CS 308 Data Structures

Spring 2003 - Dr. George Bebis

Midterm

Duration: 1:00 - 2:15 pm

Name:

1. **True/False** (5 pts each) **To get credit, you must give brief reasons for your answers !!**

(1.1) **T F** An objective way to compare two algorithms is by comparing their execution (i.e., machine) times.

(1.2) **T F** The running time of *InsertItem* (unsorted lists) is *O(N)*

(1.3) **T F** Suppose *A* is declared as *int A[5];*. If the base address of the array is 100 and integers take up 4 bytes, the address of A[2] is 110.

(1.4) **T F** Given the declaration *int x=5, *ptr;* then the statement **ptr = &x;* in main() will cause an error.

(1.5) **T F** Image processing and computer vision is the same thing.

2. **Multiple choice** (5 pts each)

(2.1) The order of adding 1 to each element in a one dimensional array of N integers.

(a) $O(1)$
(b) O$(logN)$
(c) $O(N)$
(d) $O(NlogN)$
(e) $O(N^2)$

(2.2) Which of the following C++ class member functions should be provided if class objects point to dynamic data ?

(a) a destructor
(b) a constructor
(c) a copy-constructor
(d) a and b above
(e) a, b, and c above

(2.3) To use the template construct, you must put which of the following expressions before the class definition?

(a) class template
(b) <template, class, ItemType>
(c) template <class ItemType>
(d) <template <class ItemType>>

(2.4) Given the function definition

```
void Twist(int& a, int b)
{
 int c;

 c = a + 2;
 a = a * 3;
 b = c + a;
}
```

what is the output of the following code fragment that invokes *Twist* ?

```
r = 1;
s = 2;
t = 3;
Twist(t, s);
cout << r << ’ ’ << s << ’  ’ << t << endl;
```

(a) 1 14 9
(b) 5 14 3
(c) 1 10 3
(d) 1 14 3
(e) none of the above

3. **Short answer** (5 pts each)

(3.1) Describe the running time of the code segment shown below in terms of big-O. You can assume that each list contains N elements. To get credit, you must justify your answer.

```
UsortedType<int> list1;
SortedType<int> list2;

 ...

sum=0;
for(i=0; i<N; i++) {

  if(!list1.IsFull())
    list1.InsertItem(i);

  if(!list2.IsFull())
    list2.InsertItem(i);
}
```

(3.2) 2D Arrays can be allocated either statically or dynamically. Are static and dynamic 2D arrays stored in memory the same way? Explain. Why is the number of columns required in the function prototype/heading when passing a static 2D array as a parameter to a function?

(3.3) Demonstrate the binary search algorithm on the list (array-based) shown below. The element to be retrieved is 10 (**Note:** I do not want the code).

	INIT		**ITER 1**		**ITER 2**		**ITER 3**		**ITER 4**
0	30	0	30	0	30	0	30	0	30
1	31	1	31	1	31	1	31	1	31
2	44	2	44	2	44	2	44	2	44
3	54	3	54	3	54	3	54	3	54
4	96	4	96	4	96	4	96	4	96
5	101	5	101	5	101	5	101	5	101
6	220	6	220	6	220	6	220	6	220

(3.4) In programming assignment 1, you implemented an algorithm to compute the negative of an image. Describe in simple words how this algorithm works. What is its complexity using big-O notation? Justify your answer.

(3.5) Is the linked-list implementation of a queue always more efficient (in terms of memory) than the array-based implementation ? Why or why not ?

4. **Code** (15 pts) Implement a *client* function called *copyStack* that copies the contents of one stack into another. The function must have two arguments of type StackType, one for the source stack and one for the destination stack. The order of the elements in the two stacks must be identical. The specification of *StackType* is given below (please note that no copy constructor is provided).

```
template<class ItemType>
class StackType {
 public:
   StackType();
   ~StackType();
   void MakeEmpty();
   bool IsEmpty() const;
   bool IsFull() const;
   void Push(ItemType);
   void Pop(ItemType&);
 private:
   NodeType<ItemType>* topPtr;
};
```

5. **Code** (15 pts) Implement the copy-constructor for the class *UnsortedType*. The specification of *UnsortedType* is given below.

```
template<class ItemType>
class UnsortedType {
 public:
    UnsortedType();
    ˜UnsortedType();
    void MakeEmpty();
    bool IsFull() const;
    int LengthIs() const;
    void RetrieveItem(ItemType&, bool&);
    void InsertItem(ItemType);
    void DeleteItem(ItemType);
    void ResetList();
    void GetNextItem(ItemType&);
    bool IsLastItem()
 private:
    int length;
    NodeType<ItemType> *listData;
    NodeType<ItemType> *currentPos;
};

template<class ItemType>
struct NodeType {
  ItemType info;
  NodeType* next;
};
```