1.Add Two Numbers

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */

Struct ListNode* addTwoNumbers(struct ListNode* l1, struct ListNode* l2) {
    Struct ListNode* head = NULL;
    Struct ListNode* tail = NULL;
    Int carry = 0;

    While (l1 != NULL || l2 != NULL) {
        Int x = (l1 != NULL) ? l1->val : 0;
        Int y = (l2 != NULL) ? l2->val : 0;
        Int sum = carry + x + y;

        Carry = sum / 10;
        Sum = sum % 10;

        Struct ListNode* node = (struct ListNode*)malloc(sizeof(struct ListNode));
        Node->val = sum;
        Node->next = NULL;

        If (head == NULL) {
            Head = tail = node;
```

```
        } else {
            Tail->next = node;
            Tail = node;
        }

        If (l1 != NULL) l1 = l1->next;
        If (l2 != NULL) l2 = l2->next;
    }

    If (carry > 0) {
        Struct ListNode* node = (struct ListNode*)malloc(sizeof(struct ListNode));
        Node->val = carry;
        Node->next = NULL;
        Tail->next = node;
    }

    Return head;
}



Int* twoSum(int* nums, int numsSize, int target, int* returnSize){
    Int* answer;
    Answer = malloc(2*sizeof(int));
    Bool success = false;
    For(int first_number = 0; first_number < numsSize; first_number++)
    {
        If(first_number < numsSize)
        {
            For(int second_number=(first_number + 1); second_number < numsSize; second_number++)
```

```
        {
            If((nums[first_number] + nums[second_number] == target))
            {
                Answer[0] = first_number;
                Answer[1] = second_number;
                Success = true;
            }
        }
    }
    If(success){
        Break;
    }
  }

  Return answer;
}
```

2.Longest Substring Without Repeating Characters

```
#define MAX_CHARS 256

Int lengthOfLongestSubstring(char* s) {
  Int last_occurrence[MAX_CHARS];
  Int start = 0;
  Int max_len = 0;

  Memset(last_occurrence, -1, sizeof(last_occurrence));

  For (int I = 0; s[i]; i++) {
    If (last_occurrence[s[i]] >= start) {
```

```
        Start = last_occurrence[s[i]] + 1;
      }
      Last_occurrence[s[i]] = I;
      Max_len = (I – start + 1 > max_len) ? (I – start + 1) : max_len;
   }

   Return max_len;
}
```

3.Median of Two Sorted Arrays

```
Double findMedianSortedArrays(int* nums1, int nums1Size, int* nums2, int nums2Size) {
   If(nums2Size < nums1Size) {
      Return findMedianSortedArrays(nums2, nums2Size, nums1, nums1Size);
   }
   Int n1 = nums1Size;
   Int n2 = nums2Size;
   Int low = 0, high = n1;
   While(low <= high) {
      Int cut1 = (low+high) >> 1;
      Int cut2 = (n1 + n2 + 1) / 2 – cut1;
      Int left1 = cut1 == 0 ? INT_MIN : nums1[cut1-1];
      Int left2 = cut2 == 0 ? INT_MIN : nums2[cut2-1];
      Int right1 = cut1 == n1 ? INT_MAX : nums1[cut1];
      Int right2 = cut2 == n2 ? INT_MAX : nums2[cut2];
      If(left1 <= right2 && left2 <= right1) {
         If( (n1 + n2) % 2 == 0 ) {
            Return (fmax(left1, left2) + fmin(right1, right2)) / 2.0;
         }
         Else {
```

```
            Return fmax(left1, left2);
        }
    }
    Else if(left1 > right2) {
        High = cut1 – 1;
    }
    Else {
        Low = cut1 + 1;
    }
  }
  Return 0.0;
}
```

4.Reverse Integer

```
#include <limits.h>

Int reverse(int x) {
  Int num = x;
  Long int rev = 0;
  While(num != 0){
    Int digit = num % 10;
    Rev = 10 * rev + digit;
    If(rev > INT_MAX) return 0;
    If(rev < INT_MIN) return 0;
    Num /= 10;
  }
  Return (int)rev;
}
```

5.3Sum

```
#include <stdio.h>
#include <stdlib.h>

Int cmp(const void *a, const void *b) {
   Return *(int*)a - *(int*)b;
}

Int** threeSum(int* nums, int numsSize, int* returnSize, int** returnColumnSizes) {
   Qsort(nums, numsSize, sizeof(int), cmp);
   Int** ans = malloc(sizeof(int*) * numsSize * numsSize); // allocate maximum possible memory
   *returnSize = 0;
   *returnColumnSizes = malloc(sizeof(int) * numsSize * numsSize); // allocate maximum possible memory
   Int l, r, sum3;
   For (int I = 0; I < numsSize; i++) {
      If (nums[i] > 0) break;
      If (I > 0 && nums[i] == nums[i-1]) continue;
      L = I + 1;
      R = numsSize – 1;
      While (l < r) {
         Sum3 = nums[i] + nums[l] + nums[r];
         If (sum3 == 0) {
            Ans[*returnSize] = malloc(sizeof(int) * 3);
            Ans[*returnSize][0] = nums[i];
            Ans[*returnSize][1] = nums[l];
            Ans[*returnSize][2] = nums[r];
            (*returnColumnSizes)[*returnSize] = 3;
            (*returnSize)++;
            While (l+1 < numsSize && nums[l+1] == nums[l]) l++;
```

```
            While (r-1 >= 0 && nums[r-1] == nums[r]) r--;
            L++;
            r--;
        } else if (sum3 < 0) {
            L++;
        } else {
            r--;
        }
      }
   }
   *returnColumnSizes = realloc(*returnColumnSizes, sizeof(int) * (*returnSize));
   Ans = realloc(ans, sizeof(int*) * (*returnSize));
   Return ans;
}
```

6.Remove Element

```
#include <stdio.h>

Int removeElement(int* nums, int numsSize, int val) {
   Int safe = numsSize – 1;
   Int j = numsSize – 1;
   While (j >= 0) {
      If (nums[j] == val) {
         Int temp = nums[j];
         Nums[j] = nums[safe];
         Nums[safe] = temp;
         Safe--;
      }
      j--;
```

```
    }
    Return safe + 1;
}
```

7.Rotate Image

```
#include <stdio.h>
Void rotate(int** matrix, int matrixSize, int* matrixColSize){
    Int n = *matrixColSize;
    For (int I = 0; I < n / 2 + n % 2; i++) {
        For (int j = 0; j < n / 2; j++) {
            Int temp = matrix[n – 1 – j][i];
            Matrix[n – 1 – j][i] = matrix[n – 1 – i][n – j – 1];
            Matrix[n – 1 – i][n – j – 1] = matrix[j][n – 1 -i];
            Matrix[j][n – 1 – i] = matrix[i][j];
            Matrix[i][j] = temp;
        }
    }
}
```

8.Single Number

```
#include <stdio.h>
Int singleNumber(int* nums, int numsSize){
    Int ans = nums[0];
    For (int I = 1; I < numsSize; ++i)
    {
        Ans ^= nums[i];
    }
    Return ans;
```

```
}
```

9.Number of 1 Bits

```
Int hammingWeight(uint32_t num) {
    Int res = 0;
    While (num > 0) {
        If ((num & 1) == 0) {
            Num >>= 1;
        } else {
            Num &= ~1;
            Res++;
        }
    }
    Return res;
}
```

10.Remove Duplicate Letters

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define MAX_CHARS 26

Char* removeDuplicateLetters(char* s) {
    Int len = strlen(s);
    Int map[MAX_CHARS] = {0};
```

```
    For (int I = 0; I < len; i++) {
        Map[s[i] – ‘a’]++;
    }

    Char* ans = (char*) malloc((MAX_CHARS + 1) * sizeof(char)); // Allocate enough memory for the resulting string
    Int top = -1;
    Bool visited[MAX_CHARS] = {false};

    For (int I = 0; I < len; i++) {
        If (visited[s[i] – ‘a’]) {
            Map[s[i] – ‘a’]--;
            Continue;
        }

        While (top >= 0 && visited[s[i] – ‘a’] == false && ans[top] >= s[i] && map[ans[top] – ‘a’] > 1) {
            Visited[ans[top] – ‘a’] = false;
            Map[ans[top] – ‘a’]--;
            Top--;
        }

        Ans[++top] = s[i];
        Visited[s[i] – ‘a’] = true;
    }

    Ans[top+1] = ‘\0’;
    Return ans;
}
```

11.Baseball Game

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_OPS 1000

Void remove_element(int* list, int* size)
{
   (*size)--;
}

Void double_element(int* list, int* size)
{
   Int last_element = list[*size – 1];
   List[*size] = last_element * 2;
   (*size)++;
}

Void add_elements(int* list, int* size)
{
   Int last_element = list[*size – 1];
   Int second_last_element = list[*size – 2];
   Int sum = last_element + second_last_element;
   List[*size] = sum;
   (*size)++;
}

Void add_integer(int* list, int* size, int val)
{
```

```
    List[*size] = val;
    (*size)++;
}

Int calPoints(char** ops, int opsSize)
{
    Int* list = (int*) malloc(sizeof(int) * MAX_OPS);
    Int size = 0;

    For (int I = 0; I < opsSize; i++) {
        Int val = 0;
        Int flag = 1;
        Char* current_op = ops[i];
        Char current_char = current_op[0];

        Switch (current_char) {
            Case ‘C’:
                Remove_element(list, &size);
                Break;
            Case ‘D’:
                Double_element(list, &size);
                Break;
            Case ‘+’:
                Add_elements(list, &size);
                Break;
            Default:
                If (current_char == ‘-‘) {
                    Flag = -1;
                    Current_op++;
```

```
            }

            Int length = strlen(current_op);
            For (int j = 0; j < length; j++) {
               Val *= 10;
               Val += current_op[j] – ‘0’;
            }

            Add_integer(list, &size, val * flag);
            Break;
        }
    }

    Int ans = 0;
    For (int I = 0; I < size; i++) {
       Ans += list[i];
    }

    Free(list);
    Return ans;
}
```

12.Counting Words With a Given Prefix

```
#include <stdio.h>
#include <string.h>

Int prefixCount(char** words, int wordsSize, char* pref) {
    Int p_len = strlen(pref);
    Int cnt = 0;
```

```
    For (int I = 0; I < wordsSize; i++) {
        If (strncmp(words[i], pref, p_len) == 0) {
            Cnt++;
        }
    }
    Return cnt;
}
```