Cameron White

Programming Assignment #1

Source Code:

#include <iostream>

#include <cstdio>

using namespace std;

template <class T>

class Sequence

{

private: // Internal Representation

class NodeRecord {

public:

T value;

NodeRecord\* next;

};

NodeRecord\* head;

int size;

public:

Sequence ();

~Sequence ();

void clear (void);

//implement the following four functions

void add (T& x, int pos);

//! updates self

//! restores pos

//! clears x

//! requires: 0 â‰¤ pos â‰¤ |self|

//! ensures: self = #self[0, pos) \* <#x> \* #self[pos, |#self|)

void remove (T& x, int pos);

//! updates self

//! restores pos //! replaces x

//!requires: 0 â‰¤ pos < |self|

//!ensures: <x> = #self[pos, pos+1) and

//! self = #self[0, pos) \* #self[pos+1, |#self|)

T& entry (int pos);

//! restores self, pos

//! requires: 0 â‰¤ pos < |self|

//! ensures: <entry> = self[pos, pos+1)

int length (void);

//! restores self

//! ensures: length = |self|

void outputSequence(void);

//! restores self

//! ensures: self = #self

private: // Internal operations

void reclaimAllNodes (NodeRecord\*& p);

};

template <class T>

Sequence<T>::Sequence ()

{

head = NULL;

size = 0;

}// Sequence

template <class T>

Sequence<T>::~Sequence ()

{

reclaimAllNodes(head);

} // ~Sequence

template <class T>

void Sequence<T>::clear (void)

{

reclaimAllNodes(head);

head = NULL;

} // clear

template <class T>

void Sequence<T>::reclaimAllNodes (NodeRecord\*& initialP)

// requires: Null(initialP) or Alive(initialP)

// ensures: all nodes in linked list

// beginning with initialP are reclaimed

{

if (initialP != NULL) {

reclaimAllNodes(initialP->next);

delete (initialP);

} // end if

} // reclaimAllNodes

template <class T>

void Sequence<T>:: add (T& x, int pos)

{

if(pos >= 0 && pos <= size)

{

NodeRecord \*node = new NodeRecord();

node->value = x;

node->next = NULL;

NodeRecord \* curr = head;

NodeRecord \* pre = NULL;

int curr\_pos = 0;

while(curr\_pos < pos)

{

pre = curr;

curr = curr -> next;

curr\_pos++;

}

if(pre == NULL)

{

node -> next = head;

head = node;

}

else

{

pre -> next = node;

node -> next = curr;

}

size++;

}

else

{

cout << endl << "ERROR: " <<pos << " outside range" << endl;

}

}

template <class T>

void Sequence<T>:: remove(T& x, int pos)

{

if(pos >= 0 && pos < size)

{

NodeRecord \*curr = head;

NodeRecord \*pre = NULL;

int curr\_pos = 0;

while(curr\_pos < pos)

{

pre = curr;

curr = curr -> next;

curr\_pos++;

}

if(pre == NULL)

head = head -> next;

else

{

pre -> next = curr -> next;

}

x = curr -> value;

curr -> next = NULL;

delete(curr);

curr = NULL;

size--;

}

else

{

cout<<endl<<"ERROR: " <<pos<< " outside range" <<endl;

}

}

template <class T>

T& Sequence<T>:: entry(int pos)

{

if(pos >= 0 && pos < size)

{

NodeRecord \* curr = head;

int curr\_pos = 0;

while(curr\_pos < pos)

{

curr = curr -> next;

curr\_pos++;

}

return curr -> value;

}

else

{

T x;

cout << endl <<"ERROR: " << pos << " outside range" << endl;

return x;

}

}

template <class T>

int Sequence<T>::length (void)

{

return size;

}

template <class T>

void Sequence<T>:: outputSequence(void)

{

NodeRecord \* curr = head;

cout<<"<";

while(curr != NULL)

{

cout << curr -> value;

curr = curr -> next;

if(curr != NULL)

cout<<", ";

}

cout<<">"<<endl;

}

// You are required to implement add, remove, entry, length, and outputSequence functions here.

int main()

{

Sequence<int> s1;

Sequence<int> s2;

Sequence<int> s3;

int xi = 0;

cout << "SEQUENCE ONE\nS1: ";

s1.outputSequence();

cout << "\n\nTESTING ADD, OUTPUTSEQUENCE, LENGTH\nFilled S1 with <23, 14, 18, 31, 11>";

xi = 23;

s1.add(xi, 0);

xi = 14;

s1.add(xi, 1);

xi = 18;

s1.add(xi, 2);

xi = 31;

s1.add(xi, 3);

xi = 11;

s1.add(xi, 4);

cout << "\nS1: ";

s1.outputSequence();

cout << "\nS1 size: " << s1.length();

cout << "\n\nADD, OUTPUTSEQUENCE, LENGTH\nAdded 17 to position 0";

xi = 17;

s1.add(xi, 0);

cout << "\nS1: ";

s1.outputSequence();

cout << "\nS1 size: " << s1.length();

cout << "\n\nTESTING REMOVE\n+ OUTPUTSEQUENCE, LENGTH\nRemoved entry at position 3";

s1.remove(xi, 3);

cout << "\nElement removed: "<<xi;

cout << "\nS1: ";

s1.outputSequence();

cout << "\nS1 size: " << s1.length();

cout << "\n\nTESTING ENTRY\nS1[1]: "<< s1.entry(1);

cout << "\nS1[0]: " << s1.entry(0);

cout << "\n\nTESTING CLEAR\n+ OUTPUTSEQUENCE\nClearing Sequence One\nS1:";

s1.clear();

s1.outputSequence();

/\* TESTING SEQUENCE TWO

TESTED:

ADD

OUTPUTSEQUENCE

LENGTH

ENTRY

REMOVE

CLEAR

\*/

//Client does NOT abide by requires clauses

//Goes over size

cout << "\n\n\nSEQUENCE TWO\nS2: ";

s2.outputSequence();

cout << "\n\nADD, OUTPUTSEQUENCE, LENGTH\nFilled S2 with <1>";

xi = 1;

s2.add(xi, 0);

cout << "\nS2: ";

s2.outputSequence();

cout << "\nS2 size: " << s2.length();

cout << "\n\nADD, OUTPUTSEQUENCE, LENGTH\nAdded 3 to position 3";

xi = 3;

s2.add(xi, 3); //error because 3 > size

cout << "\nS2: ";

s2.outputSequence();

cout << "\nS2 size: " << s2.length();

cout << "\n\nENTRY\nS2[3]: ";

cout << s2.entry(3); //error because 3 > size

cout << "\n\nREMOVE, OUTPUTSEQUENCE\nRemoved entry at position 2";

s2.remove(xi, 2);

cout << "Element removed: " << xi;

cout << "\nS2: ";

s2.outputSequence();

cout << "\n\nENTRY";

cout << "\nS2[0]: ";

cout << s2.entry(0);

cout << "\nS2[1]: ";

cout << s2.entry(1); //error because 1 = size

cout << "\nS2[2]: ";

cout << s2.entry(2); //error because 2 > size

cout << "\nS2[3]: ";

cout << s2.entry(3); // ||

cout << "\n\nCLEAR, OUTPUTSEQUENCE, LENGTH\nClearing Sequence Two";

s2.clear();

cout << "\nS2: ";

s2.outputSequence();

cout << "\nS2 size: " << s2.length();

/\* TESTING SEQUENCE THREE

TESTED:

ADD

OUTPUTSEQUENCE

LENGTH

ENTRY

REMOVE

CLEAR

\*/

cout << "\n\n\nSEQUENCE THREE\nS3: ";

s3.outputSequence();

cout << "\n\nADD, OUTPUTSEQUENCE, LENGTH\nFilled S3 with <8, 23, 2000>";

xi = 8; //08 => invalid octal digit ??

s3.add(xi, 0);

xi = 23;

s3.add(xi, 1);

xi = 2000;

s3.add(xi, 2);

cout << "\nS3: ";

s3.outputSequence();

cout << "\nS3 size: " << s3.length();

cout << "\n\nADD, OUTPUTSEQUENCE, LENGTH\nAdded 1 to position 0 + 6 to position 1 + 2002 to position 3>";

xi = 1;

s3.add(xi, 0);

xi = 6;

s3.add(xi, 1);

xi = 2002;

s3.add(xi, 2);

cout << "\nS3: ";

s3.outputSequence();

cout << "\nS3 size: " << s3.length();

cout << "\n\nENTRY, REMOVE";

cout << "\nS3[2]: " << s3.entry(2);

cout << "\nRemoved entry at position 2";

s3.remove(xi, 2);

cout << "Element removed: " << xi;

cout << "\nS3[2]: " << s3.entry(2);

cout << "\nRemoved entry at position 2";

s3.remove(xi, 2);

cout << "Element removed: " << xi;

cout << "\nS3[3]: " << s3.entry(2);

cout << "\n\nOUTPUTSEQUENCE, LENGTH\nS3: ";

s3.outputSequence();

cout << "\nS3 size: " << s3.length();

cout << "\n\nCLEAR, OUTPUTSEQUENCE, LENGTH\nClearing Sequence Three";

s3.clear();

cout << "\nS3: ";

s3.outputSequence();

cout << "\nS3 size: " << s3.length();

cout << "\n\nADD, OUTPUTSEQUENCE, LENGTH\nAdded 6 to position 0, 2019 to position 1, and 25 to position 1";

xi = 6;

s3.add(xi, 0);

xi = 2019;

s3.add(xi, 1);

xi = 25;

s3.add(xi, 1);

cout << "\nS3: ";

s3.outputSequence();

cout << "\nS3 size: " << s3.length();

return 0;

} //main

/\*

SEQUENCE ONE

S1: <>

TESTING ADD, OUTPUTSEQUENCE, LENGTH

Filled S1 with <23, 14, 18, 31, 11>

S1: <23, 14, 18, 31, 11>

S1 size: 5

ADD, OUTPUTSEQUENCE, LENGTH

Added 17 to position 0

S1: <17, 23, 14, 18, 31, 11>

S1 size: 6

TESTING REMOVE

+ OUTPUTSEQUENCE, LENGTH

Removed entry at position 3

Element removed: 18

S1: <17, 23, 14, 31, 11>

S1 size: 5

TESTING ENTRY

S1[1]: 23

S1[0]: 17

TESTING CLEAR

+ OUTPUTSEQUENCE

Clearing Sequence One

S1:<>

SEQUENCE TWO

S2: <>

ADD, OUTPUTSEQUENCE, LENGTH

Filled S2 with <1>

S2: <1>

S2 size: 1

ADD, OUTPUTSEQUENCE, LENGTH

Added 3 to position 3

ERROR: 3 outside range

S2: <1>

S2 size: 1

ENTRY

S2[3]:

ERROR: 3 outside range

29651

REMOVE, OUTPUTSEQUENCE

Removed entry at position 2

ERROR: 2 outside range

Element removed: 3

S2: <1>

ENTRY

S2[0]: 1

S2[1]:

ERROR: 1 outside range

29651

S2[2]:

ERROR: 2 outside range

29651

S2[3]:

ERROR: 3 outside range

29651

CLEAR, OUTPUTSEQUENCE, LENGTH

Clearing Sequence Two

S2: <>

S2 size: 1

SEQUENCE THREE

S3: <>

ADD, OUTPUTSEQUENCE, LENGTH

Filled S3 with <8, 23, 2000>

S3: <8, 23, 2000>

S3 size: 3

ADD, OUTPUTSEQUENCE, LENGTH

Added 1 to position 0 + 6 to position 1 + 2002 to position 3>

S3: <1, 6, 2002, 8, 23, 2000>

S3 size: 6

ENTRY, REMOVE

S3[2]: 2002

Removed entry at position 2Element removed: 2002

S3[2]: 8

Removed entry at position 2Element removed: 8

S3[3]: 23

OUTPUTSEQUENCE, LENGTH

S3: <1, 6, 23, 2000>

S3 size: 4

CLEAR, OUTPUTSEQUENCE, LENGTH

Clearing Sequence Three

S3: <>

S3 size: 4

ADD, OUTPUTSEQUENCE, LENGTH

Added 6 to position 0, 2019 to position 1, and 25 to position 1

S3: <6, 25, 2019>

S3 size: 7

Exit code: 0 (normal program termination)

\*/