ET-580 – Pointers & Dynamic Arrays - Practice

Pointers

1. Pointer Basics. Implement the following:

   a. Create an integer variable *i* and assign it the value 5.
   b. Create two integer pointer variables, *p* and *q*.
   c. Point *p* and *q* to *i*.
   d. Dereference *p* to update the value of *i* to 10.
   e. Print *i*, **p*, and **q* to verify they have the same value.

Output Example

```
i: 10
*p: 10
*q: 10
```

2. Pointer functions. Implement the following:

   a. Write a function *create* which returns a pointer to an int on the heap.
   b. Write a void function *update* which accepts an integer pointer *p* and increments the value of the variable *p* points to.
   c. Create a pointer *q* in main.
   d. Initialize *q* with the *create* function, print the value of q.
   e. Modify *q* with the *update* function, print the value of q.

Output Example

```
5
6
```

3. Swap. Implement the following.

   a. Write a function *swapVar* that accepts two pointers and swap what they point to. For example, if a->b and c->d, then a->d and c->b.
   b. Write a function *swapVal* that accepts two pointers and swaps the values of the variables they point to.
      For example, if a->b, c->d, b==1 and d==2, then b==2 and d==1.
   c. Create two pointers to test the functions and print the results.

Output Example

```
1 2
2 1
1 2
```

Dynamic Arrays and Pointer Arithmetic

3. Pointer arithmetic. Implement the following:

   a. Implement a *print* function with array and size parameters.
      This function should print the array using pointer arithmetic.
   b. Overload the *print* function to accept array, size and index parameters.
      If the index is between 1 and size-2 inclusive:
      Use pointer arithmetic to print:
         1. the value at index
         2. the value previous to the index
         3. the value after the index
   c. Implement a dynamic array with 10 values and test both functions.

Output Example

```
10 20 30 40 50 60 70 80 90 100
30 40 50 (when index is 3)
```

5. Dynamic N-Queens Map. Implement the following:

   a. Implement a dynamic array named *queens* with values:
      {3,6,2,7,1,4,0,5}

      This array represents a n-queens map where the index/value of each position in the array is the row/column coordinate of a queen.

   b. Implement an 8x8 dynamic array of arrays named *board*.

   c. Implement a function named *translate*:
      a. accept the arrays *queens*, *board* and *size* as parameters
      b. store a *1* in *board* for every Q coordinate in *queens*, *0* otherwise
         example: at *0,3* in board store a *1*
                  at *1,6* in board store a *1*
      c. all code must use pointer arithmetic instead of indexing.

   d. Implement a *print* function which accepts the array *board* and array *size* then prints an n-Queens map ('Q' when value is 1, '.' when value is 0). This function must use pointer arithmetic instead of indexing.

Output Example

```
. . . Q . . . .
. . . . . . Q .
. . Q . . . . .
. . . . . . . Q
. Q . . . . . .
. . . . Q . . .
Q . . . . . . .
. . . . . Q . .
```

6. Dynamic Array Copy. Implement the following:

   a. Initialize a dynamic array *a* of *size* 10 with values 10 through 100.

   b. Implement a *print* function for an array using pointer arithmetic.

   c. Implement a *copy* function:
      1. This function must use pointer arithmetic (no indexing).
      2. Parameters: *array*, *oldSize*, *newSize*
      3. Returns: a new array of *newSize* that is a copy of the original
      4. The new array capacity is equal to the *newSize* parameter.
         If the new array is smaller, only store the first *newSize* values.
         If the new array is larger, remaining space should be zeros.

   d. Print the original array.

   e. Make a larger copy of the original array and print it.

   f. Make a smaller copy of the original array and print it.

Output Example (original, larger copy, smaller copy)

```
10 20 30 40 50 60 70 80 90 100
10 20 30 40 50 60 70 80 90 100 0 0 0 0 0
10 20 30 40 50
```