Priority Queue using Linked List

```cpp
// Aneesh Panchal
// 2K20/MC/21

#include<bits/stdc++.h>
using namespace std;

class Node {
public:
    int data;
    int priority;
    Node* next=NULL;
};

class PriorityQueueLinkedList {
    Node* head;

    public:
        PriorityQueueLinkedList(){head = NULL;}

    void enqueue(int data_,int prior){
        Node* newNode = new Node();
        newNode->data = data_;
        newNode->priority = prior;
        Node* temp=head;

        if(head == NULL){
            head = newNode;
            return;
        }

        else if(head->priority > prior){
            newNode->next = head;
            head = newNode;
            return;
        }

        while(temp->next != NULL && temp->next->priority <= prior){
            temp = temp->next;
        }
        newNode->next = temp->next;
        temp->next = newNode;
    }

    void show(){
        Node* temp=head;

        while(temp!=NULL){
            cout<< temp->data << "(" << temp->priority << ")" <<" -> ";
            temp = temp->next;
        }
        cout<<"NULL"<<endl<<endl;
```

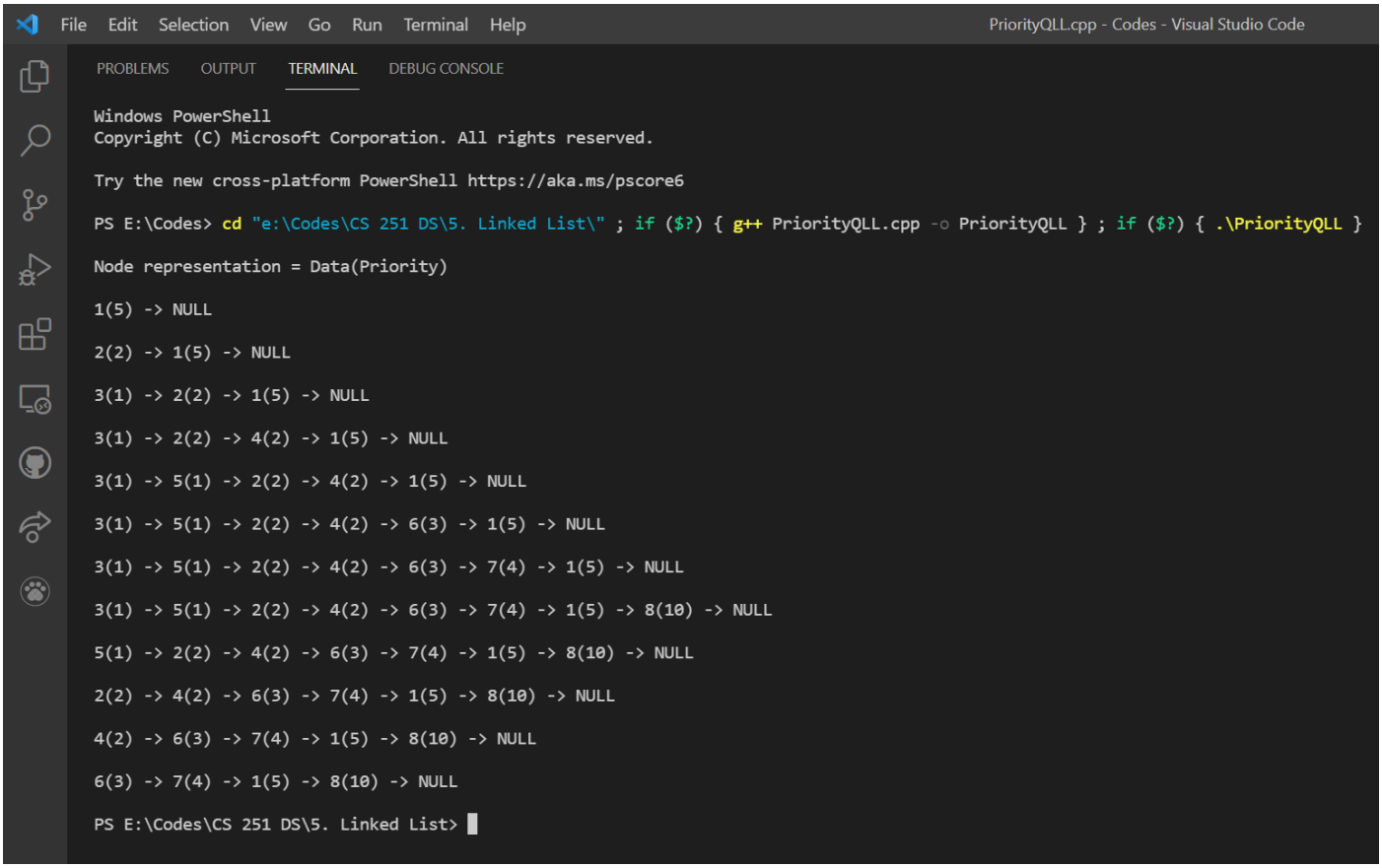
```
    }

    void dequeue(){
        Node* temp=head;

        if(head==NULL){
            cout<<"List empty"<<endl<<endl;
            return;
        }
        Node* del = head;
        head = temp->next;
        delete del;
    }
};

int main()
{
    PriorityQueueLinkedList PQLL;
    cout<<endl<<"Node representation = Data(Priority)"<<endl<<endl;
    PQLL.enqueue(1,5);
    PQLL.show();
    PQLL.enqueue(2,2);
    PQLL.show();
    PQLL.enqueue(3,1);
    PQLL.show();
    PQLL.enqueue(4,2);
    PQLL.show();
    PQLL.enqueue(5,1);
    PQLL.show();
    PQLL.enqueue(6,3);
    PQLL.show();
    PQLL.enqueue(7,4);
    PQLL.show();
    PQLL.enqueue(8,10);
    PQLL.show();

    PQLL.dequeue();
    PQLL.show();
    PQLL.dequeue();
    PQLL.show();
    PQLL.dequeue();
    PQLL.show();
    PQLL.dequeue();
    PQLL.show();
    return 0;
}
```


File Edit Selection View Go Run Terminal Help
PriorityQLL.cpp - Codes - Visual Studio Code
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS E:\Codes> cd "e:\Codes\CS 251 DS\5. Linked List\" ; if ($?) { g++ PriorityQLL.cpp -o PriorityQLL } ; if ($?) { .\PriorityQLL }
Node representation = Data(Priority)
1(5) -> NULL
2(2) -> 1(5) -> NULL
3(1) -> 2(2) -> 1(5) -> NULL
3(1) -> 2(2) -> 4(2) -> 1(5) -> NULL
3(1) -> 5(1) -> 2(2) -> 4(2) -> 1(5) -> NULL
3(1) -> 5(1) -> 2(2) -> 4(2) -> 6(3) -> 1(5) -> NULL
3(1) -> 5(1) -> 2(2) -> 4(2) -> 6(3) -> 7(4) -> 1(5) -> NULL
3(1) -> 5(1) -> 2(2) -> 4(2) -> 6(3) -> 7(4) -> 1(5) -> 8(10) -> NULL
5(1) -> 2(2) -> 4(2) -> 6(3) -> 7(4) -> 1(5) -> 8(10) -> NULL
2(2) -> 4(2) -> 6(3) -> 7(4) -> 1(5) -> 8(10) -> NULL
4(2) -> 6(3) -> 7(4) -> 1(5) -> 8(10) -> NULL
6(3) -> 7(4) -> 1(5) -> 8(10) -> NULL
PS E:\Codes\CS 251 DS\5. Linked List>

Recursive Tree Traversals

```cpp
// Aneesh Panchal
// 2K20/MC/21

#include<bits/stdc++.h>
using namespace std;

class Node {
public:
    int data;
    Node* left=NULL;
    Node* right=NULL;
};

void inorder(Node *Root){
    if(Root==NULL){
        return;
    }
    inorder(Root->left);
    cout<<Root->data<<" ";
    inorder(Root->right);
}

void preorder(Node *Root){
    if(Root==NULL){
        return;
    }
    cout<<Root->data<<" ";
    preorder(Root->left);
    preorder(Root->right);
}

void postorder(Node *Root){
    if(Root==NULL){
        return;
    }
    postorder(Root->left);
    postorder(Root->right);
    cout<<Root->data<<" ";
}

int main(){
    Node *n1 = new Node;
    Node *n2 = new Node;
    Node *n3 = new Node;
    Node *n4 = new Node;
    Node *n5 = new Node;
    Node *n6 = new Node;
    Node *n7 = new Node;
    Node *n8 = new Node;

    n1->data = 1;
```

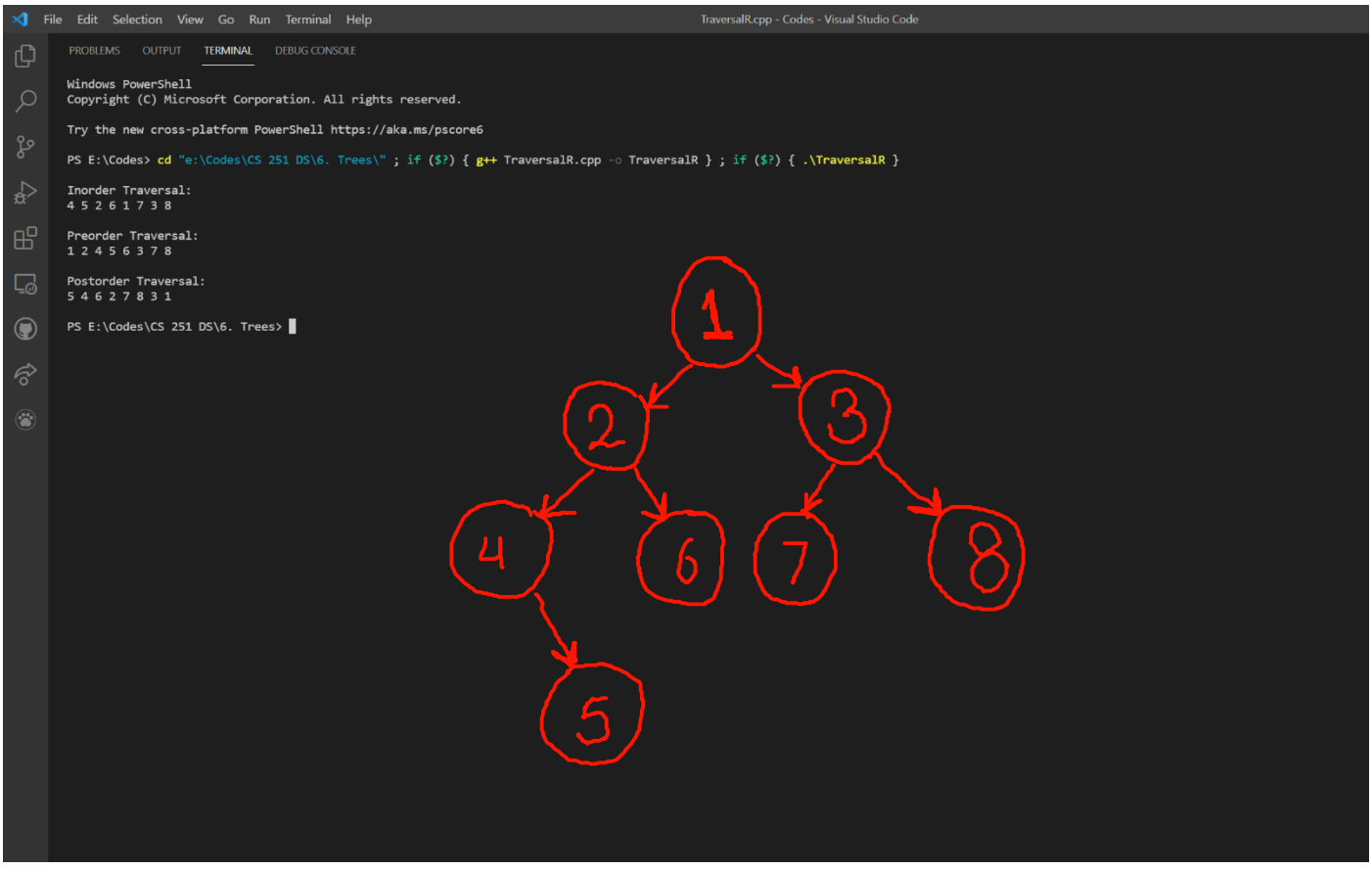
```
    n2->data = 2;
    n3->data = 3;
    n4->data = 4;
    n5->data = 5;
    n6->data = 6;
    n7->data = 7;
    n8->data = 8;

    n1->left = n2; n1->right = n3;
    n2->left = n4; n2->right = n6;
    n4->right = n5;
    n3->left = n7; n3->right = n8;

    cout<<endl<<"Inorder Traversal: "<<endl;
    inorder(n1);
    cout<<endl;

    cout<<endl<<"Preorder Traversal: "<<endl;
    preorder(n1);
    cout<<endl;

    cout<<endl<<"Postorder Traversal: "<<endl;
    postorder(n1);
    cout<<endl<<endl;
    return 0 ;
}
```



Non Recursive Tree Traversals

```cpp
// Aneesh Panchal
// 2K20/MC/21

#include<bits/stdc++.h>
using namespace std;

class Node {
public:
    int data;
    Node* left=NULL;
    Node* right=NULL;
};

void inorder(Node *Root){
    stack<Node*> nodestack;
    Node *curr = Root;

    while (curr != NULL || nodestack.empty() == false){
        while (curr !=  NULL){
            nodestack.push(curr);
            curr = curr->left;
        }

        curr = nodestack.top();
        nodestack.pop();

        cout << curr->data << " ";
        curr = curr->right;
    }
}

void preorder(Node *Root){
    if (Root == NULL)
        return;

    stack<Node*> nodeStack;
    nodeStack.push(Root);

    while (nodeStack.empty() == false) {
        Node* node = nodeStack.top();
        printf("%d ", node->data);
        nodeStack.pop();

        if (node->right!=NULL)
            nodeStack.push(node->right);
        if (node->left!=NULL)
            nodeStack.push(node->left);
    }
}

void postorder(Node *Root){
```

```
    if (Root == NULL)
        return;

    stack<Node*> nodeStack;
    nodeStack.push(Root);
    stack<int> out;

    while (!nodeStack.empty()){
        Node* curr = nodeStack.top();
        nodeStack.pop();
        out.push(curr->data);

        if (curr->left!=NULL)
            nodeStack.push(curr->left);
        if (curr->right!=NULL)
            nodeStack.push(curr->right);
    }

    while (!out.empty()){
        cout << out.top() << " ";
        out.pop();
    }
}

int main(){
    Node *n1 = new Node;
    Node *n2 = new Node;
    Node *n3 = new Node;
    Node *n4 = new Node;
    Node *n5 = new Node;
    Node *n6 = new Node;
    Node *n7 = new Node;
    Node *n8 = new Node;

    n1->data = 1;
    n2->data = 2;
    n3->data = 3;
    n4->data = 4;
    n5->data = 5;
    n6->data = 6;
    n7->data = 7;
    n8->data = 8;

    n1->left = n2; n1->right = n3;
    n2->left = n4; n2->right = n6;
    n4->right = n5;
    n3->left = n7; n3->right = n8;

    cout<<endl<<"Inorder Traversal: "<<endl;
    inorder(n1);
    cout<<endl;

    cout<<endl<<"Preorder Traversal: "<<endl;
    preorder(n1);
```

```
    cout<<endl;

    cout<<endl<<"Postorder Traversal: "<<endl;
    postorder(n1);
    cout<<endl<<endl;
    return 0 ;
}
```