



AP Computer Science AB 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 function `GiveAdvice`, as started below. If parameter `T` points to a question node, `GiveAdvice` should print the question, read a 'Y' or 'N' response from `cin`, and then continue down the appropriate path in the tree. (Assume that users always respond with either 'Y' or 'N'.) If parameter `T` points to an answer node, `GiveAdvice` should print the answer.

For example, if `T` is the tree given above and the user always answers 'Y', then the output would be as follows (where user input is given in boldface).

```
Do you want to see a romantic movie? Y
A classic movie? Y
Casablanca
```

In writing `GiveAdvice`, you may call function `IsQuestionNode` specified at the beginning of this question. Assume that function `IsQuestionNode` works as specified.

Complete function `GiveAdvice` below.

```
void GiveAdvice(AdviceNode * T)
// precondition: T is not NULL
{
    while(IsQuestionNode(T))
    {
        char a;
        cout << T->QorA << "\n";
        cin >> a;
        if(a == 'Y')
            T = T->yes;
        else
            T = T->no;
    }
    cout << T->QorA;
}
```

Part (b) begins on page 20.

- (b) Write function `TracePath`, as started below. `TracePath` should search for the parameter `movie` in decision tree `T`, recursively tracing the path from the root of the tree to the movie. If the movie is in tree `T`, `TracePath` pushes the questions and answers along that path onto the parameter `pathStack` and returns `true`. If the movie is not in tree `T`, `TracePath` returns `false`. For question nodes, the question and its answer should be stored together as a single string in the stack.

For example, if `T` is a pointer to the root of the decision tree shown at the beginning of this question and `S` is an empty stack, then after the call `TracePath(T, "Jaws", S)`, `S` would contain the following elements.

```
top -> "Do you want to see a romantic movie? No"
      "Science Fiction? No"
      "Suspense? Yes"
      "Jaws"
```

In writing `TracePath`, you may call function `IsQuestionNode` specified at the beginning of this question. Assume that function `IsQuestionNode` works as specified. You may also assume that a given movie appears at most once in the decision tree.

Complete function `TracePath` below.

```
bool TracePath(AdviceNode * T, const apstring & movie,
               apstack<string> & pathStack)
// precondition: T is not NULL
```

```
{
    if (!IsQuestionNode(T))
    {
        if (T->QorA == movie)
        {
            pathStack.push(T->QorA);
            return true;
        }
        return false;
    }
    if (TracePath(T->yes, movie, pathStack))
    {
        pathStack.push(T->QorA + apstring(" Yes"));
        return true;
    }
    if (TracePath(T->no, movie, pathStack))
    {
        pathStack.push(T->QorA + apstring(" Yes"));
        return true;
    }
    return false;
}
```

- (a) Write function `GiveAdvice`, as started below. If parameter `T` points to a question node, `GiveAdvice` should print the question, read a 'Y' or 'N' response from `cin`, and then continue down the appropriate path in the tree. (Assume that users always respond with either 'Y' or 'N'.) If parameter `T` points to an answer node, `GiveAdvice` should print the answer.

For example, if `T` is the tree given above and the user always answers 'Y', then the output would be as follows (where user input is given in boldface).

```
Do you want to see a romantic movie? Y
A classic movie? Y
Casablanca
```

In writing `GiveAdvice`, you may call function `IsQuestionNode` specified at the beginning of this question. Assume that function `IsQuestionNode` works as specified.

Complete function `GiveAdvice` below.

```
void GiveAdvice(AdviceNode * T)
{ // precondition: T is not NULL
  char ans;

  if (IsQuestionNode(T))
  {
    cout << T.QorA;
    cin << ans;
    if (ans == 'n') GiveAdvice(T->no);
    if (ans == 'y') GiveAdvice(T->yes);
  }
  else {
    cout << T.QorA;
  }
}
```

Part (b) begins on page 20.

- (b) Write function `TracePath`, as started below. `TracePath` should search for the parameter `movie` in decision tree `T`, recursively tracing the path from the root of the tree to the movie. If the movie is in tree `T`, `TracePath` pushes the questions and answers along that path onto the parameter `pathStack` and returns `true`. If the movie is not in tree `T`, `TracePath` returns `false`. For question nodes, the question and its answer should be stored together as a single string in the stack.

For example, if `T` is a pointer to the root of the decision tree shown at the beginning of this question and `S` is an empty stack, then after the call `TracePath(T, "Jaws", S)`, `S` would contain the following elements.

```
top -> "Do you want to see a romantic movie? No"
      "Science Fiction? No"
      "Suspense? Yes"
      "Jaws"
```

In writing `TracePath`, you may call function `IsQuestionNode` specified at the beginning of this question. Assume that function `IsQuestionNode` works as specified. You may also assume that a given movie appears at most once in the decision tree.

Complete function `TracePath` below.

```
bool TracePath(AdviceNode * T, const apstring & movie,
               apstack<string> & pathStack)
{ // precondition: T is not NULL

    if (T.QorA == movie) return true;

    pathStack.push(T.QorA);
    return (TracePath(T->no, movie, pathStack) || (TracePath(T->yes, movie, pathStack)));
    return false;
}
```

- (a) Write function `GiveAdvice`, as started below. If parameter `T` points to a question node, `GiveAdvice` should print the question, read a 'Y' or 'N' response from `cin`, and then continue down the appropriate path in the tree. (Assume that users always respond with either 'Y' or 'N'.) If parameter `T` points to an answer node, `GiveAdvice` should print the answer.

For example, if `T` is the tree given above and the user always answers 'Y', then the output would be as follows (where user input is given in boldface).

Do you want to see a romantic movie? **Y**
 A classic movie? **Y**
 Casablanca

In writing `GiveAdvice`, you may call function `IsQuestionNode` specified at the beginning of this question. Assume that function `IsQuestionNode` works as specified.

Complete function `GiveAdvice` below.

```
void GiveAdvice(AdviceNode * T)
// precondition: T is not NULL
{
  char ans = 'N';
  if (IsQuestionNode(T))
  {
    cout << (T->QorA) << endl;
    cin >> ans;
    if (ans == 'Y')
      GiveAdvice(T->left);
    GiveAdvice(T->right);
  }
  else
    cout << (T->QorA) << endl;
}
```

Part (b) begins on page 20.

- (b) Write function `TracePath`, as started below. `TracePath` should search for the parameter `movie` in decision tree `T`, recursively tracing the path from the root of the tree to the movie. If the movie is in tree `T`, `TracePath` pushes the questions and answers along that path onto the parameter `pathStack` and returns `true`. If the movie is not in tree `T`, `TracePath` returns `false`. For question nodes, the question and its answer should be stored together as a single string in the stack.

For example, if `T` is a pointer to the root of the decision tree shown at the beginning of this question and `S` is an empty stack, then after the call `TracePath(T, "Jaws", S)`, `S` would contain the following elements.

```
top -> "Do you want to see a romantic movie? No"
      "Science Fiction? No"
      "Suspense? Yes"
      "Jaws"
```

In writing `TracePath`, you may call function `IsQuestionNode` specified at the beginning of this question. Assume that function `IsQuestionNode` works as specified. You may also assume that a given movie appears at most once in the decision tree.

Complete function `TracePath` below.

```
bool TracePath(AdviceNode * T, const apstring & movie,
               apstack<string> & pathStack)
// precondition: T is not NULL
{
    if (T->QorA == movie)
    { pathStack.push(T->QorA); return true; }
    else
    { TracePath(T->left, movie, pathStack);
      TracePath(T->right, movie, pathStack);
    }
    return false;
}
```