



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.

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.

Copyright © 2001 by College Entrance Examination Board. All rights reserved. College Board, Advanced Placement Program, AP, and the acorn logo are registered trademarks of the College Entrance Examination Board.

- (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)
// precondition: 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.

```
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.

```
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 <= C.Size())
  {
    word = C.findKth(j);
    if (Occurrences(L, word) > k)
    {
      MCom = word;
      k = Occurrences(L, word);
    }
    j++;
  }
  return MCom;
}
```

- (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.FindKthEi() == C.FindKthEi(i+1))
            found++;
    return found;
}
```

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.

```
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.

```
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.GetSize() - 1; i++) {
        temp = C.FindWith(i);
        numWords = Occurrences(C, temp);
        if (numWords > most) {
            most = numWords;
            mostWords = temp; // temp has most occurrences
        } // end if
    } // end for
    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.

```
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 < C.size(); i++)
    {
        if (C[i] == word)
            count++;
    }
    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.

```
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 (C.Occurrences(word) > 1)
    {
        for (i = C.Occurrences(word) - 1; i >= 0; i--)
            C.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++)
{
    temp = Occurrences(C, word[count]);
    if (temp > times)
        position = word[count];
}
return position;

```

GO ON TO THE NEXT PAGE.