AP Computer Science A
2000 Student Samples

A3 A

(a) Write free function `Occurrences`, as started below. `Occurrences` returns the number of times that `word` appears in `WordCollection C`. If `word` is not in `C`, `Occurrences` should return 0.

In writing `Occurrences`, you may call any of the member functions of the `WordCollection` class. Assume that the member functions work as specified.

Complete function `Occurrences` below.

```
int Occurrences(const WordCollection & C, const apstring & word)
// postcondition: returns the number of occurrences of word in C
{ int K;
   int num=0;
   for(K=1; K<=C.Size(); K++)
   { if (C.FindKth(K)==word)
          num++;
   }
 }
```

Part (b) begins on page 12.

GO ON TO THE NEXT PAGE.

(b) Write free function `RemoveDuplicates`, as started below. `RemoveDuplicates` removes all but one occurrence of `word` from `C`. If `word` is not in collection `C`, then `RemoveDuplicates` does nothing.

In writing `RemoveDuplicates`, you may call function `Occurrences` specified in part (a). Assume that `Occurrences` works as specified, regardless of what you wrote in part (a).

Complete function `RemoveDuplicates` below.

```
void RemoveDuplicates(WordCollection & C, const apstring & word)
// postcondition: if word is present in C, all but one occurrence
//                is removed; otherwise, C is unchanged
{while (Occurrences(C,word) >1)
   {
      C.Remove(word);
   }
 }
```

GO ON TO THE NEXT PAGE.

(c) Write free function `MostCommon`, as started below. `MostCommon` returns the word that appears most often in the collection. If there is more than one such word, return any one of them. You may assume that `C` is not empty.

In writing `MostCommon`, you may call function `Occurrences` specified in part (a). Assume that `Occurrences` works as specified, regardless of what you wrote in part (a).

Complete function `MostCommon` below.

```
apstring MostCommon(const WordCollection & C)
// precondition:  C is not empty
// postcondition: returns the word that appears most often in C;
//                if there is more than one such word,
//                returns any one of those words
```

```
{ int j=1;
  int k=0;
  apstring word;
  apstring MCom;
  while(j<=L.Size())
  { word=L.findKth(j);
    if(Occurrences(L,word)>k)
    { MCom=word;
      k=Occurrences(L,word);
    }
    j++;
  }
  return MCom;
}
```

A3 B

(a) Write free function `Occurrences`, as started below. `Occurrences` returns the number of times that `word` appears in `WordCollection C`. If `word` is not in `C`, `Occurrences` should return 0.

In writing `Occurrences`, you may call any of the member functions of the `WordCollection` class. Assume that the member functions work as specified.

Complete function `Occurrences` below.

```
int Occurrences(const WordCollection & C, const apstring & word){
// postcondition: returns the number of occurrences of word in C
      int found = 0;
      for(int i = 0; i < C.size()-1; i++)
            if(C.findKth[i] == C.FindKth[i+1])
                  found ++;
      return fund;
      }
```

Part (b) begins on page 12.

GO ON TO THE NEXT PAGE.

A3 B

(b) Write free function `RemoveDuplicates`, as started below. `RemoveDuplicates` removes all but one occurrence of `word` from `C`. If `word` is not in collection `C`, then `RemoveDuplicates` does nothing.

In writing `RemoveDuplicates`, you may call function `Occurrences` specified in part (a). Assume that `Occurrences` works as specified, regardless of what you wrote in part (a).

Complete function `RemoveDuplicates` below.

```
void RemoveDuplicates(WordCollection & C, const apstring & word)
// postcondition: if word is present in C, all but one occurrence
//                is removed; otherwise, C is unchanged
      int numWords = Occurrences(C, word);
      while(numWords > 1) {
            Remove(word);
            numWords--;
           } //end while
         } // end function
```

GO ON TO THE NEXT PAGE.

A3 B

(c) Write free function `MostCommon`, as started below. `MostCommon` returns the word that appears most often in the collection. If there is more than one such word, return any one of them. You may assume that `C` is not empty.

In writing `MostCommon`, you may call function `Occurrences` specified in part (a). Assume that `Occurrences` works as specified, regardless of what you wrote in part (a).

Complete function `MostCommon` below.

```
apstring MostCommon(const WordCollection & C) {
// precondition:  C is not empty
// postcondition: returns the word that appears most often in C;
//                if there is more than one such word,
//                returns any one of those words
         int  most =0, numWords;
        apstring  temp, mostWords;
          for(int i =0; i <C.size()-1; i++){
               temp= C. FindKth(i);
               numWords= Occurrences(C, temp);
               if(numWords> most){
                  most = numWords;
                  mostWords=temp; // temp has most occurrences
                }//end  if
             }//end for
            return  mostWords;
          }// end function
```

GO ON TO THE NEXT PAGE.

A3 C

(a) Write free function `Occurrences`, as started below. `Occurrences` returns the number of times that `word` appears in `WordCollection C`. If `word` is not in `C`, `Occurrences` should return 0.

In writing `Occurrences`, you may call any of the member functions of the `WordCollection` class. Assume that the member functions work as specified.

Complete function `Occurrences` below.

```
int Occurrences(const WordCollection & C, const apstring & word)
// postcondition: returns the number of occurrences of word in C
{
  int i , count;
  for (i=0 ; i < S.Size() ; i++ )
  {
      if (s[i] == word)
          count ++;
  }
 return count;
}
```

Part (b) begins on page 12.

GO ON TO THE NEXT PAGE.

A3 C

(b) Write free function `RemoveDuplicates`, as started below. `RemoveDuplicates` removes all but one occurrence of `word` from `C`. If `word` is not in collection `C`, then `RemoveDuplicates` does nothing.

In writing `RemoveDuplicates`, you may call function `Occurrences` specified in part (a). Assume that `Occurrences` works as specified, regardless of what you wrote in part (a).

Complete function `RemoveDuplicates` below.

```
void RemoveDuplicates(WordCollection & C, const apstring & word)
// postcondition: if word is present in C, all but one occurrence
//                is removed; otherwise, C is unchanged
{
 int i= 0;
 if (s. Occurren ces()>1)
 {
      for (i= s.Occurences() -1; i>= 0; i--)
        s. Remove (word)

 }
}
```

A3 C

(c) Write free function `MostCommon`, as started below. `MostCommon` returns the word that appears most often in the collection. If there is more than one such word, return any one of them. You may assume that `C` is not empty.

In writing `MostCommon`, you may call function `Occurrences` specified in part (a). Assume that `Occurrences` works as specified, regardless of what you wrote in part (a).

Complete function `MostCommon` below.

```
apstring MostCommon(const WordCollection & C)
// precondition:  C is not empty
// postcondition: returns the word that appears most often in C;
//                if there is more than one such word,
//                returns any one of those words
{int count, times, temp;
 apstring position;
 for (count =0; count <= C.size(); count ++)
 {
    tem Occurrence (C, word[count])
    if temp > times
          position = word;
 }
return (position);
}
```