AP Computer Science A
1999 Sample Student Responses

These materials were produced by Educational Testing Service (ETS), which develops and administers the examinations of the Advanced Placement Program for the College Board. The College Board and Educational Testing Service (ETS) are dedicated to the principle of equal opportunity, and their programs, services, and employment policies are guided by that principle.

The College Board is a national nonprofit membership association dedicated to preparing, inspiring, and connecting students to college and opportunity. Founded in 1900, the association is composed of more than 3,900 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three million students and their parents, 22,000 high schools, and 3,500 colleges, through major programs and services in college admission, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT™, the Advanced Placement Program® (AP®), and Pacesetter®. The College Board is committed to the principles of equity and excellence, and that commitment is embodied in all of its programs, services, activities, and concerns.

A2
A

2.

(a) Write function `WordIndex`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If `word` is already in `wordList`, then `WordIndex` should return the index of `word` in `wordList`. Otherwise, `WordIndex` should return the index of the first string in `wordList` that comes after `word` in alphabetical order; it should return `numWords` if `word` comes after all of the strings in `wordList` in alphabetical order.

For example, assume that array `wordList` is as follows:

0	1	2	3
"apple"	"berry"	"pear"	"quince"

Function Call	Value Returned
`WordIndex("air", wordList, 4)`	0
`WordIndex("apple", wordList, 4)`	0
`WordIndex("orange", wordList, 4)`	2
`WordIndex("raspberry", wordList, 4)`	4

Complete function `WordIndex` below. Assume that `WordIndex` is called only with parameters that satisfy its precondition.

```
int WordIndex(const apstring & word,
              const apvector<apstring> & wordList, int numWords)
// precondition: wordList contains numWords strings in alphabetical
//               order, 0 ≤ numWords < wordList.length()
{
   int x;
   for(x=0; x< numWords; x++)
   {
      if( word <= wordList[x])
      {
        return x;
      }
   }
   return numWords;
}
```

GO ON TO THE NEXT PAGE

A

(b) Write function `InsertInOrder`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If the string `word` is already in `wordList`, `InsertInOrder` should not change any of its parameters. Otherwise, it should insert `word` into `wordList` in alphabetical order (i.e., all values greater than `word` should be moved one place to the right to make room for `word`), and it should also increment `numWords` by 1. Assume that `wordList.length()` is greater than `numWords`.

In the examples below, `numWords = 3` before the following call is made.

```
InsertInOrder("pear", wordList, numWords)
```

Before the call	After the call	
`wordList`	`wordList`	`numWords`
`"apple" "berry" "quince"`	`"apple" "berry" "pear" "quince"`	4
`"apple" "berry" "pear"`	`"apple" "berry" "pear"`	3
`"apple" "fig" "peach"`	`"apple" "fig" "peach" "pear"`	4
`"quince" "raisin" "tart"`	`"pear" "quince" "raisin" "tart"`	4

In writing `InsertInOrder`, you may include calls to function `WordIndex` specified in part (a). Assume that `WordIndex` works as specified, regardless of what you wrote in part (a).

Complete function `InsertInOrder` below. Assume that `InsertInOrder` is called only with parameters that satisfy its precondition.

```
void InsertInOrder(const apstring & word,
                   apvector <apstring> & wordList, int & numWords)
// precondition:  wordList contains numWords strings in alphabetical
//                order, 0 ≤ numWords < wordList.length()
// postcondition: if word was already in wordList, then wordList and
//                numWords are unchanged;
//                otherwise, word has been inserted into wordList in
//                sorted order, and numWords has.been incremented by 1
{
   int place = WordIndex(word, wordList, numWords), x;
  if(place == numWords || wordList[place] != word)
  {
   for(x = numWords; x > place; x--)
    {
       wordList[x] = wordList[x-1];
    }

    wordList[place] = word;
    numWords++;
  }
}
```

GO ON TO THE NEXT PAGE

2.

(a) Write function `WordIndex`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If `word` is already in `wordList`, then `WordIndex` should return the index of `word` in `wordList`. Otherwise, `WordIndex` should return the index of the first string in `wordList` that comes after `word` in alphabetical order; it should return `numWords` if `word` comes after all of the strings in `wordList` in alphabetical order.

For example, assume that array `wordList` is as follows:

0	1	2	3
"apple"	"berry"	"pear"	"quince"

Function Call	Value Returned
`WordIndex("air", wordList, 4)`	0
`WordIndex("apple", wordList, 4)`	0
`WordIndex("orange", wordList, 4)`	2
`WordIndex("raspberry", wordList, 4)`	4

Complete function `WordIndex` below. Assume that `WordIndex` is called only with parameters that satisfy its precondition.

```
int WordIndex(const apstring & word,
              const apvector<apstring> & wordList, int numWords)
// precondition: wordList contains numWords strings in alphabetical
//               order, 0 ≤ numWords < wordList.length()
{
  int K;
  for (K=0; K<numWords; K++)
   { if (wordList[K] == word)
       return K;
     if (word < wordList[K])
       return K;
      if (word > wordList[K])
       return numWords;
   }
}
```

A2 G

(b) Write function `InsertInOrder`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If the string `word` is already in `wordList`, `InsertInOrder` should not change any of its parameters. Otherwise, it should insert `word` into `wordList` in alphabetical order (i.e., all values greater than `word` should be moved one place to the right to make room for `word`), and it should also increment `numWords` by 1. Assume that `wordList.length()` is greater than `numWords`.

In the examples below, `numWords = 3` before the following call is made.

```
InsertInOrder("pear", wordList, numWords)
```

Before the call	After the call	
`wordList`	`wordList`	`numWords`
`"apple" "berry" "quince"`	`"apple" "berry" "pear" "quince"`	4
`"apple" "berry" "pear"`	`"apple" "berry" "pear"`	3
`"apple" "fig" "peach"`	`"apple" "fig" "peach" "pear"`	4
`"quince" "raisin" "tart"`	`"pear" "quince" "raisin" "tart"`	4

In writing `InsertInOrder`, you may include calls to function `WordIndex` specified in part (a). Assume that `WordIndex` works as specified, regardless of what you wrote in part (a).

Complete function `InsertInOrder` below. Assume that `InsertInOrder` is called only with parameters that satisfy its precondition.

```
void InsertInOrder(const apstring & word,
                   apvector <apstring> & wordList, int & numWords)
// precondition:  wordList contains numWords strings in alphabetical
//                order, 0 ≤ numWords < wordList.length()
// postcondition: if word was already in wordList, then wordList and
//                numWords are unchanged;
//                otherwise, word has been inserted into wordList in
//                sorted order, and numWords has been incremented by 1
{
  int K, Index, L;
  for(K=0; K< numWords; K++)
  {
    if(word != wordList[K])
    {
      Index = WordIndex(word, wordList, numWords);
    }
  }
  for(L=numWords; L>Index; L--)
    wordList[L]= wordList[L-1];
  wordList[Index] = word;

}
```

A2 I

2.

(a) Write function `WordIndex`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If `word` is already in `wordList`, then `WordIndex` should return the index of `word` in `wordList`. Otherwise, `WordIndex` should return the index of the first string in `wordList` that comes after `word` in alphabetical order; it should return `numWords` if `word` comes after all of the strings in `wordList` in alphabetical order.

For example, assume that array `wordList` is as follows:

0	1	2	3
"apple"	"berry"	"pear"	"quince"

Function Call	Value Returned
`WordIndex("air", wordList, 4)`	0
`WordIndex("apple", wordList, 4)`	0
`WordIndex("orange", wordList, 4)`	2
`WordIndex("raspberry", wordList, 4)`	4

Complete function `WordIndex` below. Assume that `WordIndex` is called only with parameters that satisfy its precondition.

```
int WordIndex(const apstring & word,
              const apvector<apstring> & wordList, int numWords)
// precondition: wordList contains numWords strings in alphabetical
//               order, 0 ≤ numWords < wordList.length()
```

```
{
  numWords = wordlist.length()-1
  int count=0;
  for(numWords; numWords>0; numWords--)
  {
    if (word<=wordlist[numWords])
      count=numWords;
  }
  return count;
```

(b) Write function `InsertInOrder`, as started below. The array `wordList` contains `numWords` strings in alphabetical order. If the string `word` is already in `wordList`, `InsertInOrder` should not change any of its parameters. Otherwise, it should insert `word` into `wordList` in alphabetical order (i.e., all values greater than `word` should be moved one place to the right to make room for `word`), and it should also increment `numWords` by 1. Assume that `wordList.length()` is greater than `numWords`.

In the examples below, `numWords = 3` before the following call is made.

```
InsertInOrder("pear", wordList, numWords)
```

Before the call	After the call	
`wordList`	`wordList`	`numWords`
`"apple" "berry" "quince"`	`"apple" "berry" "pear" "quince"`	4
`"apple" "berry" "pear"`	`"apple" "berry" "pear"`	3
`"apple" "fig" "peach"`	`"apple" "fig" "peach" "pear"`	4
`"quince" "raisin" "tart"`	`"pear" "quince" "raisin" "tart"`	4

In writing `InsertInOrder`, you may include calls to function `WordIndex` specified in part (a). Assume that `WordIndex` works as specified, regardless of what you wrote in part (a).

Complete function `InsertInOrder` below. Assume that `InsertInOrder` is called only with parameters that satisfy its precondition.

```
void InsertInOrder(const apstring & word,
                   apvector <apstring> & wordList, int & numWords)
// precondition:  wordList contains numWords strings in alphabetical
//                order, 0 ≤ numWords < wordList.length()
// postcondition: if word was already in wordList, then wordList and
//                numWords are unchanged;
//                otherwise, word has been inserted into wordList in
//                sorted order, and numWords has been incremented by 1
{
  int num=0
  int count=0;
  for(num=WordList.length-1; num>=0; num--)
  {
    if word==wordList[num]
      int num =-1;
   if (word< wordList[num]
      count =num;
  }
  if(count!=-1)
  { for(num=0; num=numWords; num++)
   {
    wordList[count+num+1]= wordList[count+num] ;
   }
   wordlist[count] = word;
}
```


GO ON TO THE NEXT PAGE