BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJ.)

Second Semester 2019-20 CS F111 Computer Programming

LABORATORY SESSION #13

(Dynamic Memory Allocation; Linked Lists)

1. Write a program to store only the non-negative elements of an integer array into a dynamically allocated array, and then compute the square root of each of them and print it out.

2. Modern C standards allow the use of variable-length arrays (VLAs) whose size can be specified during runtime by the user. VLAs are different from dynamically allocated arrays. In the following example program, `str1` is dynamically allocated, whereas `str2` is a VLA. Study this code carefully:

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

char * f1(int size) {
char *str1;
str1 = malloc(size);
scanf("%s",str1);
return str1;
}

char * f2(int size) {
char str2[size];
scanf("%s",str2);
return str2;
}

int main() {
int num, i;
char *p, *q;
printf("Word length? ");
scanf("%d",&num);
p=f1(num);
puts(p);
q=f2(num);
puts(q);
}
```

Both the functions `f1()` and `f2()` are intended to obtain string input from the user, which would be returned to `main()` and then printed. However, when the program is executed, it is found that the input string is printed as expected after `f1()` is called, whereas we get a garbled output / segmentation fault after `f2()` is called (instead of the string).

a. How will you explain the difference in output observed?

b. How are VLAs different from dynamically allocated arrays in terms of memory allocation? Explain.

3. Declare an array of **m** pointers, each pointing to a dynamically allocated array of **n** integers **(m** and **n** are both user inputs). Then, take **m x n** inputs for the matrix elements. Next, display the entire matrix on screen in a neatly formatted manner. If you wish, you can use the data file `/home/share/dyn_arrays.txt` for taking inputs (first line contains m and n, and the remaining lines have the values for the **m x n** matrix).

4. This program involves taking input for **m** rows of a matrix, but each row having a varying number of integer elements. Take a look at `/home/share/ raggedarrays.txt` (and copy to your current directory) to get an idea of this matrix with varying number of columns for each row. The first row contains the number of rows. The first element of each row contains the number of columns of that particular row, followed by the actual elements themselves.

Now declare a suitable array of pointers, dynamically allocate memory for each row, read the elements (including the number of elements as the first entry of each row), and then display the entire matrix on the screen. Verify to make sure that your output is the same as the matrix entries given in the data file you copied (`/home/share/ raggedarrays.txt`).

PRACTICE QUESTION

5. Consider the following type definition:

```
typedef struct student STUDENT;
typedef struct student
{
   char name[MAX];   // MAX is #defined to 50
   int count;
   int length;
   STUDENT *next;
}STUDENT;
```

A file named *roster.txt* present in the folder **/home/share** which contains the first name of the all the students, each row containing a name; the first row however contains the number of students. Write a program to read the data from the file and create a linked list of unique names, by implementing the following functions:

a. Define a function **`STUDENT * addName(STUDENT * list, char *name);`** This function should create a new list if **list** is empty. Otherwise it should check for **name** in the linked list and if present, should increment the **count** accordingly. If **name** does not already exist in the list, the function should dynamically allocate memory and add a new element with name **name** at the end of the list, populate the individual members of the new structure element and return the new list.

b. Implement the function **`void printUniqueNames(STUDENT *);`** that prints the unique names in the list along with the frequency of occurrence of each name. Use output redirection to redirect the output to the file *unique.txt*. How many unique names has your function been able to find? Compare your output *unique.txt* with the contents in *names.txt* (in/home/share) to verify your answer.

c. Implement the function **`int findMostFrequent(STUDENT *);`** which prints the most frequently occurring name in the list and returns its frequency.

d. To delete one student from the list, implement the following function:
`STUDENT * deleteSingle(STUDENT * list, char *name);`
This function should return the original list, if **name** is not present, decrement the **count** by one if **count** >1, or delete the node if **name** is present and **count** is equal to 1.

e. Modify the part (a) by implementing the following function:

`STUDENT * addSortOrder(STUDENT * list, char *name);` While adding a name to the list, this function will insert the name at the right spot so that the list is always in lexicographic ordering of the names.

f. Modify the structure definition by making the first name dynamically allocated and stored. In other words, the first field will be **`char *nm;`** (and not an array)