Objective:

- For a refresher of most important Programming Fundamentals concepts; pointers, arrays
- This lab should enable us to understand the disadvantages of using global variables.
- Understanding the benefits of C structs.

Challenge-A *Age Calculator*

You should be quite familiar with the `struct Date`. You are required to implement following function related to `Date struct`.

```
Date getAge ( Date dob );
        //It receives the date of birth and returns the age.
        //For example, if dob has: 9-May-1995 as date of birth in its attributes then your function
        will return another Date object which represents the age of person.
```

Sample Run:

```
Date d = {9,5,1995};
Date age = getAge (d);
cout << age.year << " year, " << age.month << " months, " << age.day << " days.";//prints 26
years, 5 months, and 4 days.//System Date when this code was executed: 13-10-2021
```

But you may only implement this function if you know the way to find system date (the current date on your computer).
When you will execute the following code, It will store the system date in your Date object named below currentDate. We don't need to know much about the code given below just paste the code snippet in your function to find the system date. In future, you will know each and every bit of it hopefully.

Code to get System Date	Short Explanation
`Date currentDate;`	Date object in which system date will be stored.
`time_t t = time(0);`	The C library function ***time_t time(time_t *seconds)*** returns the time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds.
`tm curTime = * localtime(&t);`	localtime function converts given time since epoch as std::time_t value into calendar time, expressed in local time. tm structure holding a calendar date and time broken down into its components.

`currentDate.day = curTime.tm_mday;`
`currentDate.month = curTime.tm_mon + 1;`
`currentDate.year = curTime.tm_year + 1900;`
	Storing the system date into your Date object

In order for the above code to work, you need to include ctime library.

```
#include<ctime>
```

And also have to add following on top of your code (even before including any header file).

```
#define _CRT_SECURE_NO_WARNINGS
```

To verify your function output, you may seek help from following link.
https://www.calculator.net/age-calculator.html

Challenge-B *Global Variables and Arguments Passing*

In this challenge, our purpose is to build a library with some basic operations of a Set (Math).
We want to build sets library exhibiting some basic Set operations while using global identifiers and functions vs C language structs.
The library should be able to support the following operations: can be extended further by defining operations like set-difference, set-union, Cartesian product etc.

A.1. Should support to insert an element in a set.
A.2. Should support to remove a given element in a set.
A.3. Intersection of two sets.
A.4. ***But*** all the tasks given depends on two things heavily:

- How to create a structure for set
- How to take input

Note: We shall review and implement multiple versions of the above library based on different design/structure.

➔ *A.1: First version of the above library will be as follows:*

Global Data:

Global Variables:

```
int * data;         //used to hold the set/array of integers
int capacity;       //represents the capacity/size of set
int noOfElements;   //number of elements in the set
```

Global Functions:

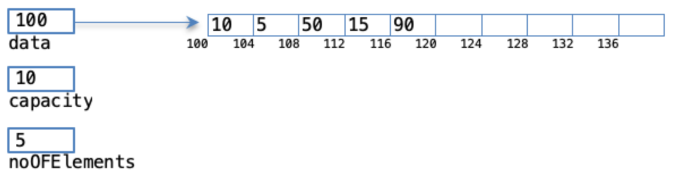
```
void createSet ( int cap)          //initialize the data array according to capacity
void insertElement ( int elem )    //insert given value in set/data-array
void removeElement ( int elem )    //remove given value in set if found
void displaySet()                  //display the set on console
void freeSet()                     //free the array/set created in createSet
```

Sample Run:

```
int * data;
int capacity;
int noOfElements;
int main()
{
        createSet(10);
        insertElement(10);
        insertElement(5);
        insertElement(50);
        insertElement(5);
        insertElement(15);
        insertElement(100);
        insertElement(90);
        removeElement(100);
        displaySet();
        freeSet();
        return 0;
}
```

The code on left side will create following memory layout in RAM.

And *displaySet* will print 10, 5, 50, 15, 90 on console.



Stop Here! Do not read below. Complete the above operations used in sample run and test them thoroughly.

Now *we start working on intersection* ***But*** *how do we get another set so that two sets intersection can be achieved?*
Certainly, we need to make the same global variables that we created for the first set.
We can name them as: `data2`, `capacity2`, `noOfElements2`
But the next question will be that which function will be responsible to create the second set or take input because the '`createSet`*', and '*`insertElement`*', and '*`removeElement`*' are designed to work only with '*`data`*', '*`capacity`*', '*`noOfElements`*'.*

→So, we need to make/code three more functions like creating/inserting/removing for the second set. ☹ that kills one of the main purpose of functions.
→So, should we do that or look out for a better option?

→ I hope you agree that we should go for a better alternative i.e., to write functions which are more generic and may work on any identifier. How do we do that? By passing arguments to functions.
*For this:Stop working on **A.1** and start your work on next version **A.2**.*

➔ A.2: *Second version of the above library will be as follows:*

Local variables and parameter passing

We are doing a very big blunder by using global data and not defining appropriate prototypes of the function, as a result: every function strongly binds with some specific global data.
→We should make function prototypes in such a way that they should be able to work with any given data/set as follows. It will make our functions generic and will also allow us to work without any global data.

Global Functions

```
void createSet ( int * * data, int capacity )
        //initialize the data array according to capacity
        Think yourself that why data is received as a double pointer. Sample run given
        below may also help you to get the reason behind it.

bool insertElement ( int * data, int capacity, int * noe, int elem )
        //insert given elem in set/data-array keeping in view the noe(noOfElements) and
        capacity of the set.
        Think yourself that why noOfElement is received by referenced but capacity by
        value.
        It returns true if element is added successfully otherwise false. Do think that
        under which cases element will not be added.

bool removeElement ( int * data, int capacity, int * noe, int elem )
        //removes the given elem in set/data-array keeping in view the noe(noOfElements) of
        the set.
        It returns true if element is removed otherwise false.

void displaySet(int * data, int noe)
        //display the set on console

void freeSet(int * * data)
        //free the array/set created in createSet and assign nullptr to it.

int * intersection(int * set1, int set1NOE, int * set2, int set2NOE, int &
intersectionSetNOE);
        //it creates new set which is intersection of set1 and set2 and return the
        intersection/set array and also saves the no of elements of the resultant set in
        intersectionSetNOE.
```

Sample Run:

int main()	**Console Output**

```
int main()
{
        int * set1;
        int cap1=10;
        int NOE1=0;
        createSet(&set1, cap1);
        insertElement(set1, cap1, &NOE1, 10);
        insertElement(set1, cap1, &NOE1, 5);
        insertElement(set1, cap1, &NOE1, 50);
        insertElement(set1, cap1, &NOE1, 5);
        insertElement(set1, cap1, &NOE1, 15);
        insertElement(set1, cap1, &NOE1, 100);
        insertElement(set1, cap1, &NOE1, 90);
        removeElement(set1, cap1, &NOE1, 100);
        cout<<"\nSet 1 members: = ";
        displaySet(set1, NOE1);

        int * set2;
```
	```
	Set 1 members: = {10, 5, 50, 15, 90}
	Set 2 members: = {5, 45, 10, 15, 25,
	17, 30}
	Intersection of Set 1 and 2: = {10, 5,
	15}
	```

```
        int cap2=8;
        int NOE2=0;
        createSet(&set2, cap2);
        insertElement(set2, cap2, &NOE2, 5);
        insertElement(set2, cap2, &NOE2, 45);
        insertElement(set2, cap2, &NOE2, 10);
        insertElement(set2, cap2, &NOE2, 15);
        insertElement(set2, cap2, &NOE2, 25);
        insertElement(set2, cap2, &NOE2, 17);
        insertElement(set2, cap2, &NOE2, 30);
        cout<<"\nSet 2 members: = ";
        displaySet(set2, NOE2);

        int * set3;
        int cap3=0;
        int NOE3=0;
        set3 = intersection(set1,NOE1, set2,
NOE2, NOE3);
        cap3 = NOE3;
        cout<<"\nIntersection of Set 1 and 2: =
";
        displaySet(set3, NOE3);

        freeSet(&set1);
        freeSet(&set2);
        freeSet(&set3);

        return 0;
}
```

*Your code should output in the same format as depicted in the sample run.

As we know that three identifiers together represent one set i.e. (`int * set, int NOE, int capacity`):

- We may have many sets in our code with which we may be dealing with and while coding, we need to keep in mind that which three identifiers belongs to which set.
- A careful naming convention is needed but it is still prone to logical errors. For example , we may call a function like *insertElement(set2, cap1, NOE2, 30);*
- Dealing with lot of parameters is not easy, it becomes more awkward while returning a set as in case of `intersection` function. It makes the function prototype less readable.
- In second version sample run: we preferred to define local identifiers for set instead of global because it restricts the set functions to mistakenly update/access any set identifier.

The struct feature in C language give us an elegant way to handle the issues mentioned above. It allows the programmer to define user define types (UDT). Makes the code more readable and maintainable.

➔ A.3: *Third version of the set library will be as follows:*

C struct

In this version: we shall treat set as a noun in our code which consists of three properties: (`int * data, int noOfElements, int capacity`).

```
struct Set
{
        int * data;
        int noOfElements;
        int capacity;
};
```

Global Functions

```
void createSet ( Set *, int capacity )
        //initialize the data array according to capacity

bool insertElement ( Set * , int elem )
```

```
        //insert given elem in set/data-array keeping in view the noe(noOfElements) and
        capacity of the set.
        It returns true if element is added successfully otherwise.

bool removeElement ( Set *, int elem )
        //removes the given elem in set/data-array keeping in view the noe(noOfElements) of
        the set.
        It returns true if element is removed otherwise false.

void displaySet(Set)
        //display the set on console

void freeSet(Set *)
        //free the array/set created in createSet assign nullptr to it.

Set intersection(Set s1, Set s2);
        //it creates new set which is intersection of set s1 and s2 and returns the
        intersection/set.
```

int main()	**Console Output**

```
int main()
{
        Set s1;
        createSet(&s1, 10);
        insertElement(&s1, 10);
        insertElement(&s1, 5);
        insertElement(&s1, 50);
        insertElement(&s1, 5);
        insertElement(&s1, 15);
        insertElement(&s1, 100);
        insertElement(&s1, 90);
        removeElement(&s1, 100);
        cout<<"\nSet 1 members: = ";
        displaySet(s1);

        Set s2;
        createSet(&s2, 8);
        insertElement(&s2, 5);
        insertElement(&s2, 45);
        insertElement(&s2, 10);
        insertElement(&s2, 15);
        insertElement(&s2, 25);
        insertElement(&s2, 17);
        insertElement(&s2, 30);
        cout<<"\nSet 2 members: = ";
        displaySet(s2);

        Set s3;
        s3 = intersection(s1, s2);
        cout<<"\nIntersection of Set 1 and 2: =
";
        displaySet(s3);

        freeSet(&s1);
        freeSet(&s2);
        freeSet(&s3);

        return 0;
}
```
	Same output as in 2nd version but makes the programmer life easier.

C struct help us to introduce a level of abstraction (will read about it soon) in our code by combining the related items together which gives a unique meaningful concept. Hence make the code more readable and maintainable.
Logic was also not in one place for some functions in 2nd version but with structs its quite composed. For example, observe your *createSet,* and *intersection* implementation for 2nd and 3rd version.

Collaboration/Cheating/Sniffing

- Traces of any sort of collaboration/cheating/sniffing will be served with grade 'F'.
- Copy/Pasting is not the only cheating instance. Discussion on the questions with anyone (excluding me) even for the sake of understanding is considered as cheating.
- When the Lab deadline is over, you will be asked to participate in a Poll on Piazza. The Poll will remain active for 6 hours after the Lab deadline is over.

What to Submit?

Challenge-A Solution
Coding for Challenge-B Version-2 Solution
Coding for Challenge-B Version-3 Solution

How to Submit?

- You will prepare/paste all your solutions in one MS Office document and then convert it into a pdf document.
- Last but not the least, make sure that after pasting your code in word, code indentation is not disturbed.
  - Set the font of your code in word file as verdana or Menlo with font size 9.
- Once you are done with solution and converted it into pdf. Name your pdf file as follows:
  - Lab2_RollNo → Example: Lab2_BSEF20M001
- Once your pdf is prepared, you will submit the pdf file by posting privately on piazza (tag the post with lab2). I hope you know, how to attach a file in the Piazza post.
- Remember: you have already been given enough time for submission. No excuses will be accepted at all for late submissions.

Sample Layout for Pdf/Word File:

RollNo: BSEF20M001
Name: Muhammad Abdullah

Challenge-A
Challenge-A solution come here.

Code for Challenge-B Version-2:

Here comes your code for Challenge-B version-2.

Code for Challenge-B Version-3:

Here comes your code for Challenge-B version-3.

"Some of the best programming is done on paper, really. Putting it into the computer is just a minor detail."
-- Max Kanat-Alexander --