Adnar Lozano
CSE 330
4 / 11 / 17
Lab 2s

1) Review of Writing a class

Source code for **Writing.cpp**, containing **CPoint CVec3** and **main**() functions

```
#include <iostream>
using namespace std;
template <typename T>
class CPoint {
public:
	T x,y,z;
	CPoint () {};
	CPoint (T a, T b, T c) : x(a), y(b), z(c) {}
	CPoint operator- (const CPoint& param) {
	CPoint temp;
		temp.x = x - param.x;
		temp.y = y - param.y;
		temp.z = z - param.z;
		return temp;
	}
};
template <typename T>
class CVec3 {
public:
	T x,y,z;
	CVec3 () {};
	CVec3 (T a, T b, T c) : x(a), y(b), z(c) {}
	CVec3 operator + (const CPoint& lhs, const CVec3& rhs) {
		CPoint temp;
		temp.x = lhs.x + rhs.x;
		temp.y = lhs.y + rhs.y;
		temp.z = lhs.z + rhs.z;
		return temp;
	}
	CVec3 operator+ (const CVec3& param) {
		CVec3 temp;
		temp.x = x + param.x;
		 temp.y = y + param.y;
		temp.z = z + param.z;
		return temp;
	}
};

int main () {
	CPoint <int> p1 (34,15,87);
```

```
        CPoint <int> p2 (14,22,53);
        CPoint <int> pntRslt = p2 - p1;
        cout << "Difference between two CPoint objects: "
                << pntRslt.x <<','
                << pntRslt.y <<','
                << pntRslt.z <<'\n';
        CPoint <int> p0 (34,15,87);
        CVec3 <int> v0 (14,22,53);
        CPoint <int> rslt = p0 + v0;
        cout << "Sum of CPoint and CVec3: "
                << rslt.x <<','
                << rslt.y <<','
                << rslt.z <<'\n';
        CVec3 <int> v1 (34,15,87);
        CVec3 <int> v2 (14,22,53);
        CVec3 <int> vecRslt = v1 + v2;
        cout << "Sum of two CVec3 objects: "
                << vecRslt.x <<','
                << vecRslt.y <<','
                << vecRslt.z <<'\n';
        return 0;
}
```

2) Card Class Study

a) I did some modifications to the Makefile clean target, at first it only removed the OBJS but now it removes the EXEC as well.

Source code for **Makefile**

```
#Makefile for suit
EXEC=game
SRCS=card.cpp deck.cpp player.cpp game.cpp
OBJS=$(SRCS:.cpp=.o)
$(EXEC): $(OBJS)
        g++ -o $@ $(OBJS)
#$< evaluates to the target's dependencies,
#$@ evaluates to the target

$(OBJS):
        g++ -c  $*.cpp

clean:
        rm $(OBJS) $(EXEC)
```

b) Source code for **suit_test.cpp**

```
//suit_test.cpp
#include "card.h"
int main(){
        Card cards[52];
```

```
        int  topCard;
        for ( int i = 1; i <= 13; i++ )
        {
                Card c1(diamond, i ), c2( spade, i), c3( heart, i ), c4( club, i );
                cards[++topCard] = c1;
                cards[++topCard] = c2;
                cards[++topCard] = c3;
                cards[++topCard] = c4;
        }
        //prints out all 52 cards
        for (int i = 0; i < 52; i++ )
        {
                cout<<cards[i]<<"\n";
        }
        Card c1;
        Card c2( diamond, 8 );

        cout << "Card one\n";
        cout << c1.rank << endl;
        cout << "Card two\n";
        cout << c2.rank << endl;
        return 0;
}
```

Source code for **card.cpp**

```
//card.cpp
#include "card.h"
Card::Card()
//default
{
        rank = 1;
        suit = spade;
}
Card::Card( suits s, int r )
{
        rank = r;
        suit = s;
}
ostream & operator << ( ostream &out, Card c )
{
        //first output rank
        switch ( c.rank ) {
                case 1: out << "Ace"; break;
                case 2: out << "Two"; break;
                case 3: out << "Three"; break;
                case 4: out << "Four"; break;
                case 5: out << "Five"; break;
```

```
                case 6: out << "Six"; break;
                case 7: out << "Seven"; break;
                case 8: out << "Eight"; break;
                case 9: out << "Nine"; break;
                case 10: out << "Ten"; break;
                case 11: out << "Jack"; break;
                case 12: out << "Queen"; break;
                case 13: out << "King"; break;
                default:
                out << c.rank; break;
        }
        //then output suit
        switch( c.suit ) {
                case diamond: out << " of Diamonds"; break;
                case spade: out << " of Spadess"; break;
                case heart: out << " of Heartss"; break;
                case club: out << " of Clubs"; break;
        }
        return out;
}
```

3) Card War Game

a) The keyword ***extern*** is used as a global variable. Typically, it is located inside the header or interface file. Every source or implementation file that includes this header would know of the extern object, data type, class, struct, function or a simple variable. Basically, the keyword tells the compiler that this object (class, etc) exists somewhere else so it doesn't give us an error about it being undefined or redefined in a particular file.

b) If we wished to implement another deck of cards and shuffle them together, then we would need to create two additional objects of type Deck, 1 for the second Deck, and 1 for the Total Deck (which is this case would be a Deck of 104 cards, double the standard Deck)
see zip folder for source code