Quinn Roemer

CISP - 430

Assignment 4

2/22 /2018

Program 4.0 - CircList Implementation

Description:

The goal for this part of the assignment was to get some code working in my compiler that the professor provided for us. This proved to be a simple matter of copying what he wrote. This code takes an array and implements it as a circular array. It supports operations to add or remove data as well as find specific data.

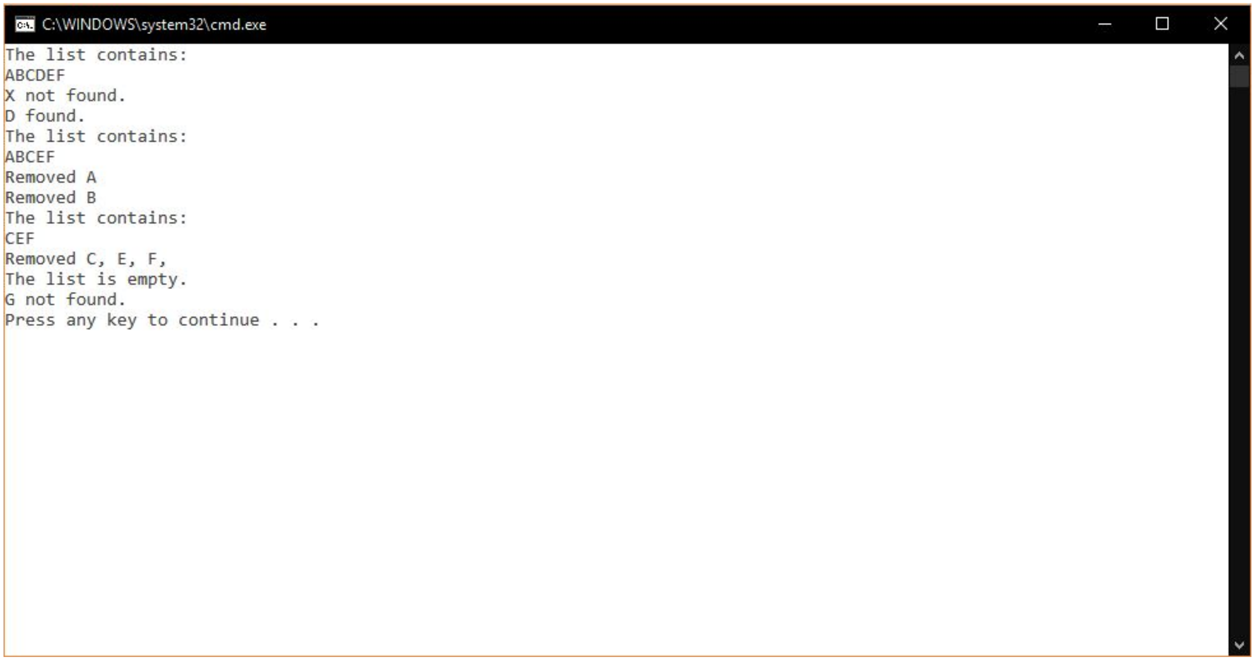
Source Code:

```
//Code written by Professor Ross.
#include <iostream>

using namespace std;

//The List
#define SIZE 10
char myList[SIZE];
int head, tail, used;

//Function Declarations.
char remove(void);
void append(char);
int find(char);
void traverse(void);
int isEmpty(void);

//Main function to execute.
void main(void)
{
   //Initialization.
   head = tail = used = 0;

   append('A');
   append('B');
   append('C');
   append('D');
   append('E');
   append('F');
   traverse();

   find('X');
   find('D');
   traverse();
   cout << "Removed " << remove() << endl;
   cout << "Removed " << remove() << endl;
   traverse();

   //Empty the list.
   cout << "Removed ";

   while (!isEmpty())
   {
             cout << remove() << ", ";
   }

   cout << endl;
   traverse();

   find('G');
}

//Receives a data element and appends it to the tail of the list.
void append(char d)
{
   //If list is empty.
   if (!used)
   {
             myList[tail] = d;
             used++;
             return;
   }

   //Prevent Overflow.
   if ((tail + 1) % SIZE == head)
   {
             cout << "Overflow. Element not appended.\n";
             return;
   }

   //Append data.
   tail = (tail + 1) % SIZE;
   myList[tail] = d;
   used++;
}

//Traverses the list from the head to the tail and prints out each data element.
void traverse(void)
{
   //Pointer
   char p;
   //Empty list
   if (isEmpty())
   {
            cout << "The list is empty.\n";
            return;
   }

   //1 element.
   if (used == 1)
   {
            cout << "The list contains: " << myList[head] << endl;
            return;
   }

   //More than 1 element.
   p = head;
   cout << "The list contains: \n";
   do
   {
            cout << myList[p];
            p = (p + 1) % SIZE;
   } while (p != (tail + 1) % SIZE);

   cout << endl;
}

//Returns true if the list is empty, returns false otherwise.
int isEmpty(void)
{
   if (used)
   {
            return 0;
   }
   else
   {
            return 1;
   }
}

//Removes a data element from the head of the list and returns it.
//Returns -1 if the list is empty.
char remove(void)
{
   char temp;

   //Empty list.
   if (isEmpty())
   {
```

```
            return -1;
   }
   //1 element.
   if (used == 1)
   {
            used = 0;
            return myList[head];
   }
   //More than 1 element.
   //Remove data.
   temp = myList[head];
   head = (head + 1) % SIZE;
   used--;
   return temp;
}

//Searches the list for a data element.
//If the data is found, removes the data element and returns 1.
//If the data is not found, returns 0;
int find(char d)
{
   //Pointer
   int p;

   //Empty.
   if (isEmpty())
   {
            cout << d << " not found." << endl;
            return 0;
   }

   //1 element.
   if (used == 1)
   {
            if (myList[head] == d)
            {
                        used = 0;
                        cout << d << " found." << endl;
                        return 1;
            }
            else
            {
                        cout << d << " not found." << endl;
                        return 0;
            }
   }

   //More than 1 element.
```

```
    p = head;
    do
    {
        if (myList[p] == d)
        {
            while (p != tail)
            {
                myList[p] = myList[(p + 1) % SIZE];
                p = (p + 1) % SIZE;
            }
            tail--;
            if (tail < 0)
            {
                tail = SIZE - 1;
            }
            used--;
            cout << d << " found." << endl;
            return 1;
        }
        p = (p + 1) % SIZE;
    } while (p != (tail + 1) % SIZE);

    cout << d << " not found." << endl;
    return 0;
}
```

Output:



```
The list contains:
ABCDEF
X not found.
D found.
The list contains:
ABCEF
Removed A
Removed B
The list contains:
CEF
Removed C, E, F,
The list is empty.
G not found.
Press any key to continue . . .
```

Program 4.1 - Linked List Implementation

Description:

The goal for this part of the assignment was to get some code working in my compiler that the professor provided for us. This proved to be a simple matter of copying what he wrote. This code takes a node structure and implements it as a list of linked nodes. It supports operations to add or remove data as well as find specific data.

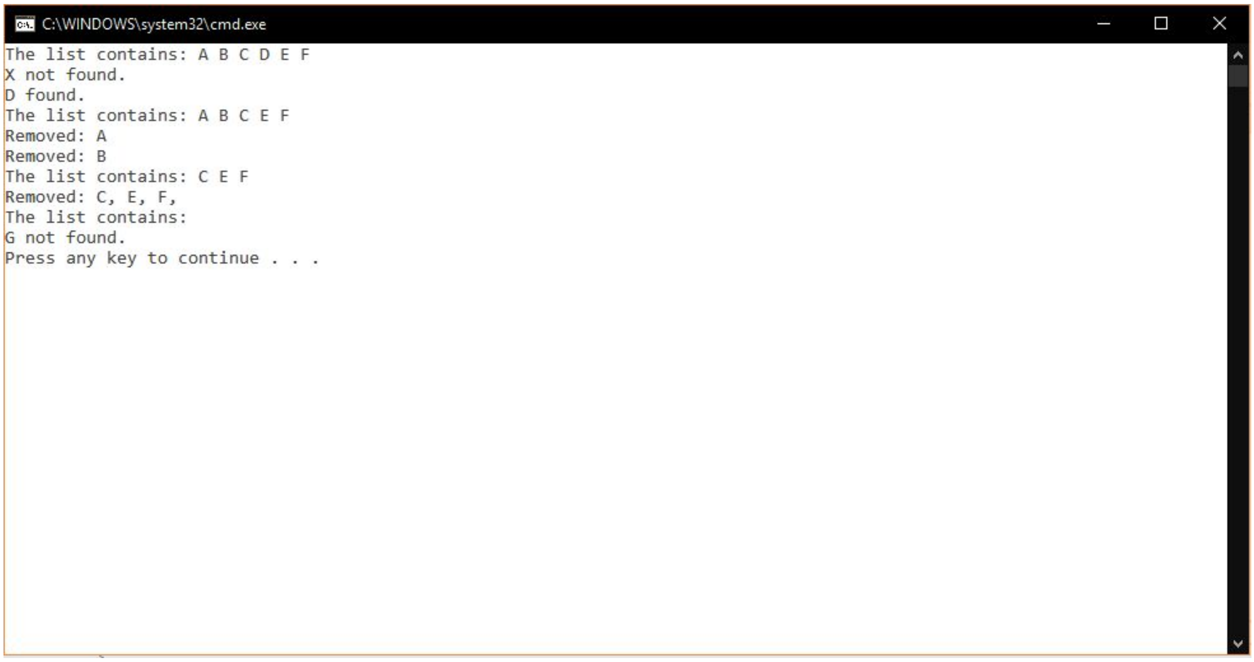
Source Code:

```
//Code written by Professor Ross.
#include <iostream>

using namespace std;

//Our node.
struct node {
   node* next;
   char d;
};

//Head and tail pointers.
node* head = 0;
node* tail = 0;

//Function declarations.
char remove(void);
void append(char);
int find(char);
void traverse(void);
int isEmpty(void);

//Main function to execute.
void main(void)
{
   append('A');
   append('B');
   append('C');
   append('D');
   append('E');
   append('F');

   traverse();

   find('X');
   find('D');
   traverse();

   cout << "Removed: " << remove() << endl;
   cout << "Removed: " << remove() << endl;
   traverse();

   //Empty the list.
   cout << "Removed: ";
   while (!isEmpty())
   {
          cout << remove() << ", ";
   }

   cout << endl;

   traverse();

   find('G');
}

//Receives a char element and appends it to the tail of the list.
void append(char d)
{
   //Make a new node.
   node* p = new node;
   p->next = 0;
   p->d = d;

   //List is empty.
   if (!head)
   {
          head = tail = p;
   }

   //Append to the tail end.
   else
   {
          tail->next = p;
          tail = p;
   }
}

//Traverses the list from the head to the tail, and prints out each char element.
void traverse(void)
{
   node* p = head;
   cout << "The list contains: ";
   while (p)
   {
            cout << p->d << " ";
            p = p->next;
   }
   cout << endl;
}

//Returns true if the list is empty, returns false otherwise.
int isEmpty(void)
{
   if (head)
   {
            return 0;
   }
   else
   {
            return 1;
   }
}

//Removes a char element from the head of the list and returns it.
//Returns -1 if the list is empty.
char remove(void)
{
   node* p;
   char temp;

   //Return null if the list is empty.
   if (!head)
   {
            return -1;
   }

   //One node.
   if (head == tail)
   {
            //Remove and destroy head node.
            temp = head->d;
            delete head;
            head = tail = 0;
            return temp;
   }

   //More than one node. Remove and destroy head node.
   p = head;
   head = head->next;
```

```
  temp = p->d;
  delete p;
  return temp;
}

//Searches the list for a char element.
//If the char is found, removes the char element and returns 1. Otherwise returns 0.
int find(char d)
{
  node* c;
  node* pc;

  //List is empty.
  if (!head)
  {
        cout << d << " not found." << endl;
        return 0;
  }

  //One node.
  if (head == tail)
  {
        if (head->d == d)
        {
                delete head;
                head = tail = 0;
                cout << d << " found." << endl;
                return 1;
        }
        else
        {
                cout << d << " not found." << endl;
                return 0;
        }
  }

  //Two or more nodes.
  pc = head;
  c = head->next;

  //Found at the head.
  if (pc->d == d)
  {
        head = head->next;
        delete pc;
        cout << d << " found." << endl;
        return 1;
  }
```

```
    //Look at nodes after the head node.
    while (c)
    {
        //Found it after head node.
        if (c->d == d)
        {
            pc->next = c->next;
            if (c == tail)
            {
                tail = pc;
            }

            delete c;
            cout << d << " found." << endl;
            return 1;
        }
        pc = c;
        c = c->next;
    }

    cout << d << " not found." << endl;
    return 0;
}
```

Output:



```
The list contains: A B C D E F
X not found.
D found.
The list contains: A B C E F
Removed: A
Removed: B
The list contains: C E F
Removed: C, E, F,
The list contains:
G not found.
Press any key to continue . . .
```

Program 4.2 - CircList Stack

Description:

In this section of the assignment I needed to take some previously written code and convert it to act like a stack. This meant adding functions for the pop, push, and peek operations. Note, I have removed all of the unnecessary functions to save room on this report.

Source Code:

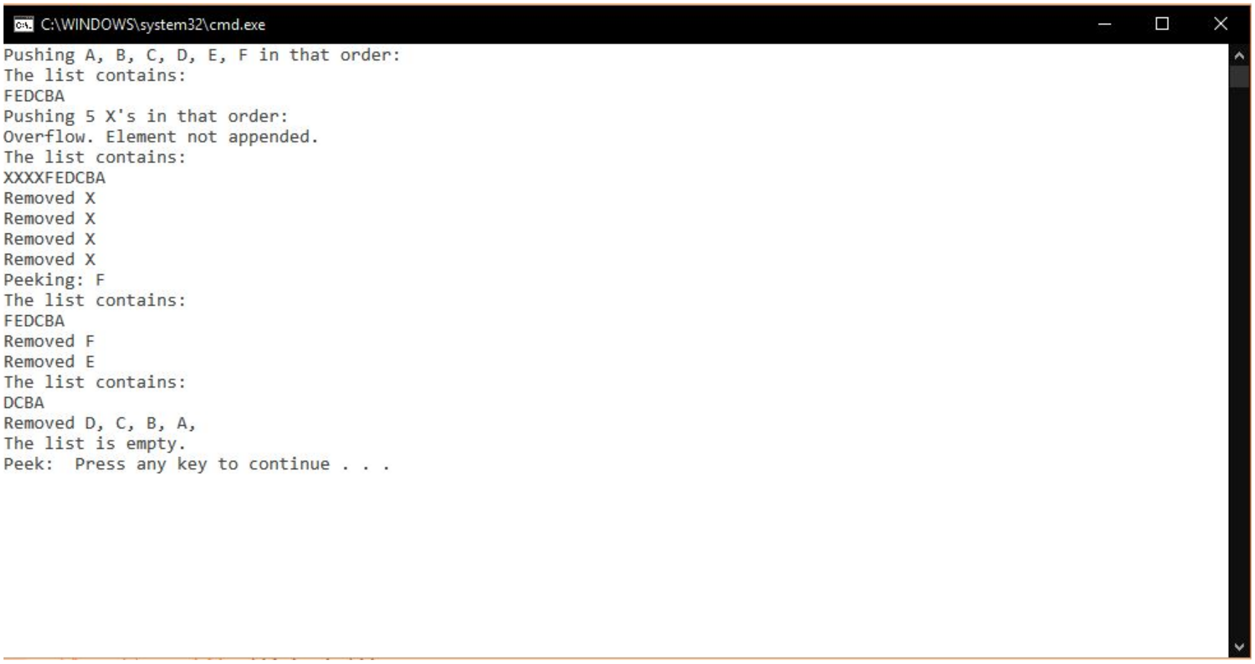
```
//Code written by Professor Ross. Modified by Quinn Roemer
#include <iostream>
using namespace std;

//The List
#define SIZE 10
char myList[SIZE];
int head, tail, used;

//Function Declarations.
void push(char);
void traverse(void);
int isEmpty(void);
char pop(void);
char peek(void);

//Main function to execute.
void main(void)
{
   //Initialization.
   head = tail = used = 0;
   cout << "Pushing A, B, C, D, E, F in that order:" << endl;
   push('A');
   push('B');
   push('C');
   push('D');
   push('E');
   push('F');
   traverse();
   cout << "Pushing 5 X's in that order:" << endl;
   push('X');
   push('X');
   push('X');
   push('X');
   push('X');
   traverse();
```

```
    cout << "Removed " << pop() << endl;
    cout << "Removed " << pop() << endl;
    cout << "Removed " << pop() << endl;
    cout << "Removed " << pop() << endl;
    cout << "Peeking: " << peek() << endl;
    traverse();
    cout << "Removed " << pop() << endl;
    cout << "Removed " << pop() << endl;
    traverse();
    //Empty the list.
    cout << "Removed ";
    while (!isEmpty())
    {
            cout << pop() << ", ";
    }
    cout << endl;
    traverse();
    cout << "Peek: " << peek();
}
//Receives a data element and appends it to the head of the list.
void push(char d)
{
    //If list is empty.
    if (!used)
    {
            myList[head] = d;
            used++;
            return;
    }
    //Prevent Overflow.
    if ((head + 1) % SIZE == tail)
    {
            cout << "Overflow. Element not appended.\n";
            return;
    }
    //Append data.
    head = (head + 1) % SIZE;
    myList[head] = d;
    used++;
}
//Traverses the list from the head to the tail and prints out each data element.
void traverse(void)
{
    //Pointer
    char p;

    //Empty list
    if (isEmpty())
```

```
  {
          cout << "The list is empty.\n";
          return;
  }
  //1 element.
  if (used == 1)
  {
          cout << "The list contains: " << myList[head] << endl;
          return;
  }

  //More than 1 element.
  p = head;
  cout << "The list contains: \n";
  do
  {
          cout << myList[p];
          p = (p - 1) % SIZE;
  } while (p != (tail - 1) % SIZE);

  cout << endl;
}
//Returns true if the list is empty, returns false otherwise.
int isEmpty(void)
{
  if (used)
  {
          return 0;
  }
  else
  {
          return 1;
  }
}
//Returns the top item on the stack and deletes it. Returns -1 if empty.
char pop(void)
{
  if (isEmpty())
  {
          return -1;
  }

  char temp = myList[head];
  head--;
  used--;
  return temp;
}
//Returns the top item on the stack. Returns -1 if empty.
```

```
char peek(void)
{
   if (isEmpty())
   {
            return -1;
   }
   else
   {
            return myList[head];
   }
}
```

Output/Testing:



```
Pushing A, B, C, D, E, F in that order:
The list contains:
FEDCBA
Pushing 5 X's in that order:
Overflow. Element not appended.
The list contains:
XXXXFEDCBA
Removed X
Removed X
Removed X
Removed X
Peeking: F
The list contains:
FEDCBA
Removed F
Removed E
The list contains:
DCBA
Removed D, C, B, A,
The list is empty.
Peek:  Press any key to continue . . .
```

BigO of peek():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of push():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of pop():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of isEmpty():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

Program 4.3 - CircList Queue

Description:

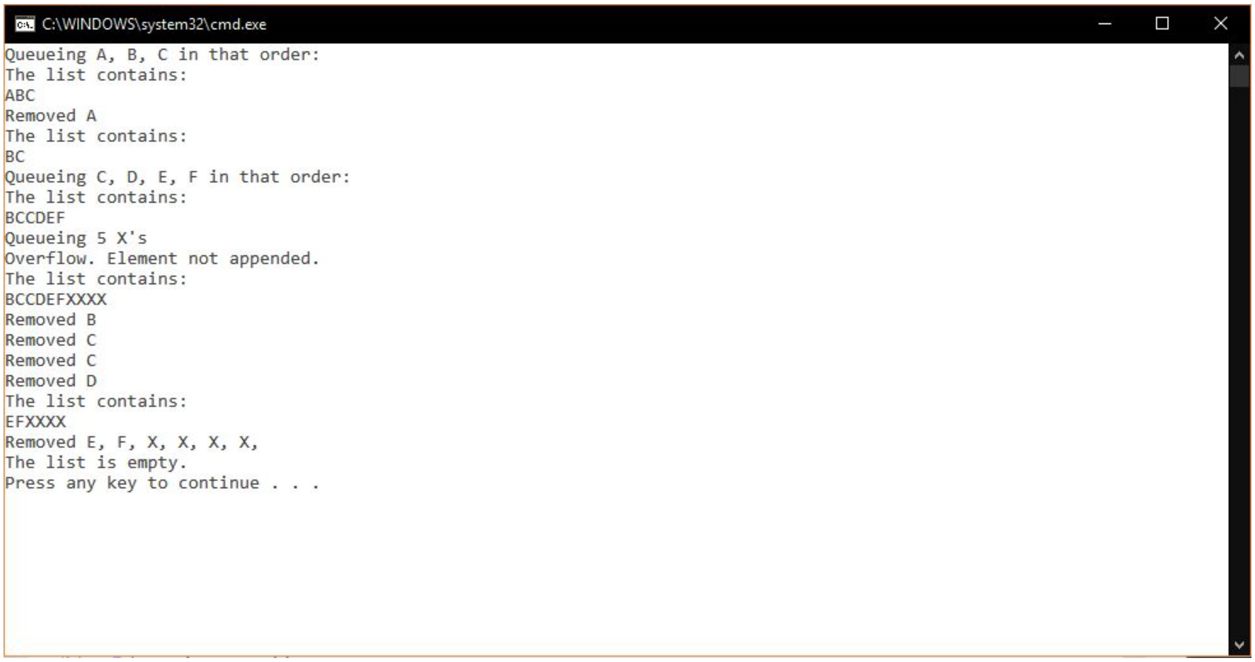
In this section of the assignment I needed to take some previously written code and convert it to act like a queue. This meant adding functions for the queue, and dequeue operations. Note, I have removed all of the unnecessary functions to save room on this report.

Source Code:

```
//Code written by Professor Ross. Modified by Quinn Roemer
#include <iostream>
using namespace std;

//The List
#define SIZE 10
char myList[SIZE];
int head, tail, used;

//Function Declarations.
void q(char);
void traverse(void);
bool isEmpty(void);
char dq(void);

//Main function to execute.
void main(void)
{
  //Initialization.
  head = tail = used = 0;
  cout << "Queueing A, B, C in that order:" << endl;
  q('A');
  q('B');
  q('C');
  traverse();
  cout << "Removed " << dq() << endl;
  traverse();
  cout << "Queueing C, D, E, F in that order:" << endl;
  q('C');
  q('D');
  q('E');
  q('F');
  traverse();
  cout << "Queueing 5 X's" << endl;
  q('X');
  q('X');
```

```
   q('X');
   q('X');
   q('X');
   traverse();
   cout << "Removed " << dq() << endl;
   cout << "Removed " << dq() << endl;
   cout << "Removed " << dq() << endl;
   cout << "Removed " << dq() << endl;
   traverse();
   //Empty the list.
   cout << "Removed ";
   while (!isEmpty())
   {
            cout << dq() << ", ";
   }
   cout << endl;
   traverse();
}
//Receives a data element and appends it to the tail of the list.
void q(char d)
{
   //If list is empty.
   if (!used)
   {
            myList[head] = d;
            used++;
            return;
   }
   //Prevent Overflow.
   if ((head + 1) % SIZE == tail)
   {
            cout << "Overflow. Element not appended.\n";
            return;
   }
   //Append data.
   head = (head + 1) % SIZE;
   myList[head] = d;
   used++;
}
//Traverses the list from the head to the tail and prints out each data element.
void traverse(void)
{
   //Pointer
   char p;

   //Empty list
   if (isEmpty())
   {
          cout << "The list is empty.\n";
          return;
   }

   //1 element.
   if (used == 1)
   {
          cout << "The list contains: " << myList[head] << endl;
          return;
   }
   //More than 1 element.
   p = tail;
   cout << "The list contains: \n";
   do
   {
          cout << myList[p];
          p = (p + 1) % SIZE;
   } while (p != (head + 1) % SIZE);

   cout << endl;
}
//Returns true if the list is empty, returns false otherwise.
bool isEmpty(void)
{
   if (!used)
   {
          return true;
   }
   else
   {
          return false;
   }
}
//Returns the tail element and removes it from the list. Returns -1 if empty.
char dq(void)
{
   char p = tail;
   char temp = myList[tail];
   if (isEmpty())
   {
          return -1;
   }

   if (used == 1)
   {
          used--;
          return temp;
   }
   while (p != head)
   {
            myList[p] = myList[p + 1];
            p++;
   }

   head--;
   used--;
   return temp;
}
```

Output/Testing:



```
Queueing A, B, C in that order:
The list contains:
ABC
Removed A
The list contains:
BC
Queueing C, D, E, F in that order:
The list contains:
BCCDEF
Queueing 5 X's
Overflow. Element not appended.
The list contains:
BCCDEFXXXX
Removed B
Removed C
Removed C
Removed D
The list contains:
EFXXXX
Removed E, F, X, X, X, X,
The list is empty.
Press any key to continue . . .
```

BigO of q():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of dq():

This function contains one loop. During a worst case scenario the Big0 of this function would be O(N). Meaning that as the list gets bigger the time it takes for this function to complete its operation increases as well.

BigO of isEmpty():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

Program 4.4 - CircList Priority Queue

Description:

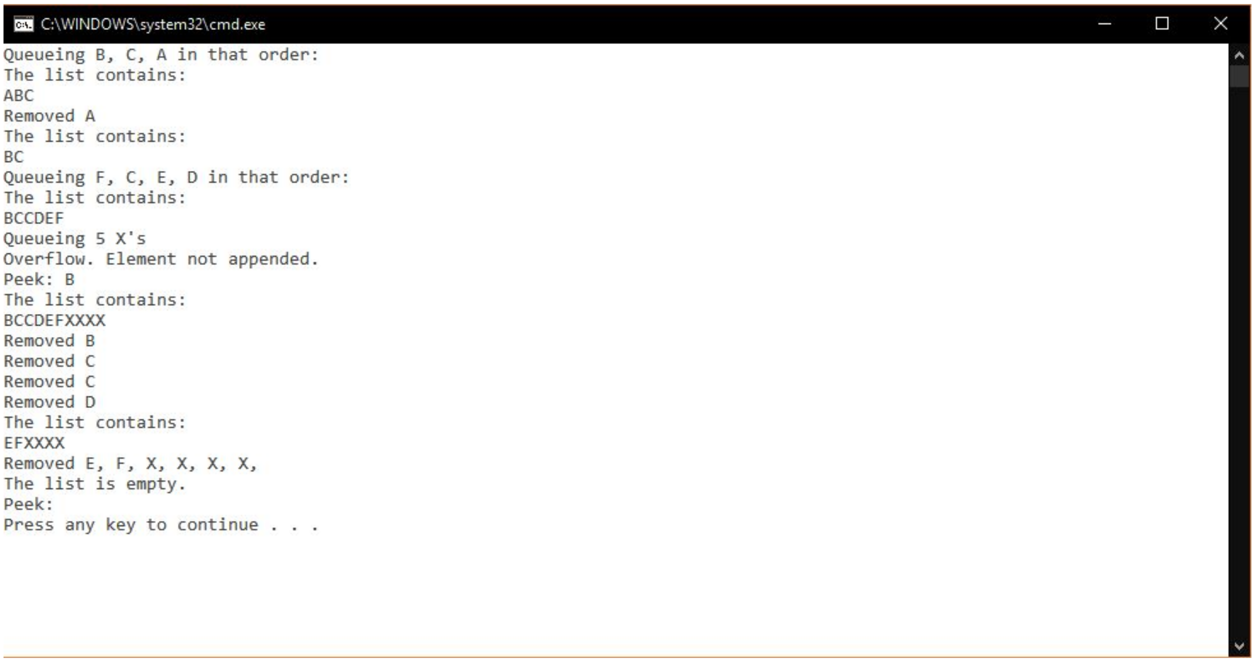
In this section of the assignment I needed to take some previously written code and convert it to act like a priority queue. This meant adding functions for the insert, peek, and dequeue operations. I personally had trouble figuring this one out. I eventually decided to use a Bubble Sort after the insert operation. This is by no means the most efficient method to achieve the task. Note, I have removed all of the unnecessary functions to save room on this report.

Source Code:

```
//Code written by Professor Ross. Modified by Quinn Roemer.
#include <iostream>
using namespace std;

//The List
#define SIZE 10
char myList[SIZE];
int head, tail, used;

//Function Declarations.
void insert(char);
void traverse(void);
bool isEmpty(void);
char dq(void);
char peek(void);

//Main function to execute.
void main(void)
{
   //Initialization.
   head = tail = used = 0;
   cout << "Queueing B, C, A in that order:" << endl;
   insert('B');
   insert('C');
   insert('A');
   traverse();
   cout << "Removed " << dq() << endl;
   traverse();
   cout << "Queueing F, C, E, D in that order:" << endl;
   insert('F');
   insert('C');
   insert('E');
   insert('D');
   traverse();
   cout << "Queueing 5 X's" << endl;
   insert('X');
   insert('X');
   insert('X');
   insert('X');
   insert('X');
   cout << "Peek: " << peek() << endl;
   traverse();
   cout << "Removed " << dq() << endl;
   cout << "Removed " << dq() << endl;
   cout << "Removed " << dq() << endl;
   cout << "Removed " << dq() << endl;
   traverse();
   //Empty the list.
   cout << "Removed ";
   while (!isEmpty())
   {
            cout << dq() << ", ";
   }
   cout << endl;
   traverse();
   cout << "Peek: " << peek() << endl;
}
//Receives a data element and appends it to the tail of the list.
void insert(char d)
{
   char temp;
   //If list is empty.
   if (!used)
   {
            myList[head] = d;
            used++;
            return;
   }
   //Prevent Overflow.
   if ((head + 1) % SIZE == tail)
   {
            cout << "Overflow. Element not appended.\n";
            return;
   }
   //Append data.
   head = (head + 1) % SIZE;
   myList[head] = d;
   used++;

   //Sort data
   for (int i = 0; i < head; i++)
   {
        for (int j = 0; j < head; j++)
        {
                if (myList[j] >= myList[j + 1])
                {
                        temp = myList[j + 1];
                        myList[j + 1] = myList[j];
                        myList[j] = temp;
                }
        }
  }
}
//Traverses the list from the head to the tail and prints out each data element.
void traverse(void)
{
  //Pointer
  char p;

  //Empty list
  if (isEmpty())
  {
        cout << "The list is empty.\n";
        return;
  }
  //1 element.
  if (used == 1)
  {
        cout << "The list contains: " << myList[head] << endl;
        return;
  }
  //More than 1 element.
  p = tail;
  cout << "The list contains: \n";
  do
  {     cout << myList[p];
        p = (p + 1) % SIZE;
  } while (p != (head + 1) % SIZE);

  cout << endl;
}
//Returns true if the list is empty, returns false otherwise.
bool isEmpty(void)
{
  if (!used)
  {
        return true;
  }
  else
  {
```

```
            return false;
   }
}
//Returns the tail element and returns it. Returns -1 if empty.
char dq(void)
{
   char p = tail;
   char temp = myList[tail];

   if (isEmpty())
   {
            return -1;
   }
   if (used == 1)
   {
            used--;
            return temp;
   }
   while (p != head)
   {
            myList[p] = myList[p + 1];
            p++;
   }
   head--;
   used--;
   return temp;
}
//Returns the tail element.
char peek(void)
{
   if (isEmpty())
   {
            return -1;
   }
   else
   {
            return myList[tail];
   }
}
```

Output/Testing:



```
Queueing B, C, A in that order:
The list contains:
ABC
Removed A
The list contains:
BC
Queueing F, C, E, D in that order:
The list contains:
BCCDEF
Queueing 5 X's
Overflow. Element not appended.
Peek: B
The list contains:
BCCDEFXXXX
Removed B
Removed C
Removed C
Removed D
The list contains:
EFXXXX
Removed E, F, X, X, X, X,
The list is empty.
Peek:
Press any key to continue . . .
```

BigO of insert():

This function contains two nested for loops. According to our iterative rule of thumb the BigO for this function is O(N^2). This means that the time that this function takes to complete a task would increase exponentially as the list grows larger. This is by no means the most efficient implementation of this function.

BigO of dq():

This function contains one loop. During a worst case scenario the Big0 of this function would be O(N). Meaning that as the list gets bigger the time it takes for this function to complete its operation increases as well.

BigO of isEmpty():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of peek():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

Program 4.5 - Linked List Stack

Description:

In this section of the assignment I needed to take some previously written code and convert it to act like a stack. This meant adding functions for the push, pop, and peek operations. Note, I have removed all of the unnecessary functions to save room on this report.

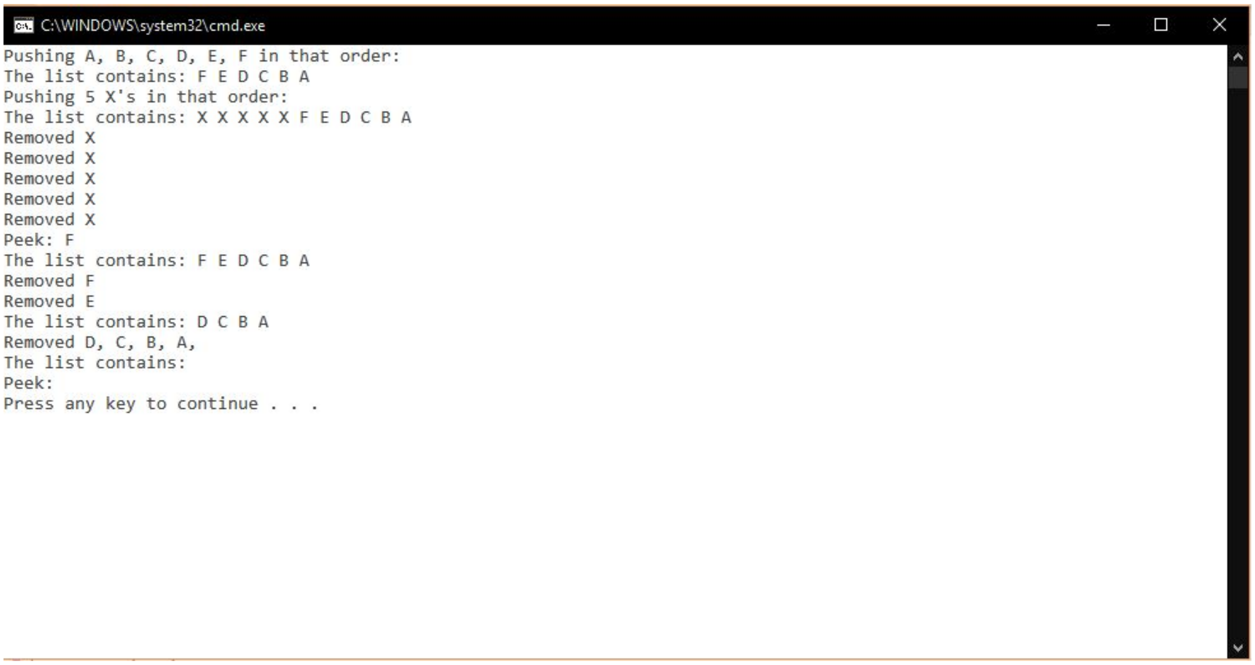
Source Code:

```
//Code written by Professor Ross. Modified by Quinn Roemer
#include <iostream>
using namespace std;

//Our node.
struct node {
        node* next;
        char d;
};

//Head and tail pointers.
node* head = 0;
node* tail = 0;

//Function declarations.
char pop(void);
void push(char);
void traverse(void);
bool isEmpty(void);
char peek(void);

//Main function to execute.
void main(void)
{
  cout << "Pushing A, B, C, D, E, F in that order:" << endl;
  push('A');
  push('B');
  push('C');
  push('D');
  push('E');
  push('F');
  traverse();
  cout << "Pushing 5 X's in that order:" << endl;
  push('X');
  push('X');
  push('X');
```

```
    push('X');
    push('X');
    traverse();
    cout << "Removed " << pop() << endl;
    cout << "Removed " << pop() << endl;
    cout << "Removed " << pop() << endl;
    cout << "Removed " << pop() << endl;
    cout << "Removed " << pop() << endl;
    cout << "Peek: " << peek() << endl;
    traverse();
    cout << "Removed " << pop() << endl;
    cout << "Removed " << pop() << endl;
    traverse();
    //Empty the list.
    cout << "Removed ";
    while (isEmpty())
    {
            cout << pop() << ", ";
    }
    cout << endl;
    traverse();
    cout << "Peek: " << peek() << endl;
}
//Receives a char element and appends it to the head of the list.
void push(char d)
{
        //Make a new node.
        node* p = new node;
        p->next = 0;
        p->d = d;

        //List is empty.
        if (!head)
        {
                head = tail = p;
        }

        //Append to the head end.
        else
        {
                p->next = head;
                head = p;
        }
}
//Traverses the list from the head to the tail, and prints out each char element.
void traverse(void)
{
        node* p = head;
    cout << "The list contains: ";
    while (p)
    {
        cout << p->d << " ";
        p = p->next;
    }
    cout << endl;
}
//Returns true if the list is empty, returns false otherwise.
bool isEmpty(void)
{
    if (!head)
    {
        return false;
    }
    else
    {
        return true;
    }
}
//Removes a char element from the head of the list and returns it.
//Returns -1 if the list is empty.
char pop(void)
{
    node* p;
    char temp;

    //Return null of the list is empty.
    if (!head)
    {
        return -1;
    }
    //One node.
    if (head == tail)
    {
        //Remove and destroy head node.
        temp = head->d;
        delete head;
        head = tail = 0;
        return temp;
    }
    //More than one node. Remove and destroy head node.
    p = head;
    head = head->next;
    temp = p->d;
    delete p;
    return temp;
```

```
}
//Returns the head element. Returns -1 if empty.
char peek(void)
{
        //List is empty.
        if (!head)
        {
                return -1;
        }
        else
        {
                return head->d;
        }
}
```

Output/Testing:



```
Pushing A, B, C, D, E, F in that order:
The list contains: F E D C B A
Pushing 5 X's in that order:
The list contains: X X X X X F E D C B A
Removed X
Removed X
Removed X
Removed X
Removed X
Peek: F
The list contains: F E D C B A
Removed F
Removed E
The list contains: D C B A
Removed D, C, B, A,
The list contains:
Peek:
Press any key to continue . . .
```

BigO of push():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of pop():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of peek():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of isEmpty():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

Program 4.6 - Linked List Queue

Description:

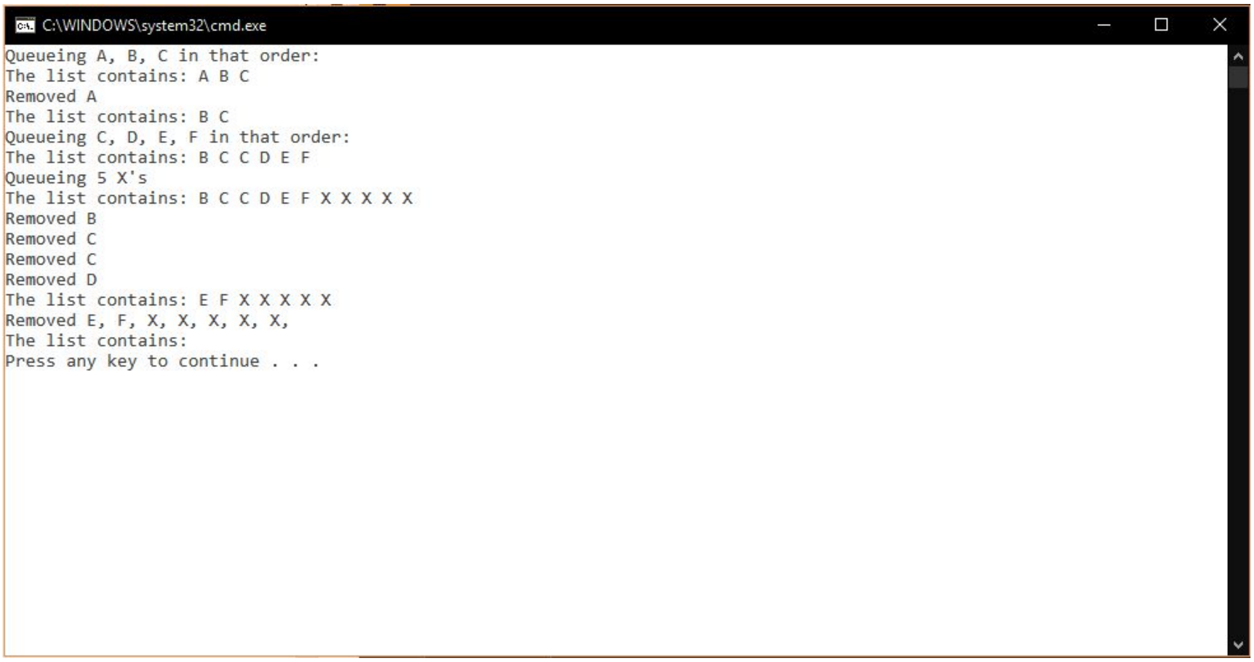
In this section of the assignment I needed to take some previously written code and convert it to act like a queue. This meant adding functions for the queue, and dequeue operations. Note, I have removed all of the unnecessary functions to save room on this report.

Source Code:

```
//Code written by Professor Ross. Modified by Quinn Roemer
#include <iostream>
using namespace std;

//Our node.
struct node {
   node* next;
   char d;
};
//Head and tail pointers.
node* head = 0;
node* tail = 0;

//Function declarations.
char dq(void);
void q(char);
void traverse(void);
bool isEmpty(void);

//Main function to execute.
void main(void)
{
   cout << "Queueing A, B, C in that order:" << endl;
   q('A');
   q('B');
   q('C');
   traverse();
   cout << "Removed " << dq() << endl;
   traverse();
   cout << "Queueing C, D, E, F in that order:" << endl;
   q('C');
   q('D');
   q('E');
   q('F');
   traverse();
   cout << "Queueing 5 X's" << endl;
   q('X');
   q('X');
   q('X');
   q('X');
   q('X');
   traverse();
   cout << "Removed " << dq() << endl;
   cout << "Removed " << dq() << endl;
   cout << "Removed " << dq() << endl;
   cout << "Removed " << dq() << endl;
   traverse();
   //Empty the list.
   cout << "Removed ";
   while (isEmpty())
   {
             cout << dq() << ", ";
   }
   cout << endl;
   traverse();
}
//Receives a char element and appends it to the tail of the list.
void q(char d)
{
   //Make a new node.
   node* p = new node;
   p->next = 0;
   p->d = d;

   //List is empty.
   if (!head)
   {
             head = tail = p;
   }
   //Append to the tail end.
   else
   {
             tail->next = p;
             tail = p;
   }
}
//Traverses the list from the head to the tail, and prints out each char element.
void traverse(void)
{
   node* p = head;

   cout << "The list contains: ";
   while (p)
   {
```

```
		cout << p->d << " ";
		p = p->next;
	}
	cout << endl;
}
//Returns true if the list is empty, returns false otherwise.
bool isEmpty(void)
{
	if (!head)
	{
		return false;
	}
	else
	{
		return true;
	}
}

//Removes a char element from the head of the list and returns it.
//Returns -1 if the list is empty.
char dq(void)
{
	node* p;
	char temp;

	//Return null of the list is empty.
	if (!head)
	{
		return -1;
	}
	//One node.
	if (head == tail)
	{
		//Remove and destroy head node.
		temp = head->d;
		delete head;
		head = tail = 0;
		return temp;
	}
	//More than one node. Remove and destroy head node.
	p = head;
	head = head->next;
	temp = p->d;
	delete p;
	return temp;
}
```

Output/Testing:



```
Queueing A, B, C in that order:
The list contains: A B C
Removed A
The list contains: B C
Queueing C, D, E, F in that order:
The list contains: B C C D E F
Queueing 5 X's
The list contains: B C C D E F X X X X X
Removed B
Removed C
Removed C
Removed D
The list contains: E F X X X X X
Removed E, F, X, X, X, X, X,
The list contains:
Press any key to continue . . .
```

BigO of q():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of dq():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of isEmpty():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

Program 4.7 - Linked List Priority Queue

Description:

In this section of the assignment I needed to take some previously written code and convert it to act like a priority queue. This meant adding functions for the insert, peek, and dequeue operations. Note, I have removed all of the unnecessary functions to save room on this report.

Source Code:

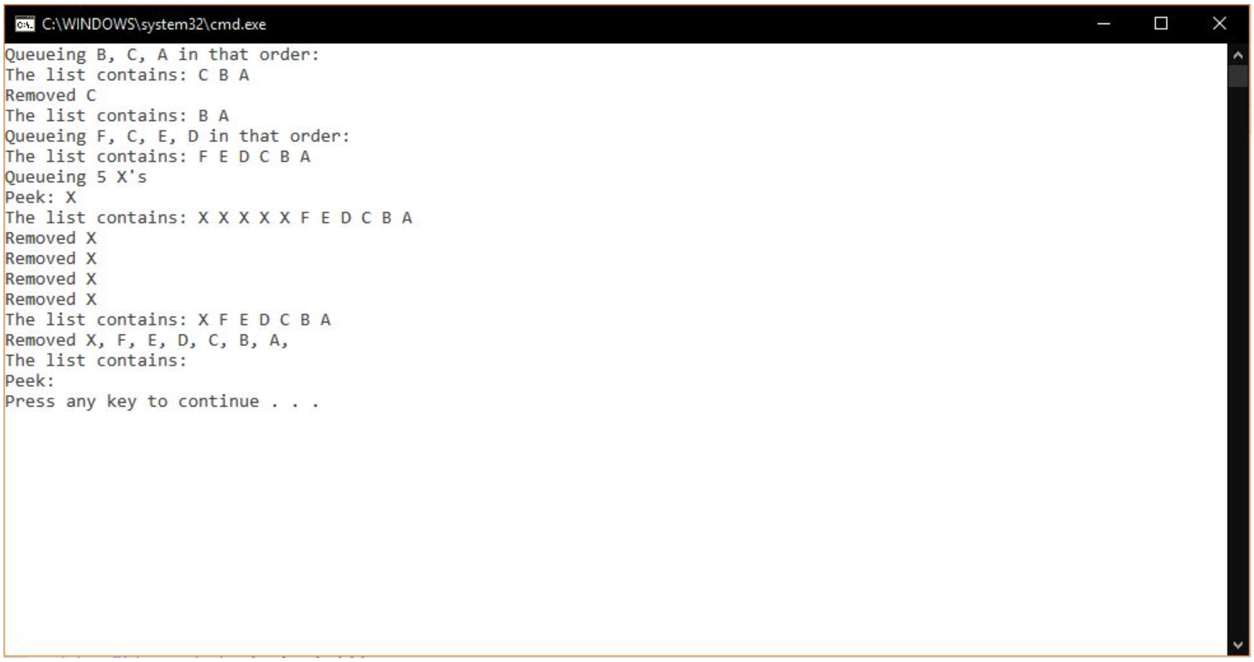
```
//Code written by Professor Ross. Modified by Quinn Roemer.
#include <iostream>
using namespace std;

//Our node.
struct node {
   node* next;
   char d;
};

//Head and tail pointers.
node* head = 0;
node* tail = 0;

//Function declarations.
char dq(void);
void traverse(void);
bool isEmpty(void);
char peek(void);
void insert(char);

//Main function to execute.
void main(void)
{
   cout << "Queueing B, C, A in that order:" << endl;
   insert('B');
   insert('C');
   insert('A');
   traverse();
   cout << "Removed " << dq() << endl;
   traverse();
   cout << "Queueing F, C, E, D in that order:" << endl;
   insert('F');
   insert('C');
   insert('E');
   insert('D');
   traverse();
   cout << "Queueing 5 X's" << endl;
   insert('X');
   insert('X');
   insert('X');
   insert('X');
   insert('X');
   cout << "Peek: " << peek() << endl;
   traverse();
   cout << "Removed " << dq() << endl;
   cout << "Removed " << dq() << endl;
   cout << "Removed " << dq() << endl;
   cout << "Removed " << dq() << endl;
   traverse();
   //Empty the list.
   cout << "Removed ";
   while (isEmpty())
   {
             cout << dq() << ", ";
   }
   cout << endl;
   traverse();
   cout << "Peek: " << peek() << endl;
}
//Traverses the list from the head to the tail, and prints out each char element.
void traverse(void)
{
   node* p = head;
   cout << "The list contains: ";
   while (p)
   {
             cout << p->d << " ";
             p = p->next;
   }
   cout << endl;
}
//Returns true if the list is empty, returns false otherwise.
bool isEmpty(void)
{
   if (!head)
   {
             return false;
   }
   else
   {
             return true;
   }
}
```

```
//Removes a char element from the head of the list and returns it.
//Returns -1 if the list is empty.
char dq(void)
{
   node* p;
   char temp;
   //Return null of the list is empty.
   if (!head)
   {
             return -1;
   }

   //One node.
   if (head == tail)
   {
             //Remove and destroy head node.
             temp = head->d;
             delete head;
             head = tail = 0;
             return temp;
   }
   //More than one node. Remove and destroy head node.
   p = head;
   head = head->next;
   temp = p->d;
   delete p;
   return temp;
}

//Returns a char element from the list.
char peek(void)
{
   //List is empty.
   if (!head)
   {
             return -1;
   }

   else
   {
             return head->d;
   }
}

//Inserts an element into the list in a greater to less fashion.
void insert(char d)
{
   node* c;
```

```
node* pc;
//Create a new node.
node* p = new node;
p->d = d;
//List is empty.
if (!head)
{
        head = tail = p;
        p->next = 0;
        return;
}
//One node.
if (head == tail)
{
        if (p->d >= head->d)
        {
                p->next = head;
                head = p;
                return;
        }
        else
        {
                head->next = p;
                p->next = 0;
                tail = p;
                return;
        }
}
//Two or more nodes.
pc = head;
c = head->next;
//Found at the head.
if (pc->d <= p->d)
{
        p->next = head;
        head = p;
        return;
}
//Look at nodes after the head node.
while (c)
{
        //Place at tail node.
        if (c == tail)
        {
                tail->next = p;
                tail = p;
                tail->next = 0;
                return;
```

```
            }
            if (c->d <= p->d)
            {
                        pc->next = p;
                        p->next = c;
                        return;
            }
            pc = c;
            c = c->next;
   }
   return;
}
```

Output/Testing:



```
Queueing B, C, A in that order:
The list contains: C B A
Removed C
The list contains: B A
Queueing F, C, E, D in that order:
The list contains: F E D C B A
Queueing 5 X's
Peek: X
The list contains: X X X X X F E D C B A
Removed X
Removed X
Removed X
Removed X
The list contains: X F E D C B A
Removed X, F, E, D, C, B, A,
The list contains:
Peek:
Press any key to continue . . .
```

BigO of insert():

This function contains one loop. During a worst case scenario the Big0 of this function would be O(N). Meaning that as the list gets bigger the time it takes for this function to complete its operation increases as well.

BigO of dq():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of isEmpty():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of peek():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

Program 4.8 - Doubly Linked List

Description:

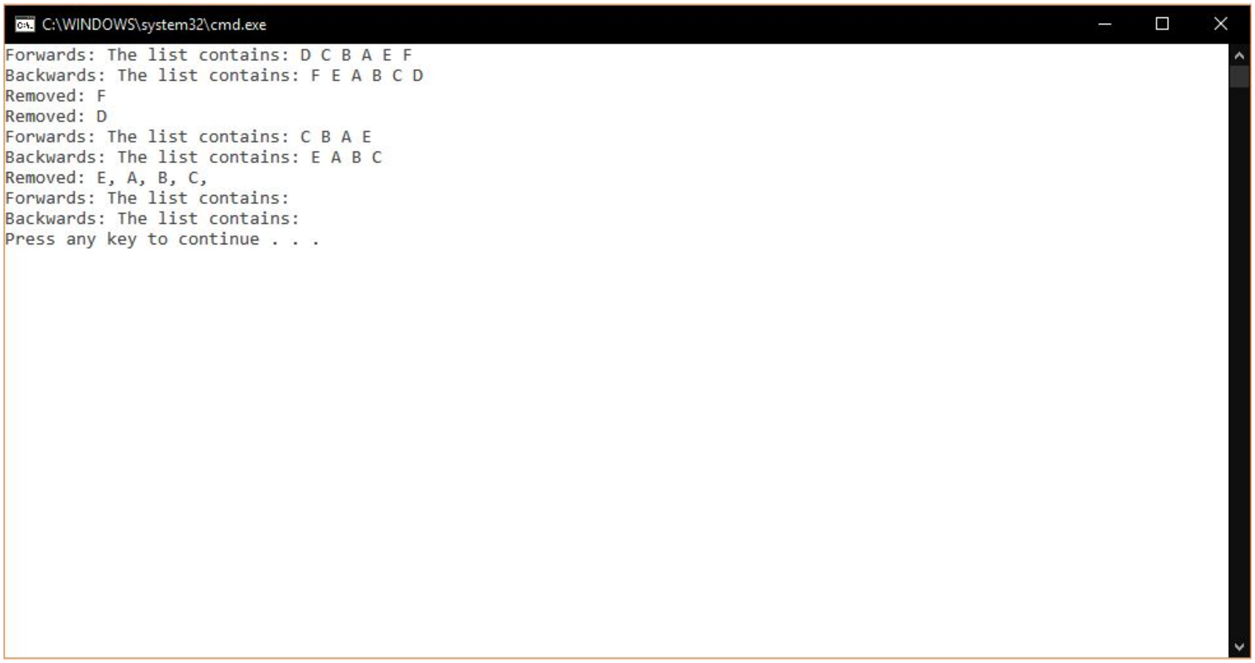
In this section of the assignment I needed to take some previously written code and convert it to act like a doubly linked list. This meant adding functions for the traverse, remove, and append operations for both directions. Note, I have removed all of the unnecessary functions to save room on this report.

Source Code:

```
//Code written by Professor Ross. Modified by Quinn Roemer.
#include <iostream>
using namespace std;

//Our node.
struct node {
   node* previous;
   node* next;
   char d;
};

//Head and tail pointers.
node* head = 0;
node* tail = 0;

//Function declarations.
char removeTail(void);
char removeHead(void);
void appendTail(char);
void appendHead(char);
void traverseFWD(void);
void traverseBWD(void);
int isEmpty(void);

//Main function to execute.
void main(void)
{
   appendTail('A');
   appendHead('B');
   appendHead('C');
   appendHead('D');
   appendTail('E');
   appendTail('F');
   cout << "Forwards: ";
   traverseFWD();
   cout << "Backwards: ";
   traverseBWD();
   cout << "Removed: " << removeTail() << endl;
   cout << "Removed: " << removeHead() << endl;
   cout << "Forwards: ";
   traverseFWD();
   cout << "Backwards: ";
   traverseBWD();

   //Empty the list.
   cout << "Removed: ";
   while (!isEmpty())
   {
            cout << removeTail() << ", ";
   }
   cout << endl;
   cout << "Forwards: ";
   traverseFWD();
   cout << "Backwards: ";
   traverseBWD();
}
//Recieves a char element and appends it to the tail of the list.
void appendTail(char d)
{
   //Make a new node.
   node* p = new node;
   p->previous = 0;
   p->next = 0;
   p->d = d;

   //List is empty.
   if (!head)
   {
            head = tail = p;
   }
   //Append to the tail end.
   else
   {
            tail->next = p;
            p->previous = tail;
            tail = p;
   }
}
//Recieves a char element and appends it to the head of the list.
void appendHead(char d)
{
   //Make a new node.
   node* p = new node;
   p->previous = 0;
   p->next = 0;
   p->d = d;

   //List is empty.
   if (!head)
   {
             head = tail = p;
   }
   //Append to the head end.
   else
   {
             head->previous = p;
             p->next = head;
             head = p;
   }
}
//Traverses the list from the head to the tail, and prints out each char element.
void traverseFWD(void)
{
   node* p = head;
   cout << "The list contains: ";
   while (p)
   {
             cout << p->d << " ";
             p = p->next;
   }
   cout << endl;
}
//Traverses the list from the tail to the head, and prints out each char element.
void traverseBWD(void)
{
   node* p = tail;
   cout << "The list contains: ";
   while (p)
   {
             cout << p->d << " ";
             p = p->previous;
   }
   cout << endl;
}
//Returns true if the list is empty, returns false otherwise.
int isEmpty(void)
{
   if (head)
   {
             return 0;
   }
   else
   {
```

```
            return 1;
   }
}
//Removes a char element from the head of the list and returns it.
//Returns -1 if the list is empty.
char removeHead(void)
{
   node* p;
   char temp;
   //Return null of the list is empty.
   if (!head)
   {
            return -1;
   }
   //One node.
   if (head == tail)
   {
            //Remove and destroy head node.
            temp = head->d;
            delete head;
            head = tail = 0;
            return temp;
   }
   //More than one node. Remove and destroy head node.
   p = head;
   head = head->next;
   head->previous = 0;
   temp = p->d;
   delete p;
   return temp;
}
//Removes a char element from the tail of the list and returns it.
//Returns -1 if the list is empty.
char removeTail(void)
{
   node* p;
   char temp;
   //Return null of the list is empty.
   if (!head)
   {
            return -1;
   }
   //One node.
   if (head == tail)
   {
            //Remove and destroy tail node.
            temp = head->d;
            delete head;
            head = tail = 0;
            return temp;
    }
    //More than one node. Remove and destroy tail node.
    p = tail;
    tail = tail->previous;
    tail->next = 0;
    temp = p->d;
    delete p;
    return temp;
}
```

Output/Testing:



```
Forwards: The list contains: D C B A E F
Backwards: The list contains: F E A B C D
Removed: F
Removed: D
Forwards: The list contains: C B A E
Backwards: The list contains: E A B C
Removed: E, A, B, C,
Forwards: The list contains:
Backwards: The list contains:
Press any key to continue . . .
```

BigO of appendTail():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of appendHead():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of removeTail():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of removeHead():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

BigO of traverseFWD():

This function contains one loop. During an operation, the Big0 of this function would be O(N). Meaning that as the list gets bigger the time it takes for this function to complete its operation increases as well.

BigO of traverseBWD():

This function contains one loop. During an operation, the Big0 of this function would be O(N). Meaning that as the list gets bigger the time it takes for this function to complete its operation increases as well.

BigO of isEmpty():

This function contains no loops and only uses conditional statements. According to our rules of thumb this means the BigO is O(1). Or constant time.

Conclusion

This assignment was fun! Being able to build containers that work like libraries I would just usually call was an interesting task. By doing this my understanding of each structure increased dramatically. Looking forward to future homework assignments.