ENSF 694 – Summer 2024

**Principles of Software Development II**

**University of Calgary**

**Lab Assignment 1**

Student Name: Yael Gonzalez

Instructor: M. Moussavi, PhD, Peng

Submission Date: July 3, 2024

# Exercise A

## Source Code

/\*

 \*  File Name: lab1exe\_A.cpp

 \*  Assignment: ENSF 694 Lab 1, exercise A

 \*  Created by: Mahmood Moussavi

 \*  Completed by: Yael Gonzalez

 \*  Submission Date: July 3, 2024

 \*/

#include <iostream>

#include <cmath>

#include <format>

using namespace std;

const double G = 9.8; /\* gravitation acceleration 9.8 m/s^2 \*/

const double PI = 3.141592654;

void create\_table(double *v*);

/\*\*

 \* REQUIRES: v >= 0.

 \* PROMISES: prints a table showing projectile distance (d) and time (t) of flight for angles from

 \* 0 to 90 degrees given the specified initial velocity (v) of the projectile.

 \*/

double Projectile\_travel\_time(double *a*, double *v*);

/\*\*

 \* REQUIRES: a >= 0 and a <= 90, and v >= 0.

 \* PROMISES: calculates the time of flight (t) for a projectile given the specified launch angle (a)

 \* and initial velocity (v).

 \*/

double Projectile\_travel\_distance(double *a*, double *v*);

/\*\*

 \* REQUIRES: a >= 0 and a <= 90, and v >= 0.

 \* PROMISES: calculates the horizontal distance traveled (d) by a projectile given the specified

 \* launch angle (a) and initial velocity (v).

 \*/

double degree\_to\_radian(double *d*);

/\*\*

 \* REQUIRES: d >= 0 and d <= 90.

 \* PROMISES: converts an angle (d) in degrees to radians.

 \*/

int main(void)

{

    double velocity;

    cout << "Please enter the velocity at which the projectile is launched (m/sec): ";

    cin >> velocity;

    if (!cin) // means if cin failed to read

    {

        cout << "Invalid input. Bye...\n";

        exit(1);

    }

    while (velocity < 0)

    {

        cout << "\nplease enter a positive number for velocity: ";

        cin >> velocity;

        if (!cin)

        {

            cout << "Invalid input. Bye...";

            exit(1);

        }

    }

    create\_table(velocity);

    return 0;

}

void create\_table(double *v*)

{

    // Add column headers (names) and subheaders (units)

    cout << format("{:<10} {:<10} {:<10}\n", "Angle", "t", "d");

    cout << format("{:<10} {:<10} {:<10}\n", "(deg)", "(sec)", "(m)");

    // Iterate over angles from 0 to 90 degrees in steps of 5 degrees

    for (int deg = 0; deg <= 90; deg += 5)

    {

        // Convert degrees to radians

        double rad = degree\_to\_radian(deg);

        // Calculate time of flight

        double time = Projectile\_travel\_time(rad, *v*);

        // Calculate distance traveled

        double distance = Projectile\_travel\_distance(rad, *v*);

        // Print in console the angle, time, and distance

        cout << format("{:<10} {:<10.6f} {:<10.6f}\n",

                       deg,

                       time,

                       // Ensure distance is not negative (happens in 90°)

                       (distance < 0.0000001) ? 0.000000 : distance);

    }

}

double Projectile\_travel\_time(double *a*, double *v*)

{

    return 2 \* *v* \* sin(*a*) / G;

}

double Projectile\_travel\_distance(double *a*, double *v*)

{

    return (pow(*v*, 2.0) / G) \* sin(2 \* *a*);

}

double degree\_to\_radian(double *d*)

{

    return *d* \* PI / 180;

}

## Program Output

A screen shot of a computer

Description automatically generated

# Exercise B – Part II AR diagram

A diagram of a diagram

Description automatically generated

# Exercise C

## Source Code

/\*

 \*  File Name: lab1exe\_C.cpp

 \*  Assignment: ENSF 694 Lab 1 Exercise C

 \*  Created by: Mahmood Moussavi

 \*  Completed by: Yael Gonzalez

 \*  Submission Date: July 3, 2024

 \*/

#include <iostream>

using namespace std;

void time\_convert(int *ms\_time*, int \**minutes\_ptr*, double \**seconds\_ptr*);

/\*

 \* Converts time in milliseconds to time in minutes and seconds.

 \* For example, converts 123400 ms to 2 minutes and 3.4 seconds.

 \* REQUIRES:

 \*    ms\_time >= 0.

 \*    minutes\_ptr and seconds\_ptr point to variables.

 \* PROMISES:

 \*    0 <= \*seconds\_ptr & \*seconds\_ptr < 60.0

 \*    \*minutes\_ptr minutes + \*seconds\_ptr seconds is equivalent to

 \*    ms\_time ms.

 \*/

int main(void)

{

  int millisec;

  int minutes;

  double seconds;

  cout << "Enter a time interval as an integer number of milliseconds: ";

  // printf("Enter a time interval as an integer number of milliseconds: ");

  cin >> millisec;

  if (!cin)

  {

    cout << "Unable to convert your input to an int.\n";

    exit(1);

  }

  cout << "Doing conversion for input of " << millisec << " milliseconds ... \n";

  /\* MAKE A CALL TO time\_convert HERE. \*/

  time\_convert(millisec, &minutes, &seconds);

  cout << "That is equivalent to " << minutes << " minute(s) and " << seconds << " second(s).\n";

  return 0;

}

/\* PUT YOUR FUNCTION DEFINITION FOR time\_convert HERE. \*/

void time\_convert(int *ms\_time*, int \**minutes\_ptr*, double \**seconds\_ptr*)

{

  \**minutes\_ptr* = *ms\_time* / (60 \* 1000);

  \**seconds\_ptr* = (double)(*ms\_time* % (60 \* 1000)) / 1000.0;

}

## Program Output

A screen shot of a computer code

Description automatically generated

# Exercise D

## Part I – AR diagrams for points one, two, and three

### Point One

A screenshot of a computer

Description automatically generated

### Point Two

A screenshot of a calculator

Description automatically generated

### Point Three

A screenshot of a math form

Description automatically generated

## Part II

### Source Code

/\*

 \*  File Name: lab1exe\_D.cpp

 \*  Assignment: ENSF 694 Lab 1 Exercise D

 \*  Created by: Mahmood Moussavi

 \*  Completed by: Yael Gonzalez

 \*  Submission Date: July 3, 2024

 \*/

#include <iostream>

#include <iomanip>

using namespace std;

const int COL\_SIZE = 3;

const int ROW\_SIZE = 3;

void try\_to\_change(double \**dest*);

void try\_to\_copy(double *dest*[], double *source*[]);

double add\_them(double *a*[5]);

void print\_matrix(double *matrix*[][COL\_SIZE], int *rows*);

/\*

 \* PROMISES: displays the values in the elements of the 2-D array, matrix,

 \* formated in rows columns separated with one or more spaces.

 \*/

void good\_copy(double \**dest*, double \**source*, int *n*);

/\* REQUIRES: dest and source points to two array of double numbers with n to n-1 elements

 \* PROMISES: copies the values in each element of array source to the corresponding element

 \* in array dest.

 \*/

int main(void)

{

    double sum = 0;

    double x[4];

    double y[] = {2.3, 1.2, 2.2, 4.1};

    double matrix[ROW\_SIZE][COL\_SIZE] = {{10, 20, 30}, {40, 50, 60}, {70, 80, 90}};

    cout << " sizeof(double) is " << (int)sizeof(double) << " bytes.\n";

    cout << " size of x in main is: " << (int)sizeof(x) << " bytes.\n";

    cout << " y has " << (int)(sizeof(y) / sizeof(double)) << " elements and its size is: " << (int)sizeof(y) << " bytes.\n";

    cout << " matrix has " << (int)(sizeof(matrix) / sizeof(double)) << " elements and its size is: " << (int)sizeof(matrix) << " bytes.\n";

    try\_to\_copy(x, y);

    try\_to\_change(x);

    sum = add\_them(&y[1]);

    cout << "\n sum of values in y[1], y[2] and y[3] is: " << sum << endl;

    good\_copy(x, y, 4);

    cout << "\nThe values in array x after call to good\_copy are expected to be:";

    cout << "\n2.30, -8.25, 2.20, 4.10\n";

    cout << "And the values are:\n";

    for (int i = 0; i < 4; i++)

        cout << fixed << setprecision(2) << x[i] << "  ";

    cout << "\nThe values in matrix are:\n";

    print\_matrix(matrix, 3);

    cout << "\nProgram Ends...\n";

    return 0;

}

void try\_to\_copy(double *dest*[], double *source*[])

{

*dest* = *source*;

    /\* point one\*/

    return;

}

void try\_to\_change(double \**dest*)

{

*dest*[3] = 49.0;

    /\* point two\*/

    cout << "\n sizeof(dest) in try\_to\_change is " << (int)sizeof(*dest*) << " bytes.\n";

    return;

}

double add\_them(double *arg*[5])

{

    \**arg* = -8.25;

    /\* point three \*/

    cout << "\n sizeof(arg) in add\_them is " << (int)sizeof(*arg*) << " bytes.\n";

    cout << "\n Incorrect array size computation: add\_them says arg has " << (int)(sizeof(*arg*) / sizeof(double)) << " element.\n";

    return *arg*[0] + *arg*[1] + *arg*[2];

}

void good\_copy(double \**dest*, double \**source*, int *n*)

{

    for (int i = 0; i < *n*; i++)

    {

*dest*[i] = *source*[i];

    }

}

void print\_matrix(double *matrix*[][COL\_SIZE], int *rows*)

{

    for (int i = 0; i < *rows*; i++)

    {

        for (int j = 0; j < COL\_SIZE; j++)

        {

            // Print the element at the current row (i) and column (j) followed by a space

            cout << *matrix*[i][j] << " ";

        }

        // After printing current row, print a newline character

        cout << endl;

    }

}

### Program Output

A screenshot of a computer program

Description automatically generated

# Exercise E

A diagram of a diagram with colored wires

Description automatically generated

# Exercise F

*Note on how to submit: Copy and paste your source code, MyArray.cpp, and the program output into your lab report. Also upload your source code: MyArray.h,and MyArray.cpp into the Dropbox on the D2L.*

### Source Code

#### MyArray.h

/\*

 \*  File Name: MyArray.h

 \*  Assignment: ENSF 694 Lab 1 Exercise F

 \*  Created by: Mahmood Moussavi

 \*  Completed by: Yael Gonzalez

 \*  Submission Date: July 3, 2024

 \*/

#ifndef MY\_ARRAY\_H

#define MY\_ARRAY\_H

#include <iostream>

using namespace std;

#define SIZE 5

struct *MyArray*

{

    int array[SIZE];

    int list\_size;

};

void initialize(*MyArray* \**myArray*);

/\* REQUIRES: pointer myArray points to an object of struct MyArray

 \* PROMISES: initilizes the member myArray->list\_size to zero. In other words since

 \* myArray->array is empty the list\_size is set to zero.

 \*/

int search(const *MyArray* \**myArray*, int *obj*);

/\*

 \* REQUIRES: pointer myArray points to an object of struct MyArray.

 \* PROMISES: returns the position of first occurance of obj in myArray->array. Returns -1

 \* if there is no match for obj.

 \*/

void append(*MyArray* \**myArray*, int *array*[], int *n*);

/\*

 \* REQUIRES: pointer list points to an object of struct MyArray and array points to

 \* an arrary of n integer numbers.

 \* PROMISES: If (myArray->list\_size + n), is less than or equal SIZE appends the numbers in

 \* array to the end of the myArray->array. Otherwise, it does nothing.

 \*/

int retrieve\_at(*MyArray* \**myArray*, int *pos*);

/\*

 \* REQUIRES: pos >= 0, and pos < size(), and pointer myArray points to an object of struct

 \* MyArray.

 \* PROMISES: returns the value of myArray->array at the position pos.

 \*/

void insert\_at(*MyArray* \**myArray*, int *pos*, int *val*);

/\*

 \* REQUIRES: pos >= 0 and pos <= size(), and pointer myArray points to an object of struct

 \* MyArray.

 \* PROMISES: inserts the value of val in myArray->array[pos], after moving the values in the

 \* myArray->array to the right of element pos. Then, increments that list\_size by one.

 \*/

int remove\_at(*MyArray* \**myArray*, int *pos*);

/\*

 \* REQUIRES: pos >= 0 and pos <= size(), and pointer myArray points to an object of struct

 \* MyArray.

 \* PROMISES: removes the value of element myArray->array[pos], by moving the values in the

 \* elements of myArray->array, starting from position pos+1, to the left. if process is

 \* successful, increments list\_size by one. Also, returns the value of the element that was

 \* removed.

 \*/

int remove\_all(*MyArray* \**myArray*, int *value*);

/\*

 \* REQUIRES: value to be removed, and pointer myArray points to an object of struct MyArray.

 \* PROMISES: removes the values of all elements that match the specified value, by using the

 \* remove\_at function. Also, returns the count of the removed elements.

 \*/

void display\_all(*MyArray* \**myArray*);

/\*

 \* REQUIRES: pointer myArray points to an object of struct MyArray.

 \* PROMISES: displays the value in myArray->array from elment 0 to list\_size-1.

 \*/

bool is\_full(*MyArray* \**myArray*);

/\*

 \* REQUIRES: pointer myArray points to an object of struct MyArray.

 \* PROMISES: returns true is myArray->list\_size equals SIZE. Otherwise returns false.

 \*/

bool isEmpty(*MyArray* \**myArray*);

/\*

 \* REQUIRES: pointer myArray points to an object of struct MyArray.

 \* PROMISES: returns true is myArray->list\_size equals zero. Otherwise returns false.

 \*/

int size(*MyArray* \**myArray*);

/\*

 \* REQUIRES: pointer myArray points to an object of struct MyArray.

 \* PROMISES: returns value of myArray->list\_size.

 \*/

int count(*MyArray* \**myArray*, int *obj*);

/\*

 \* REQUIRES: pointer myArray points to an object of struct MyArray.

 \* PROMISES: returns the count of elements of myArray->array that their value is equal to obj.

 \*/

#endif

#### MyArray.cpp

/\*

 \*  File Name: MyArray.cpp

 \*  Assignment: ENSF 694 Lab 1 Exercise F

 \*  Created by: Yael Gonzalez

 \*  Submission Date: July 3, 2024

 \*/

#include "MyArray.h"

int search(const *MyArray* \**myArray*, int *obj*)

{

    // For each element in the list, if the element matches the search object return the index

    // (position) of the element.

    for (int i = 0; i < *myArray*->list\_size; i++)

    {

        if (*myArray*->array[i] == *obj*)

        {

            return i;

        }

    }

    return -1;

}

void initialize(*MyArray* \**myArray*)

{

*myArray*->list\_size = 0;

}

int retrieve\_at(*MyArray* \**myArray*, int *pos*)

{

    return *myArray*->array[*pos*];

}

int count(*MyArray* \**myArray*, int *obj*)

{

    int count = 0;

    // For each element in the list, if the element matches the object increment the count

    for (int i = 0; i < *myArray*->list\_size; i++)

    {

        if (*myArray*->array[i] == *obj*)

        {

            count++;

        }

    }

    return count;

}

void append(*MyArray* \**myArray*, int *array*[], int *n*)

{

    // If there is enough space to append the new elements, for each element in the input array:

    // Append it to the list and increment the list size

    if ((*myArray*->list\_size + *n*) <= SIZE)

    {

        for (int i = 0; i < *n*; i++)

        {

*myArray*->array[*myArray*->list\_size++] = *array*[i];

        }

    }

}

void insert\_at(*MyArray* \**myArray*, int *pos*, int *val*)

{

    // Shift elements to the right to make space for the new element

    for (int i = *myArray*->list\_size; i > *pos*; i--)

    {

*myArray*->array[i] = *myArray*->array[i - 1];

    }

    // Insert the new element at the specified position

*myArray*->array[*pos*] = *val*;

    // Increment the list size

*myArray*->list\_size++;

}

int remove\_at(*MyArray* \**myArray*, int *pos*)

{

    int removed\_value = *myArray*->array[*pos*];

    // Shift elements to the left to fill the gap

    for (int i = *pos*; i < *myArray*->list\_size - 1; i++)

    {

*myArray*->array[i] = *myArray*->array[i + 1];

    }

    // Decrement the list size

*myArray*->list\_size--;

    return removed\_value;

}

int remove\_all(*MyArray* \**myArray*, int *value*)

{

    int count = 0;

    // For every element in the list that matches the value to be removed:

    // Remove the element and increment the count of removed elements

    for (int i = 0; i < *myArray*->list\_size; i++)

    {

        if (*myArray*->array[i] == *value*)

        {

            remove\_at(*myArray*, i);

            count++;

            i--; // Adjust the index to account for the removed element

        }

    }

    return count;

}

void display\_all(*MyArray* \**myArray*)

{

    // Print each element in the list, one row per line

    for (int i = 0; i < *myArray*->list\_size; i++)

    {

        cout << *myArray*->array[i] << "  ";

    }

    cout << endl;

}

bool is\_full(*MyArray* \**myArray*)

{

    if (*myArray*->list\_size == SIZE)

    {

        return true;

    }

    return false;

}

bool isEmpty(*MyArray* \**myArray*)

{

    if (*myArray*->list\_size == 0)

    {

        return true;

    }

    return false;

}

int size(*MyArray* \**myArray*)

{

    return *myArray*->list\_size;

}

### Program Output

A screen shot of a computer

Description automatically generated