

Lecture No.04

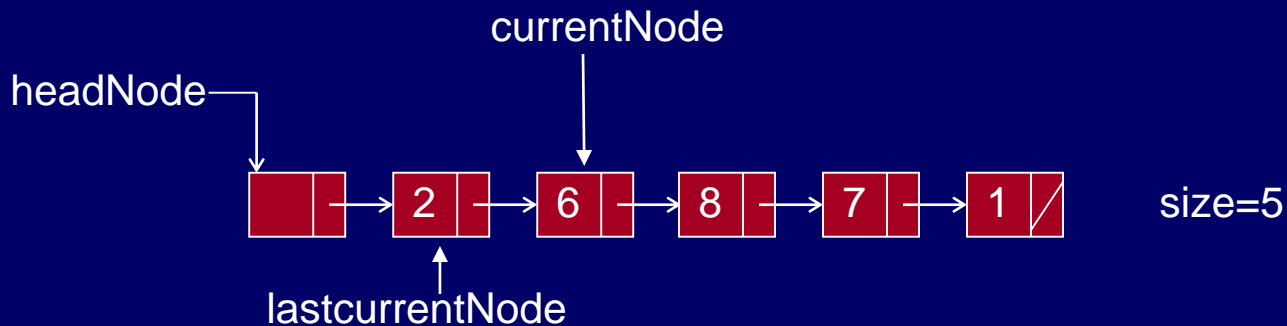
Data Structures

C++ Code for Linked List

```
// position current before the first
// list element
void start() {
    lastCurrentNode = headNode;
    currentNode = headNode;
};
```

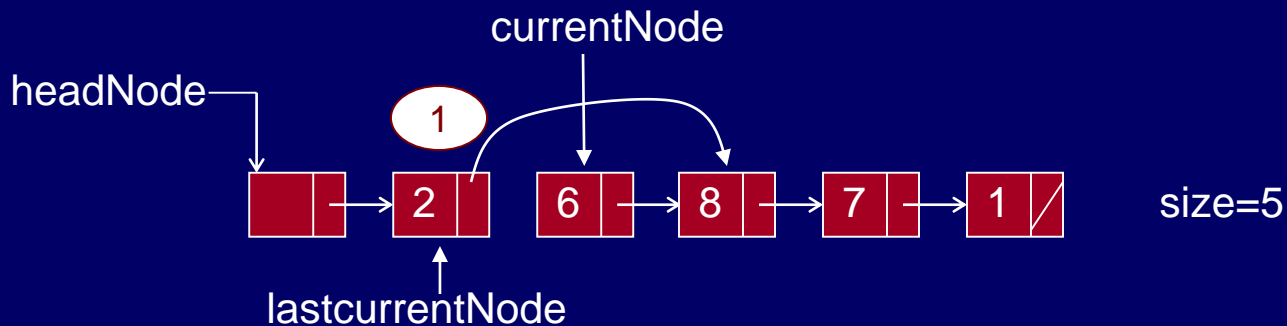
C++ Code for Linked List

```
void remove() {  
    if( currentNode != NULL &&  
        currentNode != headNode) {  
        lastCurrentNode->setNext(currentNode->getNext());  
        delete currentNode;  
        currentNode = lastCurrentNode->getNext();  
        size--;  
    }  
};
```



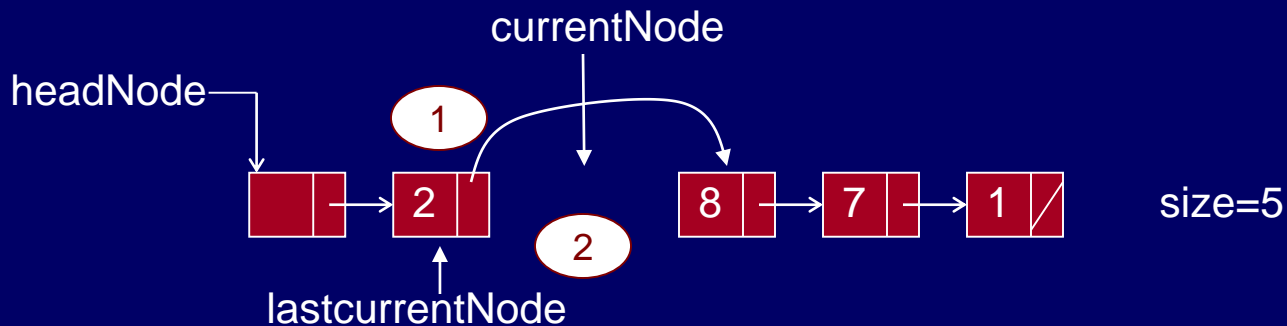
C++ Code for Linked List

```
void remove() {  
    if( currentNode != NULL &&  
        currentNode != headNode) {  
        1 lastCurrentNode->setNext(currentNode->getNext());  
        delete currentNode;  
        currentNode = lastCurrentNode->getNext();  
        size--;  
    }  
};
```



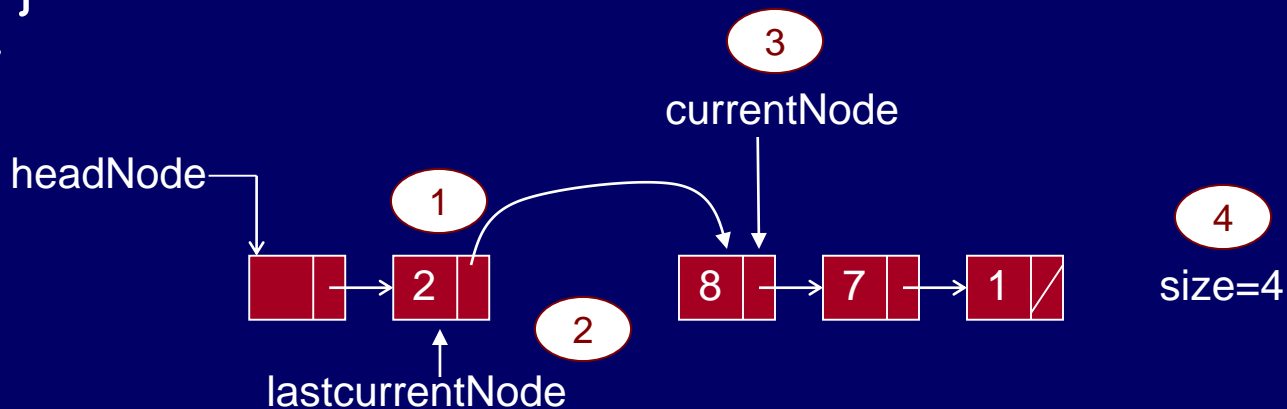
C++ Code for Linked List

```
void remove() {  
    if( currentNode != NULL &&  
        currentNode != headNode) {  
        1 lastCurrentNode->setNext(currentNode->getNext());  
        2 delete currentNode;  
        currentNode = lastCurrentNode->getNext();  
        size--;  
    }  
};
```



C++ Code for Linked List

```
void remove() {  
    if( currentNode != NULL &&  
        currentNode != headNode) {  
        1 lastCurrentNode->setNext(currentNode->getNext());  
        2 delete currentNode;  
        3 currentNode = lastCurrentNode->getNext();  
        4 size--;  
    }  
};
```



C++ Code for Linked List

```
int length()  
{  
    return size;  
};
```

```
private:  
    int size;  
    Node *headNode;  
    Node *currentNode, *lastCurrentNode;
```

Lecture No.04

Data Structures

Dr. Sohail Aslam

Example of List Usage

```
#include <iostream>
#include <stdlib.h>
#include "List.cpp"

int main(int argc, char *argv[])
{
    List list;

    list.add(5); list.add(13); list.add(4);
    list.add(8); list.add(24); list.add(48);
    list.add(12);
    list.start();
    while (list.next())
        cout << "List Element: " << list.get() << endl;
}
```

Analysis of Linked List

- add
 - we simply insert the new node after the current node. So add is a one-step operation.

Analysis of Linked List

- add
 - we simply insert the new node after the current node. So add is a one-step operation.
- remove
 - remove is also a one-step operation

Analysis of Linked List

- add
 - we simply insert the new node after the current node. So add is a one-step operation.
- remove
 - remove is also a one-step operation
- find
 - worst-case: may have to search the entire list

Analysis of Linked List

- add
 - we simply insert the new node after the current node. So add is a one-step operation.
- remove
 - remove is also a one-step operation
- find
 - worst-case: may have to search the entire list
- back
 - moving the current pointer back one node requires traversing the list from the start until the node whose next pointer points to current node.

Doubly-linked List

- Moving forward in a singly-linked list is easy; moving backwards is not so easy.

Doubly-linked List

- Moving forward in a singly-linked list is easy; moving backwards is not so easy.
- To move back one node, we have to start at the head of the singly-linked list and move forward until the node before the current.

Doubly-linked List

- Moving forward in a singly-linked list is easy; moving backwards is not so easy.
- To move back one node, we have to start at the head of the singly-linked list and move forward until the node before the current.
- To avoid this we can use *two* pointers in a node: one to point to next node and another to point to the previous node:

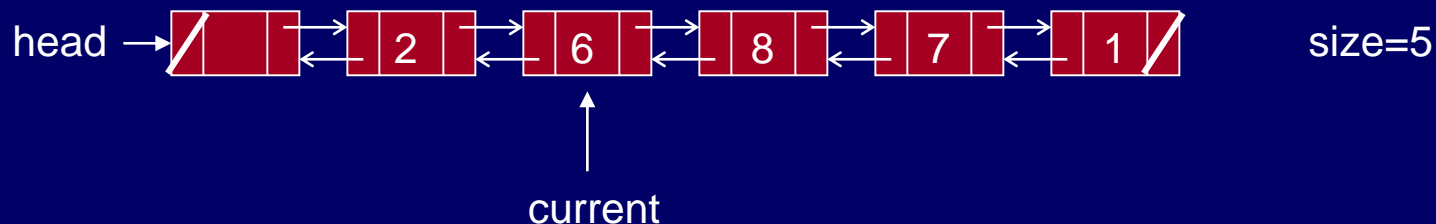


Doubly-Linked List Node

```
class Node {  
public:  
    int get() { return object; };  
    void set(int object) { this->object = object; };  
  
    Node* getNext() { return nextNode; };  
    void setNext(Node* nextNode)  
        { this->nextNode = nextNode; };  
    Node* getPrev() { return prevNode; };  
    void setPrev(Node* prevNode)  
        { this->prevNode = prevNode; };  
private:  
    int object;  
    Node* nextNode;  
    Node* prevNode;  
};
```

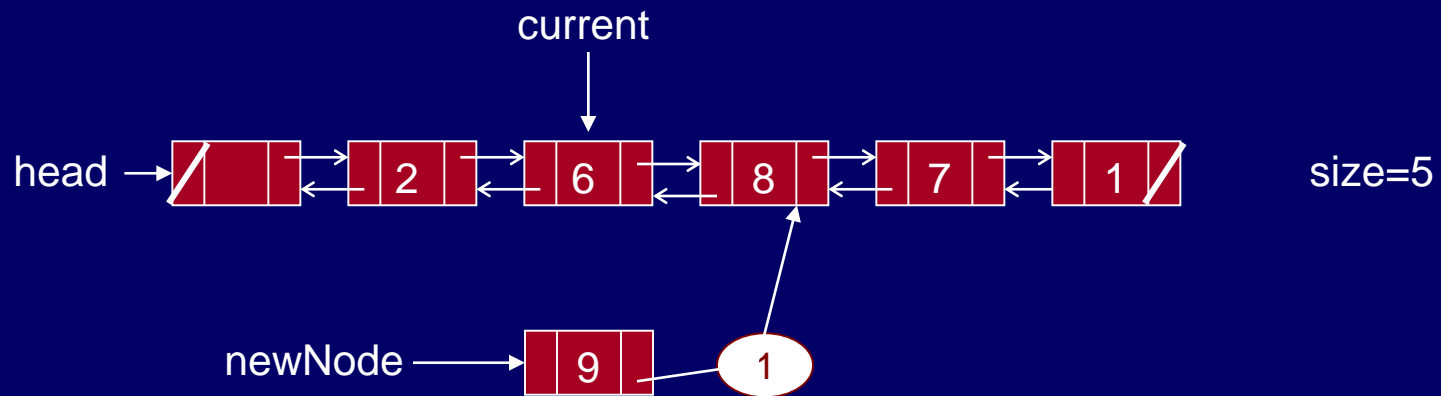
Doubly-linked List

- Need to be more careful when adding or removing a node.
- Consider add: the order in which pointers are reorganized is important:



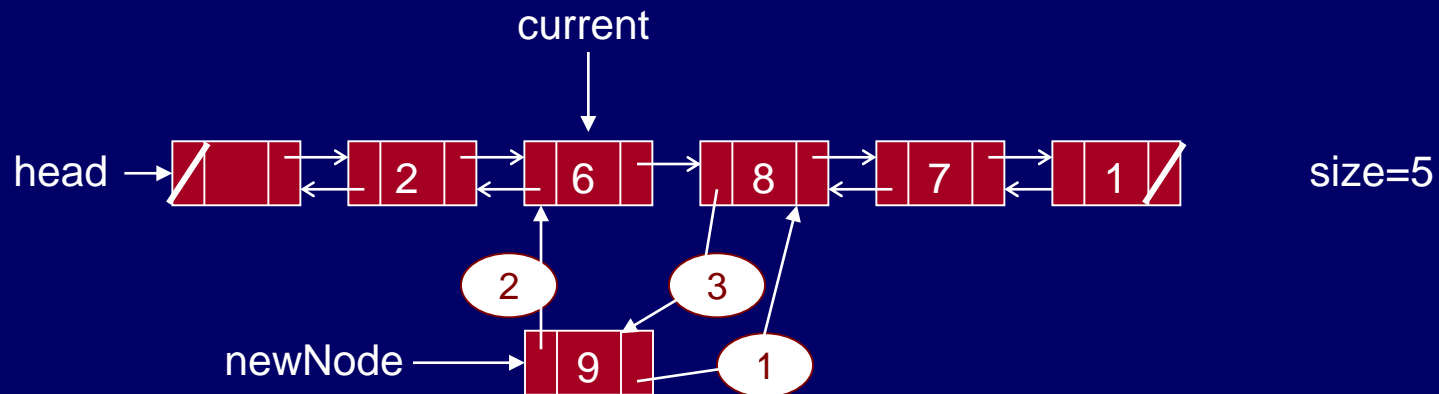
Doubly-linked List

1. `newNode->setNext(current->getNext());`



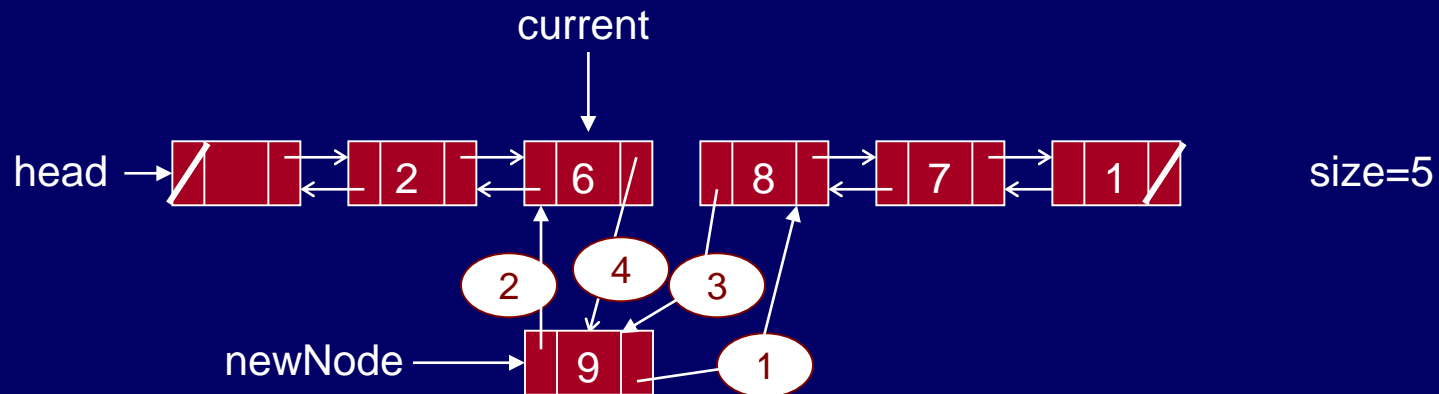
Doubly-linked List

1. `newNode->setNext(current->getNext());`
2. `newNode->setprev(current);`
3. `(current->getNext())->setPrev(newNode);`



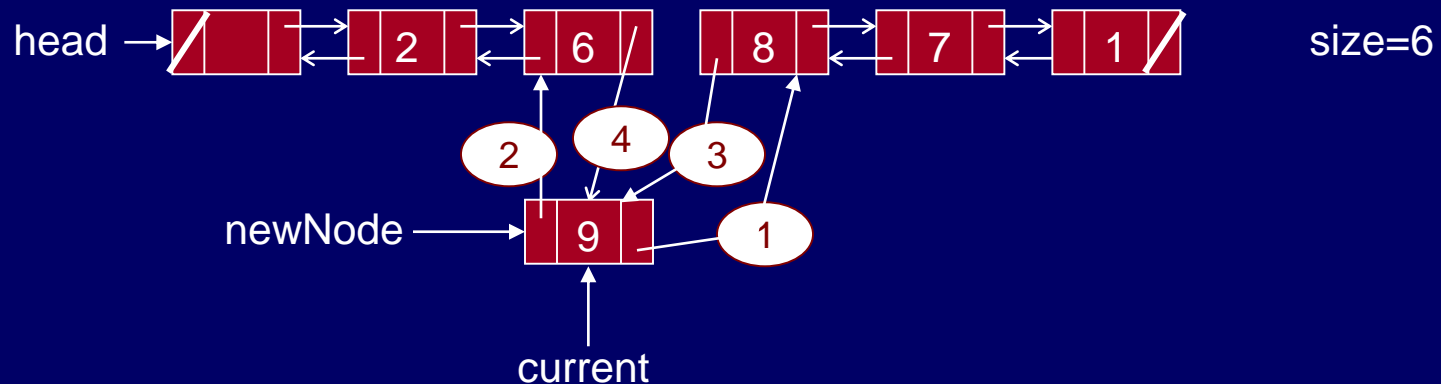
Doubly-linked List

1. `newNode->setNext(current->getNext());`
2. `newNode->setprev(current);`
3. `(current->getNext())->setPrev(newNode);`
4. `current->setNext(newNode);`



Doubly-linked List

1. `newNode->setNext(current->getNext());`
2. `newNode->setprev(current);`
3. `(current->getNext())->setPrev(newNode);`
4. `current->setNext(newNode);`
5. `current = newNode;`
6. `size++;`



Circularly-linked lists

- The next field in the last node in a singly-linked list is set to NULL.
- Moving along a singly-linked list has to be done in a watchful manner.
- Doubly-linked lists have two NULL pointers: prev in the first node and next in the last node.
- A way around this potential hazard is to link the last node with the first node in the list to create a *circularly-linked list*.

Circularly-linked lists

- The next field in the last node in a singly-linked list is set to NULL.
- Moving along a singly-linked list has to be done in a watchful manner.
- Doubly-linked lists have two NULL pointers: prev in the first node and next in the last node.
- A way around this potential hazard is to link the last node with the first node in the list to create a *circularly-linked list*.

Circularly-linked lists

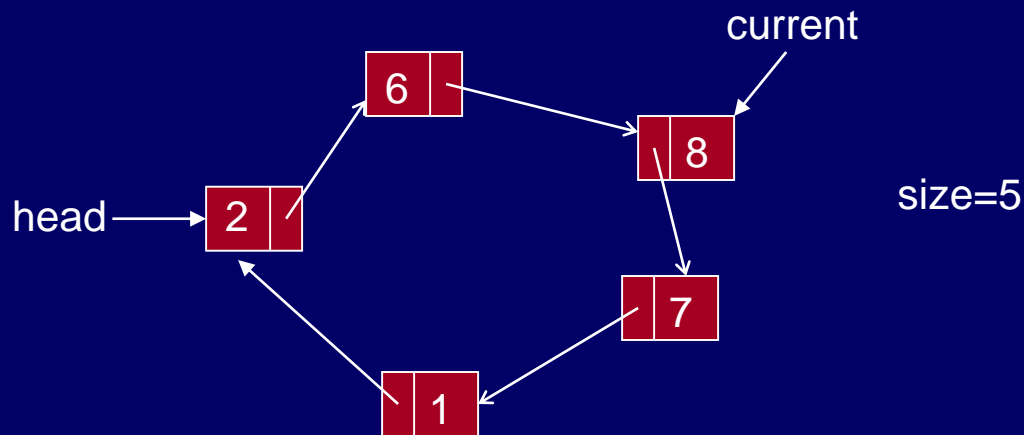
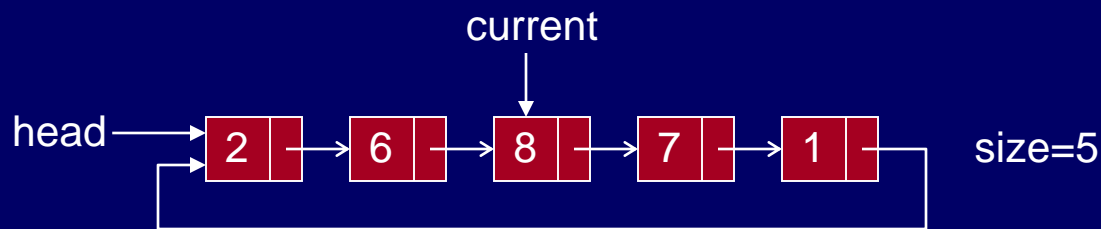
- The next field in the last node in a singly-linked list is set to NULL.
- Moving along a singly-linked list has to be done in a watchful manner.
- Doubly-linked lists have two NULL pointers: prev in the first node and next in the last node.
- A way around this potential hazard is to link the last node with the first node in the list to create a *circularly-linked list*.

Circularly-linked lists

- The next field in the last node in a singly-linked list is set to NULL.
- Moving along a singly-linked list has to be done in a watchful manner.
- Doubly-linked lists have two NULL pointers: prev in the first node and next in the last node.
- A way around this potential hazard is to link the last node with the first node in the list to create a *circularly-linked list*.

Circularly Linked List

- Two views of a circularly linked list:



Josephus Problem

- A case where circularly linked list comes in handy is the solution of the *Josephus Problem*.

Josephus Problem

- A case where circularly linked list comes in handy is the solution of the *Josephus Problem*.
- Consider there are 10 persons. They would like to choose a leader.

Josephus Problem

- A case where circularly linked list comes in handy is the solution of the *Josephus Problem*.
- Consider there are 10 persons. They would like to choose a leader.
- The way they decide is that all 10 sit in a circle.

Josephus Problem

- A case where circularly linked list comes in handy is the solution of the *Josephus Problem*.
- Consider there are 10 persons. They would like to choose a leader.
- The way they decide is that all 10 sit in a circle.
- They start a count with person 1 and go in clockwise direction and skip 3. Person 4 reached is eliminated.

Josephus Problem

- A case where circularly linked list comes in handy is the solution of the *Josephus Problem*.
- Consider there are 10 persons. They would like to choose a leader.
- The way they decide is that all 10 sit in a circle.
- They start a count with person 1 and go in clockwise direction and skip 3. Person 4 reached is eliminated.
- The count starts with the fifth and the next person to go is the fourth in count.

Josephus Problem

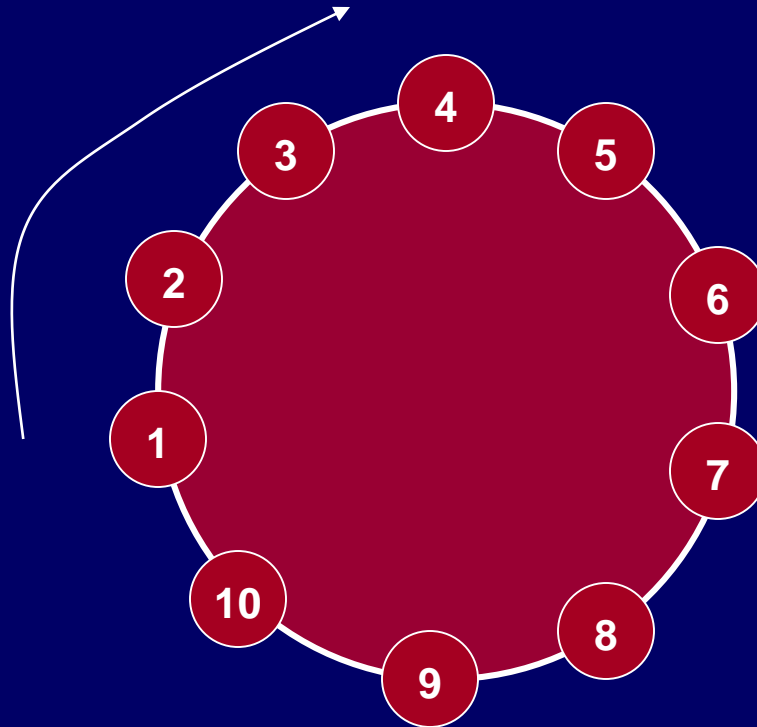
- A case where circularly linked list comes in handy is the solution of the *Josephus Problem*.
- Consider there are 10 persons. They would like to choose a leader.
- The way they decide is that all 10 sit in a circle.
- They start a count with person 1 and go in clockwise direction and skip 3. Person 4 reached is eliminated.
- The count starts with the fifth and the next person to go is the fourth in count.
- Eventually, a single person remains.

Josephus Problem

- A case where circularly linked list comes in handy is the solution of the *Josephus Problem*.
- Consider there are 10 persons. They would like to choose a leader.
- The way they decide is that all 10 sit in a circle.
- They start a count with person 1 and go in clockwise direction and skip 3. Person 4 reached is eliminated.
- The count starts with the fifth and the next person to go is the fourth in count.
- Eventually, a single person remains.

Josephus Problem

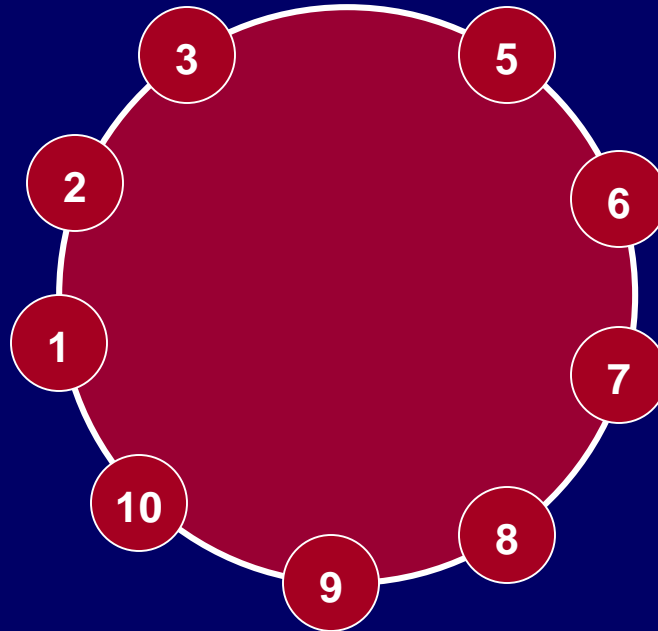
- $N=10, M=3$



Josephus Problem

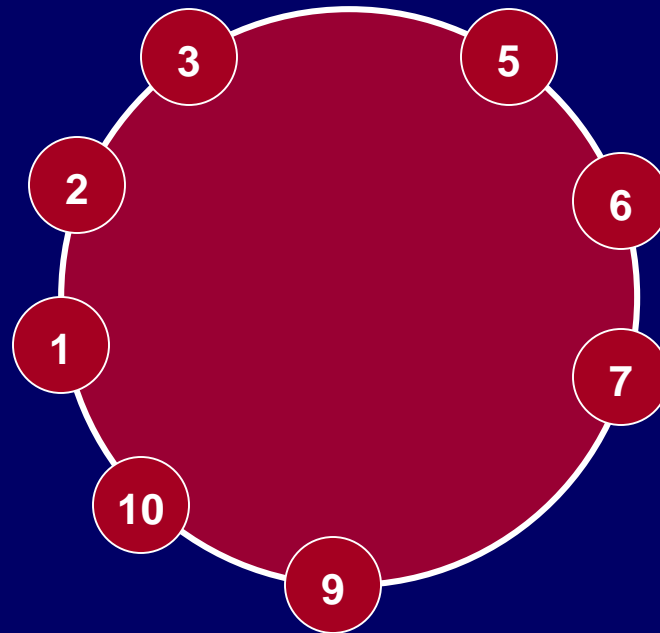
- $N=10, M=3$

eliminated



Josephus Problem

- $N=10, M=3$



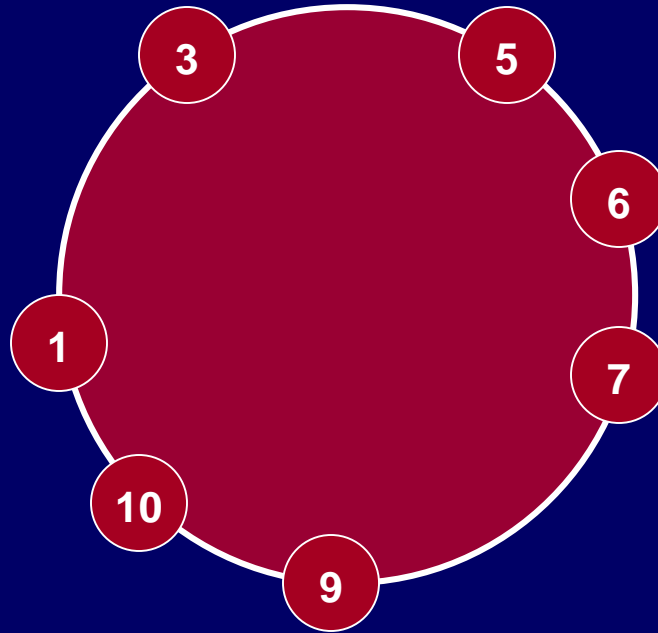
eliminated

4

8

Josephus Problem

- $N=10, M=3$

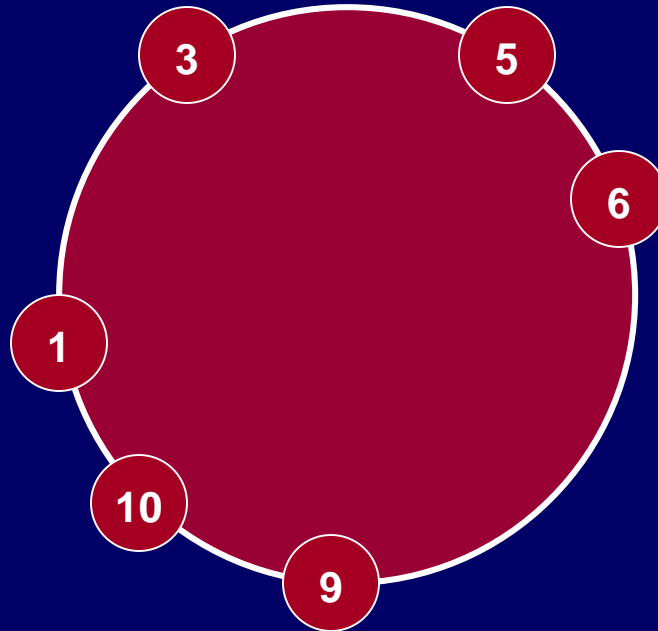


eliminated



Josephus Problem

- $N=10, M=3$



eliminated

4

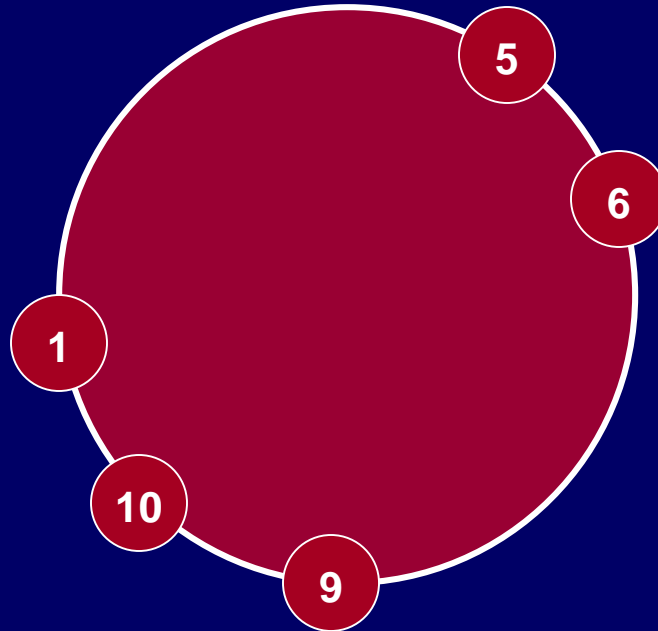
8

2

7

Josephus Problem

- $N=10, M=3$

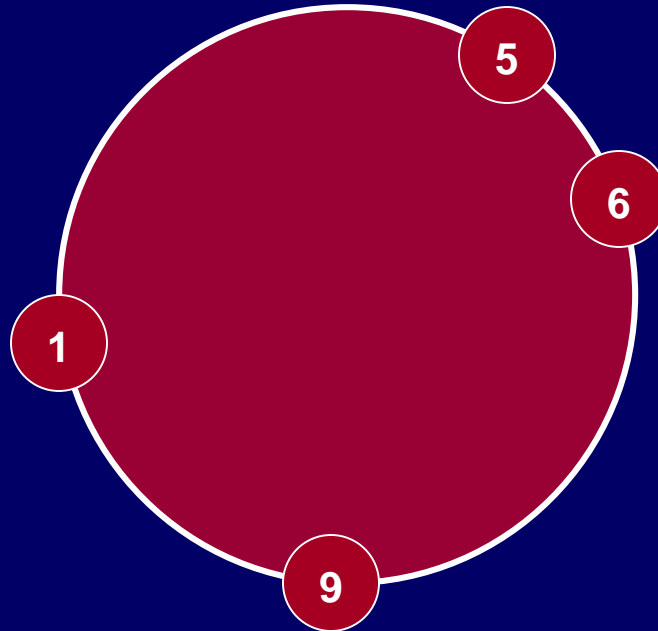


eliminated



Josephus Problem

- $N=10, M=3$

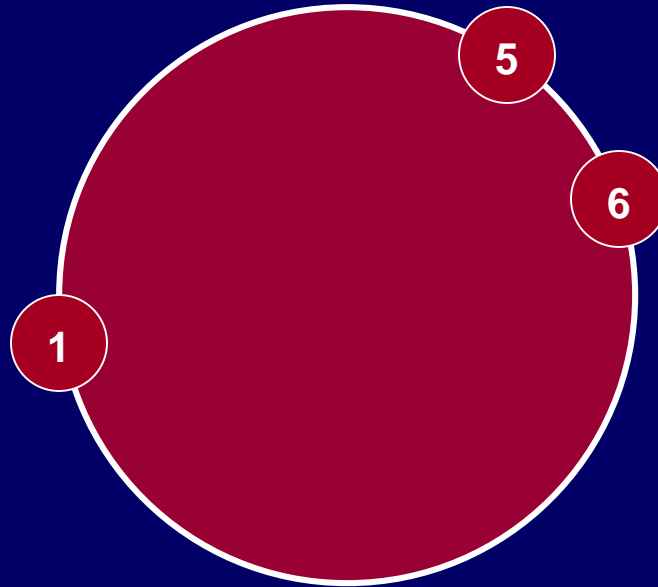


eliminated



Josephus Problem

- $N=10, M=3$

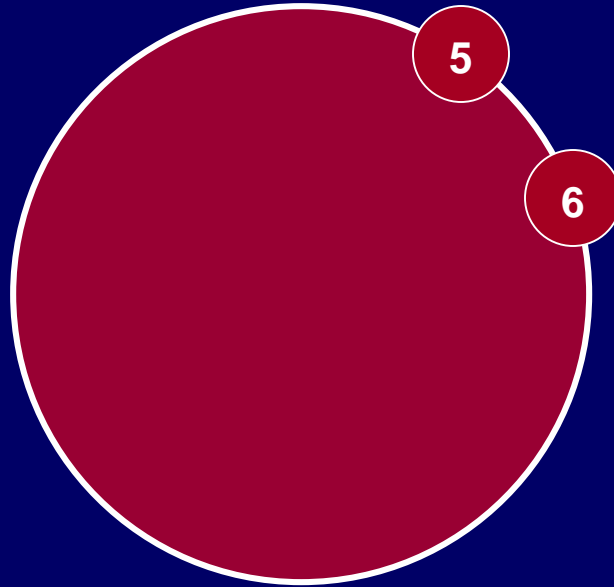


eliminated



Josephus Problem

- $N=10, M=3$

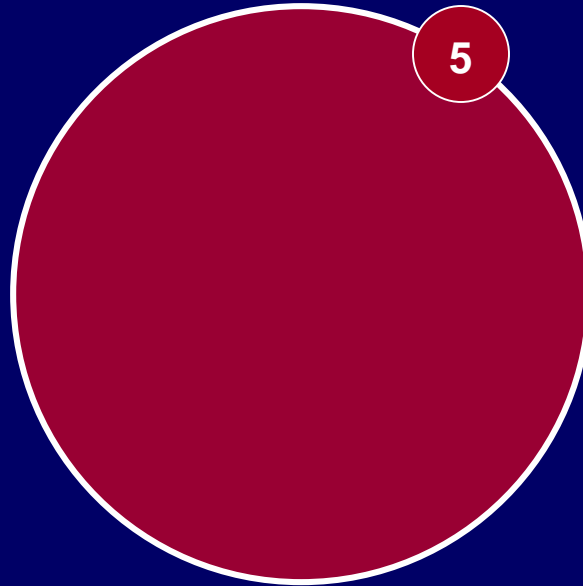


eliminated



Josephus Problem

- $N=10, M=3$



eliminated



Josephus Problem

```
➤ #include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```


Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```

Josephus Problem

```
#include "CList.cpp"
void main(int argc, char *argv[])
{
    CList list;
    int i, N=10, M=3;
    for(i=1; i <= N; i++ ) list.add(i);

    list.start();
    while( list.length() > 1 ) {
        for(i=1; i <= M; i++ ) list.next();
        cout << "remove: " << list.get() << endl;
        list.remove();
    }
    cout << "leader is: " << list.get() << endl;
}
```