CSE225L: Data Structures and Algorithm Lab

Lab 01: Dynamic Memory Allocation

North South University

When we declare a variable in our program, memory is allocated for that variable in the memory before the program execution starts. In case of dynamic allocation, memory allocation is done in runtime, that is after program execution starts.

To allocate memory in runtime, we use the operator "**new**".

General Syntax:

```
new datatype;
```

Example:

```
new char;        //this allocates memory for a character variable
                 //in the memory
```

To be able to access this allocated memory later in the program, we need the concept of pointers. The operator **new** allocates the memory for the datatype that is specified and returns the location of the allocated memory. As we know from the concept of pointers, pointer variables can be used to store the location in the memory.

For example:

```
char *ptr = new char;

/* this allocates memory for a character variable in the memory
and then stores the location of the allocated memory in a
character pointer variable ptr */
```

To assign or modify or access the value stored in the allocated memory location, we again look back to our concept of pointers.

Extending previous example:

```
*ptr = 'X';            // assigns value 'X' to the allocated
                       // memory

cout<<*ptr<<endl;      //prints X

cin>>*ptr;             //takes a character as input and stores in
                       //the allocated memory
```

Whenever dynamic allocation is used, that memory should be deallocated as well, because dynamically allocated memories are not automatically deallocated after the execution of a program is finished. Thus, if not deallocated manually, this creates memory leaks.

To deallocate, delete operator is used.

Extending previous example:

```
delete ptr;      // this deallocates the memory pointed by the
                 // pointer variable ptr.
```

We can also allocate memory for arrays dynamically. For example:

```
char *ptr = new char[5];
```

This example creates a character array of size 5. We can access and modify values of this array like any other arrays.

```
ptr[2] = 'A';          // this assigns value 'A' to the ptr array
                       // at index 2
```

To deallocate, we need to use delete operator again.

```
delete [] ptr;
```

We can also use dynamic allocation to create 2d arrays. Following is an example that creates 2d character array with size of 5 × 3.

```
char **arr = new char*[5];  // this allocates memory for an array
                            // of character pointers of size 5
for (int i = 0; i<5; i++)
     arr[i] = new char [3]; // this creates a character array
                            // of size 3

/* After the loop is finished, memory for a 2d character array
of size 5 × 3 is created. */
```

This array can be used like any statically allocated arrays as well.

```
arr[i][j] = 'A';
```

Like before we need to be able to deallocate this array as well.

```
for (int i=0; i<5;i++)
     delete [] arr[i];

delete [] arr;
```

Task:

1. Dynamically allocate an integer array of user specified size (the user gives the size of the array as input). Assign values to the array elements by taking user inputs and then print the values. Finally, de-allocate the array.
2. Dynamically allocate a two-dimensional integer array. Number of row and column should be taken as user input. Take integers as input to fill up the array with integer values. Print the values and finally deallocate the array.
3. Dynamically allocate a two dimensional integer array. The number of rows and columns are going to be provided by the user as input. However, in this task, all of the rows are not the same size (the array is uneven). The user will specify how many elements the individual rows will have. Assign values to the array elements by taking user inputs and then print the values. Finally, de-allocate the array.