AP® Computer Science A
2002 Sample Student Responses

The materials included in these files are intended for use by AP teachers for course and exam preparation in the classroom; 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 4,200 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®, and the Advanced Placement Program® (AP®). 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.

AA/AB1 A

Complete function `EmptySeatCount` below.

```
int Flight::EmptySeatCount(const apstring & seatType) const
// postcondition: returns the number of empty seats
//                whose type is seatType;
//                if seatType is "any", returns the
//                total number of empty seats
{  int row,col,counter = 0;
   int rowlen = mySeats.numrows(), collen = mySeats.numcols();
   for(row=0; row<rowlen; row++)
   {  for(col=0;col<collen;col++)
      {  if(mySeats[row][col].GetPassenger().GetName()=="")
         {  if(mySeats[row][col].GetType()==seatType ||
               mySeats[row][col].GetType()=="any")
            {  counter++;
            }
         }
      }
   }
   return counter;
}
```

GO ON TO THE NEXT PAGE.

Complete function FindBlock below.

```
int Flight::FindBlock(int row, int seatsNeeded) const
// postcondition: returns column index of the first (lowest index)
//                seat in a block of seatsNeeded adjacent
//                empty seats in the specified row;
//                if no such block exists, returns -1
{  int col, collen = mySeats.NumCols(), counter = 0;
   for(col=0; col<collen; col++)
   {  if(mySeats[row][col].GetPassenger().GetName() == "")
         counter++;
      else
         counter = 0;
      if(counter == seatsNeeded)
         return col - seatsNeeded + 1;
   }
   return -1;
}
```

GO ON TO THE NEXT PAGE.

Complete function AssignGroup below.

```
bool Flight::AssignGroup(const apvector<Passenger> & group)
// postcondition: if possible, assigns the group.length() passengers
//                from group to adjacent empty seats in a single row
//                and returns true;
//                otherwise, makes no changes and returns false
{  int i, row, rowlen = mySeats.numrows(), grouplen = group.length(), temp;
   for (row = 0; row < rowlen; row++)
   {  temp = FindBlock(row, grouplen);
      if (temp != -1)
      {  for ( i = 0; i < grouplen; i++)
            mySeats[row][temp+i].SetPassenger(group[i]);
         return true;
      }
   }
   return false;
}
```

GO ON TO THE NEXT PAGE.

A4/AB1 B

Complete function EmptySeatCount below.

```
int Flight::EmptySeatCount(const apstring & seatType) const
// postcondition: returns the number of empty seats
//                whose type is seatType;
//                if seatType is "any", returns the
//                total number of empty seats
{
    int i,j, num=0;

    for(i=0; i< mySeats.numrows(); i++)
        for(j=0; j<mySeats.numcols(); j++)
            if (mySeats[i][j].GetType == seatType || seatType=="any")
                if (mySeats[i][j].GetPassenger().GetName() == "")
                    num++;

    return num;
}
```

GO ON TO THE NEXT PAGE.

Complete function FindBlock below.

```
int Flight::FindBlock(int row, int seatsNeeded) const
// postcondition: returns column index of the first (lowest index)
//                seat in a block of seatsNeeded adjacent
//                empty seats in the specified row;
//                if no such block exists, returns -1
{
    int i, num = 0;

    for( i = 0; i < mySeats.numCols(); i++)

        if( mySeats[row][i].GetPassenger().GetName() == " ")
        {
            num ++;
            if( num == seatsNeeded)
                return i - seatsNeeded +1;
        }
        else
            num = 0;

    return -1;
}
```

GO ON TO THE NEXT PAGE.

Complete function AssignGroup below.

```
bool Flight::AssignGroup(const apvector<Passenger> & group)
// postcondition: if possible, assigns the group.length() passengers
//                from group to adjacent empty seats in a single row
//                and returns true;
//                otherwise, makes no changes and returns false
{
    int i, pos, x = group.length(), q;

    for (i = 0; i < mySeats.numRows(); i++)
    {
        pos = FindBlock (i, x);
        if (pos != -1)
            break;
    }

    if (pos == -1)
      return false;
    else
      for (q = 0; q < x; q++)
        mySeats [i] [pos + q].SetPassenger (group [q]);
    return true;
  }
}
```

GO ON TO THE NEXT PAGE.

A4/AB1 C

Complete function EmptySeatCount below.

```
int Flight::EmptySeatCount(const apstring & seatType) const
// postcondition: returns the number of empty seats
//                whose type is seatType;
//                if seatType is "any", returns the
//                total number of empty seats
```

```
{   int empty=0;
    for (int j=0; j<myScats.numrows(); j++)
    {
        for (int k=0; k<myScats.numcols(); k++)
        { if (myScats[j][k].GetType()==seatType || seatType=="any")
            if (myScats[j][k].GetName()=="")
                empty=empty+1;
        }
    }
    return empty;
}
```

Complete function `FindBlock` below.

```
int Flight::FindBlock(int row, int seatsNeeded) const
// postcondition: returns column index of the first (lowest index)
//                seat in a block of seatsNeeded adjacent
//                empty seats in the specified row;
//                if no such block exists, returns -1
{
    for (int k=0; k < mySeats.numcols(); k++)
    {
        if (mySeats[row][k].GetName() == "")
        {
            for (int j=0; j < seatsNeeded; j++)
            {
                if (mySeats[row][k+j].GetName() == "") &&
                    (j == seatsNeeded)
                    return k;
                else
                    j = seatsNeeded
            }
        }
    }
    return -1;
}
```

GO ON TO THE NEXT PAGE.

Complete function AssignGroup below.

```
bool Flight::AssignGroup(const apvector<Passenger> & group)
// postcondition: if possible, assigns the group.length() passengers
//                from group to adjacent empty seats in a single row
//                and returns true;
//                otherwise, makes no changes and returns false

{
    int seatsNeeded, BlockStart, i)

    seatsNeeded = group.length();
    i = 0;
    for (int k = 0; k < mySeats.numRows(); k++)
    {
        BlockStart = FindBlock(k, seatsNeeded);
        if (BlockStart > 0)
        {
            for (int j = BlockStart; j < seatsNeeded; j++)
            {
                mySeats[k][j].SetPassenger(group[i]);
                i += 1;
            }
        return true;
    }
}

return false;

}
```

GO ON TO THE NEXT PAGE.