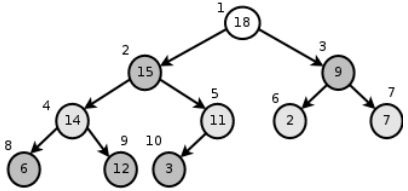
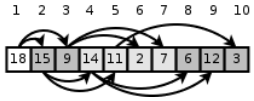
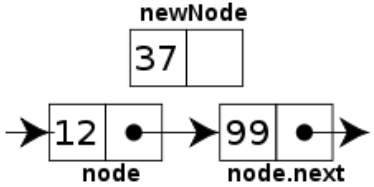
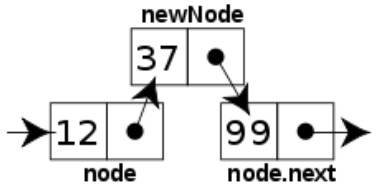




INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] = key$	c_8	$n-1$





WELCOME TO CS 24!

Problem Solving with Computers-II

Instructor: Diba Mirza

Read the syllabus. Know what's required. Know how to get help.

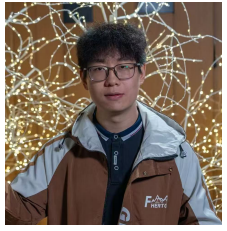
Course website: https://ucsb-cs24.github.io/w26

About the team: we are here to support you. Use us!

- Prof. Mirza's Office hours: W R 1p - 2p, HFH 2119, or by appointment
- Communication with staff via **Ed**
- Sections start this week on Thursday
- Office hours start next week

***Ask questions about class examples, assignment questions, or other CS* topics.**

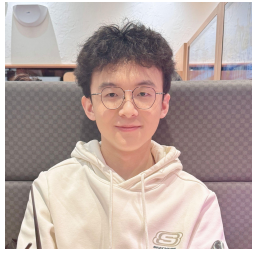


TAs: Sarah Towhidul Yupeng Zhuoer





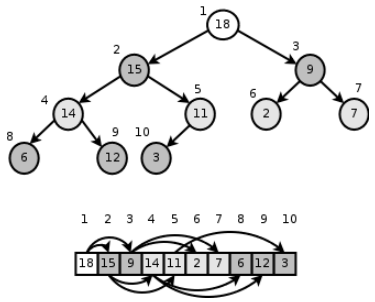


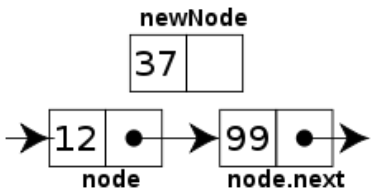

LAs: Nikhil Olivia Ritam Kyle Samuel Jackson

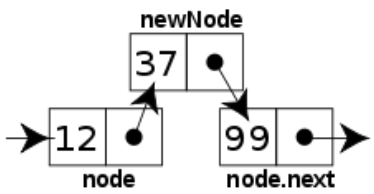
About this course:

Fast coding, clear thinking, no AI shortcuts

- Design and implement **larger programs** that **run fast**
  - Organize **data** in programs using **data structures**
  - **Analyze** the **complexity** of your programs
- Prep for **technical interviews**
- **Today: CS16 Review Abstract Data Types + Linked List**







INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] = key$	c_8	$n-1$

Data Structures and C++

Complexity Analysis

Course Logistics

- Course website: https://ucsb-cs24.github.io/w26
  - schedule, assignments, course setup
- Read the syllabus.
- Today: I'll focus on the *why* behind the course policies

Graded Components

- **Quizzes:** 20% (paper pencil, during lecture time)

- **Leet Code Practice**: 10%
  - 10 medium problems from assigned problem sets (2 from each set)
  - Why LeetCode problems? They mirror interview questions.

- **Mock Interview:** 10%
  - Must schedule by week 2
  - At least one mock interview with an LA/TA by week 10

- **Programming assignments:** 25%
  - includes shorter lab assignments (10%)
  - more complex programming assignments (15%)

- **Final Exam:** 35% (on **03/18**, noon - 3p) in person
  - Final exam score will replace lowest up to two quiz scores, if final is higher

Policy on use of AI

You may use AI tools (ChatGPT, Copilot, etc.) on programming assignments and labs.

However, 65% of your grade comes from paper exams and live coding interviews where AI is not permitted.

Blunt reality: If you use AI to complete assignments without understanding the code, you will likely fail exams and interviews. I cannot verify your AI usage on assignments, but the exams will.

Recommended AI usage:

- ✓ Use AI to explain concepts, debug errors, understand syntax
- ✓ Write pseudocode/algorithm first, implement, then use AI to help refine/improve.
- ✓ Study AI-generated solutions to learn patterns, but write your own code
- ✗ Copy-paste AI solutions without understanding each line
- ✗ Use AI as a substitute for learning to code

For LeetCode-style practice problems: Do these without AI to prepare for exams. Think of these as exam practice.

Bottom line: Use AI as a tutor, not a ghostwriter. Your exam performance will reflect your actual understanding.

Preparing for lectures

- Print the handout for each lecture or download a copy electronically to annotate.

  I'll make these available 24 hours before each lecture.
- **Prep with assigned reading before lectures:** come ready to solve problems.
  - **DS: *Data Structures and Other Objects Using C++* (Savitch, 4th ed.)**
  - **OP: *Open Data Structures* by Pat Morin (Free)**
    - https://opendatastructures.org/ods-cpp/Contents.html
  - **Dasgupta: *Algorithms* by Dasgupta & Vazirani**

About lectures

- Restricted use of electronic devices: okay only for lecture related activities. Otherwise put away your phones!
- Lectures aren't textbook recap, they're problem-solving sessions. Ask questions, discuss with neighbors, work through handouts, answer via iClicker.
- Why interactive? You learn by doing e.g., tracing pointers, mock interview today.
- Take a moment to introduce yourself to the people sitting near you.
    - Talk about…
        - your background,
        - experience in CS so far, and
        - what you hope to get out of this class!

About you…

What is your familiarity/confidence in C++?

A. Know nothing or almost nothing about it.
B. Used it a little, beginner level.
C. Some expertise, lots of gaps though.
D. Lots of expertise, a few gaps.
E. Know too much; I have no life.

- **Why iClicker?** Join at https://join.iclicker.com/ZHLY
  - its practice, not points,
  - to engage with concepts like today's linked list

About you…

What is your familiarity/confidence with using git or any version control system?

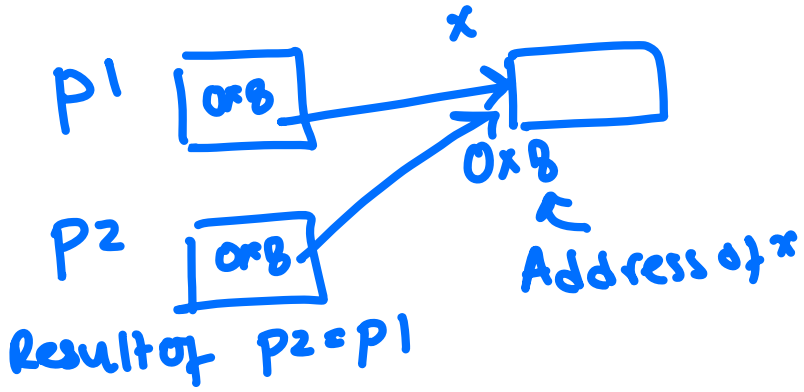
A. Know nothing or almost nothing about it.
B. Used it a little, beginner level.
C. Some expertise, lots of gaps though.
D. Lots of expertise, a few gaps.
E. Know too much; I have no life.

Remember to:

1) accept the invitation sent to your @umail.ucsb.edu account to join the class GitHub Organization (org): **ucsb-cs24-w26**
2) If unfamiliar with git complete optional lab00 by Friday

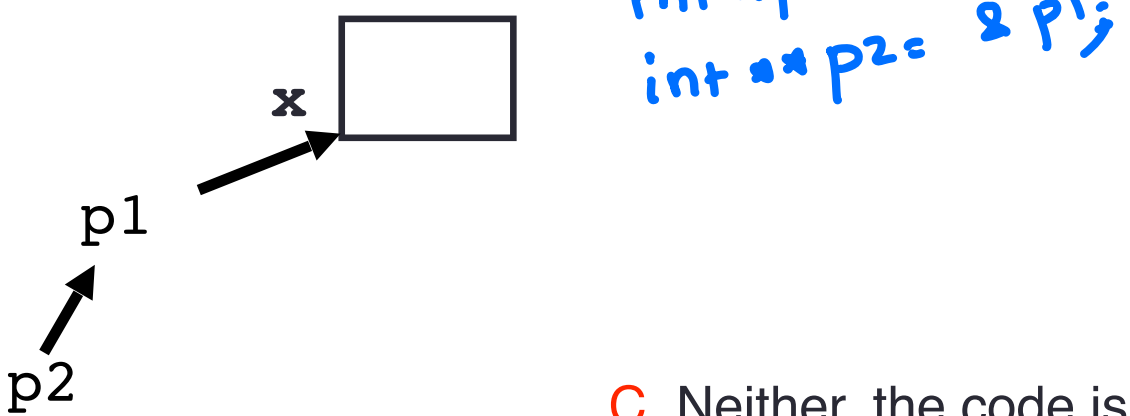
Review: Pointer assignment

```
int* p1, *p2, x;
p1 = &x;
p2 = p1;
```



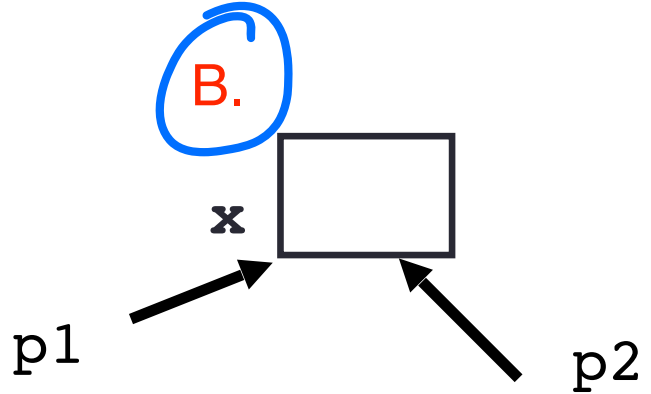
Q: Which of the following pointer diagrams best represents the outcome of the above code?

A.



int x;
int *p1 = &x;
int **p2 = &p1;

B.

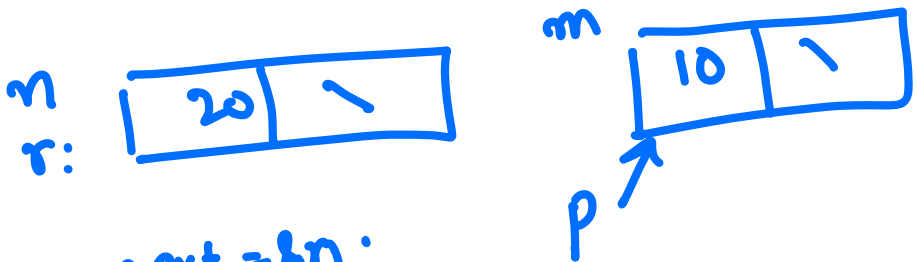


C. Neither, the code is incorrect

Review: Accessing structs using pointers and references

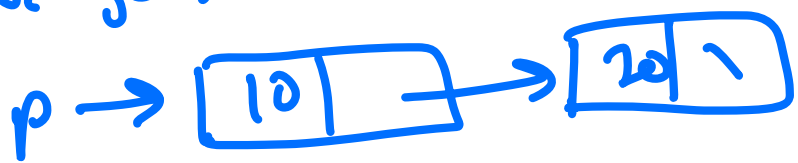
```
Node n {20, nullptr};
Node m {10, nullptr};
Node *p = &m;
Node &r = n;
```

```
struct Node {
      int data;
      Node* next;
};
```



p->next = &n;

We get:

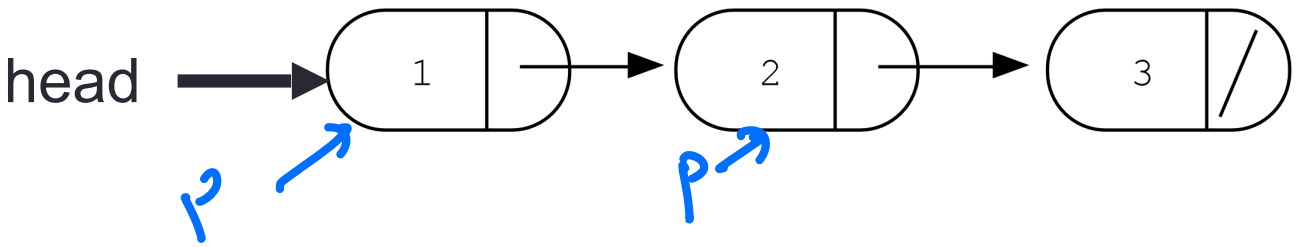


(Linked list)

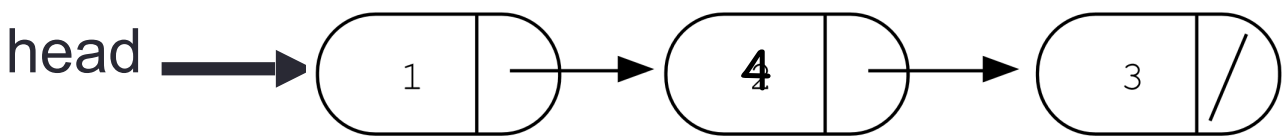
How does the given code modify the provided linked list?

```
Node* p = head;
p = p→next;
p→data = 4;
```

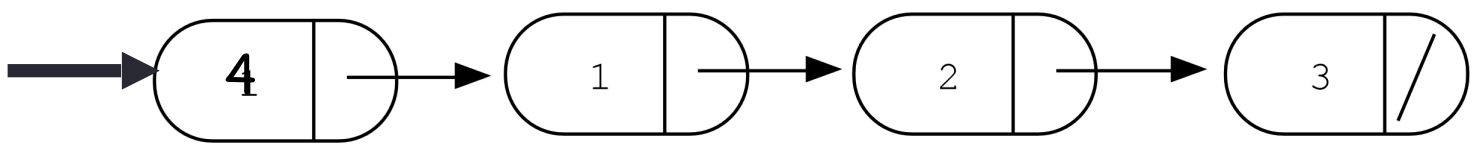
```
struct Node {
    int data;
    Node* next;
};
```



A.



B.



C. Something else

Abstract Data Type (ADT)

- Abstract Data Type (ADT) is defined by data + operations on the data.
- Key features
  - **Abstraction:** hide implementation details
  - **Encapsulation:** bundle data and operations on the data, restrict access to data only through permitted operations

```
class customList {
public:
    customList();
    // other public methods


private:
    struct Node {
        string info;
        Node* next;
    };
    Node* head;
    Node* tail;
};
```

Handout Activity 1:

Critique a flawed customList ADT implementation

Questions to ask about any ADT:

- **What operations does the ADT support?**

  The list ADT supports the following operations on a sequence:

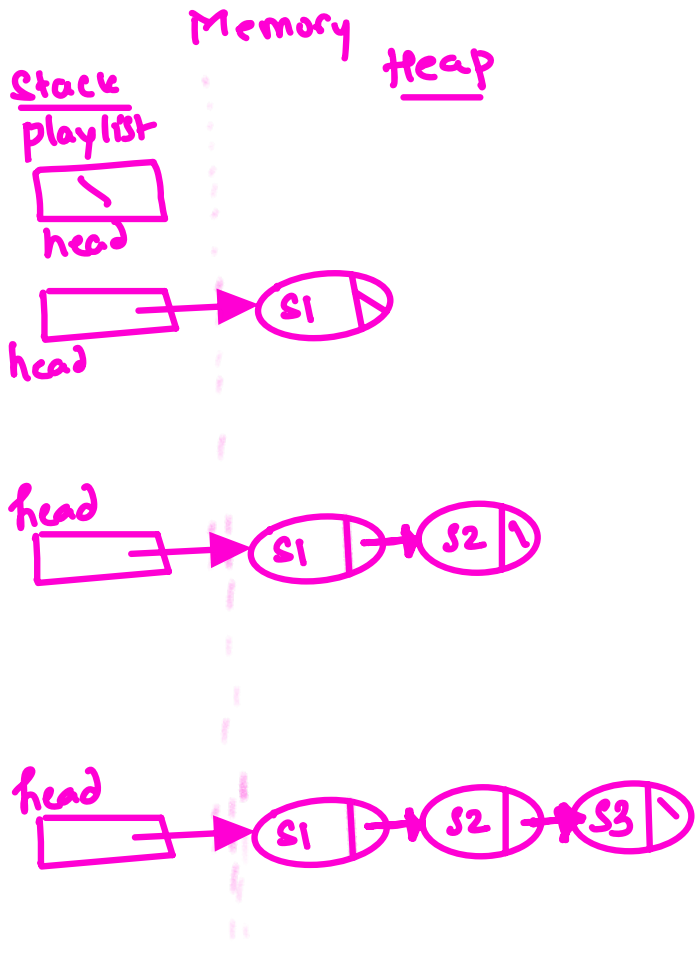
  1. push_front (add a value to the beginning of the sequence)
  2. push_back (add a value to the end of the sequence)
  3. pop_front (delete the first value in the sequence)
  4. pop_back (delete the last value in he sequence)
  5. front() (return the first value)
  6. back() (return the last value)
  7. delete (a value)
  8. print all values
- **How do you implement each operation (data structure used)?**
- **How fast is each operation?**

Handout Activity 2:

Design an improved customList ADT

calls constructor

```
CustomList playlist;
playlist.pushback("S1");
playlist.pushback("S2");
playlist.pushback("S3");
```



Once playlist goes out of function scope the destructor function is called automatically.

The default destructor does nothing (empty function)

So, we need to implement our own to clear all the heap memory. Otherwise the nodes will persist on the heap resulting in a memory leak.

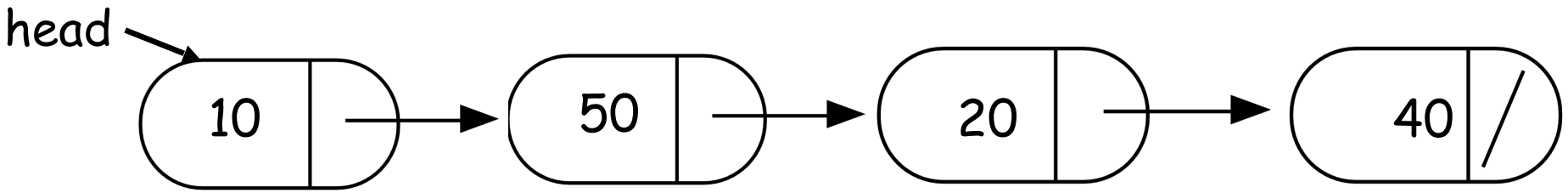
Sol: Implement a clear() function, then reuse it in the destructor.

Helper functions

- Sometimes your functions takes an input that is not easy to recurse on
- In that case define a new function with appropriate parameters: This is your helper function
- Call the helper function to perform the recursion
- Usually the helper function is private

For example

```
void customList::clear(){
   //helper function that performs the recursion.
   clear(head);
   head = nullptr;
}
```




```
Void customList::clear(Node* p){

}
```

See code from lecture to recursively clear the list

Approximate Terminology

- instance = object
- field = instance variable
- method = function
- sending a message to an object = calling a function

Some advice on designing classes

- Always, *always* strive for a narrow interface
- Follow the principle of abstraction and encapsulation:
  - the caller should know as little as possible about how the method does its job
  - the method should know little or nothing about where or why it is being called
  - Your class is responsible for it's own data; don't allow other classes to easily modify it! Make as much as possible private

Next time

- Operator Overloading and Rule of Three
- Be sure to do the required reading listed on the course website