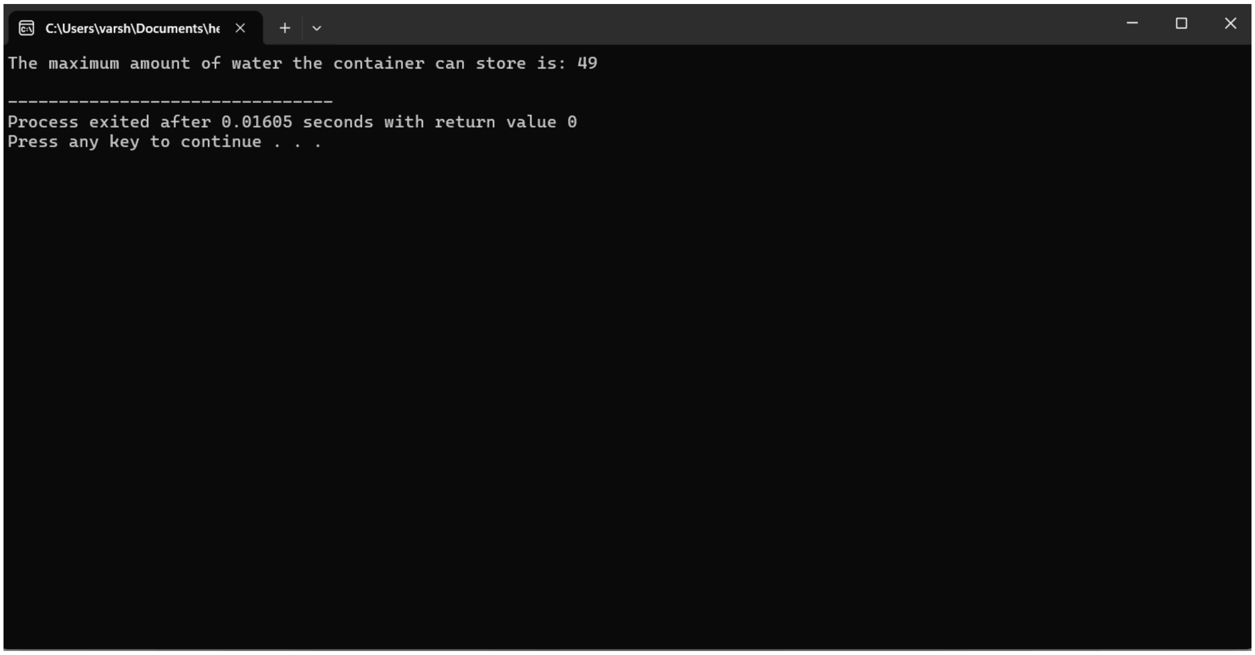
DAA ASSIGNMENT 2:

1. Container with Most Water:

```
#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 h = height[left] < height[right] ? height[left] : height[right];
        int current_area = width * h;
        if (current_area > max_area) {
            max_area = current_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]);
    printf("Max area: %d\n", maxArea(height, heightSize));
    return 0;
}
```

OUTPUT:



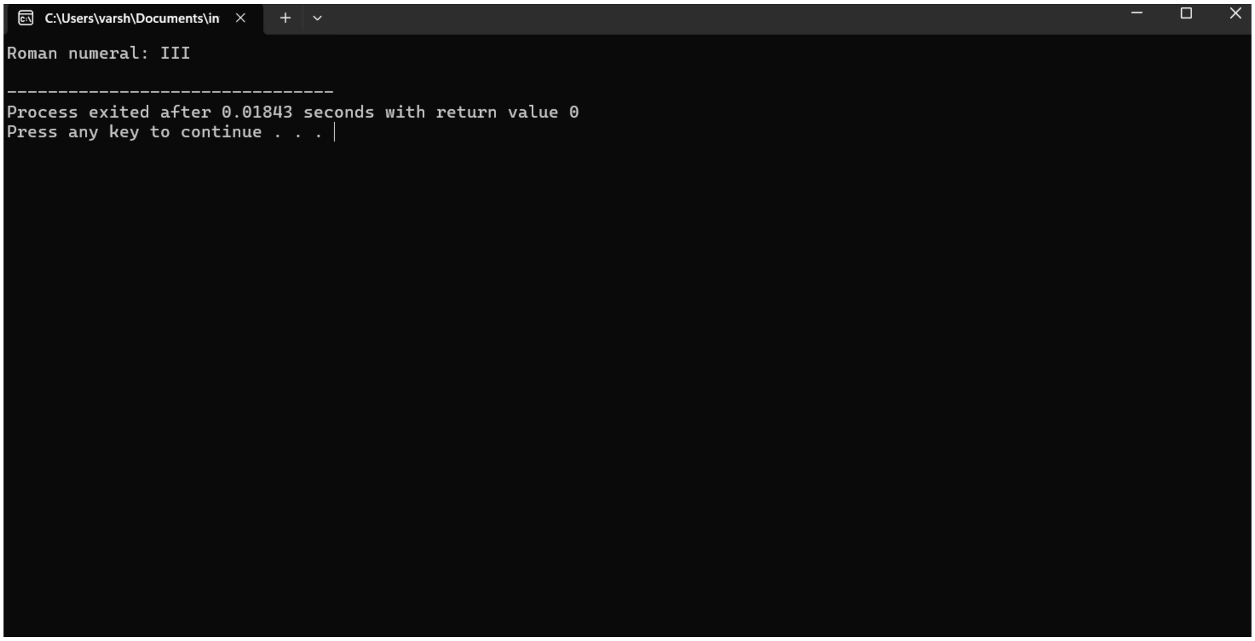
```
The maximum amount of water the container can store is: 49

--------------------------------
Process exited after 0.01605 seconds with return value 0
Press any key to continue . . .
```

2.Integer to Roman

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

void intToRoman(int num, char* result) {
       const char *roman[] = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX",
"V", "IV", "I"};
    const int values[] = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};

        result[0] = '\0';

      for (int i = 0; i < 13; i++) {
                while (num >= values[i]) {
            num -= values[i];
            strcat(result, roman[i]);
        }
    }
}
```

```
int main() {
    int num = 3;
    char result[20]; // Buffer to store the Roman numeral result
    intToRoman(num, result);
    printf("Roman numeral: %s\n", result);
    return 0;
}
```

OUTPUT:

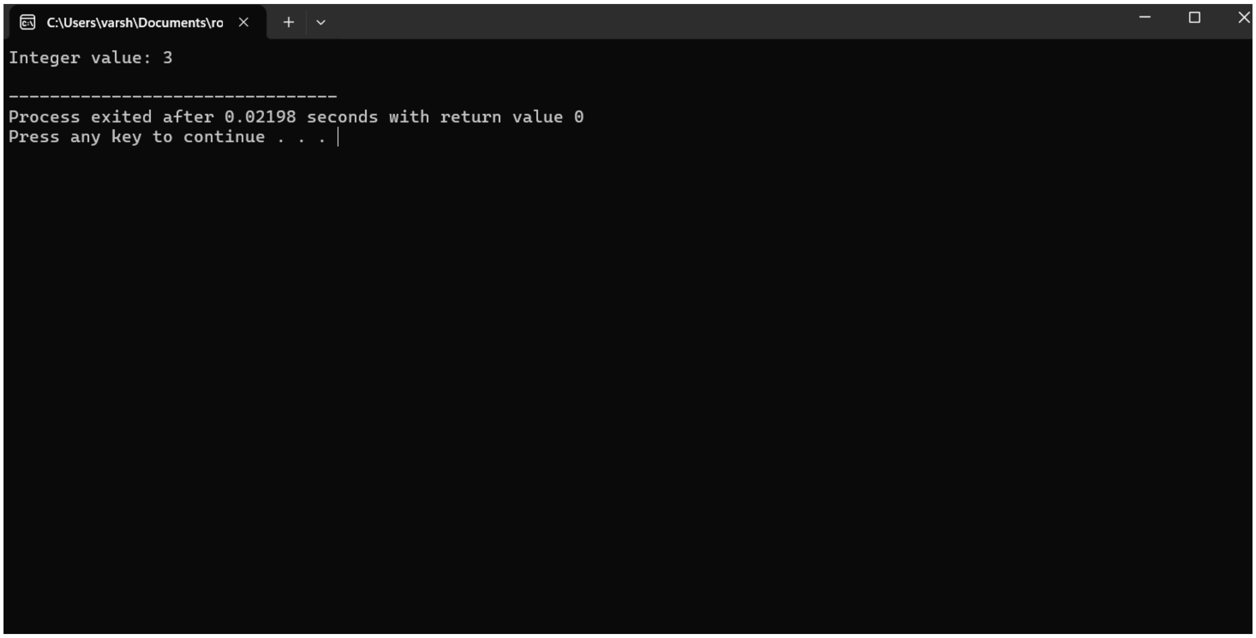


```
Roman numeral: III

--------------------------------
Process exited after 0.01843 seconds with return value 0
Press any key to continue . . .
```

3.Roman to Integer:

```
#include <stdio.h>
#include <string.h>
int romanCharToValue(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 romanToInt(char *s) {
    int length = strlen(s);
    int total = 0;

    for (int i = 0; i < length; i++) {
        int current = romanCharToValue(s[i]);
        int next = (i + 1 < length) ? romanCharToValue(s[i + 1]) : 0;

        if (current < next) {
            total -= current;
        } else {
            total += current;
        }
    }

    return total;
}

int main() {
    char s[] = "III";
    printf("Integer value: %d\n", romanToInt(s));
    return 0;
}
```

OUTPUT:

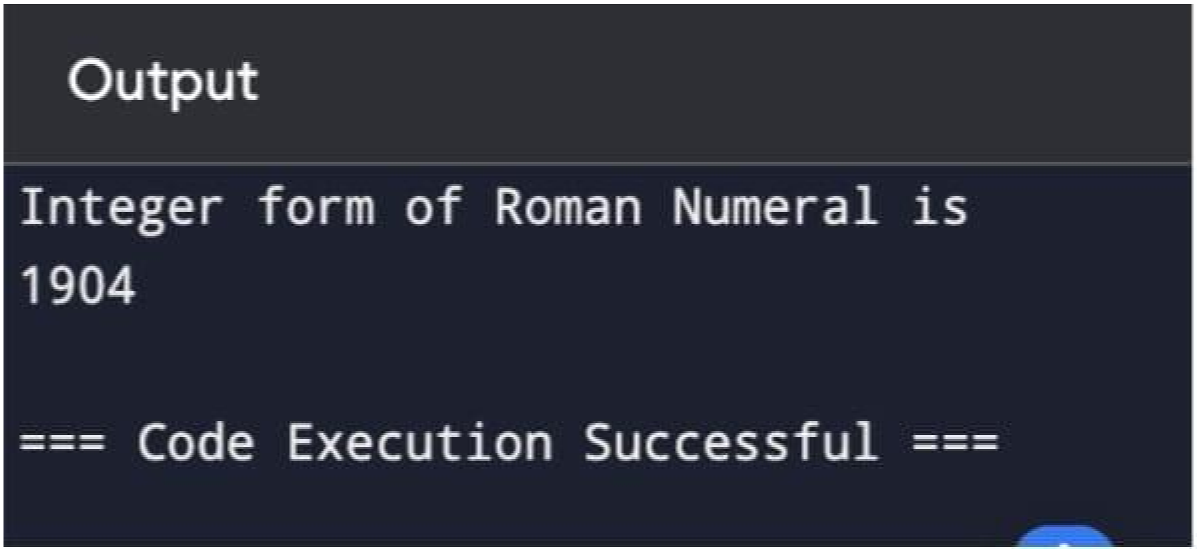


```
Integer value: 3

--------------------------------
Process exited after 0.02198 seconds with return value 0
Press any key to continue . . .
```

4.Longest Common Prefix

```
#include <stdio.h>
#include <string.h>
char* longestCommonPrefix(char** strs, int strsSize) {
    if (strsSize == 0) {
        return "";
    }

     char* prefix = strs[0];

    for (int i = 1; i < strsSize; i++) {
        int j = 0;
              while (prefix[j] && strs[i][j] && prefix[j] == strs[i][j]) {
            j++;
        }
               prefix[j] = '\0';

               if (j == 0) {
            return "";
        }
    }

    return prefix;
}

int main() {
    char* strs[] = {"flower", "flow", "flight"};
    int strsSize = sizeof(strs) / sizeof(strs[0]);
    printf("Longest common prefix: %s\n", longestCommonPrefix(strs, strsSize));
    return 0;
}
```

OUTPUT



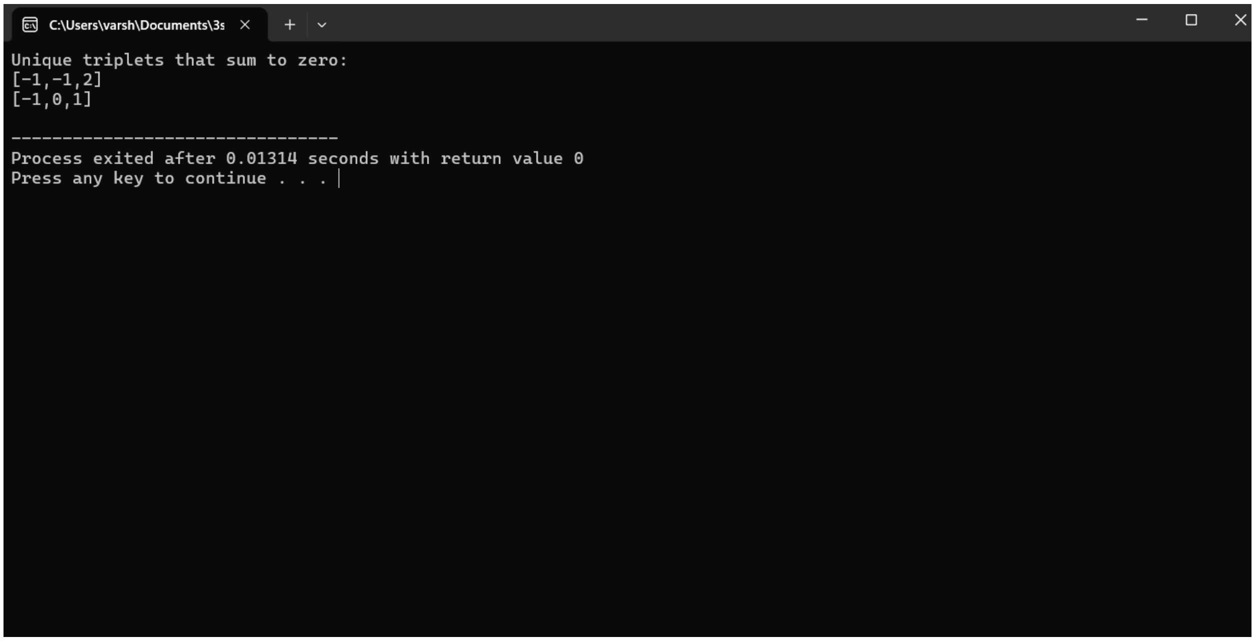
```
Integer form of Roman Numeral is
1904


=== Code Execution Successful ===
```

5.3Sum:

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

int compare(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}

void threeSum(int* nums, int numsSize) {
    if (numsSize < 3) {
        printf("[]\n");
        return;
    }

      qsort(nums, numsSize, sizeof(int), compare);

      for (int i = 0; i < numsSize - 2; i++) {
        if (i > 0 && nums[i] == nums[i - 1]) {
            continue; // Skip duplicate elements
        }

        int left = i + 1;
        int right = numsSize - 1;

        while (left < right) {
            int sum = nums[i] + nums[left] + nums[right];

            if (sum == 0) {
                printf("[%d,%d,%d]\n", nums[i], nums[left], nums[right]);

                             while (left < right && nums[left] == nums[left + 1]) left++;
                while (left < right && nums[right] == nums[right - 1]) right--;

                left++;
                right--;
            } else if (sum < 0) {
                left++;
            } else {
                right--;
            }
        }
    }
}
```

```
int main() {
    int nums[] = {-1, 0, 1, 2, -1, -4};
    int numsSize = sizeof(nums) / sizeof(nums[0]);
    printf("Unique triplets that sum to zero:\n");
    threeSum(nums, numsSize);
    return 0;
}
```

OUTPUT:



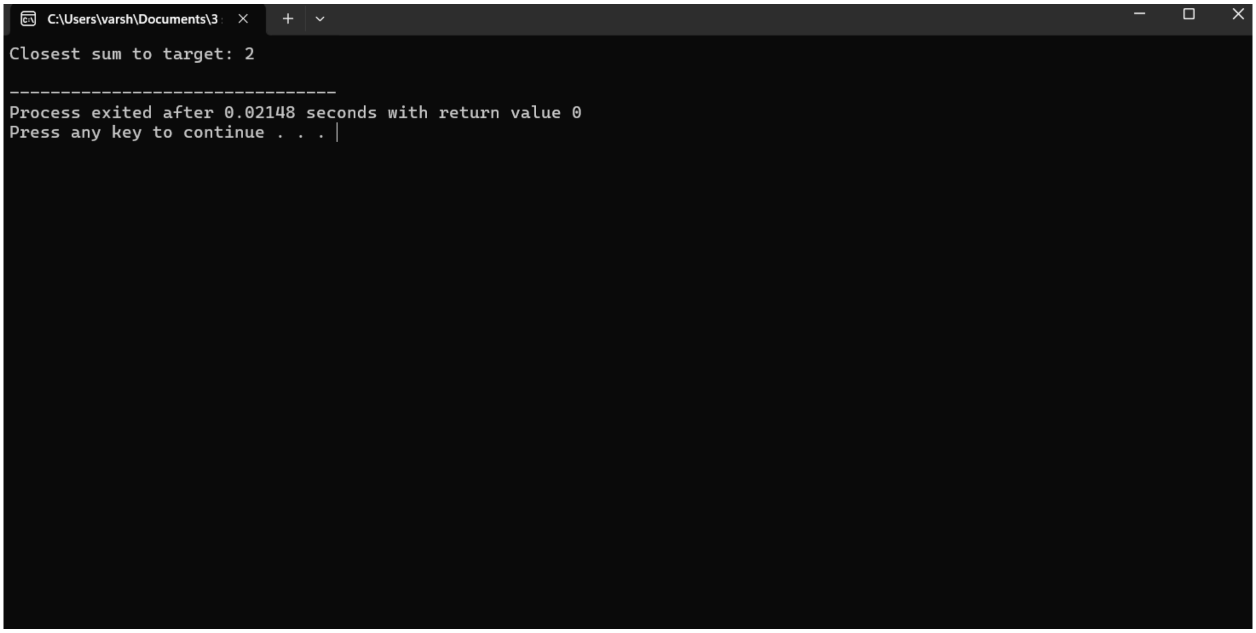
```
Unique triplets that sum to zero:
[-1,-1,2]
[-1,0,1]

--------------------------------
Process exited after 0.01314 seconds with return value 0
Press any key to continue . . .
```

6.3Sum Closest:

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

int compare(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}
```

```
int threeSumClosest(int* nums, int numsSize, int target) {
    if (numsSize < 3) {
        return INT_MAX; // Invalid input
    }

      qsort(nums, numsSize, sizeof(int), compare);

      int closestSum = INT_MAX;
    int minDiff = INT_MAX;

      for (int i = 0; i < numsSize - 2; i++) {
        int left = i + 1;
        int right = numsSize - 1;

        while (left < right) {
            int sum = nums[i] + nums[left] + nums[right];
            int diff = abs(sum - target);

                     if (diff < minDiff) {
                minDiff = diff;
                closestSum = sum;
            }

                       if (sum < target) {
                left++;
            } else if (sum > target) {
                right--;
            } else {
                return target;             }
        }
    }

    return closestSum;
}

int main() {
    int nums[] = {-1, 2, 1, -4};
    int numsSize = sizeof(nums) / sizeof(nums[0]);
    int target = 1;
    printf("Closest sum to target: %d\n", threeSumClosest(nums, numsSize, 
target));
    return 0;
}
```

OUTPUT:

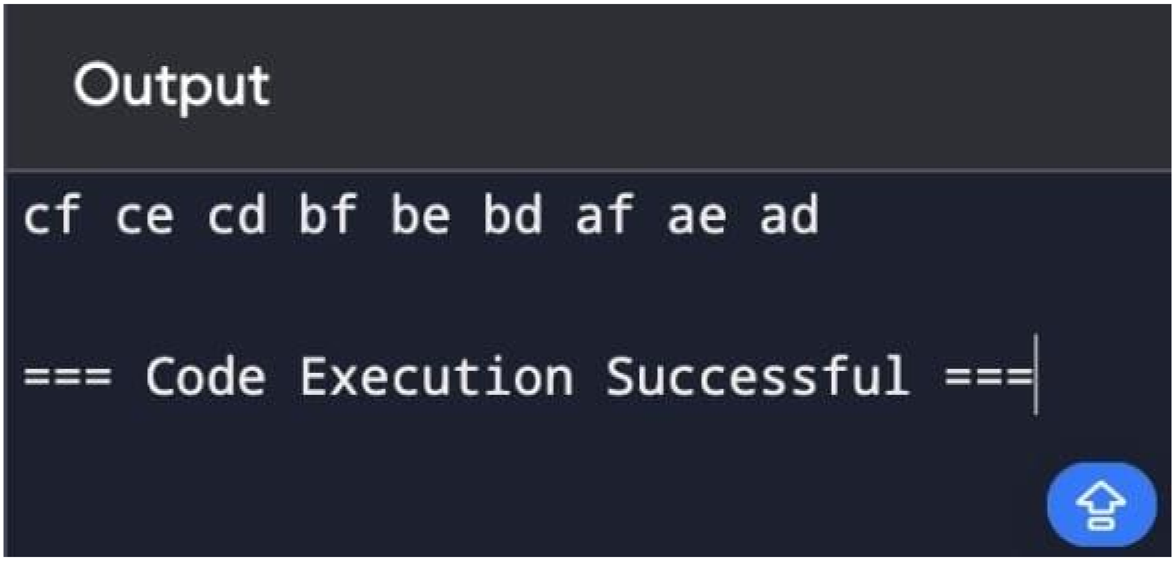


```
Closest sum to target: 2

--------------------------------
Process exited after 0.02148 seconds with return value 0
Press any key to continue . . .
```

7.Letter Combination of a Phone Number:

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

char ** letterCombinations(char * digits, int* returnSize) {
    if (digits == NULL || *digits == '\0') {
        *returnSize = 0;
        return NULL;
    }

       char *map[] = {"abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
       int length = strlen(digits);
    *returnSize = 1;
    for (int i = 0; i < length; i++) {
        int index = digits[i] - '0' - 2;
        *returnSize *= strlen(map[index]);
    }
     char **result = (char **)malloc(*returnSize * sizeof(char *));
    for (int i = 0; i < *returnSize; i++) {
        result[i] = (char *)malloc((length + 1) * sizeof(char));
        result[i][length] = '\0';
    }

        int idx = 0;
    for (int i = 0; i < length; i++) {
        int index = digits[i] - '0' - 2;
        int len = strlen(map[index]);
        int repeat = *returnSize / len;

        for (int j = 0; j < repeat; j++) {
            for (int k = 0; k < len; k++) {
                for (int l = 0; l < repeat; l++) {
                    result[idx][i] = map[index][k];
                    idx++;
                }
            }
        }
        *returnSize /= len;
    }

    return result;
}

int main() {
    char digits[] = "23";
    int returnSize;
    char **result = letterCombinations(digits, &returnSize);
    printf("Letter combinations:\n");
    for (int i = 0; i < returnSize; i++) {
        printf("%s\n", result[i]);
        free(result[i]);
    }

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

OUTPUT



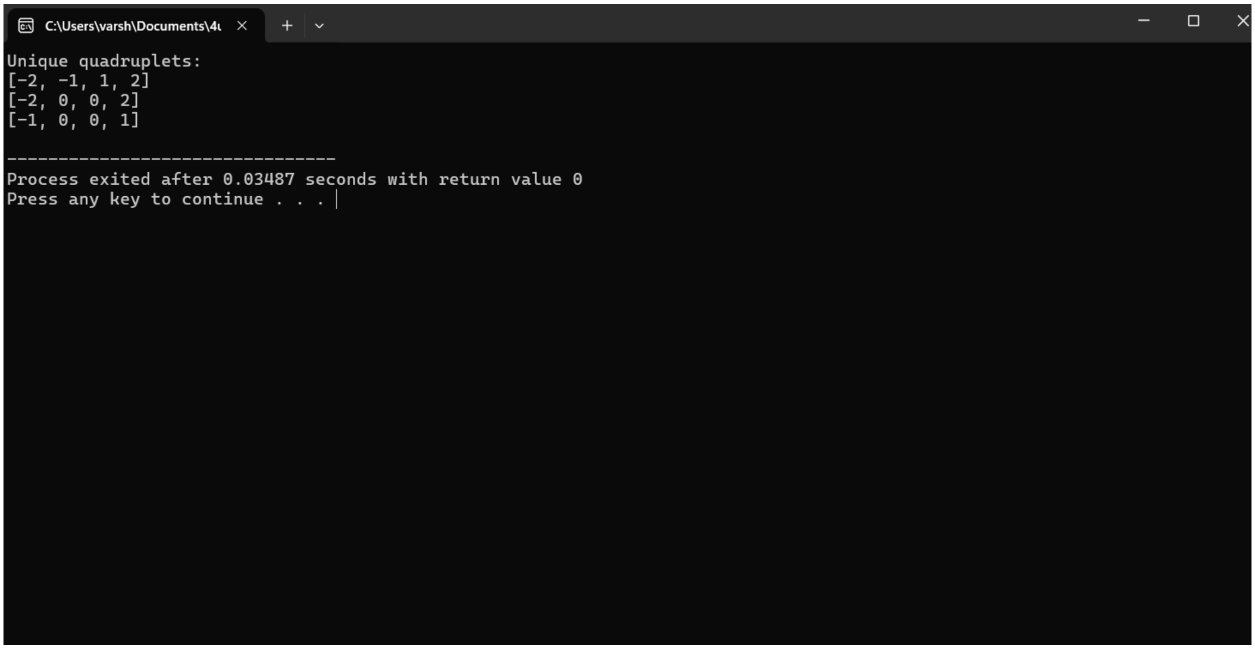
```
cf ce cd bf be bd af ae ad

=== Code Execution Successful ===
```

8.4Sum:

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

int compare(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}

int** fourSum(int* nums, int numsSize, int target, int* returnSize, int**
returnColumnSizes) {
  qsort(nums, numsSize, sizeof(int), compare);

  int maxSize = 1000;
*returnSize = 0;
int **result = (int **)malloc(maxSize * sizeof(int *));
*returnColumnSizes = (int *)malloc(maxSize * sizeof(int));

   for (int i = 0; i < numsSize - 3; i++) {
    if (i > 0 && nums[i] == nums[i - 1]) {
        continue;        }

    for (int j = i + 1; j < numsSize - 2; j++) {
        if (j > i + 1 && nums[j] == nums[j - 1]) {
            continue;
        }

        int left = j + 1;
        int right = numsSize - 1;

        while (left < right) {
            int sum = nums[i] + nums[j] + nums[left] + nums[right];

            if (sum == target) {

                result[*returnSize] = (int *)malloc(4 * sizeof(int));
                result[*returnSize][0] = nums[i];
                result[*returnSize][1] = nums[j];
                result[*returnSize][2] = nums[left];
                result[*returnSize][3] = nums[right];
                (*returnColumnSizes)[*returnSize] = 4;
                (*returnSize)++;


                while (left < right && nums[left] == nums[left + 1]) left++;
                while (left < right && nums[right] == nums[right - 1]) right--;
                left++;
                right--;
                } else if (sum < target) {
                    left++;
                } else {
                    right--;
                }
            }
        }
    }

    return result;
}

int main() {
    int nums[] = {1, 0, -1, 0, -2, 2};
    int numsSize = sizeof(nums) / sizeof(nums[0]);
    int target = 0;
    int returnSize;
    int *returnColumnSizes;
    int **result = fourSum(nums, numsSize, target, &returnSize,
&returnColumnSizes);

    printf("Unique quadruplets:\n");
    for (int i = 0; i < returnSize; i++) {
        printf("[");
        for (int j = 0; j < returnColumnSizes[i]; j++) {
            printf("%d", result[i][j]);
            if (j < returnColumnSizes[i] - 1) {
                printf(", ");
            }
        }
        printf("]\n");
        free(result[i]);
    }

    free(result);
    free(returnColumnSizes);
    return 0;
```

```
}
```

OUTPUT:



```
Unique quadruplets:
[-2, -1, 1, 2]
[-2, 0, 0, 2]
[-1, 0, 0, 1]

--------------------------------
Process exited after 0.03487 seconds with return value 0
Press any key to continue . . .
```

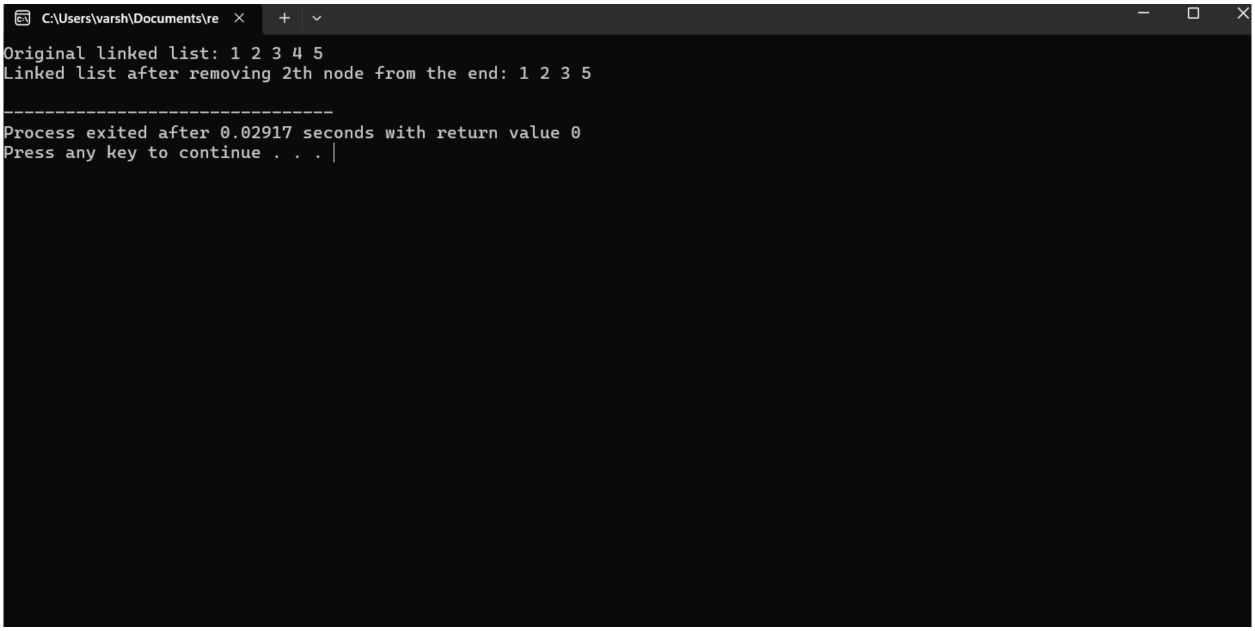
9.Remove the Nth Node From End of List:

```
#include <stdio.h>
#include <stdlib.h>
struct ListNode {
    int val;
    struct ListNode *next;
};

struct ListNode* removeNthFromEnd(struct ListNode* head, int n) {
        struct ListNode dummy;
    dummy.next = head;

        struct ListNode *first = &dummy;
    struct ListNode *second = &dummy;
```

```
      for (int i = 0; i < n; i++) {
        second = second->next;
    }

       while (second->next != NULL) {
        first = first->next;
        second = second->next;
    }

        struct ListNode *temp = first->next;
    first->next = first->next->next;
    free(temp);

    return dummy.next;
}

struct ListNode* createNode(int val) {
    struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->val = val;
    newNode->next = NULL;
    return newNode;
}

void printLinkedList(struct ListNode* head) {
    struct ListNode* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->val);
        temp = temp->next;
    }
    printf("\n");
}
```

```
int main() {
    // Create the 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 linked list: ");
    printLinkedList(head);

       int n = 2;
    head = removeNthFromEnd(head, n);

    printf("Linked list after removing %dth node from the end: ", n);
    printLinkedList(head);

    return 0;
}
```

OUTPUT:

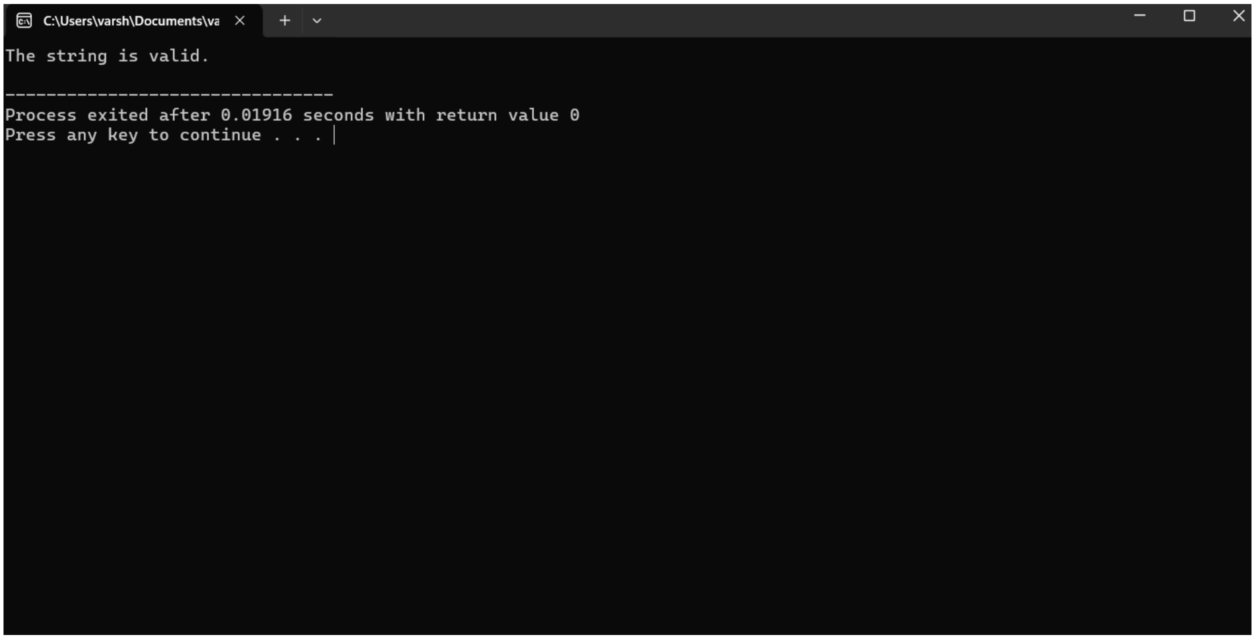


```
Original linked list: 1 2 3 4 5
Linked list after removing 2th node from the end: 1 2 3 5

--------------------------------
Process exited after 0.02917 seconds with return value 0
Press any key to continue . . .
```

10.Valid Parentheses:

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

struct StackNode {
    char data;
    struct StackNode* next;
};

struct StackNode* createNode(char data) {
    struct StackNode* newNode = (struct StackNode*)malloc(sizeof(struct
StackNode));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void push(struct StackNode** top, char data) {
    struct StackNode* newNode = createNode(data);
    newNode->next = *top;
    *top = newNode;
}

char pop(struct StackNode** top) {
    if (*top == NULL) {
        printf("Stack underflow.\n");
        exit(1);
    }
    struct StackNode* temp = *top;
    char data = temp->data;
    *top = (*top)->next;
    free(temp);
    return data;
}

bool isValid(char * s) {
        struct StackNode* stack = NULL;

    // Traverse the string
    for (int i = 0; s[i] != '\0'; i++) {
        char ch = s[i];

        // If the current character is an opening parenthesis, push it onto the stack
        if (ch == '(' || ch == '{' || ch == '[') {
            push(&stack, ch);
        } else {
            // If the stack is empty or the current closing parenthesis does not match
the top of the stack, return false
            if (stack == NULL || (ch == ')' && stack->data != '(') || (ch == '}' &&
stack->data != '{') || (ch == ']' && stack->data != '[')) {
                return false;
            }
            // Otherwise, pop the top of the stack
            pop(&stack);
        }
    }

    // If the stack is empty after processing all characters, return true; otherwise,
return false
    return stack == NULL;
}

int main() {
    // Test the function
    char s[] = "()";
    if (isValid(s)) {
        printf("The string is valid.\n");
    } else {
        printf("The string is not valid.\n");
    }
    return 0;
}
```

OUTPUT:



```
The string is valid.

--------------------------------
Process exited after 0.01916 seconds with return value 0
Press any key to continue . . .
```

TIME COMPLEXITY:

1.O(n)
2.O(1)
3.O(n)
5.O(n^2)
6.O(n^2)
8.O(n^3)
9.O(n)
10.O(n)