1. Given the following definitions:

```
#define MAX_STACK 100
typedef struct stack_type {
        ITEM_TYPE   item [MAX_STACK];
        int         top;
      } STACK_TYPE;
```

The definition of ITEM_TYPE is unspecified and left to the stack user.
Please implement the stack functions below. (30%)

```
void     create_stack (STACK_TYPE stack); /* Make stack logically accessible*/
void     destroy_stack (STACK_TYPE *stack); /* Make stack logically accessible*/
BOOLEAN empty_stack (STACK_TYPE stack); /* True if stack is empty */
BOOLEAN full_stack (STACK_TYPE *stack); /* True if stack is full */
void     push     (STACK_TYPE stack, ITEM_TYPE new_item);
                   /* Add item to the top of the stack */
void     pop      (STACK_TYPE *stack, ITEM_TYPE *old_item);
/* Remove item from the top of the stack */
```

2. Since precedence plays an important role in transforming infix to postfix, let us assume the existence of a function pred(op1, op2), where op1 and op2 are characters representing operators. This function returns TRUE if op1 has precedence over op2 when op1 appears to the left of op2 in an infix expression without parentheses. pred(op1, op2) returns FALSE otherwise. For example, pred('*', '+') and pred('+', '+') are TRUE, whereas pred('+', '*') is FALSE. To use the function to accommodate parentheses, please set the following precedence rules for parentheses using TRUE or FALSE: (20%)

   pred('(', op) = for any operator op
   pred(op, '(') = for any operator op other than ')'
   pred(op, ')') = for any operator op other than '('
   pred(')', op) = for any operator op

3. Consider a data structure to represent the queue. A queue node consists of an information field and a field holding a pointer to the next node. Given the following declarations:

```
typedef struct node_type {
      ITEM_TYPE         info;
      struct node_type  *next;
  }   NODE_TYPE;
typedef struct {
            NODE_TYPE    *front;
            NODE_TYPE    *rear;
      } Q_TYPE;
```

Please fill the following blanks in the implementation of the operation enqueue. The empty_queue (queue) returns true if the queue is empty. (25%)

```
void enqueue (Q_TYPE *queue, ITEM_TYPE item)   /add a new item to the rear of
the queue */
{
    NODE_PTR   new_node;
    new_node = ( NODE_PTR ) malloc (sizeof (NODE_TYPE));
    if (new_node != NULL ) {
    ________(1)______;
    ________(2)______;
        if ( empty_queue (queue ) == TRUE )
            ________(3)________;
        else
            ________(4)________;
        ________(5)________;
    }
}
```

4. Given a strictly binary tree *t* in which the *n* leaves are labeled as nodes 1 through n, let *level(i)* be the level of node *i* and let freq(i) be an integer assigned to node i. Define the weighted path length of *t* as the sum of *freq(i)*level(i)* over all leaves of *t*. Which one of the following is the strictly binary tree with minimum weighted path length. (a) Huffman tree (b) Binary search tree (c) Heap tree (d) Threaded binary tree (5%)

5. We are given a set of 6 positive weights 2, 3, 5, 7, 9, and 13. Exactly one of these weights is to be associated with each of the 6 external nodes in a binary tree with 5 internal nodes. The weighted external path length of such a binary tree is defined to be $\sum_{1 \le i \le 6} q_i k_i$ where k_i is the distance from the root node to the external node with weight q_i. Please compute the minimal weighted external path length of the tree. (20%)