(This is version 4 of this document.)

This document is derived from material in "An experimental evaluation of the assumption of independence in multiversion programming," J. C. Knight and N. G. Leveson, IEEE Transactions on Software Engineering 12(1):96–109, January 1986.

LAUNCH INTERCEPTOR PROGRAM: REQUIREMENTS SPECIFICATION

1 INTRODUCTION

As part of a hypothetical anti-ballistic missile system, you will write a parameterless C function called `DECIDE()`. It will generate a signal which determines whether an interceptor should be launched based upon input radar tracking information. This radar tracking information is available at the instant the function is called. In the following sections, the names of input variables are in bold, as in **name**. Terms in italics such as *term* are defined in the glossary in Appendix A.

Values of quantities which are parameters of the problem are provided and will determine which combination of the several possible *Launch Interceptor Conditions* (LIC's) are relevant to the immediate situation. The interceptor launch button is normally considered locked; only if all relevant combinations of launch conditions are met will the launch-unlock signal be issued.

Your function will determine whether each of fifteen LIC's is true for an input set of up to 100 *planar data points* representing radar echoes. The LIC's are specified in the Functional Requirements section of this document. To indicate which LIC's are satisfied by the set of points, the fifteen elements of a *Conditions Met Vector* (CMV) will be assigned boolean values true or false; each element of the CMV corresponds to one LIC.

Another input, the *Logical Connector Matrix* (LCM), defines which individual LIC's must be considered jointly in some way. The LCM is a 15x15 symmetric *matrix* with elements valued ANDD, ORR, or NOTUSED. CMV elements are combined as indicated by the LCM, and the resulting boolean values are stored in the *off-diagonal elements* of the *Preliminary Unlocking Matrix* (PUM), a 15x15 symmetric *matrix*. Thus, the *off-diagonal elements* of the PUM are an output of your function. The PUM's *diagonal elements* are an input to your function and represent which LIC's actually matter in this particular launch determination. Each *diagonal element* of the PUM indicates how to combine the off-diagonal values in the same PUM row to form the corresponding element of the *Final Unlocking Vector* (FUV), a 15-element *vector*. If, and only if, all the values in the FUV are true, should the launch-unlock signal be generated.

No actual reading or writing to physical I/O devices will take place. Instead, inputs to your function will be available as global variables, and outputs from your function will be placed in other global variables.

2 HARDWARE AND SOFTWARE SUPPORT

2.1 Software

- Your software must compile without warnings using:

  ```
  gcc -c -O2 -ffloat-store -Wall -Wextra -Werror [C file(s)]
  ```

- Your C file(s) must include `decide.h`, may include `math.h`, and must not include any other header files. You may use utility functions from the C math library.

2.2 Hardware

After being compiled, your function must execute properly both on ARM and x86-64 processors.

3 FUNCTIONAL REQUIREMENTS

All communication with software which calls your function is to be accomplished through the global variables and constant defined in this section.

3.1 Constant

Available to your function is the value of the global constant, PI, representing the number of radians in 180 degrees.

3.2 Input Variables

The values of the following global variables are available to your function:

NUMPOINTS The number of *planar data points*.

X,Y Parallel arrays containing the coordinates of data points.

PARAMETERS Struct holding parameters for LIC's.

LCM Logical Connector Matrix.

PUM (diagonal elements) Preliminary Unlocking Matrix.

3.3 Output Variables

The values of the following global variables are to be set by your function:

PUM (off-diagonal elements) Preliminary Unlocking Matrix.

CMV Conditions Met Vector.

FUV Final Unlocking Vector.

LAUNCH Final launch/no launch decision.

3.4 Global Declarations

The global declarations in `decide.h` may be found at the end of this document in Appendix B.

3.5 Required Computations

It can be assumed that all input data and parameters that are measured in some form of units use the same, consistent units. For example, all lengths are measured in the same units that are used to define the planar space from which the input data comes. Therefore, no unit conversion is necessary.

Given the parameter values in the global struct **PARAMETERS**, the function `DECIDE()` must evaluate each of the *Launch Interceptor Conditions* (LICs) described below for the set of **NUMPOINTS** points:

(X[0],Y[0]) , ... , (X[**NUMPOINTS**-1],Y[**NUMPOINTS**-1])

where 2 ≤ **NUMPOINTS** ≤ 100

The *Conditions Met Vector* (CMV) should be set according to the results of these calculations, i.e. the global array element CMV[i] should be set to true if and only if the ith LIC is met.

The Launch Interceptor Conditions (LIC) are defined as follows:

0. There exists at least one set of two *consecutive* data points that are a distance greater than the length, **LENGTH1**, apart.

   (0 ≤ **LENGTH1**)

1. There exists at least one set of three *consecutive* data points that cannot all be contained within or on a circle of radius **RADIUS1**.

   $(0 \leq \textbf{RADIUS1})$

2. There exists at least one set of three *consecutive* data points which form an *angle* such that:

   angle $< (\textbf{PI} - \textbf{EPSILON})$

   or

   angle $> (\textbf{PI} + \textbf{EPSILON})$

   The second of the three *consecutive* points is always the *vertex* of the *angle*. If either the first point or the last point (or both) coincides with the *vertex*, the *angle* is undefined and the LIC is not satisfied by those three points.

   $(0 \leq \textbf{EPSILON} < \textbf{PI})$

3. There exists at least one set of three *consecutive* data points that are the vertices of a triangle with area greater than **AREA1**.

   $(0 \leq \textbf{AREA1})$

4. There exists at least one set of **Q_PTS** *consecutive* data points that lie in more than **QUADS** *quadrants*. Where there is ambiguity as to which *quadrant* contains a given point, priority of decision will be by *quadrant* number, i.e., I, II, III, IV. For example, the data point (0,0) is in quadrant I, the point (-l,0) is in quadrant II, the point (0,-l) is in quadrant III, the point (0,1) is in quadrant I and the point (1,0) is in quadrant I.

   $(2 \leq \textbf{Q\_PTS} \leq \textbf{NUMPOINTS})$, $(1 \leq \textbf{QUADS} \leq 3)$

5. There exists at least one set of two *consecutive* data points, (X[i],Y[i]) and (X[j],Y[j]), such that X[j] - X[i] < 0. (where i = j-1)

6. There exists at least one set of **N_PTS** *consecutive* data points such that at least one of the points lies a distance greater than **DIST** from the line joining the first and last of these **N_PTS** points. If the first and last points of these **N_PTS** are identical, then the calculated distance to compare with **DIST** will be the distance from the coincident point to all other points of the **N_PTS** *consecutive* points. The condition is not met when **NUMPOINTS** < 3.

   $(3 \leq \textbf{N\_PTS} \leq \textbf{NUMPOINTS})$, $(0 \leq \textbf{DIST})$

7. There exists at least one set of two data points separated by exactly **K_PTS** *consecutive* intervening points that are a distance greater than the length, **LENGTH1**, apart. The condition is not met when **NUMPOINTS** < 3.

   $1 \leq \textbf{K\_PTS} \leq (\textbf{NUMPOINTS} - 2)$

8. There exists at least one set of three data points separated by exactly **A_PTS** and **B_PTS** *consecutive* intervening points, respectively, that cannot be contained within or on a circle of *radius* **RADIUS1**. The condition is not met when **NUMPOINTS** < 5.

   $1 \leq \textbf{A\_PTS}$, $1 \leq \textbf{B\_PTS}$

   $\textbf{A\_PTS} + \textbf{B\_PTS} \leq (\textbf{NUMPOINTS} - 3)$

9. There exists at least one set of three data points separated by exactly **C_PTS** and **D_PTS** *consecutive* intervening points, respectively, that form an *angle* such that:

   angle $< (\textbf{PI} - \textbf{EPSILON})$

   or

   angle $> (\textbf{PI} + \textbf{EPSILON})$

   The second point of the set of three points is always the *vertex* of the *angle*. If either the first point or the last point (or both) coincide with the *vertex*, the *angle* is undefined and the LIC is not satisfied by those three points. When **NUMPOINTS** < 5, the condition is not met.

$1 \leq$ **C_PTS**, $1 \leq$ **D_PTS**

C_PTS + **D_PTS** $\leq$ **NUMPOINTS** $- 3$

10. There exists at least one set of three data points separated by exactly **E_PTS** and **F_PTS** *consecutive* intervening points, respectively, that are the vertices of a triangle with area greater than **AREA1**. The condition is not met when **NUMPOINTS** < 5.

    $1 \leq$ **E_PTS**, $1 \leq$ **F_PTS**

    E_PTS + **F_PTS** $\leq$ **NUMPOINTS** $- 3$

11. There exists at least one set of two data points, (X[i],Y[i]) and (X[j],Y[j]), separated by exactly **G_PTS** *consecutive* intervening points, such that X[j] - X[i] < 0. (where i $<$ j) The condition is not met when **NUMPOINTS** < 3.

    $1 \leq$ **G_PTS** $\leq$ **NUMPOINTS** $- 2$

12. There exists at least one set of two data points, separated by exactly **K_PTS** *consecutive* intervening points, which are a distance greater than the length, **LENGTH1**, apart. In addition, there exists at least one set of two data points (which can be the same or different from the two data points just mentioned), separated by exactly **K_PTS** *consecutive* intervening points, that are a distance less than the length, **LENGTH2**, apart. Both parts must be true for the LIC to be true. The condition is not met when **NUMPOINTS** < 3.

    $0 \leq$ **LENGTH2**

13. There exists at least one set of three data points, separated by exactly **A_PTS** and **B_PTS** *consecutive* intervening points, respectively, that cannot be contained within or on a circle of *radius* **RADIUS1**. In addition, there exists at least one set of three data points (which can be the same or different from the three data points just mentioned) separated by exactly **A_PTS** and **B_PTS** *consecutive* intervening points, respectively, that can be contained in or on a circle of *radius* **RADIUS2**. Both parts must be true for the LIC to be true. The condition is not met when **NUMPOINTS** < 5.

    $0 \leq$ **RADIUS2**

14. There exists at least one set of three data points, separated by exactly **E_PTS** and **F_PTS** *consecutive* intervening points, respectively, that are the vertices of a triangle with area greater than **AREA1**. In addition, there exist three data points (which can be the same or different from the three data points just mentioned) separated by exactly **E_PTS** and **F_PTS** *consecutive* intervening points, respectively, that are the vertices of a triangle with area less than **AREA2**. Both parts must be true for the LIC to be true. The condition is not met when **NUMPOINTS** < 5.

    $0 \leq$ **AREA2**

The *Conditions Met Vector* (CMV) can now be used in conjunction with the *Logical Connector Matrix* (LCM) to form the *off-diagonal elements* of the *Preliminary Unlocking Matrix* (PUM). The entries in the LCM represent the logical connectors to be used between pairs of LICs to determine the corresponding entry in the PUM, i.e. LCM[i,j] represents the boolean operator to be applied to CMV[i] and CMV[j]. PUM[i,j] is set according to the result of this operation. If LCM[i,j] is NOTUSED, then PUM[i,j] should be set to true. If LCM[i,j] is ANDD, PUM[i,j] should be set to true only if (CMV[i] AND CMV[j]) is true. If LCM[i,j] is ORR, PUM[i,j] should be set to true if (CMV[i] OR CMV[j]) is true. (Note that the LCM is symmetric, i.e. LCM[i,j]=LCM[j,i] for all i and j.)

3.6 Example 1

Assume that the given *Logical Connector Matrix* is as shown in Table 1. Also assume that the entries in the CMV have been computed as described, giving the results in Table 2. The PUM in Table 3 is generated.

Explanation of selected PUM entries:

1. PUM[0,1] is false because LCM[0,1] is ANDD, and at least one of CMV[0] and CMV[1] is false.

LIC	0	1	2	3	4	...	14
0	ANDD	ANDD	ORR	ANDD	NOTUSED	...	NOTUSED
1	ANDD	ANDD	ORR	ORR	NOTUSED	...	NOTUSED
2	ORR	ORR	ANDD	ANDD	NOTUSED	...	NOTUSED
3	ANDD	ORR	ANDD	ANDD	NOTUSED	...	NOTUSED
4	NOTUSED	NOTUSED	NOTUSED	NOTUSED	NOTUSED	...	NOTUSED
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
14	NOTUSED	NOTUSED	NOTUSED	NOTUSED	NOTUSED	...	NOTUSED

Table 1: Logical Connector Matrix (LCM) for Example 1

Condition	Value
0	false
1	true
2	true
3	true
4	false
.	.
.	.
.	.
14	false

Table 2: Conditions Met Vector (CMV) for Example 1

LIC	0	1	2	3	4	...	14
0	*	false	true	false	true	...	true
1	false	*	true	true	true	...	true
2	true	true	*	true	true	...	true
3	false	true	true	*	true	...	true
4	true	true	true	true	*	...	true
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
14	true	true	true	true	true	...	*

*Diagonal elements are input, not computed values.

Table 3: Preliminary Unlocking Matrix (PUM) for Example 1

LIC	0	1	2	3	4	.	.	.	14
0	true	false	true	false	true	.	.	.	true
1	false	false	true	true	true	.	.	.	true
2	true	true	true	true	true	.	.	.	true
3	false	true	true	false	true	.	.	.	true
4	true	true	true	true	false	.	.	.	true
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
14	true	true	true	true	true	.	.	.	false

Table 4: Preliminary Unlocking Matrix (PUM) for Example 2

Condition	Value
0	false
1	true
2	true
3	true
4	true
.	.
.	.
.	.
14	true

Table 5: Final Unlocking Vector (FUV) for Example 2

2. PUM[0,2] is true because LCM[0,2] is ORR, and at least one of CMV[0] and CMV[2] is true.
3. PUM[1,2] is true because LCM[1,2] is ORR, and at least one of CMV[1] and CMV[2] is true.
4. PUM[2,3] is true because LCM[2,3] is ANDD, and both CMV[2] and CMV[3] are true.
5. PUM[0,4] is true because LCM[0,4] is NOTUSED.

The *Final Unlocking Vector* (FUV) is generated from the *Preliminary Unlocking Matrix*. The input *diagonal elements* of the PUM indicate whether the corresponding LIC is to be considered as a factor in signaling interceptor launch. FUV[i] should be set to true if PUM[i,i] is false (indicating that the associated LIC should not hold back launch) or if all elements in PUM row i are true.

3.7 Example 2

Assume that the PUM now appears as in Table 4. The FUV generated is in Table 5.

Explanation of selected FUV entries:

1. FUV[0] is false because PUM[0,0] is true, but PUM[0,1] and PUM[0,3] are false.
2. FUV[1] is true because PUM[1,1] is false.
3. FUV[2] is true because PUM[2,i] is true for all i, $0 \leq i \leq 14$.

The final launch/no launch decision is based on the FUV.The decision to launch requires that all elements in the FUV be true, i.e. LAUNCH should be set to true if and only if FUV[i] is true for all i, $0 \leq i \leq 14$. For the example, LAUNCH is false because FUV[0] is false.

4 NONFUNCTIONAL REQUIREMENTS

1. The functional requirements are to be implemented by a parameterless C function named `DECIDE()`. It will perform no input or output, because the calling program will provide input data through global variables. Likewise, `DECIDE()` should store its results in global variables.

2. Whenever floating point numbers must be compared within the function `DECIDE()`, that comparison should be made with a fixed amount of precision. The program which calls `DECIDE()` will provide a function called `DOUBLECOMPARE()`. This function compares two floating point numbers, A and B, with respect to the six most significant digits. `DOUBLECOMPARE()` returns LT if $A < B$, EQ if $A = B$, or GT if $A > B$. `DECIDE()` should call this function for all comparisons of floating point numbers.

3. Every floating point number that is an input will be in the range $[-1,000,000...1,000,000]$. Furthermore, every floating point number that is an input will have been converted from an ASCII representation containing zeros in all decimal places below the thousandths place. Therefore, 1.253000 is a valid input but 1.253300 is not.

4. Information contained in the global variables when `DECIDE()` is called will remain valid throughout the execution of the function. There are no feedback or time series effects during a call to `DECIDE()`, or from multiple calls to `DECIDE()`.

5. Do not include input error checking. Assume that the calling program insures inputs are complete and within the specified range.

6. All floating point values used in your code must be of type `double`.

7. There are no constraints on memory space or execution time, but efficient, well-structured code with descriptive comments is preferred.

8. Each *matrix* shall be represented as follows. The matrix itself is a pointer to pointer to element. For example, `BMATRIX` in `decide.h` has type `boolean **`. This pointer refers to an array of type pointer to element. The number of elements of this top-level array is equal to the range of the first array dimension, and each of its members should point to a second-level array of type element. Each second-level array should have a number of members equal to the range of the second array dimension.

9. The `DECIDE()` function is intended to operate in real time and therefore should not perform memory management such as calling `malloc()` or `free()`.

A GLOSSARY

angle An angle is formed by two rays which share a common endpoint called a vertex. If one ray is rotated about the vertex until it coincides with the other ray,the amount of rotation required is the measure of the angle. Three points can be used to determine an angle by drawing a ray from the second point through the first point and another ray from the second point through the third point. Note that different angles are described according to whether the ray is rotated clockwise or counterclockwise. Either can be used in this problem because of the way the LIC's are defined.

CMV (Conditions Met Vector) The CMV is a boolean vector whose elements have a one-to-one correspondence with the launch interceptor conditions. If the radar tracking data satisfy a certain LIC, then the corresponding element of the CMV is to be set to true.

consecutive Two points are consecutive if they are adjacent in the input data vectors X and Y.Thus (X[i],Y[i]) and (X[i+1],Y[i+1]) are adjacent.

diagonal element Consider a matrix M, with n rows and n columns. The diagonal elements of the matrix are M[i,i] , where i=0,..,n-1.

FUV (Final Unlocking Vector) The FUV is a boolean vector which is the basis for deciding whether to launch an interceptor.If all elements of the FUV are true, a launch should occur.

LCM (Logical Connector Matrix) The LCM describes how individual LIC's should be logically combined. For example, the value of LCM[i,j] indicates whether LIC #i should combine with LIC #j by the logical AND, OR, or not at all.

LIC (Launch Interceptor Condition) If radar tracking data exhibit a certain combination of characteristics, then an interceptor should be launched. Each characteristic is an LIC.

matrix For purposes of this problem, a matrix can be considered to be a two-dimensional array.

off-diagonal element An off-diagonal element of a matrix is any element which is not a diagonal element.

planar data points Planar data points are points that are all located within the same plane.

PUM (Preliminary Unlocking Matrix) Every element of the boolean PUM corresponds to an element of the LCM. If the logical connection dictated by the LCM element gives the value "true", the corresponding PUM element is to be set to true.

quadrant The x and y axes of the Cartesian coordinate system divide a plane into four areas called quadrants. They are labeled I, II, III, IV, beginning with the area where both coordinates are positive and numbering counter-clockwise.

radius The length of the radius of a circle is the distance from the center of the circle to any point on the circle's circumference.

ray A ray is a straight line that extends from a point.

vector For purposes of this problem, a vector may be considered to be a one-dimensional array.

vertex When two rays originate from a common point to form an angle, the point of their origination is called the vertex of that angle.

B GLOBAL DECLARATIONS FROM `decide.h`

```
// This is version 4 of this file.

#include <math.h>

/////////// CONSTANT //////////

static const double PI = 3.1415926535;

/////////// TYPE DECLARATIONS ///////////

typedef enum { NOTUSED=777, ORR, ANDD } CONNECTORS;

// pointer to an array of 100 doubles
typedef double *COORDINATE;

// pointer to a 2-D array of [15,15] CONNECTORS
typedef CONNECTORS **CMATRIX;
```

```
// always in the range [0..1]
typedef int boolean;

// pointer to a 2-D array of [15,15] booleans
typedef boolean **BMATRIX;

// pointer to an array of 15 booleans
typedef boolean *VECTOR;

typedef enum { LT=1111, EQ, GT} COMPTYPE;

// inputs to the decide() function
typedef struct {
  double LENGTH1;    // Length in LICs 0, 7, 12
  double RADIUS1;    // Radius in LICs 1, 8, 13
  double EPSILON;    // Deviation from PI in LICs 2, 9
  double AREA1;      // Area in LICs 3, 10, 14
  int Q_PTS;         // No. of consecutive points in LIC 4
  int QUADS;         // No. of quadrants in LIC 4
  double DIST;       // Distance in LIC 6
  int N_PTS;         // No. of consecutive pts. in LIC 6
  int K_PTS;         // No. of int. pts. in LICs 7, 12
  int A_PTS;         // No. of int. pts. in LICs 8, 13
  int B_PTS;         // No. of int. pts. in LICs 8, 13
  int C_PTS;         // No. of int. pts. in LIC 9
  int D_PTS;         // No. of int. pts. in LIC 9
  int E_PTS;         // No. of int. pts. in LICs 10, 14
  int F_PTS;         // No. of int. pts. in LICs 10, 14
  int G_PTS;         // No. of int. pts. in LIC 11
  double LENGTH2;    // Maximum length in LIC 12
  double RADIUS2;    // Maximum radius in LIC 13
  double AREA2;      // Maximum area in LIC 14
} PARAMETERS_T;

///////////// global variable declarations ////////////

PARAMETERS_T PARAMETERS;

// X coordinates of data points
COORDINATE X;

// Y coordinates of data points
COORDINATE Y;

// Number of data points
int NUMPOINTS;

// Logical Connector Matrix
CMATRIX LCM;

// Preliminary Unlocking Matrix
```

```
BMATRIX PUM;

// Conditions Met Vector
VECTOR CMV;

// Final Unlocking Vector
VECTOR FUV;

// Decision: Launch or No Launch
boolean LAUNCH;

// compares floating point numbers -- see Nonfunctional Requirements
static inline
COMPTYPE DOUBLECOMPARE (double A,
                        double B)
{
  if (fabs(A-B)<0.000001) return EQ;
  if (A<B) return LT;
  return GT;
}

////////////// end of file //////////////
```