1)INTEGER TO ROMAN NUMBERS

CODE

```c
#include<stdio.h>
typedef struct{
    char *sym;
    int val;
}numeral;
int maxNume(numeral *nu, int num){
    int i, index;
    for(i = 0; i<15; i++){//15 numerals in array
        if(nu[i].val <= num)
            index = i;
    }
    return index;
}
void decToRoman(numeral *nu, int num){
    int max;
    if(num != 0){
        max = maxNume(nu, num);
        printf("%s", nu[max].sym);
        num -= nu[max].val;//decrease number
        decToRoman(nu, num);//recursively print numerals
    }
}
main(){
    int number;
    numeral nume[15] = {{"I",1},{"IV",4},{"V",5},{"IX",9}, {"X",10},{"XL",40},{"L",50},{"XC",90},
{"C",100},{"CD",400},{"D",500},{"CM",900},{"M",1000},{"MMMM",4000},{"V'",5000}};
    printf("Enter a decimal number: ");
    scanf("%d", &number);
    if(number >0 && number <= 5000){//checking input number
        printf("The Roman equivalent of %d is ", number);
        decToRoman(nume, number);
    }
    else{
```

OUTPUT

```
C:\Users\selco\OneDrive\Doc
Enter a decimal number: 38
The Roman equivalent of 38 is XXXVIII
--------------------------------
Process exited after 4.734 seconds with return value 0
Press any key to continue . . .
```

2)ROMAN NUMBERS TO INTEGER

CODE:

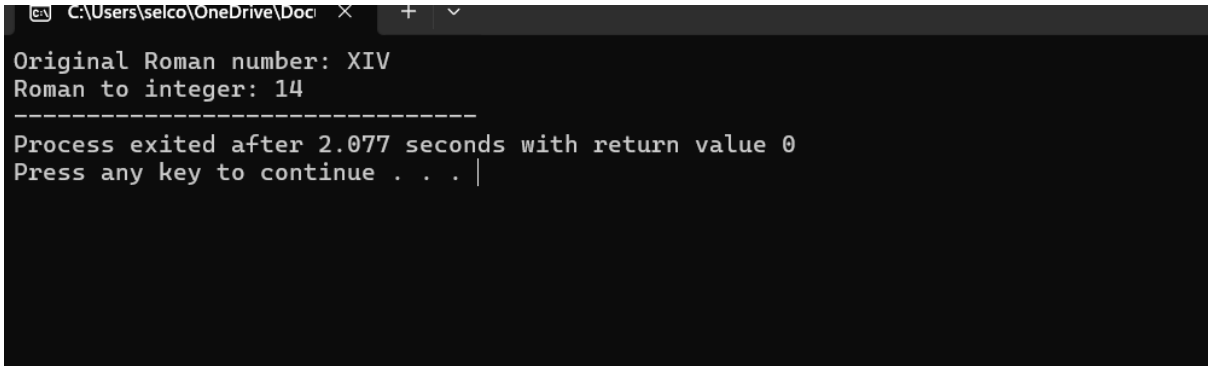
```c
#include <stdio.h>
#include <stdlib.h>
static int roman_to_integer(char c)
{
    switch(c) {
    case 'I':
        return 1;
    case 'V':
        return 5;
    case 'X':
        return 10;
    case 'L':
        return 50;
    case 'C':
        return 100;
    case 'D':
        return 500;
    case 'M':
        return 1000;
    default:
        return 0;
    }
}
int roman_to_int (char *s)
{
    int i, int_num = roman_to_integer(s[0]);

    for (i = 1; s[i] != '\0'; i++) {
        int prev_num = roman_to_integer(s[i - 1]);
        int cur_num = roman_to_integer(s[i]);
```

```c
        return 500;
    case 'M':
        return 1000;
    default:
        return 0;
    }
}
int roman_to_int (char *s)
{
    int i, int_num = roman_to_integer(s[0]);

    for (i = 1; s[i] != '\0'; i++) {
        int prev_num = roman_to_integer(s[i - 1]);
        int cur_num = roman_to_integer(s[i]);
        if (prev_num < cur_num) {
            int_num = int_num - prev_num + (cur_num - prev_num);
        } else {
            int_num += cur_num;
        }
    }
    return int_num;
}
int main(void)
{
  char *str1 = "XIV";
    printf("Original Roman number: %s", str1);
    printf("\nRoman to integer: %d", roman_to_int(str1));
    return 0;
}
```

OUTPUT


C:\Users\selco\OneDrive\Doc
Original Roman number: XIV
Roman to integer: 14

Process exited after 2.077 seconds with return value 0
Press any key to continue . . .

3)COMMON PREFIX

CODE:

```c
#include <stdio.h>
#include <string.h>
char* commonPrefixUtil(char* str1, char* str2)
{
    char* result = (char*)malloc(100 * sizeof(char));
    int len = strlen(str1) < strlen(str2) ? strlen(str1)
                                          : strlen(str2);
    int i;
    for (i = 0; i < len; i++) {
        if (str1[i] != str2[i])
            break;
        result[i] = str1[i];
    }
    result[len] = '\0';
    return result;
}
char* commonPrefix(char* arr[], int n)
{
    char* prefix = arr[0];
    int i;
    for (i = 1; i < n; i++) {
        prefix = commonPrefixUtil(prefix, arr[i]);
    }
    return prefix;
}
int main()
{
    char* arr[]
        = { "geeksforgeeks", "geeks", "geek", "geezer" };
    int n = sizeof(arr) / sizeof(arr[0]);
```

```c
        = { "geeksforgeeks", "geeks", "geek", "geezer" };
    int n = sizeof(arr) / sizeof(arr[0]);
    char* ans = commonPrefix(arr, n);
    if (strlen(ans))
        printf("The longest common prefix is - %s", ans);
    else
        printf("There is no common prefix");

    free(ans);
    return 0;
}
```

OUTPUT

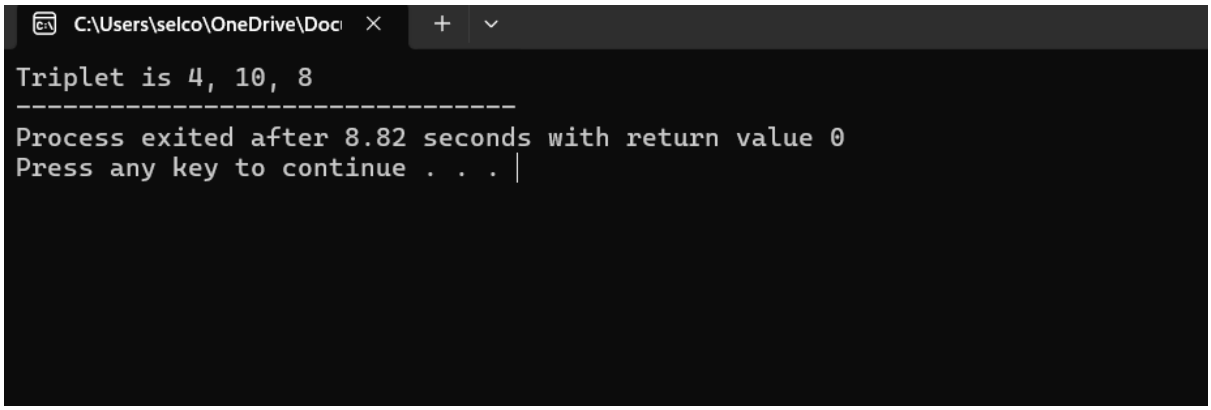
```
The longest common prefix is - gee
--------------------------------
Process exited after 2.149 seconds with return value 0
Press any key to continue . . .
```

4)3 SUM IN ARRAY OF INTEGERS

CODE

```c
#include <stdio.h>
#include<stdbool.h>
bool find3Numbers(int A[], int arr_size, int sum)
{
    int l, r;
    int i,j,k;
    for (i = 0; i < arr_size - 2; i++) {
        for (j = i + 1; j < arr_size - 1; j++) {
            for (k = j + 1; k < arr_size; k++) {
                if (A[i] + A[j] + A[k] == sum) {
                    printf("Triplet is %d, %d, %d",
                           A[i], A[j], A[k]);
                    return true;
                }
            }
        }
    }
    return false;
}
int main()
{
    int A[] = { 1, 4, 45, 6, 10, 8 };
    int sum = 22;
    int arr_size = sizeof(A) / sizeof(A[0]);
    find3Numbers(A, arr_size, sum);
    return 0;
}
```

OUTPUT:

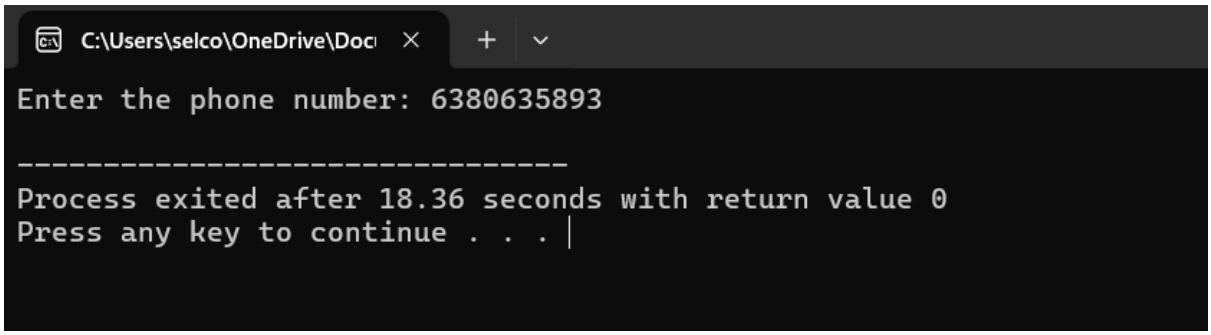


5)LETTER MAPPING TO A PHONE NUMBER

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
const char *keypad[] = {
    "",     "",     "abc",  "def",
    "ghi",  "jkl",  "mno",
    "pqrs", "tuv",  "wxyz"
};
void generateCombinations(char *number, int curr_digit, char *output, int n) {
    if (curr_digit == n) {
        output[curr_digit] = '\0';
        printf("%s\n", output);
        return;
    }
    const char *letters = keypad[number[curr_digit] - '0'];
    for (int i = 0; i < strlen(letters); i++) {
        output[curr_digit] = letters[i];
        generateCombinations(number, curr_digit + 1, output, n);
    }
}
int main() {
    char number[100];
    printf("Enter the phone number: ");
    scanf("%s", number);
    int n = strlen(number);
    char *output = (char *)malloc((n + 1) * sizeof(char));
    generateCombinations(number, 0, output, n);
    free(output);
    return 0;
}
```

OUTPUT:



6)4 SUM IN ARRAY OF INTEGERS

CODE:

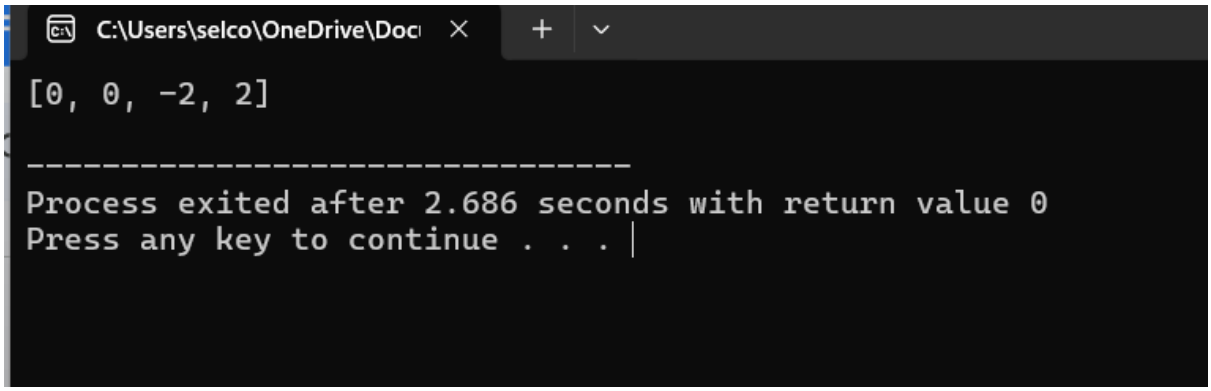
```c
#include <stdio.h>
#include <stdlib.h>
int compare(const void *a, const void *b) {
    return ((int)a - (int)b);
}
void fourSum(int* nums, int numsSize, int target) {
    qsort(nums, numsSize, sizeof(int), compare);

    for (int i = 0; i < numsSize - 3; i++) {
        if (i > 0 && nums[i] == nums[i - 1]) continue;  // Skip duplicates
        for (int j = i + 1; j < numsSize - 2; j++) {
            if (j > i + 1 && nums[j] == nums[j - 1]) continue;  // Skip duplicates
            int left = j + 1;
            int right = numsSize - 1;
            while (left < right) {
                int sum = nums[i] + nums[j] + nums[left] + nums[right];
                if (sum == target) {
                    printf("[%d, %d, %d, %d]\n", nums[i], nums[j], nums[left], nums[right]);
                    while (left < right && nums[left] == nums[left + 1]) left++;  // Skip duplicates
                    while (left < right && nums[right] == nums[right - 1]) right--;  // Skip duplicates
                    left++;
                    right--;
                } else if (sum < target) {
                    left++;
                } else {
                    right--;
                }
            }
        }
    }
}
int main() {
    int nums[] = {1, 0, -1, 0, -2, 2};
    int target = 0;
    int numsSize = sizeof(nums) / sizeof(nums[0]);

    fourSum(nums, numsSize, target);

    return 0;
}
```

OUTPUT:



7)REMOVE NTH NODE OF END OF LIST

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
struct ListNode {
    int val;
    struct ListNode *next;
};
struct ListNode* createNode(int val) {
    struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
    newNode->val = val;
    newNode->next = NULL;
    return newNode;
}
void printList(struct ListNode* head) {
    struct ListNode* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->val);
        temp = temp->next;
    }
    printf("NULL\n");
}
struct ListNode* removeNthFromEnd(struct ListNode* head, int n) {
    struct ListNode dummy;
    dummy.next = head;
    int i;
    struct ListNode *first = &dummy, *second = &dummy;
    for ( i = 0; i <= n; i++) {
        if (first == NULL) return head;
        first = first->next;
    }
    while (first != NULL) {
        first = first->next;
        second = second->next;
    }
    struct ListNode* nodeToRemove = second->next;
    second->next = second->next->next;
    free(nodeToRemove);
    return dummy.next;
}
int main() {
    // Creating a linked list: 1 -> 2 -> 3 -> 4 -> 5
    struct ListNode* head = createNode(1);
    head->next = createNode(2);
    head->next->next = createNode(3);
    head->next->next->next = createNode(4);
    head->next->next->next->next = createNode(5);
    printf("Original list: ");
    printList(head);
    int n = 2;
    head = removeNthFromEnd(head, n);
    printf("List after removing %dth node from the end: ", n);
    printList(head);
    while (head != NULL) {
        struct ListNode* temp = head;
        head = head->next;
        free(temp);
    }
    return 0;
}
```

OUTPUT:

```
C:\Users\selco\OneDrive\Doc
Original list: 1 -> 2 -> 3 -> 4 -> 5 -> NULL
List after removing 2th node from the end: 1 -> 2 -> 3 -> 5 -> NULL

--------------------------------
Process exited after 2.048 seconds with return value 0
Press any key to continue . . .
```

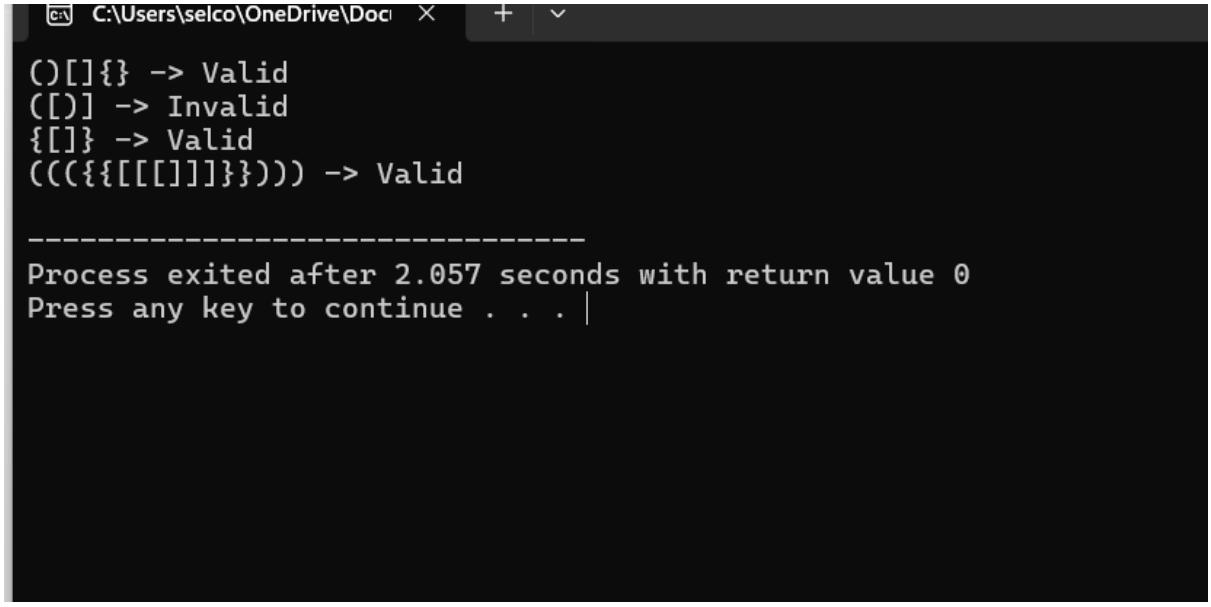
8)VALID PARENTHESSES

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
typedef struct {
    char *data;
    int top;
    int capacity;
} Stack;
Stack* createStack(int capacity) {
    Stack* stack = (Stack*)malloc(sizeof(Stack));
    stack->data = (char*)malloc(capacity * sizeof(char));
    stack->top = -1;
    stack->capacity = capacity;
    return stack;
}
void push(Stack* stack, char ch) {
    if (stack->top == stack->capacity - 1) {
        // Resize the stack if it's full
        stack->capacity *= 2;
        stack->data = (char*)realloc(stack->data, stack->capacity * sizeof(char));
    }
    stack->data[++(stack->top)] = ch;
}
char pop(Stack* stack) {
    if (stack->top == -1) return '\0';
    return stack->data[(stack->top)--];
}
bool isEmpty(Stack* stack) {
    return stack->top == -1;
}
bool isValid(char* s) {
    int len = strlen(s);
    Stack* stack = createStack(len);

    for (int i = 0; i < len; i++) {
        char ch = s[i];
        if (ch == '(' || ch == '{' || ch == '[') {
            push(stack, ch);
        } else {
            if (isEmpty(stack)) {
                free(stack->data);
                free(stack);
                return false;
            }
            char top = pop(stack);
            if ((ch == ')' && top != '(') ||
                (ch == '}' && top != '{') ||
                (ch == ']' && top != '[')) {
                free(stack->data);
                free(stack);
                return false;
            }
        }
    }
    bool result = isEmpty(stack);
    free(stack->data);
    free(stack);
    return result;
    free(stack->data);
    free(stack);
    return result;
}
int main() {
    char s1[] = "()[]{}";
    char s2[] = "([)]";
    char s3[] = "{[]}";
    char s4[] = "(((((((((]]]]})))))))";
    printf("%s -> %s\n", s1, isValid(s1) ? "Valid" : "Invalid");
    printf("%s -> %s\n", s2, isValid(s2) ? "Valid" : "Invalid");
    printf("%s -> %s\n", s3, isValid(s3) ? "Valid" : "Invalid");
    printf("%s -> %s\n", s4, isValid(s4) ? "Valid" : "Invalid");
    return 0;
}
```

OUTPUT:



9)3 SUM CLOSEST TO THE ARRAY OF INTERGERS:

CODE:

```c
#include <stdio.h>
#include<stdbool.h>
bool find3Numbers(int A[], int arr_size, int sum)
{
    int l, r;
    for (int i = 0; i < arr_size - 2; i++) {
        for (int j = i + 1; j < arr_size - 1; j++) {
            for (int k = j + 1; k < arr_size; k++) {
                if (A[i] + A[j] + A[k] == sum) {
                    printf("Triplet is %d, %d, %d",
                           A[i], A[j], A[k]);
                    return true;
                }
            }
        }
    }
    return false;
}
int main()
{
    int A[] = { 1, 4, 45, 6, 10, 8 };
    int sum = 22;
    int arr_size = sizeof(A) / sizeof(A[0]);
    find3Numbers(A, arr_size, sum);
    return 0;
}
```

OUTPUT:

```
Triplet is 4, 10, 8
--------------------------------
Process exited after 2.841 seconds with return value 0
Press any key to continue . . .
```

10) CONTAINER WITH MOST WATER

CODE:

```c
#include <stdio.h>
int maxArea(int* height, int heightSize) {
    int left = 0;
    int right = heightSize - 1;
    int max_area = 0;
    while (left < right) {
        int width = right - left;
        int height_min = height[left] < height[right]? height[left] : height[right];
        int area = width * height_min;
        if (area > max_area) {
            max_area = area;
        }
        if (height[left] < height[right]) {
            left++;
        } else {
            right--;
        }
    }
    return max_area;
}
int main() {
    int height[] = {1,8,6,2,5,4,8,3,7};
    int heightSize = sizeof(height) / sizeof(height[0]);

    int max_area = maxArea(height, heightSize);
    printf("Maximum area: %d\n", max_area);

    return 0;
}
```

OUTPUT:

```
Maximum area: 49

--------------------------------
Process exited after 2.053 seconds with return value 0
Press any key to continue . . .
```