Bag class Decription:

The bag class contains a bag of ItemType (string, unsigned long, etc); once an item is inserted into the bag, it is inserted in alphabetical order or from least to greatest.

This bag list consists of a head pointer, tail pointer, and dummy node.

The node consists of a int, ItemType, node\*next, node\*prev.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Bag b; | b.insert("Jim"); |  | //List |  |
|  |  |  |  | //ADDRESS |  |
| m\_headBag | NULL | &Jim |  | Jim |  |
| m\_tailBag | NULL | &Jim |  | 1 |  |
| m\_size | 0 | 1 |  | NULL | //\*next |
| m\_uniqueSize | 0 | 1 |  | NULL | //\*prev |
|  |  |  |  |  |  |
|  |  | b.insert("Air"); |  |  |  |
|  | m\_headBag | &Air |  | Air |  |
|  | m\_tailBag | &Jim |  | 1 |  |
|  | m\_size | 2 |  | &Jim | //\*next |
|  | m\_uniqueSize | 2 |  | NULL | //\*prev |
|  |  |  |  |  |  |
| struct ptrBag |  |  |  | Jim |  |
| {  int | num; |  |  | 1 |  |
| ItemType | Item; |  |  | NULL | //\*next |
| ptrBag\* | next; |  |  | &Air | //\*prev |
| ptrBag\* | prev; }; |  |  |  |  |

Pseudocode:

Bag& operator=(const Bag &rhs)

{

check aliasing if current bag != rhs bag,

deallocate dynamic memory of current bag

create a temp node pointer

Use traversal loop to copy and assign each member variable

Create a new dynamic node

If first item, point head pointer to new node

Else, using dummy to keep previous node address

Point previous item to new item

Point new item to previous item

If last item, point tail pointer to new node

Set dummy node to current new node address

Set temp to next temp address

Copy size from rhs

Copy unique Size from rhs

return pointer to this bag

}

bool insert (const ItemType& value)

{

call find() to get slotNum of value within bag list

if slotNum < 0, something is wrong

if current bag contains value

loop to value numSlot

increment m\_size and numCount

current bag does not contain value

insert value into a dynamic location

if slotNum = 0

insert at beginning of list

implement special case for only 1 item in bag

loop to slotNum

if slotNum = tail pointer

insert at end of list

implement special case for only 1 item in bag

else insert value to middle of bag list

}

int erase (const ItemType& value)

{

if value is not in bag

do not erase anything

call find() to get slotNum

loop to slotNum

if item at slotNum position appear more than once in bag

decrement m\_size and numCount

else slotNum position contain only one item

if slotNum = head pointer

implement special case for only 1 item in bag

erase value at beginning of list

if slotNum = tail pointer

erase value at end of list

else slotNum is somewhere in the middle

erase value at the middle of list

}

void swap (Bag& other)

{

create temporary pointer and int

swap head pointer with other bag

swap tail pointer with other bag

swap size with other bag

swap uniqueSize with other bag

}

int find (const ItemType& value) const

{

create int slotNum counter

if head pointer point to NULL

return slotNum = 0

loop forward to each item in bag

if current loop item = value

return slotNum

increment slotNum by 1

check if current loop item = tail pointer

implement special case for only 1 item in bag

return 0 if value less than loop item

return 1 if value greater than loop item

loop backwards to each item in bag

return slotNum-1, if loop position is at beginning of list

return slotNum, if loop position is at end of list

return slotNum+1, if loop position is at middle of list

decrement slotNum

}

void combine (Bag& b1, Bag& b2, Bag\* result)

{

create an new bag

create a loop that loops two times

check for aliasing, if (b1 = b2 or b1 = b2 = result)

aliasing is true

add items of b1 to new result bag twice.

Break from loop

If not aliasing

during first loop, set bag pointer to b1

during second loop, set bag pointer to b2

for each loop, loop through b1 or b2 bag

for each item in bag loop once

insert item to new result bag

set result bag equal new bag

}

void subtract (Bag& b1, Bag& b2, Bag\* result)

{

create an new bag

check for NO aliasing, (b1 = b2 or b1 = b2 = result)

loop through each item of b1

if b2 contains the b1 loop item

insert b1 loop item to new result bag ONLY if difference of b1 and b2 of this current loop item is positive

else, b2 does not contain b1 loop item

insert loop item to new result bag for each time it appears in b1

swap result bag with new bag

}

Test cases for each function: (assume ItemType std::string)

1).

Bag() //Default Bag constructor

Bag a; //ok empty bag

Bag b(5); //error

Bag\* c = new Bag; //dynamic Bag created

2).

Bag(const Bag &cpyBag) //Copy Constructor

Bag a; //ok empty bag

a.insert(“John”); //insert John to bag a

Bag b(a) or Bag b =a; //bag b copies items on bag a

Bag b(\*a); //error because parameter is suppose to accept an address to a bag

Bag\* c = new Bag(b); //dynamic bag c created and copies bag b

3).

Bag& operator=(const Bag &rhs) //Assignment Constructor

Bag a, b, c; //create bag a and b

assert(a.insert(“Yes”); //insert “Yes” to bag a

b = a; //bag b copies the contents of bag a

c = b(a); //error since bag b is not a new bag being created it cannot

//call the copy constructor

4).

~Bag() //Destructor

//called when a bag is deleted or exits from a function

5).

bool empty() const

int size() const

int uniqueSize() const

Bag a, b; //create bag a

assert(a.empty()); //test for empty bag

assert(a.insert(“Yes”)); //insert something to bag a

assert(a.empty()); //flag an error, bag a is not empty

assert(a.size() == 1); //test pass

assert(a.uniqueSize() == 1); //test pass

b(a);

assert(!b.empty() && b.size() == a.size()) //test copy contructor if contents are copied

6).

bool insert(const ItemType& value)

*int* erase(const ItemType& value)

int eraseAll(const ItemType& value)

int count(const ItemType& value) const

void swap(Bag& other)

INPUT:

|  |
| --- |
| Bag b;  cout << "Bag b was just created!" << endl;  assert(b.empty());  assert(b.size() == 0 && b.uniqueSize() == 0);    //assert(! b.contains("")); //only test for string  assert(b.erase("Ben") == 0);  b.insert("Ben");  b.insert("Ben");  b.insert("Ben"); //add to struct 3 times  b.insert("Eva");  b.insert("Eva"); //add to struct 2 times  b.insert("Joe");  b.insert("Peter");  b.insert("Frank");  b.insert("Zebra");  b.insert("Joe");;  b.insert("Zebra");  b.insert("Zephyr"); //appear as last item  b.insert("Apple"); //appear as first item    int erased = 0; //create a counter    //Test case for erase function    erased += b.erase("Zebra"); //add 1 to counter  erased += b.erase("Apple"); //add 1 to counter  erased += b.erase("Ben"); //add 1 to counter  erased += b.erase("Ben"); //add 1 to counter  erased += b.erase("Ben"); //add 1 to counter  erased += b.erase("Joe"); //add 1 to counter  erased += b.erase("Peter"); //add 1 to counter  erased += b.erase("Frank"); //add 1 to counter  erased += b.erase("Joe"); //add 1 to counter  erased += b.erase("Zebra"); //add 1 to counter  erased += b.erase("Zephyr"); //add 1 to counter  erased += b.erase("Test"); //add 0 to counter  /\*  //Test case for eraseAll function  erased += b.eraseAll("Zebra"); //add 2 to counter  erased += b.eraseAll("Apple"); //add 1 to counter  erased += b.eraseAll("Ben"); //add 3 to counter  erased += b.eraseAll("Joe"); //add 2 to counter  erased += b.eraseAll("Peter"); //add 1 to counter  erased += b.eraseAll("Frank"); //add 1 to counter  erased += b.eraseAll("Zephyr"); //add 1 to counter  erased += b.eraseAll(“Test”); //add 0 to counter  \*/  b.insert("Victor"); //insert victor,  b.insert("Victor"); //appear as last item  b.dump();  cout << "Total erased was:\t" << erased << endl;  cout << "The bag has " << b.count("Eva") << " Eva, " << b.count(" Victor") << "Victor, but ";  cout << b.count("Justin") << " Justin.\n";    Bag c;  c.insert("Jones");  cout << endl << "Bag c was just created! And Jones was added." << endl;  b.swap(c);  cout << "After swap bag b with c, size of b and c is: " << b.size() << " " << c.size() << endl;  cout << "\nContents of bag c\n\n";  c.dump();  cout << "nContents of bag b\n\n";  b.dump(); |

OUTPUT:

|  |  |
| --- | --- |
| //With erase function  Bag b was just created!  Eva 2  Victor 2  m\_size: 4 m\_uniqueSize: 2  m\_headBag: Eva m\_tailBag: Victor  Total erased was: 11  The bag has 2 Eva, 2 Victor, but 0 Justin.  Bag c was just created! And Jones was added.  After swap bag b with c, size of b and c is: 1 4  Contents of bag c  Eva 2  Victor 2  m\_size: 4 m\_uniqueSize: 2  m\_headBag: Eva m\_tailBag: Victor  Contents of bag b  Jones 1  m\_size: 1 m\_uniqueSize: 1  m\_headBag: Jones m\_tailBag: Jones | //with eraseAll function  Bag b was just created!  Eva 2  Victor 2  m\_size: 4 m\_uniqueSize: 2  m\_headBag: Eva m\_tailBag: Victor  Total erased was: 11  The bag has 2 Eva, 2 Victor, but 0 Justin.  Bag c was just created! And Jones was added.  After swap bag b with c, size of b and c is: 1 4  Contents of bag c  Eva 2  Victor 2  m\_size: 4 m\_uniqueSize: 2  m\_headBag: Eva m\_tailBag: Victor  Contents of bag b  Jones 1  m\_size: 1 m\_uniqueSize: 1  m\_headBag: Jones m\_tailBag: Jones |

7).

void start();

void next();

bool ended() const;

const ItemType& currentValue() const;

int currentCount() const;

INPUT: OUTPUT:

|  |  |
| --- | --- |
| Bag d;  cout << endl << "Bag d was created and testing...\n"  d.insert("Kobe Byrant");  d.insert("Plays");  d.insert("Tennis");  d.insert("Michael Jordan");  d.insert("Plays");  d.insert("Golf");  for (d.start(); !d.ended(); d.next())  cout << d.currentValue() << " occurs " << d.currentCount() << " times." << endl; | Bag d was created and testing...  Golf occurs 1 times.  Kobe Byrant occurs 1 times.  Michael Jordan occurs 1 times.  Plays occurs 2 times.  Tennis occurs 1 times. |

8).

void combine(Bag& b1, Bag& b2, Bag& result);

void subtract(Bag& b1, Bag& b2, Bag& result);

//Assume ItemType is unsigned long

Input:

|  |
| --- |
| Bag a, b, d, test;  a.insert(1); //insert 1 3x, 5 1x,and 9 3x, total of 7 items  a.insert(1);  a.insert(1);  a.insert(5);  a.insert(9);  a.insert(9);  a.insert(9);  //a.insert(“John”) //error since ItemType is unsigned long  b.insert(2); //Total of 7 items inserted to b  b.insert(3);  b.insert(4);  b.insert(6);  b.insert(7);  b.insert(8);  b.insert(0);  Bag c = a; |
| //copies All contents of a to bag c using copy constructor |
| d = b; |
| //copies all contents of b to bag d using assignment constructor |
| //testing combine function  combine(a, b, test); //note combine(b, a, test) is same result  assert(test.size() == 14 && test.count(1) == 3 && test.count(0) == 1);  /\*test should contain contents of both a and b\*/  combine(a,a,test);  assert(test.size() == 14 && test.count(1) == 6 && test.count(9) == 6);  /\*test bag should contain 2 times the contents of a\*/  combine(b, b ,test);  assert(test.size() == 14 && test.count(0) == 2 && (! test.contains(1)) );  /\*test bag should contain 2 times the contents of b\*/  combine(a,a,a);  assert(a.size() == 14 && a.count(9) == 6 && (! a.contains(0)) );  /\*a bag should contain twice its own content\*/ |
| //testing subtract function ignore above combine test!  subtract(c, d, test);  assert(test.size() == 7 && !test.contains(0) && test.uniqueSize() == 3);  /\*There’s nothing in common in bag c and d, so test contains only content of c or a\*/  subtract(d, d, test);  assert(test.size() == 0 && test.count(0) == 0 && (! test.contains(8)) );  /\*test bag should not contain anything, thus a empty bag\*/  combine(a, b, test);  subtract(test, b, test);  a.insert(1);  assert(test.size() == 7 && test.count(0) == 0 && (! test.contains(8)) );  assert(test.uniqueSize() == 3 && test.count(1) == 3 && test.count(5) == 1);  a.erase(1);  /\*test bag should contain only contents of bag a\*/  combine(a,a,test);  assert(test.size() == 14 && test.count(1) == 6 && (! test.contains(0)) );  subtract(test,a,test);  assert(test.size() == 7 && test.count(1) == 3 && test.uniqueSize()==3 );  combine(test, test, test);  assert(test.size() == 14 && test.count(1) == 6 && test.contains(9));  subtract(test, test, d);  assert(d.size() == 0 && d.count(1) == 0 && (! d.contains(5)) );  /\*d bag should be an empty bag\*/  d.insert(1);  assert(d.size() == 1 && d.count(1) == 1 && d.contains(1) );  /\*d bag should contain only one item which is 1\*/ |