Assign5a:

```
//Comments were inherited form the sample code
//I essentially just changed it into a templated format
//Template doesn't work for things such as strings and chars

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <climits> // For INT_MIN

using namespace std;

template <typename item> //template
class linkedlist
{

public:

   struct stack
   {
      item data;
      struct stack* next;
   };

   int size = 0;

   //Function to push a new element in stack.
   void push(item element)
   {
```

```
    //Check stack overflow
    if (size >= CAPACITY)
    {
       cout << "Stack Overflow, can't add more element to stack.\n";
       return;
    }

    //Create a new node and push to stack
    stack* newNode = new stack;

    //Assign data to new node in stack
    newNode->data = element;

    //Next element after new node should be current top element
    newNode->next = top;

    //Make sure new node is always at top
    top = newNode;

    //Increase element count in stack
    size++;

    cout << "Data pushed to stack.\n";

}

//Function to pop element from top of stack.

item pop()
```

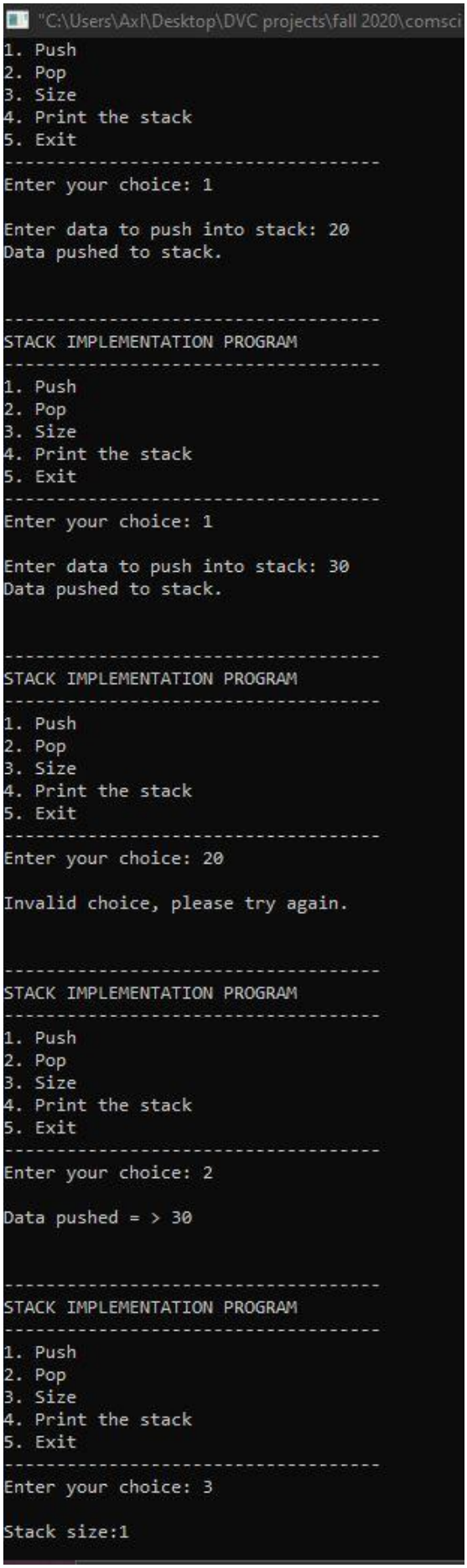
```
{
    item data = 0;
    stack* topNode;

    //Check stack underflow
    if (size <= 0 || !top)
    {
        cout << "Stack is empty. \n";

        //Throw empty stack error/exception
        //Since C does not have concept of exception
        //Hence return minimum integer values as error value
        //Later in code check if return value is INT_MIN, then
        //stack is empty

        return INT_MIN;
    }

    //Copy reference of stack top to some temp variable
    //Since we need to delete current stack top and make
    //stack top its next element
    topNode = top;

    //Copy data from stack's top element
    data = top->data;

    //Move top to its next element
    top = top->next;
```

```
        //Delete the previous top most stack element from memory
        free(topNode);

        //Decrement stack size
        size--;

        return data;
    }

    void display()
    {

        stack* temp;
        for(temp = top; temp != NULL; temp = temp->next)
            cout << temp->data << " ";
        cout << endl;

    }

private:
    //private definitions
    stack *top =  NULL;
    int CAPACITY = 1000;

};



int main()
```

```
{
   //Type definition can only work with definitions such as
   //int, float, double, long, short
   linkedlist<int> item;
   int choice, data;

   while(1)
   {
      //Menu
      cout << "-------------------------------------\n";
      cout << "STACK IMPLEMENTATION PROGRAM\n";
      cout << "-------------------------------------\n";
      cout << "1. Push\n";
      cout << "2. Pop\n";
      cout << "3. Size\n";
      cout << "4. Print the stack\n";
      cout << "5. Exit\n";
      cout << "-------------------------------------\n";
      cout << "Enter your choice: ";

      cin >> choice;
      cout << endl;

      switch(choice) //Added overall template definitions (as seen from the item.)
      {
      case 1:
         cout << "Enter data to push into stack: ";
         cin >> data;
```

```
        //Push element to stack
        item.push(data);
        break;

    case 2:
        data = item.pop();
        //if stack is not empty
        if (data != INT_MIN)
            cout << "Data pushed = > " << data << endl;
        break;

    case 3:
        cout << "Stack size:" << item.size << endl;
        break;

    case 4:
        item.display();
        break;

    case 5:
        cout << "Exiting from app.\n" << endl;
        exit(0);
        break;

    default:
        cout << "Invalid choice, please try again.\n";

    }
    cout << "\n\n";
```

```
    }
    return 0;

}
```


"C:\Users\Axl\Desktop\DVC projects\fall 2020\comsci
1. Push
2. Pop
3. Size
4. Print the stack
5. Exit
Enter your choice: 1
Enter data to push into stack: 20
Data pushed to stack.
STACK IMPLEMENTATION PROGRAM
1. Push
2. Pop
3. Size
4. Print the stack
5. Exit
Enter your choice: 1
Enter data to push into stack: 30
Data pushed to stack.
STACK IMPLEMENTATION PROGRAM
1. Push
2. Pop
3. Size
4. Print the stack
5. Exit
Enter your choice: 20
Invalid choice, please try again.
STACK IMPLEMENTATION PROGRAM
1. Push
2. Pop
3. Size
4. Print the stack
5. Exit
Enter your choice: 2
Data pushed = > 30
STACK IMPLEMENTATION PROGRAM
1. Push
2. Pop
3. Size
4. Print the stack
5. Exit
Enter your choice: 3
Stack size:1

Assign 5b:

```
/**
Much like the previous case form Assignment 5a, alot of the comments are kept form the actual and sample code as
what I did was merely modification of certain aspects
**/

#include <cstdio>
#include <cstdlib>
#include <climits>
#include <iostream>


#define CAPACITY 100 ///Queue max capacity

using namespace std;

/** Queue structure definition */
template<class item> ///templated structure definition
struct QueueType
{
public:
   item data;
   struct QueueType * next;
};

/** Queue size */
unsigned int size = 0;
template<class item>

///Class definition
```

```
class QType
{
   QueueType<item> *front, *rear; ///definition for the front and rear pointers
   public:

      QType() ///constructor that appoints the pointers to a null value
      {
         rear = NULL;
         front = NULL;
      }

      int enqueue(int data)
      {
         QueueType<item> * newNode = NULL;
         if (isFull())
         {
            return 0;
         }
         newNode = new QueueType<item>;
         newNode->data = data;
         newNode->next = NULL;
         if ( (rear) )
         {
            rear->next = newNode;
         }
         rear = newNode;
         if ( !( front) )
         {
            front = rear;
         }
         size++;
```

```
    return 1;
}



int dequeue()
{
    QueueType<item> *toDequque = NULL;
    int data = INT_MIN;
    if (isEmpty())
    {
        return INT_MIN;
    }
    toDequque = front;
    data = toDequque->data;
    front = (front)->next;
    size--;
    free(toDequque);
    return data;
}

int getRear()
{
return (isEmpty())
    ? INT_MIN
    : rear->data;
}

int getFront()
{
// Return INT_MIN if queue is empty otherwise front.
return (isEmpty())
```

```
    ? INT_MIN
    : front->data;
}



/**
* Checks, if queue is empty or not.
*/
int isEmpty()
{
    return (size <= 0);
}



/**
* Checks, if queue is within the maximum queue capacity.
*/
int isFull()
{
    return (size > CAPACITY);
}
string prepMenu()
{
    string menu = "";

    menu+= " \n----------------------------------------------------------------\n";
    menu+= "1.Enqueue 2.Dequeue 3.Size 4.Get Rear 5.Get Front 6.Display 7.Exit\n";
    menu+= "----------------------------------------------------------------\n";
    menu+= "Select an option: ";
    return menu;
}
```

```
    void display()
    {
       for ( QueueType<item> *t = front; t !=NULL; t = t->next)
       cout <<t->data << " ";
       cout << endl << endl;
    }
};

/**
The only changes to the main function between the sample code and this one is that
I have removed the pointer definition for rear and front and moved them to the class
The functions containing rear and front have also had the two removed as it is now part of the class up there
hence not needing any reason to be called again from main.
Functions are also now added with the header call of item. to ensure that they are being called from the class
**/
int main()
{
   int option, data;

   QType<int> item;

   string menu = item.prepMenu();
   cout << menu << endl;
   cin >> option;
   while (option !=7)
   {

      switch (option)
      {
         case 1:
```

```
cout << "\nEnter data to enqueue (-99 to stop): ";
cin >> data;
while (data != -99)
{
   /// Enqueue function returns 1 on success
   /// otherwise 0
   if (item.enqueue(data))
   cout << "Element added to queue.";
   else
   cout << "Queue is full." << endl;
   cout << "\nEnter data to enqueue (-99 to stop): ";
   cin >> data;
}



break;


case 2:
data = item.dequeue();

/// on success dequeue returns element removed
/// otherwise returns INT_MIN
if (data == INT_MIN)
cout << "Queue is empty."<< endl;
else
cout << "Data => " << data << endl;

break;


case 3:
```

```
/// isEmpty() function returns 1 if queue is emtpy
/// otherwise returns 0
if (item.isEmpty())
cout << "Queue is empty."<< endl;
else
cout << "Queue size => "<< size << endl;

break;

case 4:
data = item.getRear();

if (data == INT_MIN)
cout << "Queue is empty." << endl;
else
cout << "Rear => " << data << endl;

break;

case 5:

data = item.getFront();

if (data == INT_MIN)
cout <<"Queue is empty."<< endl;
else
cout <<"Front => " << data << endl;

break;

case 6:
```

```
            item.display();
            break;

            default:
            cout <<"Invalid choice, please input number between (0-5).\n";
            break;
        }

        cout <<"\n\n";
        cout << menu<< endl;
        cin >> option;
    }
}
```

```
-----------------------------------------------------------------
1.Enqueue 2.Dequeue 3.Size 4.Get Rear 5.Get Front 6.Display 7.Exit
-----------------------------------------------------------------
Select an option:
1

Enter data to enqueue (-99 to stop): 100
Element added to queue.
Enter data to enqueue (-99 to stop): 1
Element added to queue.
Enter data to enqueue (-99 to stop): 30
Element added to queue.
Enter data to enqueue (-99 to stop): 1
Element added to queue.
Enter data to enqueue (-99 to stop): 40
Element added to queue.
Enter data to enqueue (-99 to stop): 2
Element added to queue.
Enter data to enqueue (-99 to stop): -99



-----------------------------------------------------------------
1.Enqueue 2.Dequeue 3.Size 4.Get Rear 5.Get Front 6.Display 7.Exit
-----------------------------------------------------------------
Select an option:
2
Data => 100



-----------------------------------------------------------------
1.Enqueue 2.Dequeue 3.Size 4.Get Rear 5.Get Front 6.Display 7.Exit
-----------------------------------------------------------------
Select an option:
3
Queue size => 5



-----------------------------------------------------------------
1.Enqueue 2.Dequeue 3.Size 4.Get Rear 5.Get Front 6.Display 7.Exit
-----------------------------------------------------------------
Select an option:
4
Rear => 2



-----------------------------------------------------------------
1.Enqueue 2.Dequeue 3.Size 4.Get Rear 5.Get Front 6.Display 7.Exit
-----------------------------------------------------------------
Select an option:
5
Front => 1



-----------------------------------------------------------------
1.Enqueue 2.Dequeue 3.Size 4.Get Rear 5.Get Front 6.Display 7.Exit
-----------------------------------------------------------------
Select an option:
6
1 30 1 40 2
```