1. Given the following declaration and definition
   ```
   char a[6] = {'z', 'x', 'm', 's', 'e', 'h'};
   ```
   and using only pointer notation, write a loop to rotate all values in `a` to the right by one element.

2. Write a function `IndexArray(n)` that returns a pointer to dynamically allocated integer array with `n` elements, each of which is initialized to its own index. For example, assuming that `ip` is declared as
   ```
   int * ip;
   ```
   the statement
   ```
   ip = IndexArray(8);
   ```
   should produce the following memory configuration :

   ![Memory Configuration Diagram]

3. Write a simple C++ program that uses a dynamic array to hold a list of pop song title (at least six) entered from the keyboard. If you need a space in the name, use an underscore like _this instead.

4. Write a program that prompts the user for a string that contains. Store the string in a dynamic allocated array. Write a function that prints a string in reverse, and pass the user’s string to this program. Use pointer syntax throughout.

5. Write a function named `subem()` with two call-by-reference integer parameters. The function is to sub 32 to the first parameter and 16 to the second parameter. Test the function by calling it with the values of 77 and 34

6. Write a function `flip_sign()` that expects a pointer to an array of integers and an integer that gives the array’s size. The function `flip_sign()` multiplies each number in the array by -1. Use pointer syntax throughout.

7. Write a function that computes the median of a set of integers (1~99). If the integers are stored in an array that is then sorted, the median occurs at the array’s midpoint. If there are an even number of integers, we average the two middle numbers to obtain the median. For example, the median of `{88, 34, 9, 23, -2}` is 23. **Use selection sort.** Use pointer notation throughout.
   Example :
8. Write a program that randomly generated (1–99) integer (each \( n \)) as inserted count computes the mean \( m \), the difference \( n-m \). Use pointer syntax throughout.

9. Write a function that add two polynomials of at most degree \( n \). Test your function with a simple main program that \( n \) is inserted and \( f, g \) is randomly generate. Use pointer syntax throughout.

\[
\text{/* } f = g + h; \text{ } n \text{ is the max degree of } f, g, h*/
\]

\[
\text{void add(double* } f, \text{ double* } g, \text{ double* } h, \text{ int } n)\]

Example : \( n \) is 5

<table>
<thead>
<tr>
<th>Insert generation number: 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f = 42 + 98x^1 + 63x^2 + 94x^3 + 75x^4 + 63x^5 )</td>
</tr>
<tr>
<td>( g = 54 + 68x^1 + 83x^2 + 55x^3 + 12x^4 + 38x^5 )</td>
</tr>
<tr>
<td>( h = 96 + 166x^1 + 146x^2 + 149x^3 + 47x^4 + 93x^5 )</td>
</tr>
</tbody>
</table>

10. Design a function prototype that would allow a single function to find and return simultaneously both the lowest and highest values in an array of type double. Use pointer syntax throughout. Implement and test your function as shown in the following sample run.

Enter the elements of the arrays, one per line.
Use -1 to signal the end of the list.

? 67
? 78
? 75
? 70
? 71
? 80
? 69
? 86
? 54
? 76
? 78
? 70
? 77
? -1

The range of values is 54–86
11. Create and test a function with the following prototype:

```c
char* to_upper (const char* array);
```

where the function `to_upper()` converts any lower-case letters found in the array and returns a pointer to a string that contains upper-case characters. The advantage of this function is that the contents of the original array need not be changed.

12. Write a function `GetDate()` with the prototype

```c
void GetDate(int *dp, int *mp, int *yp);
```

that reads in a date from the user in the form `dd-mmm-yyyy` where `dd` is a one- or two-digit day, `mmm` is a three-letter abbreviation for a month, and `yyyy` is a four-digit year. Your implementation should take apart the components of the date and give them back to the caller in numeric form by assigning values to the three arguments, each of which is passed by reference. Test your function with a simple main program that is capable of generating this sample run:

```
Enter a date as dd-mon-yyyy: 7-Apr-2008
Day = 7
Month = 4
Year = 2008
```

13. Write a program that reads simple data declarations and responds with the amount of memory that would be allocated to that variable. For example, the following sample run shows the output of the program for two possible input lines:

```
Enter variable declarations, ending with a blank line.
?int x, y;
x requires 4 bytes
y requires 4 bytes
?char c, *ptr, array[80];
c requires 1 byte
ptr requires 4 bytes
array requires 80 bytes
?int a
A semicolon marking the end of the line.
?
```

Enter input line should consist of

A. A type name, which must be one of the following: `char`, `int`, `short`, `long`, `float`, or `double`.
B. One or more individual declaration specifications separated by commas.
C. A semicolon marking the end of the line.

Your program should exit if it reads a blank input line.

The individual declaration specifications must consist of a variable name, which can be modified in either or both of the following way:
14. Write a function `find()` with two parameters: `str` and `substr`. The parameters `str` and `substr` are pointer to char. The function `find` returns the address of the first occurrence of the string pointed to by `substr` in the string pointed to by `str`. If the string pointed to by `substr` does not occur in the string pointed to by `str` `find` return `NULL`. Examples: If `str` points to “Let’s talk about the black bird.” and `substr` points to “black”, the `find` returns a pointer to the character b (in black) in the string pointed to by `str`.

If `str` points to “Let’s talk about the black bird.” and `substr` points to “blue”, then `find` returns the value `NULL`.

Write a `main()` function that invokes `find` several times to demonstrate that it is working properly.

Example:

```
search string : This chapter covers the process of realigning those streams at the receiver.
search word : process
process of realigning those streams at the receiver.
```

15. Write your own version of `strncat()`. Use only pointer syntax. Write a `main()` function that invokes `find` several times to demonstrate that it is working properly.