Notes for Midterm

Tuesday, October 6, 2020 1:03 PM

3 Types of Polymorphism

- Virtual
- Name overloading
- Operator overloading

How to declare a pure virtual method:

```
virtual void method() = 0;
```

T/F, a pure virtual class has all pure virtual methods

T

Outline the base and derived hierarchy of creation & deletion

B
D
D
B

Create a pointer named Student to be placed on the heap.

```
Student *ptr = New Student();
```

Name the Six components that make a make file

Shell lines
Dependency lines
Comments
Macros
Rules
Inference Rules

What are the 3 types of inheritance

Public, Private, Protected

What is visible in class A?

```
class A
{
public:
    int x;
protected:
    int y;
private:
    int z;
};

class B : public A
{
    // x is public
```

```
};

class B : public A
{
    // x is public
    // y is protected
    // z is not accessible from B
};

class C : protected A
{
    // x is protected
    // y is protected
    // z is not accessible from C
};

class D : private A    // 'private' is default for classes
{
```

What is the default inheritence visibility for classes?

Private

What is a template in c++?

a generic ADT

T/F, Virtual Functions or Methods support Static polymorphism

False, they support dynamic polymorphism

What is an ADT?

data & operations on that data

- attributes/opperations

What are the 2 types of abstraction

Data & Functional

What are the 4 phases of an ADT's life

Construction
Initialization
Usage

Initialization
Usage
destruction

Lifetime is a temporal concept

True

T/F, << this operator shifts to the right

False

What is data abstraction & why is it important?

What is the OO concept

it boils down to the idea that we can model things in software using terminology that we use to describe the same objects in the domain world

What are the 4 pillars of OO paradigm

- Inheritance: classes inherit properties from other classes
- Polymorphism: objects determine appropriate operations at execution time
- Abstraction: Separate purpose of a module from its implementation
- Encapsulation: objects combine data & operations

What is coupling?

- dependency of one class on another
- tied closely with cohesion

What are the types of coupling?

- Common
- Control
- Stamp
- method
- External

What is a data structure?

- An implementation of an ADT within a language

What's the difference between Scope & lifetime?

- Scope: textual or spacial
- Lifetime: Temporal concept

T/F, Lists are homogeneous

True

Whats the difference between linked-based & array-based implementations?

- Arrays: - Dynamic allocation is costly
  - Array could waste space
  - Easy to use
- Link-Based: - Traversals are expensive
  - No fixed size
  - Next item/element
  - more memory

Vector Notes:

- Vectors are another way to hold data items of the same type
- Predifined operations
- Can grow
- Elements can be of any type
- Elements are placed in contiguous storage: they can be accessed and traversed using iterators
- No need to manage our memory
- Good: Random Read access and insertion and

- No need to manage our memory
- Good: Random Read access and insertion and deletion at the end
- Bad: Insertions and deletions in the front or any other position

Linked-Lists vs Vectors

- Good: inserting and deleting items in the front, end, middle
  - Address of Node does not change
- Bad: Random element access

Bitwise operations:

- AND &
- inclusive OR |
- Exclusive OR ^
- Ones complement ~
- Left Shift <<
- Right Shift >>

More Notes:

an abstract base class allows the client to take full advantage of polymorphism.

Final Exam

Monday, December 7, 2020 11:31 AM

BUTLER UNIVERSITY

Final Exam

CS 341

100 Points

Dr. Ryan Rybarczyk

12/7/2020

Name: N8 Swalley

Instructions: You may use any material posted on our Canvas or on your personal course GitHub repository – you are not allowed to use Google or any other Internet Resource on this exam. Any failure to adhere to this policy will result in a case of Academic Dishonesty and will subject to a 0 on the exam.

Please answer each question to the best of your ability. Make sure your answer clearly addresses the question(s) included – many questions contain multiple parts. You may either print this exam off and work on a hard copy and scan it and submit it on your GitHub repository or you may write your answers in a separate document and submit that document via your GitHub course repository. If you have any questions please don't hesitate to ask.

Honor Pledge:

I, N8 Swalley, pledge that I did not receive any additional assistance on this exam and that I have read the above statement on Academic Integrity and will adhere to that policy on this exam. I acknowledge that failure to do so will result in a 0 on this exam.

Best of Success!

1

```
!false….it’s funny because it’s true.
```

1. We discussed the ability to use Bitwise Operation in lecture and through our exploration of Bit Vectors in our Assignment #3. What are the four (4) bitwise operators that we discussed in lecture? Provide an example of how each operates in practice.

- << left shift: This operator shifts the bits to the left by the given value. Example: x << 2;

- >> right shift: This operator shifts the bits to the right by the given value. Example: x >>2;

- & And: This operator and's the bits. Example: andBits = x & y;

- || inclusive or: This operator or's the bits. Example: orBits = x || y;

2

```
using namespace std; is bad
```

2. Why is it important for a Binary Search Tree to be balanced? What benefit does it provide the user? Please provide an example to support your claim.

- It is important for a Binary Search Tree to be balanced because this allows for much easier and more efficient traversals by the user. Example: if we had a tree that was completely out of balance it would be very costly and inefficient to perform the different traversals (inorder, preorder and postorder). A balanced search tree also makes operations on the tree much easier (insertions and deletions)

3. Why when inserting a new node into a Red-Black Tree do we insert a 'Red' node as opposed to inserting a 'Black' node by default?

- This is required so that the RB tree can be

- This is required so that the RB tree can be properly balanced using the rest of the 6 rules. If inserted nodes were Black by default, then the rule that says every path from a node to a nullptr must contain the same number of black nodes would become much harder to account for. The more I think about it, if the nodes were black by default, there would never even be red nodes!!! We wouldn't have a reason to change nodes to red with the way the rules are setup

3

```
Help!! I'm stuck! Can you give me a few pointers??????
            *ptrHint, *ptrAnswer, *ptrTaco
```

4. In hashing we can encounter what is known as a collision. Describe what this means in terms of hashing and provide examples of how this can be handled in practice.

- A collision in hashing is when we try to insert a value into a spot in the hash table that is already occupied by another value based on the hash function and key. A way to handle these collisions is through probing. Specifically, in linear probing, we'd account for collisions by inserting the colliding value into the next open place in the hash table (after the hash value).

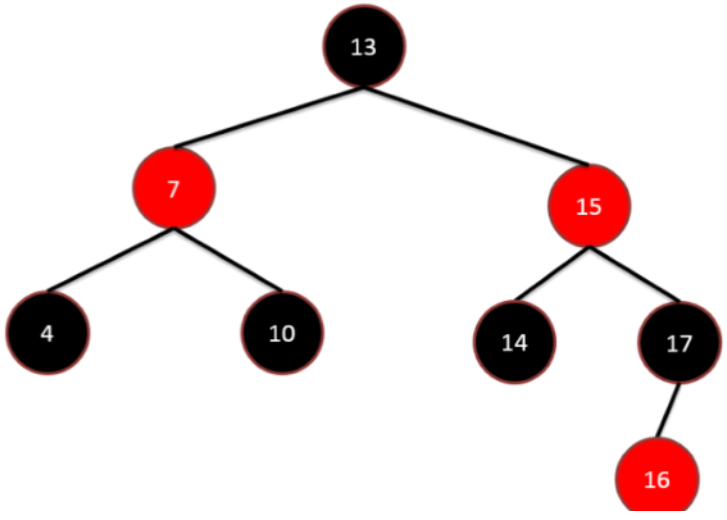
open place in the hash table (after the hash value).

5. Why is Separate Chaining a more efficient strategy than both Linear Probing and Quadratic Probing when it comes to hashing?

- Separate Chaining is a more efficient strategy than both Linear Probing and Quadratic Probing because there are no collisions in separate chaining!!! Its simply added to the end of the linked list

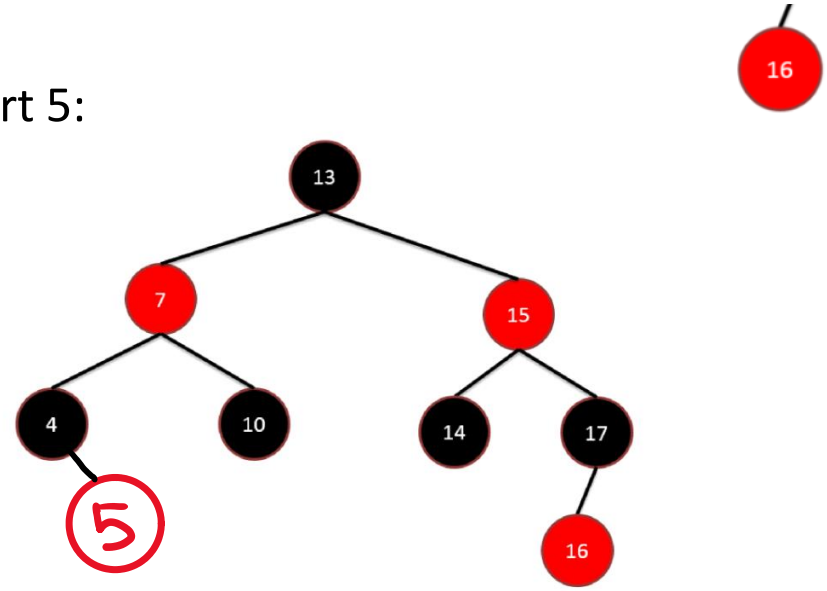
The best thing about a Boolean is that even if you are wrong, you are only off by a bit.

6. If we added the following nodes (5, 6, 8, 11, 12) to the Red-Black Tree shown below what would the resulting tree look like?



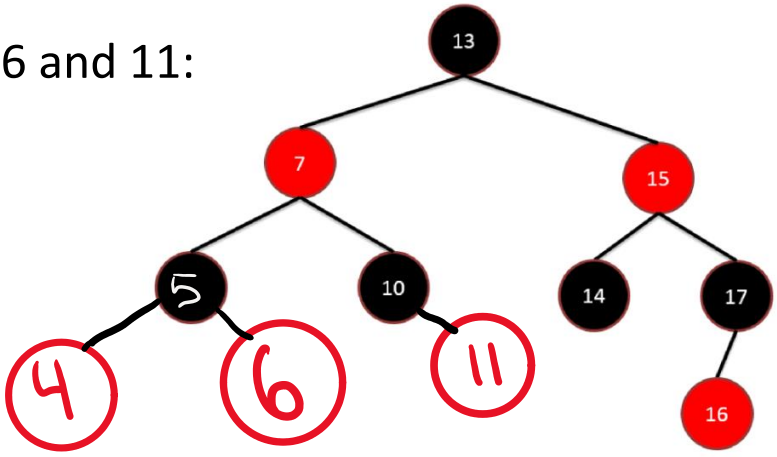
Insert 5:

Insert 5:



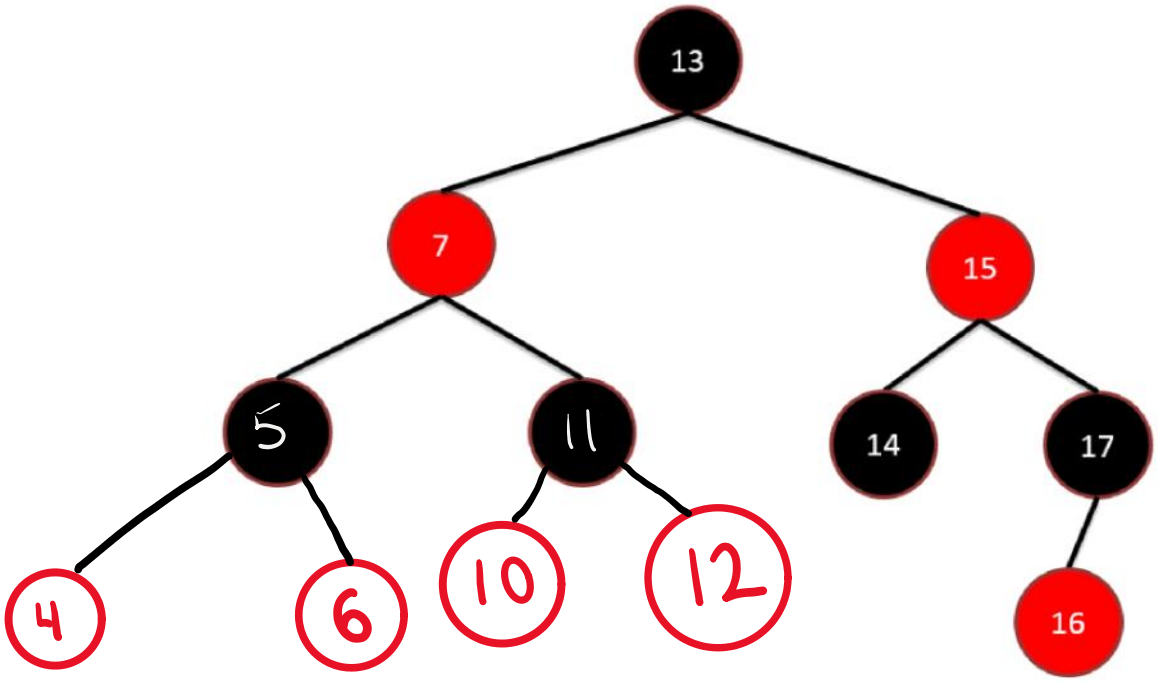
Why do Java Programmers wear glasses?
They don't C#

Insert 6 and 11:



Insert 12:

Insert 12:



7. Given a Hash Table size of 8 and the following values { 8, 15, 14, 12, 16, 10, 24, 32 } what would the resulting Hash Table look like if we used Linear Probing? Quadratic Probing? Separate Chaining? (**NOTE:** Assuming a simple modulus hashing function is used)

Linear Probing:

0	8
1	16
2	10
3	24
4	12
5	32
6	14
7	15

Quadratic Probing:

0	8
1	16
2	10

3

4	12
5	
6	14
7	15

7	15

Separate Chaining:

0	8	16	24	32

1

2	10

3

4	12

5

6	14
7	15

6

Eight bytes walk into a bar. The bartender asks, "Can I get you anything?"
"Make us a double."

8. We discussed in Coupling & Cohesion in lecture and their impact on software development. What are these two concepts and how are they related? What do we desire when building robust and highly successful software?

- Coupling: dependency of one class on another.
- Cohesion: the degree at which the component of an element belongs together

- Coupling = between and cohesion = within

We desire to have low coupling and high cohesion when building robust software.

building robust software.

Merry Christmas & Happy New Year!

7

```
["hip","hip"]
```

Notes for project 3

Thursday, October 8, 2020 10:26 AM

hiearcy: Node → Dictionary

BitVector

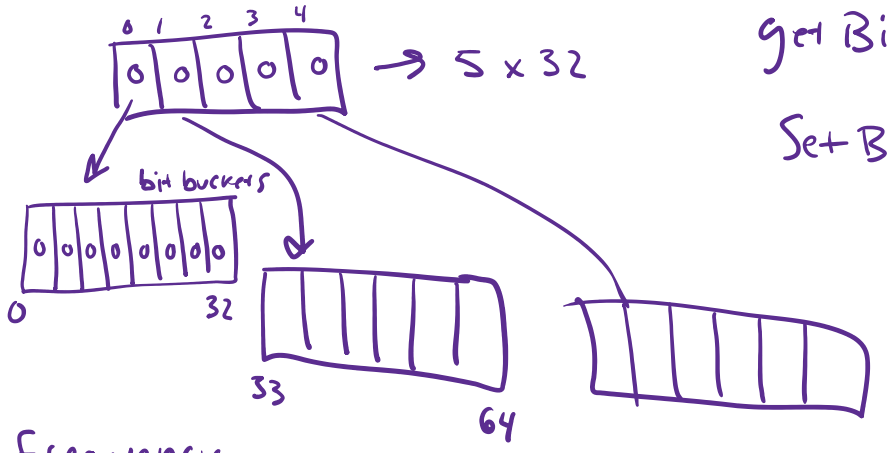
- default constructor for node
  Should be protected

- Dictionary accesor methods
  What should their parameters be?

- Dictionary "entries"

BitVector:- Use array to store bits

int x 10001000



getBit(15) 1

setBit(4) 1

Frequency

- Use a vector
- if the item shows up in duplicates, its already 2

Syllabus

Sunday, August 23, 2020 1:00 PM

Course Syllabus
CS 341 - Advanced Data Structures
Fall 2020

Instructor: Dr. Ryan Rybarczyk
Credit Hours: 3
Class: Tuesday/Thursday 1:00 PM – 2:15 PM (FC 148)
Office: FC 144
Office Hours: Tuesday/Thursday 10:30 AM – 12:00 PM
OR
By Appointment/Zoom
Email: rrybarcz@butler.edu

Description: This course will explore the implementation and use of the Object-Oriented (OO) Programming Paradigm and Advanced Data Structures in the C++ programming language. We will focus on the creation of classes and objects in C++ and compare/contrast this process with Java (as taught in CS 248) – including how to create and use makefiles to compile and run your programs. We will use revision control, in the form of GIT, in order to manage our code as we progress through the semester. Through our exploration of C++ we will discuss advanced memory management techniques (heap and stack) and the use of pointers (including pointer arithmetic). We will also explore Advanced Data Structures including linked lists, hash tables, binary search and trees. We will conclude the course with exploration of a trie in C++ and explore how this data structure compares to other common data structures that we already know. This course will be programming intensive.

Textbook(s):
- Data Abstraction & Problem Solving with C++ (7th Edition), ISBN: 978-0134463971

Course Objectives: The primary goal of this course is to provide an in-depth exploration of advanced data structures and advanced memory management techniques.

Students should expect to leave this course with a better understanding of the role that memory management plays in a programming language and the role that advanced data structures play in successful programming. By the end of the course, students will have developed an understanding of the following main topics:
- Be able to compare/contrast two Object-Oriented (OO) languages (C++ & Java)
- The knowledge of how to use common tools (code analysis tools, debuggers, makefiles) to create robust programming solutions
- The knowledge of advanced memory management techniques
- Be able to identify/debug/avoid memory leaks in code
- Understand what a pointer is in C++ and how it can properly be used
- Understand how to create a Class and use an Object in C++
- Be able to implement a Linked List in C++
- Be able to implement a Hash Table in C++
- Be able to implement a Binary Search Tree in C++
- Be able to implement an Red Black Tree in C++

CS 341 – Fall 2020 Tentative Schedule

Date	Topic	Assignment
8/25/2020	Course Introduction	
8/27/2020	Course Introduction & C++ Review	Assignment #1 Due: 9/10/2020
9/1/2020		
9/3/2020	Makefiles & GIT	
9/8/2020		
9/10/2020	Object-Oriented Programming: C++	Assignment #2 Due: 9/24/2020
9/15/2020		
9/17/2020		
9/22/2020	Arrays	
9/24/2020		Assignment #3 Due: 10/10/2020
9/29/2020	Pointers	
10/1/2020		
10/3/2020	Midterm Exam Review	
10/8/2020	**Midterm Exam**	
10/10/2020	Advanced Data Structures: Hash Tables	Assignment #4 Due: 10/24/2020
10/15/2020		
10/17/2020	Advanced Memory Management	
10/22/2020		
10/24/2020	Advanced Data Structures: Red-Black Trees	Assignment #5 Due: 11/5/2020
10/29/2020		
10/31/2020	Advanced Data Structures: Trees & Traversals	
11/3/2020		
11/5/2020	Advanced Data Structures: Trie	Assignment #6 Due: 11/24/2020
11/10/2020		
11/12/2020	Advanced Data Structures: Runtime Analysis	
11/17/2020		
11/19/2020		
11/24/2020	Final Exam Review	
12/7/2020	**Final Exam: 11:30 AM – 1:30 PM**	

Notes 1

Tuesday, August 25, 2020 1:03 PM

What is Data Abstraction?

- *What is the OO Concept?*
  - It really boils down to the idea that we can model things in software using terminology that we use to describe the same "objects" in the domain world.
- Pillars of OO Programming Paradigm:
  - Inheritance
  - Polymorphism
  - Data Abstraction
  - Encapsulation
- Encapsulation
  - Objects combine data and operations.
- Inheritance
  - Classes inherit properties from other classes.
- Polymorphism
  - Objects determine appropriate operations at execution time.
- Abstraction
  - ?

What is Abstraction???

- Separate purpose of a module from its implementation
- Specifications do not indicate how to implement
- Able to use without knowing implementation

Book Notes: Ch.1

A module is a self-contained unit of code and could be a single, stand-alone function, a class method, a class itself, a group of several functions or classes that work closely together, or other blocks of code

Note: ADTs versus data structures

- An abstract data type is a specification for a group of values and the operations on those values.
- A data structure is an implementation of an ADT within a programming language.

A class combines the attributes —or characteristics—of objects of a single type together with the objects' operations— or behaviors—into a single unit. The individual data items specified in a class are called data members. The operations specified in the class are referred to as methods or member functions . Attributes are typically data, and the behaviors, or methods, often operate on that data. In programming languages such as C++ and Java, classes specify the attributes and operations for the objects.

Encapsulation is a technique that hides inner details. Whereas functions encapsulate behavior, objects encapsulate data as well as behavior.

Note: A guiding principle of OOD is that each class should have a single, well-defined responsibility. The methods of a class should be highly cohesive and related directly to supporting the responsibility of the class. The responsibilities of a class are functionally equivalent to the tasks that the class needs to perform. If a class has too many responsibilities, it should be split into multiple classes, each with a single responsibility taken from the original class.

Note: An operation contract completely specifies a module's purpose, assumptions, input, and output.

Note: The program component that uses a module is the module's **client**. The **user** is a person who uses a program.

Note: A first draft of a module's specifications often overlooks or ignores situations that you really need to consider. You might intentionally make these omissions to simplify this first draft. Once you have written the major portions of the specifications, you can concentrate on the details that make the specifications complete.

Consider now a collection of data and a set of operations on the data. The operations might include ones that add new data to the collection, remove data from the collection, or search for some data. **Data abstraction** focuses on what the operations do with the collection of data, instead of on how you implement them. The other modules of the solution "know" what operations they can perform, but they do not know how the data is stored or how the operations are performed.

Book Notes: C++ Classes:

- Class templates: Allow you to def
  that are independent of the type
  Stored in the data structure
- Header files partially separate the d
  from its implementation.
- A constructor allocates memory for new insta
  initialize the object's data to specified value
  an instance of a class when the object's li
- To provide a public interface for an
  an abstract base class. This allows the
  advantage of polymorphism.
- Virtual Method: tells the compiler tha
  executes is determined at runtime. W
  - Declaring ADT methods as virtual all
    using our class to take advantage of Pol
    methods are invoked.
- A pure virtual method is a virtual
  implementation. An abstract class is one that
- Abstract classes cannot be directly insta

C++ Review

```
class Rectangle
{
        int width,height;
    public:
        Rectangle(int,int);
        int area()          ;
        {
            return width*height;
        }
};
```

```
class Rectangle
{
        int width,height;
    public:
        Rectangle(int x,int y);
        int area();
```

```
#include <iostream>
class Rectangle
{
        int width,height;
    public:
        Rectangle();
        Rectangle(int w,int h)
        {
            width = w;
            height = h;
        }
        void printArea()
        {
            std::cout << width*height;
        }
};
```

every rectangle has w,h → properties

→ default constructor

Object-Oriented Programming

- Ok – so when we first learned how to write a C++ program we put all of our code into a file with an extension of .cpp.
- But before when we created the file we did not create something with the name class or wrap it in { }
  - No less the crazy semi-colon there at the end of the file...
- With Java everything was self-contained in the same file – but we mentioned that in C++ we should separate the definition from the implementation.
- How can this be achieved?
  - Header (.h)/Source (.cpp)
- What do we separate and how do we format our code?
- How do we separate our definitions from our implementation?
- What would we put in our Header file in the previous slides code?
- What would we put in our Implementation file in the previous slides code?

C++ Header File (.h)

```
class Rectangle
{
    private:
        // Private Members
        int width,height;

    public:
        // Default (void) Constructor
        Rectangle();

        // Another constructor…
        Rectangle(int w,int h);

        // Print Area Function
        void printArea();
};
```

C++ Source File (.cpp)

```
#include "Rectangle.h"

// Default Constructor
Rectangle::Rectangle()
{
    // Empty Constructor
}
```

- What do we see here?
  - What can we say about what is going on?
- How is this different than the previous example we saw?

- Clone -> Bring a repository that is hosted somewhere like Github into a folder on your local machine
- add -> Track your files and changes in Git
- commit -> Save your files in Git
- push -> Upload Git commits to a remote repo, like Github
- pull -> Download changes from remote repo to your local machine, the opposite of push

Book Review: Ch.1

- Encapsulation: hide the inner details of functions, methods and objects.
  Functions and methods encapsulate behavior, and objects (instances of a class) encapsulate data as well as behavior.
- Inheritance: allows you to reuse already defined classes by extending their
  slight modifications

ine classes
e of data

esign of a class

nces of a class and can
s. A destructor destroys
fetime ends.

ADT, you can write
client to take full

the code this method
hy use one??
ows an application
ymorphism when the ADT's

method that has no
has at least 1 pure virtual methods.

ntiated

functions and methods ... encapsulate data as well as behavior.

- **Inheritance:** allows you to reuse already defined classes by extending their definitions or making slight modifications.
- **Polymorphism:** objects determine appropriate operations at execution time.
- **Data Abstraction:** technique for controlling the interaction between a program and its data structures. It builds walls around a program's data structures, just as other aspects of modularity build walls around a program's algorithms. These walls make programs easier to design, implement, read and modify.
- **Coupling:** a measure of the dependence among modules. Modules should be loosely coupled. A function or method should be as independent as possible.

- For problems that primarily involve data management, encapsulate data with operations on that data by designing classes. Practice abstraction: focus on <u>what</u> a module does instead of <u>how</u>.

ADT: collection of data and a set of operations on the data.

C++ Review:

- Class templates allow you to define classes that are independent of the type of data stored in the data structure. When a client is ready to instantiate an object of the class, the client can specify the type of data the object holds.
- Header (.h) files partially separate the design of a class from the implementation.
- To provide a public interface for an ADT (abstract data type), you can write an abstract base class, separating design from implementation and this allows us to take advantage polymorphism when using our class.

dependency lines
Skill lines
rules
inference lines
macros
comments

Notes 2

Thursday, September 10, 2020 12:51 PM

Makefile:

- Comments
- Rules
- Dependency lines
- Shell lines
- Macros
- Inference Rules

Templates:

Code example:

```
template <typename T, typename T2>
T2 Subtract_num(T a, T2 b)
{
  return a-b;
}

int main()
{
  int j=7;
  double i=15.0;
  cout << Subtract_num(i,j);
  return 0;
}
```

Abstraction:

- Separates purpose of a module from its implementation
  "Black Box"
- Abstraction Types:
  - Functional (Procedural)
  - Data

* Functional Abstraction
- Separating the purpose of a module from its implementation
  - Car/Engine example

* Data Abstraction
- What the operations do w the collection of data
- Know what operations can be performed, but do not know how the data is stored or how the operations are performed

Ask – What, not how
treat them as black box

Pointers:

- nullptr
  - allows you to specify that a pointer is essentially pointing at nothing.
    #include <cstddef>
  - if we initialize a pointer and then null it we have a memory leak. Why?

Ternary operator:
"?" – combines if & else in one line
example:

```
a < b ? b : a;
```

if → then → else

The stack is limited

Heap is dynamic
Some things are better on the heap.

Pointers: points to a place in memory

```
int *a[ ];
std::cout<< *a;
```


```
Polygon * p1 = new Rect();
        ⋮
auto p2 = p1;
```

Auto Keyword

Forces compiler to declare type.

Data Members

- Members should always be private.
  - Prevents unwanted access
  - Create getters/setters in order to access them.

Hint: node should have a value in the constructor (Should be private).

2 pieces of ADT:
- operations/methods
- attributes/data

Every function you write this semester should follow this format –
- Comment/Description
- Precondition
- Postcondition

Array-Based Implementation:

- arrays are homo
- arrays are a fixed size
- arrays are of one type

- Fixed-Size Array
  - Fixed maximum size

Valgrind: tool to check for memory leaks

C++ Review: Abstract Data Types

- The ADT Lifecycle:
  1. **Construction:** The ADT is created.
     - Locally on the stack.
     - Dynamically on the heap.
  2. **Initialization**: The variables of the ADT are set to their initial value.
     - Set the ADT's initial state.
  3. **Usage**: The ADT is now ready for use in program.
     - Invokes its member functions.
  4. **Destruction**: The ADT and it's memory is no longer usable.
     - Removed from the local stack.
     - Memory released back to heap.

C++ Review: Type Inference

- How do we allow for type inference in C++ templates?
  - C++11 Standard
- `Auto` Keyword
  - If the compiler can infer the type of a variable at the point of declaration you can use `auto` instead of the type name.
    - `int x = 4;` VS. `auto x = 4;`
- Benefit:
  - Cleaner, more readable, code.
- Consequence:
  - More readable?

Vectors:

- take the best of both worlds
- No need to manage our memory
- Vector is dynamically sized
- We can treat it like an array

Vectors

- We can even access our vector in a similar syntax as if it were an array in C++:

```
v1[10];
```

- The [] operator is present on the vector Class.
  - It also provides the method `at()` which provides the same functionality with the additional exception thrown if it is out of range (`std::out_of_range`)

Valgrind: tool to check for memory leaks

Valgrind test: type "Valgrind" space "--log-file=" "valgrind.txt" Ex1.ex

What a memory leak looks like:

Check Valgrind.txt
look for
no leaks are possible

B
D
D
B

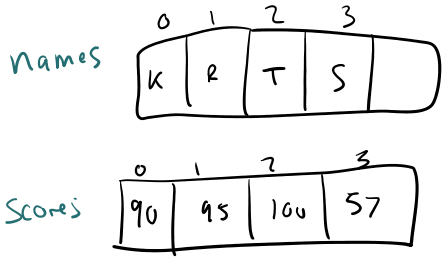
Delete

- The [] operator is present on the vector Class.
  - It also provides the method `at()` which provides the same functionality with the additional exception thrown if it is out of range (`std::out_of_range`)

Vectors

- In order to accomplish our goal of storing both the names and the scores we will need to create two different vectors.
  - Remember vectors are a templated class.

```
std::vector<std::string> names;
std::vector<double> scores;
```



names.at(0);
Scores.at(0);
names.push_back("D")

Bitwise Operations

```
#include <iostream>

int main()
{
	unsigned int x = 15;
	unsigned int y = 87;

	int z(0);

	z = x & y;

	std::cout << "Bitwise AND: " << z << std::endl ;

	return 0;
}
```

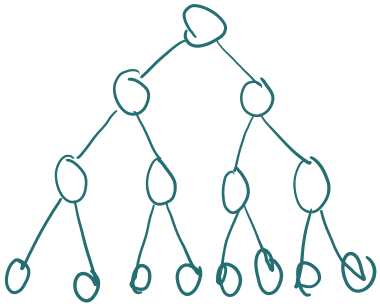
Bitwise Operations

- Left Shift Operator (<<)
  - Shifts the bits to the left by the number of positions specified by expression.
- Right Shift Operator (>>)
  - Shifts the bits to the right by the number of positions specified by expression.

Notes 3

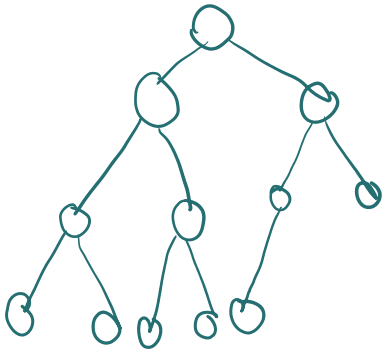
Thursday, October 22, 2020 12:14 PM

Binary Trees:



- **Full Binary Tree:** Assume we have a tree of height h, all nodes are at a level less than h have two children each
- Proof: If T is empty, T is a full binary tree of height 0. If T is not empty and has a height h > 0, T is a full binary tree if its root's subtrees are both full binary trees of height h-1
- **Full Binary Tree:** A full binary tree of height h >= 0 has 2^h -1 nodes. You cannot add nodes to a full binary tree without increasing its height. The maximum number of nodes that a binary tree of height h can have is 2^h-1.

A Binary Tree is a full binary tree if every node has 0 or 2 children



- **Complete Binary Tree:** A binary tree that is full down to level h-1, with level h filled in from left to right. 1. All nodes at level h-2 and above have two children each. 2. When a node at level h-1 has children, all nodes to its left at the same level have two children each. 3. When a node at level h-1 has one child it is a left child.

A Binary Tree is a complete Binary Tree if all the levels are completely filled except possibly the last level and the last level has all keys as left as possible

- **Balanced:** If the height of any node's right subtree differs from the height of the node's left subtree by no more than 1

- **Red Black Tree:** a type of balanced BST

A red black tree with n internal nodes has a height of at-most $2\log(n+1)$

Hashing

- Many different ways we can map a key to a specific "bucket."
- One of the most common and the simplest is to use modulus.
- Our hashing function will be calculated as follows:

```
key % size_of_hash_table
```

Hashing

- When we want to insert a new element into a Hash Table we need to convert our key into the index of the Array.
- This conversion process is known as **hashing** and the index of the Array is known as the **hash value** of the key.

Hashing

- The Hash Function in the ideal situation is called a **perfect hash function**. → No collisions
  - This function maps each search key into a unique location of the hash table.
- A perfect hash function is only possible if you know all of the search keys.
  - What is the likelihood of this occurring?

Collisions

- Selecting Digits:
  - You select specific digits in your search key to pick the index that the item should be inserted into.
- Problem:
  - Does not distribute the entries evenly in the Hash Table.

Collisions

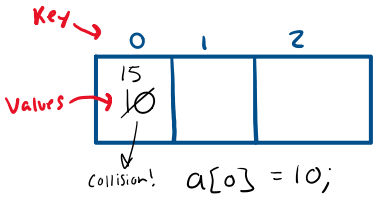
- Modulo
  - Computing the modulus of the table size and placing the entry into that position.
    - This eliminates one of the problems with respect to folding.
- Problem:
  - See the same explanation that we saw with folding in terms of multiple collisions – this approach tends to be a popular solution.

Linear Probing

- You search the Hash Table sequentially, starting from the original hash location until we find an open, unoccupied, spot/bucket to place the entry into.
  - If necessary you can "wrap around" to find an open spot

Hashing

- Let's take a look at an example…
- What is the simplest Hash Table?
  - An Array of records.
- We have a key and we can get the value – in theory the index of the element is the key.
  - Integers make great keys – but anything can be a key.



- If a collision occurs in an Array, its simply overridden
- If a collision occurs in a hash table, we need to take care of it!

Hashing

- Now the key is how do we select a hash function that will allow us to place our element into our "hash table?"
- We could use the modulus of the key to find the appropriate "bucket" or Array indices.
  - This is a good approach in the given approach as it will allow us to manage our Array.

Collisions

- How do we resolve a collision when one happens? – override it
- The easiest solution would to just move forward until we find an open position within our Array.
  - Now what do we do in the above scenario where our key of 152 is not located in Bucket #2?

Collisions

- Folding:
  - Here we could add up all of the digits of our search key and then place the entry in the summation index.
- Problem:
  - We still have the potential for future collisions and you may have to "fold" a number of times before resolving the collision.

Collisions

- Approach 1: Open Addressing
  - Linear Probing
  - Quadratic Probing
  - Double Hashing
  - Increase size of Hash Table

Linear Probing

- `Insert(x)`
  - Keep probing until an empty slot is found. Once an

Coupling & Cohesion

- Coupling = between
- Cohesion = within

we find an open, unoccupied, spot/bucket to place the entry into.
  - If necessary you can "wrap around" to find an open spot.

- Removals present a challenge however…

Linear Probing

- Solution:
  - Occupied
  - Empty
  - Removed

→ if the bucket is occupied, compare the value

  - Removed:
    - This status tells us to keep looking as when we originally inserted the value it had to use a different hash to find an open position.

Quadratic Probing

- We can avoid the problems with clustering in Linear Probing by moving towards a quadratic approach.

- The same approach that we saw in Linear Probing is found here, except we search quadratic for an open position.

Quadratic Probing

- Quadratic Probing:
  - Collision Scheme

$$\texttt{hash(x)} = \texttt{hash(x) \% size} + n^2$$

Collisions

- Approach 2:
  - Resolving collisions by restructuring the Hash Table.

- Methods:
  - Buckets
  - **Separate Chaining**
    - Singularly Linked List
    - Doubly Linked List

- `Insert(x)`
  - Keep probing until an empty slot is found. Once an empty slot is found, insert key.
- `Search(x)`
  - Keep probing until slot's key doesn't become equal to x or an empty slot is reached.
- `Delete(k)`
  - Keep probing until the keys slot is found and remove the key.
    - More on this later, as it is quite complex…

Linear Probing

- Occupied:
  - This indexed position is currently held by an entry in the Hash Table.

- Empty:
  - This indexed position has never been used by an entry.

- Removed:
  - This indexed position previously held an entry (Occupied) but is now available.

Quadratic Probing

- Let `hash(x)` be the slot index computed using the hash function – remember we are using key % size.
  - If the bucket `hash(x) % size` is `OCCUPIED`, then we try `(hash(x) + 1*1) % size`.
  - If `(hash(x) + 1*1) % size` is also `OCCUPIED`, then we try `(hash(x) + 2*2) % size`.
  - If `(hash(x) + 2*2) % size` is also `OCCUPIED`, then we try `(hash(x) + 3*3) % size`.
  - This process is repeated for all the values of `i` until an empty slot is found.

Notes for Exam

Monday, December 7, 2020 7:52 AM

Linked List vs. Vectors

- Vectors:
  - Good:
    - Random read access and insertion and deletion at the end.
  - Bad:
    - Insertions and deletions in the front or any other position within the Vector.

Linked List vs. Vectors

- Linked Lists:
  - Good:
    - Inserting and deleting items in the front, end, or middle.
    - Address of Node does not change – just update the link.
  - Bad:
    - Random element access.

Inorder Traversal

- An Inorder Traversal will cause all the nodes to be visited in **ascending order**.

- Inorder Traversal (Recursion):
  - Traverse Left Subtree
  - Visit the Node
  - Traverse Right Subtree

Preorder Traversal

- A Preorder Traversal:
  - Visit the node
  - Traverse the Left Subtree
  - Traverse the Right Subtree

Postorder Traversal

- A Postorder Traversal:
  - Traverse the Left Subtree
  - Traverse the Right Subtree
  - Visit the node

Red-Black Trees

Explain the difference between a List and a Tree in terms of their structure of the data:

Linear vs Hierarchical

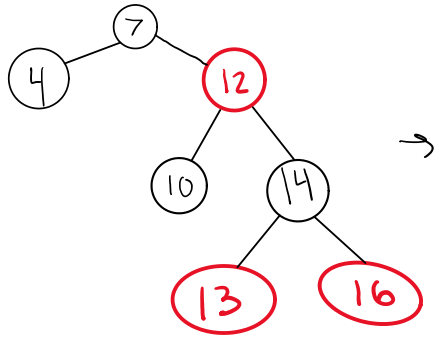
What are the three key properties of a Binary Tree?

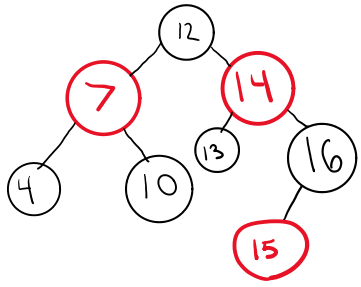
For each node N,
1. N's value is greater than all values in its left subtree
2. N's value is less than all values in its rights subtree
3. Both the left and right subtrees are BST's

What are the six key properties of a Red-Black-Tree?

1. A RB tree must be a type of BST
2. The root Node is black
3. Every Node when inserted begins as a RedNode
4. If a node is red, then its children must be black
5. Every path from a Node to a nullptr must contain the same number of black nodes.
6. Every leaf node must be black

Given the following Red-Black tree, what would the tree look like after inserting '15'





8,14,15,22,26,27 Quadratic:

0 14
1 8
2 15
3 27
4
5 22
6 26

8,14,15,22,26,27 Linear:

0 14
1 8
2 15
3 22
4
5 26
6 27

Red-Black Trees

- Rules:
  1. A Red-Black Tree must be a type of Binary Search Tree.
  2. The root Node is Black
  3. Every Node when inserted begins as a Red Node.
  4. If a Node is Red then its Children must be Black.
  5. Every path from a Node to a `nullptr` must contain the same number of Black Nodes.
     - This ensures that the tree will remain balanced.
  6. Every leaf (`nullptr`) must be Black.

Recursion Review

- Recursion breaks problem into smaller identical problems.
  - An alternative to iteration.
  - Divide & Conquer
- Anything we write with Recursion we can write with Iteration and vice versa.
  - Benefit?

Binary Trees

- Full Binary Tree
  - Assume we have a tree of height h, all nodes that are at a level less than h have two children each.
- Proof:
  - If T is empty, T is a full binary tree of height 0.
  - If T is not empty and has a height h > 0, T is a full binary tree if its root's subtrees are both full binary trees of height h - 1.

Trie

- Type of Search Tree
  - Prefix Tree
  - Ordered Tree
- Pronounced:
  - Try
    - It is really "tree" but that is very confusing since we already have Trees.

Trie

- A Trie is NOT a Binary Tree – a node can have MANY children.
- Goes hand in hand with a Dictionary or a Hash Table.
- We could have a root node or a list of root nodes.
  - Say What!?

Notes 5

Thursday, November 19, 2020 12:51 PM

What is Coupling?

- Coupling between classes is nothing but dependency of one class on another class.
- Coupling is one of the primary factors to consider in improving a design.
- Measures the strength of all relationships between functional units.
- Tied closely with cohesion (Low → High).

Low Coupling

- Reduces the impact of change between objects.
  - When one object is changed it does not "break" other objects.
- Problem:
  - Need to reduce dependency, change impact, and increase in reuse.
- Solution:
  - Evaluate alternate relations. Assign responsibilities so that the coupling remains low.

What is Cohesion?

- The degree at which the elements of a component belong together.
- Measure of strong relationships.
- Where should this component go?
- What is the role of this class and how does it fit into the overall system?

High Cohesion

- How functionally related are the operations of a software element are to one another.
- A measure of how much work a software element is “doing.”
- Keep similar things together
  - Otherwise, delegate the responsibility

Couple & Cohesion

- Highly Coupled programs have components that are dependent on each other (Low Cohesion).
- Loosely Coupled programs are made up of components that are independent or nearly independent (High Cohesion).
- Coupling = "**Between**"
- Cohesion = "**Within**"
- Ideally:
  - We want components to "stick together" and do their job, and only their job with unnecessary responsibilities delegated.

Trie Notes

- Type of Search Tree
  - Prefix Tree
  - Ordered Tree
- How do we search for a specific value within a tree?
- We could search a set for all values that start with the given prefix
  - We would have to keep a list of the retrieved values.
  - May require a search of the entire data structure - O(n)
- A Trie is NOT a Binary Tree - a node can have MANY children
- Goes hand in hand with a Dictionary or a Hash Table
- We could have a root node or a list of root nodes

- Root node is always null

$f(2+h) = (2+h)^2 - 2(2+h)$

$= 4 + 4h + h^2 - 4 - 2h$

$4h + h^2 + 2h$

$4h^2 + 10h + 4$

$f(2) = 2^2 - 2(2) \quad f(2) = 4$

$8 - 4$

$\frac{4h^2 + 10h + 8}{h} \rightarrow 4(.1)^2 + 10(.1) + 8 \quad 8 \quad 7.9004$

$\frac{0.04 + 1 + 8}{.1} \quad \frac{9.04}{.1} \quad \frac{0.0004 + .1 + 8}{.1}$

90.4

90.004

$.1^3$

$.001$

$2 + 10(.1) + 8$

$.06 + 1 + 8$

$\frac{C(105) - C(100)}{105 - 100}$

$C'(t) = 1.6x + 19$

$1.6(100) + 19$

$s' = v$

$s' = 9t^2$

$\frac{9(7)^2 - 9(2)^2}{7 - 2}$

$S(7) = 135$

$S(2) = 16$

$S(t) = t^2 + 6t$

$v(t) = 2t + 6$

$v(5) = 2(5) + 6$

16

$\frac{135 - 16}{7 - 2}$

$f(x+h) = -(4(x+h)^3 + 8(x+h))$

$4(x+h)(x+h)(x+h)$

$\frac{3}{\sqrt[4]{x}} \quad 3x^{-\frac{1}{4}} \quad -\frac{3}{4}x^{-\frac{1}{4}-\frac{4}{4}} = -\frac{3}{4}x^{-\frac{5}{4}}$

$-\frac{3}{4}\frac{-5}{4}$

$x = 3$

$y = 25$

$y = m(x - x_1) + y_1$

$y = 19(x - 3) + 25$

$y - (-2) = -7(x - (-2))$

$y = -7(x + (2)) + (-2)$

$-7x + 12$

$79t - 0.8t^2 = 0$

$v(t) = 79 - 1.6t$

$v(7) = 79 - 1.6(7)$

$S = -16t^2 + 120t + 10$

$\frac{f(1) - f(-3)}{1 - (-3)}$

$(-3)^2$

$-9 +$

$\frac{-12}{4}$

$-12 +$

$1 +$

$\frac{6x + 16}{x}$

$(6x + 16) \cdot 1 - (x)[6]$

$\frac{x^4 + 10x^3 - 10(2x) - x^2(4x^3 + 30x^2)}{(x^2)^2}$

$10 - 20 -$

$h(x) = 12f(x) + 11g(x) - 2$

$h'(2) = 12f'(2) + 11g'(2) - 2$

$(24) + 11(9) - 2$

$+ 97$

24

121

$-\frac{1}{2}(17)$

$\frac{1}{2} - \frac{2}{2} = -\frac{1}{2}$

$\left(-\frac{17}{2}\right)$

$8x^{-1} + 7x^{-2}$

$\frac{f(k)}{g(x)}$

$h'(x$

$h'(0$

$(5x)^4$

$4(5x$

$y = \frac{3}{2}x^{\frac{1}{2}}(x - 1) + 1$

x^{-2}

$-2x^{-2-1} = -3$

$-2x^{-3}$

$\frac{1}{\left(x^6 - \frac{6}{x}\right)^3}$

$y = \frac{1}{u^3}$

$\frac{dy}{dx} = -3u^{-4}\frac{du}{dx}$

$u = x^6 - 6x^{-1-1}$

$+6x^{-2}$

$= -3\left(x^6 - \frac{6}{x}\right.$

$18 - 5$

$-\frac{9}{4}$

$\frac{-16}{4} = -4$

$(+3)$

$' = 3x^2 - 10x$

$\frac{1}{8}$

$y - (-11) = \frac{1}{8}x +$

$) = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2} = \frac{(2)(5) - (5)(1)}{g(0)^2}$

$) = \frac{(5 \cdot 2) - (5 \cdot 1)}{5 \cdot 5} = \frac{10 - 5}{25} = \frac{5}{25} = \frac{1}{5}$

$= 3(x^2+2x)^{-5} \Rightarrow y = 3u^{-5}$

$\frac{dy}{dx} = -5 \cdot 3\, u^{-6} \frac{du}{dx}$

$u = (x^2+2x)$

$\frac{du}{dx} = 2x+2$

$\frac{dy}{dx} = -15(x^2+2x)^{-6} \cdot [2x+2]$

$y = (5x^2 + 2x + 6)^{1/2}$

$u = 5x^2 + 2x + 6$

$\frac{du}{dx} = 10x + 2$

$y = u^{1/2 - \frac{1}{2} = -\frac{1}{2}}$

$\frac{dy}{dx} = \frac{1}{2}u^{-1/2} \cdot \frac{du}{dx}$

$\frac{1}{2}(5x^2 + 2x + 6)^{-1/2} \cdot [10x + 2]$

$)^{-4} \cdot \left[6x^5 + \frac{6}{x^2}\right]$

$u = x^6 - \quad +6x^{-2}$ $= -3(x^6 - \frac{6}{x}$

$\frac{du}{dx} = 6x^5$

$a^4 - t^4 = 6a^2 t$

$4a^3 \frac{da}{dt} - 4t^3 \frac{dt}{dt} = 6a^2[1] - t[12a]$

$4a^3 \frac{da}{dt} - 4t^3 \frac{dt}{dt} = 6a^2 - 12at$

$\frac{-3(-2)}{(-1)} + \frac{1}{(-1)^2}$

$4(6x^2 - 3x + 5)^{-3}$

$u = 6x^2 - 3x + 5$

$\frac{du}{dx} = 12x - 3$

$y =$

$\frac{dy}{dx}$

-1

$\frac{2}{3}x^{-1/3} + \frac{2}{3}y^{-1/3} = 0$

$\frac{2}{3} + \frac{2}{9} \quad \frac{1}{3} \cdot \frac{2}{3} = \frac{2}{9}$

$y - 25 =$

$1770x - (8150 + 590x + 0.2x^2)$

$(8150 + 1180x + 0.2x^2)$

$\frac{67600}{x} + 800 + x$

$1180 + 0.4x = 0$

$0.4x = -1180$

$y = 1000$

$-\frac{67600}{x^2} + 1 = 0$

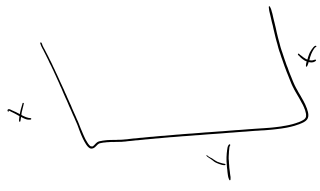
$10000 = 2x + y$

$y = 10000 -$

$A = x \cdot y$

$A = x(10000 - 2x$

$x^2 = 67600$



$A' = 10000 -$

$x =$

$-1 =$

$590 + 0.4x = 0$

$\frac{0.4x}{0.4} = \frac{590}{0.4}$

$f'(x) = 6e^{2x} + -5e^{-x}$

$f''(x) = 12e^{2x} + 5e^{x}$

$0 \quad 12e^0 + 5e^0$

$3 = A(.5)^{2/4}$

double time

17% per year

$P = 70 - 0.01x$

$C = 8000 + 5x +$

$(70 - 0.01x)x$

$70x - 0.01$

$0.02x^2 -$

$) \cdot [6x^2 + \frac{1}{x^2}]$

$4u^{-3}$

$= -12u^{-4}\frac{du}{dx}$

$2\left(6x^2 - 3x + \right)^{-4}(12x - 3)$

$\frac{4}{\sqrt{6x^2+6}} = 4(6x^2+6)^{-1/2}$

$-\frac{1}{2} + \frac{-2}{2} = \frac{3}{2}$

$y = 4u^{-1/2 - \frac{2}{2}}$

$u = 6x^2 + 6$

$\frac{du}{dx} = 12x$

$\frac{dy}{dx} = -2u^{-3/2}\frac{du}{dx}$

$-2(6x^2+6)^{-3/2} \cdot [12x]$

$00 - 2(2500)$

$2x$

$) = 10000x - 2x^2$

$4x$

2500

5000

$.03x^2$

$- 8000 + 5x + 0.03x^2$

$x^2 - 8000 + 5x + 0.03x^2$

$75x - 8000$

171. per year

$0.02x^2 -$

$0.04x$

$f(g(x))$

$f'(g(4)) \cdot g'(x)$

$f'(6) \cdot g'(5)$

$-5 \cdot 3$

$f'(x) = 3x^2 - 4x + 7$

$f(x) = 6x + 4$

$-6 + 4$

-2

$f'(4) = 0$

$f''(4) = -8$

$(f+g)'(4)$

$f'(x) + g'(x)$

$5 + 3 = 8$

$f(x) = \frac{3}{x^2}$ $g(x) = x^4 + 5$

$f(g(x))$

$\frac{3}{(x^4+5)^2}$

True

$$\int_{-1}^{6} f(x) - g(x)\, dx$$

$\left(\frac{g}{f}\right)'(7)$

$f(x)g'(x) - g(x)f'(x)$

$75x - 8000$

$-75 = 0$

$0.04x = 75$

$x = 1875$

$\left(\frac{f}{f}\right)'$

$$\frac{f(x)g'(x) - g(x)f'(x)}{f(x)f(x)}$$

$$\frac{(1\cdot -10) - (7\cdot 9)}{1} = -10 - ($$

$= -73$

Change #1,

leave the rest.

that should score a 96%.

$10000(1+$)

$y = \sqrt{x^2-3x+1}$ $\quad u = x^2-3x+1$, $du = 2x-3$

$= (x^2-3x+1)^{1/2}$

$y = u^{1/2}$ $\quad 1/2 - \frac{1}{2}$

$\frac{1}{2}u^{-1/2} \cdot du$

$y' = \frac{1}{2}(x^2-3x+1)^{-1/2} \cdot 2x-3$

5.)

Quiz 1

Tuesday, August 25, 2020 1:01 PM

CS 341 – Advanced Data Structures

Quiz 1

Name: N8 Swalley

1. What is your year in school?

   Sophomore

2. What programming languages have you used to write code?

   Java, C++, python(very little), C#(very little)

3. What language do you feel the most comfortable writing code in and why?

   C++ because I was able to complete ALL the CS142 projects AND understand what's going on in them. I can't say the same for Java. (I struggled with the GUI project)

4. How would you define ***what*** data abstraction is?

   Well I know that abstraction deals with ABSTRACTING: removing or deriving something. So my guess would be that Data Abstraction is a method of deriving the data in a program to do something useful...

   After the lecture: Abstraction in simple terms is using something without knowing how its actually implemented.

Assignment 1

Thursday, August 27, 2020 1:04 PM

CS 341 – Fall 2020
Assignment #1 – Keepin' It Classy
Due: 9/10/2020

This first assignment will serve as a refresher for the C++ programming language and allow you to become more comfortable with the tools we will be using this semester to accomplish our goals. This project will serve as the foundation for exploring the many benefits that the C++ programming language can provide and will help you re-explore Object-Oriented concepts that you previously learned in CS 248 (in Java) all the while learning Make and Git in the process.

For this assignment you will not be completing a full design and implementation, instead I will provide a skeleton of a project to begin as your base. This program, as specified by the client, is designed to examine the differences between various popular sorting algorithms. It has been designed to make use of Inheritance, multiple files, Objects/Classes, and therefore necessitates linking of files together in order to compile and run the solution. Your job is to produce a working program and to upload the solution to your GitHub repository.

A few notes about the specific requirements of the program:

- The program should load SPACE delimited numerical data from a text file.
    - The user should input the name of this file during program execution.
- The program should then give the user the choice between two different sorting techniques (in this case Insertion sort and Bubble sort).
    - The program should sort the numbers in descending order.
- The program should then prompt the user to enter the name of the file in which to store the resulted sorted list.
    - The sorted list should NOT be displayed in the terminal.
- The program should then prompt the user if they wish to continue ('Y') or to terminate ('N').
- The program should handle invalid cases (e.g., invalid text entry, file I/O, etc.).

Development Process:

You will assume that all of the existing code is correct – at least the code provided. You will need to Fork the project found at:

https://github.butler.edu/rrybarcz/sorting

This will be the code you use as a starting point for this assignment.

You will need to complete the Header and Source files for Insertion and Bubble sort Classes. You will then need to write a Driver that will test these two Classes and provide the necessary functionality as listed above. Once you have completed that task, you will need to create a Makefile that properly links all of this code together and creates an executable named **A1.exe**

For this assignment, all development must take place on the **master branch** in your GitHub repository. It is strongly recommended that you commit and push often! This will help to familiarize you with the workings of a source code repository and its importance in software design and development. We will be using this the entire semester – so best to become very well versed in it now!

All assignments must be submitted on Butler GitHub (github.butler.edu). This an individual assignment – meaning that each student should submit their OWN work. The name of your Butler GitHub repository must be as follows: **cs341_fall2020**

workings of a source code repository and its importance in software design and development. We will be using this the entire semester – so best to become very well versed in it now!

All assignments must be submitted on Butler GitHub (github.butler.edu). This an individual assignment – meaning that each student should submit their OWN work. The name of your Butler GitHub repository must be as follows: **cs341_fall2020**

The directory structure of the repository must contain the following files:

- **driver.cpp**
- **Insertion.h**
- **Insertion.cpp**
- **Bubble.h**
- **Bubble.cpp**
- **Sort.h**
- **Sort.cpp**
- **makefile**

Each source file (.cpp/.h) **must** include the Honor Pledge and digital signature – more details about this can be found in the lecture slides.

Assignment 2

Thursday, September 17, 2020 1:00 PM

CS 341 – Fall 2020
Assignment #2 – Hop Dawg
Due: 9/29/2020

Have you ever wondered how on your smart device (PC/Phone/Tablet/etc.) that a screen of apps can be displayed in order of most recently used? We know that there is only so much screen real estate and it makes sense from a usability aspect that we display those apps in groupings based upon their usage. One such solution for handling this task is to use a Doubly Linked List (DLL). A DLL is a Linked List in which each Node in the List is linked to both its predecessor as well as its successor. This means that you can traverse the List both forwards and backwards. The method for doing this is to maintain a list of the apps by their usage and to move or insert apps into the slots provided. Your second assignment will allow you to explore this concept of a DLL and explore how ordering and insertions can be done on such a data structure. This will build upon your existing knowledge of C++ and help reinforce the topics of Inheritance, Polymorphism, Pointers, and Memory Management that we have discussed in lecture.

For this project you are given a large data set and have been tasked with inserting new numbers into the data set…but how!? You really have two options at your disposal: you could simply insert the new number onto the head, or front, of the list OR you could insert the number onto the tail, the end, of the list (*we will ignore the third option which would allow you to "randomly" insert the number into the existing list, for now…*). For simplicity sake, we will leverage your familiarity with Arrays and assume that when building your DLL that inserts will be Tail Inserts, meaning that we will append the element to the end of the DLL.

Once you have built your DLL you will need to insert new numbers into the existing List. Blue IV (our loyal and faithful CEO) does not simply want the values inserted into the DLL – instead, his desire is for the elements to be inserted such that the insert can **only** be executed when the location is found where the number is followed by a larger value (think pseudo-sorted).

For example, given the following list:

```
15, 7, 10, 44, 54
```

And given the following number to insert:

```
34
```

With our two existing options (Head Insertion/Tail Insertion) we have the following two scenarios:

1. If you start at the head of the DLL: '34' is greater than '15', '7', '10' but not '44', so we would need to insert it between the '10' and '44' in the existing DLL.
2. If we start at the tail: '34' is not larger than '54' and it is also not larger than '44', but it is larger than '10', so we would insert it between the '10' and '44' in the list. In this case, we came to the same conclusions….what next!?

Now comes the catch…Blue IV has instructed us that we MUST select the optimal insertion, meaning the insertion that will require the **least** amount of hops. We define a "hop" to be measured by the number of comparisons needed in order to insert the number into the existing DLL. In our above example, Option #1 requires three (3) hops - while Option #2 requires only two (2) hops. Thus, we should insert the number '34' from the Tail of the DLL and not the Head, and thus we are left with the following DLL:

'34' from the Tail of the DLL and not the Head, and thus we are left with the following DLL:

```
15, 7, 10, 34, 44, 54
```

Note: If the steps required are the same (e.g., both head and tail insertion require two (2) hops) we will use the default of Tail Insertion.

Note: We will assume that if the value is equal to (==) the existing number in the DLL we will always place the new number AFTER the existing number.

Blue IV has given you two data files that you will need to load (`data.txt` and `sorted.txt`) and one insertion file (`inserts.txt`). You will need to run your program twice – **once** for `data.txt` and **once** for `sorted.txt`. In both instances you should insert the data elements found in the file `inserts.txt`. He would then like two output text files (`output.txt` and `sortedOutput.txt`) along with a Hop Count for **each** execution – the Hop Count should be stored in a `README.txt` file. We will assume that both input files contain only space delimited Integer entries.

A sample scenario is shown below:

```
Inserting 15...
Inserting 7...
Inserting 10...
Inserting 44...
Inserting 54...
15<-->7<-->10<-->44<-->54

                    (Inserting 34, 36, 1, 2, 2)

15<-->7<-->10<-->34<-->44<-->54
15<-->7<-->10<-->34<-->36<-->44<-->54
1<-->15<-->7<-->10<-->34<-->36<-->44<-->54
1<-->2<-->15<-->7<-->10<-->34<-->36<-->44<-->54
1<-->2<-->2<-->15<-->7<-->10<-->34<-->36<-->44<-->54

Total Number of Big Dawg Hops: 7
```

A few notes about the specific requirements of the program:

- The program should use proper Class structure and hierarchy – we will outline this in lecture.
  - `List` → `LinkedList` → `DoublyLinkedList`
  - `Node` → `LinkedNode`
- List Class:
  - Pure Virtual (Abstract) Class
    - `List() {}`
    - `virtual ~List() {}`
    - `virtual bool isEmpty() = 0;`
    - `virtual int getLength() = 0;`
    - `virtual void insert(int element) = 0;`
    - `virtual void printList() = 0;`

```
virtual int getLength() = 0;
virtual void insert(int element) = 0;
virtual void printList() = 0;
```

- LinkedList Class Attributes:
  - `Node * head_`
  - `Node * tail_`
- DLL Class Methods:
  - Only Insertions - No Deletions!
  - `DoublyLinkedList();`
  - `virtual ~DoublyLinkedList()`
  - `virtual void printList()`
  - `virtual void insertNode(Node * node, int data)`
  - `void insertAfter(Node * node, int data)`
  - `void insertBefore(Node * node, int data)`
- Node Attributes:
  - `int data_`
- LinkedNode Attributes:
  - `Node * nextNode_`
  - `Node * prevNode_`
- You should include a Constructor/Destructor in every Class.
  - Remember the impact of the `virtual` keyword!
- Your Driver (`driver.cpp`) should handle all File I/O.
  - No File I/O should be in any of your Classes.
- You must store your nodes (`Node`) and DLL on the Heap.
  - No Memory Leaks!
    - Use Valgrind: `valgrind --log-file=valgrind.txt A2.exe`

Development Process:

You **may** (but are not required to) work with a partner (groups of two (2) students) on this assignment. If you choose to do so, you are both expected to work on and maintain a single GitHub repository – be sure to add the appropriate collaborators. This will allow you to practice your skills of Forking, Cloning, and Merging using Git. Please let me know if you do plan to work with a partner on this project – if you do not notify me I will assume you are working independently on this assignment.

For the development of your code: you should create an Abstract Class called `List` and then inherit from `List` in your creation of your `LinkedList`. From there we will create another Class called `DoubleLinkedList`. This Class will lean on both `LinkedList` and `List` in order to accomplish its task. You will use OO pillar of Polymorphism to support this behavior. You will also need to create a `Node` Class and then inherit from it into a `LinkedNode` Class that will represent each element of the DLL. The `Node` Class will be linked (no pun intended) to the `LinkedList` Class via Aggregation (Composition). Tying this all together you will need to write a `driver` that will test this hierarchy and provide the necessary functionality as described earlier as outlined by Blue IV. Your driver program should allow for the user to input a space delimited text file (`.txt`) to be loaded and built into a DLL and then allow for the input of a second file (`.txt`) that will insert the new values into the DLL at the appropriate position with the final resulting list being output to another text file (`.txt`). Each Class, except `List`, should have a Header/Source file (`.h/.cpp`). Please include the Hop Count for each test in

should allow for the user to input a space delimited text file (`.txt`) to be loaded and built like a DLL and then allow for the input of a second file (`.txt`) that will insert the new values into the DLL at the appropriate position with the final resulting list being output to another text file (`.txt`). Each Class, except `List`, should have a Header/Source file (`.h/.cpp`). Please include the Hop Count for each test in a `README.txt` file. Finally, you will need to create a `makefile` that properly links all of this code together and creates an executable named **A2.exe**

I will be grading what is located in the **master branch** of your GitHub repository. It is strongly recommended that you commit and push often! If you opt to work on this assignment with a partner I will expect to see "equal" contributions in the Git history. Please make use of the Git feature of leaving descriptive messages along with your commits. Be sure to add me to any repository as a collaborator so I can view and grade your submissions. Failure to do so will result in a 0 on the assignment as I will have nothing to grade!

Submission:

All assignments must be submitted on Butler GitHub (github.butler.edu). The name of the repository should be: **cs341_fall2020_linked**

The directory structure of the repository must contain the following files:

- **driver.cpp**
- **List.h**
- **LinkedList.h**
- **LinkedList.cpp**
- **DoubleLinkedList.h**
- **DoubleLinkedList.cpp**
- **LinkedNode.h**
- **LinkedNode.cpp**
- **Node.h**
- **Node.cpp**
- **makefile**
- **README.txt**
- **sortedOutput.txt**
- **output.txt**
- **data.txt**
- **sorted.txt**

Each source file (`.h/.cpp`) **must** include the Honor Pledge and digital signature – in the case of a partner submission both digital signatures should be present.

Assignment 3

Tuesday, September 29, 2020 1:11 PM

CS 341 – Fall 2020
Assignment #3 – Duplicate Dawgs
Due: 10/13/2020

Have you ever been in a position where you have a large data set and you wish to know whether a specific number is a member of the set? Obviously, you could scan the list – but this is very expensive in terms of memory. Instead, you want to optimize this process and use your space efficiently. The Linux kernel actually makes use of this approach to provide optimization. This third assignment will explore a new data structure – the Vector, and see how we can leverage it to explore bit manipulation in C++ and provide such optimization.

For this assignment you will be analyzing a large data set consisting of Integers. Our faithful CEO (Blue IV) has again entrusted us with a task to improve his existing data. This time around Blue IV has given us a very LARGE data set comprised of Integers. These Integers are not ordered and are machine generated log entries for memory access. Blue IV wants to see who the biggest memory hogs are in the machine!

His task for you, is to find all the duplicate entries in the set of numbers. Sounds easy enough…but we know Blue IV, he may be cute and cuddly on the outside but he always is lurking with a catch just around the next corner and here it is: he has been reading up on this great new class at Butler, called CS 341 – Advanced Data Structures, and has learned about this concept of a Bit Vector. A Bit Vector can be defined as follows:

"A bit vector is a mapping from a domain to values in the set {0, 1}"

*Why a Bit Vector – by capitalizing on this mapping we can efficiently store the same information using the minimum number of bits necessary – this optimizes space consumption. The '0' represents the condition where the bit is NOT set and the '1' represents the condition where the bit IS set. The integers we are using required 32 bits of memory for storage.

Now Blue IV has thrown down the gauntlet – he wants us to accomplish all of this in 4 KB of memory (Why? "4 is my lucky number!"). If that wasn't enough (really Blue IV!?), he wants to not only have the list of duplicate values but also ORDER the new list by frequency – meaning that he would want to first element in the list to be the most frequently occurring number and so on and so forth. Are you up to the challenge!?

A few notes about the specific requirements of the program:

- The program should load SPACE delimited numerical data from a text file (`data.txt`).
- The program should then insert the data into a Bit Vector.
    - The duplicates should be identified and placed into a new vector.
- The program should then output to a file (`duplicates.txt`) a list of duplicates listed in descending order in terms of frequency.
- The program should handle invalid cases (e.g., invalid text entry, file I/O, etc.).
- The Bit Vector should be placed on the Heap (memory management).
    - The program should contain no memory leaks – make sure to use Valgrind!
        - `valgrind --log-file=valgrind.txt A3.exe`

Development Process:

For the development of your code: you should create `BitVector` and `Dictionary` Classes. The `BitVector` Class should have the following attributes and operations:

- Private:
  - `std::vector<int> * data_`
    - `// This will hold the data from the text file`
- Public:
  - Constructor(s)
  - Destructor
  - Accessor Method(s)
  - `bool getBit(int position);`
    - `// Gets the bit at the given position`
  - `void setBit(int position);`
    - `// Sets the bit at the given position`
  - `void findDuplicates(std::vector<int> data, std::vector<int> & duplicates)`
    - `// This will identify the duplicates via our bit vector and store the list of them in another vector`

For the `Dictionary` Class you should include the following attributes and operations (Note: We can reuse our `Node` Class here through Aggregation):

- Private:
  - `Node id_`
    - `// This will hold number entry`
  - `Node data_`
    - `// This will hold the frequency value`
- Public:
  - Constructor(s)
  - Destructor
  - Accessor Method(s)

Tying this all together you will need to write a `driver` that will test this hierarchy and provide the necessary functionality as described earlier as outlined by Blue IV. Your driver program should allow for the user to input a space delimited text file (`data.txt`) to be loaded and built into our Bit Vector and then with the final resulting duplicate sorted list by frequency being output to another text file (`duplicates.txt`). Finally, you will need to create a `makefile` that properly links all of this code together and creates an executable named **A3.exe**

An example of sample output is as follows, given the following sample input file:

```
29 7 37 47 16 34 22 47 7 3
```

Your `duplicates.txt` file would contain the following:

```
47 2
7 2
```

I will be grading what is located in the **master branch** of your GitHub repository. It is strongly recommended that you commit and push often! Please make use of the Git feature of leaving descriptive messages along with your commits. Be sure to add me to any repository as a collaborator so I can view and grade your submissions. Failure to do so will result in a 0 on the assignment as I will have nothing to grade!

I will be grading what is located in the **master branch** of your GitHub repository. It is strongly recommended that you commit and push often! Please make use of the Git feature of leaving descriptive messages along with your commits. Be sure to add me to any repository as a collaborator so I can view and grade your submissions. Failure to do so will result in a 0 on the assignment as I will have nothing to grade!

Submission:

All assignments must be submitted on Butler GitHub (github.butler.edu). I will allow you to work on this assignment with up to one (1) other person. If you choose to work on this project with a partner you need to email me letting me know of your "group." Be sure to make note of specific contributions of each team member so I can assign grades accordingly. **Note:** You are NOT required to work with a partner. The name of your Butler GitHub repository must be as follows: **cs341_fall2020_bits**

The directory structure of the repository must contain the following files:

- **driver.cpp**
- **BitVector.h**
- **BitVector.cpp**
- **Dictionary.h**
- **Dictionary.cpp**
- **Node.h**
- **Node.cpp**
- **data.txt**
- **duplicates.txt**
- **makefile**

Each source file (`.cpp/.h`) **must** include the Honor Pledge and digital signature – if working in pairs, please ensure both students' digital signatures are present on the files.

Helpful Hints:

Bitwise Operations

& (Bitwise AND)	Takes two numbers as operands and performs an AND on every bit.
| (Bitwise OR)	Takes two numbers as operands and performs an OR on every bit.
^ (Bitwise XOR)	Takes two numbers as operands and performs a XOR on every bit.
<< (Left Shift)	Takes two numbers and left shifts the bits of the first operand by the number of places specified by the second operand.
>> (Right Shift)	Takes two numbers and right shifts the bits of the first operand by the number of places specified by the second operand.
~ (Bitwise NOT)	Inverts all the bit of one operand.

- The Dictionary Class is responsible for holding each duplicate number and then their frequency. Think of it as literally a dictionary – you look up a word and get its definition. Here, you look up a number and get its frequency. It is really just a container for two different but related pieces of data.
- How do we represent a LARGE data set using Bit Vectors???? Here is the trick – you maybe thinking well an INT isn't

sufficient as an INT only has 32 bits…but what if we had an ARRAY of INT's? Now we could have an array of say five (5) integer values stored in the array and thus we actually have 160 bits…what about an array of size 100? Or 1000? Blue IV told us that we have 4 KB available to use, so we actually have 32 * 2^10 which is 32,768 bits. That means in that 4 KB we could actually index over 32k integers...but if we didn't use a Bit Vector in order to perform the same analysis on 32k integers we would need 125 KB.

Hopefully everyone is making good progress and can use this as a helpful aid in your development. If you have any questions please don't hesitate to ask. The only bad question is the question unasked.

From <https://butler.instructure.com/courses/14064/discussion_topics/71172>

- In your constructor you will need to initialize your Array/Vector to a specified size - this size has to accommodate all the necessary data values...so it will be a size. I selected 320000 as the size of my array - I will explain why in just a moment - this will tell us how many values we can store in our bit vector. When you initialize your Array the question becomes how do I know when to move from one "space/bucket" to the next - well we know how big an INT is in C++ - as it is called INT 32 (4 bytes) - so we can do the following:

  data_ = new int[right shift by 5 (think binary representation here to understand why 5) + 1] <-- for INT 32 (2^5)

- In your getBit function, you pass in a position p and you can find the index of our bit vector by right shifting the position p by 5 again...this moves us through the actual array. Once we have that we can get the respective value of the bit based upon this index position. Remember this will be either a '0' or '1'.
- In your setBit function, same as the getBit function you need to use the position and right shift to get the respective array index. You can then use Bitwise OR to either flip the bit - meaning it is 0 or if it is already 1 then we just leave it as is. Remember we will need to left shift here as we are working "backwards" from the end of the bit vector (e.g., 0001 = 1....1000 != 1).

From <https://butler.instructure.com/courses/14064/discussion_topics/72857>

Assignment 4

Saturday, October 17, 2020 7:55 PM

CS 341 – Fall 2020
Assignment #4 – Color Me Dawgs
Due: 10/29/2020

You are tasked with creating a binary tree but told that it must remain balanced for optimal searching – how can you accomplish this?

Red-Black Trees are self-balancing binary trees. It achieves this by coloring the nodes of the tree to ensure that insertions and deletions are handled in a balanced sense. The really nice benefit of a Red-Black Tree is that it provides a worst-case guarantee on insertion, deletion, and search times. This is great for real-time applications with strict requirements. On top of that by ensuring balance – they are ideal for searching and provide a significant benefit over using a Linked (or Doubly Linked) List.

For this assignment you are tasked with creating a Red-Black Tree. Blue IV (our faithful CEO) has heard about such a data structure and wants to use it to model athletic roster numbers (here we assume that no double numbers are possible). Blue IV wants to know what the root node of our tree is after all insertions and which nodes are RED and which nodes are BLACK after its all said and done. He has given us a text file that contains athletic roster numbers (integers) and our job is to read-in this file and build a tree from it. He only wants us to deal with insertions and searches for this project – no deletions. Just a reminder (discussed in lecture) a Red-Black Tree has the following properties:

- A Red-Black Tree must be a type of Binary Search Tree
- Every node has a color either RED or BLACK
- The root of the tree is always BLACK
- When a node is added to the tree it begins its life as a RED node
- There are no two adjacent RED nodes
  - A RED node cannot have a RED parent or a RED child
- Every path from a node to any of its descendant (down to the `nullptr`) has the same number of BLACK nodes

A few notes about the specific requirements of the program:

- The program should load SPACE delimited numerical data from a text file (`data.txt`).
- The program should then insert piece of data into a Tree.
  - The Nodes should contain a pointer to their left child, right child, and parent.
  - The Nodes should be assigned an initial color of RED.
- The program will need to ensure balance of the tree by shifting Nodes either left or right in order to maintain optimal ordering.
  - By doing this, Nodes may need to have their color updated to reflect their new position within the Tree to ensure adhering to the Red-Black Tree rules.
- You should display (output) into the console window the following information to the user:
  - True Root
  - Red Nodes (Inorder Traversal)
  - Black Nodes (Preorder Traversal)
- The program should handle invalid cases (e.g., invalid text entry, file I/O, etc.).
- The Tree should be placed on the Heap (memory management).
  - The program should contain no memory leaks – make sure to use Valgrind!
    - `valgrind --log-file=valgrind.txt A4.exe`

Development Process:

For the development of your code: you should create three new classes: `RedBlackTree`, `BinaryTree` and `TreeNode`. We will reuse our existing `Node` class – no changes are needed for it.

The `TreeNode` Class should inherit publically from our `Node` Class. This will allow us to access the attributes and methods of that Class to store our data for our tree nodes. The `TreeNode` Class should have the following attributes and operations:

- Private:
  - `TreeNode * leftChild_;`
  - `TreeNode * rightChild_;`
  - `TreeNode * parentNode_;`
  - `Color color_;`
    - This is an Enumeration type in my instance – but you could make it a Class if you so desire.
- Public:
  - Contructor(s)
  - Destructor
  - Accessor Method(s)

The `BinaryTree` Class should have the following attributes and operations:

- Private:
  - `TreeNode * root_;`
- Public:
  - Construction(s)
  - Destructor
  - Accessor Method(s)
  - `virtual void insert(int data);`
    - This is a convenience method that allows the User to insert data into a Tree without needing to know the specifics – it will simply call the function below.
  - `TreeNode * insertNode(TreeNode * root, TreeNode * node);` → root and new node
    - This method is responsible for inserting a new `TreeNode` into the tree. The result should be returned as a pointer to the root of the tree.

The `RedBlackTree` Class should inherit publically from our `BinaryTree` Class as a Red-Black Tree is a type of Binary Tree. This will allow us to access the attributes and methods of that Class to allow us to represent a specific type of Binary Tree and include color as a tree feature. The `RedBlackTree` Class should have the following attributes and operations (`&` = Pass-by-Pointer Reference):

- Private:
  - `void rotateLeft(TreeNode *& root, TreeNode *& newNode);`
    - This allows us to perform a left rotational shift of our tree to ensure proper balance is maintained. It will be called internally from our `balanceColor` method.
  - `void rotateRight(TreeNode *& root, TreeNode *& newNode);`
    - This allows us to perform a right rotational shift of our tree to ensure proper balance is maintained. It will be called internally from our `balanceColor` method.
  - `void balanceColor(TreeNode *& root, TreeNode *& newNode);`
    - This method allows us to maintain proper balance within our tree and properly adjusts the color of each node, if necessary, to adhere to the rules of a Red-Black Tree. This will be a large and complex function – you will need to implement the Red-Black Tree rules that we discuss in lecture here.
- Public:
  - Construction(s)
  - Destructor
  - `virtual void insert(int data);`
    - We need to override (from our Base Class) this method as in a Red-Black Tree when we insert a new `TreeNode` it must have an associated color designation. We can use the Base Class method `insertNode` within this methods definition. This method will also call the `balanceColor` method described above.
  - `void printRedNodes(TreeNode * root);`
    - You will need to use an Inorder Traversal in this method to search for Red colored `TreeNodes`.
  - `void printBlackNodes(TreeNode * root);`
    - You will need to use a Preorder Traversal in this method to search for Black colored `TreeNodes`.

Tying this all together you will need to write a `driver` that will test your Red-Black Tree and provide the necessary functionality as described earlier as outlined by Blue IV. Your driver program should allow for the user to input a space delimited text file (`data.txt`) to be loaded and built into our Red-Black Tree and then display the Red Nodes, Black Nodes, and Root Node of the constructed Red-Black Tree. Finally, you will need to create a `makefile` that properly links all of this code together and creates an executable named **A4.exe**

An example of sample output is as follows - given the following sample input file:

```
3 18 7 10 22 8 15 29
```

Your output would contain the following:

```
Red Nodes: 8  15  18  29
Black Nodes: 7  3  10  22
Root: 7
```

I will be grading what is located in the **master branch** of your GitHub repository. It is strongly recommended that you commit and push often! Please make use of the Git feature of leaving descriptive messages along with your commits. Be sure to add me to any repository as a collaborator so I can view and grade your submissions. Failure to do so will result in a 0 on the assignment as I will have nothing to grade!

Submission:

All assignments must be submitted on Butler GitHub (github.butler.edu). I will allow you to work on this assignment with up to one (1) other person. If you choose to work on this project with a partner you need to email me letting me know of your "group." Be sure to make note of specific contributions of each team member so I can assign grades accordingly. **Note:** You are NOT required to work with a partner. The name of your Butler GitHub repository must be as follows: **cs341_fall2020_redblack**

The directory structure of the repository must contain the following files:

- **driver.cpp**
- **RedBlackTree.cpp**
- **RedBlackTree.h**
- **BinaryTree.cpp**
- **BinaryTree.h**
- **TreeNode.cpp**
- **TreeNode.h**
- **Node.h**
- **Node.cpp**
- **data.txt**
- **makefile**

Each source file (`.cpp/.h`) **must** include the Honor Pledge and digital signature – if working in pairs, please ensure both students' digital signatures are present on the files.

A red-black tree satisfies the following properties:

1. **Red/Black Property:** Every node is colored, either red or black.
2. **Root Property:** The root is black.
3. **Leaf Property:** Every leaf (NIL) is black.
4. **Red Property:** If a red node has children then, the children are always black.
5. **Depth Property:** For each node, any simple path from this node to any of its descendant leaf has the same black-depth (the number of black nodes).

What we need help with:

- Making sure our insert method is correct
- How to use the getters and setters to take care of the rotations
- When to set the color? ?

Remember: check to see if my newNode puts the tree out of balance

Balance color

While

(newNode!=root) &&
(newNode->getColor() != BLACK) &&
(newNode->getParentNode()->getColor() == RED)

If (uncleNode != nullptr && uncleNode->getColor() == RED)
    Recolor the nodes!

Else
    Make the rotation (right or left)

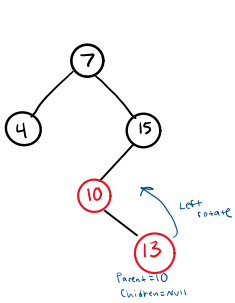
    Check the parent node of our newNode
    If the right child is a problem, left rotate
    If the left child is a problem, right rotate
    Then recolor

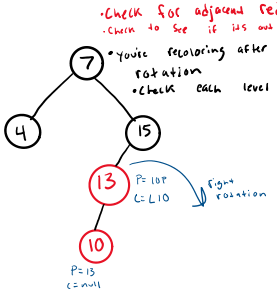
1. A Red-Black Tree must be a type of Binary Search Tree.
2. The root Node is Black
3. Every Node when inserted begins as a Red Node.
4. If a Node is Red then its Children must be Black.
5. Every path from a Node to a `nullptr` must contain the same number of Black Nodes.
   - This ensures that the tree will remain balanced.
6. Every leaf (`nullptr`) must be Black.

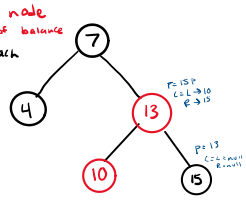
- adjacent red nodes & black nodes
- checking grandparents & aunts & uncles

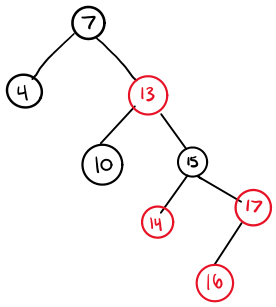
easy check to see if you need a full tree rotation: if a subtree has more than one black node from the root

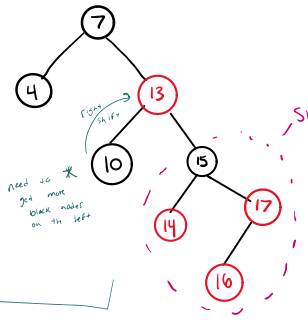
- first check (is parent node red)
- check for adjacent red node
- check to see if its out of balance
- you're recoloring after each rotation
- check each level

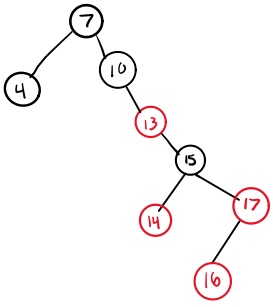


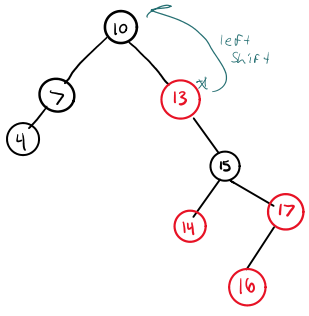


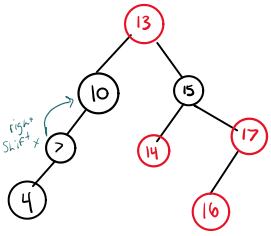


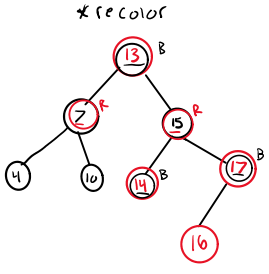












```
LOOP (newNode != root && newNode.color != BLACK && newNode.parent.color == RED)
    Set parentNode
    Set grandParentNode
    IF (parentNode == grandParentNode.leftChild)
        Set uncleNode
        IF (uncleNode != LEAF_NODE && uncleNode.color == RED)
            RECOLOR
        ELSE
            IF (newNode == parentNode.rightChild)
                ROTATE_LEFT
            ELSE
                ROTATE_RIGHT
    IF (parentNode == grandParentNode.rightChild)
        Set uncleNode
        IF (uncleNode != LEAF_NODE && uncleNode.color == RED)
            RECOLOR
        ELSE
            IF (newNode == parentNode.leftChild)
                ROTATE_RIGHT
            ELSE
                ROTATE_LEFT
root.color = BLACK
```

Assignment 5

Wednesday, November 4, 2020 3:44 PM

CS 341 – Fall 2020
Assignment #5 – The Last Can Of Who Hash
Due: 11/24/2020

We have explored some new data structures as part of your journey in this course and we want to wrap things up (as Lebowski said, "this assignment really tied the whole course together"….or something like that) with an exploration of data structures and their performance. Each data structure has a unique and special purpose. For instance, we want to use a Binary Search Tree (or Red-Black Tree) if we want to quickly search for an element – can be done in `log(n)` time; while insertions are much easier in a Linked List than in an Array. In this assignment, we are going to explore a Hash Table and see how it can provide the ability to add or remove an element in constant time!

Now our faithful CEO (Blue IV) has observed all of our hard work this semester and wants to reward us with a can of Who Hash – only one small problem he doesn't understand how Hash Tables work. Now he hears that we have been learning all about Hash Tables from Dr. Ryan. He also recently learned that collisions can happen in Hash Tables and they create big headaches. What to do!? It just so happens that a certainly data structure, a Linked List, can be used to avoid collisions in Hash Tables. Now this is GREAT news because it just so happens that in Assignment #2 we created a Doubly Linked List!

Blue IV wants to see how different hashing approaches, that we have learned about in lecture, compare. Specifically he wants us to compare three techniques for solving collisions in Hash Tables: Linear Probing, Quadratic Probing, and Separate Chaining (we will discuss each of these in lecture). For our data set he wants us to use the duplicates list that we created in Assignment #3 – as it just so happens that our Dictionary was essentially a Hash Entry (key, value)!

For this assignment you are tasked with creating a Hash Table using our Doubly Linked List from Assignment #2 along with providing the implementation necessary to model both Linear and Quadratic Probing techniques – through an Array.

A few notes about the specific requirements of the program:

- The program should load SPACE delimited (key, value) numerical data from a text file (`duplicates.txt`) generated in Assignment #3.
- The program should allow you to choose which Collision Technique to use (Linear Probing, Quadratic Probing, and Separate Chaining).
    - You should then insert each (key, value) into the Hash Table using the selected technique.
    - For our Hash function we will always use modulus to perform the calculation.
        - `hashValue = key % size`
- You will be using an Array and a List (Doubly Linked List) to represent the Hash Table.
    - Each implementation should provide the ability to Insert, Search, Print, and Remove from the Hash Table.
- The program then should allow the user to Search, Remove, Print the Hash Table.
- The program should handle invalid cases (e.g., invalid text entry, file I/O, etc.).
- The Hash Table should be placed on the Heap (memory management).
    - The program should contain no memory leaks – make sure to use Valgrind!
        - `valgrind --log-file=valgrind.txt A5.exe`

The `HashTableChaining` Class should inherit publically from our `HashTable` Class. The `HashTableChaining` Class should have the following attributes and operations:

- Private:
  - `DoublyLinkedList * entry_;`
  - `int size_;`
- Public:
  - Constructor(s)
  - Destructor
  - Accessor Methods
  - `void insert(int key, int value);`
  - `int search(int key);`
  - `void remove(int key);`
  - `void print();`

You will need to modify your `DoublyLinkedList` class as follows:

- Public:
  - `int find(int key);`
    - This will return the value of the element stored in the `LinkedNode`.
  - `void removeLinkedNode(int key);`
    - This will add the ability to remove a `LinkedNode` from our `DoublyLinkedList`.

You will need to modify all the files from Assignment #2 to store a `HashEntry` rather than just an `int` in your files. We will cover this in lecture to ensure this is done properly.

Tying this all together you will need to write a `driver` that will test your Hash Table and provide the necessary functionality as described earlier as outlined by Blue IV. Your driver program should allow for the user to input a space delimited text file (`duplicates.txt`) to be loaded and inserted into our Hash Table. Finally, you will need to create a `makefile` that properly links all of this code together and creates an executable named **A5.exe**

An example of sample output is as follows - given the following sample input file (user input in **RED**):

```
76 1
40 1
48 1
5 1
55 1
```

Your output would contain the following:

```
Welcome to Blue IV's Can of Who Hash!

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Quit Program

Please enter your choice: 1

**********************************
[0]: 48
[1]: 5
```

```
[2]: 55
[3]:
[4]:
[5]: 40
[6]: 76
***********************************

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 1

Search (Please enter a Key): 48

Key: 48 Value: 1

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 1

Search (Please enter a Key): 0

Invalid key! Key 0 not found in table!

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 2

Remove (Please enter a Key): 48

Key 48 removed.

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 3

***********************************
[0]:
[1]: 5
[2]: 55
[3]:
[4]:
[5]: 40
[6]: 76
***********************************

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 4
```

```
1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Quit Program

Please enter your choice: 2

***********************************
[0]: 48
[1]:
[2]: 5
[3]: 55
[4]:
[5]: 40
[6]: 76
***********************************
1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 4

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Quit Program

Please enter your choice: 3

***********************************
[0]:
[1]:
[2]:
[3]:
[4]:
[5]: 40<-->5
[6]: 76<-->48<-->55
***********************************

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 1

Search (Please enter a Key): 5

Key: 5  Value: 1

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 1

Search (Please enter a Key): 0

Invalid key! Key 0 not found in table!

1) Search For Entry
2) Remove Entry
```

```
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 2

Remove (Please enter a Key): 48

Key 48 removed.

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 3

************************************
[0]:
[1]:
[2]:
[3]:
[4]:
[5]: 40<-->5
[6]: 76<-->55
************************************

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 4

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Quit Program

Please enter your choice: 4

Thank you for using Blue IV's program - Goodbye!
```

I will be grading what is located in the **master branch** of your GitHub repository. It is strongly recommended that you commit and push often! Please make use of the Git feature of leaving descriptive messages along with your commits. Be sure to add me to any repository as a collaborator so I can view and grade your submissions. Failure to do so will result in a 0 on the assignment as I will have nothing to grade!

Submission:

All assignments must be submitted on Butler GitHub (github.butler.edu). I will allow you to work on this assignment with up to one (1) other person. If you choose to work on this project with a partner you need to email me letting me know of your "group." Be sure to make note of specific contributions of each team member so I can assign grades accordingly. **Note:** You are NOT required to work with a partner. The name of your Butler GitHub repository must be as follows: **cs341_fall2020_whohash**

The directory structure of the repository must contain the following files:

- **driver.cpp**
- **HashTable.h**
- **HashTableChaining.h**
- **HashTableChaining.cpp**
- **HashTableArray.h**
- **HashTableArray.cpp**
- **HashEntry.h**
- **HashEntry.cpp**
- **List.h**
- **LinkedList.h**
- **LinkedList.cpp**
- **DoubleLinkedList.h**
- **DoubleLinkedList.cpp**
- **LinkedNode.h**
- **LinkedNode.cpp**
- **Node.h**
- **Node.cpp**
- **data.txt**
- **makefile**

Each source file (`.cpp/.h`) **must** include the Honor Pledge and digital signature – if working in pairs, please ensure both students' digital signatures are present on the files.

****Extra Credit****

Modify your Assignment #2 to make use of Templates such that it would work for both this assignment as well as THAT assignment.

HashEntry * entry_ = new HashEntry[size]

- Store data after user choice

```
HashTable * hash = new HashTableArray();
```

CS 341 – Fall 2020
Assignment #5 – The Last Can Of Who Hash
Due: 11/24/2020

We have explored some new data structures as part of your journey in this course and we want to wrap things up (as Lebowski said, "this assignment really tied the whole course together"….or something like that) with an exploration of data structures and their performance. Each data structure has a unique and special purpose. For instance, we want to use a Binary Search Tree (or Red-Black Tree) if we want to quickly search for an element – can be done in `log(n)` time; while insertions are much easier in a Linked List than in an Array. In this assignment, we are going to explore a Hash Table and see how it can provide the ability to add or remove an element in constant time!

Now our faithful CEO (Blue IV) has observed all of our hard work this semester and wants to reward us with a can of Who Hash – only one small problem he doesn't understand how Hash Tables work. Now he hears that we have been learning all about Hash Tables from Dr. Ryan. He also recently learned that collisions can happen in Hash Tables and they create big headaches. What to do!? It just so happens that a certainly data structure, a Linked List, can be used to avoid collisions in Hash Tables. Now this is GREAT news because it just so happens that in Assignment #2 we created a Doubly Linked List!

Blue IV wants to see how different hashing approaches, that we have learned about in lecture, compare. Specifically he wants us to compare three techniques for solving collisions in Hash Tables: Linear Probing, Quadratic Probing, and Separate Chaining (we will discuss each of these in lecture). For our data set he wants us to use the duplicates list that we created in Assignment #3 – as it just so happens that our Dictionary was essentially a Hash Entry (key, value)!

For this assignment you are tasked with creating a Hash Table using our Doubly Linked List from Assignment #2 along with providing the implementation necessary to model both Linear and Quadratic Probing techniques – through an Array.

A few notes about the specific requirements of the program:

- The program should load SPACE delimited (key, value) numerical data from a text file (`duplicates.txt`) generated in Assignment #3.
- The program should allow you to choose which Collision Technique to use (Linear Probing, Quadratic Probing, and Separate Chaining).
    - You should then insert each (key, value) into the Hash Table using the selected technique.
    - For our Hash function we will always use modulus to perform the calculation.
        - `hashValue = key % size`
- You will be using an Array and a List (Doubly Linked List) to represent the Hash Table.
    - Each implementation should provide the ability to Insert, Search, Print, and Remove from the Hash Table.
- The program then should allow the user to Search, Remove, Print the Hash Table.
- The program should handle invalid cases (e.g., invalid text entry, file I/O, etc.).
- The Hash Table should be placed on the Heap (memory management).
    - The program should contain no memory leaks – make sure to use Valgrind!
        - `valgrind --log-file=valgrind.txt A5.exe`

Development Process:

For the development of your code: you should create four (4) new classes: `HashTable`, `HashEntry` `HashTableArray` and `HashTableChaining`. For the `HashEntry` class, you can modify/use your existing `Dictionary` class from Assignment #3 if you wish. You will also be using your `DoublyLinkedList` (and associated files) from Assignment #2.

The `HashTable` Class should be a Base Class that both `HashTableArray` and `HashTableChaining` should inherit from. The `HashTable` Class should have the following attributes and operations:

- Public:
  - Contructor
  - Destructor
  - `void insert(int key, int value) = 0;`
  - `int search(int key) = 0;`
    - This function should return the value.
  - `void remove(int key) = 0;`
  - `void print() = 0;`

The `HashEntry` Class (you can modify your `Dictionary` class from Assignment #3 if you wish here) should have the following attributes and operations:

- Private:
  - `int key_;`
  - `int value_;`
  - `Status status_;`
- Public:
  - Constructor(s)
  - Destructor
  - Accessor Method(s)
- `enum Status {EMPTY, OCCUPIED, REMOVED};`

The `HashTableArray` Class should inherit publically from our `HashTable` Class. The `HashTableArray` Class should have the following attributes and operations:

- Private:
  - `HashEntry * entry_;`
  - `int size_;`
- Public:
  - Constructor(s)
  - Destructor
  - Accessor Methods
  - `void insert(int key, int value);`
  - `int search(int key);`
  - `void remove(int key);`
  - `void print();`

You can add a convenience method to keep track of which type of probing we are using.

The `HashTableChaining` Class should inherit publically from our `HashTable` Class. The `HashTableChaining` Class should have the following attributes and operations:

- Private:
  - `DoublyLinkedList * entry_;`
  - `int size_;`
- Public:
  - Constructor(s)
  - Destructor
  - Accessor Methods
  - `void insert(int key, int value);`
  - `int search(int key);`
  - `void remove(int key);`
  - `void print();`

You will need to modify your `DoublyLinkedList` class as follows:

- Public:
  - `int find(int key);`
    - This will return the value of the element stored in the `LinkedNode`.
  - `void removeLinkedNode(int key);`
    - This will add the ability to remove a `LinkedNode` from our `DoublyLinkedList`.

You will need to modify all the files from Assignment #2 to store a `HashEntry` rather than just an `int` in your files. We will cover this in lecture to ensure this is done properly.

Tying this all together you will need to write a `driver` that will test your Hash Table and provide the necessary functionality as described earlier as outlined by Blue IV. Your driver program should allow for the user to input a space delimited text file (`duplicates.txt`) to be loaded and inserted into our Hash Table. Finally, you will need to create a `makefile` that properly links all of this code together and creates an executable named **A5.exe**

An example of sample output is as follows - given the following sample input file (user input in **RED**):

```
76 1
40 1
48 1
5 1
55 1
```

Your output would contain the following:

```
Welcome to Blue IV's Can of Who Hash!

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Quit Program

Please enter your choice: 1
```

```
// Honor Pledge:
//
// I pledge that I have neither given nor
// received any help on this assignment.
//
// n8swalley & k8edwards
#include "HashTableArray.h"
#include <iostream>
HashTableArray::HashTableArray(int size) : size_(size), choice_(0)
{
    entry_ = new HashEntry[size];
    for(int i=0; i < size; i++)
    {
        HashEntry newEntry;
        entry_[i] = newEntry;
    }
}
void HashTableArray::insert(int key, int value)
{
    int index(0);
    int newIndex(0);
    HashEntry newEntry;
    if(getUserChoice() == 1) //Linear probing
    {
        index = value % getSize();
        while(entry_[index].getStatus() == 'O')
        {
            index = (index + 1) % getSize();
        }
        entry_[index] = newEntry;
        entry_[index].setValue(value);
        entry_[index].setKey(key);
        entry_[index].setStatus('O');
    }
    // else if (getUserChoice() == 2)
    // {
    //         index = value % getSize();
    //         while(!newEntry->getStatus() == 'E')
    //         {
    //             for(int i = 0; i < getSize(); i++)
    //             {
    //                 newIndex = (index + i*i) % getSize();
    //             }
    //             //entry_[newIndex] = newEntry;
    //         }
    //         entry_[index] = newEntry;
    // }
}
int HashTableArray::search(int key)
{
    bool isFound = false;
    for(int i=0; i < getSize(); i++)
    {
        if((entry_[i].getStatus() == 'O') && (entry_[i].getValue() ==
key))
        {
            std::cout<<"Key: "<< entry_[i].getValue() << " Value: "<<e
ntry_[i].getKey() <<std::endl;
            isFound = true;
        }
    }
    if(isFound!=true)
    {
        std::cout<<"Invalid key! Key "<<key <<" not found in table!"<<
std::endl;
    }
}
void HashTableArray::remove(int key)
{
    bool isFound = false;
    for(int i=0; i < getSize(); i++)
    {
        if((entry_[i].getStatus() == 'O') && (entry_[i].getValue() ==
key))
        {
            std::cout<<"Key: "<< entry_[i].getValue() << " removed."<<
std::endl;
            isFound = true;
            HashEntry empty;
            entry_[i] = empty;
```

```
1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Quit Program

Please enter your choice: 1

***********************************
[0]: 48
[1]: 5
```

```
[2]: 55
[3]:
[4]:
[5]: 40
[6]: 76
***********************************

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 1

Search (Please enter a Key): 48

Key: 48 Value: 1

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 1

Search (Please enter a Key): 0

Invalid key! Key 0 not found in table!

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 2

Remove (Please enter a Key): 48

Key 48 removed.

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 3

***********************************
[0]:
[1]: 5
[2]: 55
[3]:
[4]:
[5]: 40
[6]: 76
***********************************

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 4
```

```
            std::cout<<"Key: "<< entry_[i].getValue() << " removed."<<
std::endl;
            isFound = true;
            HashEntry empty;
            entry_[i] = empty;
            entry_[i].setStatus('R');

        }
    }
    if(isFound!=true)
    {
        std::cout<<"Invalid key! Key "<<key <<" not found in table!"<<
std::endl;
    }
}
void HashTableArray::print()
{
    for(int i=0; i < getSize(); i++)
    {
        if(entry_[i].getStatus() == 'E' || entry_[i].getStatus() == 'R
')
        {
            std::cout<<"["<<i<<"]: "<<" "<<std::endl;
        }
        if((entry_[i].getStatus() == 'O'))
        {
            std::cout<<"["<<i<<"]: "<<entry_[i].getValue()<<std::endl;
        }
    }
    std::cout<<"***********************************" <<std::endl;
}
int HashTableArray::getSize()
{
    return size_;
}
void HashTableArray::setSize(int size)
{
    size_ = size;
}
void HashTableArray::setUserChoice(int num)
{
    choice_ = num;
}
int HashTableArray::getUserChoice()
{
    return choice_;
}
```

```
1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Quit Program

Please enter your choice: 2

***********************************
[0]: 48
[1]:
[2]: 5
[3]: 55
[4]:
[5]: 40
[6]: 76
***********************************
1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 4

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Quit Program

Please enter your choice: 3

***********************************
[0]:
[1]:
[2]:
[3]:
[4]:
[5]: 40<-->5
[6]: 76<-->48<-->55
***********************************

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 1

Search (Please enter a Key): 5

Key: 5  Value: 1

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 1

Search (Please enter a Key): 0

Invalid key! Key 0 not found in table!

1) Search For Entry
2) Remove Entry
```

```
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 2

Remove (Please enter a Key): 48

Key 48 removed.

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 3

***********************************
[0]:
[1]:
[2]:
[3]:
[4]:
[5]: 40<-->5
[6]: 76<-->55
***********************************

1) Search For Entry
2) Remove Entry
3) Print Hash Table
4) Return to Main Menu

Please enter your choice: 4

1) Linear Probing
2) Quadratic Probing
3) Separate Chaining
4) Quit Program

Please enter your choice: 4

Thank you for using Blue IV's program - Goodbye!
```

I will be grading what is located in the **master branch** of your GitHub repository. It is strongly recommended that you commit and push often! Please make use of the Git feature of leaving descriptive messages along with your commits. Be sure to add me to any repository as a collaborator so I can view and grade your submissions. Failure to do so will result in a 0 on the assignment as I will have nothing to grade!

```cpp
// Honor Pledge:
//
// I pledge that I have neither given nor
// received any help on this assignment.
//
// n8swalley & k8edwards
#include "HashTableArray.h"
#include "HashTableChaining.h"
#include <fstream>
void menu1()
{
        std::cout<<"1) Linear Probing" <<std::endl;
        std::cout<<"2) Quadratic Probing" <<std::endl;
        std::cout<<"3) Separate Chaining" <<std::endl;
        std::cout<<"4) Quit Program" <<std::endl;
        std::cout<<std::endl;
        std::cout<<"Please enter your choice: ";
}
void menu2()
{
        std::cout<<"1) Search for Entry" <<std::endl;
        std::cout<<"2) Remove Entry" <<std::endl;
        std::cout<<"3) Print Hash Table" <<std::endl;
        std::cout<<"4) Return to Main Menu" <<std::endl;
        std::cout<<std::endl;
        std::cout<<"Please enter your choice: ";
}
void userChoice1(HashTableArray * tbl)
{
    int choice(0);
    std::cout<<std::endl;
    menu1();
    std::cin>>choice;
    tbl->setUserChoice(choice);
    std::cout<<std::endl;
    std::cout<<"*****************************" <<std::endl;
}
void userChoice2(HashTableArray * tbl)
{
    int choice2(0);
    std::cout<<std::endl;
    menu2();
    std::cin>>choice2;
    std::cout<<std::endl;
    std::cout<<"*****************************" <<std::endl;
    if (choice2 == 1)
        {
            int userKey(0);
            std::cout<<"Search (Please enter a key): ";
            std::cin>>userKey;
            std::cout<<std::endl;
            tbl->search(userKey);
            userChoice2(tbl);
        }
        else if(choice2==2)
        {
            int userKey(0);
            std::cout<<"Remove (Please enter a key): ";
            std::cin>>userKey;
            std::cout<<std::endl;
            tbl->remove(userKey);
            userChoice2(tbl);
        }
        else if(choice2==3)
        {
            tbl->print();
            userChoice2(tbl);
        }
        else if(choice2==4)
        {
            //userChoice1(tbl);
        }
        else
        {
            std::cout<<"ERROR: invalid input"<<std::endl;
            exit(0);
        }
}
int main()
{
    int key(0);
    int value(0);
    int choice(0);
    int choice2(0);
    int size(7);
    std::ifstream fileToReadIn; //input stream handler
    std::string nameOfInputFile = ""; //user input variable fo
r filename
```

Submission:

All assignments must be submitted on Butler GitHub (github.butler.edu). I will allow you to work on this assignment with up to one (1) other person. If you choose to work on this project with a partner you need to email me letting me know of your "group." Be sure to make note of specific contributions of each team member so I can assign grades accordingly. **Note:** You are NOT required to work with a partner. The name of your Butler GitHub repository must be as follows: **cs341_fall2020_whohash**

The directory structure of the repository must contain the following files:

- **driver.cpp**
- **HashTable.h**
- **HashTableChaining.h**
- **HashTableChaining.cpp**
- **HashTableArray.h**
- **HashTableArray.cpp**
- **HashEntry.h**
- **HashEntry.cpp**
- **List.h**
- **LinkedList.h**
- **LinkedList.cpp**
- **DoubleLinkedList.h**
- **DoubleLinkedList.cpp**
- **LinkedNode.h**
- **LinkedNode.cpp**
- **Node.h**
- **Node.cpp**
- **data.txt**
- **makefile**

Each source file (`.cpp/.h`) **must** include the Honor Pledge and digital signature – if working in pairs, please ensure both students' digital signatures are present on the files.

Extra Credit

Modify your Assignment #2 to make use of Templates such that it would work for both this assignment as well as THAT assignment.

```
    int size(7);
    std::ifstream fileToReadIn; //input stream handler
    std::string nameOfInputFile = ""; //user input variable fo
r filename

    HashTableArray * tbl = new HashTableArray(7);
    HashTableChaining * dllTbl = new HashTableChaining(7);
    std::cout<<"Welcome to Blue IV's Can of Who Hash!"<<std::e
ndl;
    std::cout<<std::endl;
    std::cout<<"Please enter the filename to read in: ";
    std::cin >> nameOfInputFile;
    std::cout<<std::endl;
    menu1();
    std::cin>>choice;
    tbl->setUserChoice(choice);
    std::cout<<std::endl;
    std::cout<<"*****************************" <<std::endl;

    fileToReadIn.open(nameOfInputFile.c_str());
    if(fileToReadIn.fail())
    {
        std::cout << "ERROR: file not found" <<std::endl;
    }
    else
    {
        if(tbl->getUserChoice() == 1 || tbl->
getUserChoice() == 2)
        {
            while(fileToReadIn>>value>>key)
            {
                tbl->insert(key, value);
            }
        }
        else if(tbl->getUserChoice() == 3)
        {
            while(fileToReadIn>>value>>key)
            {
                dllTbl->insert(key, value);
            }
        }
        else if(tbl->getUserChoice() == 4)
        {
            std::cout<<"Goodbye!"<<std::endl;
            exit(0);
        }
        else
        {
            std::cout<<"ERROR: invalid input"<<std::endl;
            exit(0);
        }
        tbl->print();
        userChoice2(tbl);
        // std::cout<<std::endl;
        // menu2();
        // std::cin>>choice2;
        // std::cout<<std::endl;
        // if (choice2 == 1)
        // {
        //     int userKey(0);
        //     std::cout<<"Search (Please enter a key): ";
        //     std::cin>>userKey;
        //     tbl->search(userKey);
        //     menu2();
        //     std::cin>>choice2;
        //     std::cout<<std::endl;
        // }
        // else if(choice2==2)
        // {
        // }
        // else if(choice2==3)
        // {
        // }
        // else if(choice2==4)
        // {
        // }
        // else
        // {
        //     std::cout<<"ERROR: invalid input"<<std::endl;
        //     exit(0);
        // }
    }
}
```