E.g.: enum FlagColor { RED, WHITE, BLUE };
  - Coming in Java 1.5?
- Refs (&) are an alternative to pointers (*) that are never null and that automatically dereference
  - Good for call-by-reference arguments