A4: Deque Implementation: Air-Gapped Coding

Overview

You will implement a deque (double-ended queue) in an **air-gapped environment without internet access**. This assignment emphasizes independent problem-solving and effective debugging strategies.

Environment Rules

Air-Gapped Coding Session

- You must complete the majority of your coding work **without internet access**
- Disconnect from Wi-Fi or use the designated lab computers with no network connection
- You may use **offline resources only**: course notes, textbook, previously downloaded materials, and your own code from prior assignments
- No communication tools (Slack, email, messaging apps) during air-gapped sessions

Allowed Offline Resources

- Course lecture slides and notes (downloaded beforehand)
- Your textbook and reference books
- Standard IDE documentation (built-in help)
- Your own previous assignments and code
- Compiler error messages and debugging tools (GDB, debugger, valgrind)

Prohibited During Air-Gapped Sessions

- Internet searches (Google, Stack Overflow, documentation websites)
- AI assistants (ChatGPT, Copilot, Claude)
- Online forums or discussion boards
- Synchronous communication with peers or TAs
- Code repositories (GitHub, GitLab) unless cloned locally beforehand

---

Why Air-Gapped Coding?

The Problem with Always-On Internet Access

- **Instant gratification undermines learning**: Immediately searching for solutions prevents you from developing problem-solving muscles
- **Surface-level understanding**: Copying solutions without struggle leads to shallow knowledge that evaporates after the exam
- **False confidence**: Students who rely heavily on internet resources often can't perform basic tasks in technical interviews or on-the-job
- **Dependency cycle**: The more you search immediately when stuck, the less capable you become of working independently

Real-World Relevance

Technical Interviews

- Most coding interviews are conducted on **whiteboards or locked-down systems** without internet access
- Interviewers want to see **your problem-solving process**, not your Googling skills
- Companies explicitly test your ability to work through problems with limited resources

Professional Development Environments

- Many companies work with **proprietary codebases** and technologies not documented online
- Secure environments (defense, finance, healthcare) often require **air-gapped development** for security reasons
- Senior developers aren't always available immediately—you need to make progress independently
- Debugging production issues may require working **without external resources** under time pressure

Fundamental Skill Development

- Reading compiler errors and documentation carefully
- Systematic debugging and hypothesis testing
- Building mental models of how code executes
- Developing persistence and frustration tolerance
- Learning to learn from your own mistakes

What Research Shows

- **Desirable difficulty**: Struggling with problems (within reason) leads to **deeper learning and better retention**
- **Productive failure**: Students who wrestle with problems before receiving instruction learn more than those given immediate solutions

- **Transfer of learning**: Skills developed through independent problem-solving transfer better to novel situations
- **Metacognition**: Forcing yourself to articulate problems (as required in token forms) enhances understanding

What This Assignment Teaches

Technical Skills

- Deep understanding of deque implementation, not just surface-level copying
- Reading and interpreting compiler messages and documentation
- Systematic debugging methodologies
- Code tracing and mental execution

Professional Skills

- Persistence in the face of difficulty
- Knowing when to persist vs. when to seek help
- Documenting problems clearly for others
- Self-reliance and confidence in your abilities
- Effective use of limited resources

The Balance

This assignment **doesn't** expect you to:

- Memorize every syntax detail (offline references are allowed)
- Suffer endlessly without help (you have help tokens)
- Reinvent computer science (you've learned the necessary concepts)

This assignment **does** expect you to:

- Exhaust your own debugging strategies before seeking external help
- Develop genuine understanding rather than copy-paste solutions
- Build skills you'll need for your entire career

Employer Perspective

When hiring managers say they want "strong problem-solving skills," this is what they mean:

- Can you work through difficult problems systematically?
- Can you debug your own code effectively?
- Can you make progress when stuck without immediate hand-holding?

- Can you learn new systems by reading documentation and experimenting?

Students who develop these skills are **dramatically more successful** in internships and entry-level positions.

Help Token System

Token Allocation

- Each student receives **4 help tokens** for this assignment
- Each token allows **one instance** of seeking external help
- Tokens are **non-transferable** and **non-refundable**

When to Use a Token

- You've spent **at least 30 minutes** actively debugging the specific issue
- You've tried **at least 3 different approaches** to solve the problem
- You've used available offline resources without success
- You can clearly articulate what you don't understand

When NOT to Use a Token

- Simple syntax errors that the compiler clearly identifies
- Issues you can resolve by reviewing lecture notes or textbook
- Problems you haven't genuinely attempted to debug yourself
- Questions that can be answered by reading the assignment specification

How to Use a Token

Before seeking help:

1. Complete **Part A** of the Help Token Usage Form (all 7 questions)
2. Make a genuine attempt to solve the problem independently
3. Document your debugging attempts with code snippets

Seeking help (choose ONE per token):

- Access the internet for searches and documentation
- Visit TA office hours (must show completed Part A first)
- Email the instructor with your completed form
- Review online course materials or forums

After receiving help:

1. Complete **Part B** of the Help Token Usage Form (questions 8-12)
2. Have your form signed by TA/instructor if help was in-person
3. Return to air-gapped coding environment

Token Management Tips

- Use tokens strategically for **conceptual blocks**, not syntax issues
- Your first two bugs should be solved **without tokens** if possible
- Save at least one token for final testing/edge cases
- If you exhaust all tokens and are still stuck, contact the instructor with **all three completed forms**

Submission Requirements

What to Submit

- Your complete deque implementation (all source files)
- Makefile or build instructions
- A report with the screeshots of a program run.
- **All Help Token Usage Forms** (even if you used 0 tokens)
- Optionally, README documenting your implementation approach

Form Requirements

- If you used **0 tokens**: Submit a single form stating "No tokens used" with brief reflection on your problem-solving process
- If you used **1-3 tokens**: Submit one completed form per token used
- All forms must have **both Part A and Part B** completed
- Forms must be submitted as **PDF or Word** (no handwritten scans unless specified)

Grading Breakdown

- **Correctness (50%)**: Does your deque work according to specifications?
- **Code Quality (20%)**: Clean, readable, well-structured code
- **Problem-Solving Process (20%)**: Quality of help token forms and debugging documentation
- **Testing (10%)**: Comprehensive test cases including edge cases, which will be replaced with expected outputs since main.cpp is provided.

Problem-Solving Process Grading Criteria

- **Excellent (18-20 pts)**: Demonstrates thorough debugging attempts, clear articulation of problems, meaningful learning reflections
- **Good (15-17 pts)**: Shows reasonable debugging efforts, adequate documentation, some insight
- **Adequate (12-14 pts)**: Minimal debugging attempts documented, basic reflections
- **Poor (0-11 pts):** Incomplete forms, insufficient debugging attempts, or token misuse

---

Best Practices

Before You Start

- Download all necessary course materials and references
- Set up your development environment and test that it works offline
- Review relevant lecture notes and textbook chapters
- Create a planning document outlining your implementation approach

During Coding

- **Test incrementally**: Implement and test one function at a time
- **Use print statements**: Liberally add debug output to trace execution
- **Read compiler messages carefully**: They often tell you exactly what's wrong
- **Draw diagrams**: Visualize your circular buffer and index arithmetic
- **Take breaks**: Step away when frustrated; fresh eyes catch bugs

Debugging Strategies (Before Using Tokens)

- Use your IDE's debugger to step through code line-by-line
- Add assert statements to verify assumptions
- Create minimal test cases that isolate the problem
- Review similar examples from lecture or textbook
- Explain your code out loud (rubber duck debugging)
- Check off-by-one errors in loops and index calculations
- Verify boundary conditions (empty, full, single element)

When Stuck

- Walk away for 10-15 minutes
- Re-read the assignment specification
- Review your code from top to bottom
- Check your assumptions about how functions should work

- Ask yourself: "What would I search for if I had internet?"—then check your notes for that topic

---

Academic Integrity

Allowed Collaboration

- General discussions about deque concepts **before** air-gapped sessions
- Sharing debugging strategies (not code solutions)
- Helping each other understand error messages conceptually

Prohibited Actions

- Copying code from any source (peers, internet, previous semesters)
- Sharing your code with other students
- Using token help to get solutions rather than understanding
- Fabricating help token forms or debugging attempts
- Accessing internet without documenting token usage

Consequences

- Violations will result in **zero on the assignment** and referral to academic integrity board
- False documentation on help token forms is considered academic dishonesty

---

Timeline

- **Assignment Release**: 10/21, 11am
- **Recommended Checkpoint1**: 10/23 (Thu) - Basic functionality working
- **Recommended Checkpoint2**: 10/25 (Sat) - All core functionality implemented. If main.cpp is not provided, beginning edge case testing
- **Token Usage Deadline**: 10/26 (Sun) - Last day to use tokens and get help
- **Final Submission**: 10/27

Support Resources

If You've Used All Tokens

- Email instructor with all three completed forms
- Explain what remains unresolved
- Instructor will evaluate whether to grant additional support

Emergency Situations

- Medical or personal emergencies: Contact instructor immediately
- Technical issues (computer failure): Document and notify instructor
- These may warrant token exemptions or extensions

Questions About Rules

- Post clarification questions on [course forum] **before** starting air-gapped work
- Asking about rules/requirements does **not** require a token

---

Success Metrics

You will succeed in this assignment if you:

- Develop a working deque implementation
- Demonstrate genuine problem-solving effort
- Use help tokens strategically when truly stuck
- Learn from your debugging process
- Build confidence in your independent coding abilities

Remember: The goal isn't to never need help—it's to develop the persistence and problem-solving skills that professional developers use every day.

Good luck! You have the skills to do this. Trust your preparation and your problem-solving abilities.

Note: This material was created with the help of ChatGPT and Claude.