Section 5

Linked Lists

Presentation by *Asem Alaa*

Linked Lists

Arrays vs. LL

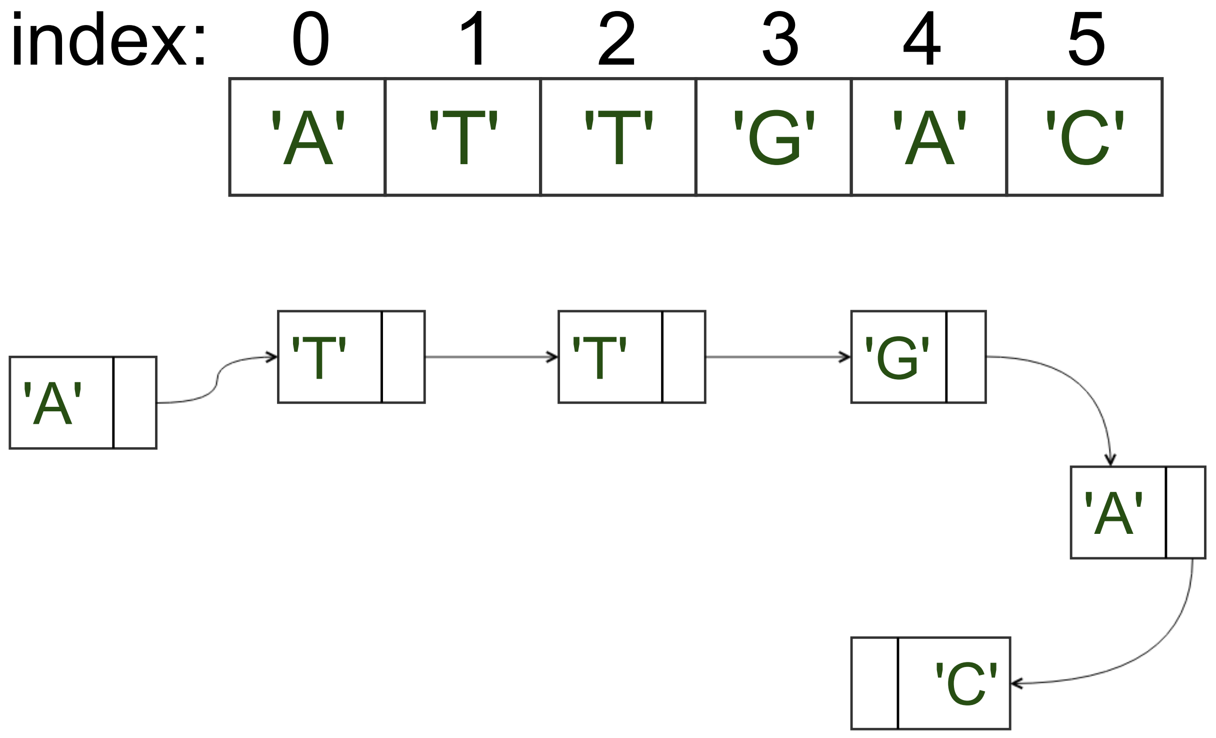
- **Arrays** => **contiguous elements** in the memory.
- **LL** => **sparse** in memory,

Each element in **LL** can see the *next* element.

Why linked lists

- Very flexible in insertion/removal.
- **Arrays** => fixed sizes
- **Arrays** => expensive insertion

The Memory Mode: Array vs. Linked List

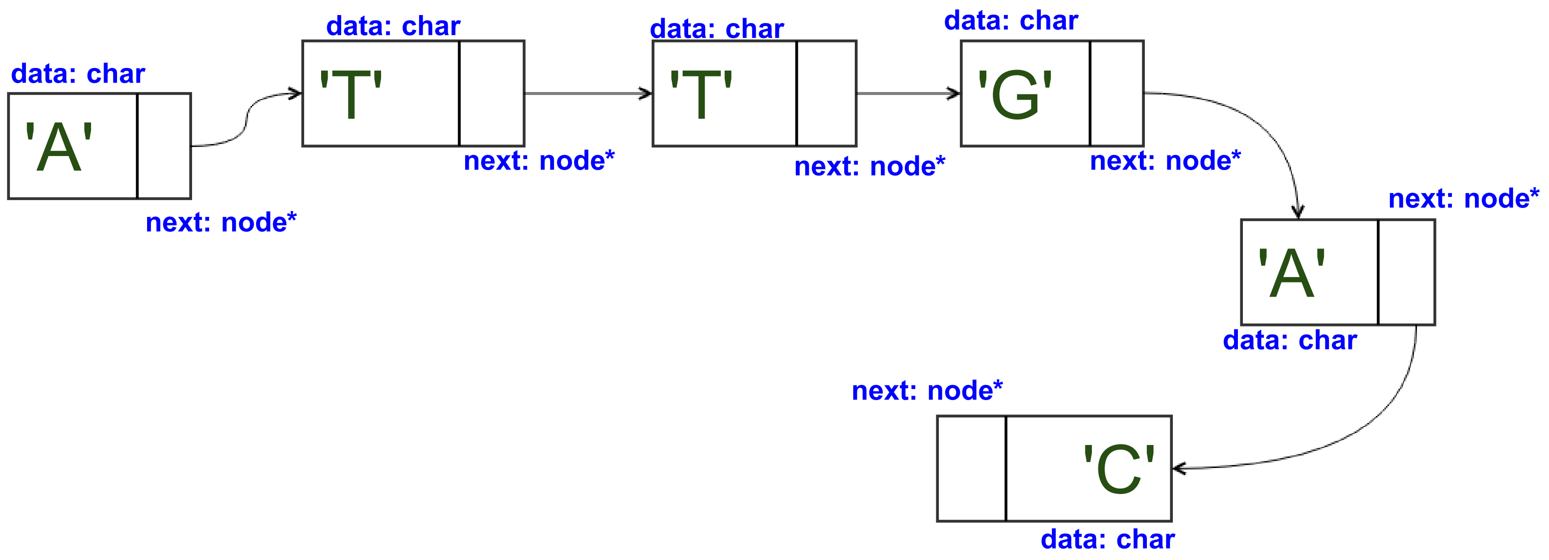


Pointers revisited

- Each element => *node*.
- To connect between nodes => **pointers**.
- *node* has a **pointer** pointing to the *next node*.

DNA sequence as a Linked List (LL)

```
struct node
{
    char data;
    node* next;
};
```



Linked list of doubles

```
struct node
{
    double data;
    node* next;
};
```

Insert element to the front:

```
node *pushFront( node *front , double data )
{
    front = new node{ data , front };
    return front;
}
```

Alternatively...

```
node *pushFront( node *front , double data )
{
    return new node{ data , front };
}
```

Insert element to the back

```
node *pushBack( node *front, double data )
{
    if( front == nullptr )
    {
        front = new node{ data , front };
        return front;
    }
    else
    {
        node *temp = front;
        while( temp->next != nullptr )
            temp = temp->next;
        temp->next = new node{ data , nullptr };
        return front;
    }
}
```

Insert element to the back (+DRY)

```
node *pushBack( node *front, double data )
{
    if( front == nullptr )
        return pushFront( front , data );
    else
    {
        node *temp = front;
        while( temp->next != nullptr )
            temp = temp->next;
        temp->next = new node{ data , nullptr };
        return front;
    }
}
```

The last node (back)

```
node *backNode( node *front )
{
    node *temp = front;
    while( temp->next != nullptr )
        temp = temp->next;
    return temp;
}
```

Insert element to the back (+DRY)

```
node *backNode( node *front )
{
    node *temp = front;
    while( temp->next != nullptr )
        temp = temp->next;
    return temp;
}

node *pushBack( node *front, double data )
{
    if( front == nullptr )
        return pushFront( front , data );
    else
    {
        node *temp = front;
        while( temp->next != nullptr )
            temp = temp->next;
        temp->next = new node{ data , nullptr };
        return front;
    }
}
```

Insert element to the back (++DRY)

```
node *backNode( node *front )
{
    node *temp = front;
    while( temp->next != nullptr )
        temp = temp->next;
    return temp;
}

node *pushBack( node *front, double data )
{
    if( front == nullptr )
        return pushFront( front , data );
    else
    {
        node *back = backNode( front );
        back->next = new node{ data , nullptr };
        return front;
    }
}
```

Linked list in main function

```cpp
#include <iostream>
struct node
{
    double data;
    node* next;
};
node *pushFront( node *front , double data ){ ... }
node *backNode( node *front ){ ... }
node *pushBack( node *front, double data ){ ... }
int main()
{
    node* l = nullptr; // Empty list
    // append elements with values 1^2 to 10^2
    for (int i=1; i<=10; ++i)
        l = pushBack( l , i*i);
    // print all elements followed by a space
    for (node *temp = l; temp != nullptr; temp = temp->next )
        std::cout << temp->data << ' ';
}
```

Linked list traversal

```
void printLL( node* front )
{
    node *current = front;
    while( current != nullptr )
    {
        std::cout << current->data;
        current = current->next;
    }
}
```

C++ STL Linked Lists vs Dynamic Arrays

Includes

```
#include <vector>
```

```
#include <list>
```

C++ STL Linked Lists vs Dynamic Arrays

Construction

```cpp
#include <vector>
#include <list>
#include <iostream>

int main()
{
    std::vector< double > v;
    std::list< double > l;
}
```

C++ STL Linked Lists vs Dynamic Arrays

Insertion of $1, 2^2, 3^2, \ldots, 10^2$

```cpp
#include <vector>
#include <list>
#include <iostream>

int main()
{
    std::vector< double > v;
    std::list< double > l;
    // append elements with values 1^2 to 10^2
    for (int i=1; i<=10; ++i) {
        v.push_back(i*i);
    }

    // append elements with values 1^2 to 10^2
    for (int i=1; i<=10; ++i) {
        l.push_back(i*i);
    }
}
```

C++ STL Linked Lists vs Dynamic Arrays

Traversal

Print all elements followed by a space

```
for (int i=0; i< v.size(); ++i) {  // O(n)
    std::cout << v[i] << ' ';
}
for (int i=0; i< l.size(); ++i) { //
    // operator[] is undefined for list.
    std::cout << l[i] << ' '; // Compiler error
}
for (int i=0; i< l.size(); ++i) { // O(n^2) time
    auto it = std::next( l.begin(), i );
    std::cout << *it << ' ';
}
for (auto it =l.begin(); it != l.end(); ++it) { // O(n)
    std::cout << *it << ' ';
}
```

C++ STL Linked Lists vs Dynamic Arrays

Traversal (Universal Approach for STL containers)

```
    // print all elements followed by a space
    for (double x : v )
        std::cout << x << ' ';

    // print all elements followed by a space
    for (double x : l)
        std::cout << x << ' ';
```

This also works

```
    // print all elements followed by a space
    for (auto x : v )
        std::cout << x << ' ';

    // print all elements followed by a space
    for (auto x : l)
        std::cout << x << ' ';
```

General Linked List (LL): 11 operations

- insertion at front (`pushFront`).
- insertion at back (`pushBack`).
- remove from front (`popFront`).
- remove from back (`popBack`).
- remove nth element (`removeAt`).
- return front (`getFront`).
- return back (`getBack`).
- return nth element (`getAt`).
- is empty? (`isEmpty`)
- print all elements (`printAll`)
- delete the whole list from the heap (`clear`).

A: LL of Integers

Define Your New Types

```
struct IntegerNode
{
    int data;
    IntegerNode *next = nullptr;
};

struct IntegerLL
{
    IntegerNode *front;
};
```

Access

```
int getFront( IntegerLL &list )
{ /* Logic */ }

IntegerNode* backNode( IntegerLL &list )
{ /* Logic */ }

int getBack( IntegerLL &list )
{ /* Logic */ }

IntegerNode* nodeAt( IntegerLL &list, int index )
{ /* Logic */ }

int getAt( IntegerLL &list , int index )
{ /* Logic */ }
```

Access

```
IntegerNode* nodeAt( IntegerLL &list, int index )
{
    IntegerNode* temp = list.front;
    for( int i = 0; i < index ; ++i )
        temp = temp->next;
    return temp;
}
```

Insertions

```cpp
void pushFront( IntegerLL &list, int data )
{ /* Logic */ }

void pushBack( IntegerLL &list, int data )
{ /* Logic */ }
```

Insertions

```
struct IntegerLL
{
    IntegerNode *front;
};

IntegerNode* pushFront( IntegerNode* front, int data )
{
    front = new node{ data , list.front };
    return front;
}

void pushFront( IntegerLL &list, int data )
{
    list.front = new node{ data , list.front };
}

void pushBack( IntegerLL &list, int data )
{ /* Logic */ }
```

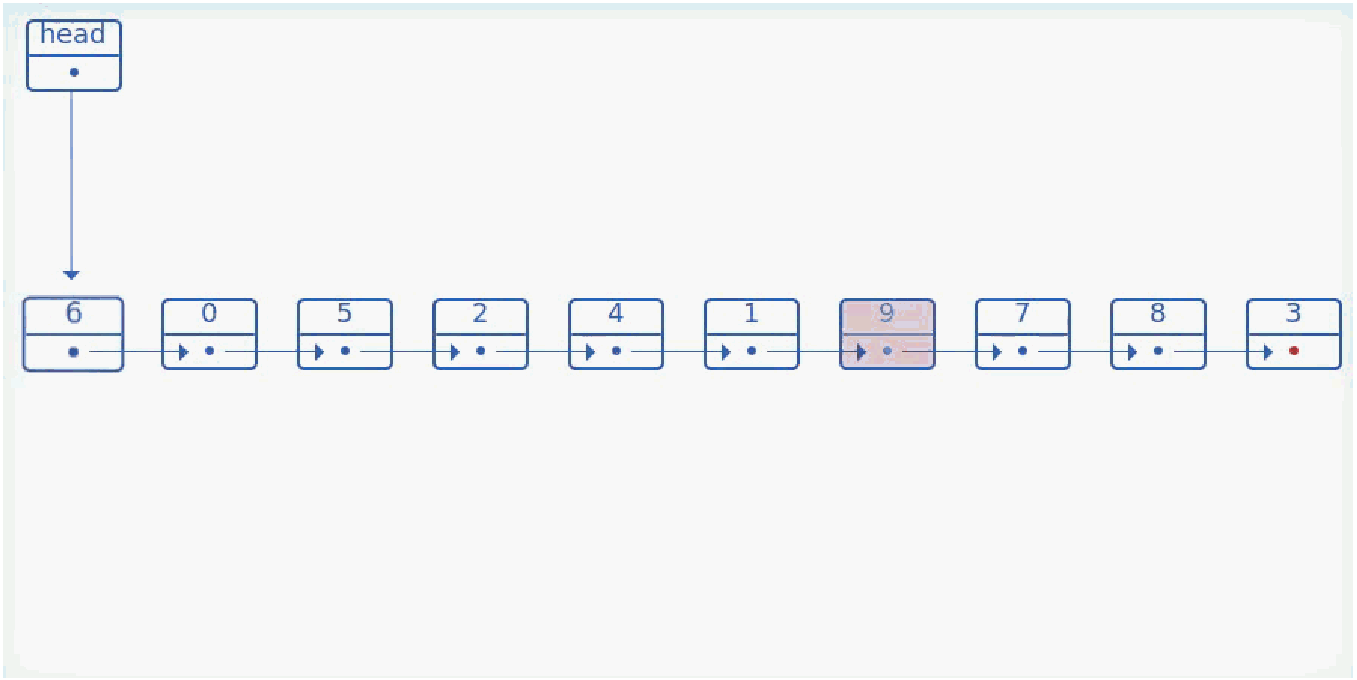
Are you empty?

```
bool isEmpty( IntegerLL &list )
{
    if( list.front == nullptr )
        return true;
    else return false;
}
```

Are you empty?

```
bool isEmpty( IntegerLL &list )
{
    return list.front == nullptr;
}
```

Removal



```
void removeBack( IntegerLL &list )
{ /* Logic */ }

void removeFront( IntegerLL &list )
{ /* Logic */ }

void removeAt( IntegerLL &list, int index )
{ /* Logic */ }
```

Image credits: © algomation.com

Removal

```
void removeBack( IntegerLL &list )
{

}
```

Removal

```
void removeBack( IntegerLL &list )
{
    if( isEmpty( list ))
        return;
    else if( list.front->next == nullptr )
        removeFront( list );
    else
    {
        IntegerNode *prev = list.front;
        while( prev->next->next != nullptr )
            prev = prev->next;
        delete prev->next;
        prev->next = nullptr;
    }
}
```

Traverse, clear

```
void printAll( IntegerLL &list )
{ /* Logic */ }


void clear( IntegerLL &list )
{ /* Logic */ }
```

Exercise: Copy your logic for Linked List of characters

Copy-paste the same logic of the Linked List of integers, but change each:

- `int` to `char`,
- `IntegerLL` to `CharLL`, and
- `IntegerNode` to `CharNode`.

Next Lab

Turning "free functions" to "methods"

```
int main()
{
    node* l = nullptr; // Empty list
    for (int i=1; i<=10; ++i)
        l = pushBack( l , i*i);
}
```

```
int main()
{
    LLDouble l; // Empty list
    for (int i=1; i<=10; ++i)
        l.pushBack( i*i );
}
```

Next Lab

Templetize (DRY+++++)

```
int main()
{
    LLDouble ld; // Empty list of doubles
    LLInteger li; // Empty list of integers
}
```

```
int main()
{
    LL<double> ld; // Empty list of doubles
    LL<int>  li; // Empty list of integers
}
```

Thank you