## A. Exercise: Volume of a Horizontal Cylinder – Using arrays with functions (5 points)

Working code:

```
/*-------------------------------------------------------------------
File: cylinderVolume.c (Lab 5)
Description: Calculates how the volume changes w.r.t the depth
of a liquid in a cylinder.
-------------------------------------------------------------------*/
#include <stdio.h>
#include <math.h>
// Define symbolic constant
#define TRUE 1
#define FALSE 0
#define NUM_VALUES 20 // Number of time/velocity values to display
#define NUM_ROWS 2 // Number of rows in 2D array - one for depth and one for volume
#define D_IX 0 // Index for elements containing depth values (row 0)
#define V_IX 1 // Index for elements containing volume values (row 1)


// Definition of a structure type for user input
typedef struct
{
char orientation; // h for horizontal and v for vertical
double radius; // in m
double length; // in m
} CYLINDER;
// Prototypes
void getInput(CYLINDER *);
double getPositiveValue(char *);
void fillArray(CYLINDER *, int n, double[][n]);
double computeVolume(CYLINDER *, double);
void displayTable(CYLINDER *, int n, double *, double *);

/*-------------------------------------------------------------------
Function: main
Description: Gets from the user values for the radius and length of the
cylinder as well as it orientation. Fills an array with
depth/volume values and then displays in a table these values.
-------------------------------------------------------------------*/
int main()
{
// Variable declarations
CYLINDER cyl; // structure variable fo cylinder
// Note in the following 2D array
// points[D_IX] is a 1D array that contains the depth values
// points[V_IX] is a 1D array that contains the volume values
double values[NUM_ROWS][NUM_VALUES]; // NUM_VALUES depth/volume value pair
```

```
// Get user input
getInput(&cyl);//there was no "&"
// Fill in the array with depth/volume values
fillArray(&cyl, NUM_VALUES, values);
// Display table on console
displayTable(&cyl, NUM_VALUES, values[D_IX], values[V_IX]);
return(0);
}
/*---------------------------------------------------------------------
Function: getOrientation
Description: Requests from the user the orientation of the
cylinder: v for vertical and h for horizontal
---------------------------------------------------------------------*/
void getInput(CYLINDER *cylPtr)
{
int flag;
// Get the cylinder radius and length
cylPtr->radius = getPositiveValue("Please enter the value for the cylinder radius: "); // radius should be
accessed using -> operator

cylPtr->length = getPositiveValue("Please enter the value for the cylinder length: ");
// Get the orientation of the cylinder
do
{
flag = FALSE;
printf("Give the cylinder's orientation, v for vertical, h for horizontal: ");
fflush(stdin);
scanf("%c", &cylPtr->orientation);
if(cylPtr->orientation != 'v' && cylPtr->orientation != 'h')
{
printf("Bad input.\n");
flag = TRUE;
}
}
while(flag);
}
/*---------------------------------------------------------------------
Function: getPositiveValue
Parameter:
prompt: reference to a prompt message to the user
Returns: A value strictly positive (>0)
Description: Prompts for and reads a real value from the user, checks that it is strickly
positive, and returns the value.
---------------------------------------------------------------------*/
double getPositiveValue(char *prompt)
{
double value;
```

```
do
{
printf("%s",prompt); // specify the format
scanf("%lf",&value);
if(value <= 0.0)
printf("The value must be greater than zero.\n");
}
while(value <= 0.0);
return(value);
}
/*---------------------------------------------------------------------
Function: fillArray
Parameters:
cyl: pointer to a variable structure that gives the cylinder characteristics
n: gives the number of columns in the 2D array and thus the number of depth/volume
pairs to store in the 2D array
mat: pointer to a 2D array of double values with n columns.
Descripton: Fills the 2D array (matrix) which contains the depth/volume values.
The values of the volume are computed with calls to computeVolume.
---------------------------------------------------------------------*/
void fillArray(CYLINDER *cyl, int n, double mat[][n])
{
// to fill the array
double depth; // depth of the liquid
int cix; // for indexing the columns in the 2D array
double inc; // for incrementing the depth
// Fill the 2D array with depth/volume values
depth = 0;
if(cyl->orientation == 'v') inc = cyl->length/(n-1); // use n instead of NUM_VALUES
else inc = cyl->radius/(n-1);
for(cix = 0; cix < n; cix = cix + 1) // loop over n columns
{
mat[D_IX][cix] = depth; // cix determines the column and D_IX determines the row number for depth
mat[V_IX][cix] = computeVolume(cyl, depth); // cix determines the column and V_IX determines the row
number for volume
depth = depth + inc;
}
}
/*---------------------------------------------------------------------
Function: computeVolume
Parameter
cyl - pointer to a CYLINDER structure variable
depth - depth of the liquid in the cylinder
Returns: The volume of the liquid in the cylinder
Description: Computes the volume of a liquid in a cylinder with
the radius, length and orientation given in cyl.
```

```
----------------------------------------------------------------------*/
double computeVolume(CYLINDER *cyl, double depth)
{
// Declaration of variables
double vol; // liquid volume
// Computation depends on orientation of the cylinder
if(cyl->orientation == 'v')
vol = M_PI*cyl->radius*cyl->radius*depth;
else
{
// Utilise plusieurs intructions pour faire le calcule
// Les valeurs intermédiaires sont accumulées dans vol
vol = sqrt(2*cyl->radius*depth - depth*depth);
vol = (cyl->radius - depth)*vol;
vol = (pow(cyl->radius,2)*acos((cyl->radius - depth)/cyl->radius)) - vol;
vol = vol*cyl->length;
}
return(vol);
}
/*----------------------------------------------------------------------
Function: displayTable
Parameters:
cyl - pointer to a variable structure that gives the cylinder characteristics.
int n - nnumber of elements in the referenced arrays
depth - pointer to an array that contains the depth values
volume - pointer to an array that contains the volume values
Description: Displays the characteristics of the cylinder followed by
a table that shows how the volume varies the depth of a liquide
in the cylinder. The data is given in the arrays referenced
by the parameters.
----------------------------------------------------------------------*/
void displayTable(CYLINDER *cyl, int n, double *depth, double *volume)
{
// Declaration of variables
int ix; // Index into arrays.
// Display results
printf("\nThe change in liquid volume of the cylinder with radius %.2f \nand length %.2f as depth changes
when ",
cyl->radius, cyl->length);
if(cyl->orientation == 'v') printf("vertical\n");
else printf("horizontal\n");
printf("%10s %10s\n","Depth", "Volume");
printf("----------------------\n");
for(ix = 0; ix < n; ix = ix + 1) // loop from 0 to n-1
printf("%10.3f %10.2f\n", depth[ix], volume[ix]);
}
```

## B. Exercise: Programming Model (5 points)

## C. Exercise: Revisiting the Free-Falling Parachutist – Using Loops to work with Arrays (10 marks)

```
/*-------------------------------------------------
File: parachutist.c (Lab 5)
Author:
Description: Calculates change in velocity for
the parachutist over time and displays a table
showing the results.
-------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
// Define symbolic constant
#define G 9.8 // in m/s^2
#define N 50 // number of data points to compute
#define TRUE 1
#define FALSE 0
// Declaration of a structure for user input
typedef struct
{
// Add the members to the structure
double weight;
double drag;
double finalTime;
double times[N];
double velocities[N];
} PARACHUTIST;
// Function prototypes
void getInput(PARACHUTIST*);
void computeVelocity(PARACHUTIST*);
void displayTable(PARACHUTIST*);
/*-------------------------------------------------------------
Function: main
Description: This function controls the overall program.
It makes three calls, one to get user input, one
to compute time/velocity data points, and a third
to display the calculated values in a table.
-------------------------------------------------------------*/
void main()
{
PARACHUTIST para; // A structure variable with parachutist data
```

```
// Get input from user
getInput(&para);
// Compute time/velocity data points
computeVelocity(&para);
// Display the results
displayTable(&para);
}
/*--------------------------------------------------------------------------
Function: getInput
Parameters
pPrt - pointer to a PARACHUTIST structure variable
Description: Obtains from the user, the weight, drag and final time
values and stores in the referenced data structure.
All values must be greater than zero.
--------------------------------------------------------------------------*/
void getInput(PARACHUTIST* pPtr)
{
// Declaration of variable
double weight;
double drag;
double finalTime;
// Get input from user
printf("Please enter the values for Weight :");
scanf("%lf", &weight);
printf("Please enter the values for drag coefficient:");
scanf("%lf", &drag);
printf("Please enter the values for final time:");
scanf("%lf", &finalTime);
pPtr->drag = drag;
pPtr->finalTime = finalTime;
pPtr->weight = weight;
}
/*--------------------------------------------------------------------------
Function: computeVelocity
Parameters
pPrt - pointer to a PARACHUTIST structure variable
Description: Using the weight, drag and final time values in the referenced
structure, computes and stores N+1 points in the two arrays
found in the referenced structure variable.
--------------------------------------------------------------------------*/
void computeVelocity(PARACHUTIST* pPtr)
{
// Declaration of variables
int i = 0;
int n = 10;
double sum = 0;
```

```
double t = (pPtr->finalTime / N);
// Calculate the velocity points and store in the arrays
for (i = 0; i <= N; i++) {
pPtr->times[i] = t * i;
pPtr->velocities[i] = ((G * pPtr->weight) / pPtr->drag) * (1 - exp(-(pPtr->drag / pPtr->weight) * pPtr->times[i]));
//printf("%.2f\t\t%.2f\n",pPtr->times[i],pPtr->velocities[i]);
}
}
/*--------------------------------------------------------------------------
Function: displayTable
Parameters
pPrt - pointer to a PARACHUTIST structure variable
Description: Displays a table of the calculated data points.
--------------------------------------------------------------------------*/
void displayTable(PARACHUTIST* pPtr)
{
// Declaration of variables
int i; // index into arrays
// Display results
printf("The change in velocity of the parachutist with weight %.2f kg\nand a drag coefficient %.2f kg/s is as
follows.\n",
pPtr->weight, pPtr->drag);
printf("%10s %10s\n", "Time t", "Velocity v");
printf("------------------------\n");
// Display the contents of the table.
for (i = 0; i <= N; i++) {
printf("%.2f\t\t%.2f\n", pPtr->times[i], pPtr->velocities[i]);
}
}
```