1a. Describe the difference between a stack and a queue.

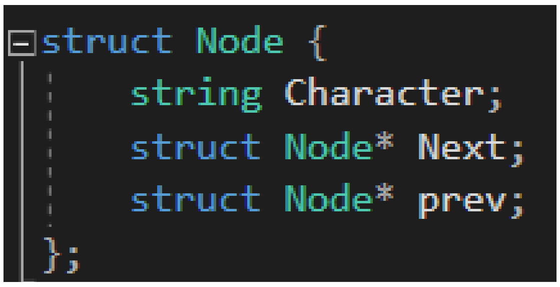
- **Both are Linear data structures, whose principal difference is whose elements are inserted or deleted in its structure. A stack follows the LIFO(Last In First Out) principle, while a queue follows the FIFO(First In First Out) principle.**

1b. Describe a typical programming use for a queue.

- **Queues can have different uses, such as a web page that receives requests to solve a problem and each of these must be organized to be attended in order of arrival. Or even, if I think of a video game, I relate those that are like "guitar hero", "beat saber" and the like, where the player must interact with the elements ("musical notes") with an order that works like a queue.**

2a. Use C/C++ to show the structure of an element in a doubly-linked list in which each element holds a single character. Describe briefly what any pointers point to.

- **In a doubly-linked list, each element can be represent like a node:**

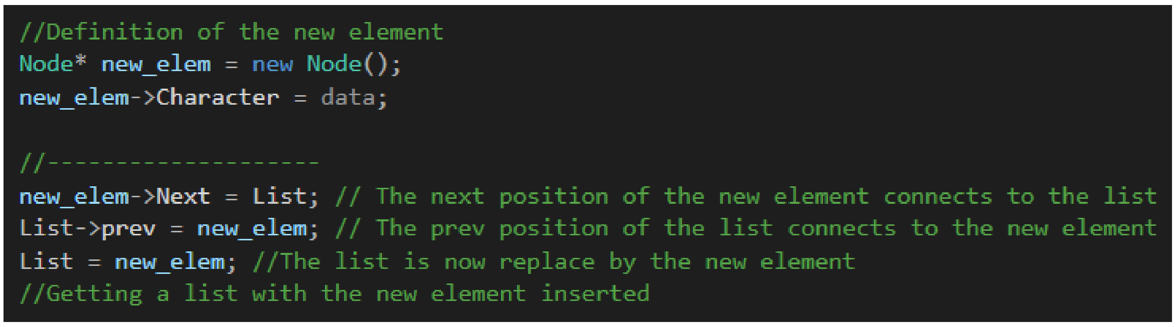


```
struct Node {
    string Character;
    struct Node* Next;
    struct Node* prev;
};
```

Each Node in the list is connected by the pointers, in this case the variables "Next" and "prev". "Next" points to the next node in its current position in the list or NULL, and "prev" points to the previous node in the current position in the list or NULL.

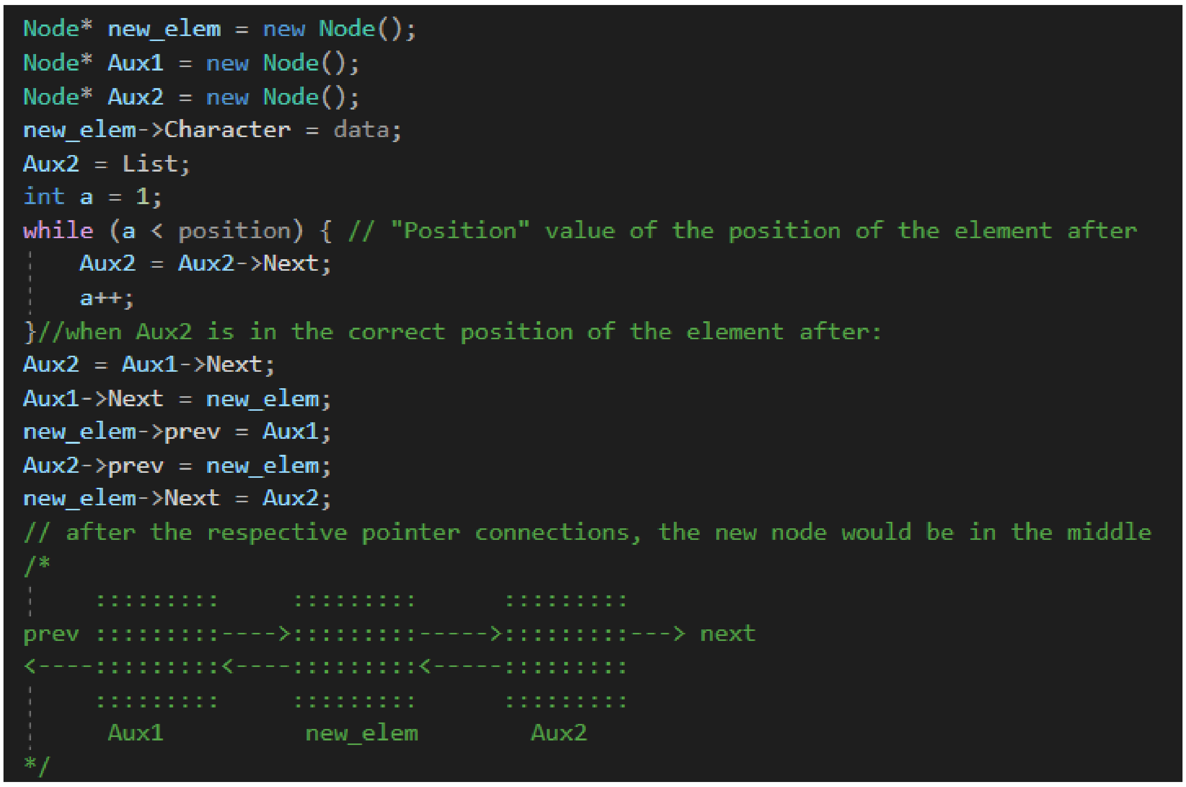
2b. Use C/C++ to show the operations necessary to insert a list element given a pointer to the new element and a pointer to the element after which the new element needs to be inserted.

if the element after is at the beginning of the list



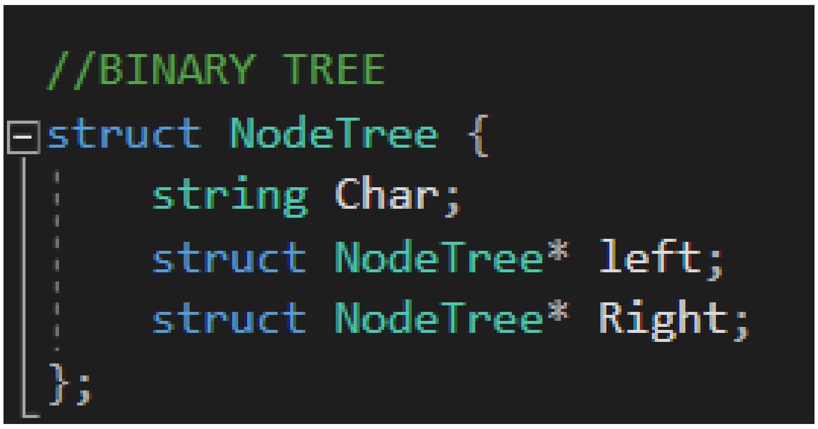
```
//Definition of the new element
Node* new_elem = new Node();
new_elem->Character = data;

//--------------------
new_elem->Next = List; // The next position of the new element connects to the list
List->prev = new_elem; // The prev position of the list connects to the new element
List = new_elem; //The list is now replace by the new element
//Getting a list with the new element inserted
```

If the element after is in a random position(i assume that i knew the position of the element after):



```
Node* new_elem = new Node();
Node* Aux1 = new Node();
Node* Aux2 = new Node();
new_elem->Character = data;
Aux2 = List;
int a = 1;
while (a < position) { // "Position" value of the position of the element after
    Aux2 = Aux2->Next;
    a++;
}//when Aux2 is in the correct position of the element after:
Aux2 = Aux1->Next;
Aux1->Next = new_elem;
new_elem->prev = Aux1;
Aux2->prev = new_elem;
new_elem->Next = Aux2;
// after the respective pointer connections, the new node would be in the middle
/*
     :::::::::     :::::::::      :::::::::
prev :::::::::---->:::::::::----->:::::::::---> next
<----:::::::::<----:::::::::<-----:::::::::
     :::::::::     :::::::::      :::::::::
      Aux1          new_elem        Aux2
*/
```

3a. Use C/C++ to show the structure of an element of a sorted, binary tree in which each element contains a single character. Describe briefly what any pointers point to.

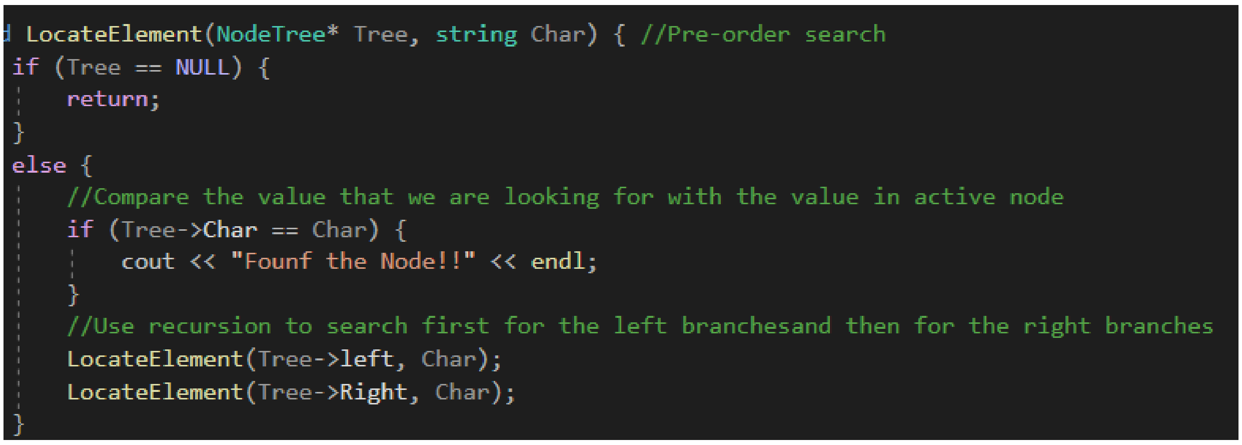
- 

```
//BINARY TREE
struct NodeTree {
    string Char;
    struct NodeTree* left;
    struct NodeTree* Right;
};
```

  In the case of the Binary Trees, each node can have at most 2 children, and the pointers can be identified like “Left” that point to the “left child” and “right” that point to the “right child”. This pointer could be point to NULL too.

3b. Use C/C++ to show the operations necessary to locate the element in the sorted binary tree which matches a given character.

- **For this exercise i used a “pre order” search method:**



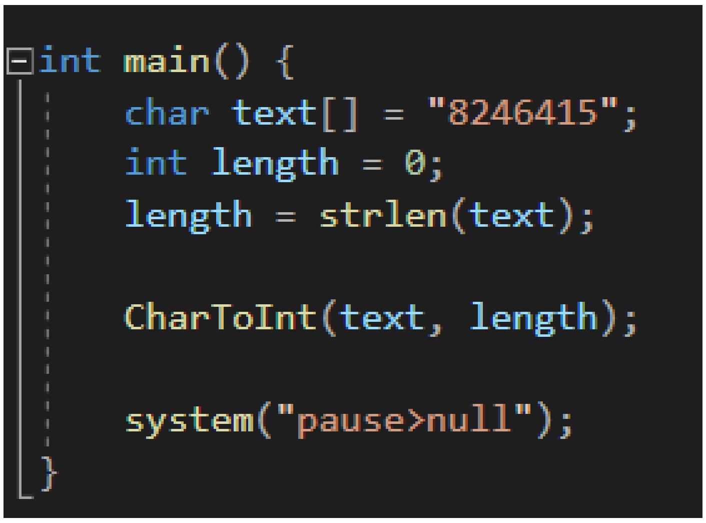
```
d LocateElement(NodeTree* Tree, string Char) { //Pre-order search
 if (Tree == NULL) {
     return;
 }
 else {
     //Compare the value that we are looking for with the value in active node
     if (Tree->Char == Char) {
         cout << "Founf the Node!!" << endl;
     }
     //Use recursion to search first for the left branchesand then for the right branches
     LocateElement(Tree->left, Char);
     LocateElement(Tree->Right, Char);
 }
```

4. Compare the following function and macro definitions. In what cases will they produce different results and/or side effects?
int square(int val) { return val*val; }
#define square(val) (val*val)

- **In the function we can see that the variables are restricted with type int, so in this caso if we use a decimal number, different results will be obtained.**

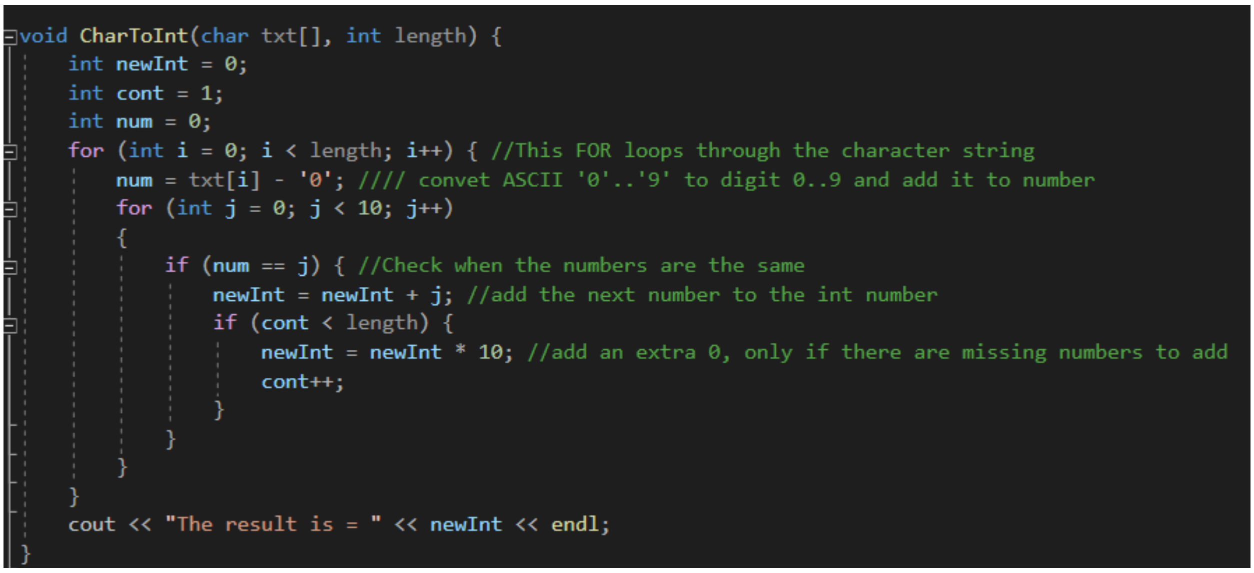
5. Write a C function that converts a character string into a signed integer without using any library functions. Assume that the string contains a valid integer and no white space.

- **First, how the main function was done:**



```
int main() {
    char text[] = "8246415";
    int length = 0;
    length = strlen(text);

    CharToInt(text, length);

    system("pause>null");
}
```

- **and then the function in charge to convert the char to int:**



```cpp
void CharToInt(char txt[], int length) {
    int newInt = 0;
    int cont = 1;
    int num = 0;
    for (int i = 0; i < length; i++) { //This FOR loops through the character string
        num = txt[i] - '0'; //// convet ASCII '0'..'9' to digit 0..9 and add it to number
        for (int j = 0; j < 10; j++)
        {
            if (num == j) { //Check when the numbers are the same
                newInt = newInt + j; //add the next number to the int number
                if (cont < length) {
                    newInt = newInt * 10; //add an extra 0, only if there are missing numbers to add
                    cont++;
                }
            }
        }
    }
    cout << "The result is = " << newInt << endl;
}
```

6a. Given a 2D vector A in the x-y plane of length |A| and angle theta to the x-axis, give the equations for the x and y components of A.

- **Ay = |A|*sin(theta) ← Y component of A**
- **Ax = |A|*cos(theta) ← X component of A**

6b. Given the x,y,z components of a 3D vector A, give the equation for the angle between the vector and the x-y plane.

- **Angle = 90° - cos^-1(|A * Nπ| / (|A| * |Nπ|))**
- **Nπ → Normal Vector to A vector**

7a. Given a 3D point starting at position P1 and moving with constant velocity vector V, write an equation for position P2 of the point after elapsed time t.

- **(P2 - P1) = V * t; → P2 = P1 + V*t;**

7b. Given a 3D point starting at position P1 and moving with initial velocity vector V and constant acceleration vector A, write an equation for the position P2 of the point after elapsed time t.

- **V = Vo + A*t;**
- **(ds/dt) = v**
- **integrate ds = integrate(Vo + A*t)dt**
- **P2 = P1 + Vo*t + (A*t^2)/2**