Practice question Answers

COMP-322, Winter 2012, All Sections

These questions are not necessarily the same as a final as they aren't necessarily exactly representative of the degree of difficulty, length, or material of the exam. That is, you should not assume that the questions on the quiz will be exactly like these.

List of important topics

Here is a rough list of the topics we've covered so far that would be examinable.

1. Pointers, pointers to struct, * and & operators, pointer arithmetic on arrays
2. Linked lists
3. References
4. Memory management, memory leaks. new/delete
5. Basics of iterators: how to iterate over a list, vector, etc. How to write a function to take as input iterators.
6. Basics of streams

Part 1 (0 points): Warm-up

Warm-up Question 1 (0 points)
Suppose I have the following functions:

```
void swap(int& a, int&b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 3;
    int y = 4;
    ?????
}
```

What would I write in place of the ????? if I wanted to call the function to properly swap x and y?

Rewrite swap() so that it took as input two `int*` instead of `int&`. How would that affect the way you call the method?

```
//Since swap takes as input 2 reference types, we can simply call it ‘‘normally"
//instead of ??????? write
swap(x,y);

//To rewrite using pointers one would put:

void swap(int* p , int *q)
{
   int temp = *p;
   *p = *q;
   *q = temp;
}

//the function call would then be:
swap(&x, &y);
```

Warm-up Question 2 (0 points)

What is the problem with the following code?

```
struct Point2d
{
     int x;
     int y;
};

Point2d* makePoint(int x, int y) {
       Point2d point;
       point.x = x;
       point.y = y;
       return &point;
}
```

How could one solve this problem and still return a Point2d?

```
The issue here is that we are returning the address of automatically assigned data.
That is, since the address returned is the address of the variable point,
and the variable point is a local variable, it will be erased once the function completes.
This means that the address will have no meaning.
One should fix this using the new operator and pointers the whole time:

Point2d* makePoint(int x, int y)
{
     Point2d* point = new Point2d;
     point->x = x;
     point->y = y;
     return point;
}
```

Warm-up Question 3 (0 points)

Suppose I have the following struct:

```
struct IntNode
{
     int data;
     IntNode* next;
}
```

Write a function that takes as input a pointer to an IntNode that represents a linked list. Assume that it is sorted in increasing order by the int data. Insert the new node in the correct location. You should obey proper memory management.

```
IntNode* insert(IntNode* root, int newValue)
{
     IntNode* newNode = new IntNode;
     newNode->data = newValue;

    if (root == NULL)
    {
        return newNode;
    }

    if (newNode < root->data)
    {
         newNode->next = root;
         return newNode;
    }

    IntNode* current = root;
    //use a while loop because we want access to current after the loop
    while (current->next != NULL)
    {
         if (newValue > current->next->data)
         {
              newNode->next = current->next;
              current->next = newNode;
              return root;
         }

         current = current->next;
   }

   //if we reach here then we must be adding to the very end.
   newNode->next = NULL;
   current->next = newNode;
   return root;
}
```

Warm-up Question 4 (0 points)

Suppose there exists a function `IntNode* fun(IntNode* x)` . Which of the following are valid? For those that are invalid, correct them. In cases that something prints, output what will print.

1.
   ```
   IntNode* bar = fun(NULL);
   //this is good
   ```

2.
   ```
   IntNode* bar;
   bar = &(*(fun(NULL)));
   //fun() returns a pointer, pointers can be deferenced with * and then
   //one can get the address of it using &
   ```

3.
   ```
   IntNode b = *(fun(NULL));
   //good
   ```

4.
   ```
   IntNode b;
   IntNode* c = *b;
   ```

```
//the above line is no good.
//*b doesn't make sense since b isn't a pointer
(*c) = b;
```

5.
   ```
   int x = 3;
   int&y = &x;
   //the above line isn't good since y is a reference
   //change to int*y = &x; or int&y = x;
   cout << x;
   ```

6.
   ```
   int x = 3;
   int&y = x;
   //this is good
   y = 10;
   cout << x;
   ```

7.
   ```
   int x = 2;
   int* y = &x;
   y = new int;
   (*y) = 3;
   cout << x;
   //This prints 2. If we removed y = new int, it would print 3
   //because *y = 3 would have changed x as well.
   //However, y = new int means that y should store a different
   //address than the address of x, meaning y and x are no longer
   //linked in any way.
   ```

8.
   ```
   int x[10];
   for (int j = 0; j < 10; j++) {
   x[j] = 0;
   }

   int* p = (x[3]);
   //the above doesn't compile, should be &x[3]
   *(p + 3) = 4;
   //if we fixed the prior mistake then this would
   //be the same as writing x[6] = 4
   cout << x[3] << " " << x[6] << endl;
   ```

   After answering the normal question, how would you rewrite the above using a reference instead of a pointer?

   ```
   This question doesn't make sense......I think I was sleepy when I wrote this.....
   You can't do the equivalent of *(p+3) with references
   ```

Warm-up Question 5 (0 points)
Write a function that takes as input a `const list<int>&` (from the stl library) and iterates over the entire list, summing all of the values.

```
int sumList(const list<int>\& list)
{
    int total = 0;
     for (list<int>::const_iterator start = list.begin();
            start != list.end();
            start++ )
     {
            total = total + *start;
```

```
     }

     return total;
}
```

Warm-up Question 6 (0 points)
Write a function that takes as input a start iterator for a container of `int` and an end (one past the finish) of a forward iterator and sums all the values in the container.

```
template<class InputIterator>
int sum(InputIterator start, InputIterator finish)
{
     int total = 0;
     for (InputIterator current = start; current != finish; current++)
     {
          total = total + *current;
     }

     return total;
}
```

Now write how you would call this function to take the sum of all the values in the `vector<int> v`

```
sum(v.begin(), v.end());
```

Warm-up Question 7 (0 points)
What is the advantage of using references? Why are they better than pointers in some cases? How does this relate to the notion of ”l-value” (something you can put on the left side of an equation)

```
The main advantage is that you are associating a variable with another variable.
This allows you to call a function and pass BY REFERENCE a variable which
is usually much faster than passing something by value which requires
an expression to be evaluated and then copied into a new variable.

Additionally, references are often syntactically cleaner since if we pass something
by reference, we just add an & in the function header and then can change
the variables normally without having to use * or & or anything.

A reference variable has to be assigned to an l-value, which is anything you can assign to.
It doesn't make sense to assign a reference to a const for example, since then
one couldn't change it.
```

Warm-up Question 8 (0 points)
What is the difference between the * operator and the & operator?

```
They are the inverse operators. * operator dereferences a pointer.
The & operator gets the address of an l-value.

So * turns an address into "data" and & turns "data" into an address.
(Of course an address is still data, so hence the quotes)
```

Warm-up Question 9 (0 points)
What is wrong with the following code?

```
//adds an element to a list at the front if it isn't already in the linked list,
//returns the element (linked to original list) or original list depending
//on if it was found or not
IntNode* insertUnique(IntNode* root, int value)
{
     IntNode* newNode = new IntNode;
     newNode->value = value;
     newNode->next = NULL;

     for (IntNode* current = root; current != NULL; current = current->next)
     {
         if (current->value == value)
         {
                 return root;
         }
     }

     //insert at the beginning now
     newNode->next = root;
     return newNode;
}
```

```
There is a memory leak. When we right return root inside the if statement, we
lose all links to the newNode. We need to add delete newNode before
return root or alternatively shift the creation of the new node to
be later on after the loop finishes (before newNode->next = root)
```

Warm-up Question 10 (0 points)

Explain the difference between how the computer evaluates `cout` in C++ vs a more standard method call such as `printf()` in C or `System.out.println()` in Java.

In other words, what is the difference between writing:

```
System.out.println("hi" + "bye");
OR
printf("hibye");
```

vs

```
cout << "hi" << "bye";
```

```
In Java or C, we are calling a function or method that prints whatever
is passed to the function as input. However when using cout or cin
we are using an operator << or >> . This operator is evaluated
as part of an expression and has the side effect of printing or reading something.
```