Test 1: CPS 103

Owen Astrachan

February 11, 1994

Name: ________________________________

Honor code acknowledgement (signature) ________________________________

<table>
<thead>
<tr>
<th></th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>14 pts.</td>
<td></td>
</tr>
<tr>
<td>Bonus extra</td>
<td>5 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>50 pts.</td>
<td></td>
</tr>
</tbody>
</table>
PROBLEM 1:  (Vocabulary: 12 points)

For each of the words/phrases below, circle the definition that is the best description as it pertains in the context of computer science, programming, and C/C++.

1. header file
   (a) a file that must be compiled ahead of all other files before linking can occur when a Makefile is used.
   (b) a file with declarations and prototypes used for importing needed information about classes and functions into a C or C++ program.
   (c) a special kind of C++ class that uses polymorphism and inheritance to implement a building block to be used in many programs.

2. built-in types
   (a) a type that the user builds from scratch to use in a program instead of relying on the system-defined type, e.g., using "String.h" instead of <String.h>.
   (b) A type that is part of the standard C or C++ language (e.g., int or double) as opposed to one defined by the user.
   (c) A class with overloaded operators as in the BigInt class.

3. precondition
   (a) steps taken to ensure that a program will execute correctly
   (b) a statement that indicates what must be true in order for a function to work as intended
   (c) a medical ailment suffered at birth (e.g., by the mathematician and logician Augustus de Morgan)

4. hashing
   (a) A method of maintaining a table that supports $O(1)$ search operations in the average case.
   (b) A method of generating random numbers based on maintaining a seed and performing a series of exclusive-or operations on the seed.
   (c) A method of turning left-over food into inedible glop.
PROBLEM 2:  \((ListSum: 12\) points\)

Linked lists of integers are implemented using the following definition:

```c
struct Node{
    int info;
    Node * next;
};
```

Assume that the function \texttt{ListSum}, whose specification is given below, has been written.

```c
int ListSum(Node * list)
// precondition: list represents \(a_1, a_2, \ldots, a_n\)
// postcondition: returns \(a_1 + a_2 + \cdots + a_n\)
// exception: returns 0 if list is empty
```

Part A: \((4\) points\)

You are to write the function \texttt{VectorSum} whose header is given below. \texttt{VectorSum} returns the cumulative sum of all lists in the parameter \texttt{vlist}, an array of linked lists. For example, if \texttt{vlist} is as diagrammed below, the expression \texttt{VectorSum(vlist, 3)} should evaluate to 19 and the expression \texttt{VectorSum(vlist, 4)} should evaluate to 40.

```
4
\downarrow
3
\downarrow
2
\downarrow
5 7
\downarrow
6 8
\downarrow
4
\downarrow
2
```

Complete \texttt{VectorSum} below the following header. In writing \texttt{VectorSum} you may call the function \texttt{ListSum}.

```c
int VectorSum(Node * vlist[], int numElts)
// precondition: numElts = number of elements in vlist
// postcondition: returns sum of all linked lists in vlist
{
```
**Part B:** (3 points) Suppose that the sum of two linked lists of integers is defined to be the sum of all the nodes in each list. Thus $vlist[0] + vlist[3] = 29$ using the lists diagrammed above. The header for an operator implementation of this kind of addition is given below. Fill in the body of the function using a maximum of three statements (you may call `ListSum`).

```c
int operator + (Node * first, Node * second)
// postcondition: returns first + second
{
}
```

**Part C:** (3 points) Suppose multiplication of a linked-list by an integer is defined in the “expected” manner, i.e., for a linked list $L$ and an integer $n$.

$$ L \times n = L + L + \cdots + L $$

Complete the implementation of `operator *` below implementing multiplication in this manner. You may use `operator +` from part B, assume it works as specified regardless of what you wrote. You may also call the function `ListSum`.

```c
int operator * (Node * list, int n)
// postcondition: returns list * n
{
}
```

**Part D:** (3 points) A user wants to be able to mix types in an arithmetic expression involving linked-lists and integers. Thus the expression

```c
cout << vlist[0] + 7 << endl;
```

should result in 15 being printed. Briefly describe a mechanism that would allow such expressions to be evaluated without writing another `operator +`, i.e., using the one specified in Part B.
PROBLEM 3: (N-lists : (12 points))

Consider a list of integers in which the integers are stored in non-decreasing order and in which the integer $k$ occurs exactly $k$ times; these lists are called N-lists where $N$ is the largest integer in the list. For example, a 4-list is $(1,2,2,3,3,4,4,4,4)$ and a five list is a 4-list with $(5,5,5,5,5)$ appended to the end.

Part A: Exactly how many nodes are there in a 2000-list?

Part B: You are to write the function MakeNlist whose header is given below. The function should return a linked list (using the definitions from the previous problem) representing an N-list. Thus MakeNlist(4) should return a pointer to the first node of the list diagrammed below.

```
1 2 2 3 3 4 4 4 4
```

A 0-list is represented by a NULL pointer.

```c
Node * MakeNlist(int n)
// precondition : n >= 0
// postcondition: return pointer to first node of an n-list
{

```
PROBLEM 4: (Priority List: 15 points)

This question involves reasoning about two implementations of priority lists. A priority list is a collection of items, each of which contains data and a unique integer priority. The “maximal” item in a priority list is the item whose integer priority has the highest value.

A priority list supports the following operations.

<table>
<thead>
<tr>
<th>operation</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(constructor)</td>
<td>initialize priority list to be empty</td>
</tr>
<tr>
<td>isEmpty</td>
<td>return 1 if the priority list is empty; else return 0</td>
</tr>
<tr>
<td>insert</td>
<td>insert a given item (data and priority) into the priority list</td>
</tr>
<tr>
<td>max</td>
<td>return the data and priority of the “maximal” item in the priority list</td>
</tr>
<tr>
<td>deleteMax</td>
<td>delete the “maximal” item from the priority list</td>
</tr>
</tbody>
</table>

A complete class definition and commented prototypes for some member functions are given before Part B.

Consider the following two methods for implementing priority lists:

**Method 1** A priority list is an unsorted linked list. The Insert operation inserts a new node at the beginning of the linked list pointed to by first.

**Method 2** A priority list is a linked list sorted by priority field from highest to lowest (so the first node in the list is always the priority list’s maximal item). The Insert operation inserts a new node at the appropriate place in the linked list pointed to by first so that the list remains sorted by priority.

**Part A (6 points):**

For each of the operations on priority lists listed below, indicate the worst-case complexity of the operation using $O$-notation. Briefly justify each answer. (Assume that the number of items in a priority list is $n$).

**Method 1**

<table>
<thead>
<tr>
<th>complexity</th>
<th>justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>constructor</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

Insert

DeleteMax

**Method 2**

<table>
<thead>
<tr>
<th>constructor</th>
</tr>
</thead>
</table>

Insert

DeleteMax | $O(1)$
Priority lists are implemented using the following definitions.

```cpp
struct Node{
    DataType data;
    int priority;
    Node * next;
};

class Plist
{
    private:
        Node * first; // NO header node used
    public:
        Plist(); // initially empty list
        int IsEmpty() const; // return 1 if empty, else 0
        void Insert(const DataType &, int); // insert into list
        void Max(DataType &, int &) const; // return maximal element
        void DeleteMax(); // delete maximal element
};
```

Precise specifications for the operations described above are given below:

- **Plist::Plist()**
  // postcondition: represents an empty priority list

- **int Plist::IsEmpty() const**
  // postcondition: returns 1 priority list is empty, otherwise returns 0

- **void Plist::Insert(const DataType & info, int pri)**
  // postcondition: an item with data field info and priority field pri
  // has been added to the priority list

- **void Plist::Max(DataType & info, int & pri) const**
  // precondition: IsEmpty() == 0 (list has at least 1 item in it)
  // postcondition: the values of info and pri are those of the maximal item
  // of the priority list

- **void Plist::DeleteMax()**
  // precondition: IsEmpty() == 0 (list has at least 1 item in it)
  // postcondition: the maximal item of the priority list has been removed

**Part B (8 points):**

You must choose one of Method 1 or Method 2 and implement the member functions Insert and Max for the method you choose. A wise choice will make the code easier to write. You must use the same method for implementing priority lists for BOTH member functions.

Circle which method you choose:

- Method 1
- Method 2
void Plist::Insert(const DataType & info, int pri)
   // postcondition: an item with data field info and priority field pri
   // has been added to the priority list

void Plist::Max(DataType & info, int & pri) const
// precondition: isEmpty() == 0 (list has at least 1 item in it)
// postcondition: the values of info and pri are those of the maximal item
// of the priority list
{

}
**PROBLEM 5:** *(Extra Credit: 5 points)*

Consider a linked-list each of whose nodes contains an array of $n$ elements. Such a linked-list can be viewed as a “flexible” array, in which, if $n = 100$, items 0-99 are in the first node, 100-199 in the second node, etc.

**Part A** In an array, finding an element given its index is $O(1)$, in a linked-list this is an $O(k)$ operation (where the index is $k$). What is the complexity, in terms of $n$ and $k$ of this operation in the flexible array described above? (Justify your answer)

**Part B** If $n = 100$ and the only items in a flexible array have indices 1 and 10001, there will be lots of wasted space if space for all potential elements is stored. Describe a method that might be used to minimize such wasted memory.