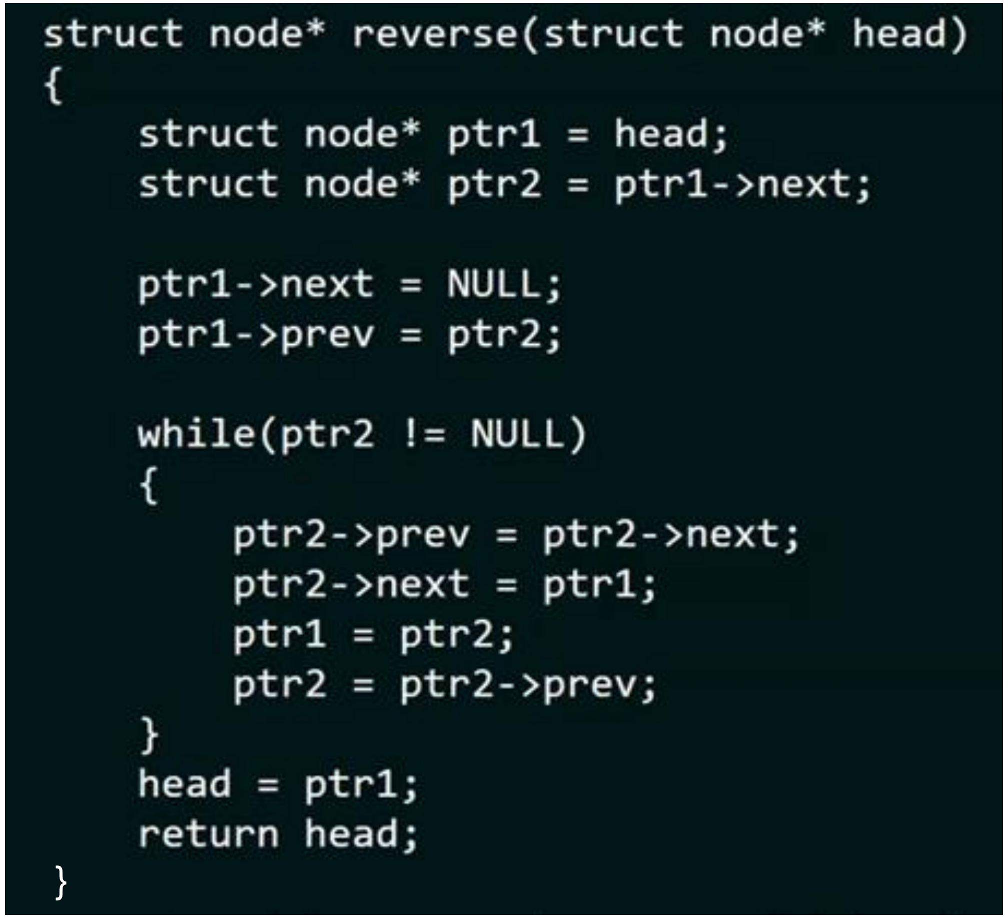
BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) DOUBLY LINKED LISTS CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

© C R Hota, BITS Hyderabad

REVERSING A DOUBLY-LINKED LIST

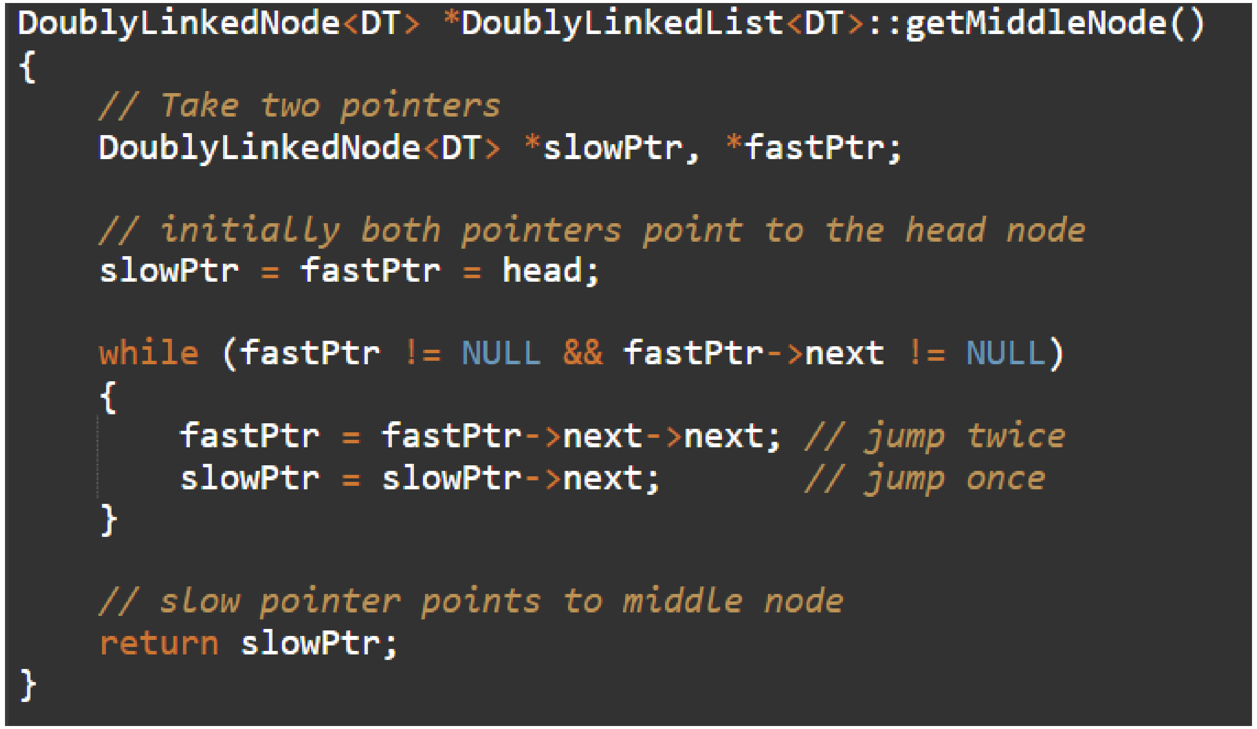
```
void listReverse(DLinkedList& L) {

    DLinkedList T; // temporary list
    while (!L.empty()) { // reverse L into T
            string s = L.front(); L.removeFront();
            T.addFront(s);
     }
    while (!T.empty()) { // copy T back to L
            string s = T.front();
            T.removeFront();
            L.addBack(s);
     }
}
```



```
struct node* reverse(struct node* head)
{
    struct node* ptr1 = head;
    struct node* ptr2 = ptr1->next;

    ptr1->next = NULL;
    ptr1->prev = ptr2;

    while(ptr2 != NULL)
    {
        ptr2->prev = ptr2->next;
        ptr2->next = ptr1;
        ptr1 = ptr2;
        ptr2 = ptr2->prev;
    }
    head = ptr1;
    return head;
}
```

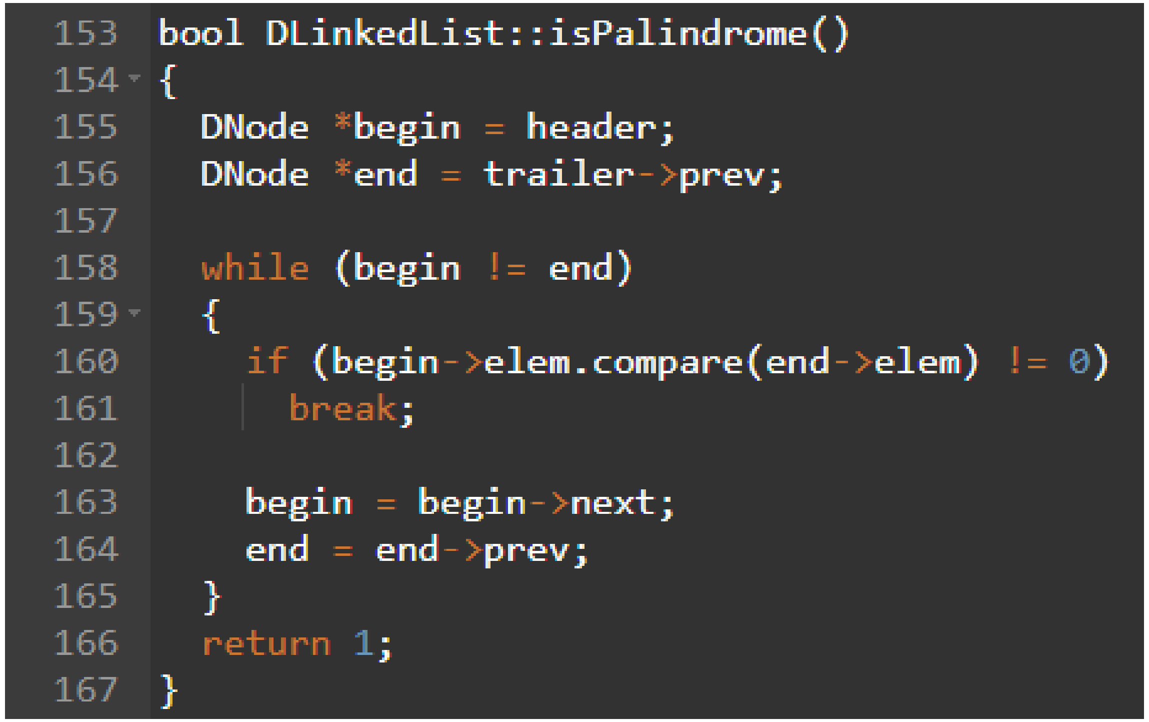
Let us see on the board its' working!

MIDDLE NODE AND LOOP IN A LINKED LIST



```
DoublyLinkedNode<DT> *DoublyLinkedList<DT>::getMiddleNode()
{
    // Take two pointers
    DoublyLinkedNode<DT> *slowPtr, *fastPtr;

    // initially both pointers point to the head node
    slowPtr = fastPtr = head;

    while (fastPtr != NULL && fastPtr->next != NULL)
    {
        fastPtr = fastPtr->next->next; // jump twice
        slowPtr = slowPtr->next;       // jump once
    }

    // slow pointer points to middle node
    return slowPtr;
}
```

Lab 4



```
bool DLinkedList::isPalindrome()
{
  DNode *begin = header;
  DNode *end = trailer->prev;

  while (begin != end)
  {
    if (begin->elem.compare(end->elem) != 0)
      break;

    begin = begin->next;
    end = end->prev;
  }
  return 1;
}
```

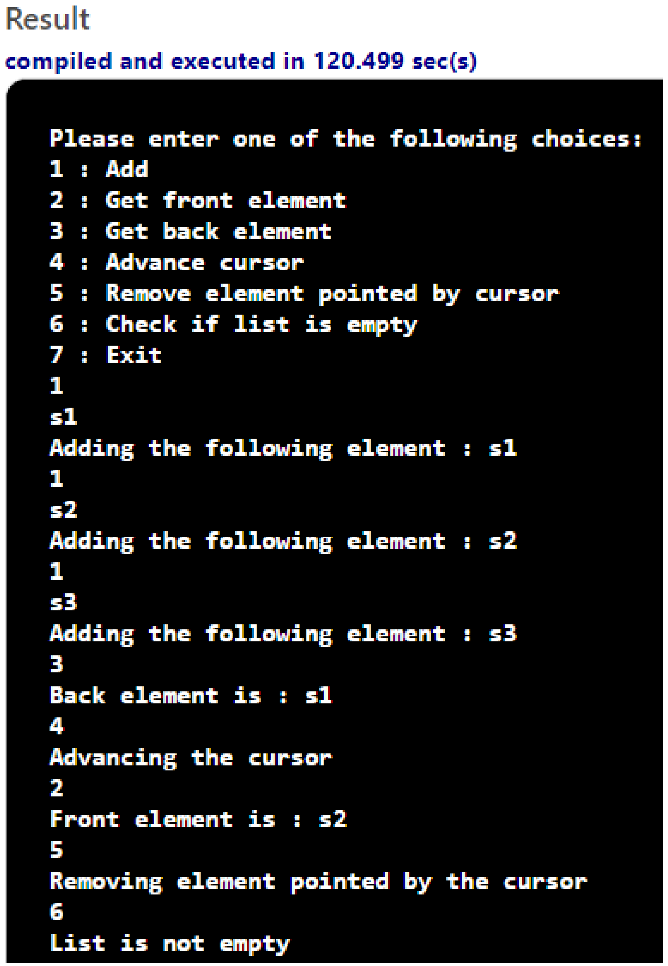
CIRCULAR LINKED LISTS

- A circular linked list is a singly-linked list except element of the list pointing to the first. Without s we can go back to the first.

- What is the need of cursor node?

```
class CircleList;
typedef string Elem;
class CNode {
private: Elem elem;
         CNode* next;
         friend class CircleList;
};
class CircleList {
public:   CircleList();
          ~CircleList();
           bool empty() const;
           const Elem& front() const;
           const Elem& back() const;
           void advance();
           void add(const Elem& e);
           void remove();
private:   CNode* cursor;
};
```

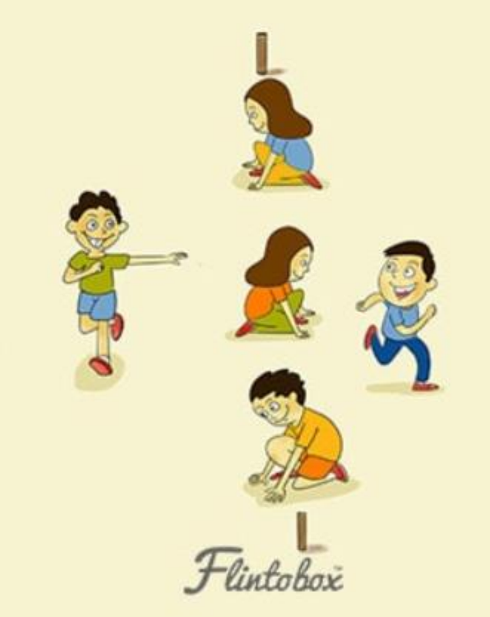
```
CircleList::CircleList() : cursor(NULL)
CircleList::~CircleList() { while (!emp
bool CircleList::empty() const {retur
const Elem& CircleList::back() const
                                  ret
const Elem& CircleList::front() const
                            return cu
void CircleList::advance() { cursor =
void CircleList::add(const Elem& e)
   CNode* v = new CNode;
   v->elem = e;
   if (cursor == NULL) {
      v->next = v;  cursor = v; }
   else {
      v->next = cursor->next; curs
   }
}
```



Result

compiled and executed in 120.499 sec(s)

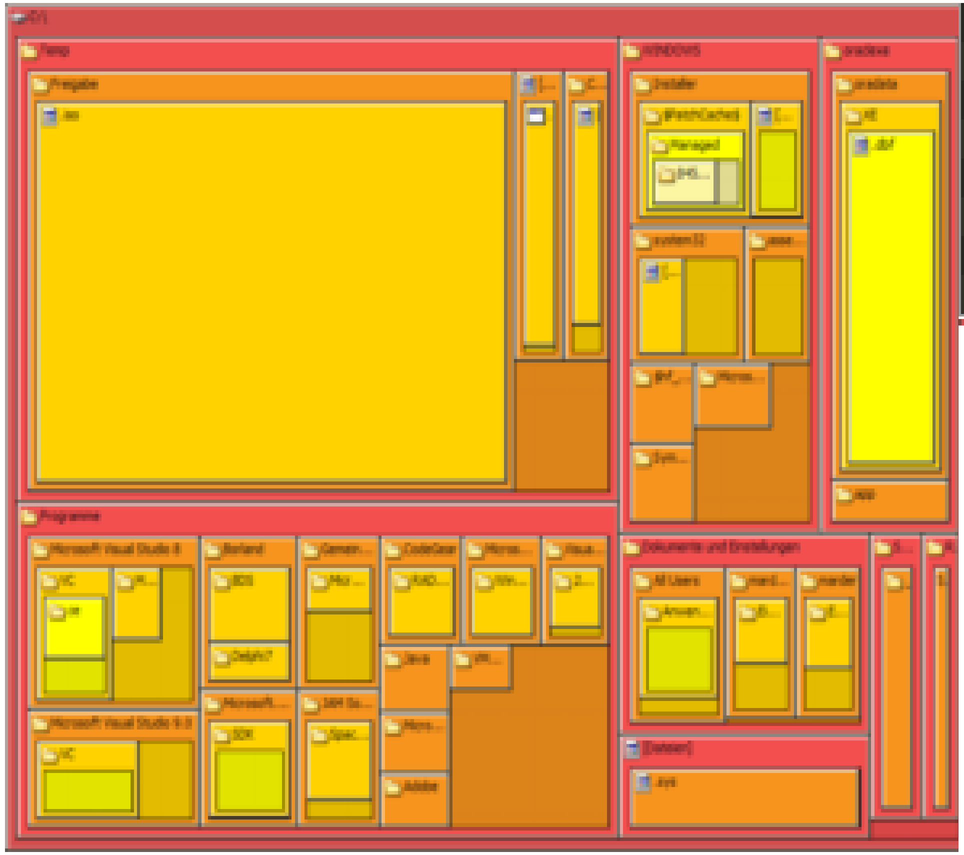
```
Please enter one of the following choices:
1 : Add
2 : Get front element
3 : Get back element
4 : Advance cursor
5 : Remove element pointed by cursor
6 : Check if list is empty
7 : Exit
1
s1
Adding the following element : s1
1
s2
Adding the following element : s2
1
s3
Adding the following element : s3
3
Back element is : s1
4
Advancing the cursor
2
Front element is : s2
5
Removing element pointed by the cursor
6
List is not empty
```

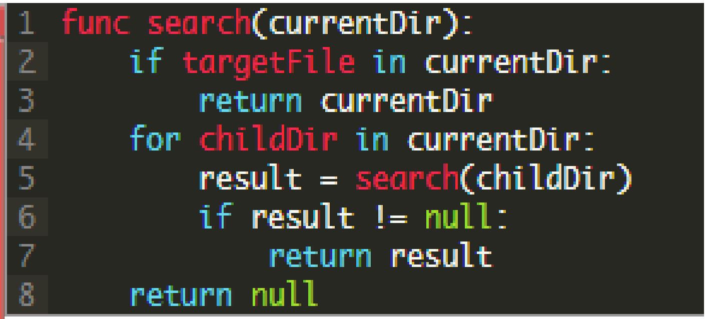


```
// remove the node after the cursor
void CircleList::remove() {
  CNode* old = cursor->next;
  if (old == cursor)
     cursor = NULL;
  else
     cursor->next = old->next;
     delete old;
}
```

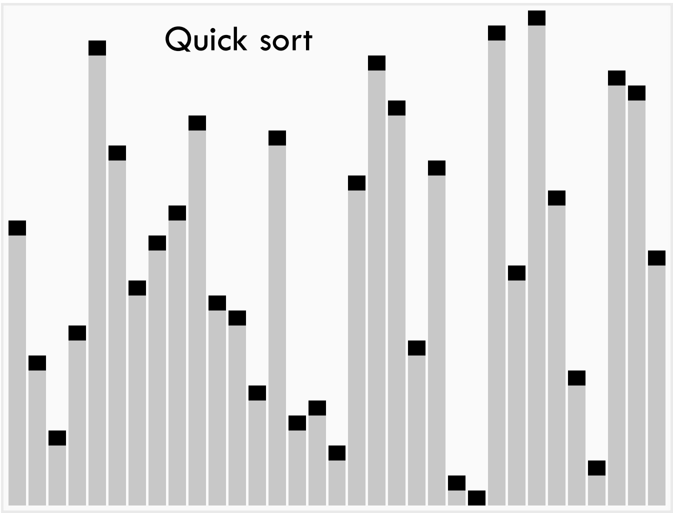
RECURSION: ELEGANT WAY FOR REPETITIVE TASKS

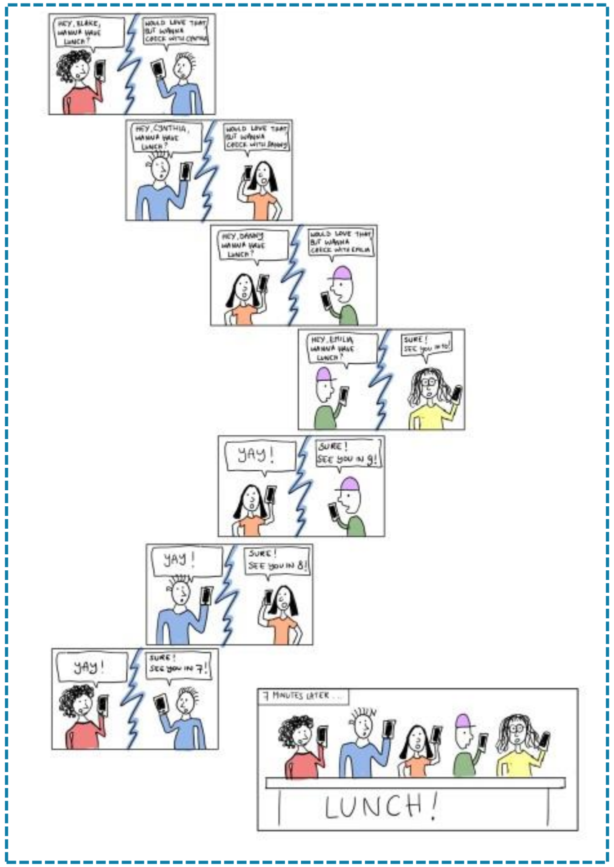
Recursion: When a function or a method calls itself. A set of problems can be solved easily using recursion (a powerful programming tool).





```
func search(currentDir):
    if targetFile in currentDir:
        return currentDir
    for childDir in currentDir:
        result = search(childDir)
        if result != null:
            return result
    return null
```





HEY, BLAKE, WANNA HAVE LUNCH?
WOULD LOVE THAT, BUT WANNA CHECK WITH CYNTHIA
HEY, CYNTHIA, WANNA HAVE LUNCH?
WOULD LOVE THAT BUT WANNA CHECK WITH DANNY
HEY, DANNY WANNA HAVE LUNCH?
WOULD LOVE THAT BUT WANNA CHECK WITH EMILIA
HEY, EMILIA, WANNA HAVE LUNCH?
SURE! SEE YOU IN 10!
YAY!
SURE! SEE YOU IN 9!
YAY!
SURE! SEE YOU IN 8!
YAY!
SURE! SEE YOU IN 7!
7 MINUTES LATER...
LUNCH!

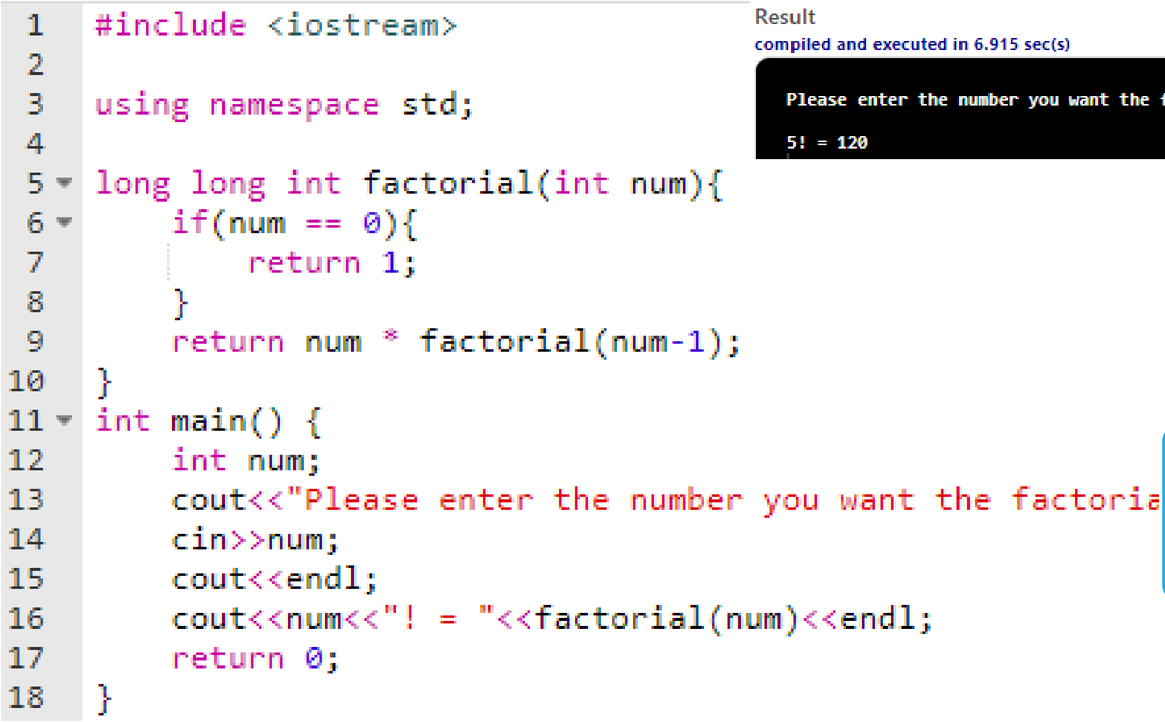
https://www.techiedelight.com/recursion-practice-problems-with-solutions/ (for practice)

https://abetterscientist.wordpress.com/

RECURSION: ELEGANT WAY FOR REPETITIVE TASKS

Recursion: When a function or a method calls itself. A set of problems can be solved easily using recursion (a powerful programming tool).

Recursive definition:

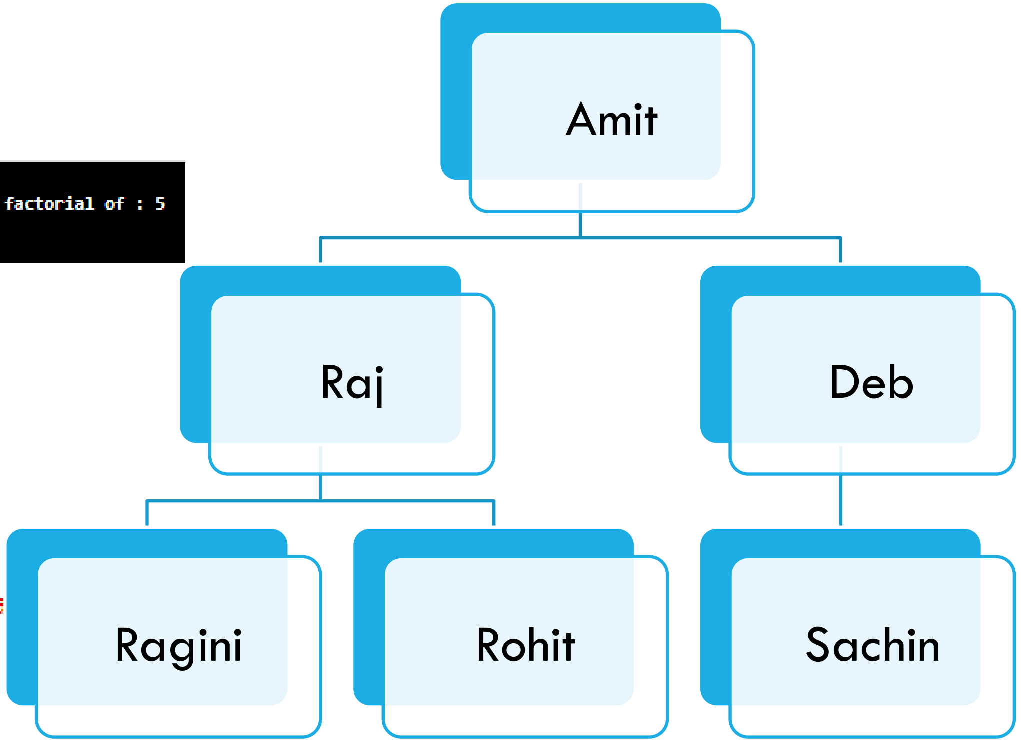


```
#include <iostream>

using namespace std;

long long int factorial(int num){
    if(num == 0){
        return 1;
    }
    return num * factorial(num-1);
}
int main() {
    int num;
    cout<<"Please enter the number you want the factoria
    cin>>num;
    cout<<endl;
    cout<<num<<"! = "<<factorial(num)<<endl;
    return 0;
}
```

Result

compiled and executed in 6.915 sec(s)

```
Please enter the number you want the factorial of : 5

5! = 120
```



(Business Organization Chart)