Quiz #1

(February 22, 2012)

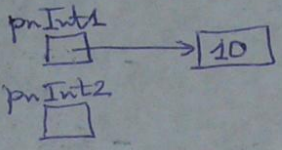
All questions are out of 2 points; awarded for correctly pointing out all flaws without incorrectly claiming any non-existing flaws.

Your Names: Sample Solutions

Question #1. Consider the following independent code snippets (a)-(d).
What, if any, coding errors are exhibited in each? Be brief, specific, and complete.
Sketch a diagram of the situation in memory in each case – to help you (reason about the code) and to help me (see your thinking more clearly and grade effectively).

(a)

```
int *pnInt1 = new int(10);
int *pnInt2 = *pnInt1;
```



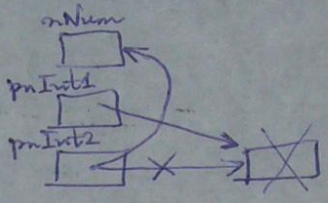
incorrect initialization: type mismatch

(b)

```
int nNum, *pnInt1, *pnInt2 = new int;
pnInt1 = pnInt2;
*pnInt1 = nNum * 2;
pnInt2 = &nNum;
delete pnInt1;
cout << *pnInt2;
delete pnInt2;
```

suspicious assignment (but ok)

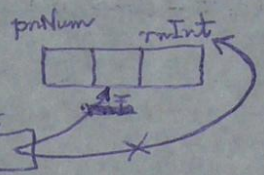
uninitialized variable



→ dangling pointer pnInt1, still pointed at memory that was released
→ can't release memory (for a variable) that hasn't been allocated with new (e.g., nNum)

(c)

```
int pnNum[3], &rnInt = pnNum[2];
int *pnInt = &rnInt;
if (pnInt != NULL && rnInt = 0)
    pnInt = pnNum + 1;
```



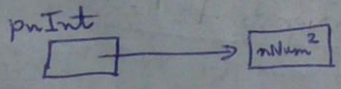
uninitialized variable

should be == instead

Note: rnInt=0 could be an assignment, making the two warnings mute, but then there are two other issues with it:
- an assignment shouldn't be (normally) part of a condition;
- it'll always have a value 0, which is false, making the condition always fail, and making the subsequent statement unreachable.

(d)

```
int myFunc( int nNum )
{
    int *pnInt = new int;
    *pnInt = nNum * nNum;
    cout << *pnInt;
    return *pnInt;
}
```



memory allocated but never released: memory leak, since pnInt goes out of scope at the end of the function and will be automatically reclaimed (as a local variable), but the memory it used to point to will become lost.

Question #2. The following are independent functions, some of which use the `ListNode` declaration below. What coding errors, if any, are present in each function? Be brief, specific, and complete.

```
struct ListNode
{
    int        item;
    ListNode * next;
};
```

(a)

```
// prints the items stored in the list, starting from a given position
void printListItems( ListNode * poListPos ) const
{
    ListNode * poCurr = poListPos;
    while (poCurr.next != NULL)
    {
        cout << poCurr.item << endl;
        poCurr = poCurr.next;
    }
}
```

→ it's preferable to use const here if part of a class (method)

→ unnecessary to create another variable; poListPos could be used instead, since changes to it won't stick

- testing for next that may not exist: a very common error by virtually everyone on the midterm: what if poCurr == NULL?
- fails to print the very last element

should be poCurr→item and poCurr→next; using →, not . operator

(b)

```
// deletes the first node of a linked list
void deleteFirst( ListNode * poListHead )
{
    ListNode * pTemp = poListHead;
    poListHead = poListHead->next;
    delete pTemp;
}
```

→ changes to poListHead will not stick; to correctly sustain/keep changes, must use *& or **

→ what if poListHead == NULL?

→ If the code gets to the point of deleting pTemp, that would cause a memory leak (in conjunction with the fact that the head pointer will not have moved after we return from this function). The reason is that we would have deleted the head and thereby lost access to the rest of the list – since outside this function we would hold a pointer to this deleted head only.

(c)

```
// puts a given character string in quotes, e.g. quiz becomes "quiz"
char * addQuotesToString( char * pcOriginal )
{
    // allocate space for the new string; it needs two extra cells
    int nNewLength = strlen(pcOriginal) + 2;
    char * pcQuoted = new char[nNewLength];

    // copy the original string to the destination,
    // starting with an offset to allow space for the initial quote: '"'
    for (int i=1; i<nNewLength-1; i++)
        pcQuoted[i] = pcOriginal[i-1];

    // add the quote symbols: one at the start, and one at the end
    pcQuoted[0] = '"';
    pcQuoted[nNewLength-1] = '"';

    return pcQuoted;
}
```

must allocate nNewLength+1 spaces – since the last one needs to be '\0', the string termination char.

good to check after the previous allocation of pcQuoted if it got the memory, i.e., !=NULL

The index here may be nNewLength-2 depending on ~~how initialized~~

→ must terminate string with pcQuoted[nNewLength] = '\0';

before returning, and for that the earlier additional character space to must have been allocated.