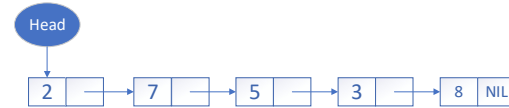


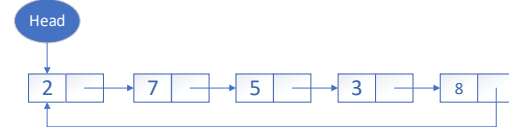
linked list

linked list

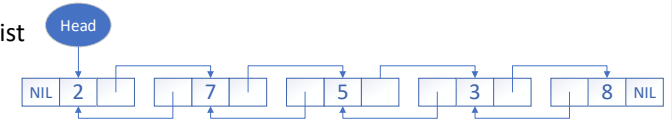
- singly linked list



- circular linked list



- doubly linked list



Singly Linked List

```
struct SLL {
    SSLNode* next;
};
```

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

- Initialize an empty list A:
 - SLL A; A.next = nullptr;
- Add a 2 to the list.
 - Create a new node and store 2 in the new node
 - The "next" of the tail node points to the new node.
- Add a 7 to the list.
 - Create a new node and store 7 in the new node
 - The "next" of the tail node points to the new node.
- Add a 5 to the list.
 - Create a new node and store 5 in the new node
 - The "next" of the tail node points to the new node.
- Do you see any problem?
 - push_back is slow, $O(n)$ (unless?)

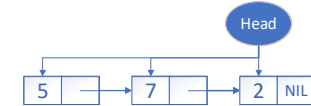


Singly Linked List

```
struct SLL {
    SSLNode* head;
};
```

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

- Initialize an empty list A:
 - SLL A; A.head = nullptr;
- Add a 2 to the list.
 - Create a new node and store 2 in the new node
 - the new node's next records head's next.
 - The head's next points to the new node
- Add a 7 to the list.
 - Create a new node and store 7 in the new node
 - the new node's next records head's next.
 - The head's next points to the new node
- Add a 5 to the list.
 - Create a new node and store 5 in the new node
 - the new node's next records head's next.
 - The head's next points to the new node.



O(1) insertion
Insert After Head

C++11's forward_list

Exercise: make the following code work

```
int main() {
    SLL a;
    a.push_front(10);
    a.push_front(20);
    a.push_front(30);
    a.show();           // {30,20,10}

    SLL b;
    b.show();           // {}

    return 0;
}
```

learn to use debugger

Exercise: make the following code work

```
int main() {
    SLL a;
    a.push_back(10);
    a.push_back(20);
    a.push_back(30);
    a.show();           // {10, 20, 30}

    SLL b;
    b.show();           // {}

    return 0;
}
```

learn to use debugger

Singly Linked List

```
struct SLL {
    SLLNode* next;
};
```

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

• Insert a 3 between 7 & 2

- Insert after 7
- Insert before 2
- Are they the same?

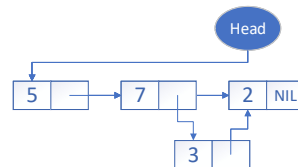
Insert after:

```
SLLNode *it = SLL.head;
it->data ?
it = it->next;
it->data ?
auto tmp = new SLLNode;
tmp->data = 3;
tmp->next = it->next;
it->next = tmp;
```

// does these steps break if we do insert after the last node?

Insert before:

```
SLLNode *it = SLL.head;
it->data ?
it = it->next;
it->data ?
it = it->next;
it->data ?
// how do we insert before?
```



Exercise: make the following code work

```
int main() {
    SLL a;
    a.push_back(10);
    a.push_back(20);
    a.push_back(30);
    a.show();           // {10, 20, 30}
    a.insert_after(20, 25);
    a.show();           // {10, 20, 25, 30}
    SLL b;
    b.show();           // {}

    return 0;
}
```

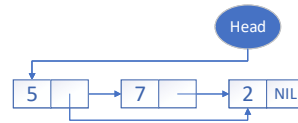
learn to use debugger

Singly Linked List

```
struct SLL {  
    SSLNode* next;  
};
```

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

- Erase 7



Erase:

```
SLLNode *it = SLL.head;  
it->data ?  
auto toRemove = it->next;  
it->next = toRemove->next;  
delete toRemove;
```

C++11's `forward_list`