GNG 1106

LAB 4 2020

Exercise: Using Structures and Array

```
/*-------------------------------------------------------------
 File:  cylinderVolume.c  (Lab 4)
 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 D_IX 0 // Index for elements containing depth values (row 0)
#define V_IX 1 // Index for elements containing volume values (row 0)
// 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
double getPositiveValue();
char getOrientation();
double computeVolume(CYLINDER, double);
/*-------------------------------------------------------------
Function: main
Description:  Gets from the user values for the radius and lenght 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 points[2][NUM_VALUES];  // NUM_VALUES points of depth/volume
   // to fill the arrays
   double depth; // depth of the liquid
   int ix;   // for indexing the columns in the 2D array
   double inc;  // for incrementing the depth

   // Get input from user, the cylinder radius and length
   printf("Please enter the value for the cylinder radius: ");
   cyl.radius = getPositiveValue();
   printf("Please enter the value for the cylinder length: ");
   cyl.length = getPositiveValue();
   cyl.orientation = getOrientation();
   // Fill the 2D array with depth/volume values
   depth = 0;
   if((cyl.orientation == 'v')) inc = cyl.length/(NUM_VALUES-1);
   else  inc = cyl.radius/(NUM_VALUES-1);
   for(ix = 0; ix < NUM_VALUES; ix = ix + 1)
   {
      points[D_IX][ix] = depth;
      points[V_IX][ix] = computeVolume(cyl, depth);
      cyl.length = cyl.length + inc;
   }
```

```
   // Display results
   printf("The 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 <= NUM_VALUES; ix = ix + 1)
      printf("%10.3f    %10.2f\n", points[V_IX][ix], points[V_IX][ix]);
   return(0);
}
/*--------------------------------------------------------------------------
Function: getPositiveValue
Returns: A value strictly positive (>0)
Description: Reads a real value from the user, checks that it is strickly
            positive, and returns the value.
--------------------------------------------------------------------------*/
double getPositiveValue()
{
   double value;
   do
   {
      scanf("%lf",&value);
      if(value <= 0.0)
         printf("The value must be greater than zero: \n");
   }
   while(value <= 0.0);
   return(value);
}
```

```
/*-------------------------------------------------------------------
Function: getOrientation
Returns: 'v' for vertical and 'h' for horizontal.
Description: Requests from the user the orientation of the
            cylinder: v for vertical and h for horizontal
-------------------------------------------------------------------*/
char getOrientation()
{
   char orientation;
   int flag;
   do
   {
      flag = FALSE;
      printf("Give the cylinder's orientation, v for vertical, h for horizontal: ");
      fflush(stdin);
      scanf("%c", &orientation);
      if(orientation != 'v' && orientation != 'h')
      {
         printf("Bad input.\n");
         flag = TRUE;
      }
   }
   while(flag);
   return(orientation);
}
/*-------------------------------------------------------------------
Function: computeVolume
Parameter
   cyl - CYLINDER structure variable
   depth - depth of the liquid in the cylinder
```

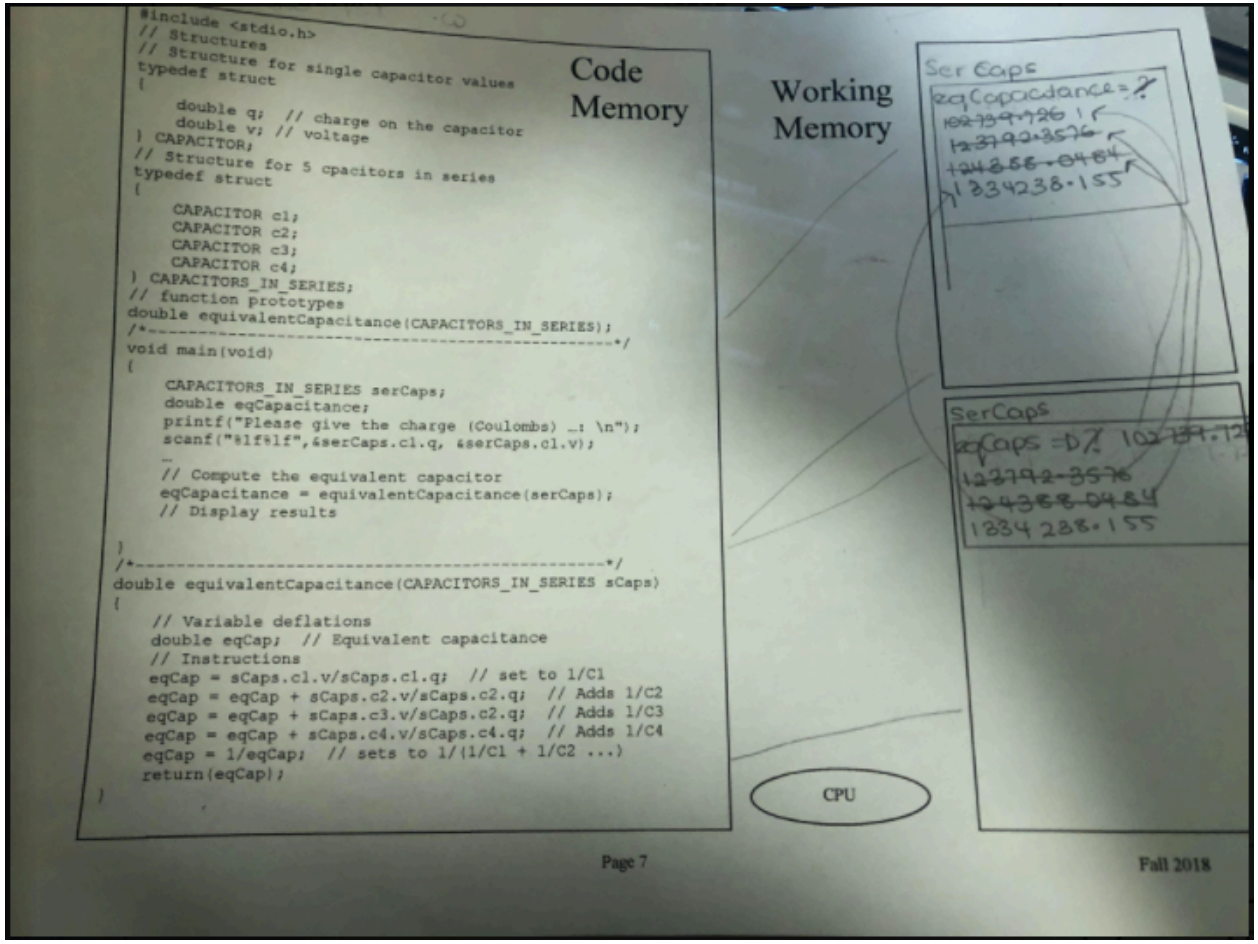
```
Returns: The volume of the liquid in the cylinder
Description: Computes the volume of a liquid in a cylinder with
             the radius, length and orientation given in cyl.
-----------------------------------------------------------------------*/
double computeVolume(CYLINDER cyl, double depth)
{
   // 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);
}
```

Exercise: Free Falling Parachutist

```
/*-------------------------------------------------------------
File: parachutist.c
Author: Gilbert Arbez, Fall 2016
Description: Computes the velocity of a parachutist after time t.
-------------------------------------------------------------*/
#include <stdio.h>
#include <math.h>
// Symbolic Constant
#define G 9.8    // in m/s^2
#define NUM_VALUES 30  // Number of time/velocity values to display
// Structure for user input
typedef struct
{
double weight;
double drag;
double finalTime;
}USER_INPUT;

void fprint_line(FILE* stream, int length);
void calculateVelocity(USER_INPUT ui,double times[],double velocities[]);

int main()
{
   //Declaring arrays
   double times[NUM_VALUES];
   double velocities[NUM_VALUES];

   //Declaring struct variable
   USER_INPUT ui;
```

```
    //Getting the input entered by the user

    printf("Please enter the values for Weight :");
    scanf("%lf",&ui.weight);
    printf("Please enter the values for drag coefficient:");
    scanf("%lf",&ui.drag);

    printf("Please enter the values for final time:");
    scanf("%lf",&ui.finalTime);

//calling the function
calculateVelocity(ui,times,velocities);
return 0;
}

void calculateVelocity(USER_INPUT ui,double times[],double velocities[])

{

    double time=0;
    int i=0;
    int n=10;
    double sum=0;
    double t=(ui.finalTime/29);

    printf("\nThe Speed of the parachutist with weight %.2f kg\n",ui.weight);
    printf("and a drag coefficient %.2f kg/s\n",ui.drag);
    printf("Time\t\tVelocity m/s\n");
    printf("----------------------\n");
```

```
    for(i=0;i<NUM_VALUES;i++)
    {
       times[i]=t*i;
       velocities[i]=((G*ui.weight)/ui.drag)*(1-exp(-(ui.drag/ui.weight)*times[i]));
       printf("%.2f\t\t%.2f\n",times[i],velocities[i]);
    }

}
```


#include <stdio.h>
// Structures
// Structure for single capacitor values
typedef struct
{
double q; // charge on the capacitor
double v; // voltage
} CAPACITOR;
// Structure for 5 cpacitors in series
typedef struct
{
CAPACITOR c1;
CAPACITOR c2;
CAPACITOR c3;
CAPACITOR c4;
} CAPACITORS_IN_SERIES;
// function prototypes
double equivalentCapacitance(CAPACITORS_IN_SERIES);
/*---*/
void main(void)
{
CAPACITORS_IN_SERIES serCaps;
double eqCapacitance;
printf("Please give the charge (Coulombs) …: \n");
scanf("%lf%lf",&serCaps.c1.q, &serCaps.c1.v);
…
// Compute the equivalent capacitor
eqCapacitance = equivalentCapacitance(serCaps);
// Display results
}
/*---*/
double equivalentCapacitance(CAPACITORS_IN_SERIES sCaps)
{
// Variable deflations
double eqCap; // Equivalent capacitance
// Instructions
eqCap = sCaps.c1.v/sCaps.c1.q; // set to 1/C1
eqCap = eqCap + sCaps.c2.v/sCaps.c2.q; // Adds 1/C2
eqCap = eqCap + sCaps.c3.v/sCaps.c2.q; // Adds 1/C3
eqCap = eqCap + sCaps.c4.v/sCaps.c4.q; // Adds 1/C4
eqCap = 1/eqCap; // sets to 1/(1/C1 + 1/C2 ...)
return(eqCap);
}
Code Memory
Working Memory
SerCaps
eqCapacitance = ?
1334238.155
SerCaps
eqCaps = D?
1334288.155
CPU
Page 7
Fall 2018