

1) INTEGER TO ROMAN NUMBERS

CODE

```
1  #include <stdio.h>
2  typedef struct{
3      char *sym;
4      int val;
5  }numeral;
6  int maxNume(numeral *nu, int num){
7      int i, index;
8      for(i = 0; i<15; i++){//15 numerals in array
9          if(nu[i].val <= num)
10             index = i;
11      }
12      return index;
13  }
14  void decToRoman(numeral *nu, int num){
15      int max;
16      if(num != 0){
17          max = maxNume(nu, num);
18          printf("%s", nu[max].sym);
19          num -= nu[max].val;//decrease number
20          decToRoman(nu, num);//recursively print numerals
21      }
22  }
23  main(){
24      int number;
25      numeral nume[15] = {{"I",1}, {"IV",4}, {"V",5}, {"IX",9}, {"X",10}, {"XL",40}, {"L",50}, {"XC",90},
26      {"C",100}, {"CD",400}, {"D",500}, {"CM",900}, {"M",1000}, {"MMMM",4000}, {"V",5000}};
27      printf("Enter a decimal number: ");
28      scanf("%d", &number);
29      if(number >0 && number <= 5000){//checking input number
30          printf("The Roman equivalent of %d is ", number);
31          decToRoman(nume, number);
32      }
33      else{
```

OUTPUT

```
C:\Users\seldo\OneDrive\Doc  ×  +  v
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:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  static int roman_to_integer(char c)
4  {
5      switch(c) {
6          case 'I':
7              return 1;
8          case 'V':
9              return 5;
10         case 'X':
11             return 10;
12         case 'L':
13             return 50;
14         case 'C':
15             return 100;
16         case 'D':
17             return 500;
18         case 'M':
19             return 1000;
20         default:
21             return 0;
22     }
23 }
24 int roman_to_int (char *s)
25 {
26     int i, int_num = roman_to_integer(s[0]);
27
28     for (i = 1; s[i] != '\0'; i++) {
29         int prev_num = roman_to_integer(s[i - 1]);
30         int cur_num = roman_to_integer(s[i]);
31
32         if (prev_num < cur_num) {
33             int_num = int_num - prev_num + (cur_num - prev_num);
34         } else {
35             int_num += cur_num;
36         }
37     }
38     return int_num;
39 }
40 int main(void)
41 {
42     char *str1 = "XIV";
43     printf("Original Roman number: %s", str1);
44     printf("\nRoman to integer: %d", roman_to_int(str1));
45     return 0;
46 }
```

OUTPUT

```
C:\Users\sclco\OneDrive\Doc  X + v
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:

```
1 #include <stdio.h>
2 #include <string.h>
3 char* commonPrefixUtil(char* str1, char* str2)
4 {
5     char* result = (char*)malloc(100 * sizeof(char));
6     int len = strlen(str1) < strlen(str2) ? strlen(str1)
7             : strlen(str2);
8     int i;
9     for (i = 0; i < len; i++) {
10         if (str1[i] != str2[i])
11             break;
12         result[i] = str1[i];
13     }
14     result[len] = '\0';
15     return result;
16 }
17 char* commonPrefix(char* arr[], int n)
18 {
19     char* prefix = arr[0];
20     int i;
21     for (i = 1; i < n; i++) {
22         prefix = commonPrefixUtil(prefix, arr[i]);
23     }
24     return prefix;
25 }
26 int main()
27 {
28     char* arr[]
29     = { "geeksforgeeks", "geeks", "geek", "geezer" };
30     int n = sizeof(arr) / sizeof(arr[0]);
31
32     char* ans = commonPrefix(arr, n);
33     if (strlen(ans))
34         printf("The longest common prefix is - %s", ans);
35     else
36         printf("There is no common prefix");
37
38     free(ans);
39     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

```
1  #include <stdio.h>
2  #include<stdbool.h>
3  bool find3Numbers(int A[], int arr_size, int sum)
4  {
5      int l, r;
6      int i,j,k;
7      for (i = 0; i < arr_size - 2; i++) {
8          for (j = i + 1; j < arr_size - 1; j++) {
9              for (k = j + 1; k < arr_size; k++) {
10                 if (A[i] + A[j] + A[k] == sum) {
11                     printf("Triplet is %d, %d, %d",
12                         A[i], A[j], A[k]);
13                     return true;
14                 }
15             }
16         }
17     }
18     return false;
19 }
20 int main()
21 {
22     int A[] = { 1, 4, 45, 6, 10, 8 };
23     int sum = 22;
24     int arr_size = sizeof(A) / sizeof(A[0]);
25     find3Numbers(A, arr_size, sum);
26     return 0;
27 }
```

OUTPUT:

```
C:\Users\seldo\OneDrive\Docu  X  +  v
Triplet is 4, 10, 8
-----
Process exited after 8.82 seconds with return value 0
Press any key to continue . . . |
```

5) LETTER MAPPING TO A PHONE NUMBER

CODE:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  const char *keypad[] = {
5      "", "", "abc", "def",
6      "ghi", "jkl", "mno",
7      "pqrs", "tuv", "wxyz"
8  };
9  void generateCombinations(char *number, int curr_digit, char *output, int n) {
10     if (curr_digit == n) {
11         output[curr_digit] = '\0';
12         printf("%s\n", output);
13         return;
14     }
15     const char *letters = keypad[number[curr_digit] - '0'];
16     for (int i = 0; i < strlen(letters); i++) {
17         output[curr_digit] = letters[i];
18         generateCombinations(number, curr_digit + 1, output, n);
19     }
20 }
21 int main() {
22     char number[100];
23     printf("Enter the phone number: ");
24     scanf("%s", number);
25     int n = strlen(number);
26     char *output = (char *)malloc((n + 1) * sizeof(char));
27     generateCombinations(number, 0, output, n);
28     free(output);
29     return 0;
30 }
```

OUTPUT:

```
C:\Users\selsco\OneDrive\Doc... X + v
Enter the phone number: 6380635893
-----
Process exited after 18.36 seconds with return value 0
Press any key to continue . . . |
```

6)4 SUM IN ARRAY OF INTEGERS

CODE:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int compare(const void *a, const void *b) {
4     return ((int)a - (int)b);
5 }
6 void fourSum(int* nums, int numsSize, int target) {
7     qsort(nums, numsSize, sizeof(int), compare);
8
9     for (int i = 0; i < numsSize - 3; i++) {
10         if (i > 0 && nums[i] == nums[i - 1]) continue; // Skip duplicates
11         for (int j = i + 1; j < numsSize - 2; j++) {
12             if (j > i + 1 && nums[j] == nums[j - 1]) continue; // Skip duplicates
13             int left = j + 1;
14             int right = numsSize - 1;
15             while (left < right) {
16                 int sum = nums[i] + nums[j] + nums[left] + nums[right];
17                 if (sum == target) {
18                     printf("[%d, %d, %d, %d]\n", nums[i], nums[j], nums[left], nums[right]);
19                     while (left < right && nums[left] == nums[left + 1]) left++; // Skip duplicates
20                     while (left < right && nums[right] == nums[right - 1]) right--; // Skip duplicates
21                     left++;
22                     right--;
23                 } else if (sum < target) {
24                     left++;
25                 } else {
26                     right--;
27                 }
28             }
29         }
30     }
31 }
32
33 int main() {
34     int nums[] = {1, 0, -1, 0, -2, 2};
35     int target = 0;
36     int numsSize = sizeof(nums) / sizeof(nums[0]);
37
38     fourSum(nums, numsSize, target);
39
40     return 0;
41 }
```

OUTPUT:

```
C:\Users\seldo\OneDrive\Docl × + v
[0, 0, -2, 2]
-----
Process exited after 2.686 seconds with return value 0
Press any key to continue . . . |
```

7)REMOVE NTH NODE OF END OF LIST

CODE:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct ListNode {
4     int val;
5     struct ListNode *next;
6 };
7 struct ListNode* createNode(int val) {
8     struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
9     newNode->val = val;
10    newNode->next = NULL;
11    return newNode;
12 }
13 void printList(struct ListNode* head) {
14     struct ListNode* temp = head;
15     while (temp != NULL) {
16         printf("%d -> ", temp->val);
17         temp = temp->next;
18     }
19     printf("NULL\n");
20 }
21 struct ListNode* removeNthFromEnd(struct ListNode* head, int n) {
22     struct ListNode dummy;
23     dummy.next = head;
24     int i;
25     struct ListNode *first = &dummy, *second = &dummy;
26     for (i = 0; i <= n; i++) {
27         if (first == NULL) return head;
28         first = first->next;
29     }
30     while (first != NULL) {
31         first = first->next;
32         second = second->next;
33     }
34     struct ListNode* nodeToRemove = second->next;
35     second->next = second->next->next;
36     free(nodeToRemove);
37     return dummy.next;
38 }
39 int main() {
40     // Creating a Linked List: 1 -> 2 -> 3 -> 4 -> 5
41     struct ListNode* head = createNode(1);
42     head->next = createNode(2);
43     head->next->next = createNode(3);
44     head->next->next->next = createNode(4);
45     head->next->next->next->next = createNode(5);
46     printf("Original list: ");
47     printList(head);
48     int n = 2;
49     head = removeNthFromEnd(head, n);
50     printf("List after removing %dth node from the end: ", n);
51     printList(head);
52     while (head != NULL) {
53         struct ListNode* temp = head;
54         head = head->next;
55         free(temp);
56     }
57     return 0;
58 }
```

OUTPUT:

```
C:\Users\seldo\OneDrive\Doc  X + v
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 PARENTHESES

CODE:


```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4  #include <string.h>
5  typedef struct {
6      char *data;
7      int top;
8      int capacity;
9  } Stack;
10 Stack* createStack(int capacity) {
11     Stack* stack = (Stack*)malloc(sizeof(Stack));
12     stack->data = (char*)malloc(capacity * sizeof(char));
13     stack->top = -1;
14     stack->capacity = capacity;
15     return stack;
16 }
17 void push(Stack* stack, char ch) {
18     if (stack->top == stack->capacity - 1) {
19         // Resize the stack if it's full
20         stack->capacity *= 2;
21         stack->data = (char*)realloc(stack->data, stack->capacity * sizeof(char));
22     }
23     stack->data[++(stack->top)] = ch;
24 }
25 char pop(Stack* stack) {
26     if (stack->top == -1) return '\0';
27     return stack->data[(stack->top)--];
28 }
29 bool isEmpty(Stack* stack) {
30     return stack->top == -1;
31 }
32 bool isValid(char* s) {
33     int len = strlen(s);
34     Stack* stack = createStack(len);
35
36     for (int i = 0; i < len; i++) {
37         char ch = s[i];
38         if (ch == '(' || ch == '{' || ch == '[') {
39             push(stack, ch);
40         } else {
41             if (isEmpty(stack)) {
42                 free(stack->data);
43                 free(stack);
44                 return false;
45             }
46             char top = pop(stack);
47             if ((ch == ')' && top != '(') ||
48                 (ch == '}' && top != '{') ||
49                 (ch == ']' && top != '[')) {
50                 free(stack->data);
51                 free(stack);
52                 return false;
53             }
54         }
55     }
56     bool result = isEmpty(stack);
57     free(stack->data);
58     free(stack);
59     return result;
60 }
61 int main() {
62     char s1[] = "()[]{}";
63     char s2[] = "([)]";
64     char s3[] = "{[]}";
65     char s4[] = "((((([[[[]]]]])))";
66     printf("%s -> %s\n", s1, isValid(s1) ? "Valid" : "Invalid");
67     printf("%s -> %s\n", s2, isValid(s2) ? "Valid" : "Invalid");
68     printf("%s -> %s\n", s3, isValid(s3) ? "Valid" : "Invalid");
69     printf("%s -> %s\n", s4, isValid(s4) ? "Valid" : "Invalid");
70     return 0;
71 }

```

OUTPUT:

```
C:\Users\sclco\OneDrive\Docl X + v
()[]{} -> Valid
([]) -> Invalid
{[]} -> Valid
((([]{[][]}))) -> Valid

-----
Process exited after 2.057 seconds with return value 0
Press any key to continue . . . |
```

9)3 SUM CLOSEST TO THE ARRAY OF INTERGERS:

CODE:

```
1 #include <stdio.h>
2 #include<stdbool.h>
3 bool find3Numbers(int A[], int arr_size, int sum)
4 {
5     int l, r;
6     for (int i = 0; i < arr_size - 2; i++) {
7         for (int j = i + 1; j < arr_size - 1; j++) {
8             for (int k = j + 1; k < arr_size; k++) {
9                 if (A[i] + A[j] + A[k] == sum) {
10                     printf("Triplet is %d, %d, %d",
11                         A[i], A[j], A[k]);
12                     return true;
13                 }
14             }
15         }
16     }
17     return false;
18 }
19 int main()
20 {
21     int A[] = { 1, 4, 45, 6, 10, 8 };
22     int sum = 22;
23     int arr_size = sizeof(A) / sizeof(A[0]);
24     find3Numbers(A, arr_size, sum);
25     return 0;
26 }
```

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:

```
1  #include <stdio.h>
2  int maxArea(int* height, int heightSize) {
3      int left = 0;
4      int right = heightSize - 1;
5      int max_area = 0;
6      while (left < right) {
7          int width = right - left;
8          int height_min = height[left] < height[right]? height[left] : height[right];
9          int area = width * height_min;
10         if (area > max_area) {
11             max_area = area;
12         }
13         if (height[left] < height[right]) {
14             left++;
15         } else {
16             right--;
17         }
18     }
19     return max_area;
20 }
21 int main() {
22     int height[] = {1,8,6,2,5,4,8,3,7};
23     int heightSize = sizeof(height) / sizeof(height[0]);
24
25     int max_area = maxArea(height, heightSize);
26     printf("Maximum area: %d\n", max_area);
27
28     return 0;
29 }
```

OUTPUT:

```
Maximum area: 49
```

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