DS Assignment I

1. Suppose an integer array by name *list* is declared of size *N* (ex:

```
#define N 10
int list[N];
```

). Write C statements to achieve the following:

1. Set a pointer by name *first* and *last* to point to the first element and last element of the list respectively.

```
int *first = list;
int *last = list + N - 1;
```

2. Give an expression to compute the middle pointer in terms of *N* and *first* pointer.

```
int *middle = first + N/2;
```

3. Suppose the array is declared of size 100, but the actual size is less than 100. Further assume that the *first* and the *last* pointers are available. Give an expression for actual size in terms of first and last pointers.

```
// Suppose 'first' is pointing to first element, and 'last' is
pointing to last element.
int size = (int)(last - first);
```

4. The equivalent expression for *p++

```
// Note. ++*p is equivalent to ++(*p). But *p++ increments the
pointer, not the value pointed by p. ∴ *p++ is equivalent to p = p + 1

p = p + 1;
```

5. Suppose the following C statements are to be executed. Write the equivalent statements for last two C statements in terms of index notation.

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int *ip1 = &a[3];
int *ip2;
ip2 = ip1 +1;
*(ip1+1) = 50;
*(ip1-2) = 4;
```

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int *ip1 = &a[3];       // ip1 pointing to a[3] ie '4'
int *ip2;
ip2 = ip1 + 1;          // ip2 pointing to a[4] ie '5'
*(ip1+1) = 50;          // set the value of a[3+1] or a[4] to '50'
*(ip1-2) = 4;           // set the value of a[3-2] or a[1] to '4'

// hence the equivalent expressions are...

a[4] = 50;
a[1] = 4;
```

2. Demonstrate 3 ways for defining constants in C language (Hint: #define, const, enumerated data types)

1. C allows you to declare *constants*. When you declare a constant it is a bit like a variable declaration except the value cannot be changed. The `const` keyword is to declare a constant.

```
int const a = 1;
const int b = 2;
```

2. The preprocessor `#define` is another more flexible method to define constants in a program.

```
#define TRUE 1
#define FALSE 0
#define CRAP_VOLUME 9001
```

3. `enum` is the abbreviation for ENUMERATE, and we can use this keyword to declare and initialize a sequence of integer constants. For e.g.

```
enum COLORS {
      COLOR_RED = 1,
      COLOR_GREEN = 2,
      COLOR_BLUE = 9,
};
```

Here, `COLORS` is the name given to the set of constants - the name is optional. Now, if you don't assign a value to a constant, the default value for the first one in the list is 0.

3A. Explain the realloc with syntax. Why is it inefficient?

The 'realloc' C library function changes the size of the memory block pointed to by the pointer passed to the function to a new size passed into the function.

```
void *realloc(void *ptr, int size)
```

The realloc function has to copy the memory block to a new location (whose address is returned by the function) if the memory resizing operation cannot be done in one place, hence temporarily acquiring huge amounts of memory, making it less efficient to the malloc/free pair.

3B. Write a program in c to allocate memory dynamically as follows:

1. Write a function allocate_mem to perform dynamic memory allocation for a single integer and assign 10 to that integer in the function (default value)

2. Write a main function to invoke this function and print the value assigned by the function. (Hint: Use pointers to pointers concept)

```
#include <stdio.h>
#include <stdlib.h>

void allocate_mem (int **n) {
      *n = (int *)malloc(sizeof(int *));
      **n = 10;
}

int main(int argc, const char * argv[]) {
      int *pn;
      int **ppn;
      ppn = &pn;
      allocate_mem(ppn);
      printf("%d", **ppn);  // Prints 10.
      return 0;
}
```

4A. With a suitable example explain how pointers are useful in achieving inter function communication? What is the importance of pointers?

A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before you can use it to store any variable address.

When we pass an argument to a function, it's passed by it's value therefore any changes made to the value are only valid inside the scope of the function in which it's passed. However when we pass a pointer into a function, an address to the memory location of an variable is passed. Any changes made to the address inside the function is reflected everywhere in the program.

e.g.

```
void swap (int a, int b) {
      int c = a;
      a = b;
      b = c;
}

void swap_ptr (int *a, int *b) {
      int c = *a;
      *a = *b;
      *b = c;
}

int main(int argc, const char * argv[]) {
      int a = 6, b = 9;
      int *p = &a, *q = &b;

      printf("%d\t%d\n", a, b); // Prints '6  9'

      swap(a, b);

      printf("%d\t%d\n", a, b); // Still prints '6  9'

      swap_ptr(p, q);

      printf("%d\t%d\n", a, b); // Now prints '9  6'
}
```

4B. Write a function that reverses the elements of an array in place. In other words, the last element must become the first; the second from last must become the second and so, on. The function must accept one pointer value and size of the array. The function must return void.

```
.
.
.
void reverse (int *arr, int n) {
      int *start = arr, i;
      for (i = n-1; start < arr + n/2; ++start, --i) {
            int c = *start;
            *start = *(arr + i);
            *(arr + i) = c;
      }
}
.
.
.
```

5. Create a structure STUDENT where STUDENT creates a variable of structures:

1. **DOB {day (use pointer), month, year} ,**
2. **STU_INFO {reg_no, name(use pointer), address},**
3. **COLLEGE {college_name(use pointer), university_name}**

whereas structure types from 1 to 3 are declared outside the STUDENT independently.

Write a C program to

1. **Initialize all structure (1 to 3) member variables through STUDENT**
2. **Read the information of the student using structure pointer variable and display the read information.**
3. **How do you read and write members variables of DOB type if the pointer variable is created for DOB inside STUDENT and STUDENT variable is also a pointer variable.**

```
#include <stdio.h>

typedef struct DOB {
      int *day;
      int month;
      int year;
} DOB_t;

typedef DOB_t* DOB_p_t;          // DOB_t pointer type
```

```c
typedef struct STU_INFO {
	long reg_no;
	char *name;
	char *address;
} STU_INFO_t;

typedef struct COLLEGE {
	char *college_name;
	char *university_name;
} COLLEGE_t;

typedef struct STUDENT {
	DOB_p_t dob;
	STU_INFO_t sinfo;
	COLLEGE_t college;
} STUDENT_t;

typedef STUDENT_t* STUDENT_p_t; // STUDENT_t pointer type

int main(int argc, const char * argv[]) {
	STUDENT_p_t pstudent;

	// Initializing...
	// SUPER FRAKKING IMPORTANT NOTE...
	// ALLOCATE THE MEMORY TO EVERY GORRAM POINTER INSIDE THE
STRUCTURE

	pstudent = (STUDENT_p_t)malloc(sizeof(STUDENT_t));
	pstudent->sinfo.name =  (char *)malloc(sizeof(char *));
	pstudent->sinfo.address = (char *)malloc(sizeof(char *));
	pstudent->dob = (DOB_p_t)malloc(sizeof(DOB_p_t));
	pstudent->dob->day = (int *)malloc(sizeof(int *));
	pstudent->college.college_name = (char *)malloc(sizeof(char
*));
	pstudent->college.university_name = (char
*)malloc(sizeof(char *));

	// Reading values...
	printf("Enter student name: ");
	scanf("%s", pstudent->sinfo.name);
	printf("Enter student reg number: ");
	scanf("%ld", &pstudent->sinfo.reg_no);
	printf("Enter student address: ");
	scanf("%s", pstudent->sinfo.address);

	printf("Enter student DOB (dd/mm/yyyy): ");
	scanf("%d%d%d", pstudent->dob->day,
&(pstudent->dob->month), &(pstudent->dob->year));
```

```c
        printf("Enter student college name: ");
        scanf("%s", pstudent->college.college_name);
        printf("Enter student university name: ");
        scanf("%s", pstudent->college.university_name);

        // Printing the stuff
        printf("\n\nNAME\t\tREG NO\t\tADDRESS\t\tDOB (dd/mm/yyyy)\t
    \tCOLLEGE\t\tUNIVERSITY\t\t");
        printf("\n\n%s\t\t%ld\t\t%s\t\t%d/%d/%d\t\t%s\t\t%s\t\t",
                pstudent->sinfo.name,
                pstudent->sinfo.reg_no,
                pstudent->sinfo.address,
                *(pstudent->dob->day),
                pstudent->dob->month,
                pstudent->dob->year,
                pstudent->college.college_name,
                pstudent->college.university_name);

        return 0;

    }
```

Output for reference:

```
Enter student name: Derp
Enter student reg number: 9669
Enter student address: Derpsville
Enter student DOB (dd/mm/yyyy): 4 5 1994
Enter student college name: Derpston
Enter student university name: HerpDerp

NAME  REG NO      ADDRESS     DOB (dd/mm/yyyy)  COLLEGE     UNIVERSITY

Derp  9669        Derpsville  4/5/1994          Derpston    HerpDerp
```