Data Structures & Algorithms

Assignment # 1

Group Members:

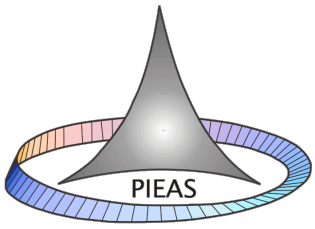
Assad Sultan (bsee1603)

Ayesha Ali (bscs1601)

Uswa Fatima (bsee1601)

Instructor:

Mr. Tanveer Ali



April 13, 2020

Question No. 1

We want to build a device that can help us in showing contents of attached devices, adding/modifying/deleting contents in attached devices. Devices may be USB, CD ROM, Hard Disk. Each device has limited warranty and cost. Build OO Model this problem. And show how you applied various object oriented concepts in it.

Solution

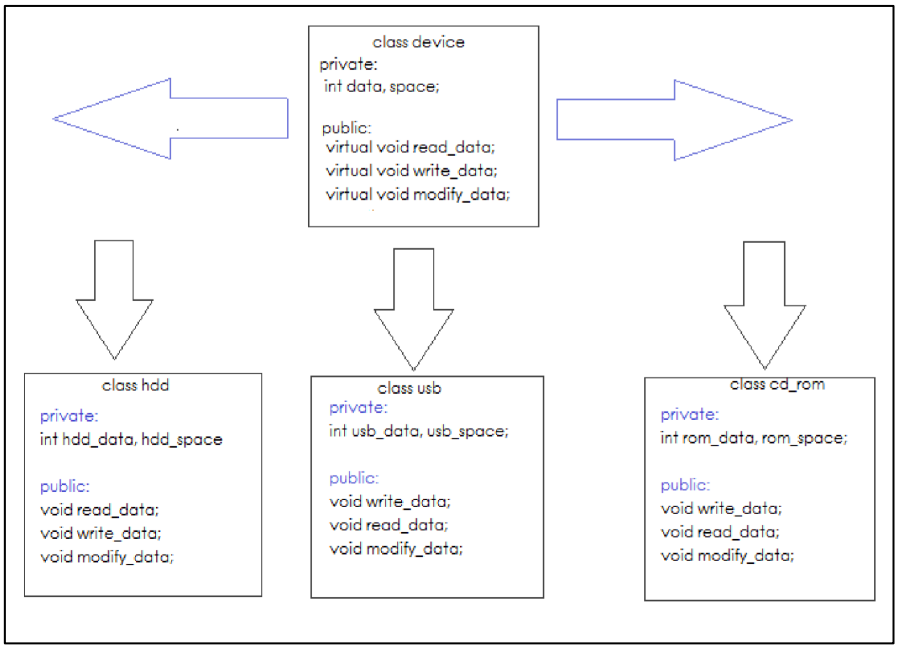


Figure 1: Figure showing contents of attached devices, adding/modifying/deleting contents in attached devices

Application of OOP Concepts

Encapsulation:

Encapsulation is defined as binding together the data and the functions that manipulates them. Encapsulation also lead to data abstraction or hiding. As using encapsulation also hides the data. Each class has its own content when it comes to data and its disk space. Hence data of each device whether it is a USB or HDD. It is securely packed.

Inheritance:

In object-oriented programming, inheritance is the mechanism of basing an object or class upon another object (prototype-based inheritance) or class (class-based inheritance), retaining similar implementation. For this problem we had different storage devices such as USB, Hard Disk Drive and CD. Each had some

properties in common and some functions as well. Like each device is required to hold some data as well as preform some operations on data. There a parent class was made which contained all such properties. Each child class inherits these properties can use them.

Polymorphism:

A key component of object-oriented programming is polymorphism, or the ability to re-use and extend code. It means you can have the same code act differently depending on the context. In terms of programming, this means you can have the same function in different contexts. As described earlier, each class inherits the functions of the parent class. But what if the device wants to have same other function with the same name. So, this is done by using the keyword virtual in C++.

Abstraction:

In object-oriented programming, abstraction is one of three central principles (along with encapsulation and inheritance). Through the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency. This OOP model perfectly shows the characteristics of abstraction. Although it may define a straight path to achieve the given problem, but it still doesn't restrict the user to use only that path. For example, you can proceed by making USB an object of class device and proceed and you can also make USB an object of class USB. Also, in using functions, you can use both functions of child and parent class.

Question No. 2

Your task is to extend the List class by writing one method for removing successive duplicate items from lists, and one method for adding them. The squish() method and twin() method will operate on singly-linked lists. Also mention worst-case Time and Space complexity of these two methods. Here is the detail about these functions.

`squish( )`

The `squish()` method performs as described: `squish()` takes the list and, wherever two or more consecutive items are equal, it removes duplicate nodes so that only one consecutive copy remains. Hence, no two consecutive items in this list are equal upon completion of the procedure. After `squish()` executes, the list may well be shorter than when squish() began. No extra items are added to make up for those removed. For example, if the input list is `[ 0 0 0 0 1 1 0 0 0 3 3 3 1 1 0 ],` the output list is `[ 0 1 0 3 1 0 ]`. Here is the prototype of the squish() function that you will use in your code.

```
List squish(List list)
```

twin( )

The `twin()` method performs as described: `twin()` takes the list and doubles its length by replacing each node with two consecutive nodes referencing the same item. For example, if the input list is `[ 3 7 4 2 2 ]`, the output list is `[ 3 3 7 7 4 4 2 2 2 2 ]`. Here is the prototype of the twin() function that you will use in your code.

```
List twin(List list)
```

Solution

Code:

```
#include<iostream>
#include<string>
using namespace std;

class Item {
private:
	string name_of_item;
	int quantity;
	string category;
	int price;
	int total_price;
public:
	//Default Constructor
	Item() {};
	Item(string name_of_item, string category, int quantity, int price)
	{
		this->name_of_item = name_of_item;
		this->category = category;
		this->quantity = quantity;
		this->price = price;
	};
	void setName(string name)
	{
		this->name_of_item = name;
	}
	string getName()
	{
		return this->name_of_item;
	}
	void setCategory(string category)
	{
		this->category = category;
	}
	string getCategory()
	{
		return this->category;
	}

	void setPrice(int price)
	{
		this->price = price;
	}
	int getPrice()
	{
		return this->price;
	}
```

```
	void setQuantity(int quantity)
	{
		this->quantity = quantity;
	}
	int getQuantity()
	{
		return this->quantity;
	}

	int calculatePrice()
	{
		return this->total_price = quantity*price;
	}

};
class Node
{
public:
	void set(Item new_item)
	{
		this->new_item = new_item;
	}
	Item get()
	{
		return this->new_item;
	}
	Node *getNext()
	{
		return nextNode;
	}
	void setNext(Node *nextNode)
	{
		this->nextNode = nextNode;
	}
private:
	Item new_item;
	Node *nextNode;
};
class LinkedList
{
public:
	LinkedList() {
		headNode = new Node;
		headNode->setNext(NULL);
		currentNode = NULL;
		size = 0;
	};

	int length()
	{
		return size;
	}

	void add(Item new_item)
	{
		Node* newNode = new Node(); //object created
		newNode->set(new_item); //set value
		if (currentNode != NULL) {//not first node
			newNode->setNext(currentNode->getNext());
			currentNode->setNext(newNode);
```

```
                lastCurrentNode = currentNode;
                currentNode = newNode;
            }
            else {
                newNode->setNext(NULL); //1st element, clear memory
                headNode->setNext(newNode);
                lastCurrentNode = headNode;
                currentNode = newNode;
            }
            size++;
      }

      void next()
      {
            lastCurrentNode = currentNode;
            currentNode = currentNode->getNext();
      }

      Item get()
      {
            return currentNode->get();
      }
    void display()
    {
      Node *tempNode = currentNode;
      currentNode = headNode;
      for (int i = 1;i <= size;i++)
      {
            this->next();
            if (currentNode != NULL)
            {
                  cout << "Item : "<< i << endl;
                  cout << "\t" << "Product\t\t" << (this-
>get()).getName()<< endl;
                        cout << "\t" << "Category\t" << (this-
>get()).getCategory()<< endl;
                        cout << "\t" << "Price\t\t" << (this-
>get()).getPrice()<< endl;
                        cout << "\t" << "Quantity\t" << (this-
>get()).getQuantity()<< endl;
                  }
       }
       currentNode = tempNode;
      }

       bool search(string item_name)
       {
            currentNode = headNode;
            bool flag = 0;
            for (int i = 1;i <= size; i++)
      {
            lastCurrentNode = currentNode;
            this->next();
            if((this->get()).getName()== item_name)
            {
                  cout<<"Item Found"<<endl;
                  flag = 1;
                return 1;
                  break;
                  }
            }
```

```
                if(flag == 0)
                {
                        cout<<"Item not found"<<endl;
                        return 0;
                }
        }

        void del(string item_name)
        {
            bool isItemFound = this->search(item_name);
            if (isItemFound == 1)
            {
                        lastCurrentNode->setNext(currentNode->getNext());
                        delete currentNode;
                        cout<<"Item Deleted"<<endl;
                        size--;
            }

        }

        void end()
        {
            currentNode = headNode;
            for (int i = 1; i <= size; i++)
            {
                if(currentNode != NULL)
                {
                        this->next();
                }
            }
        }


        void insert(int position, Item new_item)
        {
            currentNode = headNode;
            for (int i = 1 ; i < position; i++)
            {
                if(currentNode != NULL)
                {
                        this->next();
                }
            }
         this->add(new_item);
         }

      void squish()
      {
            currentNode = headNode;
            this->next();
            int tempsize = size;
            for (int i = 1 ; i <= tempsize ; i++)
            {
                if((lastCurrentNode->get()).getName() == (currentNode-
>get()).getName())
                {
                                lastCurrentNode->setNext(currentNode-
>getNext());
                                delete currentNode;
                                currentNode = lastCurrentNode->getNext();
                                size--;
```

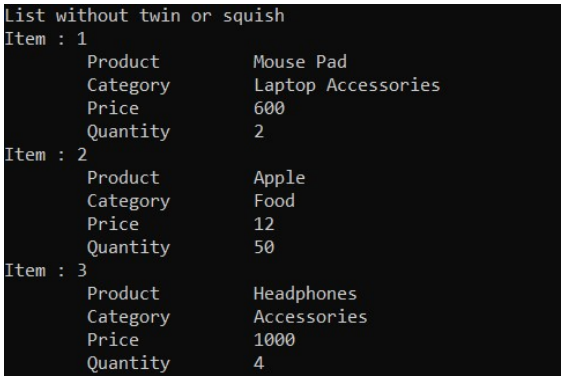
```
                }
                else this->next();
            }
        }

        void twin()
        {
            currentNode = headNode;
            this->next();
            while (currentNode != NULL)
            {
                this->add(this->get());
                this->next();
            }
        }

private:
        int size;
        Node *headNode;
        Node *currentNode, *lastCurrentNode;

};
//Driver Function
int main()
 {
        Item item1("Mouse Pad", "Laptop Accessories", 2, 600);
        Item item2("Apple", "Food", 50, 12);
        Item item3("Headphones", "Accessories", 4, 1000);

        LinkedList linkedlist;
        linkedlist.add(item1);
        linkedlist.add(item2);
        linkedlist.add(item3);
        cout << "List without twin or squish \n";
        linkedlist.display();
        linkedlist.twin();
        cout << "List after twin\n";
        linkedlist.display();
        linkedlist.squish();
        cout << "List after squish\n";
        linkedlist.display();
        return 0;

}
```

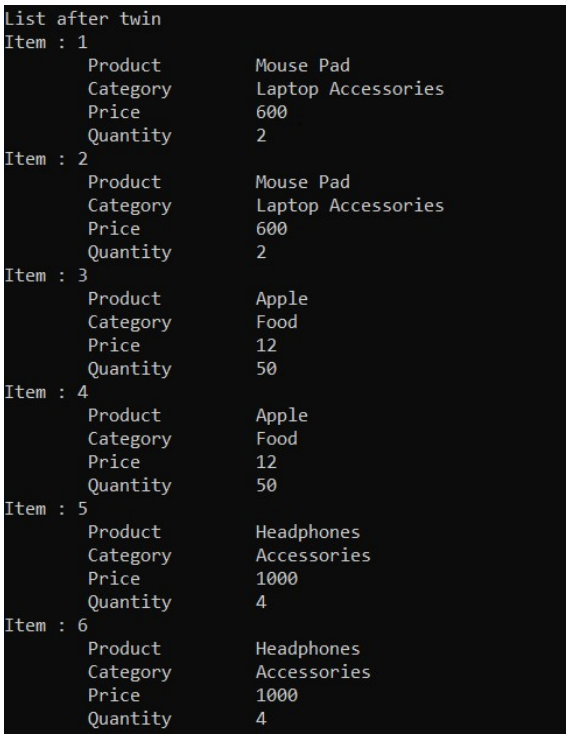
Time & Space Complexity:

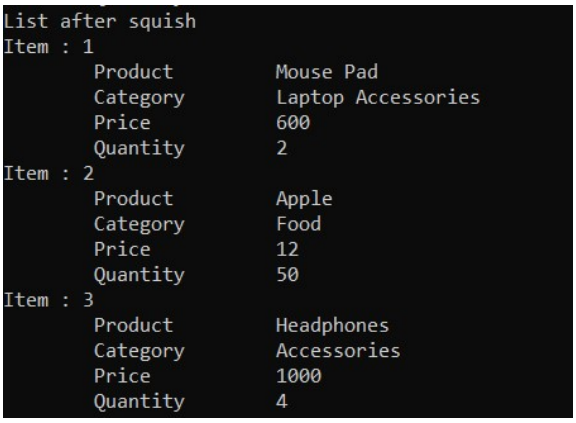
Time: O(n)

Space: O(n)

Output:







Question No. 3

The Linux ext2 file system has a file definition (inode) that includes details about the file and links to data nodes with links to index nodes at the end. Small files would not use those. A block is a sequence of bit or Bytes with a fixed length i.e. 512 bytes, 4kB,

8kB, 16kB, 32kB etc. Write code to implement miniature version of XSF. Block size should be 4kB. Maximum 10 Direct Block allowed. If file require more than 10 blocks, then we shall use Single indirect block. Each Single direct data block maximum has 5 data blocks.

Solution:

Code: