Linked Lists

Cong Chen

Computer Memory

Address	Value
…	
<110>	
<111>	
<112>	
<113>	
<114>	
<115>	
<116>	
<117>	
<118>	
<119>	
…	

Memory Allocation

Address	Value
…	
<110>	
<111>	
<112>	
<113>	
<114>	
<115>	
<116>	
<117>	
<118>	
<119>	
…	

```cpp
// Program1.cpp

int main()
{
    int m = 9;
    float n = 2.7;
    int arr[6] = { 5,8,3,4,0,0 };

    cout << arr[2]; // output: 3

    return 0;
}
```

Memory Allocation

Address	Value		Program1	
…	…			
<110>	9		m	
<111>	2.7		n	
<112>	5		arr[0]	
<113>	8		arr[1]	
<114>	3		arr[2]	
<115>	4		arr[3]	
<116>	0		arr[4]	
<117>	0		arr[5]	
<118>				
<119>				
…	…			

```
// Program1.cpp

int main()
{
    int m = 9;
    float n = 2.7;
    int arr[6] = { 5,8,3,4,0,0 };

    cout << arr[2]; // output: 3

    return 0;
}
```

Array — Insertion

Address	Value
…	…
<110>	9
<111>	2.7
<112>	5
<113>	8
<114>	3
<115>	4
<116>	0
<117>	0
<118>	1024
<119>	7
…	…

⇦ 2

m	
n	
arr[0]	arr
arr[1]	
arr[2]	
arr[3]	
arr[4]	
arr[5]	

```
// Program1.cpp

int main()
{
    int m = 9;
    float n = 2.7;
    int arr[6] = { 5,8,3,4,0,0 };

    // TODO: Insert 2 at index 1
    // ...

    // arr: { 5,2,8,3,4,0 }

    return 0;
}
```

Array — Insertion

Address	Value
…	…
<110>	9
<111>	2.7
<112>	5
<113>	8
<114>	3
<115>	4
<116>	0
<117>	0
<118>	1024
<119>	7
…	…

2

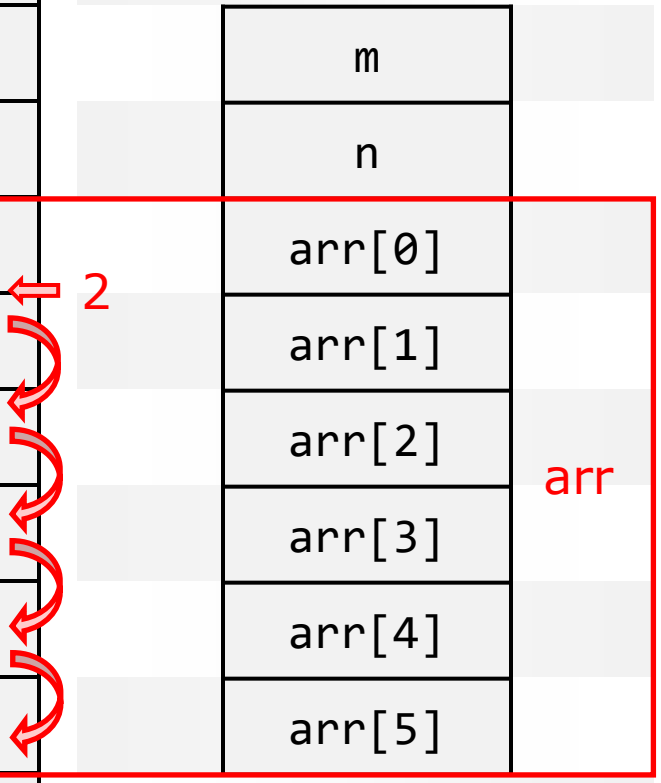
m
n
arr[0]
arr[1]
arr[2]
arr[3]
arr[4]
arr[5]

arr

```cpp
// Program1.cpp

int main()
{
    int m = 9;
    float n = 2.7;
    int arr[6] = { 5,8,3,4,0,0 };

    arr[5] = arr[4];
    arr[4] = arr[3];
    arr[3] = arr[2];
    arr[2] = arr[1];
    arr[1] = 2;

    // arr: { 5,2,8,3,4,0 }

    return 0;
}
```

Array — Insertion

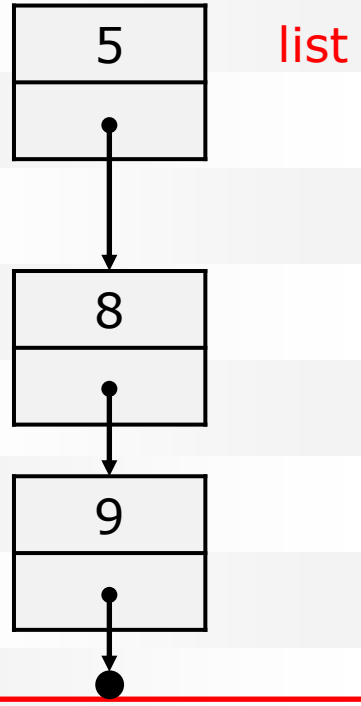
Address	Value
…	…
<110>	9
<111>	2.7
<112>	5
<113>	8
<114>	3
<115>	4
<116>	0
<117>	0
<118>	1024
<119>	7
…	…




```cpp
// Program1.cpp

int main()
{
    int m = 9;
    float n = 2.7;
    int arr[6] = { 5,8,3,4,0,0 };

    for ( int i = 5; i > 1; i-- )
        arr[i] = arr[i-1];

    arr[1] = 2;

    // arr: { 5,2,8,3,4,0 }

    return 0;
}
```

Linked List

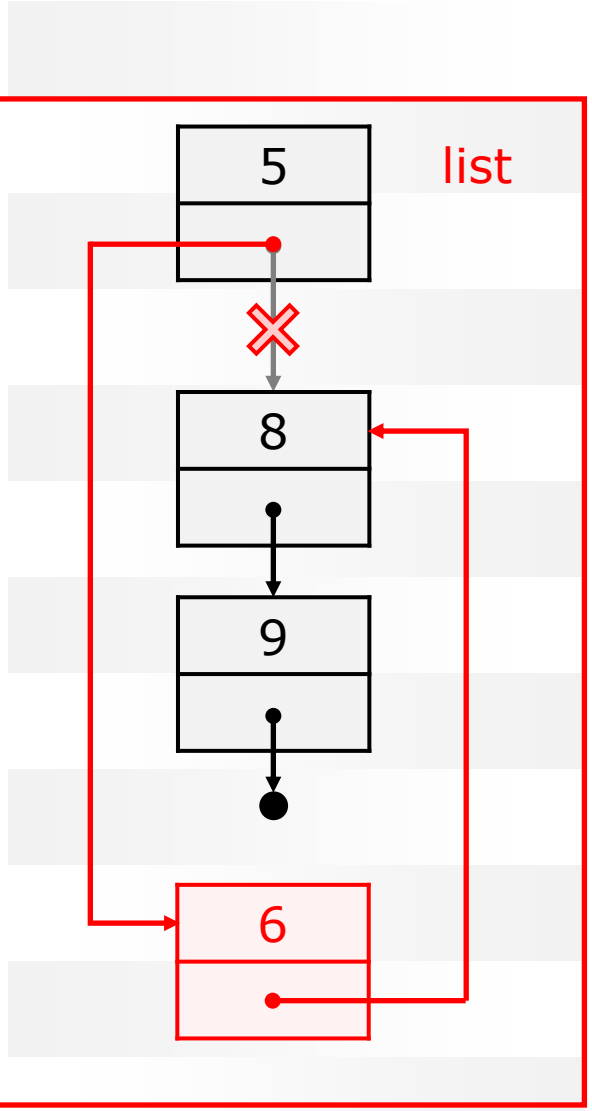
Address	Value
…	…
<110>	5
<111>	-113-
<112>	
<113>	8
<114>	-115-
<115>	9
<116>	-null-
<117>	
<118>	
<119>	
…	…




```
// LinkedList.cpp

int main()
{
    LinkedList list;
    list.append(5);
    list.append(8);
    list.append(9);

    list.print(); // output: 5 8 9

    return 0;
}
```

Linked List — Insertion

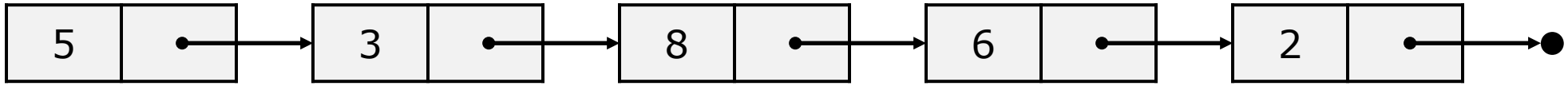
Address	Value
…	…
<110>	5
<111>	-118-
<112>	
<113>	8
<114>	-115-
<115>	9
<116>	-null-
<117>	
<118>	6
<119>	-113-
…	…




```
// LinkedList.cpp

int main()
{
    LinkedList list;
    list.append(5);
    list.append(8);
    list.append(9);

    list.insert(6);

    list.print(); // output: 5 6 8 9

    return 0;
}
```

Linked Lists

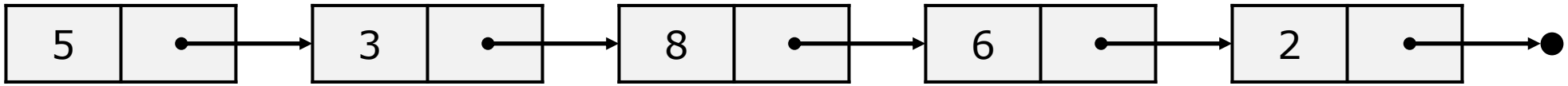
A linked list is a linear collection of nodes whose order is not given by their physical placement in memory. Instead, each node points to the next.



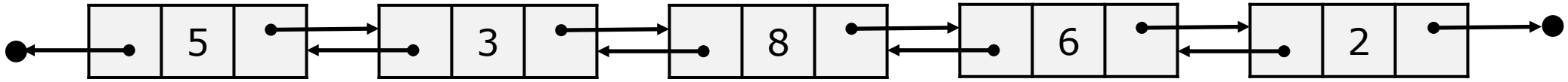
It is a data structure consisting of a collection of nodes which together represent a sequence.

Linked Lists – Types

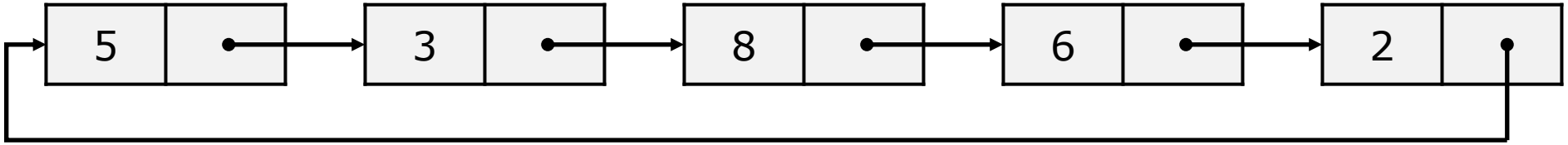
A Singly Linked List:



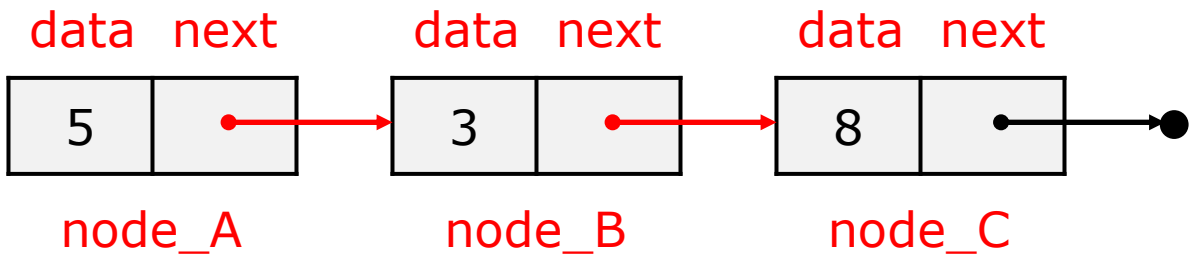
A Doubly Linked List:



A Circular Linked List:



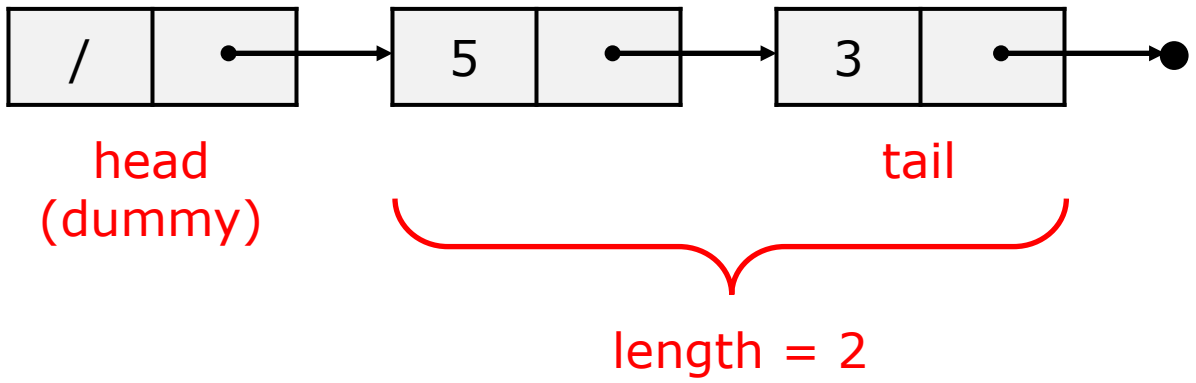
Linked Lists – Nodes




```
// Example

Node* node_D = new Node();
node_D->data = 2;
node_D->next = node_A->next;
Node_A->next = node_D
```

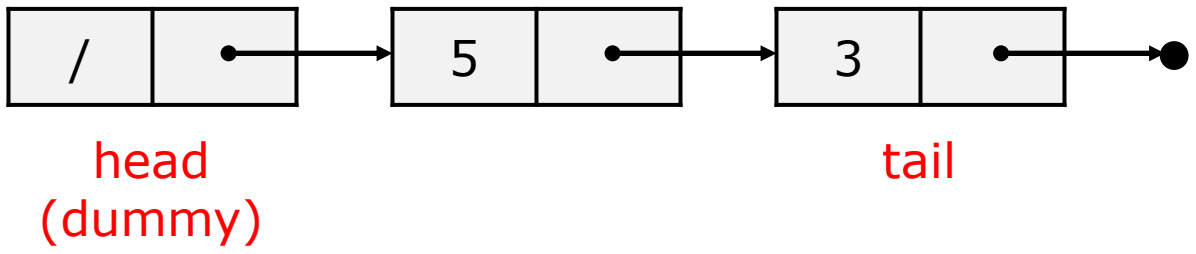
```
// Node.cpp

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

int main()
{
    Node* node_A = new Node();
    node_A->data = 5;
    Node* node_B = new Node();
    node_B->data = 3;
    Node* node_C = new Node();
    node_C->data = 8;

    node_A->next = node_B;
    node_B->next = node_C;

    // ...
```

Linked Lists – Structure



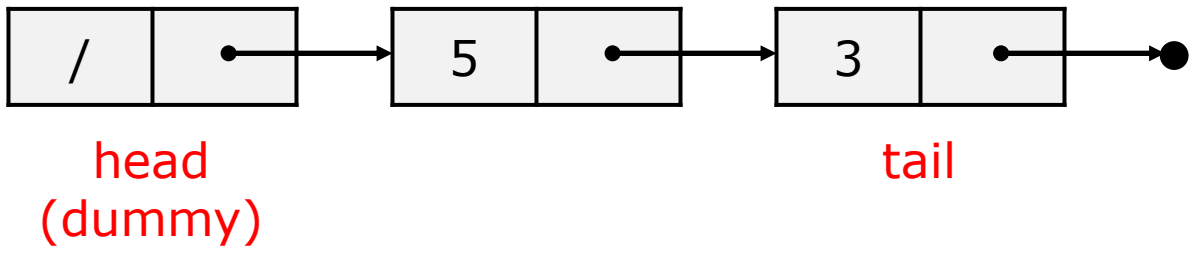

```
// LinkedList.h

struct LinkedList
{
    Node* head = new Node();
    Node* tail = head;
    int length = 0;

    void append(int d);
    void prepend(int d);
    void print();
    void insert(int d);

    // ...
};
```

Linked Lists – Append



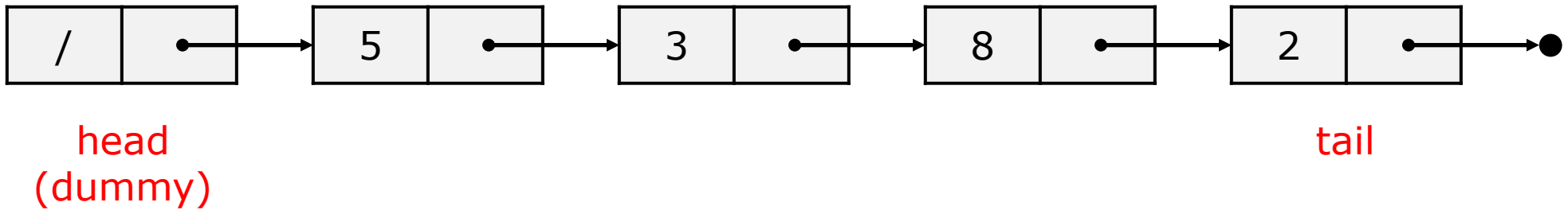

```
// LinkedList.cpp

void LinkedList::append(int d)
{
    Node* newNode = new Node();
    newNode->data = d;

    tail->next = newNode;
    tail = newNode;

    length++;
}
```

Linked Lists – Prepend



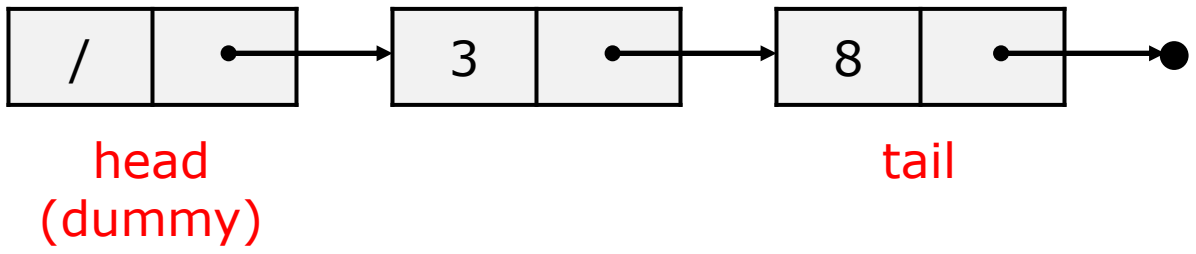

```
// LinkedList.cpp

void prepend(int d)
{
    Node* newNode = new Node();
    newNode->data = d;

    newNode->next = head->next;
    head->next = newNode;

    length++;
}
```

Linked Lists – Print (Traverse)




```
// LinkedList.cpp

void LinkedList::print()
{
    Node* curNode = head->next;

    while ( curNode )
    {
        cout << curNode->data << endl;
        curNode = curNode->next;
    }
}
```

Linked Lists – Insert (Sorted)




```cpp
// LinkedList.cpp

void LinkedList::insert(int d)
{
    Node* preNode = head;
    Node* curNode = head->next;

    while ( curNode )
    {
        if ( curNode->data > d ) break;

        preNode = curNode;
        curNode = curNode->next;
    }

    Node* newNode = new Node();
    newNode->data = d;

    preNode->next = newNode;
    newNode->next = curNode;

    length++;
}
```