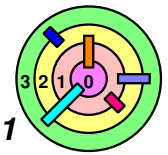
CS 402
Operating Systems

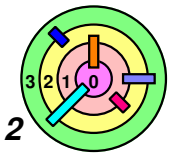
Bill Cheng

http://merlot.usc.edu/cs402-f13



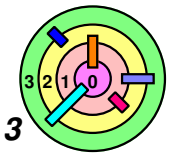
Fact Or Myth?

- Is it true that doing well in this class gets you job offers?



Fact Or Myth?

- Is it true that doing well in this class gets you job offers?
  - definitely true!
    - if you learn the course material well, and
    - if you implemented all the assignments yourself
- You want answers, you can just google
  - then you are just like everyone else
  - if you want to impress your interviewers, you need to impress them with your experience (integrated with your knowledge)
    - the "theory" stuff is just as important as "kernel hacking"

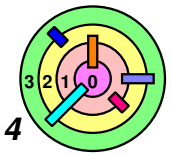


My Teaching Style

- I'm a strong believer in: (adapted from Lao Tzu)

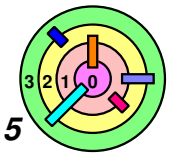
  Give a man a fish and you feed him for a day.
  Teach a man to fish and you feed him for a lifetime.

  - except for the warmup programming assignments, I tend not to give a straight answer
    - I want you to find the answer yourself (or together with your peers)



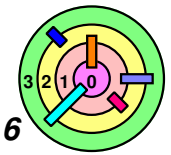
No "Spoon-feeding"

- The coverage of this class is quite vast
  - too many important concepts to cover in 15 weeks
  - it is not feasible to explain everything till you understand it perfectly
  - do not expect "spoon-feeding"
  - I will explain the concepts, but you have to work hard so you fully understand them
    - come talk to me as much as you need!



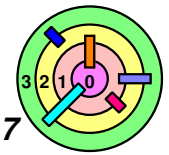
Today's Topics

- Administrative Stuff
  - the instructor *cannot* give D-clearance
    - please use the on-line D-clearance system
  - I will not waived a pre-requisite for anyone
  - undergrad students must take Prof. Govidan's section of CS 402
  - our section is *significantly different* from Prof. Govidan's section
    - textbook and programming assignments are all different
- Review Course Organization
  - please read all the administrative lecture slides
- Start you off working on "warmup" assignment #1
  - please read all the warmup #1 lecture slides
- Chapter 1
  - 1.1 introduction
  - 1.3 a simple OS



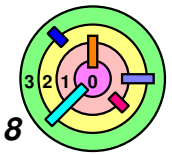
Class Structure & Teaching Staff

- Instructor: Bill Cheng <bill.cheng@usc.edu>
  - email: 24 hour turn-around
  - office Hours: in SAL 218, M/Tu/W/Th 2:00pm - 3:00pm, or by appointments
- Lectures
  - *(AM section)* MW 10:00am - 11:50am in WPH B27
  - *(PM section)* MW 12:00pm - 1:50pm in SLH 102
  - my plan is to keep the two sections synchronized
- You are expected to attend every lecture
  - OS is not just about programming, it's about important OS concepts



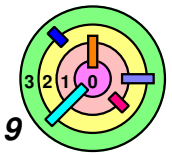
Class Structure & Teaching Staff

- **TAs**
  - *(AM section)* Sung-Han Lin <sunghan@usc.edu>
  - *(PM section)* Bo-Chun Wang <bochunwa@usc.edu>
  - email: 24 hour turn-around
  - office hours: (TBD)
  - the TA's job is to help you with the course materials, programming assignments, and grade exams
- **Course Producer**
  - *(AM section)* (none)
  - *(PM section)* Zhiyi Xu <zhiyixu@usc.edu>
  - email: 24 hour turn-around
  - help desk hours: (TBD)
  - the course producer's job is to help you with the programming assignments



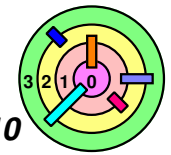
Class Structure & Teaching Staff

- **Graders:**
  - ***(AM section)*** **Junjie Jin <junjieji@usc.edu>**
  - ***(PM section)*** **Feiyi Xiang <fxiang@usc.edu>**
  - **email: 24 hour turn-around**
  - **the grader will hold regrade sessions after you get grade notification e-mail**
  - **we have different rules about grader involvement for our class**
    - **the grader's job is strictly to *grade your programming assignments***
    - **it is inappropriate to contact the grader about an assignment (especially about "how many points I would get if I do it this way") before the assignment is due**
      - **the grader will not answer such questions**



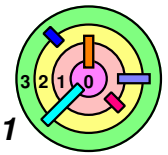
Class Resources

- Class web page: http://merlot.usc.edu/cs402-f13
  - everything about this class is there
    - anything related to grading, you are required to know
  - get familiar with it
    - if you are not used to reading a lot of stuff, you should start reading a lot of stuff in this class
      - it's important to learn how to read documents and interpret them correctly
- If you see inconsistencies, especially regarding any type of "rules" between what's on the class web page and what's on a set of lecture slides that has been covered in class
  - usually, the lecture slides are correct
  - but, you should check with the instructor as soon as possible!
    - so that things can be consistent again



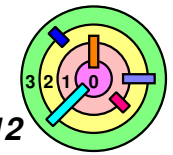
Lectures

- I will not cover every posted lecture slides
  - there's not enough time
  - although you will be responsible for everything posted in lecture slides (and the corresponding materials in the textbook)
    - except the ones that are marked with a grey ✖ at the lower left corner of the slide
    - feel free to ask me about things on the lecture slides but were not covered during lectures



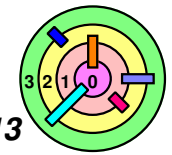
Class Discussion (Google) Group

- Google group (both sections use the same one)
  - student-to-student discussions about programming assignments and course materials
  - exchanging *ideas* are allowed
  - posting code is *not* allowed (short code segments to illustrate ideas are allowed; short means < 5 lines)
    - *first offense* (in the entire semester) gets a warning
    - *2nd offense*, you will lose 50% of the corresponding assignment points and lose posting privileges
  - instructor, TAs, course producer will also post answers to questions here
    - if appropriate for whole class
  - you can get *extra credit* if you post *good/useful* answers to other students' questions for *kernel* assignments (see slides later)



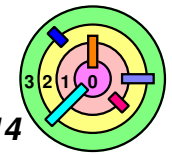
Class Discussion (Google) Group

- You *must be a member* of the *class Google Group*
  - *all* important announcements will be posted to this group
- There are two way to use the class Google Group
  1) use it as an e-mail reflector (i.e., don't need to "login" to the Google Group)
     - get an e-mail copy of everything posted to the group
     - send postings to the group
     - you *don't need* a Google account
     - this option is *not* available to you if your USC e-mail is accessed through *Google Apps at USC*
     - if you are a conscientious objector to Google, this may be your only option (and you will have to manage ALL messages posted to the class Google Group)
  2) full access (i.e., "login" to the group, search the message archive, post using web form, etc.)
     - you *must have* a Google account
     - this is the *preferred* way



Class Discussion (Google) Group

- If you are on the class roster, I will invite you to join the class Google Group
  - by default, I will use your USC e-mail address
    - therefore, the default mode of using the class Google Group is e-mail reflector (although the other mode is preferred)
    - if you read e-mail using *Google Apps at USC*
      a) *do not accept* the invitation because it won't work
      b) you have to use method (2)
  - if you want to use a different e-mail address (such as gmail) to access Google Group, please do the following
    a) *do not accept* the invitation
    b) login to Google with your other e-mail address
      - click on our class Google Group link and apply for membership
      - make sure you provide your *USC e-mail* address (not USC ID) in the *"additional information"* field (so I can verify that you are really in my class)



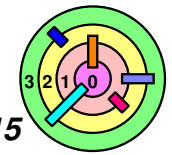
Grading

- **Rowcall** 3% starting from week 4 of classes (extra credit)
- **Projects** 50% 2 warm-ups (individual), 3 group projects
- **Midterm** 20% in class, Wed, 10/30/2013 *(firm)*
- **Final** 30% *(AM section)* 8am - 10am, Mon, 12/16/2013 *(firm)*
  (PM section) 11am - 1pm, Fri, 12/13/2013 *(firm)*

- **Extra credit**
  1) turn in *assignments* more than 48 hours before deadline
  2) giving good/useful answers in the class Google Group for *kernel assignments*

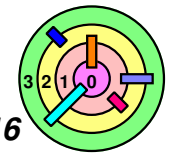
- **No *individual* extra credit**
  - try your best from the beginning!

- **Projects graded by the grader**
  Exams graded by the TA



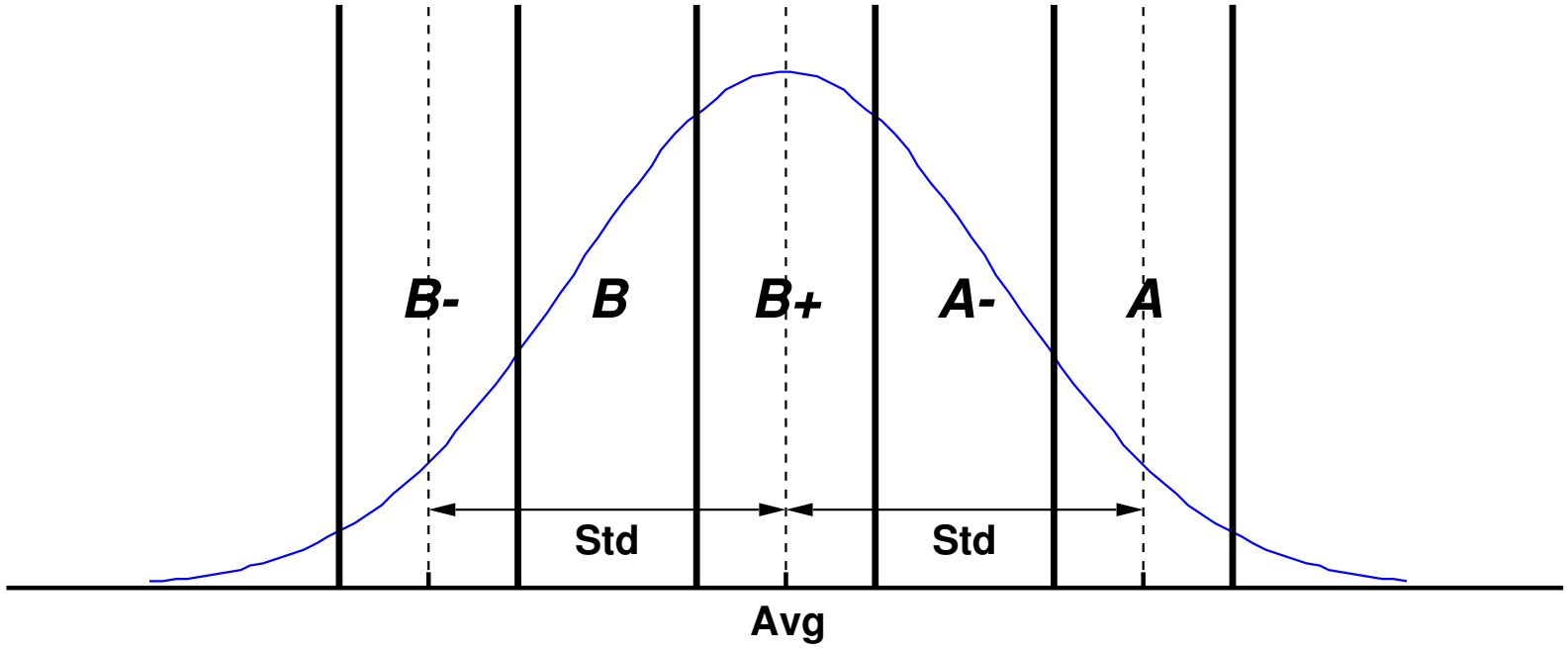
Grading

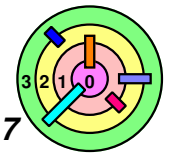
- Final letter grade assigned by the instructor
  - a grade of *incomplete* is only possible for *documented* illness or *documented* family emergency (according to USC policy)
- Two methods
  1) on a curve
  2) fixed scale
  - your class letter grade will be the higher of the two
  - C's may be given, F's if necessary



Grading

- Curve
  - class average is probably a B+
  - loose guideline depicted below:

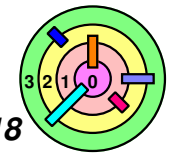




Grading

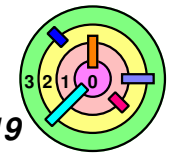
Fixed scale

Percentage	Letter Grade
90% or higher	A
80-90%	A-
70-80%	B+
60-70%	B
50-60%	B-
40-50%	C+
30-40%	C
20-30%	C-
below 20%	F



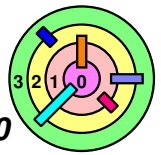
The Importance of "Written Words"

- We communicate (here and in the real world) using "words"
  - you need to learn to take "words" seriously
  - especially when it comes to ***rules*** written in words
    - things that are not written can get messy
- If it's writtent that X is the deadline and that you will get a zero if you turn things in after X
  - what would you get if you turn it in after X?
  - can you ask to turn it in after X?
    - sure, you "can" ask
- When it comes to exams, we can only grade based on what you wrote on the exam paper (and not what's in your mind)
  - you need to learn to choose your words carefully
- Please pay attention to written words in class web site, lecture slides, posting to class Google Group by the ***TA*** or ***me***
  - course producer is not involved in grading



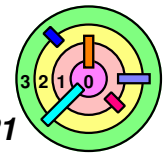
Fairness

- **Fairness**
  - without fairness, grades have little meaning
  - the instructor *must treat all students equally* and cannot give special treatment to any particular student
  - therefore, please do not ask special favors from the instructor because of your circumstances (except for ones that are explicitly allowed by the university)
  - this may seem unfair to you because you believe that your circumstances are special (understandably, everyone does)
  - bottom line, the rule the instructor must follow is that *whatever he offers you, he must offer to the entire class*



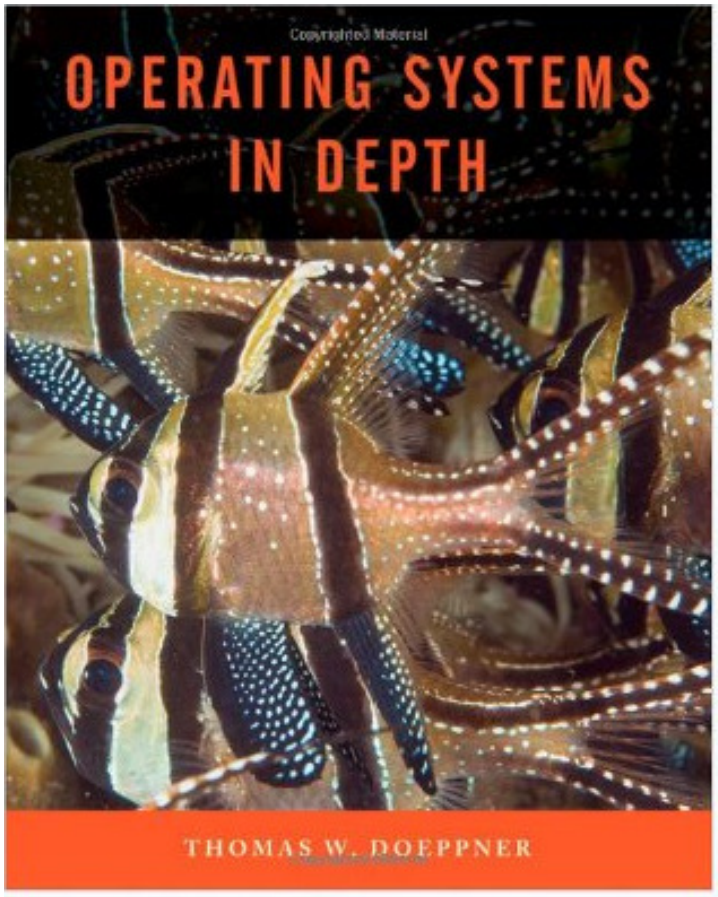
E-mail Questions

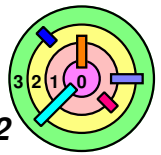
- Please do not ask these questions over ***e-mails:***
  - here is my understanding of X. "Am I right?" "Is this correct?" "Correct me if I'm wrong..." "Please confirm that I'm right..."
    - if everyone these questions about everything, I will be completely out of bandwidth
      - therefore, I will not answer such a question
      - perfectly okay to ask these questions in ***office hours***
    - find a different way to ask over e-mail
  - I don't understand X. Could you explain X to me?
    - if you attended lectures, you can ask this during office hours
  - here is what I am thinking of or doing... is it acceptable (or is this okay)?
    - can you ask this during exams?



Course Readings

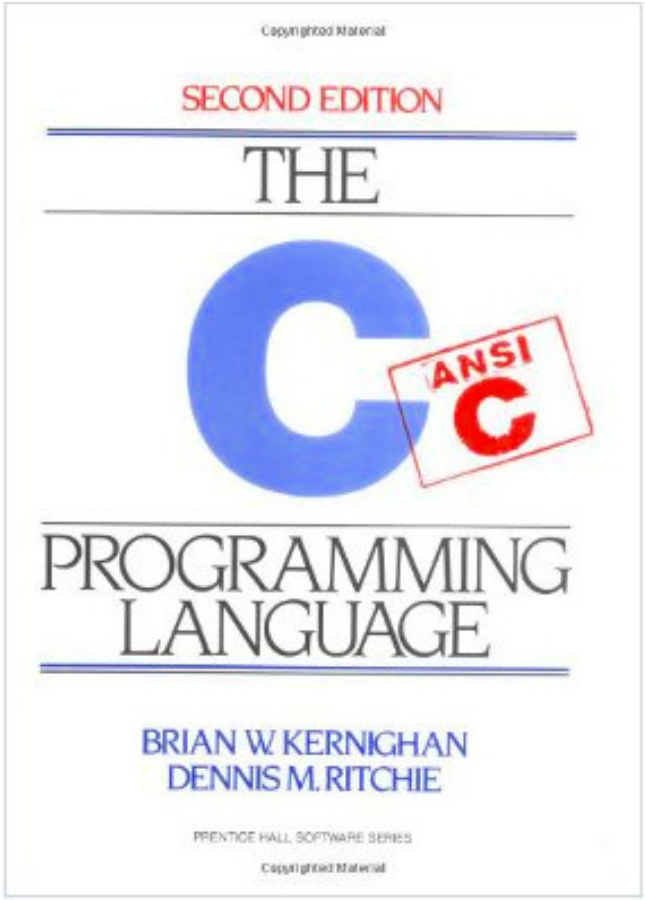
- ***Required*** **textbook**
  - ***"Operating Systems In Depth: Design and Programming"*** **by T. W. Doeppner**
  - **we will follow this book closely**





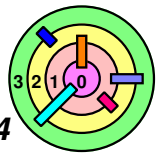
Course Readings

- *Optional* textbook
  - *"C Programming Language"*
    by B. Kernighan and D. Ritchie
  - all programming assignments must be done in C



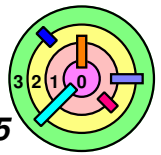
Class Structure

- Lectures mostly based on Doeppner
  - slides
  - lectures should be interactive
  - lecture slides posted on class web site soon after class
- You must keep up with the assigned readings
  - you are expected to read the relevant textbook chapter thoroughly
  - not all details will be covered in class
  - exam may test material covered in the textbook but not discussed in class
- I expect you to attend every class meeting
  - if you do happen to miss a class, you are responsible for finding out what material was covered and what administrative announcements were made
  - you will be expected to attend in-class exams



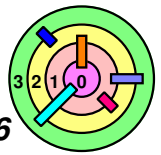
Lecture Slides

- Lecture slides came with the book
  - the purpose of lectures is to explain what's in the textbook
    - the textbook is not all that easy to understand
    - lecture slides are not meant as "extra material"
  - lecture slides have a lot of details
    - too much details?!
      - some are verbatim from the textbook
    - too much details can be a good thing
      - it tells you what's important to study
      - you don't need a study guide for exams!
- You need to understand every slide
  - do you need to read the textbook?
    - no if you just want to do well in the exams and you understand everything covered in lectures
    - although highly recommended
      - who can remember everything from lectures?



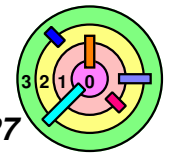
Sign Row Sheet

- For *on-campus* students only
  - If you are enrolled in the DEN section, you are not required to sign row sheets
  - DEN students normally have a slight disadvantage that they can miss stuffs happening in the class
    - this is served as an *"equalizer"*



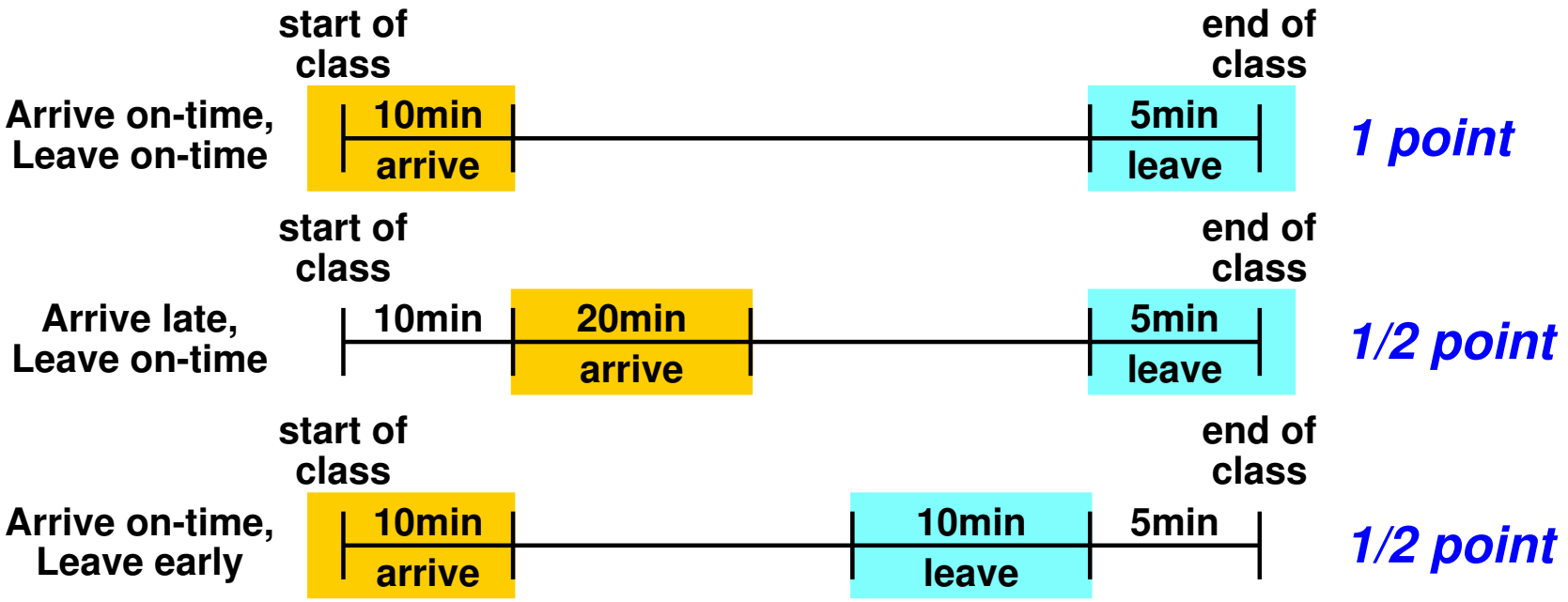
Sign Row Sheet

- Starting from *week 4*, each lecture is worth 1 point
  - you should sign the row sheet as soon as you get into class
  - if your signature is on the row sheet, you get the 1 point
    - provided that you do not leave before class ends
  - if you come in between 11 to 30 minutes into class or had to leave class within the last 15 minutes, check *late* box
    - you will receive 1/2 a point credit instead
    - if you have to leave during the last 15 minutes and cannot check the "late" box, please e-mail me to have your signature removed
      - failure to do so would be considered cheating and will lose *all* class participation credits
  - if you are sick or have a family emergency, please bring a note from a doctor to receive credit



Sign Row Sheet

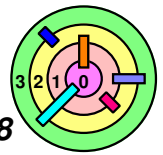
Sign Row Sheet Summary



- *0 point* if *arrive way late* or *leave way early*

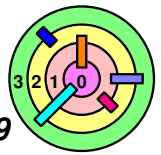
Personally, I really don't understand why anyone wants to come to class late on purpose!

- if you miss the first 10 minutes of a lecture, you may be confused for most of the lecture



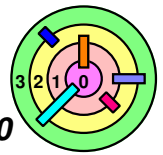
Homework Assignments

- HW (do not turn in, will not be graded)
  - based on readings and discussion
  - may help you study for the exams
  - check the back of a book chapter for problems to solve



Projects / Programming Assignments

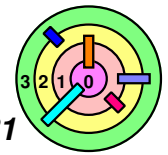
- **Programming assignments**
  - (10%, 15%) 2 warm-up projects *(to be done individually)*
    - linked list in C, pthreads (no kernel code)
  - (25%, 25%, 25%) 3 kernel projects *(to be done in a group of 2-4 students)*
    1) kernel threads
    2) virtual file system OS layer
    3) virtual memory (extremely difficult)
  - no solutions will be given
  - program in C only
    - you must learn C on your own
    - C is a proper subset of C++
      - although stream I/O and strings are not available in C
      - you must learn how to do I/O the right way
      - you must learn how to deal with C-strings (i.e., null-terminated array of chars)
  - the kernel assignments are *extremely difficult*
    - should get started as soon as possible



Projects / Programming Assignments

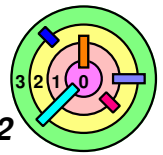
- Kernel assignments must be done on Ubuntu 11.10 (with QEMU 0.14.1)
  - this is due to some serious "bugs" in the kernel assignments
    - people at Yale are trying to get this fixed, but it can take time
- You should install Ubuntu 11.10 on your laptop/desktop as early as possible and let me know if there are problems
  - follow the instruction at the bottom of the class web page
    - if you have a Windows 7/8 laptop/desktop, use WUBI to install
    - otherwise
      - if you have a fast Windows or Mac, download VirtualBox from Oracle and install Ubuntu 11.10 into VirtualBox
      - can also try VMware
      - if you have a slower Windows or Mac, you may have to create a separate disk partition and install Ubuntu 11.10 into it

this is preferred



Projects / Programming Assignments

- For kernel (group) projects, half the grade is "team grade" and half is "individual grade"
  - in the README file your team will submit, you need to:
    - specify how to split the points (in terms of percentages and must sum to 100%)
    - give a brief justification about the split
  - for exmaple, the grader gives you 80 points and everyone contributed equally, everyone gets 80 points
  - what if it's not equal contribution?
    - what if the split is 0/50/50? what should be your scores?
    - what if the split is 20/30/50? what should be your scores?
- We would like to ***encourage*** that everyone contribute to the team ***equally*** (or at least for you to declare as so in your README file)
  - your score is maximized when you have equal contribution
  - there would be "penalty" for not dividing the scores equally
    - this can lead to unfairness, but that's the nature of group projects

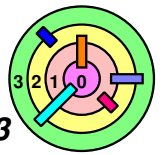


Projects / Programming Assignments

- How do we figure out the scores when there is an uneven breakdown?
  - it's kind of complicated
    - the basic idea is to calculate the standard deviation among the contribution percentages
    - if there is an even breakdown, the standard deviation is zero
      - this gives an upper bound and no penalty
    - if the breakdown is 0/0/100 (highest possible standard deviation), the would be the worst case, i.e., most penalty
      - but in this case, the person with 100% will get all the points assigned by the grader and everyone else will get 50% of the points
  - run a program nunki/aludra:
    - general syntax

```
~csci551b/bin/cs402calc grade p1 p2 ...
```

      - where `grade` is the grade the grader gave, and `p1`, `p2`, `...` are percentages



Projects / Programming Assignments

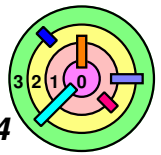
- How do we figure out the scores when there is an uneven breakdown?
  - run a program nunki/aludra:
    - for example, if the grader gave 80 points and the breakdown is 20/30/50, run:

```
~csci551b/bin/cs402calc 80 20 30 50
```

      - the scores will be 55.695, 63.5425, and 79.2375
      - so, the person who contribute the most did not get 80
    - if the grader gave 80 points and the breakdown is 0/50/50:

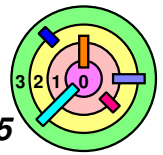
```
~csci551b/bin/cs402calc 80 0 50 50
```

      - the scores will be 40, 70, and 70
      - more penalty for not being able to figure out how to share responsibilities among 3 students!
      - may be it's not worthwhile to assign such percentages!
  - I'll leave it to you to decide!
- If you don't specify, we will assume it's an equal-share split



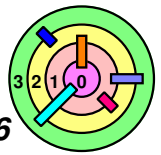
Projects / Programming Assignments (Cont...)

- Forming groups
  - you can only form a group with *students from your own section*
  - It's probably a good idea to work with other students during the warmup projects
    - although you must *write code independently* for the warmup projects
    - "work with" means working at a high level (*not* code level)
  - all students not belonging to a group by the group forming deadline will be assigned a group
    - algorithm:
      - form a random list from these students (random drawing)
      - assign 3 to a group starting from the beginning of the list
      - remaining students join these randomly formed groups
      - this is the only way to have 4-student groups
      - boundary conditions? hope we don't get there
  - after groups are formed, only mutually agreed swaps are allowed (must let us know)



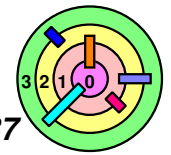
Projects / Programming Assignments (Cont...)

- We know that our algorithm is far from perfect
  - that's why it's specified way in advance
  - this way, you can plan how to form teams given that you know our algorithm for assigning teams
- If you cannot form a team of your own and end up in a team that cannot get the kernel assignments to work?
  - probably because there are problem members
  - is this unfair to you?
    - No! because you did have options
  - given that you know that this is the rule, what should you do?
    - form your own team, or
    - do the kernel projects by yourself!
    - if you *choose* neither, we cannot grade you differently *given our fairness policy*
  - during the first 6 weeks, work with other students so you know whom you want to be partners



Electronic Submissions

- Use *bsubmit* and the *Bistro* system (see web page for details)
  - you can make multiple submissions
    - will grade last submission by default
  - Bistro system gives tickets and receipts
    - these are *proofs* that the *server* got your submission
    - *we cannot trust any file timestamp*
    - we can only trust things that have made it to our server
  - very important: *read output of bsubmit*
  - very important: *verify your submissions*
    - see the bottom of the Electronic Submission Guidelines web page for details
    - if you forget a file in your submission, you are not allowed to resubmit it after the deadline
  - submit source code only (or 2 points will be deducted)
  - for team projects, only one member needs to submit
  - it is *your responsibility* to make sure that your submission is valid - *Be paranoid!*

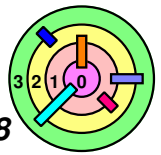


Electronic Submissions (Cont...)

- Use *bsubmit* and the *Bistro* system (see web page for details)
  - no other form of submission will be accepted
    - use your judgement under special circumstances
- Account getting full
  - please do not delete or alter anything in your `~/.bistro` directory
  - If you must delete your `~/.bistro` directory, you should tar then gzip the content of your `~/.bistro` directory and e-mail the resulting tgz file to the instructor:

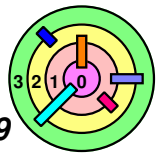
```
cd; gtar cvzf $(USER)-bistro.tgz .bistro
```

  - delete the directory only after you get an confirmation e-mail from the instructor (or at your own risk)

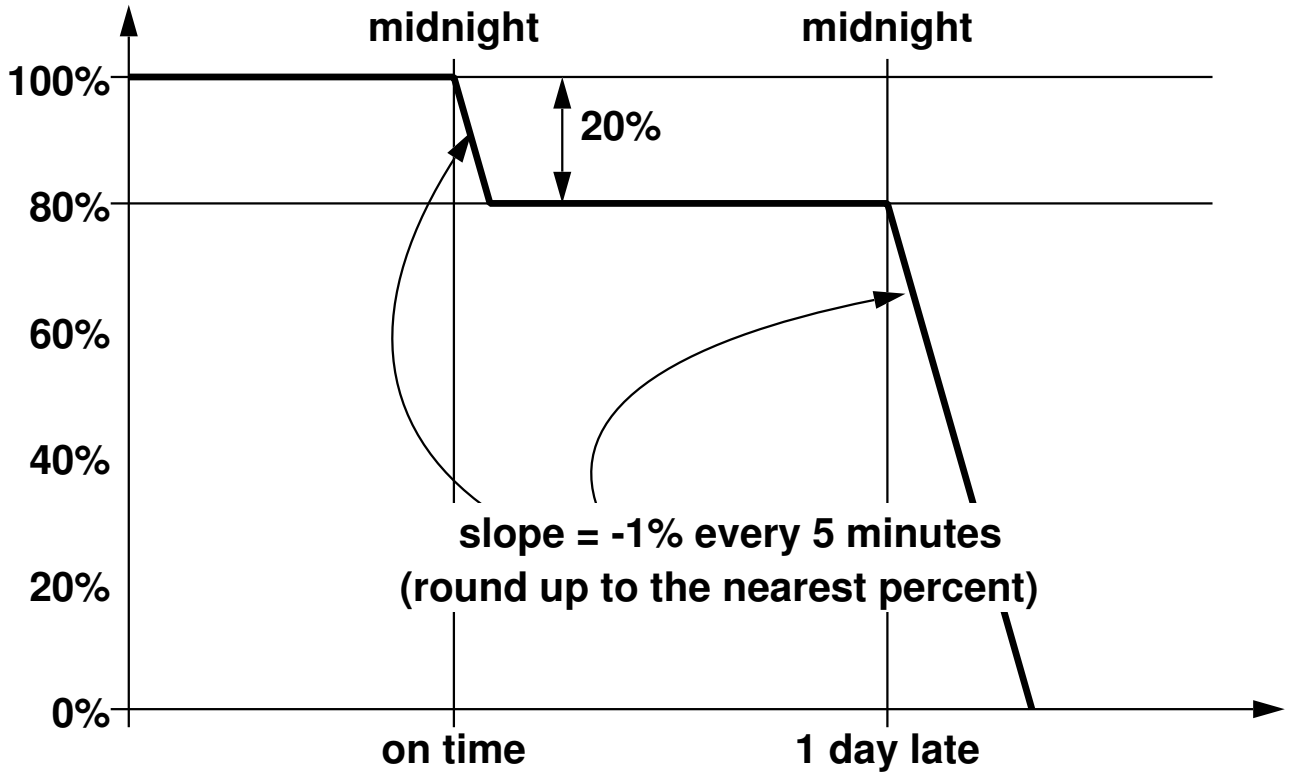


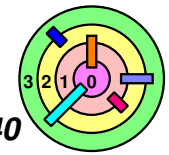
Late Policy

- **Electronically Submitted Assignments**
  - you can submit multiple times, only the last submission will be graded (unless you send us an e-mail)
  - 15 minutes grace period
  - 80% of your score if within one day late beyond grace period
  - although in the first 100 minutes of this period, you will only lose 1% of your grade every 5 minutes
  - see next page for details
  - time is based on *Bistro server timestamp*
- **Extension *only possible* if you have a *note from a doctor* or other form of official proof of family emergencies**
  - e.g., scheduling conflict with work or other classes cannot be excused



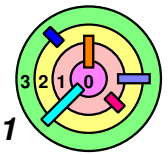
Late Policy





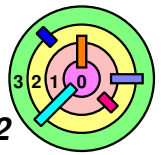
Late Policy

- I must stick to my policies
  1) please do not ask for individual extension unless you have a *documented* proof of *illness* or a *documented* proof of *family (not personal) emergency*
  2) my "fairness" policy is: *"Whatever I offer you, I must offer it to the whole class"*
     - this is why I cannot give individual extensions
- what if your laptop died? home Internet disrupted? car broken? cousin got stuck at the airport? my house was on fire? need to go to court? need to go to Miami? and so on ...
  - these are personal emergencies
  - please see (1) and (2) above



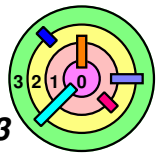
Modifications After Deadline

- After the submission deadline has past
  - you are allowed up to 3 lines of free changes, submitted via e-mail to the instructor, *within 24 hour* of the project submission deadline
    - clearly, this is not meant for major changes
    - you may want to anticipate that your submission may not be exactly what you thought you had submitted
  - one line (128 characters max) of change is defined as one of the following:
    - *add* 1 line before (or after) line x
    - *delete* line x
    - *replace* line x by 1 line
    - *move* line x before (or after) line y
  - additional modifications at 3 points per line (same deadline)
  - (cont...)



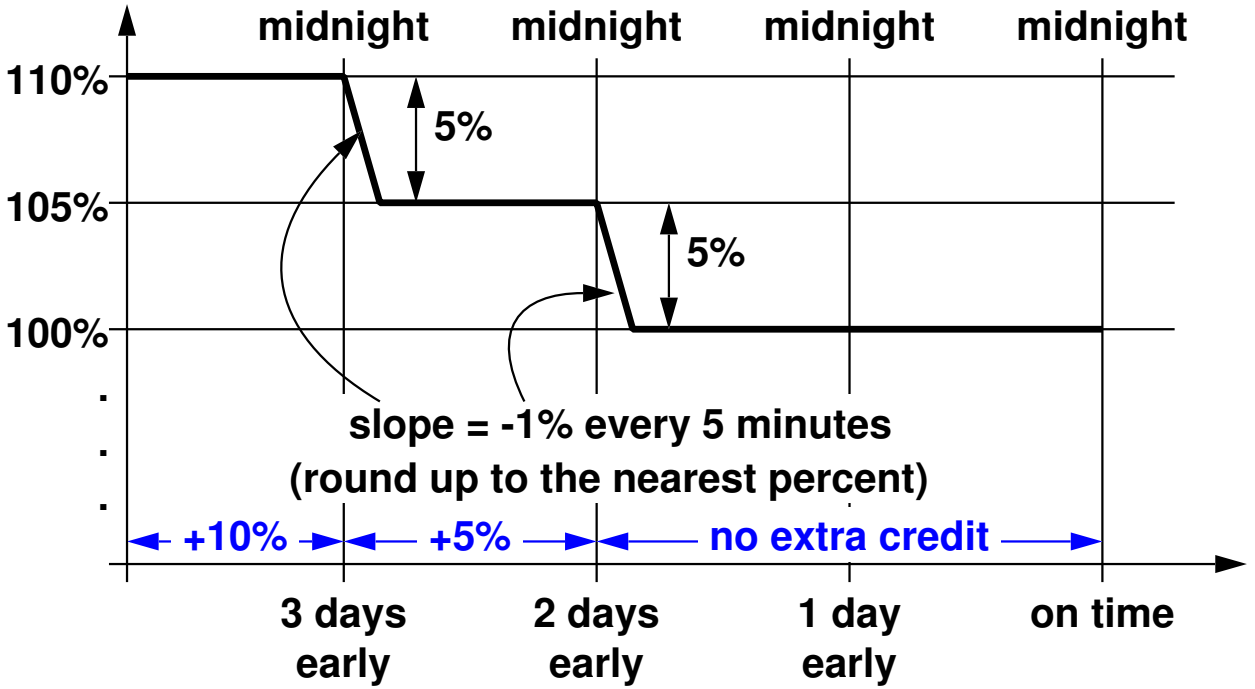
Modifications After Deadline (cont...)

- After the submission deadline has past (cont...)
  - 24 hours after the submission deadline, additional modifications cost *12 points per line* for the next 6 days
  - afterwards, it costs *30 points per line*
  - applies to source code and README files
    - do not forget to submit files, *verify* your submission
      - this is *your responsibility*
    - we cannot accept missing files after deadline
    - a *filesystem timestamp* can be easily *forged*, so they cannot beused as *proof* that you have not modified the file after deadline
  - try things out before your first submission deadline to get familiar with the *Bistro* system
  - *re-test* your code after you have submitted to be sure



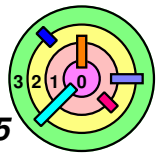
Extra Credit 1

- To encourage you to do your projects early, you will get extra credit if you turn in programming assignment 2 or 3 days early
  - if your submission is more than 72 hours before the posted deadline, you get an ***extra 10%***
  - if your submission is between 48 and 72 hours before the posted deadline, you get an ***extra 5%***



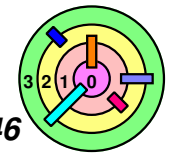
Extra Credit 2

- You can get extra credit for posting good, useful, and insightful answers to the class Google Group in response to questions posted by other students regarding *kernel* programming assignments
  - the maximum number of extra credit points you can get is *10* points for each of the kernel assigments (on a 100-point scale)
  - if you post something good, it's your responsibility to verify that it has been posted to the Google Group
  - you can *lose* extra points you have earned if you post something unprofessional to the class Google Group or exhibit poor netiquette
  - please post to the right Google Group
    - you will get *no credit* if you post to the wrong group



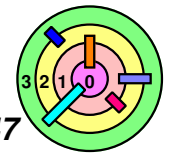
Regrade Policy

- Grades will be sent to individuals via e-mail
  - you must register for the class mailing list
- Regrade requests in writing
  - submit within 1 week of initial grade notification
    - must follow instruction in grade notification e-mail
  - regrade can be done after the 1-week deadline, but you must *initiate* a regrade request within 1 week
  - we reserve the right to regrade the whole thing



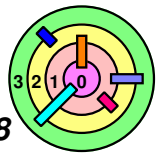
Academic Integrity Policy

- The USC Student Conduct Code *prohibits plagiarism*
  - for warmup projects, all submitted work must be *your own work*
  - for group projects, all submitted work must be work done by members of your group
- You can look at other people's code, but you must not copy a single line of code
  - you must turn in code that's *completely written by yourself*
    - if you don't know how to do that, you should not look at any code by other people!
- Two exceptions
  - you can use any code given to you as part of this class
  - code fragments done by yourself for other classes or given to you as class resource (in a class you have taken and completed) must be cited explicitly
    - must cite the code *inline* (or points will be deducted)



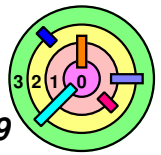
Academic Integrity Policy (Cont...)

- You are encouraged to work with other students for individual assignments or with other groups for group projects
  - "work with" does not mean "copy each other's work"
  - "work with" means discussing and solving problems together
    - this should happen at a *high level*
  - but be very careful when it's time to write code
    - must write code completely on your own
    - *do not write code together*
    - "sharing" even a *single* line of code is considered *cheating*
- If you cannot work together at a high level
  - you are advised not work together with other students for individual assignments or with other groups for group projects
- For more details, please see the *Academic Integrity Policy* section on the *Course Description* web page



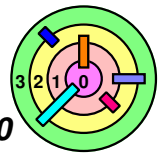
Program Checker

- Do not submit someone else's code
- How do we catch cheaters?
  - we use MOSS to analyze your submissions
    - http://theory.stanford.edu/~aiken/moss/
  - analyzes code structure intelligently
  - we have *all* projects from previous semesters



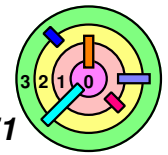
Student Commitments

- Keep up with your reading
  - complete relevant reading before class
  - browse lecture slides before class
    - lecture slides will be available on-line before class
- Do your own work
- Turn in assignments on time
- Ensure gradable assignments
  - read output of bsubmit
  - *verify* your submissions
- You are encouraged to study with other students and *discuss* (no sharing) programming assignments and HWs
- You are encouraged to ask questions, pretty much about anything related to programming
  - when you get stuck with programming, ask the TA or the instructor for help, don't wait too long



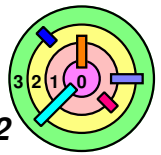
Student Commitments (Cont...)

- If you feel that you are falling behind
  - talk to the instructor as soon as possible
- How do you know you are in trouble?
  - assuming that you want a decent grade
  - you look at a set of slides that has been covered in class and have no idea what it means
    - then you read the textbook and understands it perfectly
      - you are probably not in trouble
    - if you read the textbook and are still confused
      - you are somewhat in trouble
    - if you don't read the textbook
      - you are probably in ***big trouble***
  - don't do this right before exams!
  - you should ask youself, "Am I in trouble?" like at least every week if you don't plan to come to lectures



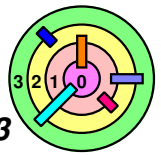
Study for Exams

- Exams are worth a lot! This class is not just about programming!
  - how many kernel programmers do we need in this world?
  - but everyone in CS must understand *OS concepts* well
- Exams mostly are based on lectures
  - I reserve the right to ask anything from required materials
- Exam questions often asks for the *best* answer
  - you get credit for including the "best answer"
  - you may get deductions for including "bad answers"
  - generic answer usually gets you very little partial credit
  - partial credit: better answer may get you more points
- Do not review all lectures only right before the exams
  - otherwise, you may only be able to give generic answers because everything is a blur
  - you need to show that you know the difference between answers of different quality



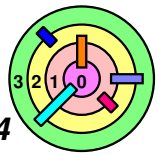
Things to Do *Today*

- Read entire class web page: http://merlot.usc.edu/cs402-f13
  - check course description and reading list
  - get access to user ID and password
    - should do this even if you are not registered for the class but plan to take this class
  - check warmup project spec and *start coding*
- Additional things to learn/do quickly
  - learn C if you don't know it already
  - learn "git" (see online book)
    - you will need it for group projects
    - should start using it for the warmup projects
  - learn a commandline editor: vim/pico/emacs
    - it's not that hard
  - install *Ubuntu 11.10* on your laptop/desktop and let me know if there are problems
    - VirtualBox preferred if you have Intel Core i3 or faster



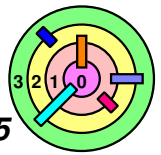
Course Content Credit

- Slides and course content primarily came with the textbook and originally written by:
  - Tomas W. Doeppner
- Additional slides and course content from:
  - Ramesh Govindan



ITS Solaris Machine Access

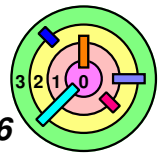
- You need to log into aludra/nunki.usc.edu
  - for warmup assignments and to run `bsubmit`
  - SSH from a console (make sure to use `"ssh -X -Y ..."`)
  - On Windows, use VirtualBox, Xwin, Cygwin or PuTTY
    - Ubuntu (Linux)
- Transferring Files
  - `"scp"` from a console
  - SFTP/SCP programs
    - Cyberduck, Fugu, etc. (Mac)
    - FileZilla, WinSCP, etc. (Windows)
- Text Editors
  - `emacs`, `pico`, `vi`
- Compiler
  - `"gcc --version"` should say it's version 4.something



Warmup #1

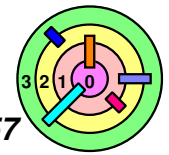
Bill Cheng

http://merlot.usc.edu/cs402-f13



Programming & Good Habbits

- ***Always*** **check return code!**
  - `open(), write()`
  - `malloc()`
  - `switch (errno) { ... }`
- **Initialize *all* variables!**
  - `int i=0;`
  - `struct timeval timeout;`
    `memset(&timeout, 0, sizeof(struct timeval));`
- ***Never*** **leak any resources!**
  - `malloc() and free()`
  - `open() and close()`
  - **Delete temporary files**

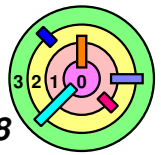


Programming & Good Habbits

- ***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

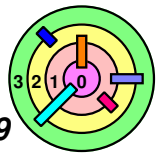


Notes on gdb

- The debugger is your friend! Get to know it!

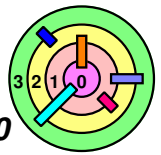
compile program with:	`-g`
start debugging:	`gdb [-tui] warmup1`
set breakpoint:	`(gdb) break foo.c:123`
run program:	`(gdb) run`
clear breakpoint:	`(gdb) clear`
stack trace:	`(gdb) where`
print field:	`(gdb) print f.BlockType`
printf():	`(gdb) printf "%02x\n",buf[0]`
single-step at same level:	`(gdb) next`
single-step into a function:	`(gdb) step`
print field after every cmd:	`(gdb) display f.BlockType`
assignment:	`(gdb) set f.BlockType=0`
continue:	`(gdb) cont`
quit:	`(gdb) quit`

- Start using the debugger with warmup 1!
  - get help from TA, course producer, and me



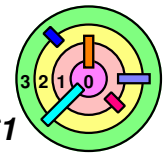
General Requirements

- Some major requirements for all programming assignments
  - severe pentalty for failing `make`
    - we will attempt to fix your Makefile you `make` fails
    - if we cannot get it to work, you need to figure out how to fix it by regrade time
  - severe pentalty for using large memory buffers
  - severe pentalty for any segmentation fault -- you must test your code well
  - 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 pentalty 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



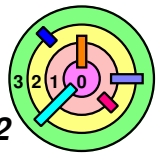
Grading Requirements

- It's important that every byte of your data is read and written correctly.
- For *warmup assignments*, you should run your code against the *grading guidelines*
  - 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 is *the only way we will grade*



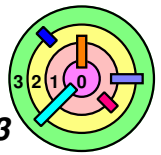
Separate Compilation

- 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 requites 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*



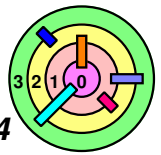
Code Design - Functional vs. Procedural

- Don’t design your program "procedurally"
- You need to learn how to write functions!
  - 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



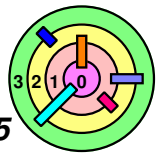
Warmup #1

- 2 parts
  - develop a *doubly-linked circular list* called *My402List*
    - 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



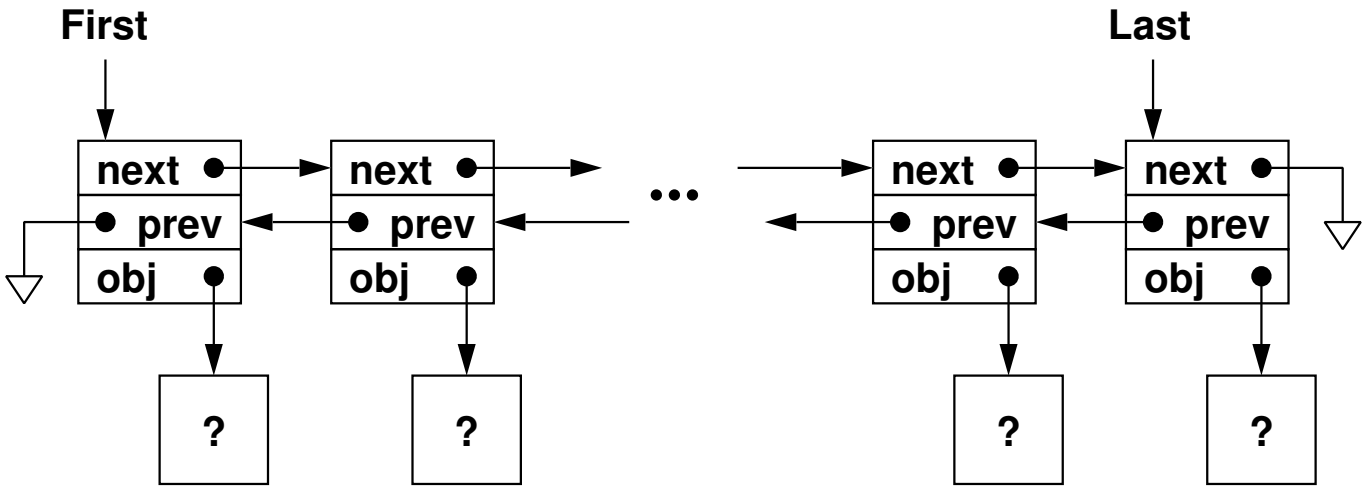
A Linked-List Abstraction

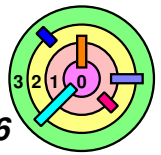
- A list of elements, linked so that you can move from one to the next (and/or previoius)
  - each element holds an object of some sort
- *Functionally:*
  - First()
  - Next()
  - Last()
  - Prev()
  - Insert()
  - Remove()
  - Count()
- Need to have a well-defined interface
  - once you have a good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



A Linked-List Abstraction

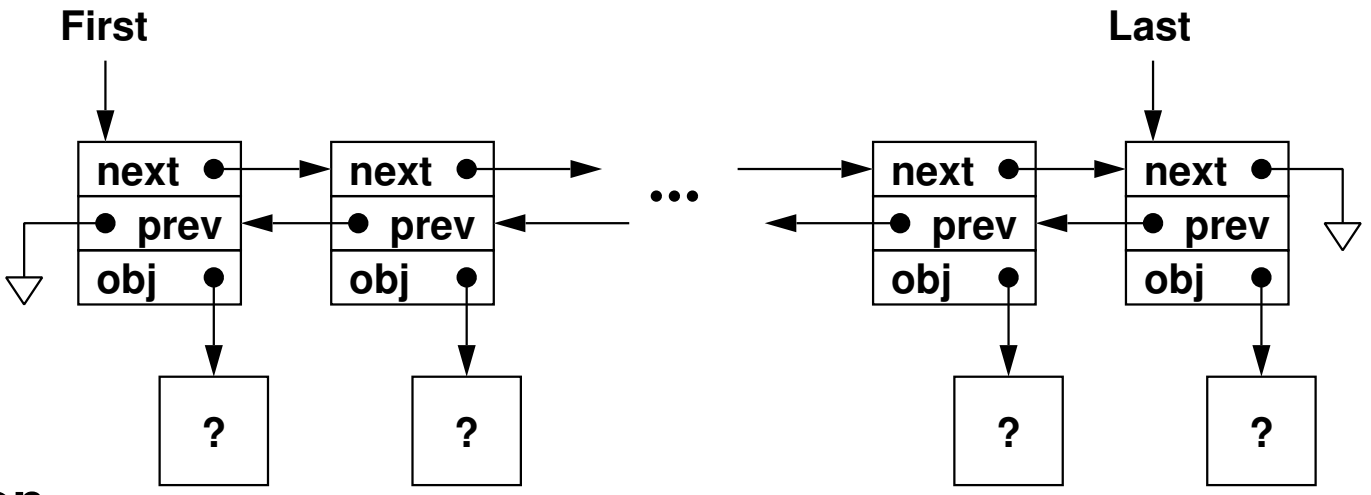
- There are basically two types of lists
  1) next/prev pointers in list items
  2) next/prev pointers outside of list items
- (1) has a major drawback that a list item cannot be inserted into multiple lists
  - We will implement (2)



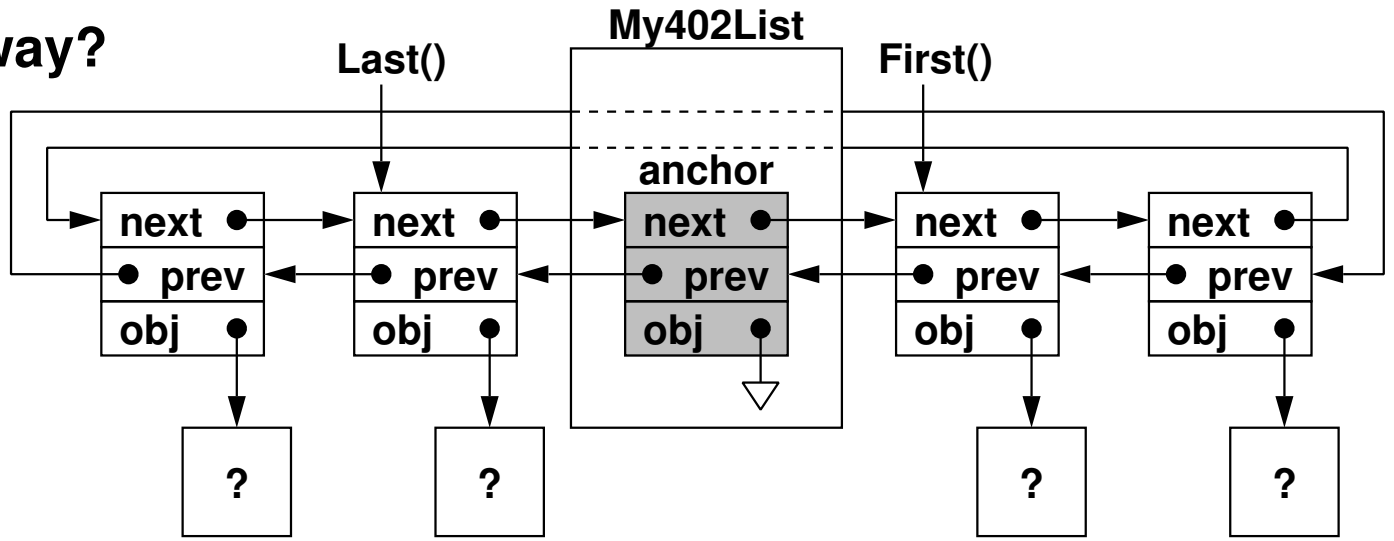


Doubly-linked Circular List

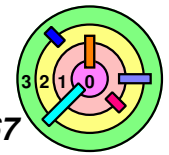
- **Abstraction**



- **Implementation**
  - why this way?

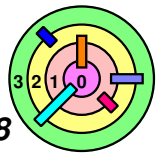


  - your job is to implement the traditional list abstraction using a circular list



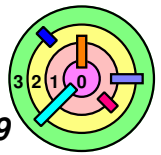
my402list.h

```
#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 *);
```

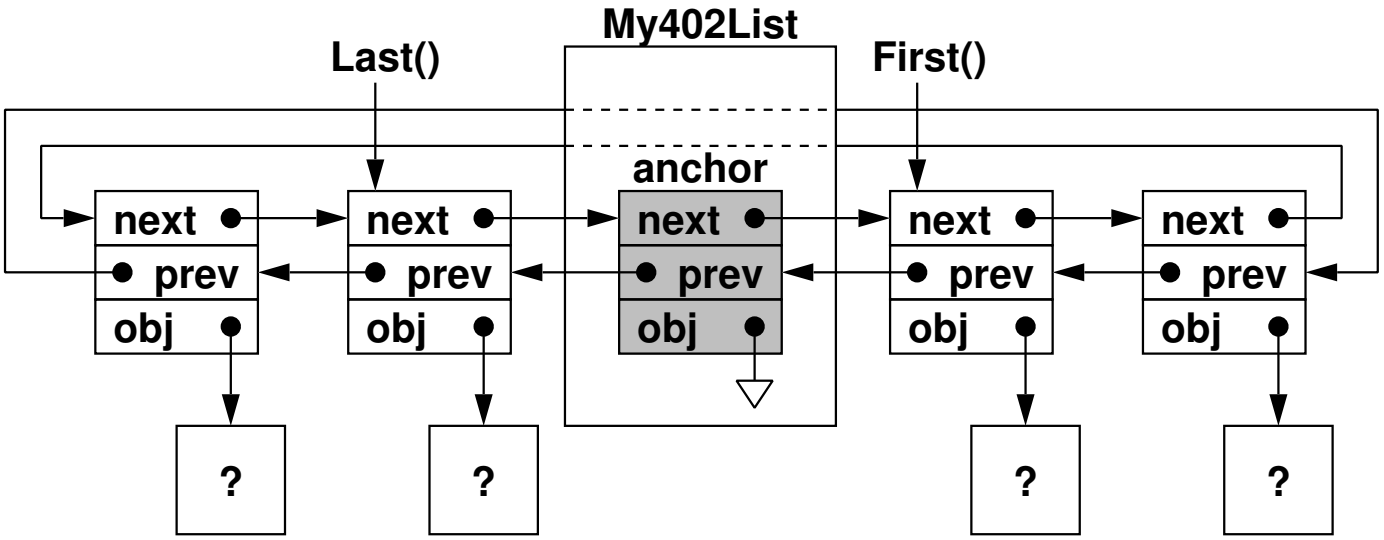


my402list.h

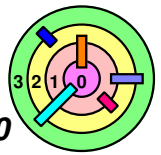
```
    int (*InsertBefore)(struct tagMy402List *, void*, My402ListElem*);
    int (*InsertAfter)(struct tagMy402List *, void*, My402ListElem*);

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

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

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

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

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

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

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



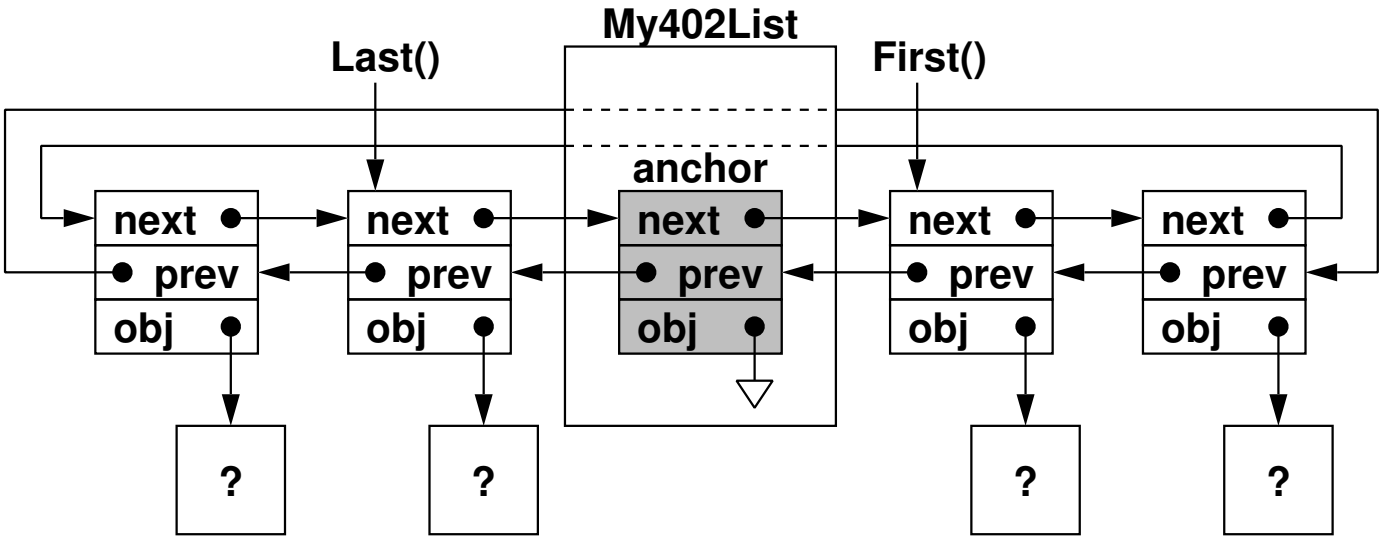
Implementation




```
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);
```



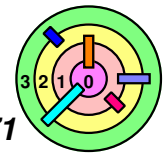
Usage - Traversing the List




```
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 */
  }
}
```

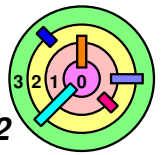
⇨ This is how an *application* will use `My402List`

- you must support your application



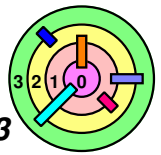
listtest

- Use provided `listtest.c` and `Makefile` to create `listtest`
  - `listtest` must run without error and you must not change `listtest.c` and `Makefile`
  - They specifies how your code 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
  - Each line in a `tfile` contains 4 fields delimited by `<TAB>`
    - transcation type (single character)
      - `"+"` for deposit
      - `"-"` for withdrawal
    - transcation time (UNIX time)
      - `man -s 2 time`
    - amount (a number, a period, two digits)
    - transcation description (textual description)
      - cannot be empty



Reading Text Input

- Read in an entire line using `fgets()`
  - especially since we know the maximum line length, according to the spec
- If a filename is given, use `fopen()` to get a file pointer (`FILE*`)

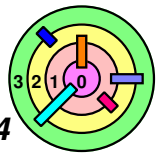
```
FILE *fp = fopen(..., "r");
```

  - read man pages of `fopen()`
  - if a filename is not given, you will be reading from "standard input" (i.e., file descriptor 0)

```
FILE *fp = stdin;
```

  - pass the file pointer around so that you run the same code whether you input comes from a file or `stdin`

```
My420List list;
if (!My402ListInit(&list)) { /* error */ }
if (!ReadInput(fp, &list)) { /* error */ }
if (fp != stdin) fclose(fp);
SortInput(&list);
PrintStatement(&list);
```



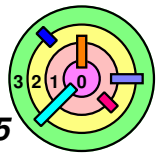
Parsing Text Input

- Read a line

```
char buf[1026];
if (fgets(buf, sizeof(buf), fp) == NULL) {
  /* end of file */
} else {
  /* parse it */
}
```

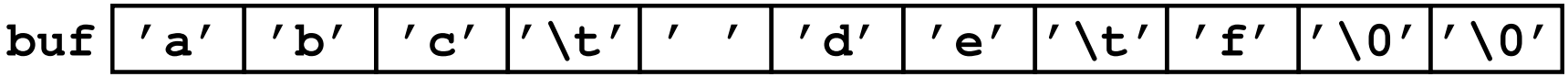
- Parse a line according to the spec
  - find an *easy* and *correct* way to parse the line
    - according to the spec, each line must have exactly 3 <TAB> characters
    - I think it's easy and correct to go after this

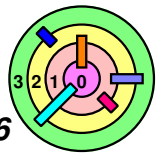
```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



Parsing Text Input

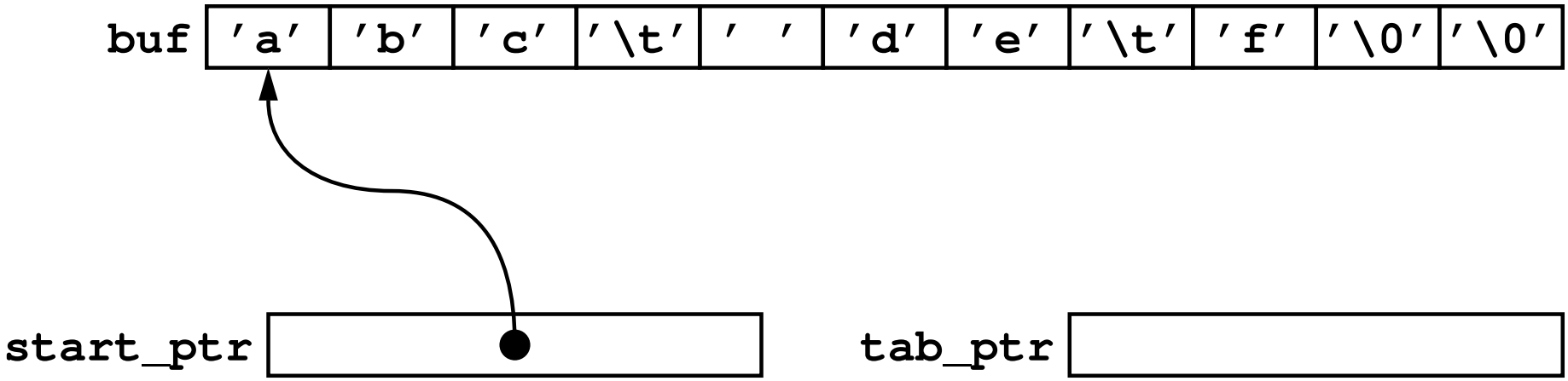
```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```

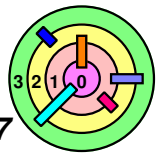




Parsing Text Input

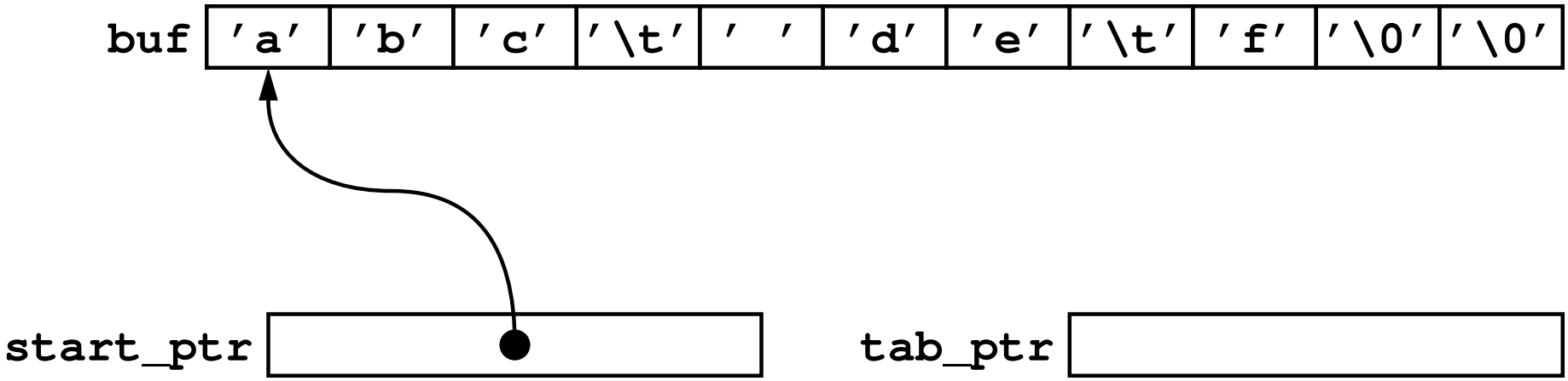
```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```

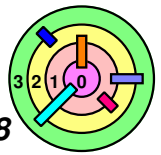




Parsing Text Input

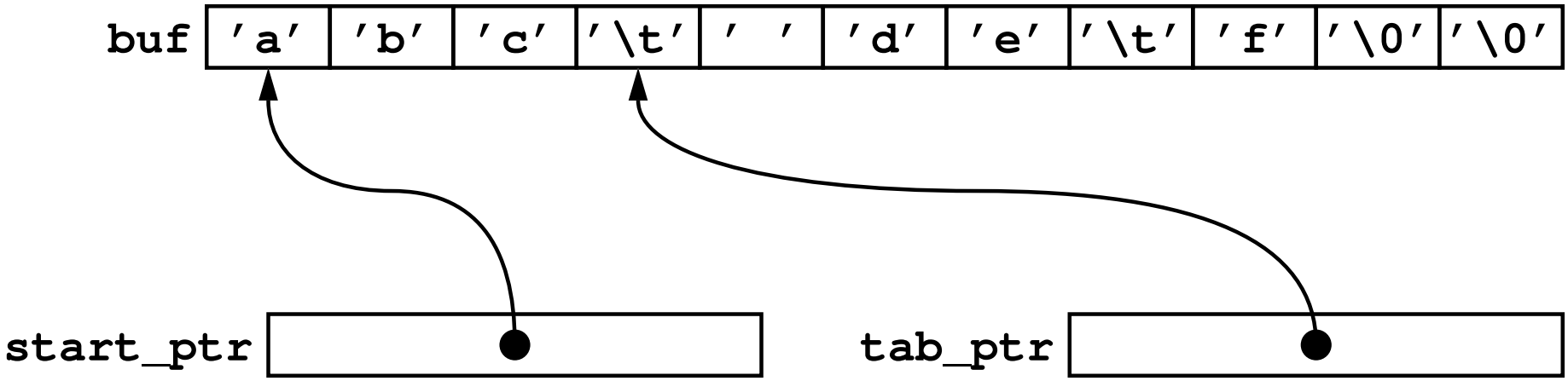
```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```

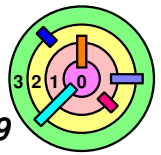




Parsing Text Input

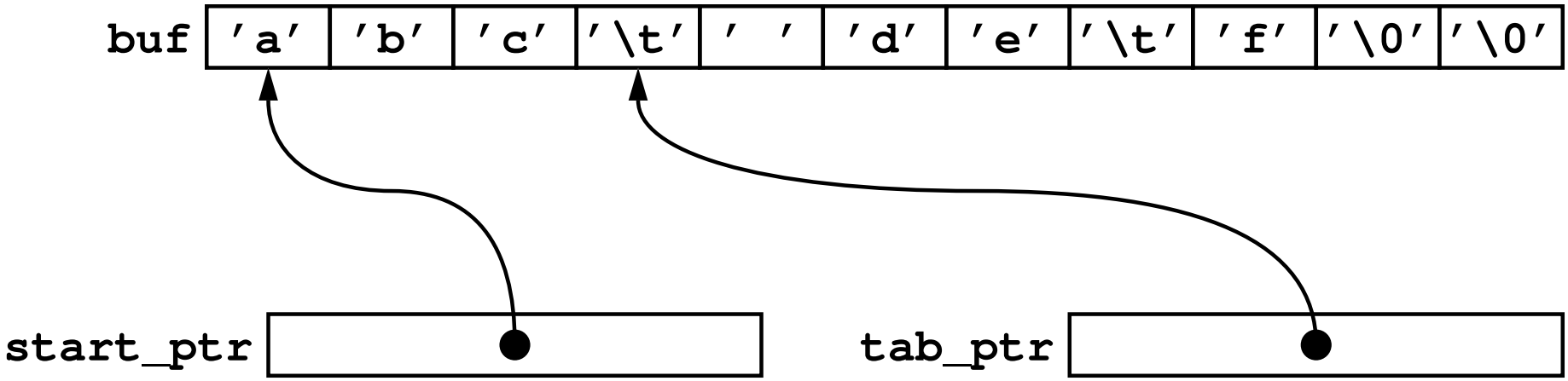
```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```

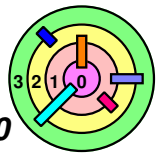




Parsing Text Input

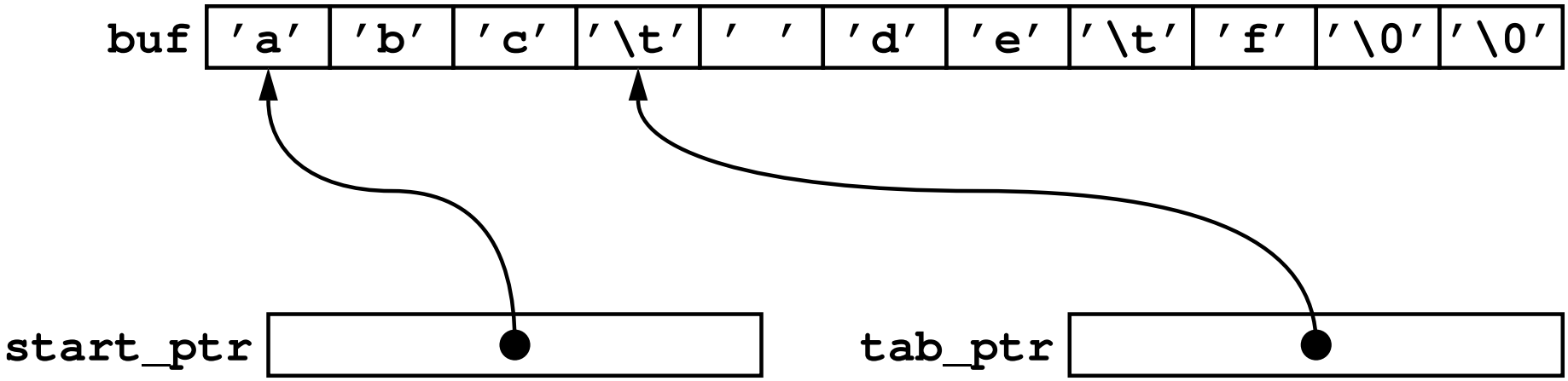
```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```

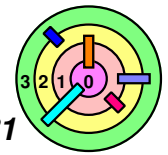




Parsing Text Input

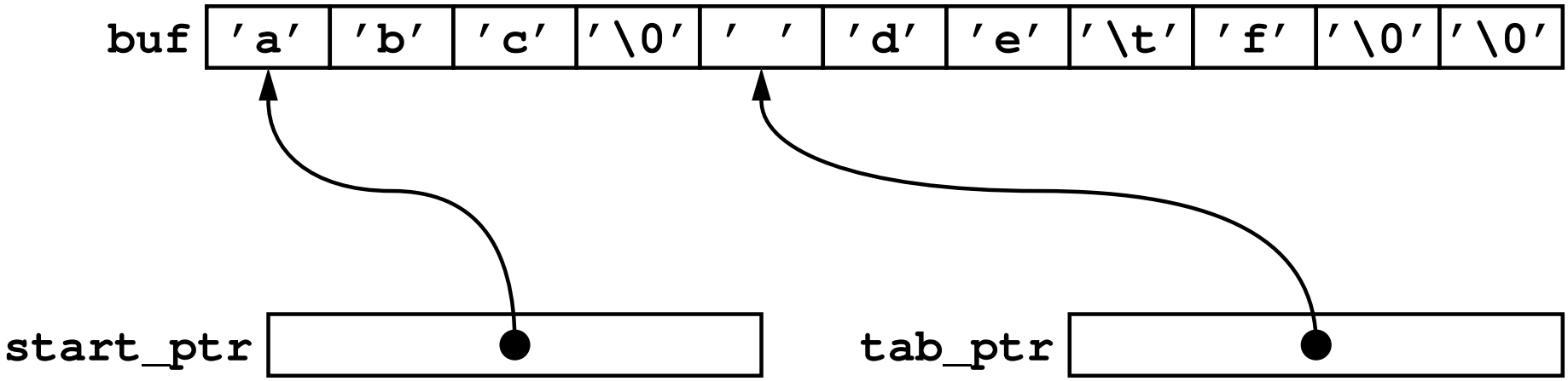
```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```

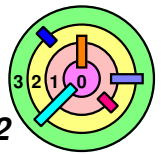




Parsing Text Input

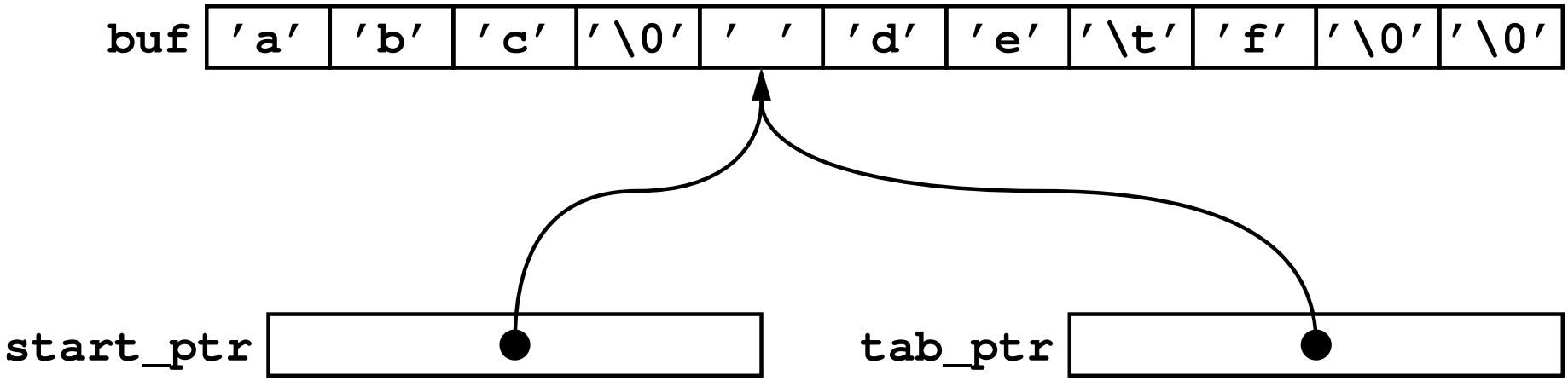
```
char *start_ptr = buf;
char *tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```

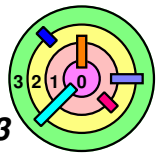




Parsing Text Input (2nd Iteration)

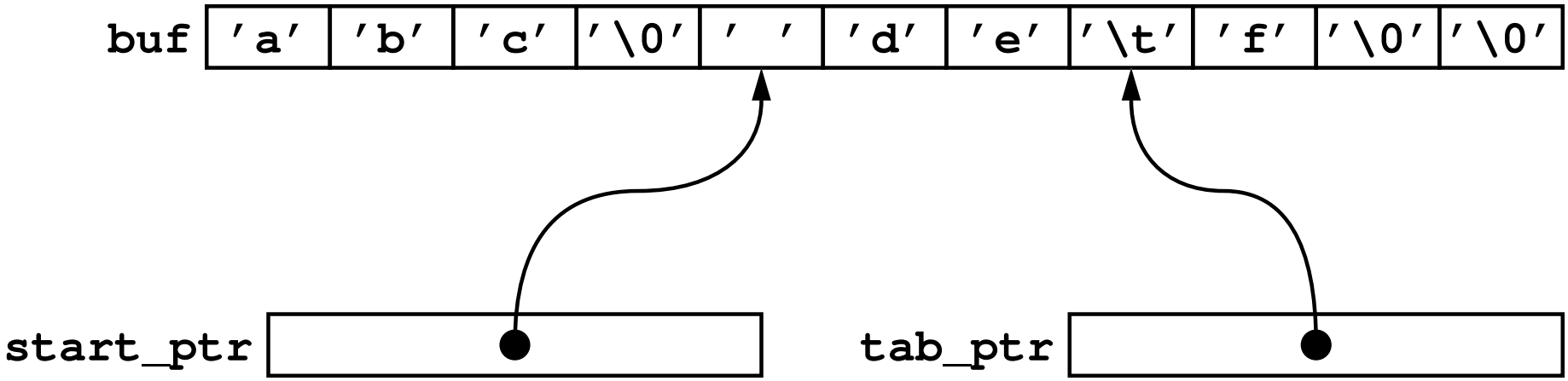
```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```

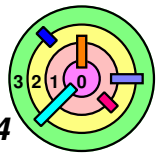




Parsing Text Input (2nd Iteration)

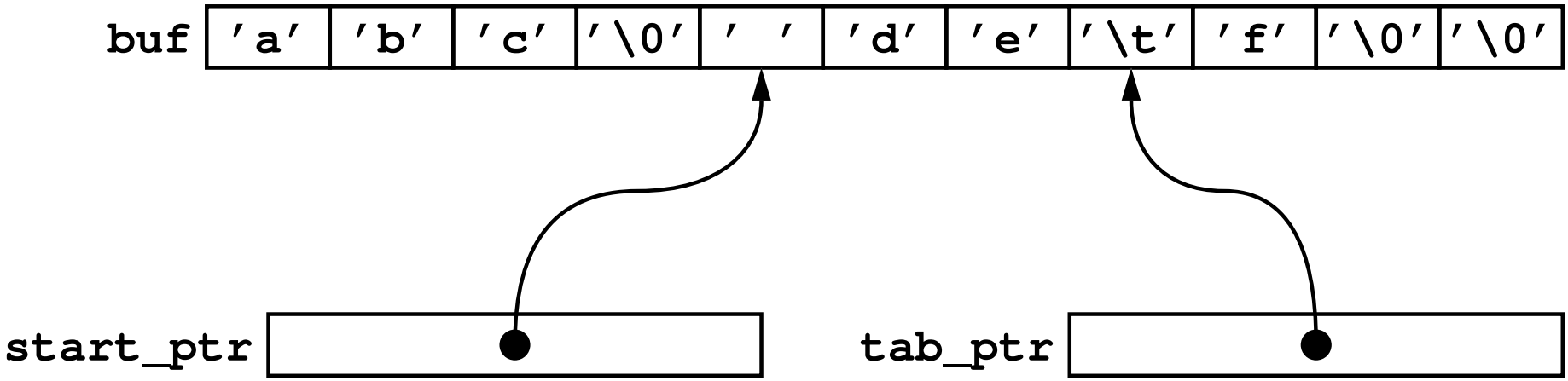
```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```

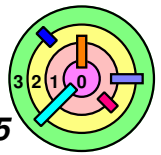




Parsing Text Input (2nd Iteration)

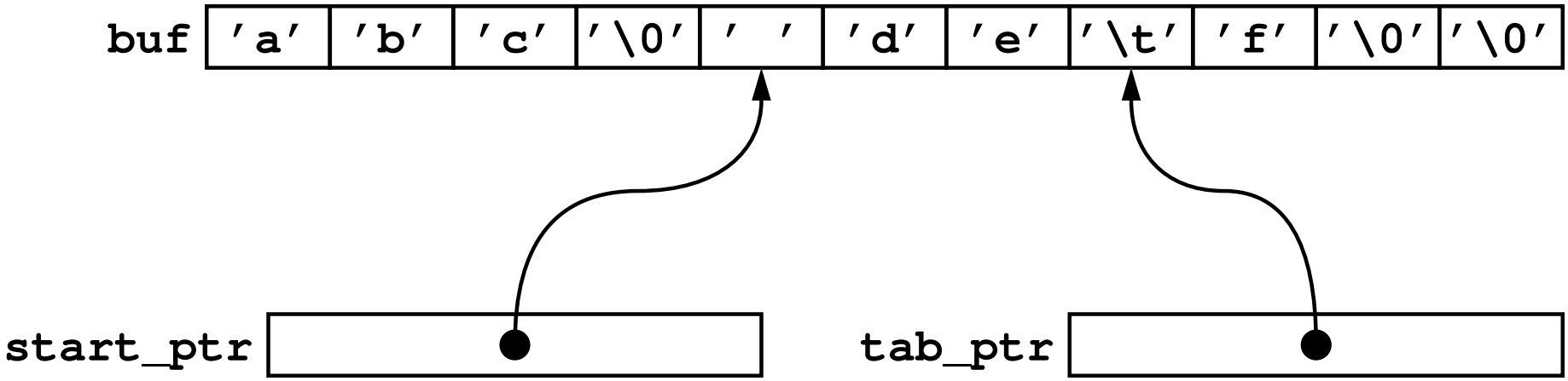
```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```





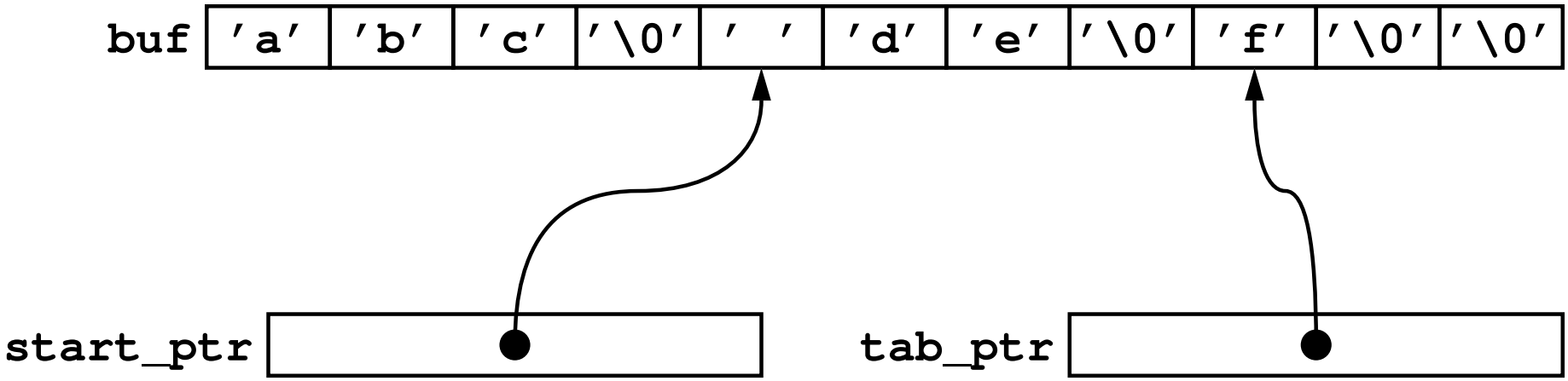
Parsing Text Input (2nd Iteration)

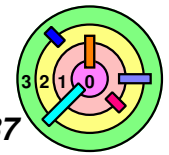
```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



Parsing Text Input (2nd Iteration)

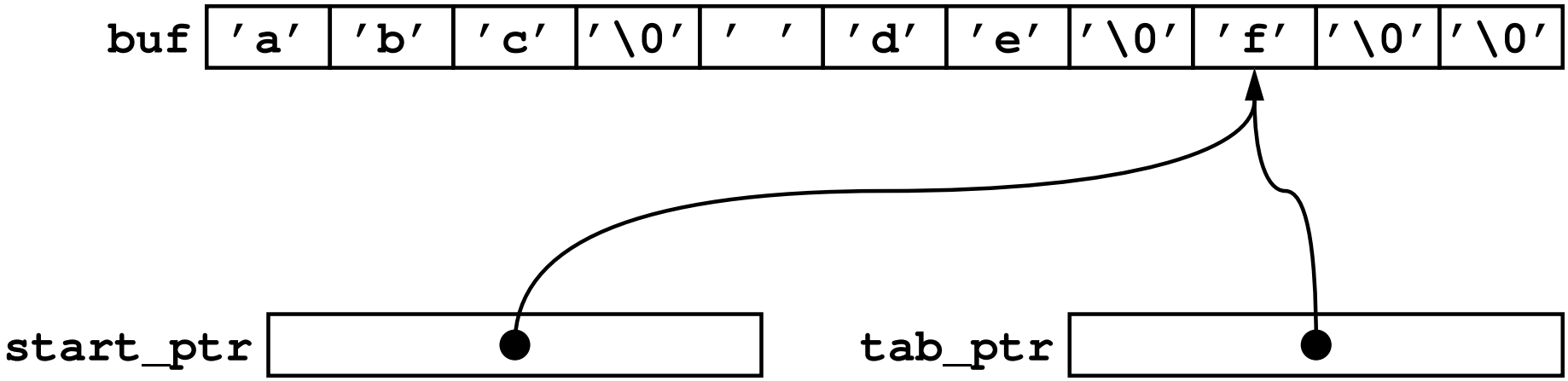
```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```

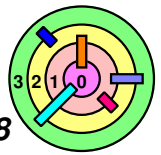




Parsing Text Input (3rd Iteration)

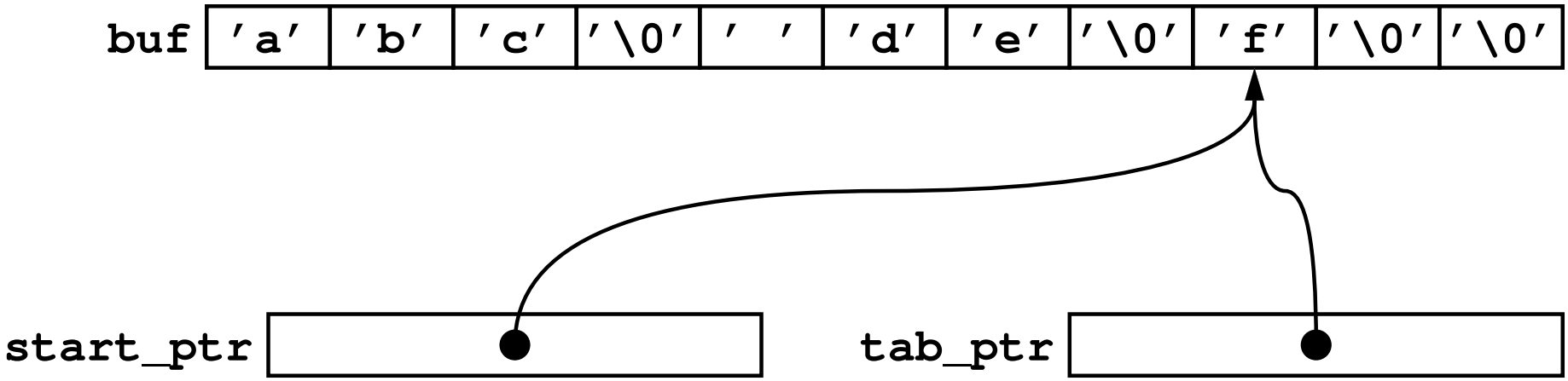
```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```

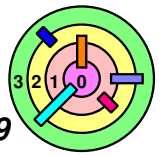




Parsing Text Input (3rd Iteration)

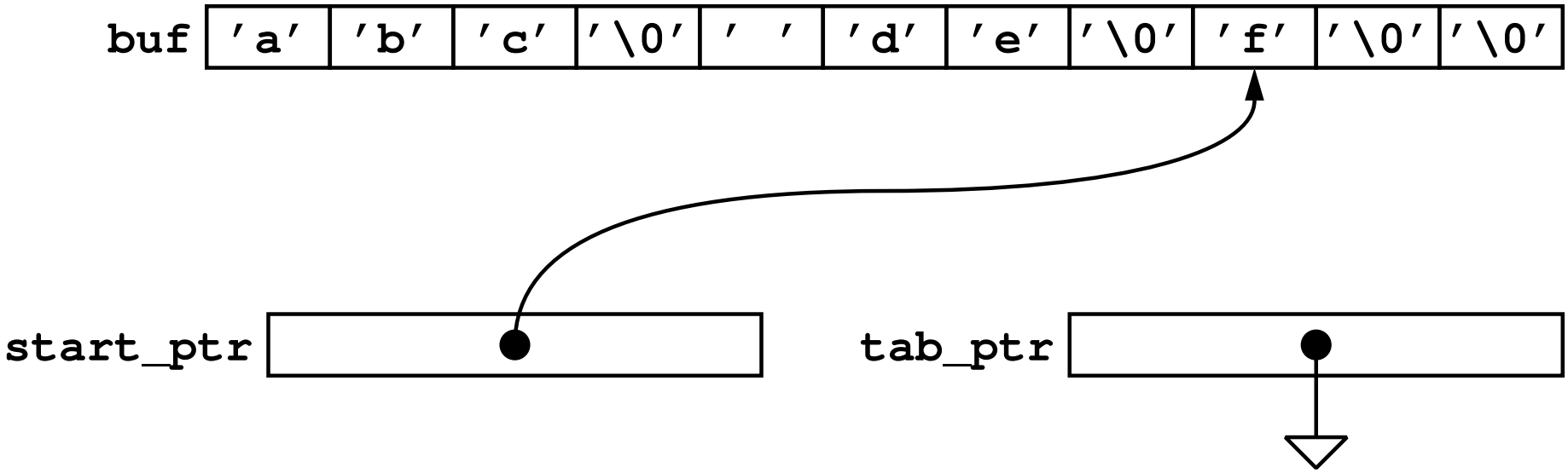
```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```

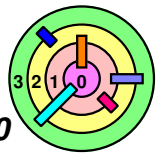




Parsing Text Input (3rd Iteration)

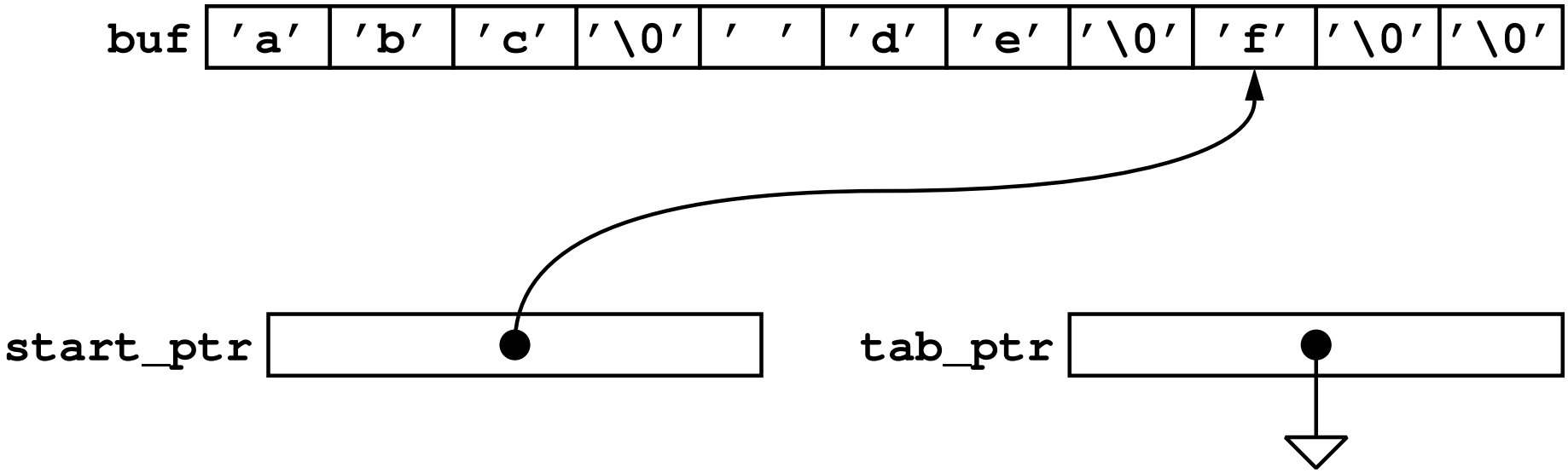
```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```





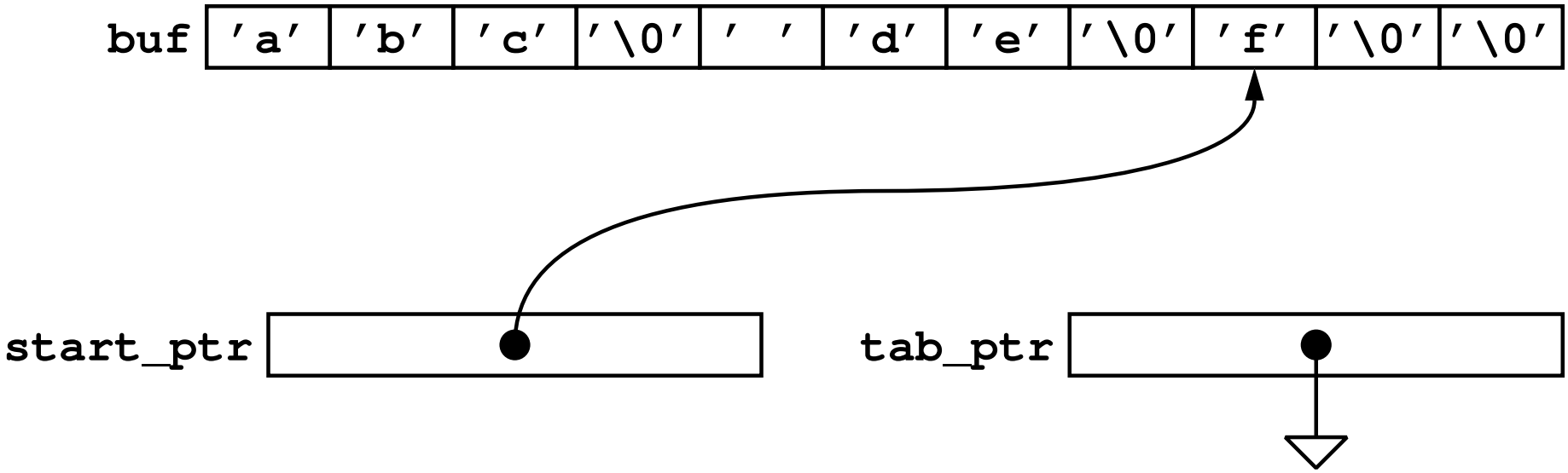
Parsing Text Input (3rd Iteration)

```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



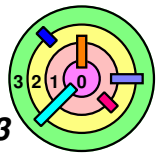
Parsing Text Input (3rd Iteration)

```
start_ptr = tab_ptr;
tab_ptr = strchr(start_ptr, '\t');
if (tab_ptr != NULL) {
  *tab_ptr++ = '\0';
}
/* start_ptr now contains a
   "null-terminated string" */
```



Validate Input

- Make sure every null-terminated string contains the right kind of value
  - if incorrect, print a *reasonable error message* and *quit* your program
    - ideally, you should clean up all your data structures (not required for an assignment like this one)
- After all fields are validated, you can put them in *one* data structure
  - allocate memory for this data structure and copy the fields into it
  - *append* pointer to this data structure to `list`
    - any pointer is compatible with (`void*`)
    - alternately, you can perform insertion sort by finding the right place to insert this pointer and call one of the insert functions of `My402List`



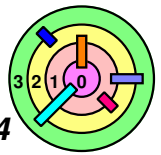
Sort Command

Output

```
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 |
+-----------------+--------------------------+----------------+----------------+
```

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!

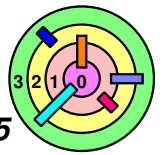


Sort Command

```
000000000111111111122222222223333333333444444444455555555556666666666777777777 8
123456789012345678901234567890123456789012345678901234567890123456789012345678 90

+-----------------+--------------------------+----------------+----------------+
|       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';
```

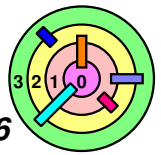


Sort Command

```
0000000001111111111222222222233333333334444444444555555555566666666667777777778
1234567890123456789012345678901234567890123456789012345678901234567890123456789
```

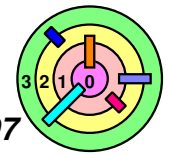
Date	Description	Amount	Balance
Thu Aug 21 2008	...	1,723.00	1,723.00
Wed Dec 31 2008	...	(45.33)	1,677.67
Mon Jul 13 2009	...	10,388.07	12,065.74
Sun Jan 10 2010	...	(654.32)	11,411.42

- **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`**



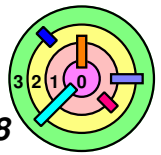
Warmup #1

- 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 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



Warmup #1 - Miscellaneous Requirements

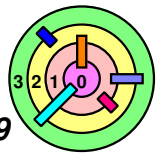
- Run your code against the *grading guidelines*
  - must not change the test program
- You must not use any *external code fragments*
- You must not use *array* to implement any list functions
  - must use pointers
- If input file is large, you must not read the whole file into into a large memory buffer
- 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
- Please see Warmup #1 spec for additional details
  - please read the *entire* spec *yourself*



Ch 1: Introduction

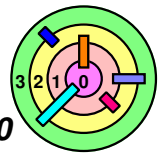
Bill Cheng

http://merlot.usc.edu/cs402-f13



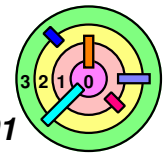
What are Operating Systems?

- **Possible definitions:**
  - **the code that {Microsoft, Apple, Linus, Google} provides**
  - **the code that you didn't write**
  - **the code that runs in privileged mode**
  - **the code that makes things work**
  - **the code that makes things crash**
  - **etc.**



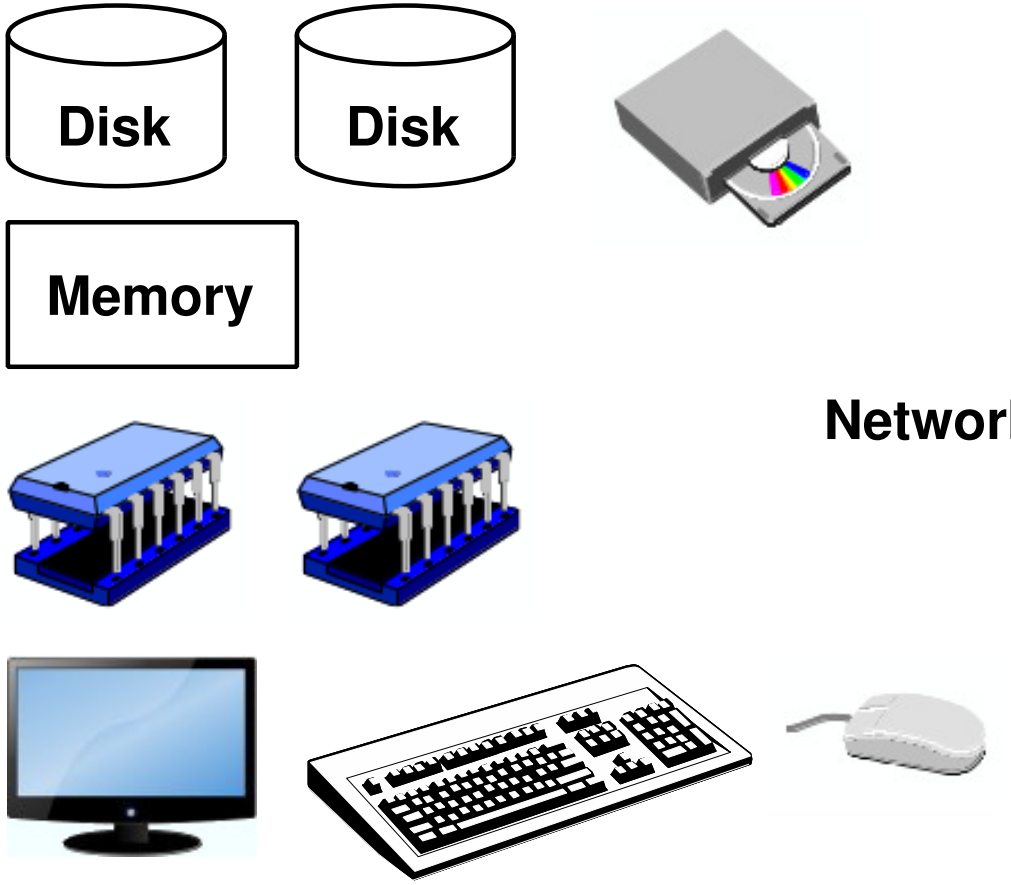
Operating Systems

- **Abstraction**
  - providing an "appropriate" interface for applications
  - but abstraction to what? (next slide)
- **Concerns**
  - performance
    - time, space, energy
  - sharing and resource management
  - failure tolerance
  - security
  - marketability



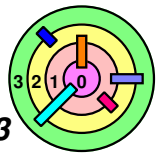
Hardware

- Hardware
  - disks
    - hard drives
    - optical drives
  - memory
  - processors
  - network
    - ethernet
    - modem
  - monitor
  - keyboard
  - mouse

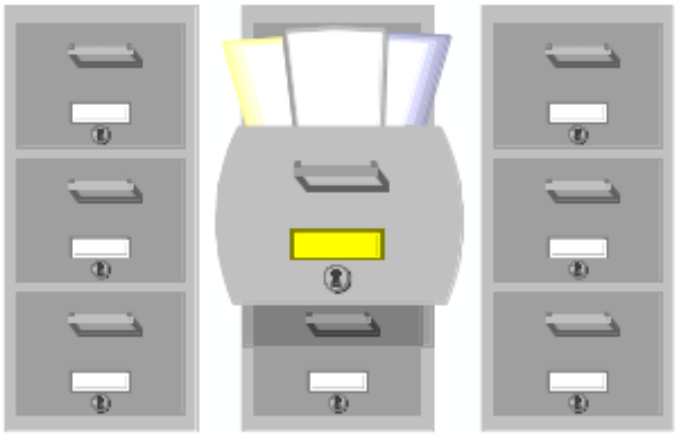


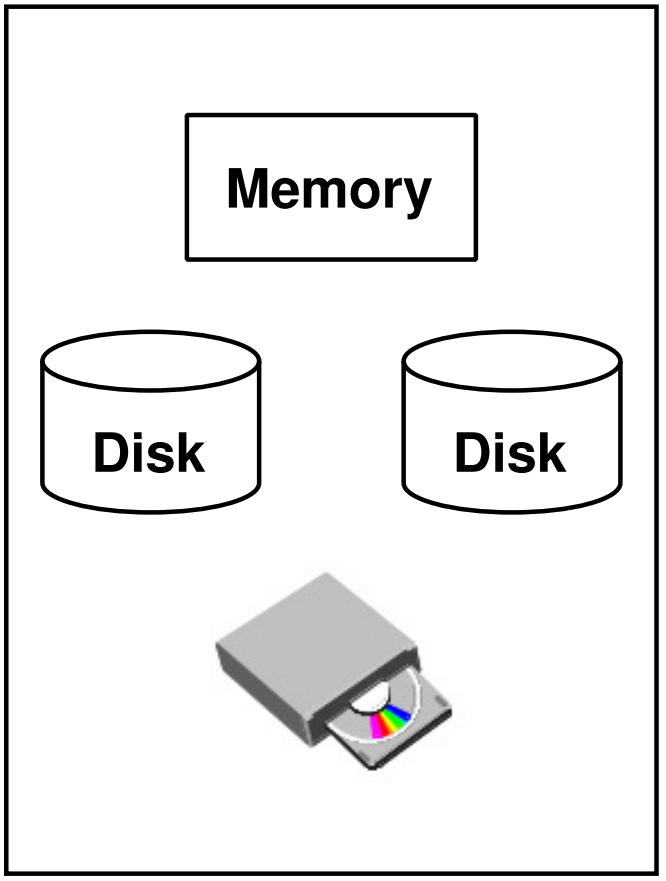
OS Abstractions

- **Hardware**
  - disks
  - memory
  - processors
  - network
  - monitor
  - keyboard
  - mouse
- **Operating system**
  - files
  - programs
  - threads of control
  - communication
  - windows, graphics
  - input
  - locator

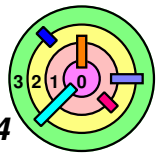


Abstraction Example: Files





- It's nice to have a simple abstraction
- Abstraction did not come for free
  - it introduces problems that need to be solved and issues to be addressed



Issues With The Files Abstraction

- Naming
- Allocating space on disk (permanent storage)
  - organized for fast access
  - minimize waste
- Shuffling data between disk and memory (high-speed temporary storage)
- Coping with crashes