**STACK**

**The Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).**

**Implementation Stack using Arrays**

**bool Stack::push(int x)**

**{**

**if (top >= (MAX-1))**

**{**

**cout << "Stack Overflow";**

**return false;**

**}**

**else**

**{**

**a[++top] = x;**

**cout<<x <<" pushed into stack\n";**

**return true;**

**}**

**}**

**int Stack::pop()**

**{**

**if (top < 0)**

**{**

**cout << "Stack Underflow";**

**return 0;**

**}**

**else**

**{**

**int x = a[top--];**

**return x;**

**}**

**====================================================**

**Stack in C++ STL**

Some Basic functions of Stack class in C++:

* **empty()** – Returns whether the stack is empty.
* **size()** – Returns the size of the stack.
* **top()** – Returns a reference to the topmost element of the stack.
* **push(g)** – Adds the element ‘g’ at the top of the stack.
* **pop()** – Deletes the topmost element of the stack.*.*

**void stackshow(stack<int> s){**

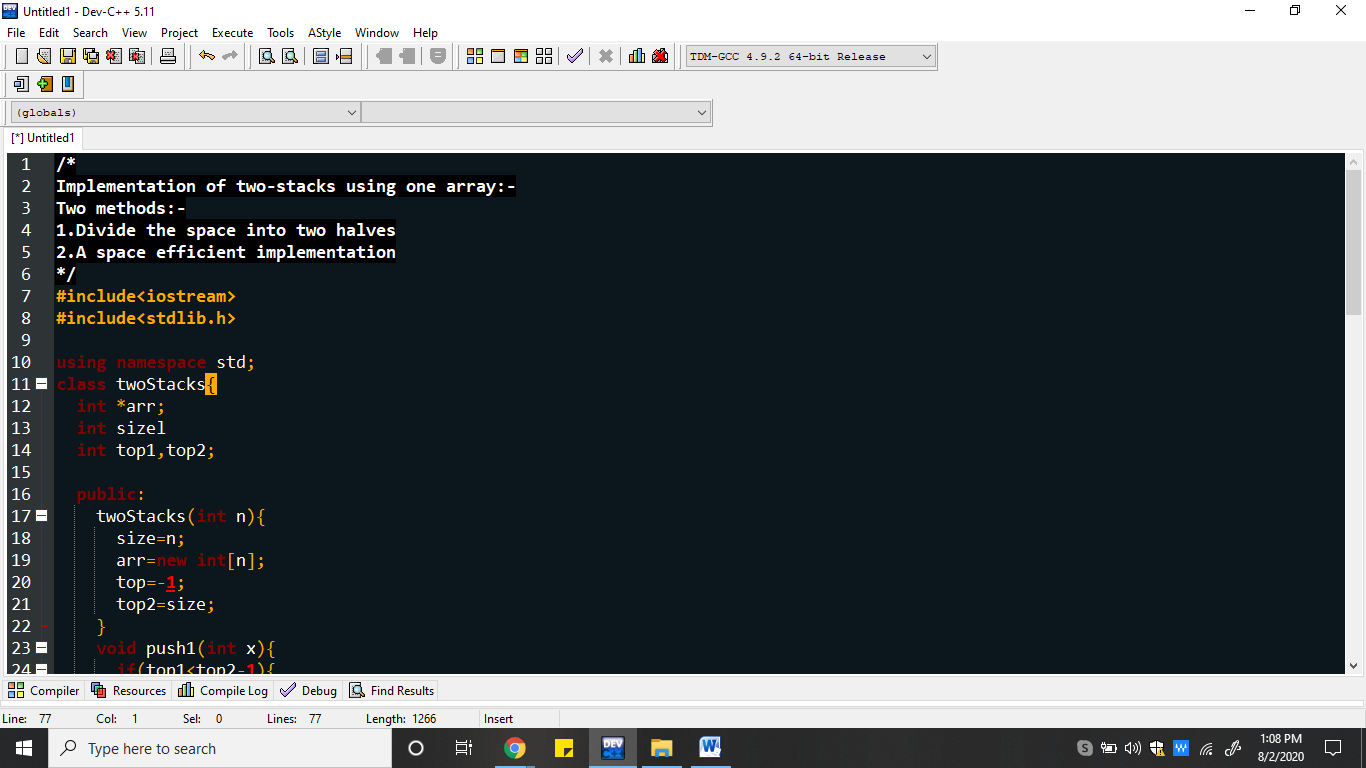
**while(!s.empty()){**

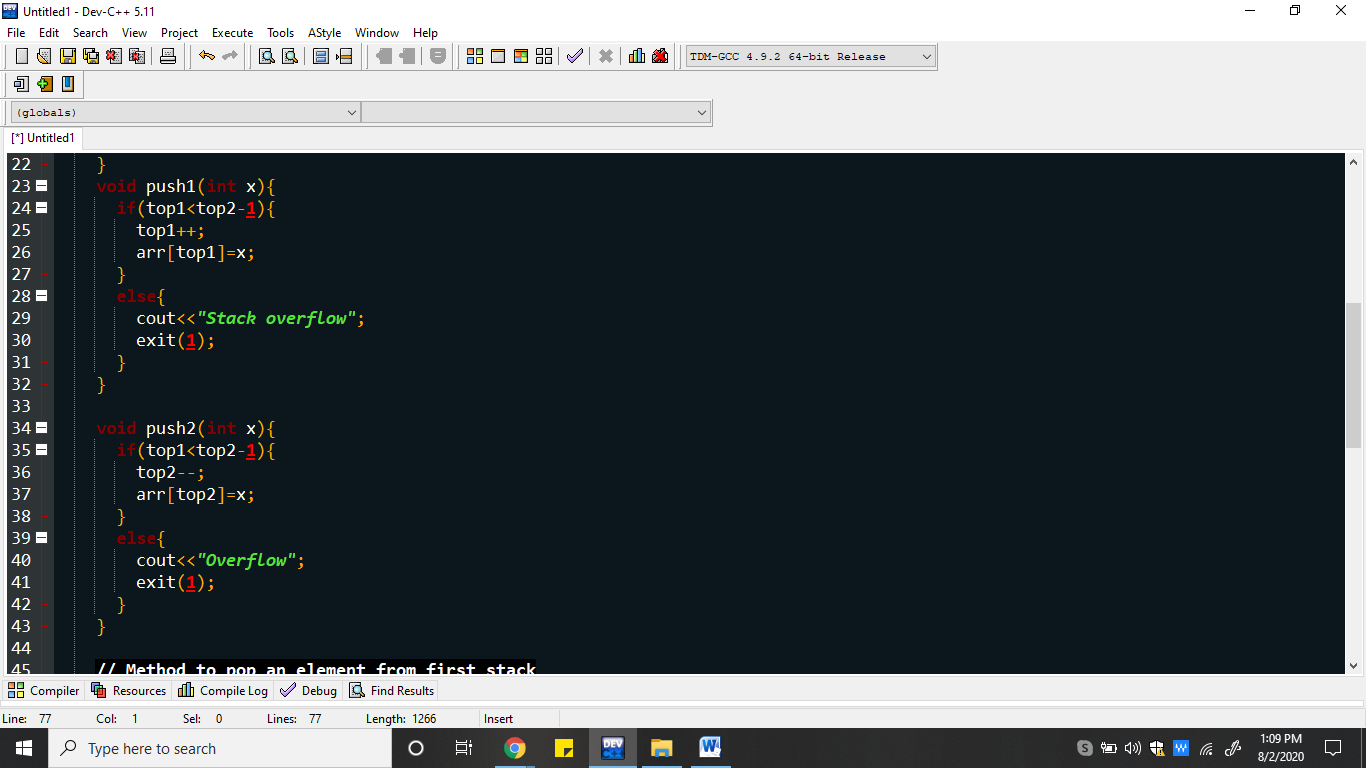
**cout<<”\t”<<s.top();**

**s.pop();}**

**====================================================**

**Implementation of two-stacks:-**





**Problem #1 : Print Reverse of linked List using Stack.**

**Description - We are given a linked list. We have to print the reverse of the linked list using Stack.**

**[1 2 3 4 5]**

**[5 4 3 2 1]**

**Solution - We will traverse the linked list and push all the nodes of the linked list to the stack. Since stack have property of Last In, First Out, List is automatically reversed when we will pop the stack elements one by one.**

**void printreverse(Node \*head)  
{  
 stack < Node\* > s  
 current = head  
 while(current != NULL)  
 {  
 s.push(current)  
 current = current->next  
 }  
 while( ! s.empty())  
 {  
 node = s.top()  
 print(node->data)  
 s.pop()  
 }**  
}

**====================================================**

### ****Problem #2 : Check for balanced parentheses in an expression****

### Description - Given an expression string exp , we have to check whether the pairs and the orders of { “ , ” } , ( “ , ” ) and [ “ , ” ] are correct in exp. For example -

### Input : [ ( ) ] { } { [ ( ) ( ) ] ( ) }

### Output : true

### Input : [ ( ] )

### Output : false