CS010C

Lab2

Announcement

- Assignments
  - “SUM25”
  - “Published”
  - See eLearn
- Submission
  - Pass the test cases
  - Submit a zip file on eLearn(CANVAS)
- The latest versions of demos and slides
  - https://github.com/SJZHZ/UCR_CS010C_25U
  - preview versions on Slack, but they won’t always be the latest

Lab1 More Hints

Due Friday!

Playlist.h

```
private:
    std::string uniqueID;
    std::string songName;
    std::string artistName;
    int songLength;
    PlaylistNode *nextNodePtr;
```

- Private member variables have been completed in template
- Public member functions
  - Mutator
    - modifies something but does not return a value

```
void InsertAfter
void SetNext(Pla
```

  - Accessor
    - retrieves a value and returns it without modifying anything
    - remember to add **const**

```
std::string GetI
std::string GetS
std::string GetA
int GetSongLengt
PlaylistNode *Ge
```

  - Printer
    - No modification & returns nothing(void)
    - Also **const**

```
void PrintPlaylistNode() const;
```

Playlist.cpp

- Constructor
  - Default constructor: construct like $\{\text{"none"}, \text{"none"}, \text{"none"}, 0, nullptr\}$
    - Neither “” nor nullptr!
  - Parameterized constructor
    - Use 4 parameters for first 4 members
    - Just set the last member `nextNodePtr` to `nullptr`

```
std::string uniqueID;
std::string songName;
std::string artistName;
int songLength;
```

- Mutator & Accessor
  - Very simple(~20 lines)
  - Just 1 line inside each function

```
19  > void PlaylistNode::SetNext(Pl
23  > std::string PlaylistNode::Get
27  > std::string PlaylistNode::Get
31  > std::string PlaylistNode::Get
35  > int PlaylistNode::GetSongLeng
39  > PlaylistNode *PlaylistNode::G
```

- Printer
  - Simple(<10 lines)
- InsertAfter
  - modify
    - `node->nextNodePtr`
    - `this->nextNodePtr`

- Insert “anothernode” behind “this” node
- Class member function example
  - someObject.InsertAfter(someObjectElse)
  - [This] is the pointer pointing to “someObject”

Main.cpp

```
#include <iostream>
#include <string>
#include "Playlist.h"
using namespace std;
```

- First several lines
- Necessary Variables in `main()`
  - You may need more, it's OK

```
int main()
{
    std::cout << "Enter playlist's title:\n";
    std::string title;
    PlaylistNode *head = nullptr;
    std::getline(std::cin, title);
```

- Input & Output in `main()`
- Construct a framework in `main()`
  - Quit: return directly. No need for destructor in Lab1
- Implement `void PrintMenu(…)`
- Implement each function with a simple output
- Test your code and debug
  - `g++ main.cpp Playlist.cpp -Wall -o a.out`
  - `./a.out`

```
while (true)
{
    std::cout << "\n";
    PrintMenu(title);
    char choice;
    std::cin >> choice;
    switch (choice)
    {
        case 'a':
            Add_song(head);
            break;
        case 'd':
            Remove_song(head);
            break;
        case 'c':
            Change_position_of_song(head);
            break;
        case 's':
            Output_songs_by_artist(head);
            break;
        case 't':
            Output_total_time(head);
            break;
        case 'o':
            Output_full_playlist(title, head);
            break;
        case 'q':
            return 0;
        default:
            std::cout << "Invalid choice. Pleas
    }
}
return 0;
```

Given we have to use
`using namespace std;`
Then "std::cout" is just "cout"

```
std::cout << title << " PLAYLIS
std::cout << "a - Add song\n";
std::cout << "d - Remove song\n
std::cout << "c - Change positi
std::cout << "s - Output songs
std::cout << "t - Output total
std::cout << "o - Output full p
std::cout << "q - Quit\n\n";
std::cout << "Choose an option:
```

```
void Output_full_playlist(std::string title, PlaylistNode *head)
{
    std::cout << title << " - OUTPUT FULL PLAYLIST\n";
```

I/O Hints

- How to read a word or a number
  - `cin >> length` OR `std::cin >> length;`
- How to read a whole line that may contain spaces
  - `std::getline(std::cin, name);`
  - OR `getline(cin, artist);`
- Using `getline` after `cin`…
  - `getline` only reads a newline character!
  - Solution1: Ignore that character

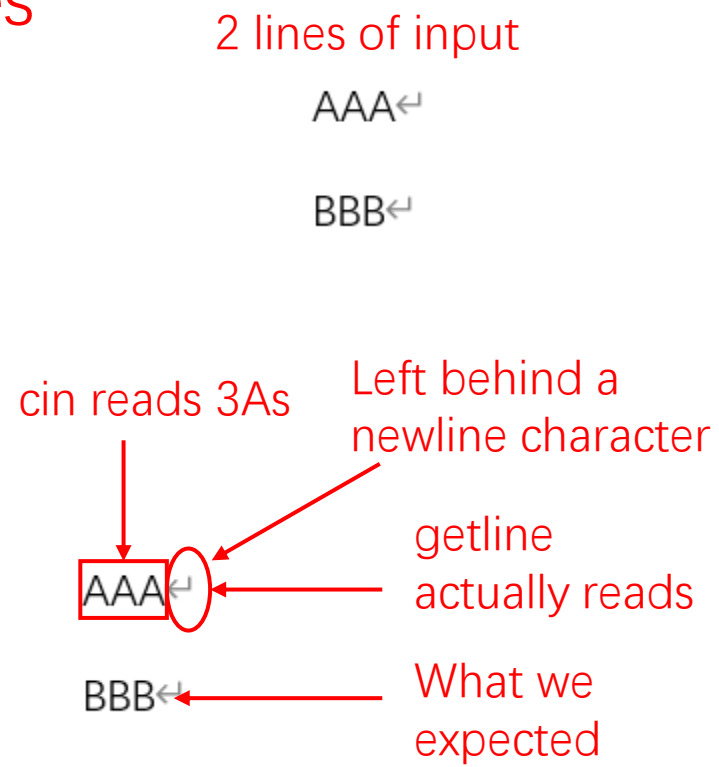
```
std::cin >> something;
std::cin.ignore();
std::getline(std::cin, somethingelse);
```

  - Solution2: Read that whitespace before the remain input

```
std::cin >> something;
std::getline(std::cin >> std::ws, somethingelse);
```

  - Solution3: Or just always use getline

```
std::getline(std::cin, something);
std::getline(std::cin, somethingelse);
```



Add song

- I/O Hints (see last page)
- Append this node to after the tail (of whole list)
  - Solution1: keep tail information (complex)
    - `PlaylistNode* tailNode = 0;`
  - Solution2: search the tail (simple)
    - `while (current->GetNext() != nullptr) do_something;`
- Corner case: related to the head node
  - `head == nullptr`
  - Handle this case separately

Output full playlist

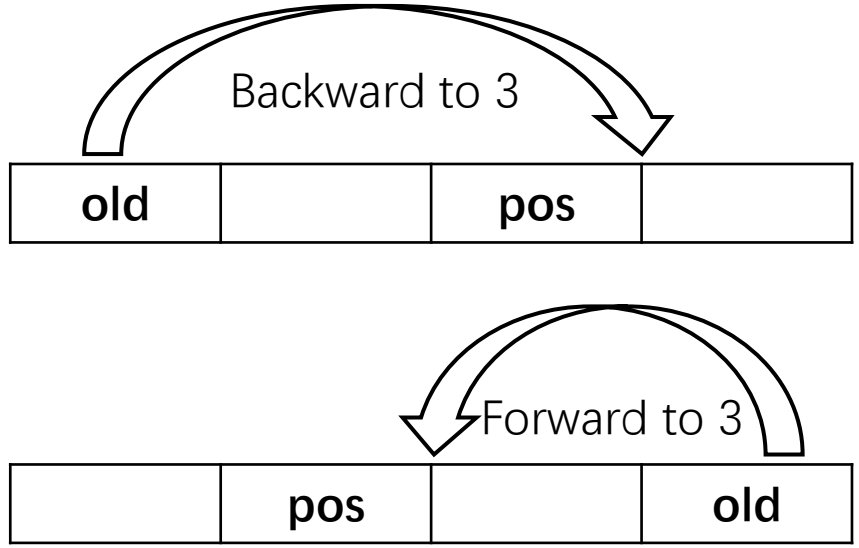
. If the list is empty, output: `Playlist is empty` (3 pts)

- Corner case
  - Empty list
  - How to know? `head == nullptr`
- Record the number
  - `int count = 1;`
  - `std::cout << count++ << ".\n";`
- Linked list traversal

```
PlaylistNode *current = head;
while (current != nullptr)
{
    do_something…
    current = current->GetNext();
}
```

Other functions

- Remove song
  - Corner case
    - Empty list? Just `return;`
    - Song not found? `"Song not found.\n"`
- Output songs by artist: simple
- Output total time: simple
- Change position of song
  - Solution1: Move the node directly (forward $k-1$, backward k)
  - Solution2: Remove the node, then insert after $k-1$
  - Corner cases: related to the head node
    - Moving head node: Store and update the new “head”
    - To the 1st position: Store and update the new “head”

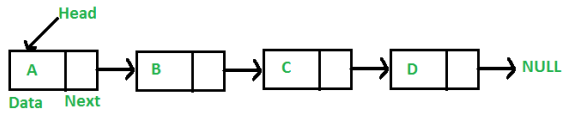


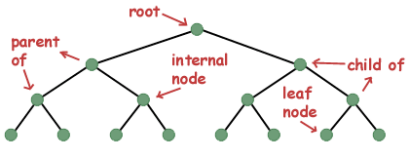
Good luck Lab1

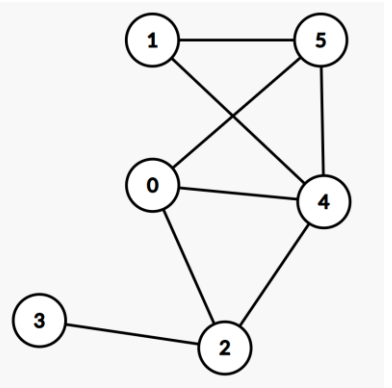
- Lab1 is hard ($\sim 200 + 20 + 50 \approx 300$ lines)
- Due: Friday
- Start early
- My programing habit
  - Construct framework: review demo
  - Step by step
- Ask questions
  - Slack channel @
  - Slack DM
  - Lab time
  - Office hour

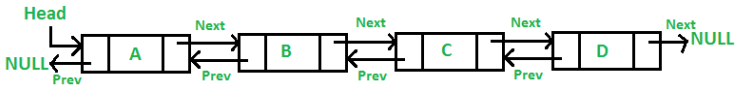
Lab2 & Program1

Node & List









- Node's member
  - must include a pointer(LL) / pointers(2-LL, tree, graph)
  - Other data(can be anything & there can be any number of them)
- Linked List's member
  - must include a head node(LL) / both head and tail nodes(2-LL)
  - Other information(can be anything & there can be any number of them)
- Do we need to implement a node class or structure?
  - Yes.
- Do we need to implement a linked list?
  - It depends.

Lab2 basic

- IntList.h (~30 lines)
- main.cpp: No description?
  - I guess it doesn't matter
  - Just return 0 in `int main()` function is OK (~10 lines)
- IntList.cpp (~80 lines)
  - I/O
    - Only print blank space between each integer
  - Corner cases: doubly-linked list is "empty"
    - Will occur in `IntList::pop_front()` and `IntList::pop_back()`
    - When `IntList::empty()`

Lab2 manually graded part

- Rule of Three: If implement 1, implement the other 2 as well!
  - Destructor
    - `~IntList();`
  - Copy constructor
    - `IntList(const IntList &other);`
  - Copy assignment operator
    - `IntList& operator=(const IntList &other);`
- To get full credit, you should include the other two as well
  - Declaration
  - Definition
    - Implementation

Program1

- Josephus Problem: Explanation
  - Initial circle n
    - only count first n people in this problem(rather than this “round”)
  - Stride k
    - Begin at next one: count k-th person = skip $k-1$ person
    - Begin at themselves: count $k+1$-th person = skip k person
- Solution1: Doubly-linked list -> circular-linked list
  - Resize the circle to n
  - Remove the nodes one-by-one

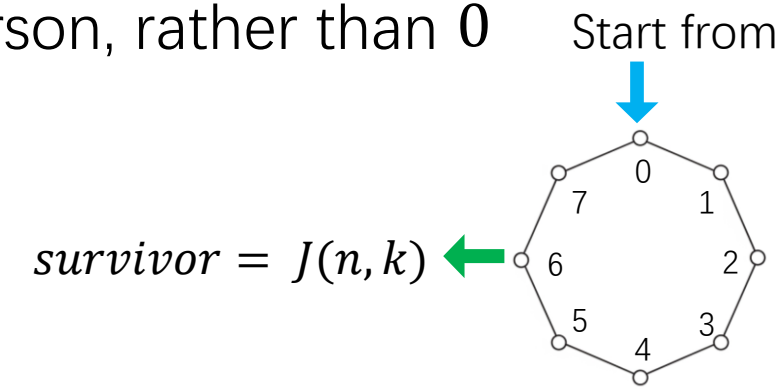
Not helpful for this class
Feel free to think about it if interested

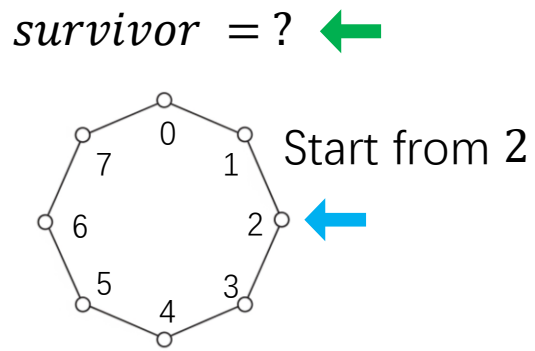
- The assignment must be submitted using the linked list(Solution1) method

- ~~Wait, is there any mathematical solution?~~

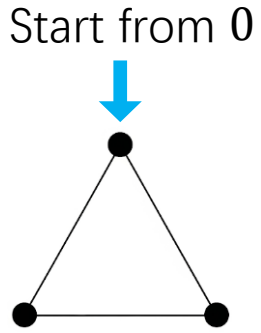
Not helpful for this class
Feel free to think about it if interested

- Definition: $J(n,k)$ represents the index of the survivor
  - n: number of people, k: number k-th person to vote out(starting from themselves)
  - $J(n,k) \in [0, n-1]$ (numbering starts from 0, and the last person is $n-1$)
- Puzzle0: Corner cases
  - Case: $k = 0$ => $J(n,0) = n-1$ (the last person survives)
  - Case: $n = 1$ => $J(1,k) = 0$ (game ends)
- Puzzle1: Rotation
  - Start from b-th person, rather than 0
  - It's a circle!
  - $survivor = ?$



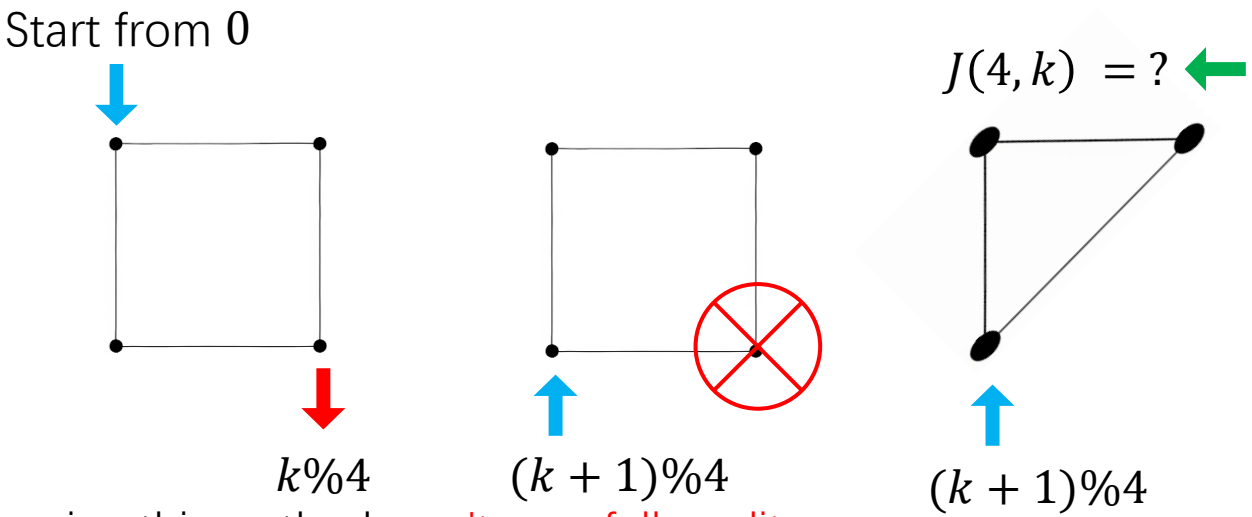


Not helpful for this class
Feel free to think about it if interested



- Puzzle2: Look in reverse
  - Given k ,
    - $J(1,k) = 0 \Rightarrow J(2,k) = ?$
    - $J(2,k) = 0 \Rightarrow J(3,k) = ?$

- Try to retrieve the person
  - instead of voting them out
- $J(i,k) = 0 \Rightarrow J(i+1,k) = ?$



- Solution2: Computing $J(n,k)$
  - Since we're learning coding rather than math, using this method won't earn full credit
  - More efficient (only arithmetic is involved — no linked list access)
  - The computation process can be simplified to just a few lines!
  - Totally 30 lines

```
28      std::cout << names[survivor] << " wins!";
29      return 0;
30  }
```

```
5   int
6   {
7
8
9
10
11  }
```