



AP[®] Computer Science

Marine Biology Case Study

Appendices



Advanced Placement Program[®]

These appendices 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 these materials, electronically or otherwise*. These appendices 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.

The program code for these appendices is protected as provided by the GNU public license. A more complete statement is available in the Computer Science section of the AP website.

This booklet was 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 3,800 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 5,000 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 © 2000 by College Entrance Examination Board and Educational Testing Service. All rights reserved.
College Board, Advanced Placement Program, AP, College Board Online, College Explorer, and the acorn logo are registered trademarks of the College Entrance Examination Board.
APCD and EssayPrep are trademarks of the College Entrance Examination Board.

**Advanced Placement Program®
Computer Science**

Marine Biology Case Study

Appendices

*The AP® Program wishes to acknowledge and to thank
Mike Clancy of the University of California, Berkeley, who developed
the Marine Biology case study and the accompanying documentation
with assistance from Owen Astrachan of Duke University and
Cary Matsuoka of Lynbrook High School in San Jose, California.*

Appendices Index

Part 1: One dimensional tank

Appendix A — Program to simulate fish moving left or right in a rectangular tank

aquafish.h	A3
aquafish.cpp	A3
aquamain.cpp	A4

Part 2: Small two-dimensional area of ocean

Appendix B — .h files

display.h	B1
environ.h	B1
fish.h	B4
nbrhood.h	B5
position.h	B6
randgen.h	B8
simulate.h	B9
utils.h	B10

Appendix C — .cpp files

display.cpp	C1
environ.cpp	C2
fish.cpp	C5
nbrhood.cpp	C7
position.cpp	C8
simulate.cpp	C10
utils.cpp	C11
fishsim.cpp	C13

Appendix D

Quick Reference for apstring	D1
Quick Reference for apvector and apmatrix	D2
Quick Reference for apstack and apqueue (AB exam only)	D3

Appendix A

Program to simulate fish moving left or right in a rectangular tank

aquafish.h

```
#ifndef _AQUAFISH_H
#define _AQUAFISH_H

class AquaFish
{
public:
    AquaFish(int tankSize);

    void Swim(); // Swim one foot.

    int BumpCount() const; // Return the bump count.
private:
    int myPosition;
    int myBumpCount;
    int myTankSize;
    bool myDebugging;
};

#endif
```

aquafish.cpp

```
#include <iostream.h>
#include "aquafish.h"
#include "randgen.h"

AquaFish::AquaFish(int tankSize)
    : myPosition(tankSize/2),
      myTankSize(tankSize),
      myBumpCount(0),
      myDebugging(true)
{
}

void AquaFish::Swim()
{
    RandGen randomVals;
    int flip;

    if (myPosition == myTankSize - 1)
    {
        myPosition--;
    }
    else if (myPosition == 0)
    {
        myPosition++;
    }
    else
    {
```

```

        flip = randomVals.RandInt(2);
        if (flip == 0)
        {
            myPosition++;
        }
        else
        {
            myPosition--;
        }
    }

    if (myDebugging)
    {
        cout << "*** Position = " << myPosition << endl;
    }

    if (myPosition == 0 || myPosition == myTankSize - 1)
    {
        myBumpCount++;
    }
}

int AquaFish::BumpCount() const
{
    return myBumpCount;
}

```

aquamain.cpp

```

#include <iostream.h>
#include "aquafish.h"

int main()
{
    int tankSize;
    int stepsPerSim;
    int step;

    cout << "Tank size? ";
    cin >> tankSize;
    cout << "Steps per simulation? ";
    cin >> stepsPerSim;

    AquaFish fish(tankSize);

    for (step = 0; step < stepsPerSim; step++)
    {
        fish.Swim();
    }

    cout << "Bump count = " << fish.BumpCount() << endl;

    return 0;
}

```

Appendix B — .h files

display.h

```
#ifndef _DISPLAY_H
#define _DISPLAY_H

/**
 * Display displays all entries in an environment
 * in a text format, e.g., using fish-supplied characters
 * for fish and space ' ' for empty-space
 *
 * Show(..) is used to display the current state of
 * an environment. In this text-based display, all information
 * for Show(..) is accessible via the Environment passed into Show(..)
 *
 */

// class declarations for those classes only used
// as references (passed by reference/const reference)

class Environment;

class Display
{
public:

    // constructor
    Display();
    // postcondition: ready to display an Environment

    // modifying function
    void Show(const Environment & env); // display state of env
    // postcondition: state of env written as text to cout

private:

    // nothing here now

};

#endif
```

environ.h

```
#ifndef _ENVIRONMENT_H
#define _ENVIRONMENT_H

/**
 * The Environment class models a grid of fish.
 * An environment is populated at construction time
 * with fish from a stream, presumably bound to a file.
 * The stream is expected to provide data in a certain format
 * as described below.
 *
 * The size of the environment can be determined using the
 * following accessor functions:
 *
 */
```

```

*   - NumRows()
*           the # of rows in the grid
*   - NumCols()
*           the # of columns in the grid
*
* The Fish in the model/environment are accessible via a function
* that returns a vector of all the fish. The Environment class
* guarantees that the fish are stored in the vector in top-down,
* left-to-right order. In the grid diagrammed below, fish are
* numbered in the order they will be stored in the vector (starting
* with the fish at index 0).
*
*   + - - - - - - - - - - +
*   | 0           1 |
*   | 2           3 |
*   |           4 |
*   | 5   6 |
*   |7           8 |
*   + - - - - - - - - - - +
*
* Sample client code for iterating over fish and printing
* all fish id's and coordinates in top-down/left-right order
* appears below.
*
* // construct environment
*
* ifstream input("fish.dat");
* Environment env(input);
* cout << "grid has dimensions " << env.NumRows() << " x "
*      << env.NumCols() << endl;
*
* // print fish
*
* apvector<Fish> fishList = env.AllFish();
* int k;
* for (k = 0; k < fishList.length(); k++)
*     cout << fishList[k] << endl;
*
* The format of the data in the stream (probably bound to a text file)
* is as follows:
*   The first line has number of rows, number of columns.
*   Each subsequent line stores row/col positions of a fish.
*   All entries are separated by white space.
* For example,
*
*           rows columns
*           row-pos col-pos
*           row-pos col-pos
*           ..
*           row-pos col-pos
*
*/

```

```

#include "fish.h"
#include "apmatrix.h"
#include <fstream.h>

// class declarations for those classes only used
// as references (passed by reference/const reference)

class Position;

class Environment
{
public:
    // constructor

    Environment(istream & input);
        // precondition: input is open for reading, in correct format
        // postcondition: environment initialized and populated from input

    // accessing functions

    int NumRows() const;
        // postcondition: returns # rows in grid

    int NumCols() const;
        // postcondition: returns # columns in grid

    apvector<Fish> AllFish() const;
        // postcondition: returned vector (call it fishList) contains all fish
        // in top-down, left-right order:
        // top-left fish in fishList[0],
        // bottom-right fish in fishList[fishList.length()-1];
        // # fish in environment is fishList.length()

    bool IsEmpty(const Position & pos) const;
        // postcondition: returns true if pos in grid and no fish at pos,
        // returns false otherwise

    // modifying functions

    void Update(const Position & oldLoc, Fish & fish);
        // precondition: fish was located at oldLoc, has been updated
        // postcondition: if (fish.Location() != oldLoc) then oldLoc is empty;
        // Fish fish is updated properly in this environment

    void AddFish(const Position & pos);
        // precondition: no fish already at pos, i.e., IsEmpty(pos)
        // postcondition: fish created at pos

private:

    bool InRange(const Position & pos) const;
        // postcondition: returns true if pos in grid,
        // returns false otherwise

    apmatrix<Fish> myWorld;    // grid of fish

    int myFishCreated;        // # fish ever created
    int myFishCount;          // # fish in current environment
};
#endif

```

fish.h

```
#ifndef _FISH_H
#define _FISH_H

/**
 * The Fish class represents a fish/swimmer. A fish has an integer id
 * and maintains its position in a grid. Both these pieces of information
 * are set by the constructor, and the id is never changed.
 * A Fish moves via the member function Move.
 *
 * A fish whose "amIDefined" field is false represents an "empty"
 * or "undefined" fish. The IsUndefined() member function distinguishes
 * defined fish from undefined fish. The default Fish() constructor
 * creates an empty/undefined fish.
 */

#include <iostream.h>
#include "apstring.h"
#include "position.h"
#include "nbrhood.h"

// class declarations for those classes only used
// as references (passed by reference/const reference)

class Environment;

class Fish
{
public:

    // constructors

    Fish();
    // postcondition: IsUndefined() == true

    Fish(int id, const Position & pos);
    // postcondition: Location() returns pos, Id() returns id,
    //                IsUndefined() == false

    // accessing functions

    int Id() const;
    // precondition: ! IsUndefined()
    // postcondition: returns id number of fish

    Position Location() const;
    // postcondition: returns current fish position

    bool IsUndefined() const;
    // postcondition: returns true if constructed via default
    //                constructor, false otherwise

    apstring ToString() const;
    // postcondition: returns a stringized form of Fish

    char ShowMe() const;
    // postcondition: returns a character that can make me visible

```

```

// modifying functions

void Move(Environment & env);
    // precondition: Fish stored in env at Location()
    // postcondition: Fish has moved to a new location in env (if possible)

private:

    // helper functions

    Neighborhood EmptyNeighbors(const Environment & env,
                                const Position & pos) const;

    void AddIfEmpty(const Environment & env,
                    Neighborhood & nbrs, const Position & pos) const;

    int myId;
    Position myPos;
    bool amIDefined;
};

ostream & operator << (ostream & out, const Fish & fish);
    // postcondition: fish inserted onto stream out

#endif

```

nbrhood.h

```

#ifndef _NBRHOOD_H
#define _NBRHOOD_H

/**
 * class Neighborhood represents a
 * collection of Positions.
 *
 * Positions can be added to a Neighborhood.
 * Each Position in a Neighborhood is accessible
 * via the functions Select() -- choose a Position --
 * and Size() -- return the # of Positions in a neighborhood.
 *
 * In the current implementation, a maximum of 4 Positions can be
 * added to a neighborhood. Any call of Add() after the fourth
 * call is ignored.
 */

#include <iostream.h>
#include "position.h"
#include "apvector.h"
#include "apstring.h"

class Neighborhood
{
public:

    // constructor

```

```

Neighborhood();
    // postcondition: Size() == 0

    // accessing functions

int Size() const;    // # Positions
    // postcondition: returns # Positions in the neighborhood

Position Select(int index) const;    // access a Position
    // precondition: 0 <= index < Size()
    // postcondition: returns the index-th Position in Neighborhood

apstring ToString() const;    // stringized representation
    // postcondition: returns a string version of all Positions
    //                    in Neighborhood

    // modifying functions

void Add(const Position & pos);    // add pos to neighborhood
    // precondition: there is room in the neighborhood
    // postcondition: pos added to Neighborhood

private:

    apvector<Position> myList;
    int myCount;
};

ostream & operator << (ostream & out, const Neighborhood & nbrhood);
    // postcondition: nbrhood inserted onto stream out

#endif

```

position.h

```

#ifndef _POSITION_H
#define _POSITION_H

/**
 * A Position represents a (row,column) in a grid
 * whose (0,0) is upper-left as in matrix coordinates.
 *
 * Once constructed, a position doesn't change
 * (all member functions are const) although
 * a Position can be assigned to an existing Position.
 * For example,
 *
 * Position p(2,3);
 * Position q;    // default (-1,-1)
 * q = p;    // q now at (2,3)
 *
 */

```

```

* Adjacent Positions of a given Position can be
* determined as illustrated:
*
*   Position p(5,5);
*   Position q;
*
*   q = p.North();    // q is (4,5)
*   q = p.South();   // q is (6,5)
*   q = p.East();    // q is (5,6)
*   q = p.West();    // q is (5,4)
*
*/

#include <iostream.h>
#include "apstring.h"

class Position
{
public:

    // constructors

    Position();
    // postcondition: Row() == -1, Col() == -1
    Position(int r, int c);
    // postcondition: Row() == r, Col() == c

    // accessing functions

    int Row() const;
    // postcondition: returns row of Position
    int Col() const;
    // postcondition: returns column of Position

    Position North() const;
    // postcondition: returns Position north of (up from) this position
    Position South() const;
    // postcondition: returns Position south of (down from) this position
    Position East() const;
    // postcondition: returns Position east (right) of this position
    Position West() const;
    // postcondition: returns Position west (left) of this position

    bool Equals(const Position & rhs) const;
    // postcondition: returns true iff this position equals rhs

    apstring ToString() const;
    // postcondition: returns stringized form of Position

private:
    int myRow;
    int myCol;
};

ostream & operator << (ostream & out, const Position & pos);
// postcondition: pos inserted onto stream out

bool operator == (const Position & lhs, const Position & rhs);
// postcondition: returns true iff lhs == rhs

#endif

```

randgen.h

```
#ifndef _RANDGEN_H
#define _RANDGEN_H

// RandGen objects provide a source of computer-generated random numbers
// (sometimes known as pseudo-random numbers).
//
// By default, a RandGen object will produce a different series of
// numbers (through repeated calls to the RandInt and RandReal
// methods) every time the program is run.  When testing, though, it is
// often useful to have a program generate the same sequence of numbers
// each time it is run; this can be achieved by specifying a "seed" when
// the first RandGen object in the program is created.
//
// To construct random integers in a given range, client programs
// should use RandInt.  To construct random doubles, client programs
// should use RandReal.  The ranges for the return values for these
// functions are indicated with mathematical notation:
// [0..max) means a number between 0 and max,
// including 0 but not including max;
// [0..max] means a number between 0 and max, including
// both 0 and max.
//
// For example,
// RandGen r;
// r.RandInt(5) an int between 0 and 5, including 0 but not 5,
// i.e., in [0, 5)
// r.RandInt(2, 5) an int between 2 and 5, including 2 and 5,
// i.e., in [2, 5]
// r.RandReal() a double between 0.0 and 1.0, including 0.0
// but not 1.0, i.e., in [0.0, 1.0)
// r.RandReal(4.2, 6.7) a double between 4.2 and 6.7, including 4.2
// but not 6.7, i.e., in [4.2, 6.7)
//
// Technical Note:
// The "seed" used by all random number generation in a program is set
// the first time a random number generator object is constructed by
// the program.  All other random number generator objects created
// later in the program will use the same seed.

class RandGen
{
public:

// Constructors
// If the first RandGen object is constructed with the default
// constructor, a different series of numbers is produced every
// time the program is run.  If the first RandGen object is
// constructed with a seed, the same series of numbers is produced
// every time.

RandGen(); // default constructor

RandGen(int seed); // produce same series every time
// (most useful during testing)
```

```

// Accessing functions

int RandInt(int max); // returns int in [0..max)

int RandInt(int low, int max); // returns int in [low..max]

double RandReal(); // returns double in [0..1)

double RandReal(double low, double max); // returns double in
// [low..max)

private:
    static int ourInitialized; // for 'per-class' initialization
};

#endif // _RANDGEN_H not defined

```

simulate.h

```

#ifndef _SIMULATION_H
#define _SIMULATION_H

/**
 * Simulation controls a simulation of fish as
 * represented in an Environment.
 *
 * One step of a simulation can be performed via Step(..),
 * an arbitrary number of steps via Run(..).
 */

// class declarations for those classes only used
// as references (passed by reference/const reference)

class Environment;

class Simulation
{
public:
    Simulation();
    // postcondition: simulation is ready to run

    void Step(Environment & env);
    // postcondition: one step of simulation in env has been made

    void Run(Environment & env, int steps);
    // postcondition: simulation on env run for # steps passed as steps
};

#endif

```

utils.h

```
#ifndef _UTILS_H
#define _UTILS_H

#include "apstring.h"
#include "apvector.h"
#include "fish.h"

// Collection of useful utility functions

apstring IntToString(int n);
    // postcondition: returns stringized form of n

void Sort(apvector<Fish> & list, int numElts);
    // precondition: list contains numElts Fish
    // postcondition: list sorted so that entries are
    //                in order top-down/left-right by Position

void DebugPrint(int level, const apstring & msg);
    // print the given debugging error message if level is low enough
    //    (which levels are actually printed can be set in utils.cpp)

#endif
```

Appendix C — .cpp files

display.cpp

```
#include <iomanip.h>
#include "display.h"
#include "environ.h"
#include "fish.h"
#include "utils.h"

// constructor

Display::Display()
// postcondition: ready to display an Environment
{
}

// public modifying function

void Display::Show(const Environment & env)
// postcondition: state of env written as text to cout
{
    const int WIDTH = 1; // for each fish

    int rows = env.NumRows();
    int cols = env.NumCols();
    int fishIndex = 0;
    int numFish;
    int r, c;
    Position pos; // position of fish to be displayed next

    apvector<Fish> fishList;
    fishList = env.AllFish();
    numFish = fishList.length();

    // find position of first fish to be displayed (if any)
    if (fishIndex < numFish)
    {
        pos = fishList[fishIndex].Location();
    }

    for (r = 0; r < rows; r++)
    {
        for (c = 0; c < cols; c++)
        {

            if (pos.Row() == r && pos.Col() == c)
            {
                // this is a position with a fish
                cout << setw(WIDTH) << fishList[fishIndex].ShowMe();
                fishIndex++;
            }
        }
    }
}
```

```

        // find position of next fish to be displayed
        if (fishIndex < numFish)
        {
            pos = fishList[fishIndex].Location();
        }
        else // no more fish to display
        {
            pos = Position(); // not in the grid, won't be displayed
        }
    }
    else // this position has no fish
    {
        cout << setw(WIDTH) << ' ';
    }
} // finished processing all columns in a row
cout << endl;
} // finished processing all rows in the grid
}

```

environ.cpp

```

#include "environ.h"
#include "fish.h"
#include "utils.h"

// constructor

Environment::Environment(istream & input)
: myWorld(0,0),
  myFishCreated(0),
  myFishCount(0)
// precondition: input is open for reading, in correct format
// postcondition: environment initialized and populated from input
{
    int numRows, numCols, row, col;

    if (input >> numRows >> numCols) // resize the matrix
    {
        myWorld.resize(numRows, numCols);
    }
    else
    {
        cerr << "reading rows/columns failed in Environment" << endl;
        exit(1);
    }
    while (input >> row >> col)
    {
        AddFish(Position(row, col));
    }
}

// public accessing functions

int Environment::NumRows() const
// postcondition: returns # rows in grid
{
    return myWorld.numrows();
}

```

```

int Environment::NumCols() const
// postcondition: returns # columns in grid
{
    return myWorld.numcols();
}

apvector<Fish> Environment::AllFish() const
// postcondition: returned vector (call it fishList) contains all fish
//                 in top-down, left-right order:
//                 top-left fish in fishList[0],
//                 bottom-right fish in fishList[fishList.length()-1];
//                 # fish in environment is fishList.length()
{
    apvector<Fish> fishList(myFishCount);
    int r, c, k;
    int count = 0;
    apstring s = "";

    // look at all grid positions, store fish found in vector fishList

    for (r = 0; r < NumRows(); r++)
    {
        for (c = 0; c < NumCols(); c++)
        {
            if (! myWorld[r][c].IsUndefined())
            {
                fishList[count] = myWorld[r][c];
                count++;
            }
        }
    }

    for (k = 0; k < count; k++)
    {
        s += fishList[k].Location().ToString() + " ";
    }
    DebugPrint(5, "Fish vector = " + s);
    return fishList;
}

bool Environment::IsEmpty(const Position & pos) const
// postcondition: returns true if pos in grid and no fish at pos,
//                 returns false otherwise
{
    if (! InRange(pos))
    {
        return false; // debug msg printed in InRange
    }

    if (myWorld[pos.Row()][pos.Col()].IsUndefined())
    {
        return true; // pos in grid and no fish at pos
    }

    DebugPrint(5, pos.ToString() + " contains a fish.");
    return false;
}

```

```

// public modifying functions

void Environment::Update(const Position & oldLoc, Fish & fish)
// precondition: fish was located at oldLoc, has been updated
// postcondition: if (fish.Location() != oldLoc) then oldLoc is empty;
//               Fish fish is updated properly in this environment
{
    Fish emptyFish;

    if (InRange(oldLoc))
    {
        if (myWorld[oldLoc.Row()][oldLoc.Col()].Id() != fish.Id())
        {
            cerr << "illegal fish move" << endl;
        }
        else
        {
            // Put an updated copy of fish in fish's current position.
            Position newLoc = fish.Location();
            myWorld[newLoc.Row()][newLoc.Col()] = fish;

            // If fish moved, empty out fish's old location.
            if (! (oldLoc == newLoc))
            {
                myWorld[oldLoc.Row()][oldLoc.Col()] = emptyFish;
            }
        }
    }
}

void Environment::AddFish(const Position & pos)
// precondition: no fish already at pos, i.e., IsEmpty(pos)
// postcondition: fish created at pos
{
    if (! IsEmpty(pos))
    {
        cerr << "error, attempt to create fish at non-empty: " << pos << endl;
        return;
    }
    myFishCreated++;
    myWorld[pos.Row()][pos.Col()] = Fish(myFishCreated, pos);
    myFishCount++;
}

// private helper functions

bool Environment::InRange(const Position & pos) const
// postcondition: returns true if pos is in the grid,
//               returns false otherwise
{
    if (0 <= pos.Row() && pos.Row() < NumRows() &&
        0 <= pos.Col() && pos.Col() < NumCols())
    {
        return true;
    }

    DebugPrint(5, pos.ToString() + " is out of range.");
    return false;
}

```

fish.cpp

```
#include "fish.h"
#include "environ.h"
#include "randgen.h"
#include "position.h"
#include "nbrhood.h"
#include "utils.h"

// constructors

Fish::Fish()
    : myId(0),
      amIDefined(false)
// postcondition: IsUndefined() == true
{
}

Fish::Fish(int id, const Position & pos)
    : myId(id),
      myPos(pos),
      amIDefined(true)
// postcondition: Location() returns pos, Id() returns id,
//                IsUndefined() == false
{
}

// public accessing functions

int Fish::Id() const
// precondition: ! IsUndefined()
// postcondition: returns id number of fish
{
    return myId;
}

Position Fish::Location() const
// postcondition: returns current fish position
{
    return myPos;
}

bool Fish::IsUndefined() const
// postcondition: returns true if constructed via default constructor,
//                false otherwise
{
    return ! amIDefined;
}
```

```

apstring Fish::ToString() const
// postcondition: returns a stringized form of Fish
{
    return IntToString(myId) + " " + myPos.ToString();
}

// public modifying functions

void Fish::Move(Environment & env)
// precondition: Fish stored in env at Location()
// postcondition: Fish has moved to a new location in env (if possible)
{
    RandGen randomVals;
    Neighborhood nbrs = EmptyNeighbors(env, myPos);
    DebugPrint(3, nbrs.ToString());

    if (nbrs.Size() > 0)
    {
        // there were some empty neighbors, so randomly choose one
        Position oldPos = myPos;
        myPos = nbrs.Select(randomVals.RandInt(0, nbrs.Size() - 1));
        DebugPrint(1, "Fish " + oldPos.ToString() + " moves to "
            + myPos.ToString());

        env.Update(oldPos, *this);          // *this means this fish
    }
    else
    {
        DebugPrint(1, "Fish " + ToString() + " can't move.");
    }
}

char Fish::ShowMe() const
// postcondition: returns a character that can make me visible
{
    if (1 <= Id() && Id() <= 26)
    {
        return 'A' + (Id() - 1);
    }
    return '*';
}

// private helper functions

Neighborhood Fish::EmptyNeighbors(const Environment & env,
                                  const Position & pos) const
// precondition: pos is in the grid being modelled
// postcondition: returns a Neighborhood of pos of empty positions
{
    Neighborhood nbrs;

    AddIfEmpty(env, nbrs, pos.North());
    AddIfEmpty(env, nbrs, pos.South());
    AddIfEmpty(env, nbrs, pos.East());
    AddIfEmpty(env, nbrs, pos.West());
}

```

```

    return nbrs;
}

void Fish::AddIfEmpty(const Environment & env,
                    Neighborhood & nbrs, const Position & pos) const
// postcondition: pos is added to nbrs if pos in env and empty
{
    if (env.IsEmpty(pos))
    {
        nbrs.Add(pos);
    }
}

// free functions

ostream & operator << (ostream & out, const Fish & fish)
// postcondition: fish inserted onto stream out
{
    out << fish.ToString();
    return out;
}

```

nbrhood.cpp

```

#include "nbrhood.h"
#include "utils.h"

// constructor

Neighborhood::Neighborhood()
    : myList(4),
      myCount(0)
// postcondition: Size() == 0
{
}

// public accessing functions

int Neighborhood::Size() const
// postcondition: returns # Positions in the neighborhood
{
    return myCount;
}

Position Neighborhood::Select(int index) const
// precondition: 0 <= index < Size()
// postcondition: returns the index-th Position in Neighborhood
{
    DebugPrint(5, "Selecting neighborhood element # " + IntToString(index));
    return myList[index];
}

```

```

apstring Neighborhood::ToString() const
// precondition: returns a string version of all Positions in Neighborhood
{
    apstring s = "Neighborhood: ";
    int k;
    for (k = 0; k < myCount; k++)
    {
        s += myList[k].ToString() + " ";
    }
    return s;
}

// public modifying function

void Neighborhood::Add(const Position & pos)
// precondition: there is room in the neighborhood
// postcondition: pos added to Neighborhood
{
    if (myCount < myList.length())
    {
        DebugPrint(5, "Adding " + pos.ToString() + " to neighborhood.");
        myList[myCount] = pos;
        myCount++;
    }
    else
    {
        DebugPrint(5, "Neighborhood had no room for " + pos.ToString());
    }
}

// free function

ostream & operator << (ostream & out, const Neighborhood & nbrhood)
// postcondition: nbrhood inserted onto stream out
{
    out << nbrhood.ToString();
    return out;
}

```

position.cpp

```

#include "position.h"
#include "utils.h"

// constructors

Position::Position()
: myRow(-1),
  myCol(-1)
// precondition: Row() == -1, Col() == -1
{
}

Position::Position(int r, int c)
: myRow(r),
  myCol(c)
// precondition: Row() == r, Col() == c
{
}

```

```

// public accessing functions

int Position::Row() const
// postcondition: returns row of Position
{
    return myRow;
}

int Position::Col() const
// postcondition: returns column of Position
{
    return myCol;
}

Position Position::North() const
// postcondition: returns Position north of (up from) this position
{
    return Position(myRow - 1, myCol);
}

Position Position::South() const
// postcondition: returns Position south of (down from) this position
{
    return Position(myRow + 1, myCol);
}

Position Position::East() const
// postcondition: returns Position east (right) of this position
{
    return Position(myRow, myCol + 1);
}

Position Position::West() const
// postcondition: returns Position west (left) of this position
{
    return Position(myRow, myCol - 1);
}

bool Position::Equals(const Position & rhs) const
// postcondition: returns true iff this position equals rhs
{
    return myRow == rhs.myRow && myCol == rhs.myCol;
}

apstring Position::ToString() const
// postcondition: returns stringized form of Position
{
    apstring s = "(" + IntToString(myRow) + ", "
                + IntToString(myCol) + ")";
    return s;
}

```

```
// free functions
```

```
ostream & operator << (ostream & out, const Position & pos)
// postcondition: pos inserted onto stream out
{
    out << pos.ToString();
    return out;
}
```

```
bool operator == (const Position & lhs, const Position & rhs)
// postcondition: returns true iff lhs == rhs
{
    return lhs.Equals(rhs);
}
```

simulate.cpp

```
#include "simulate.h"
#include "apvector.h"
#include "environ.h"
```

```
//constructor
```

```
Simulation::Simulation()
// postcondition: simulation is ready to run
{
}
}
```

```
// public modifying functions
```

```
void Simulation::Step(Environment & env)
// postcondition: one step of simulation in env has been made
{
    apvector<Fish> fishList;
    int k;

    fishList = env.AllFish();
    for (k = 0; k < fishList.length(); k++)
    {
        fishList[k].Move(env);
    }
}
```

```
void Simulation::Run(Environment & env, int steps)
// postcondition: simulation on env run for # steps passed as steps
{
    int k;

    for (k = 0; k < steps; k++)
    {
        Step(env);
    }
}
```

utils.cpp

```
#include "utils.h"
#include "position.h"

apstring IntToString(int n)
// postcondition: returns stringized form of n
{
    if (n == 0)
    {
        return "0";    // special case for 0
    }

    int k;
    apstring reverse = "";    // will be correct, but in reverse
    apstring val = "";    // the string returned

    if (n < 0)                // start with "-" if n < 0
    {
        val = "-";
        n = -n;
    }
    while (n > 0)            // get each digit, catenate in reverse
    {
        reverse += char('0' + n % 10);
        n /= 10;
    }

    // now build the string to return by "unreversing"

    for (k = reverse.length() - 1; k >= 0; k--)
    {
        val += reverse[k];
    }

    return val;
}

void Sort(apvector<Fish> & list, int numElts)
// precondition: list contains numElts Fish
// postcondition: list sorted so that entries are
//                in order top-down/left-right by Position
{
    // use selection sort

    int j, k, minIndex;
    Position min;
    Position current;
    Fish temp;
```

```

for (j = 0; j < numElts; j++)
{
    minIndex = j;
    min = list[j].Location();
    for (k = j + 1; k < numElts; k++)
    {
        current = list[k].Location();
        if (current.Row() < min.Row() ||
            (min.Row() == current.Row() && current.Col() < min.Col()))
        {
            min = current;
            minIndex = k;
        }
    }
    temp = list[minIndex];
    list[minIndex] = list[j];
    list[j] = temp;
}
}

// Indicates level of detail at which we want debugging information.
// 0 => no debugging information displayed
// 1 => fish moves only
// 3 => neighborhood contents + output for 1
// 5 => neighborhood element selection + positions added and not added
//      to the neighborhood + myFish vector + output for 3

const int LEVEL_OF_DEBUG_DETAIL = 0;

// The given msg is to be printed if level (which is positive) is less than
// or equal to LEVEL_OF_DEBUG_DETAIL, the level of detail at which we want to
// see debugging info. Indent the printed msg 2 spaces for each level of
// detail.
void DebugPrint(int level, const apstring & msg)
{
    int k;

    if (level <= LEVEL_OF_DEBUG_DETAIL)
    {
        for (k = 0; k < level; k++)
        {
            cout << ' ';
        }
        cout << "***** " << msg << endl;
    }
}

```

Driver program (fishsim.cpp)

```
#include <iostream.h>
#include <fstream.h>
#include "apstring.h"
#include "environ.h"
#include "display.h"
#include "simulate.h"

int main()
{
    // replace filename below with appropriate file (full path if necessary)
    ifstream input("fish.dat");
    Environment env(input);

    // Display display(100,100); // for graphics display
    Display display; // for text display
    apstring s;

    Simulation sim;

    int step;
    int numSteps;

    display.Show(env);

    cout << "--- initialized --- " << endl;

    cout << "How many steps? ";
    cin >> numSteps;
    getline(cin, s);

    for (step = 0; step < numSteps; step++)
    {
        sim.Step(env);
        display.Show(env);
        cout << " step " << step << " (press return)";
        getline(cin, s);
    }

    return 0;
}
```

Appendix D

Quick Reference for apstring

```
extern const int npos; // used to indicate not a position in the string

// public member functions

// constructors/destructor
apstring(); // construct empty string ""
apstring(const char * s); // construct from string literal
apstring(const apstring & str); // copy constructor
~apstring(); // destructor

// assignment
const apstring & operator= (const apstring & str); // assign str
const apstring & operator= (const char * s); // assign s
const apstring & operator= (char ch); // assign ch

// accessors
int length() const; // number of chars
int find(const apstring & str) const; // index of first occurrence
// of str
int find(char ch) const; // index of first occurrence
// of ch
apstring substr(int pos, int len) const; // substring of len chars,
// starting at pos
const char * c_str() const; // explicit conversion to char *

// indexing
char operator[ ](int k) const; // range-checked indexing
char & operator[ ](int k); // range-checked indexing

// modifiers
const apstring & operator+= (const apstring & str); // append str
const apstring & operator+= (char ch); // append char

// The following free (non-member) functions operate on strings

// I/O functions
ostream & operator<< ( ostream & os, const apstring & str );
istream & operator>> ( istream & is, apstring & str );
istream & getline( istream & is, apstring & str );

// comparison operators
bool operator== ( const apstring & lhs, const apstring & rhs );
bool operator!= ( const apstring & lhs, const apstring & rhs );
bool operator< ( const apstring & lhs, const apstring & rhs );
bool operator<= ( const apstring & lhs, const apstring & rhs );
bool operator> ( const apstring & lhs, const apstring & rhs );
bool operator>= ( const apstring & lhs, const apstring & rhs );

// concatenation operator +
apstring operator+ ( const apstring & lhs, const apstring & rhs );
apstring operator+ ( char ch, const apstring & str );
apstring operator+ ( const apstring & str, char ch );
```

Quick Reference for apvector and apmatrix

```
template <class itemType>
class apvector

    // public member functions

    // constructors/destructor
    apvector(); // default constructor (size==0)
    apvector(int size); // initial size of vector is size
    apvector(int size, const itemType & fillValue);
    // all entries == fillValue
    apvector(const apvector & vec); // copy constructor
    ~apvector(); // destructor

    // assignment
    const apvector & operator= (const apvector & vec);

    // accessors
    int length() const; // capacity of vector

    // indexing (with range checking)
    itemType & operator[ ](int index);
    const itemType & operator[ ](int index) const;

    // modifiers
    void resize(int newSize); // change size dynamically; can result
    // in losing values
```

```
template <class itemType>
class apmatrix

    // public member functions

    // constructors/destructor
    apmatrix(); // default size is 0 x 0
    apmatrix(int rows, int cols); // size is rows x cols
    apmatrix(int rows, int cols, const itemType & fillValue);
    // all entries == fillValue
    apmatrix(const apmatrix & mat); // copy constructor
    ~apmatrix( ); // destructor

    // assignment
    const apmatrix & operator = (const apmatrix & rhs);

    // accessors
    int numRows() const; // number of rows
    int numcols() const; // number of columns

    // indexing (with range checking)
    const apvector<itemType> & operator[ ](int k) const;
    apvector<itemType> & operator[ ](int k);

    // modifiers
    void resize(int newRows, int newCols);
    // resizes matrix to newRows x newCols
    // (can result in losing values)
```

Quick Reference for apstack and apqueue (AB exam only)

```
template <class itemType>
class apstack

    // public member functions

    // constructors/destructor
    apstack(); // construct empty stack
    apstack(const apstack & s); // copy constructor
    ~apstack(); // destructor

    // assignment
    const apstack & operator = (const apstack & rhs);

    // accessors
    const itemType & top() const; // return top element (NO pop)
    bool isEmpty() const; // return true if empty, else false
    int length() const; // return number of elements in stack

    // modifiers
    void push(const itemType & item); // push item onto top of stack
    void pop(); // pop top element
    void pop(itemType & item); // combines pop and top
    void makeEmpty(); // make stack empty (no elements)
```

```
template <class itemType>
class apqueue

    // public member functions

    // constructors/destructor
    apqueue(); // construct empty queue
    apqueue(const apqueue & q); // copy constructor
    ~apqueue(); // destructor

    // assignment
    const apqueue & operator= (const apqueue & rhs);

    // accessors
    const itemType & front() const; // return front (no dequeue)
    bool isEmpty() const; // return true if empty else false
    int length() const; // return number of elements in queue

    // modifiers
    void enqueue(const itemType & item); // insert item (at rear)
    void dequeue(); // remove first element
    void dequeue(itemType & item); // combine front and dequeue
    void makeEmpty(); // make queue empty
```

College Board Regional Offices

National Office

Lee Jones/Phil Arbolino/Robert DiYanni/Michael Johaneck/Frederick Wright/Trevor Packer
45 Columbus Avenue
New York, NY 10023-6992
E-mail: ap@collegeboard.org
(212) 713-8066

Middle States

Mary Alice McCullough/Michael Marsh
3440 Market Street, Suite 410
Philadelphia, PA 19104-3338
(215) 387-7600

Midwestern

Bob McDonough/Paula Herron/Ann Winship
1560 Sherman Avenue, Suite 1001
Evanston, IL 60201-4805
(847) 866-1700

New England

Fred Wetzel
470 Totten Pond Road
Waltham, MA 02451-1982
(781) 890-9150

Southern

Tom New
100 Crescent Centre Parkway, Suite 340
Tucker, GA 30084-7039
(770) 908-9737

Southwestern

Frances Brown/Mondy Raibon/Scott Kampmeier/Paul Sanders
4330 South MoPac Expressway, Suite 200
Austin, TX 78735-6734
(512) 891-8400

Dallas/Fort Worth Metroplex AP Office

Kay Wilson
P.O. Box 19666, 600 South West Street, Room 108
Arlington, TX 76019
(817) 272-7200

Western

Claire Pelton/Gail Chapman
2099 Gateway Place, Suite 480
San Jose, CA 95110-1048
(408) 452-1400

Canada (AP Program Only)

George Ewonus
Suite 212-1755 Springfield Road
Kelowna, B.C., Canada V1Y 5V5
(250) 861-9050
E-mail: gewonus@ap.ca

Staff at U.S. College Board Regional Offices other than the National Office can be reached via e-mail using their first initial and last name@collegeboard.org



1999-2000

AP Computer Science Development Committee



Susan Rodger, <i>Chair</i>	Duke University Durham, North Carolina
Alyce Brady	Kalamazoo College Kalamazoo, Michigan
Don Kirkwood	North Salem High School Salem, Oregon
Joseph W. Kmoch	Washington High School Milwaukee, Wisconsin
Kathleen A. Larson	Kingston High School Kingston, New York
Mark A. Weiss	Florida International University Miami, Florida
<i>Chief Faculty Consultant:</i> Chris Nevison	Colgate University Hamilton, New York
<i>ETS Consultants:</i> Frances E. Hunt, Esther Tesar	
<i>College Board Consultant:</i> Gail L. Chapman	