

3. (4 pts) Write a *recursive* function to print the elements of a singly linked list in reverse order. Remember to use good programming style. You may assume that the head is passed into the function the first time it is called and you have unrestricted access to the variables in the node class.

```
// tmp – head is passed in the first time the function is called  
void List::reversePrint(Node* tmp)
```

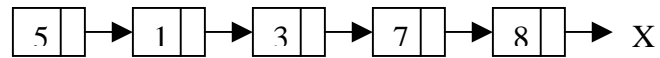
4. (4 pts) Given the following snippet of code: (There are no syntax errors in this code)

```
int* x = new int(5);  
int y = *x;  
int* z = x;  
(*x)++;  
int* a = 0;
```

Describe what the following statements will produce (either what will be printed or what will happen). If a memory address is printed, be as specific as you can by telling me what variable/value the memory address is associated with.

- a) `cout << x << endl;`
- b) `cout << &x << endl;`
- c) `cout << *x << endl;`
- d) `cout << y << endl;`
- e) `cout << *z << endl;`
- f) `cout << *a << endl;`

5. (5 pts) Assume you have a singly linked list of integers. Write a function to find and print out the integer values of two nodes in a linked list whose values add up to a given number. For instance, if the given number is 8 and the linked list contains:



then your function will print out “5” and “3”. Alternatively, your function could print out “1” and “7”. If there are multiple possibilities, you need only find one pair of numbers to add up to the given number. You may not assume that there exists a pair of numbers in the list that adds up to the given number. Your function should do something appropriate in that situation. You may assume that the head is passed into the function the first time it is called and you have unrestricted access to the variables in the node class.

```
// tmp – head is passed in the first time the function is called  
// value – the given number - find two numbers that add up to this number  
void List::findSum ( Node* tmp, int value )
```