Northern Michigan University (Marquette Co, MI)

CS201-01-22F: Introduction to C++ (Andrew A. Poe) Name: _____
Final Examination Solution Page 1/4 Monday 10 December 2022 7:00 P.M. EST

Time: 110 minutes

For the following programs, write the code as directed. Do not worry about #include files or minor syntactic errors such as semicolons or matching braces. However, your code should be as correct as possible. Make sure all special cases are handled correctly. Make sure you call and use all methods correctly. If you write more than one method to solve a problem, make sure you declare your headers correctly, but you don't have to worry about prototypes.

1. Write code for method **`string Max2LetterCombination (string s)`**. You are to compute and return the two letter combination that occurs most frequently in s. For example, if s were "ABABAB", your method should return "AB" since that occurs three times. The only other two letter combination in the string, "BA", occurs twice. If there is a tie for most frequent two letter combination, I don't care which one you return.

```
string Max2LetterCombination (string s) {

 string max = "";
 int maxct = 0;
 for (int i=0; i < s.length()-1; i++) {
  int ct = 0;
  string t = s.substr(i,2);
  for (int j=i; j < s.length()-1; j++)
   if (t==s.substr(j,2)) ct++;
  if (ct > maxct) {
   maxct = ct;
   max = t;
  }
 }
 return max;
}
```

2. Write the following method:

```
bool IsIdentity (double **M, int n);
```

M is a two-dimensional row of doubles with row length and column length both equal to n. An identity matrix is one in which all entries with identical row and column positions are 1, and all entries with differing row and column positions are 0. For example,

1 0 0
0 1 0 is an identity matrix.
0 0 1

```
bool IsIdentity (double **M, int n) {

 for (int i=0; i < n; i++)
  for (int j=0; j < n; j++) {
   if (i==j && M[i][j] != 1) return false;
```

```
   if (i!=j && M[i][j] !=0) return false;
  }
 return true;
}
```

3. Write the code for method **`int LL::CountTriples ()`**. LL is a linked list defined as follows:

```
class LL {
 private:
  LL *head;
};
```

```
class LLN {
 private:
  int i;
  LLN *next;
};
```

This method returns the number of times the same value occurs three times in a row in the linked list. For example, if the linked list were -->3-->3-->3-->4-->4-->4-->4, the method would return 3, since there are three 3's in a row, and there are three 4's in a row in two different ways, one beginning with the first 4 and one beginning with the second 4. Do not use loops; use recursion only. You may assume that appropriate accessors exist, but you must write all other helper methods that you use. You are not to modify the linked list in any way. You MAY NOT assume anything about the length of the linked list.

```
int LL::CountTriples () {

 if (!head) return 0;
 return head->CountTriples ();
}

int LLN::CountTriples () {

 int ct = 0;
 if (next) ct = next->CountTriples ();
 if (next && next->geti()==i && next->getnext() &&
     next->getnext()->geti()==i) ct++;
 return ct;
}
```

4. Write the code for method **`void LL::SwapWithHead (int p)`**. LL is a linked list defined above:

This method takes the pth. node in the linked list (counting from 0), and swaps it with the head. For example, if the linked list were -->2-->4-->6-->8-->10, SwapWithHead (3), would make the list -->8-->4-->6-->2-->10. You may assume that p will be appropriately within bounds. Do not use loops; use recursion only. You may assume that appropriate accessors and mutators exist, but you must write all other helper methods that you use. You are not to create or destroy nodes or modify the data fields of existing nodes. Solve this problem with pointer manipulation only.

```
void LL::SwapWithHead (int p) {
```

```
 if (p==0) return;
 if (p==1) {
  LLN * t = head->getnext();
  head->setnext (t->getnext());
  t->setnext (head);
  head = t;
  return;
 }
 LLN *b = head->BeforePth (p-1);
 LLN *t = b->getnext();
 LLN *s = head->getnext();
 b->setnext(head);
 head->setnext(t->getnext());
 t->setnext (s);
 head = t;
}

LLN * LLN::BeforePth (int p) {

 if (p==0) return this;
 return next->BeforePth (p-1);
}
```

5. Write code for the method **`char TreasureHunt (string f)`**. This method opens the file whose name is specified in *f*. The first four bytes of the file represent a location in the file at which a secret character may be found. You are to find and return that secret character, accessing as few characters in the file as possible to do it. For example, if the file begins 4957a6&78: . . ., the secret character is at position 4957.

```
char TreasureHunt (string f) {

 fstream fl (f,ios::in|ios::out|ios::binary);
 char c;
 int pos = 0;
 fl.seekg (0);
 for (int i=0; i < 4; i++) {
  fl.get(c);
  pos = 10*pos + (int)(c-'0');
 }
 fl.seekg (pos);
 fl.get (c);
 fl.close ();
 return c;
}
```

6. Given the following class:

```
class Stuff {

 public:
  virtual int cmp (Stuff *b) = 0;
  // a->cmp(b) > 0 if a > b.
```

```
  // a->cmp(b) < 0 if a < b.
  // a->cmp(b) == 0 if a=b.

};
```

Write the following method:

```
Stuff ** Merge (Stuff **A, int Asz, Stuff **B, int Bsz);
```

A and B are arrays of Stuff pointers, Asz, and Bsz are the lengths of those arrays. A and B are both already sorted from lowest to highest.

This returns a new array with containing all the stuff in A and all the stuff in B, but properly merged and sorted. For example if A were {2,3,5,7} and B were {1,4,6}, the method would return {1,2,3,4,5,6,7}. REMEMBER: A and B are ALREADY sorted!

```
Stuff ** Merge (Stuff **A, int Asz, Stuff **B, int Bsz) {

 Stuff ** C = new Stuff * [Asz+Bsz];
 int a=0, b = 0;
 for (int c=0; c < Asz+Bsz; c++)
  if (b==Bsz || a < Asz && A[a]->cmp(B[b]) <= 0) C[c] = A[a++];
  else C[c] = B[b++];
 return C;
}
```