```c
#include <stdio.h>

/// TOPIC 1 STRUCTURES

/* DATA STRUCTURES

Structure --> group of variables under one name which may contain different types DIFF from ARRAY

THE STRUCTURE DEFINITION AND THE STRUCTURE VARIABLE

Structure type must first be defined BEFORE using the declaration clauses within the main function
members of the structures can be:
1. Variables of basic type
2. Other structure variables; meaning that the structues can be composed of various other structures --> EX) a FILE

DEFINITION OF STRUCTURE TYPE IN C

the sturcture type must first be define before its declaration in the main function

SYNTAX
*/

typedef struct
{
    type name1;
    type name2;
    type name3;
    .
    .
    .
    type namen;

} NAME OF STRUCTURE;

/*

the keyword "struct" introduces the structure definition and gives it a name f a type(structTypeName)

the variables are declared within the {} will encompass the members of the structure
--> Members of the same structure definition MUST HAVE UNIQUE NAMES aka can't have two members with the name x_1.

NOTE defining a new structure is defining a new TYPE --> complex

NOTE defining a structure does not reserve ANY MEMORY

*/

/// consider the example below

struct cube
{
    char color;
    double height;
    double width;
    double length;
};

/* This structure definition contains a character "color"

instead of always defining a new structure via struct cube, we can define a new type as follows
"typedef struct cube CUBE;
CUBE --> is referred to as the user type
the type "struct cube" is now represented by the type CUBE which encompasses the elements
-color
```

```
-height
-width
-length

*/

/// Definition of new Types

/// it's also possible to combine the definition of a structure with the definition of a user type

typedef struct structTypeName
{
    type name1;
    type name2;
} NEW_TYPE;

/// the following demonstrates the three ways that a structure can be defined

/// METHOD 1

struct cubestr
{
    char color;
    double height;
};

/// METHOD 2

typedef struct cubestruct
{
    char color;
    double height;
} CUBE;

/// METHOD 3

typedef struct
{
    char color;
    double height;
} CUBE_1;

/// METHOD 3 IS THE METHOD ADOPTED IN THIS COURSE

/// DECLARING STRUCTURE VARIABLES --- EXAMPLES

/* In this course, the use of a USER TYPE to declare a structure variable will be common practice

therefore in the main function; following one of the examples in the 3 methods above, to declare a
variable type of the structures defined above, we may do the following
*/

void main()
{
    CUBE cube1;
}

///* We now have a variable of type CUBE (encompassing all members of the CUBE structure, termed cube1

/* once declared in the main function, the structure is now allocated memory space.

*/



/// OPERATIONS WITH STRUCTURE VARIABLES
```

```
131
132  /* The ONLY operations that can be performed on a structure variable are:
133
134  1. ASSIGNEMENT of structure variables to another structure variable susing the "=" operator
135  2. ACCESSING a structure's members
136
137  NOTE THAT STRUCTURE VARIABLES CANNOT BE COMPARED!!!! THEY MAY ONLY BE USED IN EXPRESSIONS ACCORDING
138  TO THEIR TYPES
139
140  */
141
142  /// INITIALIZING A STRUCTURE VARIABLE
143
144  /* A structure variable can be initialized when it is declared by assigning and intializer list enclosed in
{} to the variable
145
146  In other words, we do not always need to declare the variable in one line, then assign the values to the
variables in subsequent
147  lines. This process can be done in one step as shown below
148  */
149
150  /// CUBE cube1 = {'r', 2.4, 5.0, 7.0};
151
152  /* In this manner, all four members of the structure have been assigned a variable as they were declared.
color, width, height and length
153
154  */
155
156  /// HOW TO USE THE MEMBERS OF THE STRUCTURE VARIABLE?
157
158  /* members of a structure can be accessed using the structure member operator which is essential a period
"." aka
159  the DOT OPERATOR
160
161  the DOT OPERATOR is placed between the structure name and the name of the member.
162
163  EX*/
164
165  cube.color=RED;
166  cube.height=1.2;
167  cube.width=2.5;
168  cube.length=4.2;
169  volume = cube.height*cube.length*cube.width;
170
171  /// The above accesses all three members of type double to calculate volume
172
173  /// TOPIC 2: ARRAYS
174
175  /// What are ARRAYS?
176  /* An array is group of elements of the same type that are grouped together in the memory under a single
name.
177
178  an array can be composed of a single dimension [] or two dimmensions [][].
179
180  An array is also a DATA STRUCTURE that can conveniently store LARGE amounts of data of the same type -->
181  it's a STATIC data structure, meaning that its size does not vary duiring the execution of programs, since
the
182  elements in the memory are defined when declaring the array (you will see this later)
183
184  */
185
186  /// Suppose a program reads 5 integers and displays them in reverse order
187
188  scanf("%d", &i1);
189  scanf("%d", &i2);
190  scanf("%d", &i3);
```

```c
scanf("%d", &i4);
scanf("%d", &i5);

printf("%d\n", i5);
printf("%d\n", i4);
printf("%d\n", i3);
printf("%d\n", i2);
printf("%d\n", i1);

/* Displaying this is easy since there are only 5 elements, but what if the request is of 1000 integers

To express this in terms of arrays, we can declare an array (arr) as follows */

double arr[1000];
arr[0]=1.0;
arr[1]=2.0;
.
.
.
.
.
.
.
.
arr[999]=1000.0;

/// Note that the number within the [] reflects the total # of elements-1
/// This is due to the fact that an array starts listing its elements in memory from 0 to N, where N is the # of declared members-1

/// THE ONDE DIMENSIONAL ARRAY IN C

/* A one dimension array consists of an amount of contiguous memory elements that can contain data of the same type (we've already established this)
--> The number of elements in the array are fixed and cannot be surpassed --> results in ignored assigned variables.

if you have declared the array with [3] elements and assign 4 values to the array, value number 4 will be ignored.

    Naming convention for an array follows the same as any variable:
    NOTE that the name of an array corresponds to the addres of the first data element in the array

    The element of the array is accessible by citing the name of the array and the element number within the index of the array []
    INDEX VALUES START AT 0

    The position of the element or the index value must always be represented by an INTEGER --> a double value will not work
    aka

    #define NUM 3

    arr[NUM] = arr[3]

    BUT

    #define NUM 3.0

    arr[NUM] != arr[3]

    because not integer

    */
    /// Array elements can also be represented using arithematic operations --> all below is viable

```

```
    temp[1]=5+3;                    temp[1]=8
    temp[1]=temp[2-1]+1;            temp[1]=9
    temp[1]=temp[1]+1;              temp[1]=10
    temp[0]=temp[1]/2;              temp[0]=5
    temp[0]=(temp[0]<=temp[1]);     temp[0]=TRUE=1 /// remember that TRUE = any int but 0 (FALSE = 0)

    /// The above memory assignments will exist
/// remember that after declaration of an array the memory is now reserved specifically for that array, unlike structures,
/// which before declaration hold no space in the memory, despite having been defined

/// Like in a structure, elements of an array can be assigned in one line using {}
/// example

double x[5]={0.0, 1.0, 2.0, 3.0, 4.0};

/// REMEMBER that if there are more inital values than elements in the array, a SYNTAX error will be generated

/// RUNING OVER ARRAY LIMITS

/* There is no mechanism in C which can prevent a program from running over the UPPEr and LOWER limits f an array during execution

READING or accessing an element outside of the array limits will:
    1. generally introduce errors in the results obtained
    2. may not cause the computer to crash or to become hung-up "silent mistake"
ASSIGNING a value to an element outside the array limits will:
    1. generally introduce a SYNTAX error from the program
    2. will often cause the computer to crash or to become hung up

    /// The above mistakes are commonly caused by a looping mechanism gone wrong
    your counter should always refer to being less than or equal to the N-1 of the array elements
    seeing as an array begins counting elements from the 0 value

*/

/// THE 2 DIMENSIONAL ARRAY -- THE MATRIX

/* 2D array aka a matrix is essentially an array of arrays
thing of elements in a row and column

        0   1   2   3
    0   []  []  []  []
    1   []  []  []  []
    2   []  []  []  []
    3   []  []  []  []

NOTE that both dimensions of the array begin with the 0 value

NOTE that much like a 1d array, organization of the elements in the 2D array is contiguous within memory AKA sequentially grouped
to cite an index in the array, we must cite both the [row][column] for which we wish to read

*/

/// DECLARATION OF A 2D ARRAY USING SINGLE LINE FORMAT

/// much like the above examples using {} as the assigning argument, we can assign values to the variables of a 2D array directly upon declaration

arr[3][3]={(1,1,1),(2,2,2),(3,3,3)};

/* the bracket in each case (),(),() represents each row, while the elements within each bracket ('','','','') represent the elements in
each column of each of the respective rows
```

```
311
312  */
313
314  /// IN ALL CASES, THE USE OF A SYMBOLIC CONSTANT DEFINED EARLY IN THE CODE #define NUM 3 IS CONSIDERED GOOD
PRACTICE
315  /// WHEN REFERRING TO THE DECLARATION OF THE ELEMENTS IN ALL ARRAYS, this way we can change array size,
simply by modifying the above
316  /// defined value
317
318  #define NUM 1
319  #define NUMM 2
320
321  arr[NUM][NUMM];
322
323  /// this way we only need to change NUM and NUMM up above to modify the elements of the ENTIRE code
324
325  /// TOPIC 3 MORE STANDARD C FUNCTIONS
326
327  /* There exists in C a STANDARD LIBRARY aka #include <stdio.h> which is available for general use
328
329  To avoid repeating functions, we familiarize ourselves with these C standard functions to eliminate tedious
tasks that hav
330  already been simplified for us
331
332  the Standard C library contains all of the following:
333  1. input/output functions
334  2. file manipulation
335  3. functions to manipulate character strings
336  4. other.... sorry for ambiguity
337
338  even printf and scanf are comprised within said library
339
340  */
341
342  /// The "getchar" and "putchar" Functions in Standard C
343
344  /* GETCHAR --> responsible for reading in a single character from the input stream (aka  keyboard)
345  recall that characters stored in the input stream have been typed in via the keyboard --> the prototype of
getchar is
346  found withing stdio.h and it reads as follows
347
348  int getchar(void);
349
350  what does the above mean?
351
352  /// this suggests that getchar will return an integer value at all times and will expect no argument within
brackets "(void)"
353
354  getchar will read a character value intputed via the input stream and will analyze it through the ASCII
code representing a particular
355  integer number according to the ASCII index, then it will return a number int with respect to that
character
356
357  /* PUTCHAR --> prototype for putchar is as follows
358
359  int putchar(int)
360
361  This means that the putchar function requires an integer argument to be placed within the brackets "(int)"
and will return
362  a respective character with respect to the ASCII code associated with the inputted number of type int.
363
364  PUTCHAR will return one of two if an argument is entered:
365  1. the echo --> the argument itself aka the process was successful
366  2. the EOF symbol if an error is encountered, for example a real value was inputted rather than an integer
367
368  */
```

```
/// WHAT IS THE EOF SUMBOL?

/* stands for End of File, which is not a standard character and does not hold an ASCII code counter part.
the numerical value associated with the EOF cannot be an integer value within the range of 0 to 127 (ASCII)

the symbol used to express the EOF is a constant that is operating system dependent --> HOW WOULD WE DETERMINE IT VIA STANDARD C FUNCTION?

ANSWER --> simply use the printf function

printf("%d", EOF); <---- this will show you the constant value

in general this value is represented by a -1

*/

/// LETS CONSIDER INPUT THROUGH THE KEYBOARD AKA INPUT STREAM

/* standard input is done through a keyboard in C, usually we use the "enter" key or the carriage return key to indicate that
a new line of text is being entered.

in most C functions the value being read is delimited by the carriage return key, but some inputs require more than a single line of text,
thus the return key can be problematic to delimiting the entered information

we therefore use the "ctrl z" to imply that the input stream is now over. this is displayed in one of the following ways;
1. ctrl+z at the same time
2. "cntrl-z" entered in the input stream
3. "ctrl z" entered i nthe input stream
4. "^z" entered in the input stream

this sequence of keys is interpreted as the EOF constant, therefore the end of an input sequence can also be delimited using the EOF
*/

/// GENERATION OF RANDOM NUMBERS

/* rANDOM NUMBERS ARE OFTEN USED TO SIMULATE A RANDOM PROCESS such as a coin toss or a dice roll

the standard functions rand() and srand() can be used to generate a PSEUDO-RANDOM sequence of numbers --> these are pseudo-random since the numbers eventually
start repeating themselves

the function rand() is of type int and it is contained between 0 and RAND_MAX where RAND_MAX is a symbolic constant associated
with the stdlib.h library --> usually 32767.

from one executio nto another, rand() will generate the same sequence of pseudo-random numbers to generate a different
sequence of numbers we use srand() which changes the ROOT of the seed of the random number generator

--> the function srand() does not return any value
--> ht efunciton srand() requires one argument which must be a positive integer of type unsigned int --> default seed is 1 (no idea what this means)

*/

/// EXAMPLES OF RANDOM INTEGERS GENERATING

int integer;
integer= rand() %8; --> 0<= integer <= 7
integer = 1+rand()%6; ---? 1<= integer<= 6
integer = rand()%51-25 ---> -25<= integer <= 25
```

The following genral formula can be used to generate random integers within the rand a<=int<=b

```
int a, b, integer;

integer= a+rand()%(b-a+1) /// MEMORIZE THIS !!!!
```

The epxression (b-a+1) specifies the width of the range and a specifies its beginning