#include "nodes\_LLoLL.h"

#include "cnPtrQueue.h"

#include <iostream>

using namespace std;

namespace CS3358\_FA2019\_A5P2

{

void Destroy\_cList(CNode\*& cListHead)

{

int count = 0;

CNode\* cNodePtr = cListHead;

while (cListHead != 0)

{

cListHead = cListHead->link;

delete cNodePtr;

cNodePtr = cListHead;

++count;

}

cout << "Dynamic memory for " << count << " CNodes freed"

<< endl;

}

void Destroy\_pList(PNode\*& pListHead)

{

int count = 0;

PNode\* pNodePtr = pListHead;

while (pListHead != 0)

{

pListHead = pListHead->link;

Destroy\_cList(pNodePtr->data);

delete pNodePtr;

pNodePtr = pListHead;

++count;

}

cout << "Dynamic memory for " << count << " PNodes freed"

<< endl;

}

// do depth-first traversal and print data

void ShowAll\_DF(PNode\* pListHead, ostream& outs)

{

while (pListHead != 0)

{

CNode\* cListHead = pListHead->data;

while (cListHead != 0)

{

outs << cListHead->data << " ";

cListHead = cListHead->link;

}

pListHead = pListHead->link;

}

}

// Do breadth-first (level) traversal and print data

void ShowAll\_BF(PNode\* pListHead, ostream& outs)

{

if (pListHead == nullptr)

return;

CNode\* cursor = nullptr;

cnPtrQueue queue;

while(pListHead != nullptr)

{

if(pListHead->data != 0)

{

queue.push(pListHead->data);

}

pListHead = pListHead->link;

}

while(!queue.empty())

{

cursor = queue.front();

queue.pop();

outs << cursor->data << " ";

if(cursor->link != nullptr)

queue.push(cursor->link);

}

}

}