Programming Practice: Stacks and Queues

1. A stack is a container that elements are inserted and removed in the *last in first out* (*or first in last out*) order. A stack has the following operations:
   (1) is_empty: check whether the stack is empty,
   (2) top: get the element at the top of the stack,
   (3) push: place an element to the top of the stack, and
   (4) pop: get the top element and remove the top element from the stack.

   Write a C project to define a data type representing stacks using fixed-size array of integers and define the following stack operations:

```
#define max 100
// Maximum 100 element.
// Type definition of stacks using a fixed-size array.
typedef struct {
      int elem[max]; // Stack container.
      int ptr; // Top pointer.
   } Stack;
void initialStack(Stack *); // Initialize a stack.
int isEmpty(Stack); // Check if a stack is empty or not.
int top(Stack); // Check the top element of a stack.
void push(Stack *, int); // Insert an integer to the top of a stack.
int pop(Stack *); // Remove the top element from a stack.
void printStack(Stack); // Print elements of a stack from the top to the bottom.
void printStackBottomToTop(Stack); // Print elements of a stack from the
                                   // bottom to the top.
```

   The main program declares two stacks s and t. Insert 50 integers 0 to 49 to stack s and then move the elements of stack s to t in the reversed order. Solutions: stack_array_reverse.dev, stack_array.h, stack_array.c, and stack_array_reverse.c. Example of program execution:

```
Print stack s (from top to bottom):
 49  48  47  46  45  44  43  42  41  40  39  38  37  36  35  34  33  32  31  30  29  28  27  26  25  24  23  22  21  20
 19  18  17  16  15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0

Print stack t (from top to bottom):
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29
 30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49
```

2. String str is a sequence of parentheses '()', square brackets '[]', and curly braces '{}'. Use a stack and its operations to check whether a string of parentheses of any length matches. For example, “[{((){[]})()}]{()[([])()]}” and “([][[]](){{}}))" are a matching parentheses string; "([][[]]({[{]}}))", "(((()))))" are not a matching parentheses string. Solutions: stack_array_parenthesis.dev, stack_array_parenthesis.h, stack_array_parenthesis.c, and stack_array_parenthesis_main.c.

   Example of program execution:

```
Input a string of parentheses ('(', ')', '[', ']', '{', '}'): [{(((){[]})())}]{()[([])()]}
**** String [{(((){[]})())}]{()[([])()]} is all matching parentheses.

---------------------------------------

Input a string of parentheses ('(', ')', '[', ']', '{', '}'): ([][[]](){{}}))
**** String ([][[]](){{}})) is all matching parentheses.

---------------------------------------

Input a string of parentheses ('(', ')', '[', ']', '{', '}'): ([][[]]({[{}}))
**** String ([][[]]({[{}})) is not all matching parentheses.

---------------------------------------

Input a string of parentheses ('(', ')', '[', ']', '{', '}'): ((()()))
**** String ((()())) is not all matching parentheses.

---------------------------------------

Input a string of parentheses ('(', ')', '[', ']', '{', '}'): stop
```

3. A queue is a container with first-in-first-out elements. A queue can be viewed as a ticket line that the first person joining the line is called the head of the line and the last person joining the line is called the tail of the line. Hence, an enqueue operation is to add a new element to the tail of a queue and a dequeue operation is to remove an element from the head of a queue.

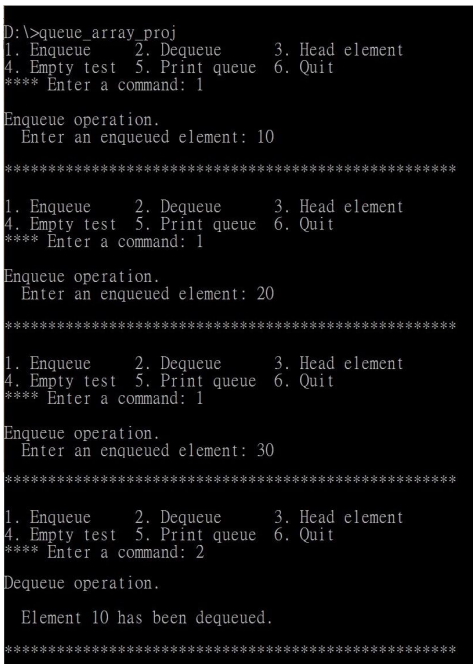
   Write a C project to define a data type representing queues using fixed-size array of integers and define the following queue operations:

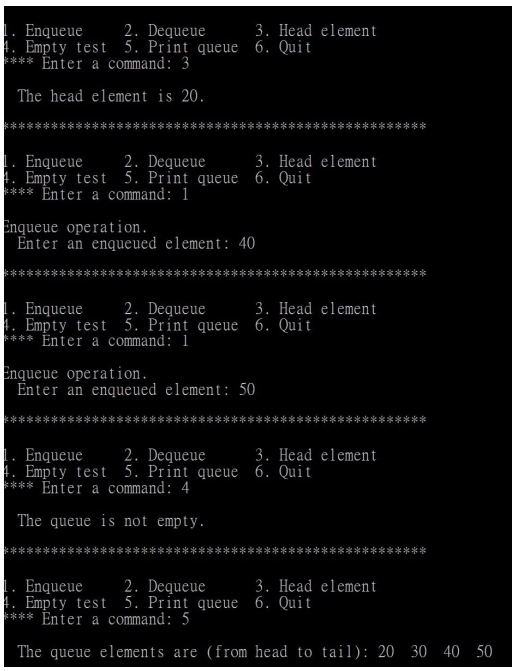
```
#define max 100 // Maximum 100 element.
// Type definition of queues using a fixed-size array.
// The queue elements of integers.
typedef struct {
  int elem[max]; //Queue container.
  int head; // Index of the queue head.
  int tail; // Index of the queue tail.
  int cnt; // Element count in a queue.
} Queue;
```

   1. **void** initial_queue(Queue *): Set a queue to empty, i.e., reset head and tail of a queue.
   2. **void** enqueue(Queue *, **int**): Add an element to the tail of a queue.
   3. **int** dequeue(Queue *): Remove an element from the head of a queue.
   4. **int** head(Queue): Get the element at the head of a queue.
   5. **int** is_empty(Queue): Check if a queue is empty or not.
   6. **void** print_queue(Queue): Print elements of a queue from the head to the tail.

The main program is a loop to repeatedly read an integer command between 1 and 6: (1) Enqueue, (2) Dequeue, (3) Head element, (4) Empty test, (5) Print queue, and (6) Quit.

Solutions: queue_array_proj.dev, queue_array.h, queue_array.c, and queue_array_main.c. Example of program execution:





4. A stack is a container that elements are inserted and removed in the *last in first out* (*or first in last out*) order. A stack has the following operations:
   (1) is_empty: check whether the stack is empty,
   (2) top: get the element at the top of the stack,
   (3) push: place an element to the top of the stack, and
   (4) pop: get the top element and remove the top element from the stack.

   Write a C project to define a data type representing stacks using dynamic array of integers and define the following stack operations:

```
#define SEGMENT 50 // Segment size.
typedef int ElemType; // Integer element type.
typedef struct {
   ElemType *elem; // Starting address of the stack continer.
   int top; // Index of the top element, initial value -1.
   int capacity; // Stack capacity, initial a segment.
} Stack;

void initial(Stack *); // Initialize a stack.
int is_empty(Stack); // Check if a stack is empty or not.
ElemType top(Stack); // Check the element at the top.
void push(Stack *, ElemType); // Insert an element to a stack.
// The following functions are not basic operations.
ElemType pop(Stack *); // Remove an element from a stack.
int get_size(Stack); // Check the size of the stack.
void clear(Stack *); // Clear the stack, set the capacity to one segment.
void print_stack(Stack) // Print the stack from bottom to top.
```

   In the main program declares a stack S. Use random number generator to get the number trials, maximum 10 trials, and perform the following operations in each trial:
   (1) Generate a random number push_count, number between 1 and 100, insert push_count elements to stack S, and print stack S from bottom to top.
   (2) Generate a random number pop_count, number between 1 and current stack size, remove pop_count elements to stack S, and print stack S from bottom to top.

Assume the value of stack elements is a random number between 0 and 99. Program solutions: stack_dynamic_array.dev, stack_dynamic_array.h, stack_dynamic_array.c, and stack_dynamic_array_main.c.