# CSE 335 Spring 2016 Exam 2

**Question 1. [20 points] C++ Template**

Use C++ template to implement a double-ended Queue class. Double-ended Queue allows you to insert or remove elements from either the front (head) or back (tails). The inserted or removed element can be any type of data. You need to implement the following functionalities: insert_front, insert_back, remove_front, remove_back, isempty, and print. Use the following main.cpp to check your implementation.

```cpp
int main() {
    dequeue<int> q;
    q.insert_front(1);
    q.insert_front(2);
    q.insert_back(3);
    q.print();      // 2, 1, 3
    q.remove_front();
    q.print();      // 1, 3
    q.remove_back();
    q.remove_front();
    cout << "Empty?" << q.isempty() << endl;    // 1
    q.remove_front();   // Queue is Empty !!!
    q.remove_back();    // Queue is Empty !!!
    q.print();      // Nothing to print

    dequeue<string> qu;
    qu.insert_front('hello');
    qu.insert_front('Sparty');
    qu.insert_back('world');
    qu.insert_back('335');
    qu.print();     // Sparty, hello, world, 335,
    qu.remove_back();
    qu.print();     // Sparty, hello, world,
    qu.remove_back();
    qu.remove_front();
    cout << "Empty?" << qu.isempty() << endl;   // 0
    qu.print();     // hello,
    qu.remove_back();
    qu.print();     // Nothing to print
    return 0;
}
```

Your output should look like the following:

```
2, 1, 3,
1, 3,
1
Queue is Empty !!!
Queue is Empty !!!
Nothing to print
Sparty, hello, world, 335,
Sparty, hello, world,
0
hello,
Nothing to print
```

**Solution:**

```cpp
#ifndef DEQUEUE_H
#define DEQUEUE_H
#include <vector>
using namespace std;
template <typename T>
class dequeue{
private:
    vector<T> a;
    typename vector<T>::iterator front;
    typename vector<T>::iterator rear;
public:
    dequeue()    {
        front=a.begin();
        rear=a.begin();
    }
    ~dequeue(){}
    void insert_back(T i)    {
        a.push_back(i);
        rear=a.end();
    }
    int insert_front(T i){
        vector<T> tmp;
        if(a.size()==0){
            a.push_back(i);
            front=a.begin();
            rear=a.end();
        }
        else{
            tmp.push_back(i);
            for(int j=0;j<a.size();j++){
                tmp.push_back(a[j]);
            }
            a=tmp;
        }
        return a.size();
    }
    int remove_front(){
        if(isempty()){
            cout<<"Queue Empty !!\n";
            return (-9999);
        }
        else{
            a.erase(a.begin());
            front=a.begin();
        }
        return(a.size());
    }
    int remove_back(){
        if(isempty()){
            cout<<"Queue Empty !!\n";
            return (-9999);
        }
        else{
            a.pop_back();
            rear=a.end();
        }
        return(a.size());
    }
    int isempty()    {
        if(a.size()<1)
                return 1;
        else
                return 0;
    }
    void print(){
        if(a.size()<1){
            cout<<"Nothing to print"<<endl;
        }
        else{
            cout<<"Current Queue:";
            for(auto itr:a){
                cout<<itr<<", ";
            }
            cout<<"\n";
        }
    }
};
#endif /* DEQUEUE_H */
```

**Question 2. [20 points] Adapter pattern**
In this problem you are required to implement a stack class using Dequeue class implemented in Question 1. You must implement the following functionalities: **push(int i)**, **pop()**, **isEmptyStack()**, and **printStack()**. You are not required to use C++ template to complete this question. *[Hint:* If you use C++ Template and get a compiler error like this: *there are no arguments to 'isempty' that depend on a template parameter, so a declaration of 'isempty' must be available [-fpermissive],* then you need to call the functions like *Dequeue<T>::isempty(), instead of just isempty().]*

The following main.cpp gives you some sense of what you have to do without template. If you want to use C++ template, then the line 2 should be *Stack<int> *st = new StackAdapter<int>();*

```
1.  int main() {
2.      Stack *st = new StackAdapter();
3.
4.      st->push(6);
5.      st->push(1);
6.      st->push(3);
7.      st->push(4);
8.      st->printStack();   // 6, 1, 3, 4
9.      st->pop();
10.     cout << "Empty? " << st->isEmptyStack() << endl;    // Empty? 0
11.     st->printStack();   // 6, 1, 3,
12.     st->pop();
13.     st->pop();
14.     st->pop();
15.     st->pop();
16.     cout << "Empty? " << st->isEmptyStack() << endl;    // Empty? 1
17.     st->printStack();   // Nothing to print
18.
19.     return 0;
20. }
```

***[Output]***
*6, 1, 3, 4*
*Empty? 0*

*6, 1, 3,*
*Empty? 1*
*Nothing to print*

**Solution with using C++ template:**

```cpp
#ifndef DEQUEUE_H
#define DEQUEUE_H
#include <vector>
using namespace std;
template <typename T>
class dequeue{
private:
    vector<T> a;
    typename vector<T>::iterator front;
    typename vector<T>::iterator rear;
public:
    dequeue()    {
        front=a.begin();
        rear=a.begin();
    }
    ~dequeue(){}
    void insert_back(T i)    {
        a.push_back(i);
        rear=a.end();
    }
    int insert_front(T i){
        vector<T> tmp;
        if(a.size()==0){
            a.push_back(i);
            front=a.begin();
            rear=a.end();
        }
        else{
            tmp.push_back(i);
            for(int j=0;j<a.size();j++){
                tmp.push_back(a[j]);
            }
            a=tmp;
        }
        return a.size();
    }
    int remove_front(){
        if(isempty()){
            cout<<"Queue Empty !!\n";
            return (-9999);
        }
        else{
            a.erase(a.begin());
            front=a.begin();
        }
        return(a.size());
    }
    int remove_back(){
        if(isempty()){
            cout<<"Queue Empty !!\n";
            return (-9999);
        }
        else{
            a.pop_back();
            rear=a.end();
        }
        return(a.size());
    }
    bool isempty()    {
        if(a.size()<1)
                return 1;
        else
                return 0;
    }
    void print(){
        if(a.size()<1){
            cout<<"Nothing to print"<<endl;
        }
        else{
            cout<<"Current Queue:";
            for(auto itr:a){
                cout<<itr<<", ";
            }
            cout<<"\n";
        }
    }
};
#endif /* DEQUEUE_H */
```

```
#ifndef STACK_H
#define STACK_H
#include<vector>
using namespace std;
template <typename T>
class stack{
    public:
        virtual void push(T)=0;
        virtual int pop()=0;
        virtual bool isEmptyStack()=0;
        virtual void printStack()=0;
};
#endif /* STACK_H */
```

```
#ifndef STACK_ADAPTER_H
#define STACK_ADAPTER_H
#include "Stack.h"
#include "dequeue.h"
template <typename T>
class stack_adapter:public stack<T>, public dequeue<T>{
public:
    stack_adapter():dequeue<T>(){}
    virtual void push(T i){
        dequeue<T>::insert_back(i);
    }
    virtual int pop(){
        return dequeue<T>::remove_back();
    }
    virtual bool isEmptyStack(){
        return dequeue<T>::isempty();
    }
    virtual void printStack(){
        dequeue<T>::print();
    }
};
#endif /* STACK_ADAPTER_H */
```

**Question 3. [60 points]** This problem will focus on **Composite and Visitor patterns**. You are asked to design a discount store which consists of 2 types of components, item and package. Each item has a name of type string and a price of type double. An item must be within some package. Each package has a name of type string and a discount percentage of type double, (such as 20% = 0.20). A package can contain an arbitrary number of items and other packages. The discounted price of the package is the sum of the prices of all the items and packages within, after applying the specified discount. For example, if a package has two items worth $12 and $8 respectively and the discount percentage of the package is 10%, then the price of the package would be 18, calculated as follows: {(12 + 8) x (1 – 0.10) = 18}

**Question 3.1 [15 points]**You are required to write implement the two classes Item and Package so that the following client code can compile correctly. You may introduce additional classes if necessary.

**Question 3.2 [10 points]** You need to implement an operation print() for these classes. The print operation on an item will print the name of the item, colon, the price of the item, and then semicolon, such as, “lipstick-pink:8;”. The print operation on a package will first print "(", the name of the package, colon, the print result of every items or packages inside the package, "):", and the total price of the package with discount, such as “(NailPolishes:Revlon-Red:4;Loreal-Black:5;SallyHanson-clear:3;):6;”

**Question 3.3 [20 points]** You also need to implement a Visitor Class for the above classes. You need to design one or more visitor classes and write syntactically correct C++ code for them. The VisitPackage() operation may use getChildren(int i) method of composite class to traverse through every item and the packages within and calculate the total price and the price of the package after applying the discount. Using the classes that you designed, the output of the client program given later should be as follows:
*(Cosmetics:lipstick-pink:8;lip_pencil-pink:5;(NailPolishes:Revlon-Red:4;Loreal-B*
*lack:5;SallyHanson-clear:3;):6;):17.1;*
*The total price before discount 19*
*The total price you have to pay after discount is 17.1*

```
int main()
{
    Package p1 ("Cosmetics", 0.10);
    Item i1 ("lipstick-pink", 8.00);
    Item i2 ("lip_pencil-pink", 5.00);

    Package p2("NailPolishes", 0.50);
    Item i3 ("Revlon-Red" , 4.00);
    Item i4 ("Loreal-Black" , 5.00);
    Item i5 ("SallyHanson-clear", 3.00);
    p1.add(&i1);
    p1.add(&i2);

    p2.add(&i3);
    p2.add(&i4);
    p2.add(&i5);

    p1.add(&p2);

    p1.print();
    cout<<endl;
    PackageVisitor pv;
    p1.acceptVisitor(&pv);
    return 0;
}
```

**Question 3.4 [15 points]** Draw a UML class diagram representing the class hierarchy that you designed. You must include attribute and operation names in your UML diagram for each class; for each operation, where applicable, you should also specify the type of parameters and return values. You should also indicate associations among your classes. Use /abc/ to denote *abc* in italics.

**Solution**:

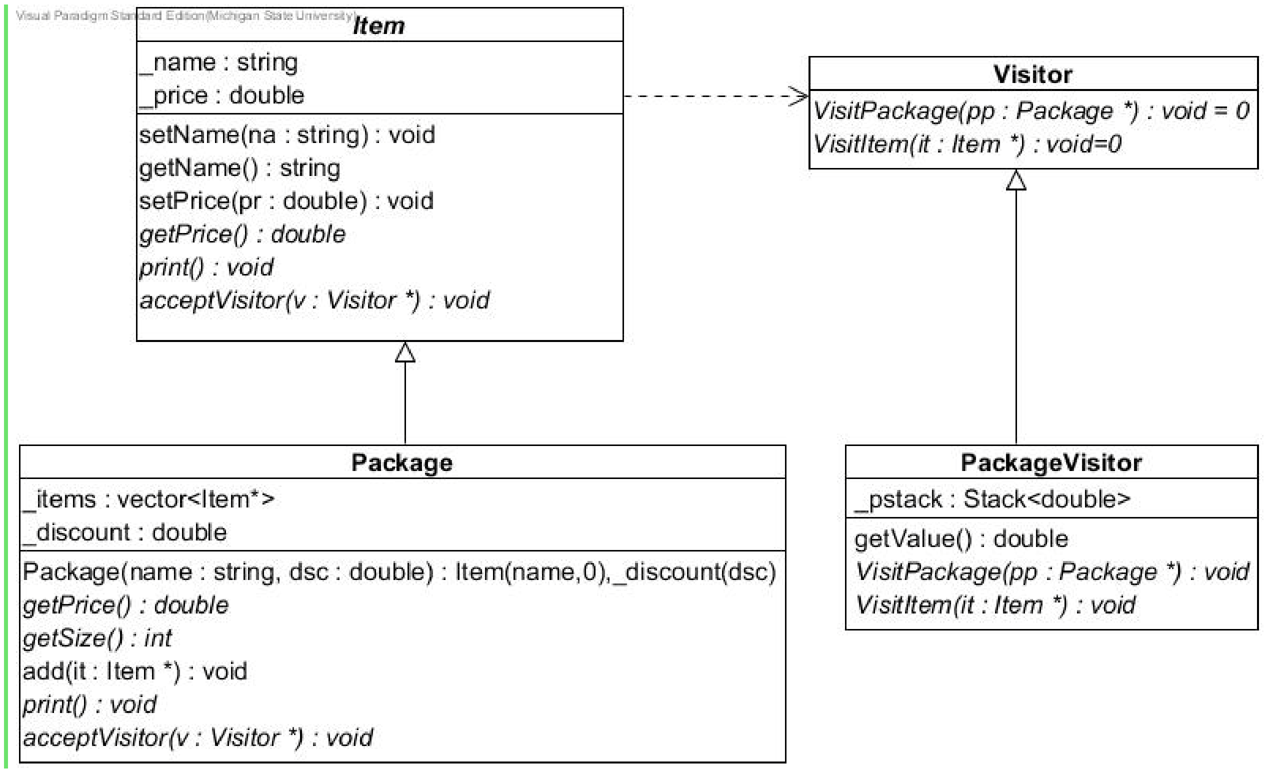
```
#ifndef ITEM_H_
#define ITEM_H_
#include <iostream>
#include <string>
#include <list>
#include "Visitor.h"
using namespace std;
class Item{
private:
    string _name;
    double _price;
public:
    Item( string na, double pr) : _name(na), _price(pr) {}
    virtual ~Item(){}
    void setPrice( double newPrice ) { _price = newPrice; };
    string getName() { return _name; };
    virtual double getPrice() { return _price; };
    virtual void acceptVisitor(Visitor* v){}
    virtual void print(){ cout << getName() << ":" << getPrice() << ";"; };
};
#endif /* ITEM_H_ */

#include <vector>
#include <stack>
#include "Visitor.h"
#include "PackageVisitor.h"
#include "Item.h"
using namespace std;
class Package: public Item{
private:
    vector <Item*> _items;
    double _discount ;
public:
    Package(string name , double dsc) : Item( name, 0), _discount(dsc){}
    virtual ~Package(){}
    virtual void print(){
        cout << "(" << getName() << ":";
        for(unsigned int i = 0; i < _items.size(); i++ )
            (_items[i])->print();
        cout << "):" << getPrice() <<";";
    };
    virtual double getPrice() {
        double totalPrice = 0;
        for( unsigned int i = 0; i < _items.size(); i++ )
            totalPrice += (_items[i])->getPrice();
        return totalPrice - totalPrice*_discount;
    };
    virtual int getSize() {return _items.size();}
    void add( Item* it ){ _items.push_back(it);}
    virtual void acceptVisitor(Visitor* v){
        v->VisitPackage(this);
    }
    Item* getChildren(int i){return _items[i];};
    double getDiscount(){return _discount;};
};
```

```
class Package;
class Item;
class Visitor{
public:
    Visitor();
    virtual ~Visitor();
    virtual void VisitPackage(Package* pp)=0;
    virtual void VisitItem(Item* it) = 0;
};
```

```
#ifndef PACKAGEVISITOR_H_
#define PACKAGEVISITOR_H_

#include "Visitor.h"
#include <stack>
using namespace std;
class PackageVisitor: public Visitor{
private:
    stack <double> _pstack;
public:
    PackageVisitor();
    virtual ~PackageVisitor();
    double getValue(){
        double result = _pstack.top();
        _pstack.pop();
        return result;
    }
    virtual void PackageVisitor::VisitPackage(Package* pp){
        double totalPrice=0;
        double discount=0;
        for(int i = 0; i< (pp->getSize()); i++){
            Item* temp = pp->getChildren(i);
            totalPrice += temp->getPrice();
        }
        cout<<"The total price before discount "<<totalPrice;
        discount = totalPrice * pp->getDiscount();
        totalPrice = totalPrice - discount;
        cout<<"\nThe total price you have to pay after discount is "<<totalPrice;
    }
    virtual void VisitItem(Item* it){}
};
```


Visual Paradigm Standard Edition(Michigan State University)
*Item*
_name : string
_price : double
setName(na : string) : void
getName() : string
setPrice(pr : double) : void
*getPrice() : double*
*print() : void*
*acceptVisitor(v : Visitor *) : void*
Visitor
*VisitPackage(pp : Package *) : void = 0*
*VisitItem(it : Item *) : void=0*
Package
_items : vector<Item*>
_discount : double
Package(name : string, dsc : double) : Item(name,0),_discount(dsc)
*getPrice() : double*
*getSize() : int*
add(it : Item *) : void
*print() : void*
*acceptVisitor(v : Visitor *) : void*
PackageVisitor
_pstack : Stack<double>
getValue() : double
*VisitPackage(pp : Package *) : void*
*VisitItem(it : Item *) : void*