AP Computer Science A
2000 Student Samples

The materials included in these files are intended for non-commercial use by AP teachers for course and exam preparation; permission for any other use must be sought from the Advanced Placement Program. Teachers may reproduce them, in whole or in part, in limited quantities, for face-to-face teaching purposes but may not mass distribute the materials, electronically or otherwise. These materials and any copies made of them may not be resold, and the copyright notices must be retained as they appear here. This permission does not apply to any third-party copyrights contained herein.
(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.

```cpp
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.
(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.

```c
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);
  }
}
```
(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.

```c
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 MCon;
  while (j <= C.size())
  {
    word = C.findKth(j);
    if (Occurrences(C, word) > k)
    {
      MCon = word;
      k = Occurrences(C, word);
    }
    j++;
  }
  return MCon;
}
```
(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.

```c
int Occurrences(const WordCollection & C, const apestr & word) { // postcondition: returns the number of occurrences of word in C
    int n = 0;
    for (int i = 0; i < C.size() + 1; i++)
        if (C[i] == word)
            f[n] += 1;
    return n;
}
```

Part (b) begins on page 12.
(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.

```c
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
```
(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.

```cpp
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.FindWord(i);
        numWords = *Occurrences C. temp);  
        if (numWords > most) {
            most = numWords;
            mostWords = temp;  // temp has most occurrences
        }
    } // end for
    // end if
    return mostWords;
} // end function
```
(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.

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

Part (b) begins on page 12.
(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.

```c
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.Occurrences(word) > 1)
    { 
      for (i = s.Occurrences(word) - 1; i >= 0; i--)
        s.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.

```cpp
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++)
    temp = occurrences(C, word[count]);
if temp > times
    position = word[i]
return (position);
```