Course: ENSF694 – Summer 2025

Lab #: Lab 1

Instructor: Mahmood Moussavi

Student Name: John Zhou

Submission Date: July 10, 2025

# Exercise A

/\*

 \*  lab1exe\_A.cpp

 \*  ENSF 694 Lab 1, exercise A

 \*  Created by Mahmood Moussavi

 \*  Completed by: John Zhou

 \*/

#include <iostream>

#include <cmath>

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:

 \*   - velocity 'v' > 0.

 \*

 \* PROMISES:

 \*   - Prints a table showing the time of flight and distance for projectile for each angle from 0° to 90°, at the given velocity.

 \*/

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

/\*

 \* REQUIRES:

 \*   - a (angle) in degrees

 \*   - v (velocity) in m/s (v > 0).

 \*

 \* PROMISES:

 \*   - Returns the time of flight for the projectile

 \*     Formula: time = (v^2 \* sin(2 \* a)) / g

 \*/

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

/\*

 \* REQUIRES:

 \*   - a (angle) in degrees

 \*   - v (velocity) in m/s (v > 0).

 \*

 \* PROMISES:

 \*   - Returns the horizontal distance the projectile will travel

 \*     Formula: distance = (2 \* v \* sin(a)) / g

 \*/

double degree\_to\_radian(double d);

/\*

 \* REQUIRES:

 \*   - angle 'd' in degrees

 \*

 \* PROMISES:

 \*   - Returns the corresponding angle in 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 << "Invlid input. Bye...\n";

        exit(1);

    }

    while (velocity < 0)

    {

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

        cin >> velocity;

        if (!cin)

        {

            cout << "Invlid input. Bye...";

            exit(1);

        }

    }

    create\_table(velocity);

    return 0;

}

double degree\_to\_radian(double d)

{

    return d \* PI / 180.0;

}

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

{

    double radian = degree\_to\_radian(a);

    return v \* v \* sin(2 \* radian) / G;

};

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

{

    double radian = degree\_to\_radian(a);

    return 2 \* v \* sin(radian) / G;

};

void create\_table(double v)

{

    cout << "Angle\t\tt\t\td\n";

    cout << " (deg)\t\t(sec)\t\t(m)\n";

    for (int a = 0; a <= 90; a += 5)

    {

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

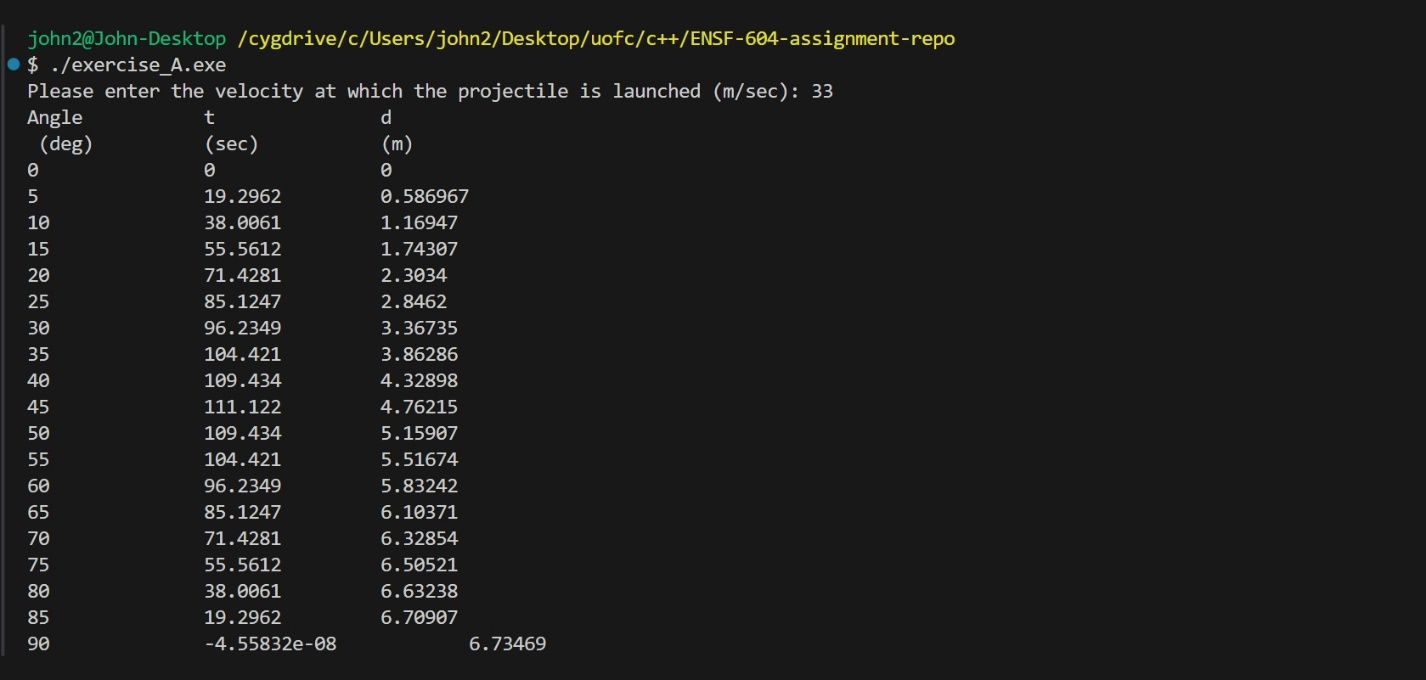
        double distance = Projectile\_travel\_distance(a, v);

        cout << a << "\t\t" << time << "\t\t" << distance << "\n";

    }

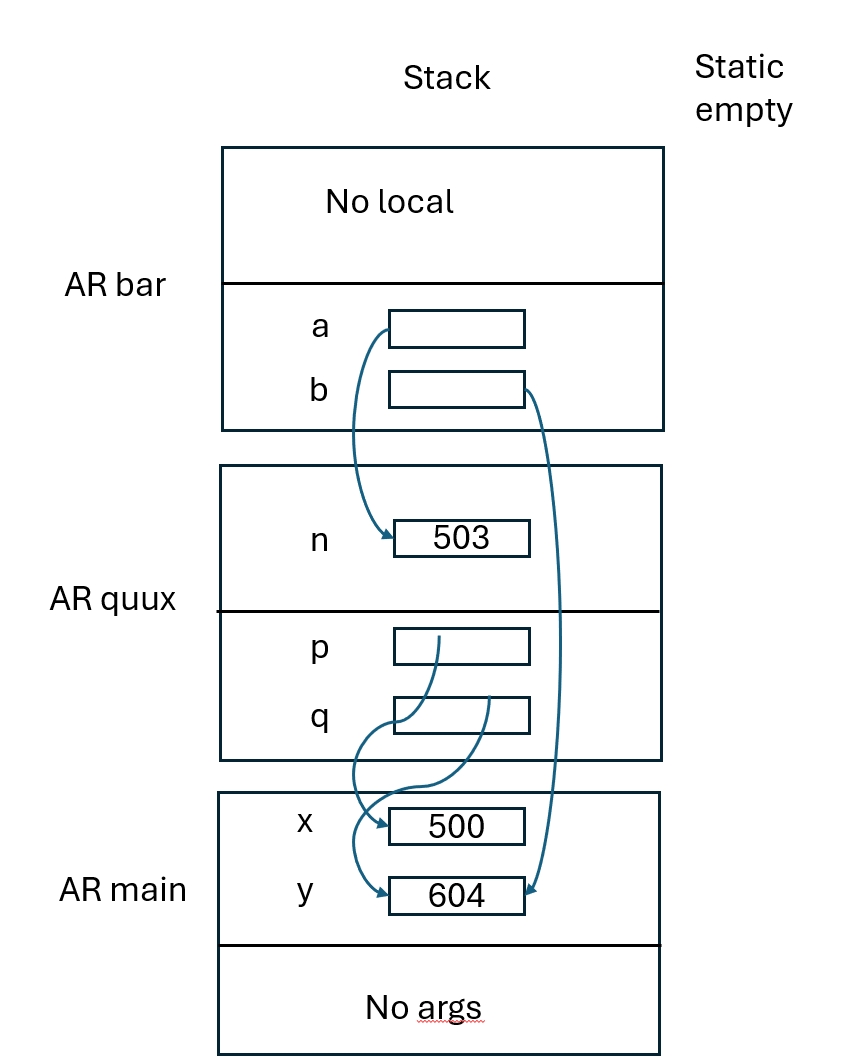
}

Execution result



Exercise B

## Part II



Exercise C

/\*

 \*  lab1exe\_C.cpp

 \*  ENSF 694 Lab 1, exercise C

 \*  Created by Mahmood Moussavi

 \*  Completed by: John Zhou

 \*/

#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;

}

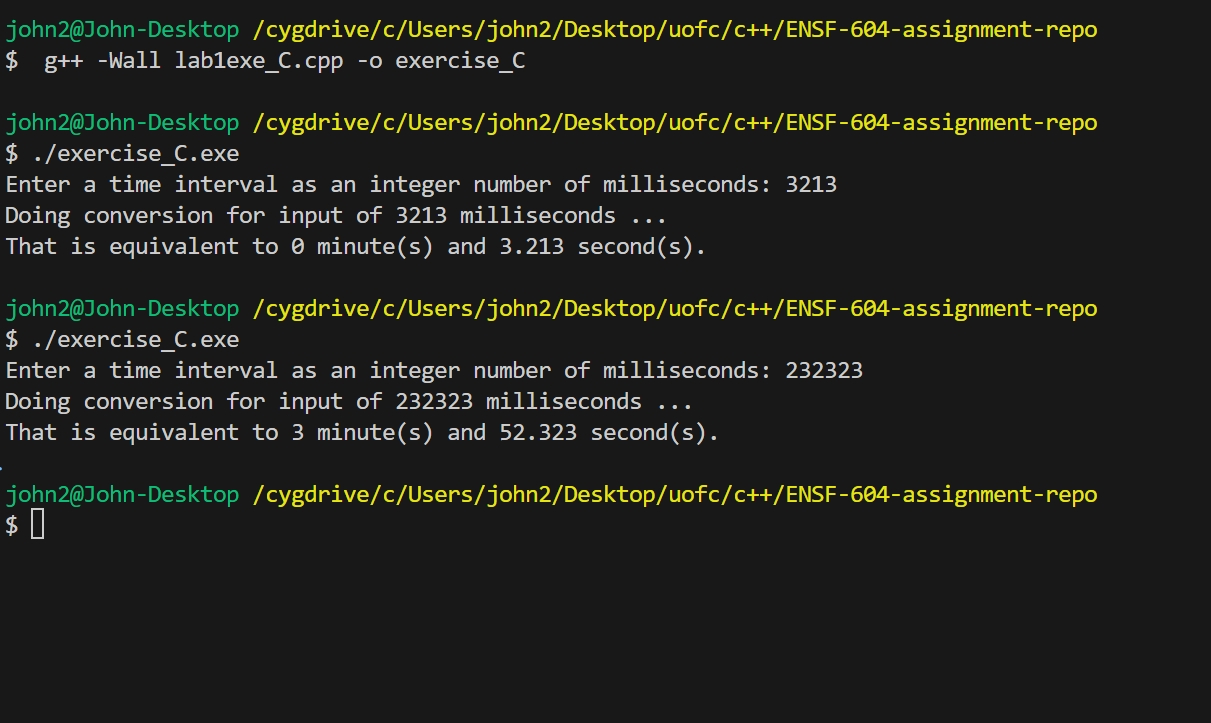
/\* PUT YOUR FUNCTION DEFINITION FOR time\_convert HERE. \*/

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

  \*minutes\_ptr=ms\_time/60000;

  \*seconds\_ptr=ms\_time%60000/1000.0;

}

Execution result  


Exercise D

## Part I

## In the static area, there will be the Global constants const int COL\_SIZE = 3;

## const int ROW\_SIZE = 3;

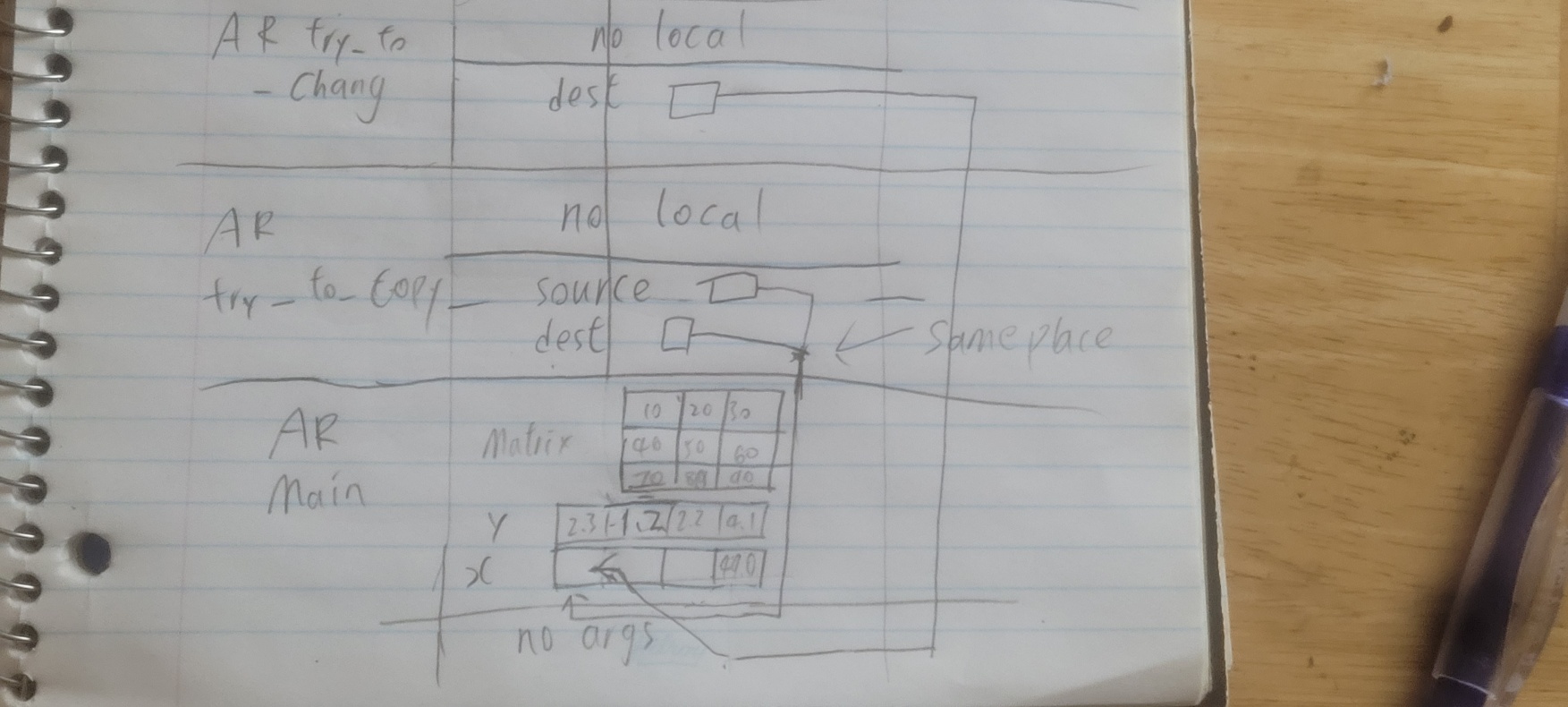
## The String constant area will have : " sizeof(double) is \0", " bytes.\n \0" etc. AR at Point 1 for stack

## 

## AR at Point 2 for stack

## 

## AR at Point 3 for stack

  
  
Part II

/\*

 \*  lab1exe\_D.cpp

 \*  ENSF 694 Lab 1, exercise D

 \*  Created by Mahmood Moussavi

 \*  Completed by: John Zhou

 \*/

#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)

{

    cout << "\_\_\_\_\_\_\_\_\_\_\_Print Matrix\_\_\_\_\_\_\_\_\_\_\n";

    for (int i = 0; i <rows; i++)

    {

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

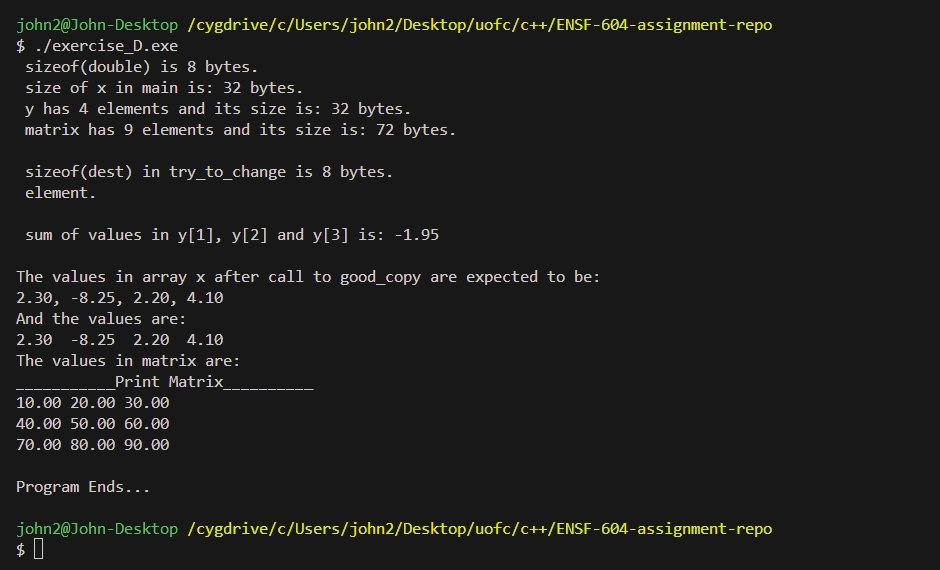
           cout << matrix[i][j]<<" ";

        }

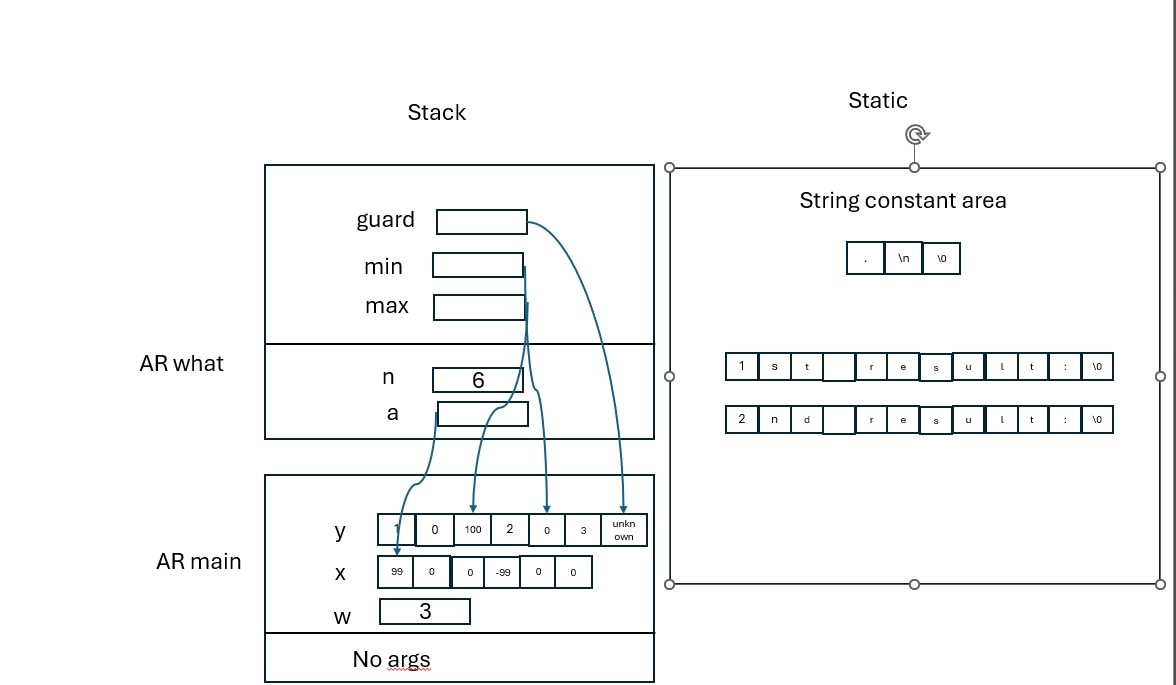
         cout << "\n";

    }

}

execution output

# Exercise E



# Exercise F

#include "MyArray.h"

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

{

     int currentSize = myArray->list\_size;

    for (int i = 0; i < currentSize; 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)

{

    int currentSize = size(myArray);

    if (pos >= 0 && pos < currentSize)

    {

        return myArray->array[pos];

    }

    return 0;

}

int count(MyArray \*myArray, int obj)

{

    int occurances\_count = 0;

    int currentSize = size(myArray);

    for (int i = 0; i < currentSize; i++)

    {

        if (myArray->array[i] == obj)

        {

            occurances\_count++;

        }

    }

    return occurances\_count;

}

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

{

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

    {

        int \*start = myArray->array + myArray->list\_size;

        for (int i = 0; i < n; i++)

        {

            start[i] = array[i];

        }

        myArray->list\_size += n;

    }

}

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

{

    int arrayCurrentSize = size(myArray);

    if (pos >= 0 && pos <= arrayCurrentSize)

    {

        for (int i = arrayCurrentSize; i > pos; i--)

        {

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

        }

        myArray->array[pos] = val;

        myArray->list\_size++;

    }

}

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

{

    int arrayCurrentSize = size(myArray);

    if (pos >= 0 && pos < arrayCurrentSize)

    {

        int removedElement = myArray->array[pos];

        for (int i = pos; i < arrayCurrentSize-1; i++)

        {

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

        }

        myArray->list\_size--;

        return removedElement;

    }

    return 0;

}

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

{

    int countRemoved = 0;

    int currentSize = size(myArray);

    for (int i = 0; i < currentSize; i++)

    {

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

        {

            remove\_at(myArray, i);

            countRemoved++;

            i--;

            currentSize--;

        }

    }

    return countRemoved;

}

// You can modify this function however you want:  it will not be tested

void display\_all(MyArray \*myArray)

{

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

    {

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

    }

    cout << endl;

}

bool is\_full(MyArray \*myArray)

{

    return myArray->list\_size == SIZE;

}

bool isEmpty(MyArray \*myArray)

{

    return myArray->list\_size == 0;

}

int size(MyArray \*myArray)

{

    return myArray->list\_size;

}

Program output

