<table>
<thead>
<tr>
<th>Problem</th>
<th>Possible</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>100</strong></td>
<td></td>
</tr>
</tbody>
</table>

(a) The exam contains 6 pages and 8 problems. Make sure your exam is complete.
(b) *Read each problem carefully before working on it.*
(c) If you write your answer on a different sheet, please indicate that you have done so.
(d) In code answers, comments are not necessary.
(e) Please put your Eos/Unity username at the top of each page.

On my honor: I affirm that I have not and will not use any external sources of assistance (notes, another student’s examination form, *etc.*) during this examination. I am the student whose name appears on this exam.

Signed: __________________________________________________________

Date: __________________________________________________________

Failure to sign this form legibly will result in a zero score on this exam.
1. [6 points] A function template is a function declaration that is parameterized with respect to template variables. A template function is a specialized version of a function template generated by the compiler.

Consider the following function template:

```cpp
template <typename T>
void f(T& x)
{
    T y = x + 3;
    x += y;
}
```

(a) Write the definition of the template function that is automatically generated by the compiler when the following code is compiled:

```cpp
double y = 17;
f(y);
```

(b) What are the restrictions on the actual type corresponding to T? In other words, if f is called with an actual parameter of type S,

```cpp
S x; f(x);
```

what must be true of type S for the compiler to accept this usage.

S must support (1) addition with integers and (2) the assignment operators = and +=. The most sensible way to achieve (1) is to supply (1a) a conversion constructor from int to S and (1b) a symmetric addition operator +.

2. [7 points] The identity function id is a function of one argument. When called, it immediately returns, with the return value being identical to the argument. For example, id(5)==5 and id(x)==x for any x.

Give the definition of id as a function template — so that the function can be used with an argument of any type. (Both the argument and the return value should be passed by reference.)

```cpp
template <typename T>
T& id (T& x)
{
    return x;
}
```
3. [12 points] What is output by the following code fragments?

(a) int* p = new int;
    *p = 4;
int* q = p;
cout << *q << endl;

(b) int x = 35;
int& y = x;
int* p = &x;
x++;
cout << x << ' ' << y << ' ' << *p << endl;

(c) int x = 17;
int& y = x;
y++;
x++;
cout << x << ' ' << y << endl;

(d) int* p = new int[5];
p[0] = 3;
p[1] = 4;
cout << *p << ' ' << p[0] << endl;

4. [12 points] What is output by the following program?

```cpp
int main() {
    D* dpt = new D;
    B* bpt = dpt;
    // sf, derived first
dpt->sf();
bpt->sf();
    // df, derived first
dpt->df();
bpt->df();
}
```

```cpp
class B {
    public:
        void sf() {cout << "B::sf\n";};
        virtual void df() {cout << "B::df\n";};
};

class D : public B {
    public:
        void sf() {cout << "D::sf\n";};
        void df() {cout << "D::df\n";};
};
```

D::sf
B::sf
D::df
D::df
5. [18 points] Each of the following code fragments contains at least one error (which could be a compiler error, a runtime error, or a logical error). Indicate the errors by writing the line-numbers where they occur. One point each for indicating the correct line; two points each for explaining why the code is erroneous.

(a) 1 int* p;
2    int x = 32767;
3    p = x;
   line 3: attempt to assign integer value to pointer variable

(b) 4 int x = 35;
5    int& y = &x;
   line 5: attempt to assign pointer value to integer reference

(c) 6 int* p = new int;
7    double* q = p;
   line 7: attempt to assign integer pointer to double pointer; no conversion takes place with pointers; int is not derived from double

(d) 8 int x = 17;
9    int* p = x;
10   *p = 17;
   line 9: attempt to assign integer value to pointer variable

(e) 11 int* p = new int[5];
12    int* q = new int[4];
13    p = q;
   line 13: memory leak; no reference to 5 element array

(f) 14 class C { public: virtual void f() = 0; };
15    C myc;
16    C* mycpt;
   line 15: cannot declare variable of type C, since C is abstract
6. [16 points] In this problem you will write the class declarations for a hierarchy of charge cards. You need not write the definition of any member functions; you need only give the class declarations. In giving the declarations, be careful to use inheritance and virtual declarations correctly; also use const when possible (there is no problem using const and virtual together).

**You must declare the classes** ChargeAccount, AmericanExpress, Visa and VisaAirMiles. In defining these, you may make use of the classes Address and Date; you need not declare the Address and Date classes.

Charge accounts keep track of

- the account number (which is fixed at time of construction),
- the balance,
- a billing Address, and
- a billing Date.

In addition:

- Visa cards have a minimum balance due, and
- VisaAirMiles cards with keep track of the number of airmiles earned.

All accounts allow methods to:

- query the account balance (simply returns the balance),
- charge the account a certain amount (adds a certain amount to the balance due, minimum balance due, and number of airmiles, as applicable),
- pay the account a certain amount (reduces the balance due and minimum balance due, as applicable), and
- bill the account holder (prints the balance, and in the case of Visa cards, the minimum balance due).

```cpp
class ChargeAccount {
public:
    ChargeAccount(int acctNum);
    virtual ~ChargeAccount();
    int query() const;
    virtual void charge(float amount) = 0;
    virtual void pay(float amount) = 0;
    virtual void bill() const = 0;

protected:
    const int acctNum_;
    float balance_; 
    Address billingAddress_; 
    Date billingDate_; 
};

class Visa : public ChargeAccount {
public:
    Visa(int acctNum);
    virtual void charge(float amount);
    virtual void pay(float amount);
    void bill() const;
private:
    float minBalanceDue_; 
};

class AmericanExpress : public ChargeAccount {
public:
    AmericanExpress(int acctNum);
    virtual void charge(float amount);
    virtual void pay(float amount);
    void bill() const;
};

class VisaAirMiles : public Visa {
public:
    VisaAirMiles(int acctNum);
    void charge(float amount);
    private:
    float milesEarned_; 
};
```

This is only one solution. It is also possible to have all ChargeAccount member functions be implemented in ChargeAccount (ie, not pure virtual).
For the next two questions, assume the class List has a data member head of type Node*. To answer the next two questions, you may define helper functions (which may be Node member functions).

```cpp
class Node {
  public:
    int info_;  
    Node* link_;  
    Node(int n, Node* npt=NULL) : info_(n), link_(npt) { };     
    ~Node() { delete link_; } 
};
```

7. [14 points] Write a List member function drop that takes an integer n and destructively removes the first n elements of the list. Suppose lpt points to the list [1,2,3,4,5]. Then after the call lpt->drop(3), lpt should point to the list [4,5]. Likewise, after the call lpt->drop(7), lpt should point to an empty list.

**Recursive solution with dummy node.**

```cpp
void List :: drop(int n)
{
    head_->drop(n);
}

void Node :: drop(int n)
{
    if (link_ && n>0) {
        Node* tmp = link_;    
        link_ = link_->link_; 
        tmp->link_ = NULL;    
        delete tmp;            
        drop(n-1);            
    }
}
```

**Iterative solution w/o dummy node. (Note the similarity.)**

```cpp
void List :: drop(int n)
{
    while (head_ && n>0) {
        Node* tmp = head_;    
        head_ = head_->link_; 
        tmp->link_ = NULL;    
        delete tmp;            
        n--;                
    }
}
```
For this question, your solution MUST define a RECURSIVE helper function on Nodes. (Half credit for an iterative solution. Quarter credit for a destructive solution.)

8. [15 points] Write a List member function pprod with the following prototype:

List* List::pprod(int n = 1) const;

The function should return a pointer to a new List which is the prefix product of the original list. For example, if lpt points to the list [3,4,5], then lpt->pprod() should return a new list [3,12,60]. The $i^{th}$ element of lpt->pprod() is the product of the first $i$ elements of lpt.

The optional parameter $n$ is a dummy argument which is multiplied into the first element before continuing. Thus if kpt points to the list [4,5], then kpt->pprod(3) should return the list [12,60].

Solution with dummy node.

```cpp
List* List :: pprod(int i=1) const
{
    List* lpt = new List;
    lpt->head_->link_ = head_->pprod(i);
    return lpt;
}

Node* Node :: pprod(int i=1) const
{
    return !link_
    ? NULL
    : new Node(i*link_->info_, link_->pprod(i*link_->info_));
}
```

Solution w/o dummy node.

```cpp
List* List :: pprod(int i=1) const
{
    List* lpt = new List;
    if (head_)
        lpt->head_ = head_->pprod(i);
    return lpt;
}

Node* Node :: pprod(int i=1) const
{
    return new Node(i*info_, !link_
    ? NULL
    : link_->pprod(i*info_));
}
```