AP® Computer Science AB
2001 Sample Student Responses

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 the `Window` member function `IsInBounds`, as started below. `IsInBounds` checks whether a single point is in the window.

For example, for any 5-by-4 `Window` `W`, the following table shows the results of several calls to `IsInBounds`.

Call	Return value
`W.IsInBounds(0, 0)`	`true`
`W.IsInBounds(2, 1)`	`true`
`W.IsInBounds(4, 3)`	`true`
`W.IsInBounds(5, 3)`	`false`
`W.IsInBounds(3, -1)`	`false`
`W.IsInBounds(8, 8)`	`false`

Complete function `IsInBounds` below.

```
bool Window::IsInBounds(int row, int col) const
// postcondition: returns true if the point (row, col) is
//                in this window;
//                otherwise, returns false
{
    return (row >= 0 && row < myNumRows) &&
           (col >= 0 && col < myNumCols);
}
```

Complete function ColorSquare below.

```
void Window::ColorSquare(int ULrow, int ULcol, int N, int val)
// postcondition: all points in this window that are also in the
//                N-by-N square with upper left corner
//                (ULrow, ULcol) have been set to val;
//                points in the square that are not in this
//                window are ignored
{
    int r, c;

    for (r = ULrow; r < ULrow + N; r++)
      for (c = ULcol; c < ULcol + N; c++)
          if (IsInBounds(r, c))
              myMat[r][c] = val;
}
```

Complete function Enlarge below.

```
void Enlarge(Window & W, const Rectangle & rect, int factor)
// precondition:  factor > 0
{
    int r, c;

    int newRow, newCol;

    for (r = 1;  r <= rect.numRows;  r++)
    {
        for (c = 1;  c <= rect.numCols;  c++)
        {
            newCol = ((rect.numCols - c ) * factor) + rect.ULcol;
            newRow = ((rect.numRows - r) * factor) + rect.ULrow;

            W.ColorSquare( newRow, newCol, factor, W.ValAt(rect.numRows - r +
                                                           rect.ULrow,
                                                           rect.numCols - c +
                                                           rect.ULcol));
        }
    }
}
```

(a) Write the `Window` member function `IsInBounds`, as started below. `IsInBounds` checks whether a single point is in the window.

For example, for any 5-by-4 `Window` `W`, the following table shows the results of several calls to `IsInBounds`.

Call	Return value
`W.IsInBounds(0, 0)`	`true`
`W.IsInBounds(2, 1)`	`true`
`W.IsInBounds(4, 3)`	`true`
`W.IsInBounds(5, 3)`	`false`
`W.IsInBounds(3, -1)`	`false`
`W.IsInBounds(8, 8)`	`false`

Complete function `IsInBounds` below.

```
bool Window::IsInBounds(int row, int col) const
// postcondition: returns true if the point (row, col) is
//                in this window;
//                otherwise, returns false
{
if ((row > myNumRows) || (col > myNumCols))
   return false;
if ((row<0) || (col<0))
    Return false;
  els return true;
}
```

Complete function ColorSquare below.

```
void Window::ColorSquare(int ULrow, int ULcol, int N, int val)
// postcondition: all points in this window that are also in the
//                N-by-N square with upper left corner
//                (ULrow, ULcol) have been set to val;
//                points in the square that are not in this
//                window are ignored
```

```
{
if (W.IsInBounds(ULrow,ULcol))
  {
for (int K=ULrow; K<ULrow+N; K++)
  {
  for(int J= ULcol; J<ULcol+N; J++)
  {
if (W.IsInBounds(K,J)
  {
W.myMat[K][J] = val;
```

Complete function Enlarge below.

```
void Enlarge(Window & W, const Rectangle & rect, int factor)
// precondition:  factor > 0
```

```
{
 apmatrix<int> temp = mymat;
int k=W.valAt(rect.ULrow, rect.ULcol);
 W.ColorSquare(ULrow, ULcol, factor, k);
for (int n =rect.ULrow+1; n<rect.numrows)
  {
   for (int m=rect.ULcol+1; m<rect.numcols)
   {
k=W.valAt(temp.n, temp.m);
 W.ColorSquare(n, m, factor, k);
     }
  }
}
```

(a) Write the `Window` member function `IsInBounds`, as started below. `IsInBounds` checks whether a single point is in the window.

For example, for any 5-by-4 `Window W`, the following table shows the results of several calls to `IsInBounds`.

Call	Return value
`W.IsInBounds(0, 0)`	`true`
`W.IsInBounds(2, 1)`	`true`
`W.IsInBounds(4, 3)`	`true`
`W.IsInBounds(5, 3)`	`false`
`W.IsInBounds(3, -1)`	`false`
`W.IsInBounds(8, 8)`	`false`

Complete function `IsInBounds` below.

```
bool Window::IsInBounds(int row, int col) const
// postcondition: returns true if the point (row, col) is
//                in this window;
//                otherwise, returns false
```

```
{bool check = true;
 if(row<0||row>myNumRows)
    check = false;
 else if(col<0||col>myNumCols)
    check = false;
 return check;}
```

Complete function ColorSquare below.

```
void Window::ColorSquare(int ULrow, int ULcol, int N, int val)
// postcondition: all points in this window that are also in the
//                N-by-N square with upper left corner
//                (ULrow, ULcol) have been set to val;
//                points in the square that are not in this
//                window are ignored
```

```
{ if (IsInBounds(ULrow, ULcol))
    {for (int r= ULrow; r< ULrow+N; r++)
      {for (int c =ULcol; c <=ULcol+N; c++)
        myMat[r][c] = val;}}}
```

Complete function Enlarge below.

```
void Enlarge(Window & W, const Rectangle & rect, int factor)
// precondition:  factor > 0
```

```
{ apvector<int> list((rect.ULrow*rect.ULcol), 0);
  for(int i=rect.ULrow; i<rect.numrows(); i++)
    {for(int j=rect.ULCol; j<rect.numcols(); j++)
       {for(int ct=0; ct<list.length(); ct++)
          List[ct] = rect[i][j];}}
W.ColorSquare(rect.ULrow, rect.ULcol, list[i];
```