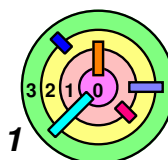


Warmup #1

Bill Cheng

<http://merlot.usc.edu/william/usc/>

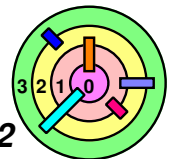


Discussion Sections



IMPORTANT:

- please understand that discussion section material are **NOT** substitute for reading the specs and the grading guidelines
 - you are expect to read the **specs**
 - you are expect to read the **requirements** the specs refer to
 - you are expect to read the **grading guidelines**
 - it's your responsibility



Programming & Good Habbits

➡ ***Always*** check return code!

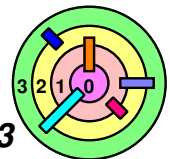
- ▬ `open()`, `write()`
- ▬ `malloc()`
- ▬ `switch (errno) { ... }`

➡ Initialize ***all*** variables!

- ▬ `int i=0;`
- ▬ `char *p=NULL;`
- ▬ `struct timeval timeout;`
`memset(&timeout, 0, sizeof(struct timeval));`

➡ ***Never*** leak any resources!

- ▬ `malloc()` and `free()`
- ▬ `open()` and `close()`
- ▬ delete temporary files



Programming & Good Habbits

➡ ***Don't*** assume external input will be short

- use `strncpy()` and not `strcpy()`
- use `snprintf()` and not `sprintf()`
- use `sizeof()` and not a constant, for example,

```
unsigned char buf[80];
```

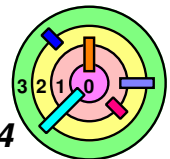
```
buf[0] = '\0'; /* initialization */
```

```
strncpy(buf, argv[1], sizeof(buf));
```

```
buf[sizeof(buf)-1] = '\0'; /* in case argv[1] is long */
```

➡ Fix your code so that you have ***zero*** compiler warnings!

- use `-Wall` when you compile to get all compiler warnings



Notes on gdb

➡ The debugger is your friend! Get to know it **NOW!**

compile program with: `-g`

start debugging: `gdb [-tui] listtest`

set breakpoint: `(gdb) break main`

`(gdb) break listtest.c:87`

run program (w/ arguments): `(gdb) run [arg1 arg2 ...]`

clear breakpoint: `(gdb) clear`

stack trace: `(gdb) where`

print field: `(gdb) print pList->anchor`

print in hex: `(gdb) print/x pList->anchor`

single-step at same level: `(gdb) next`

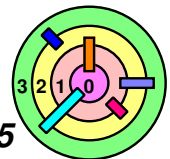
single-step into a function: `(gdb) step`

print field after every cmd: `(gdb) display pList->num_members`

assignment: `(gdb) set pList->num_members=99`

continue: `(gdb) cont`

quit: `(gdb) quit`

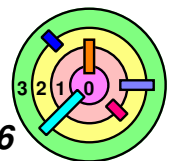


Warmup #1



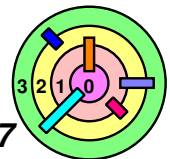
2 parts

- develop a *doubly-linked circular list* called *My402List*
 - this corresponds to part (A) of the grading guidelines
 - to implement a traditional *linked-list abstraction*
 - ◆ internally, the implementation is a *circular list*
 - ◆ internally, it behaves like a *traditional list*
 - ◆ why? circular list implementation may be a little "cleaner"
- use your doubly-linked circular list to implement a command:
 - *sort* - sort a list of bank transactions
 - this corresponds to part (B) of the grading guidelines



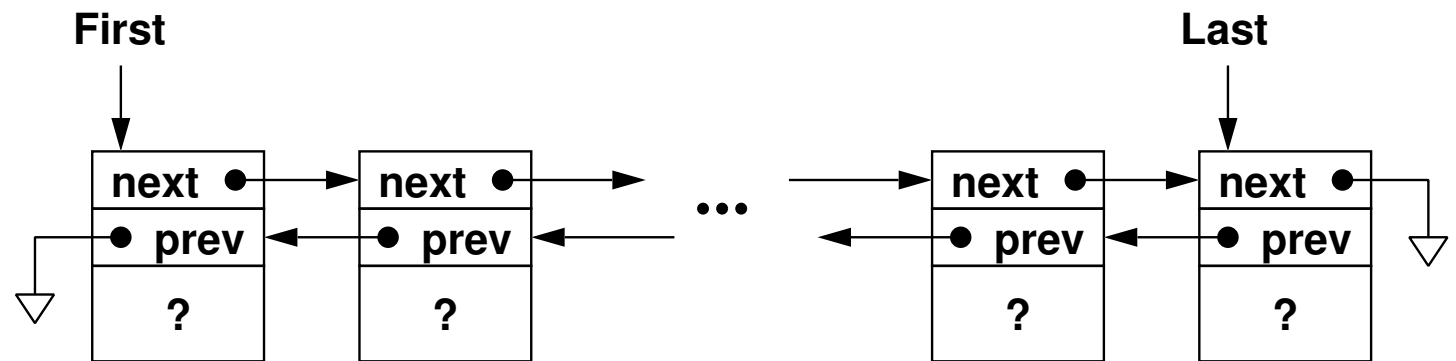
A Linked-List Abstraction

- ➡ A list of elements, linked so that you can move from one to the next (and/or previous)
 - ▬ each element holds an object of some sort
- ➡ *Functionally:*
 - ▬ First()
 - ▬ Next()
 - ▬ Last()
 - ▬ Prev()
 - ▬ Insert()
 - ▬ Remove()
 - ▬ Count()
- ➡ Need to have a well-defined interface
 - ▬ once you have a good interface, if the implementation is broken, fix the implementation!
 - don't fix the "application"

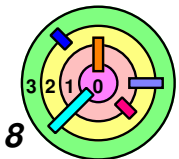
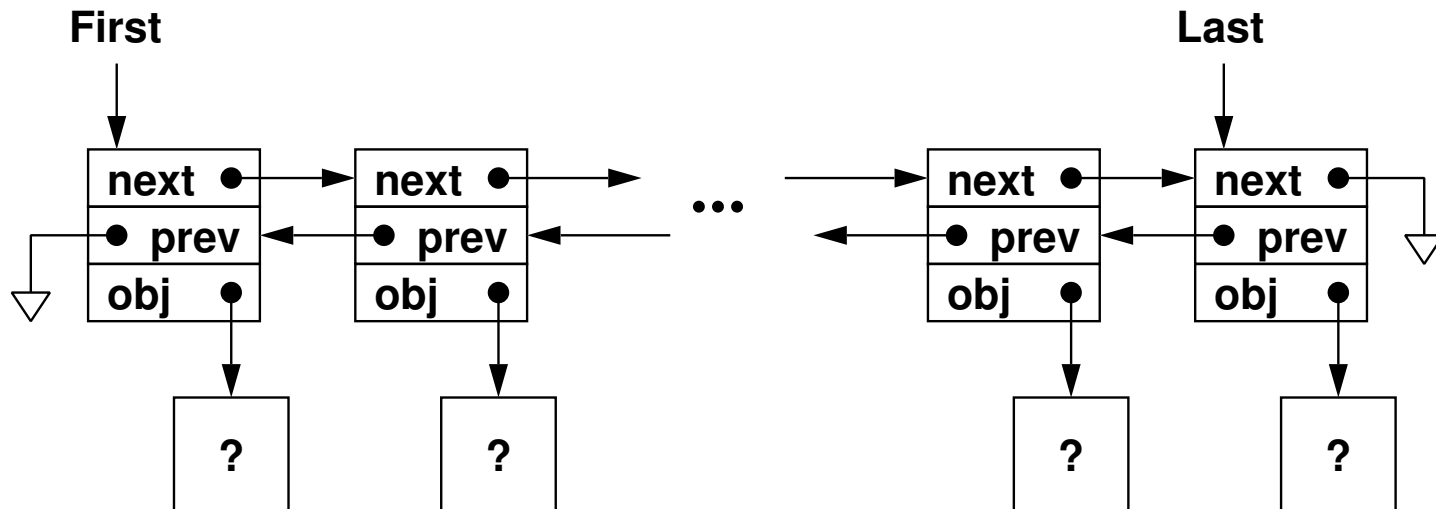


A Linked-List Abstraction

- ➡ There are basically two types of lists
 - 1) next/prev pointers in object
 - 2) next/prev pointers outside of object
- ➡ (1) has a major drawback that a list item cannot be inserted into multiple lists

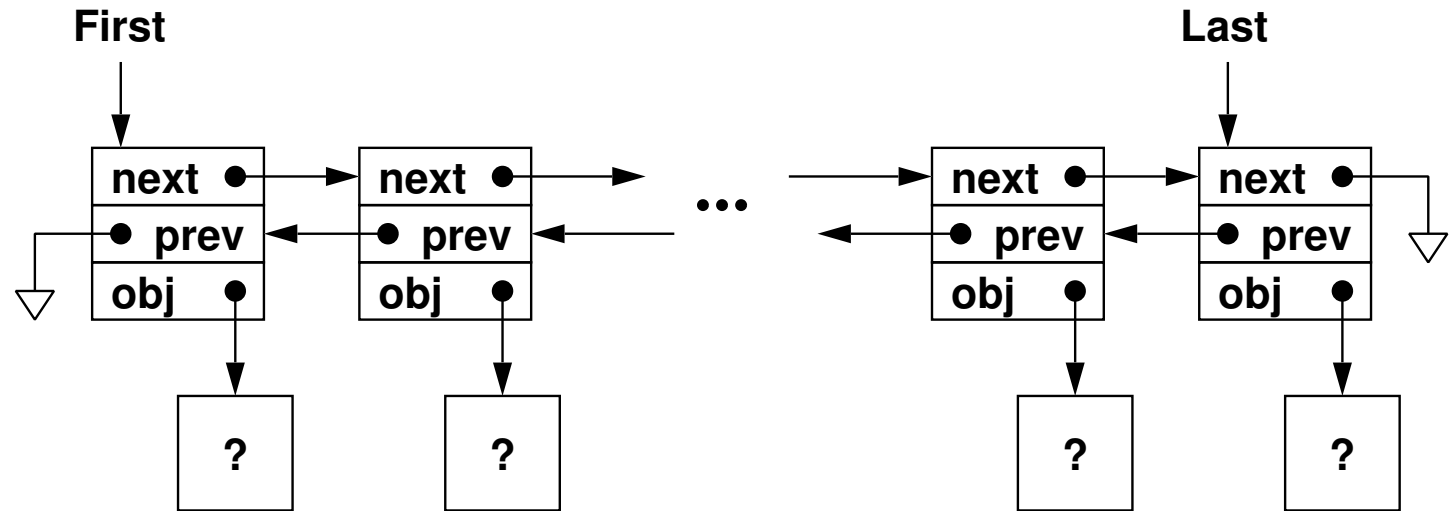


— we will implement (2) in warmup1, our kernel uses (1)



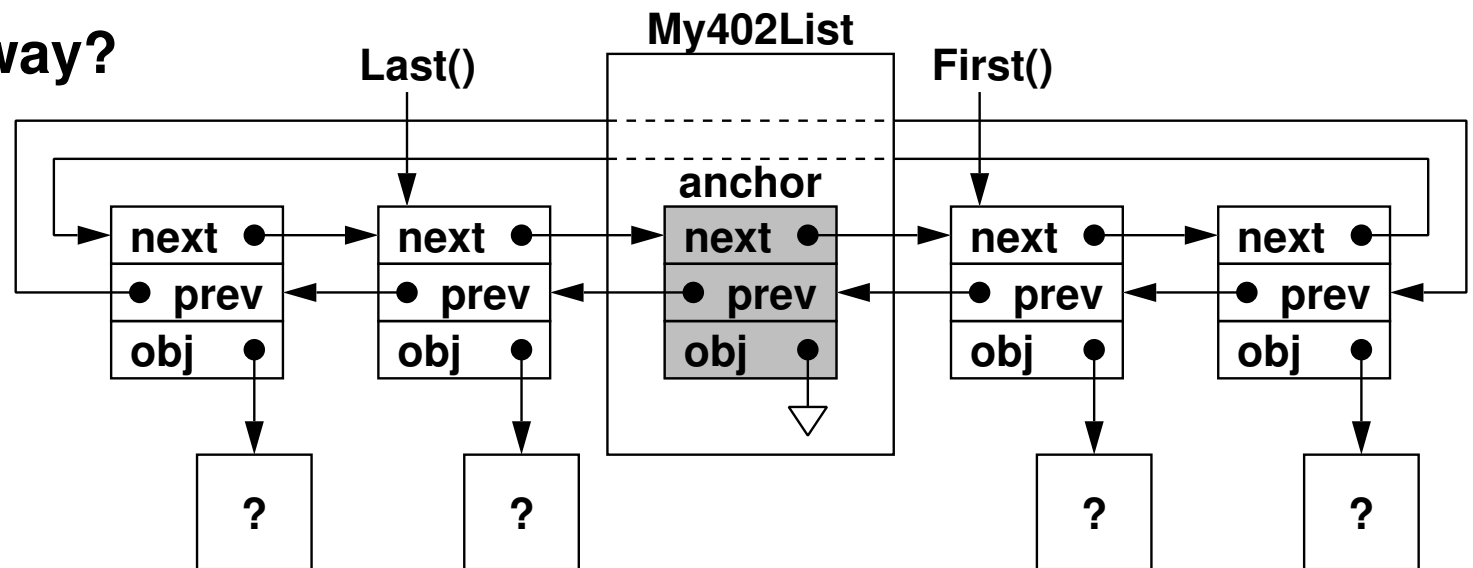
Doubly-linked Circular List

➡ Abstraction

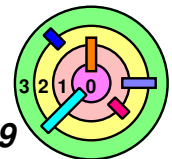


➡ Implementation

— why this way?



— your job is to implement the traditional list abstraction
using a circular list



my402list.h

```
#ifndef _MY402LIST_H_
#define _MY402LIST_H_

#include "cs402.h"

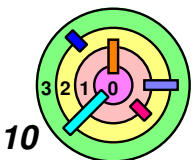
typedef struct tagMy402ListElem {
    void *obj;
    struct tagMy402ListElem *next;
    struct tagMy402ListElem *prev;
} My402ListElem;

typedef struct tagMy402List {
    int num_members;
    My402ListElem anchor;

    /* You do not have to set these function pointers */
    int (*Length)(struct tagMy402List *);
    int (*Empty)(struct tagMy402List *);

    int (*Append)(struct tagMy402List *, void*);
    int (*Prepend)(struct tagMy402List *, void*);
    void (*Unlink)(struct tagMy402List *, My402ListElem*);
    void (*UnlinkAll)(struct tagMy402List *);
}
```

➡ You need to learn to ignore things you don't understand
— assume that they are perfect



my402list.h

```

int  (*InsertBefore)(struct tagMy402List *, void*, My402ListElem*);
int  (*InsertAfter)(struct tagMy402List *, void*, My402ListElem*);

My402ListElem *(*First)(struct tagMy402List *);
My402ListElem *(*Last)(struct tagMy402List *);
My402ListElem *(*Next)(struct tagMy402List *, My402ListElem *cur);
My402ListElem *(*Prev)(struct tagMy402List *, My402ListElem *cur);

My402ListElem *(*Find)(struct tagMy402List *, void *obj);
} My402List;

extern int  My402ListLength(My402List*);
extern int  My402ListEmpty(My402List*);

extern int  My402ListAppend(My402List*, void*);
extern int  My402ListPrepend(My402List*, void*);
extern void My402ListUnlink(My402List*, My402ListElem*);
extern void My402ListUnlinkAll(My402List*);
extern int  My402ListInsertAfter(My402List*, void*, My402ListElem*);
extern int  My402ListInsertBefore(My402List*, void*, My402ListElem*);

extern My402ListElem *My402ListFirst(My402List*);
extern My402ListElem *My402ListLast(My402List*);
extern My402ListElem *My402ListNext(My402List*, My402ListElem*);
extern My402ListElem *My402ListPrev(My402List*, My402ListElem*);

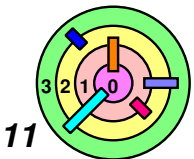
extern My402ListElem *My402ListFind(My402List*, void*);

extern int  My402ListInit(My402List*);
#endif /*_MY402LIST_H_*/

```



You need to implement all the mentioned functions



my402list.c

➡ How to start?

- ▬ `cp my402list.h my402list.c`
- ▬ edit `my402list.c` in a text editor
 - replace data structure declarations with `"#include"`
 - change all function declarations to function implementations
 - ◆ remove `"extern"` and implement function

```
#include "my402list.h"

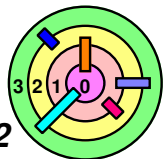
extern int  My402ListLength(My402List*);
extern int  My402ListEmpty(My402List*);

extern int  My402ListAppend(My402List*, void*);
extern int  My402ListPrepend(My402List*, void*);
extern void My402ListUnlink(My402List*, My402ListElem*);
extern void My402ListUnlinkAll(My402List*);
extern int  My402ListInsertAfter(My402List*, void*, My402ListElem*);
extern int  My402ListInsertBefore(My402List*, void*, My402ListElem*);

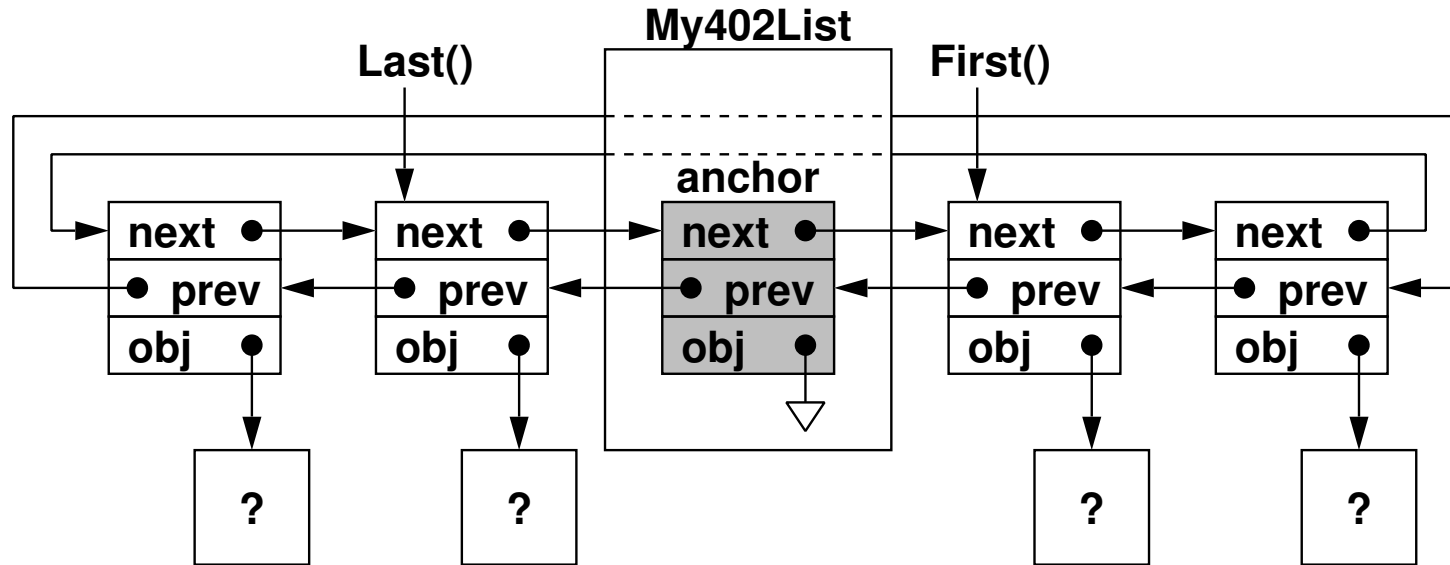
extern My402ListElem *My402ListFirst(My402List*);
extern My402ListElem *My402ListLast(My402List*);
extern My402ListElem *My402ListNext(My402List*, My402ListElem*);
extern My402ListElem *My402ListPrev(My402List*, My402ListElem*);

extern My402ListElem *My402ListFind(My402List*, void*);

extern int  My402ListInit(My402List*);
```



Implementation

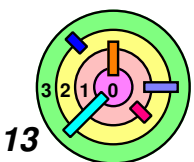


```
int Length() { return num_members; }
int Empty() { return num_members<=0; }
```

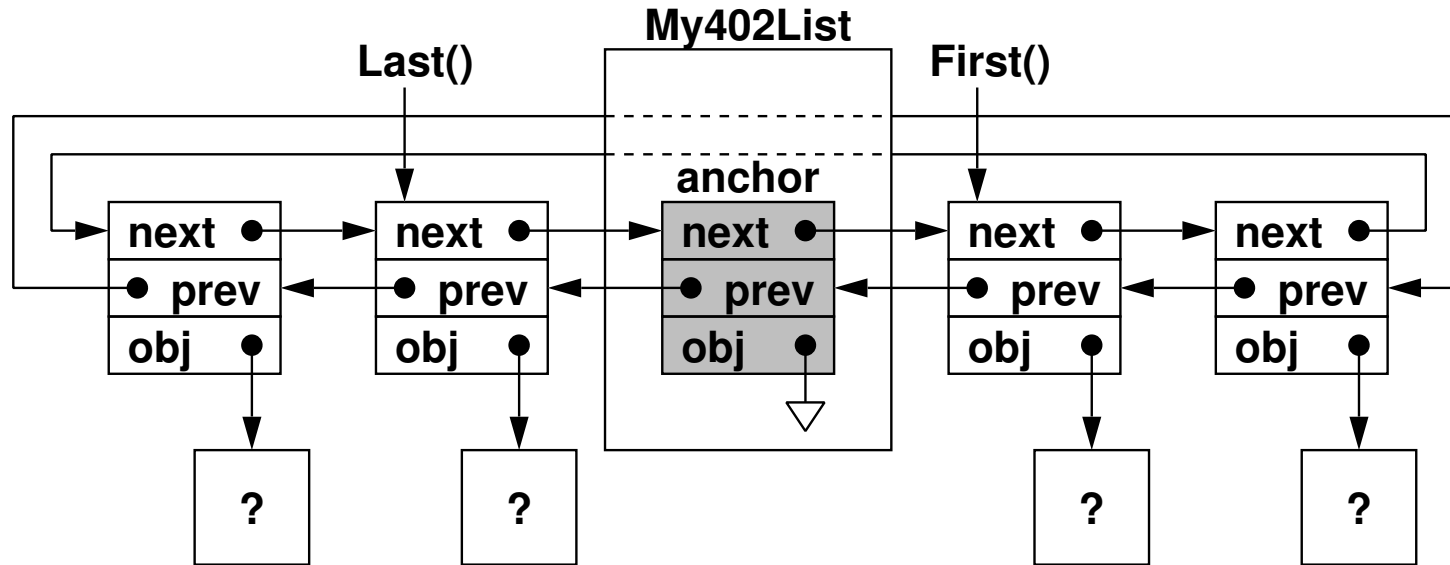
```
int Append(void *obj);
int Prepend(void *obj);
void Unlink(My402ListElem*);
void UnlinkAll();
int InsertBefore(void *obj, My402ListElem *elem);
int InsertAfter(void *obj, My402ListElem *elem);
```

```
My402ListElem *First();
My402ListElem *Last();
My402ListElem *Next(My402ListElem *cur);
My402ListElem *Prev(My402ListElem *cur);
```

```
My402ListElem *Find(void *obj);
int Init();
```



Usage - Traversing the List



```
void Traverse(My402List *list)
{
    My402ListElem *elem=NULL;

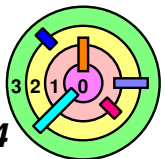
    for (elem=My402ListFirst(list);
        elem != NULL;
        elem=My402ListNext(list, elem)) {
        Foo *foo=(Foo*)(elem->obj);

        /* access foo here */
    }
}
```



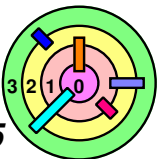
This is how an *application* will use **My402List**

- you must support this "*contract*" with you application
- if broken, fix the "implementation" and not the "application"



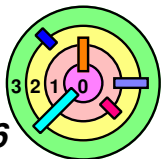
listtest

- ➡ Use provided `listtest.c` and `Makefile` to create `listtest`
 - ▬ `listtest` must run without error and you must not change `cs402.h`, `my402list.h`, `listtest.c` and `Makefile`
 - ▬ they specify how your code in `my402list.c` is expected to be used
- ➡ You should learn how to run `listtest` under `gdb`



Sort Command

- ➡ **warmup1 sort [tfile]**
 - produce a sorted transaction history for the transaction records in `tfile` (or `stdin`) and compute balances
- ➡ **Input is an ASCII text file (use `fgets()` to read a line)**
 - each line in a `tfile` contains 4 fields delimited by `<TAB>`
 - transaction type (single character)
 - ◆ "+" for deposit
 - ◆ "-" for withdrawal
 - transaction time (UNIX time)
 - ◆ `man -s 2 time`
 - amount (a number, a period, two digits)
 - transaction description (textual description)
 - ◆ cannot be empty
- ➡ **Output must be in the specified format exactly**
 - use the grading guidelines to check if you miss something
 - formatting bugs should be very easy to fix



Sort Command

➡ Output

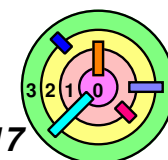
```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
1234567890123456789012345678901234567890123456789012345678901234567890
```

Date	Description	Amount	Balance
Thu Aug 21 2008	...	1,723.00	1,723.00
Wed Dec 31 2008	...	(45.33)	1,677.67
Mon Jul 13 2009	...	10,388.07	12,065.74
Sun Jan 10 2010	...	(654.32)	11,411.42

➡ How to keep track of balance

- first thing that comes to mind is to use `double`
- the weird thing is that if you are not very careful with `double`, your output will be wrong (by 1 penny) once in a while
- recommendation: keep the balance in cents, not dollars
 - no precision problem with integers!

➡ Read *grading guidelines* and find many examples of *valid input* and *expected printout*



Sort Command

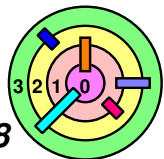
```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Date	Description	Amount	Balance
Thu Aug 21 2008	...	1,723.00	1,723.00
Wed Dec 31 2008	...	(45.33)	1,677.67
Mon Jul 13 2009	...	10,388.07	12,065.74
Sun Jan 10 2010	...	(654.32)	11,411.42

➡ The spec requires you to call `ctime()` to convert a Unix timestamp to string

- then pick the right characters to display as date
- e.g., `ctime()` returns "Thu Aug 30 08:17:32 2012\n"
- becareful, `ctime()` returns a pointer that points to a *global variable*, so you must *make a copy*

```
char date[16];
char buf[26];
strncpy(buf, ctime(...), sizeof(buf));
date[0] = buf[0];
date[1] = buf[1];
...
date[15] = '\0';
```



Sort Command

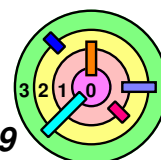
```
00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890
```

Date	Description	Amount	Balance
Thu Aug 21 2008	...	1,723.00	1,723.00
Wed Dec 31 2008	...	(45.33)	1,677.67
Mon Jul 13 2009	...	10,388.07	12,065.74
Sun Jan 10 2010	...	(654.32)	11,411.42



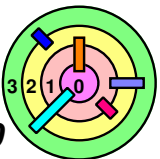
Format your data in your own buffer

- write a function to "format" numeric fields into null-terminated strings
 - it's a little more work, but you really should have this code isolated
 - ◆ in case you have bugs, just fix this function
- you can even do the formatting when you append or insert your data structure to your list
 - need more fields in your data structure
- this way, you can just print things out easily
- use `printf("%s", ...)` to print a field to `stdout`



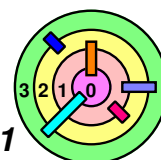
Warmup #1

- ➡ I'm giving you a lot of details on how to do things in C
 - this is the first and last assignment that I will do this!
 - you must learn C (and Unix) on your own
- ➡ Read man pages
- ➡ Ask questions in class Google Group
 - or send e-mail to me
- ➡ Come to office hours, especially if you are stuck



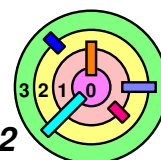
Some General Requirements

- ➡ Some major requirements for all programming assignments
- severe penalty for failing `make` (can lose up to 10 points)
 - we will attempt to fix your Makefile if we cannot compile your code
 - we are *not permitted to change your code*
 - severe penalty for using large memory buffers
 - if input file is large, you must not read the whole file into into a large memory buffer
 - ◆ must learn how to read a large file properly
 - severe penalty for any segmentation fault -- you must test your code well
 - severe penalty for not using separate compilation or for having all your source code in header files -- you must learn to plan how to write your program
 - read warmup1 FAQ to see what's the best way to go about this



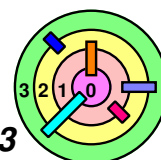
Grading Requirements

- ➡ For *warmup assignments*, it's important that **every byte** of your data is read and written correctly
 - you are not entitled to partial credit just because you wrote some code
- ➡ For *warmup assignments*, you should run your code against the *grading guidelines* on 32-bit Ubuntu 16.04
 - must not change the commands there
 - we will change the data for actual grading, but we will stick to the commands (as much as we can)
 - to be fair to all, running scripts in the grading guidelines on the grader's 32-bit Ubuntu 16.04 is *the only way we can grade*



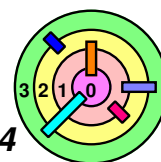
Separate Compilation

- ➡ Break up your code into *modules*
 - ➡ *compile the modules separately*, at least one rule per module per rule in the `Makefile`
 - ➡ a separate rule to *link* all the modules together
 - if your program requires additional libraries, add them to the link stage
- ➡ To receive full credit for separate compilation
 - ➡ to create an executable, at a minimum, you must run the compiler at least *twice* and the linker *once*
 - ➡ see the *warmup1 FAQ* for exactly how to avoid losing points



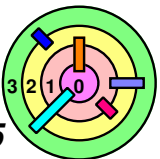
README

- ➡ Start with the README templates from the spec
 - ***BUILD & RUN*** (required)
 - replace "(Comments: ?)" with appropriate responses
 - ***SELF-GRADING*** (required)
 - replace each "?" with a **numerical score**
 - ***BUGS / TESTS TO SKIP*** (required)
 - replace "(Comments: ?)" with a list of tests to skip or "none"
 - ◆ you would still lose points, but this may prevent losing additional points in another part
 - ***ADDITIONAL INFORMATION FOR GRADER*** (optional)
 - grader must read this
 - ***OTHERS*** (optional)
 - will **not** be considered for grading
- ➡ There should be no "?" left in a response ***in a required section*** after you have filled out a README file correctly
 - 0.5 pt will be deducted if a "?" is not replaced with something appropriate or if the line is omitted



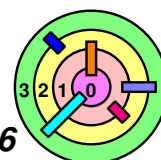
Code Design - Functional vs. Procedural

- ➡ Don't design your program "procedurally"
- ➡ You need to learn how to write functions!
 - please note that this is not "functional programming" ("functional programming" is something else)
 - a function has a well-defined interface
 - what are the meaning of the parameters
 - what does it suppose to return
 - pre-conditions
 - what must be true when the function is entered
 - you assume that these are true
 - ◆ you can verify it if you want
 - post-conditions
 - what must be true when the function returns
 - you design your program by making designing a sequence of function calls



Warmup #1 - Miscellaneous Requirements

- ➡ Run your code against the *grading guidelines*
 - must not change the test program
- ➡ You must not use any *external code fragments*
- ➡ You must not use *array* to implement any list functions
 - *must use pointers* because this is a pointer exercise
 - read my *review about pointers* in warmup1 FAQ
- ➡ It's important that every byte of your data is read and written correctly.
 - `diff` commands in the grading guidelines must *not* produce *any* output or you will not get credit
 - what does "not produce any output" mean?
 - ◆ it means exactly what it says!
- ➡ Please see Warmup #1 spec for additional details
 - please read the *entire* spec (including the grading guidelines) *yourself*



Development

- ➡ Get familiar with "*Warmup #1 FAQ*" and "*Programming FAQ*"
- ➡ Text Editors
 - emacs, pico, vi
 - some students like Sublime Text
 - you are on your own with Sublime Text
- ➡ Compiler
 - "gcc --version" should say it's version 5.4.something
- ➡ IDE
 - some students like Eclipse
 - you are on your own with Eclipse
 - the grader is *not permitted* to use an IDE to compile or run your program
 - if you use an IDE, it's your responsibility to make sure that you provide a Makefile so that the grader can type the command in the spec to compile

