



AP Computer Science AB 2000 Scoring Guidelines

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.

2000 AP[®] Computer Science

A Question 4, AB Question 1

Part A:	GetCoordinates	2 pts
----------------	----------------	--------------

- +1 Find row and column
 - +1/2 attempt (must examine `ch` and `myMat`)
 - +1/2 correct (no loop errors)
- +1 construct and return `Point`
 - +1/2 attempt (must try to construct or assign to a `Point`)
 - +1/2 correct (must get attempt to find row and column)

Part B:	EncryptTwo	4 pts
----------------	------------	--------------

- +1 get coordinates
 - +1/2 attempt (must: refer to elements of `pair` and use return value as a `Point` throughout rest of part)
 - +1/2 correct
- +1 special case(s) – same columns, same rows
 - +1/2 attempt (to check that `int` coordinates lie on a line instead of at opposite corners of a box)
 - +1/2 correct (tests and handles same column case)
(note: same rows can be done as general case)
- +2 general case
 - +1 attempt (must set elements of an `apstring` using elements of `myMat`)
 - +1 correct (including return)

Part C:	EncryptWord	3 pts
----------------	-------------	--------------

- +1 loop over pairs
 - +1/2 attempt (must attempt to process pairs of consecutive letters from `word`)
 - +1/2 correct (stay in bounds, appropriate number of iterations)
- +2 update result
 - +1/2 form two character `apstring` from consecutive letters from `word` and use later in context of encryption
 - +1/2 call `EncryptTwo()` with parameter, or reimplement perfectly
 - +1/2 correct last char when odd length
 - +1/2 result assembled as an `apstring` in proper order and returned

Usage:

- 1 incorrect use of `apstring`

```
char c, d;  
apstring s, t(pair);
```

	<u>Correct examples</u>	<u>Incorrect examples</u>
1a.	<code>s = c; s += d; or</code>	<code>s = c + d;</code>
1b.	<code>s += c; s += d;</code>	
2.	<code>t[0] = c; t[1] = d;</code>	<code>s[0] = c; s[1] = d;</code>
3.	<code>word.substr(k, 2)</code>	<code>word.substr(k, k+1)</code>
4.	<code>c = word[k];</code>	<code>c = word.substr(k, 1);</code>

- 1/2 `Encryptor.myMat` in any part
- 1/2 modifying a const parameter (parts B and/or C); deduct at most once
- 0 confuse `()` and `[]`; confuse `->` and `.;` `apstring<char>`

2000 AP[®] Computer Science

AB Question 2

Part A: `operator <` **2 pts**

- +1 attempt (must have at least two true cases)
- +1 correct

Notes:

- `return (lhs < rhs);` loses both points
- Missing `.op` more than once loses correctness point (alternate solution)
- Comparing value field in struct loses correctness point
- returning `1` for true and `0` for false is OK
- perfect coding of `>` instead of `<` gets attempt, loses correctness

Part B: `InfixToPostfix` **7 pts**

- +1 loop through all values in `tempQ`
 - +1/2 attempt (must attempt to change state of `tempQ`)
 - +1/2 correct (must dequeue `tempQ` correctly)

- +1 handle `tok.op == NUMBER` case correctly
 (note: separate handling not needed, works as part of general case – get this point if attempt for general case given)
 - +1/2 attempt
 - +1/2 correct

- +2 process items on top of stack
 - +1 attempt (attempt to enqueue all operator tokens of equal or higher precedence)
 - +1 correct (Note: `<=`, `>=`, `>`, `==` not available for `TokenType`, lose correctness)

- +1 push current operator onto stack after some attempt to process top,
 current operator must be at top of stack

- +2 process tokens remaining on stack (must have loop attempt)
 - +1 attempt
 - +1 correct

Notes:

- misuse of `void` functions loses correctness point
- using `infixQ` instead of `tempQ` loses ½ point in usage (confused identifier)
- using `Token` or `TokenType` instead of `op` loses 1 point in usage (type error)
- assuming `infixQ` not empty is OK

	rhs	PLUS	TIMES	NUMBER
lhs				
	PLUS	false	true	true
	TIMES	false	false	true
	NUMBER	false	false	false

2000 AP[®] Computer Science
AB Question 3

Part A: `AddSigDigit` **3 pts**

- +1 create node with correct character
 - +1/2 attempt (must use `new` to get attempt)
 - +1/2 correct
- +1 link node at head of list
- +1 increment `myNumDigits`

Part B: `GetDigit` **4 pts**

- +1 precondition false case
 - +1/2 attempt
 - +1/2 correct
- +2 locate correct pointer
 - +1 attempt (must have a loop involving the list)
 - +1 correct (must not modify `myDigits`)
- +1 convert to value and return (must get attempt on locate to get these points)
 - +1/2 attempt (must access the list)
 - +1/2 correct

Part C : `~BigInt` (destructor) **2 pts**

- +1 traverse list
 - +1/2 attempt
 - +1/2 correct
- +1 delete each node
 - +1/2 attempt (delete must be inside the loop to get the attempt point)
 - +1/2 correct

Usage:

- 1/2 Missing `*` in pointer declaration

2000 AP[®] Computer Science
AB Question 4

Part A:	GiveAdvice	4 pts
----------------	------------	-------

- +1 test whether node contains question/movie in some meaningful context (check both children NULL is OK; check last char is ' ? ' is not OK).
T.IsQuestionNode() ok (no usage error for this)
- +1 print questions and get responses, print final movie
 - +1/2 attempt
 - +1/2 correct, including response
- +2 examine/traverse questions according to response, stop at movie
 - +1 attempt (must have both test response Y/N and move yes or no)
 - +1 correct (loop with correct stop)**Note:** while loop and recursion loses both examine points unless entirely correct

Part B:	TracePath	5 pts
----------------	-----------	-------

- +2 Handle base case/leaf node (movie)
 - +1 attempt (must have 2 out of 3: verifies is movie node; test movie match; push movie onto stack)
 - +1 correct (all three of above and correct return in both true and false cases)
- +3 Handle recursive case/interior nodes (questions)
 - +1 attempt (must have both: - at least one recursive call including node param
- attempt to push onto path stack)
 - +1 correct (must have all:
 - verify is question node before making recursive calls or have NULL base case check
 - both recursive calls with node param (missing other params, see usage)
 - if found, append Y/N to question and push onto stack))
 - +1 return true/false depending on results from recursive calls (must be in context of recursive calls)

Notes: No check to distinguish question from movie node loses both leaf and interior correctness. This can be checked by a call to `IsQuestionNode` or by checking both children are `NULL`.

The use of `apstack<string>` rather than `apstack<apstring>` as a parameter for `TracePath` was a typographical error. However, it did not affect the responses of students. Since the C++ class `string` has all the member functions defined for `apstring`, the code would be correct either way, and students were not penalized for using `string` rather than `apstring` throughout the problem. The few students who noted the error, assumed that `apstring` was intended and answered accordingly.

Usage:

- 1/2 Using `left` and `right` instead of `yes` and `no` consistently through problem.
- 1/2 Missing one or both of `movie` and `pathStack` parameters in recursive calls
- 0 Using `.` if `->` used correctly elsewhere
- 0 Missing `T` on field access if `T->...` used correctly elsewhere
- 0 Using `T.IsQuestionNode()` instead of `IsQuestionNode(T)`