Linked Lists and ADT

CS19001: Programming and Data Structures Laboratory

05-Nov-2018

Soumyajit Dey and Aritra Hazra

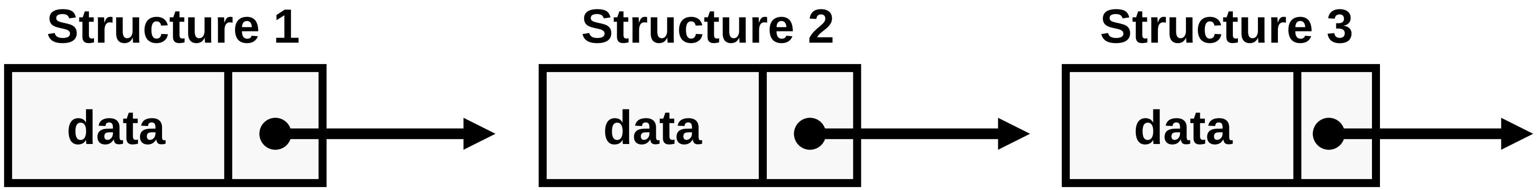
Department of Computer Science & Engineering,
Indian Institute of Technology Kharagpur,
Paschim Medinipur, West Bengal, India – 721302.

http://cse.iitkgp.ac.in/~aritrah/course/lab/PDS/Autumn2018/CS19101_PDS-Lab_Autumn2018.html

Tutorial: Linked Lists

Lists

- A list refers to a sequence of data items
  - Example: An array
    - Array index is used for accessing and manipulating array elements
  - Problems with arrays
    - Array size specified at the beginning (at least during dynamic allocation)
      - realloc can be used to readjust size in middle, but contiguous chunk of memory may not be available
    - Deleting / Inserting an element may require shifting of elements
    - Wasteful of space
- A completely different way to represent a list (Linked List)
  - Make each data in the list part of a self-referential structure
  - The structure also contains a pointer or link to the structure (of the same type) containing the next data

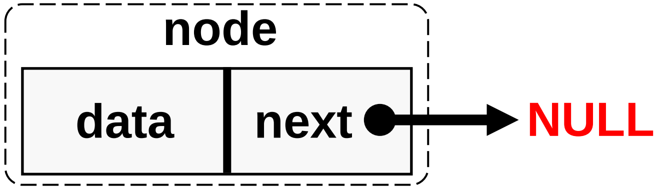


Single Linked Lists

- Let each structure of the list (lets call it node) have two fields:
  - One containing the data
  - Other containing address of the structure holding next data in the list
- The structures in the linked list need not be contiguous in memory
  - Ordered by logical links stored as part of data in the structure itself
  - The link is a pointer to another structure of the same type
- The pointer variable next contains either the address of the location in memory of the successor list element or the special value NULL
  - NULL is used to denote the end of the list (no successor element)

Definition of a Node:

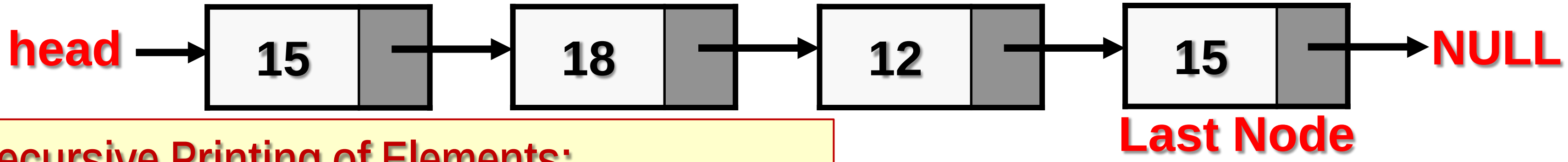
```
typedef struct node {
  int data;
  struct node *next;
} llNode;
llNode *head, *prev, *cur;
```



Creation of a Node:

```
llNode *createNode(int item)
{
  llNode *new = (llNode *)malloc(sizeof(llNode));
  if(new ==NULL) printf(“Malloc Error!”);
  else {
    new->data = item;  new->next = NULL;
  }
  return (new);
}
```

Traversal of Linked Lists



Recursive Printing of Elements:

```
void recPrintLL(llNode *head) {
  if(head != NULL){
    printf(“%d, ”, head->data);
    recPrintLL(head->next);
    printf(“%d, ”, head->data);
  }
}
```

Forward Printing: 15, 18, 12, 15,

Backward Printing: 15, 12, 18, 15,

Pointing Last Node in Single Linked List:

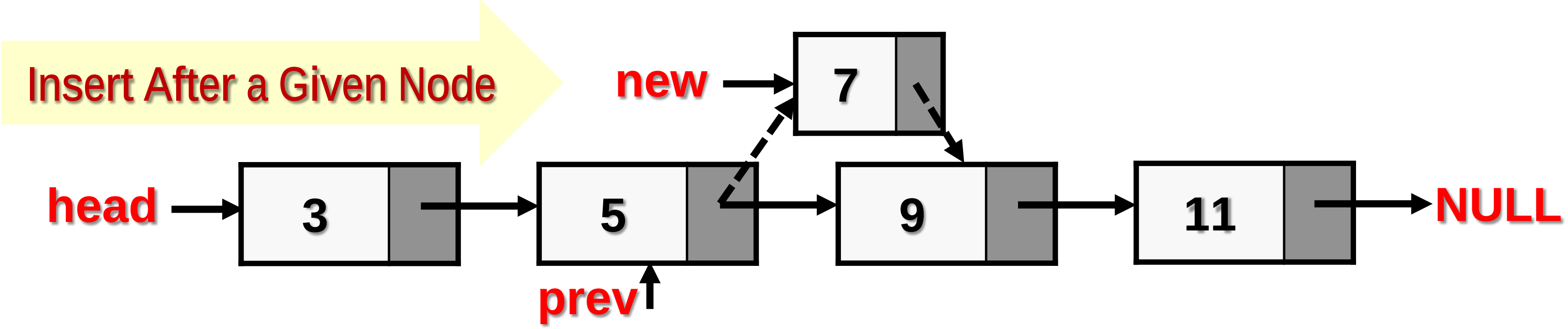
```
llNode *lastNodeLL(llNode *head)
{
  llNode *cur = head;
  if(cur != NULL) // no node
    while(cur->next != NULL)
      cur = cur->next;
  return (cur);
}
```

Finding an Element in Single Linked List:

```
llNode *searchLL
       (llNode *head, int elm)
{
  llNode *cur = head;
  while(cur != NULL) {
    if(cur->data == elm)
      break;
    cur = cur->next;
  }
  return (cur);
}
```

Need to Traverse all
N elements in list

Insertion into Linked Lists

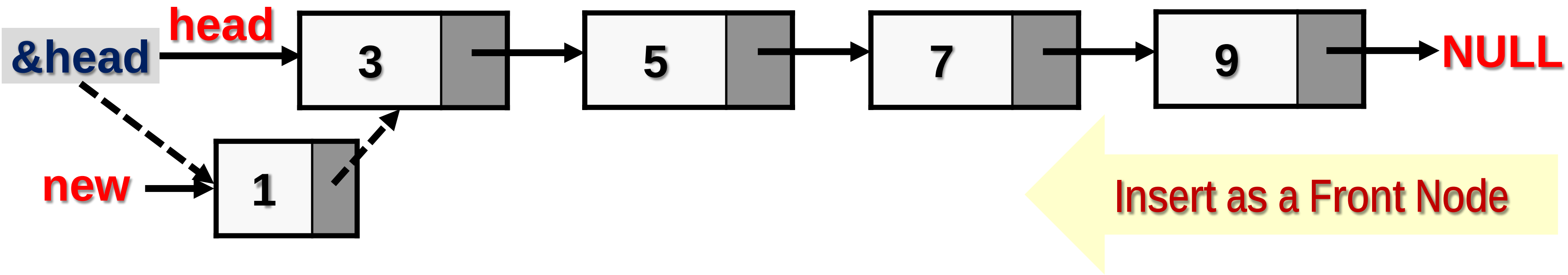


Insert a **new** Node after **prev** Node:

```
void insertAfterLL
   (llNode *prev, llNode *new)
{
   new->next = prev->next;
   prev->next = new;
}
```

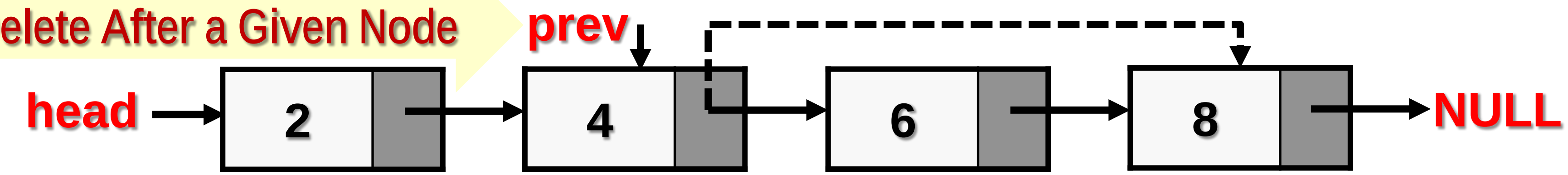
Insert a **new** Node in Front:

```
void *insertFrontLL
   (llNode **phead, llNode *new)
{
   new->next = (*phead);
   (*phead) = new;
}
```



Deletion from Linked Lists

Delete After a Given Node



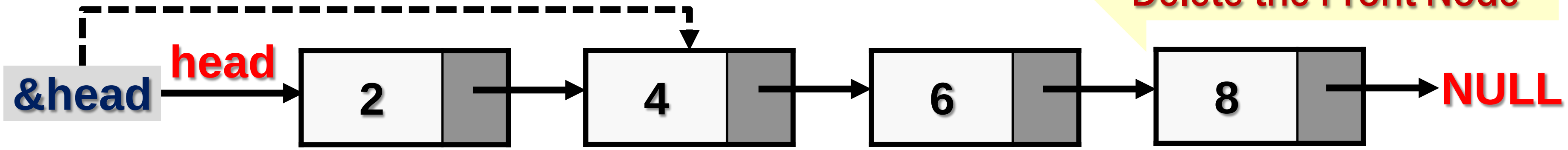
Delete a Node after **prev** Node:

```
void deleteAfterLL(llNode *prev)
{
  if(prev->next != NULL)
    prev->next =
      (prev->next)->next;
}
```

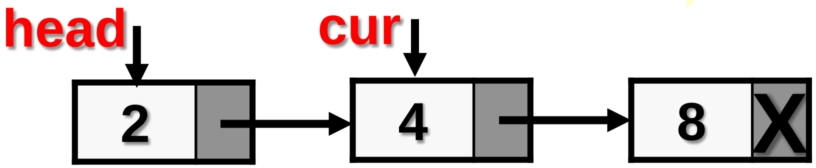
Delete the Front Node:

```
void *deleteFrontLL(llNode **phead)
{
  if((*phead) != NULL)
    (*phead) = (*phead)->next;
}
```

Delete the Front Node



Delete Random (Current) Node

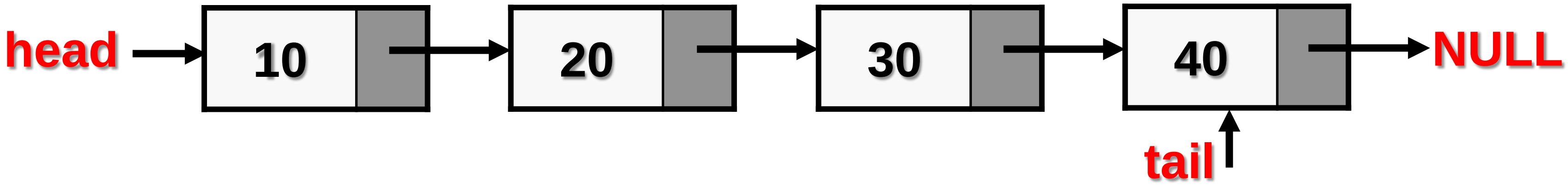


```
llNode *deleteCurLL(llNode **phead)
{
  cur->data = (cur->next)->data;
  deleteAfterLL(cur);
}
```

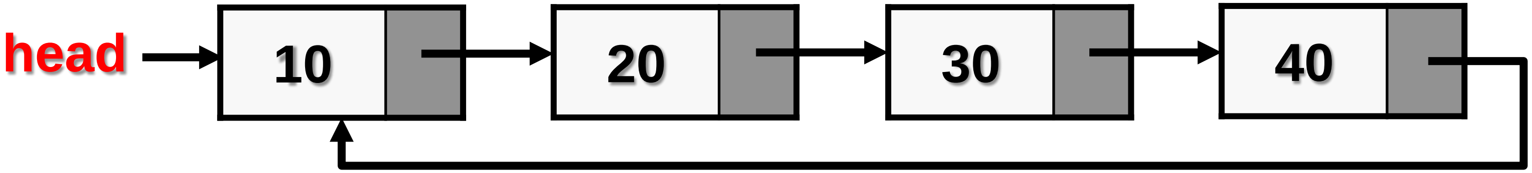
No need to traverse entire list
(except **cur** being Last Node!)

Variations of Linked Lists

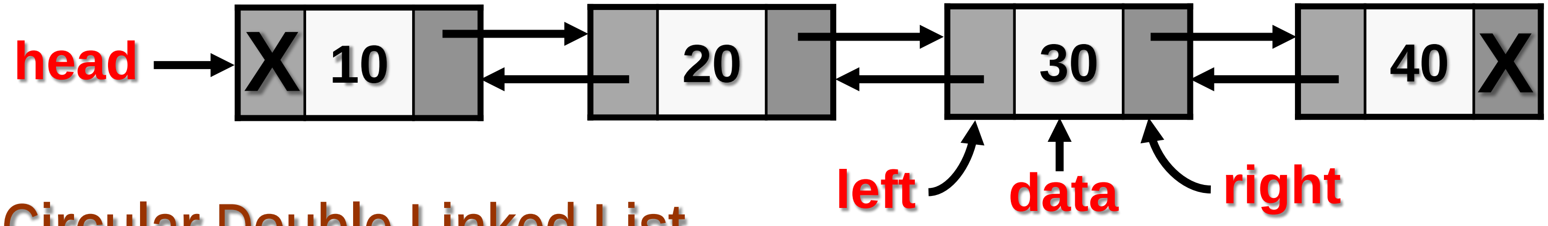
- ❑ Single Linked List with Head and Tail Pointers



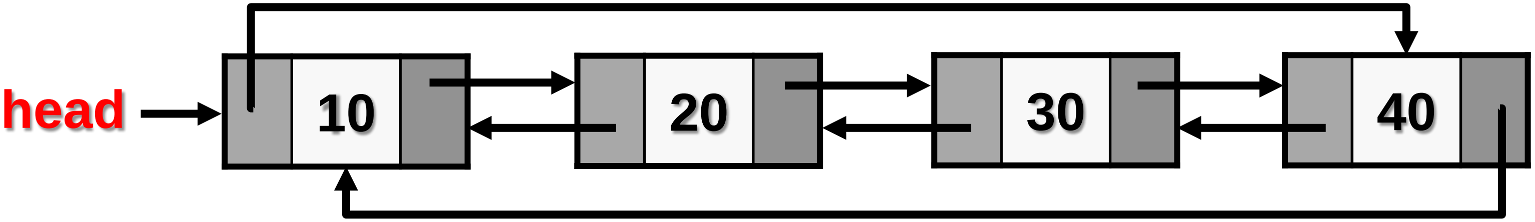
- ❑ Circular Linked List



- ❑ Double Linked List



- ❑ Circular Double Linked List

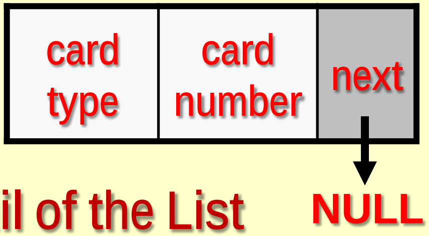


Programming Assignments

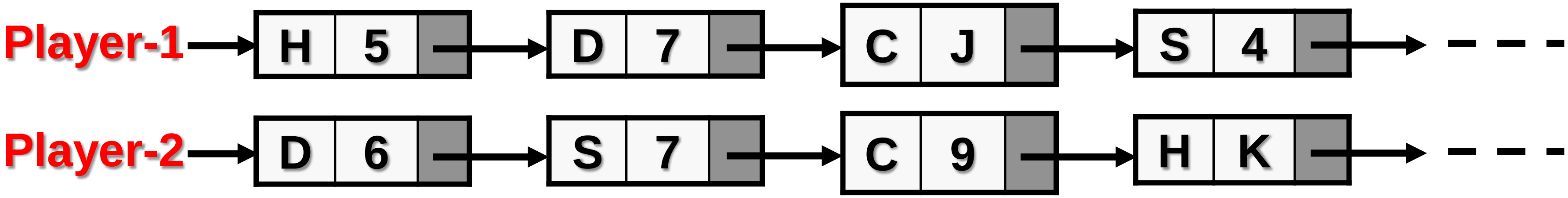
Complete and submit during lab

Assignment 1 [Card-Matching Game]

- ❑ Suit / Card Nomenclature (Suit = 4 Decks of 13 Cards Each)
  - ■ Card Numbers (Ascending Order of Value): 2–10, J, Q, K, A
    2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < J (Jack) < Q (Queen) < K (King) < A (Ace)
  - ■ Deck Types: Spades (♠/S), Hearts (♥/H), Diamonds (♦/D) and Clubs (♣/C)
- ❑ Write a C-program to simulate the following 2-player game-of-cards:
  - ■ Task-1: Write few basic Linked List functions/code
    - ● Define a node in the list as shown in the right
    - ● Write a function to Create a New Node
    - ● Write a function to Insert a New Node at the End/Tail of the List
    - ● Write a function to Delete the Head/Front Node from the List



  - ■ Task-2: Write a Function to –
    - ● Distribute total 52 cards *randomly* into two hands of 26 unique cards each
    - ● Keep these cards in two separate linked list structure for both players



Assignment 1 [Card-Matching Game]

- Task-3: Write C main-function for the game (goes in rounds) as follows:

Task-3a: Write a C-function to draw (in each round) two cards (by deleting two players' head/front nodes of list).

- Initially, Player-1 will start. From next round, the first turn will be from the player who wins the last round.
- In each round, both player takes out their first card (following the order of turns) from the beginning of their respective hands and put this pair one over the other.

Task-3b: Write a C-function to match two drawn cards (i.e. two nodes) and determine the larger or equal card number.

- If the numbers printed on the two drawn cards are SAME, the round continues, and next cards are taken out from both players (turn-wise) … so on …
- A player WINS a round when the number printed on his/her card is MORE than his/her opponent player in the last pair of cards drawn. In such case, all the standing cards (of that given round) are acquired by the winning player and (s)he puts the cards (in same order as drawn) at the end of his/her hand of cards.
- … Similarly, the next round repeats the same play … on and on …

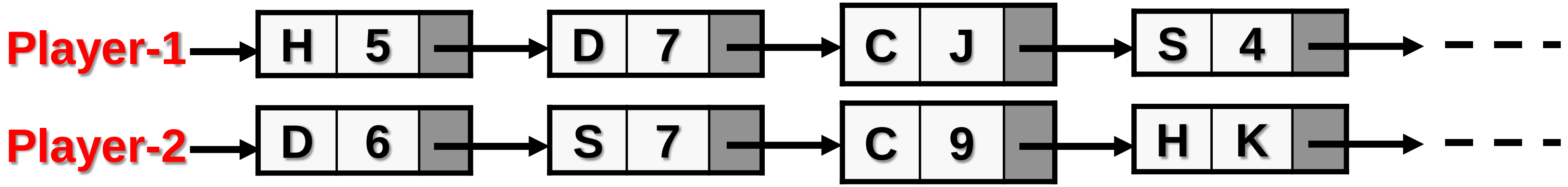
- Finally, if a player can acquire all 52 cards, then (s)he is declared as the WINNER! If both of the players hand finishes together[#], then the match is declared as TIED!

Task-3c: Write a C-function to build a list (in each round) of drawn cards turn-wise and append into the end of the player list (in order of their play-turns) whoever wins that round.

[#] It may happen rarely – only when the sequence of cards follow same numbering in both hands

Assignment 1 : *Simulation for Game-of-Cards*

❑ Initial Hands:

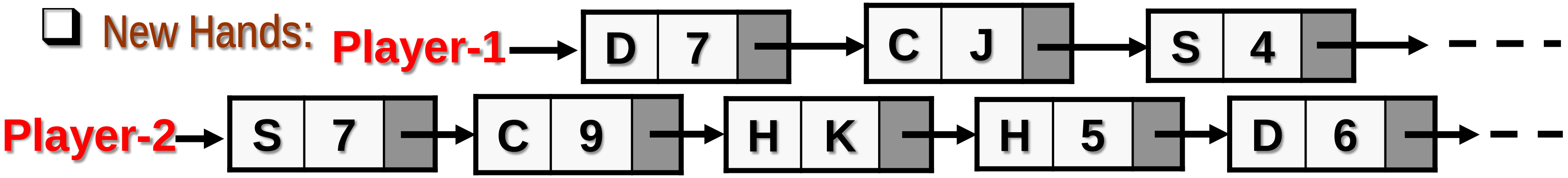


Player-1 → H 5 → D 7 → C J → S 4 → - - -

Player-2 → D 6 → S 7 → C 9 → H K → - - -

❑ Play #1 (Turn of Player-1 first): H 5 vs D 6 → Player-2 takes ...

❑ New Hands:



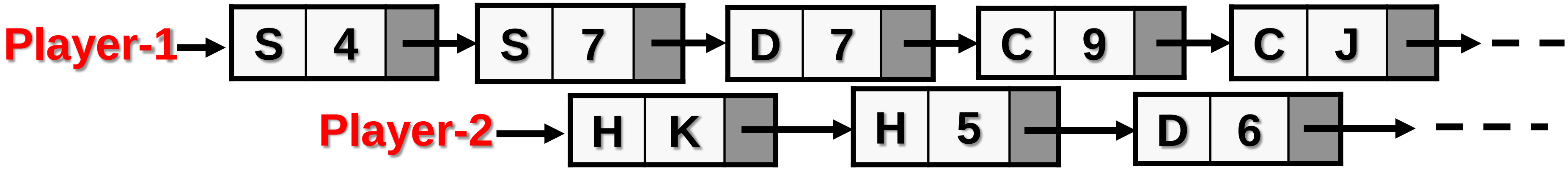
Player-1 → D 7 → C J → S 4 → - - -

Player-2 → S 7 → C 9 → H K → H 5 → D 6 → - -

❑ Play #2 (Turn of Player-2 first): D 7 vs S 7 → *Same*! Next Turn ...

❑ Play #3 (Turn of Player-2 first): C J vs C 9 → Player-1 takes ...

❑ New Hands:



Player-1 → S 4 → S 7 → D 7 → C 9 → C J → - -

Player-2 → H K → H 5 → D 6 → - - -

Assignment 1: *Solution Aids*

- Random Number Generation

```
#include <stdlib.h>        // for srand(), rand()
#include <sys/types.h>     // for getpid()
#include <unistd.h>        // for getpid()
int main()
{
   // declare srand() in the beginning of main (once)
   srand( getpid() );

   // generate any random number [0-9]
   x = rand() % 10;
   return 0;
}
```

Thank You!