**Лабораторна робота № 6. Подання**

**рядків в пам’яті**

***Мета:*** Отримати практичні навички та закріпити знання про можливі подання даних типу рядок та про операції над рядками.

**1 Вимоги**

**1.1 Розробник**

* Макаренко Владислав Олександрович
* Студент 1-го курсу
* Групи КІТ-120а

**1.2 Загальне завдання**

Написати програму, в якій передбачити виконання вказаної операції над рядками за умови подання рядків у пам’яті двома способами.

**1.2 Загальне завдання**

**Спосіб подання рядка**

1) Вектор з керованою довжиною рядка

2) Символьно-зв’язане подання двоспрямованим списком.

**Функція**

Видалити з рядка *s* підрядок, починаючи з позиції *n* довжиною *k* символів.

**2 Описи програм**

**Код програми**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <time.h>  
  
struct Node {  
 char a;  
 struct Node \*next;  
 struct Node \*prev;  
};  
  
struct List {  
 struct Node \*head;  
 struct Node \*tail;  
};  
  
struct Node \*list\_new\_node(char *a*)  
{  
 struct Node \*node = malloc(sizeof(struct Node));  
 node->a = *a*;  
 node->next = NULL;  
 node->prev = NULL;  
 return node;  
}  
  
void list\_add\_end(struct List \**ls*, char *a*)  
{  
 struct Node \*node = list\_new\_node(*a*);  
  
 node->prev = *ls*->tail;  
 if (*ls*->tail)  
 *ls*->tail->next = node;  
 else  
 *ls*->head = node;  
 *ls*->tail = node;  
}  
  
int list\_remove(struct List \**ls*, int *index*)  
{  
 if (*index* < 0)  
 return 0;  
 struct Node \*p = *ls*->head;  
 if (*index* == 0) {  
 if (!*ls*->head)  
 return 0;  
 *ls*->head = *ls*->head->next;  
 if (p->next)  
 p->next->prev = NULL;  
 else  
 ls->tail = NULL;  
 free(p);  
 return 1;  
 }  
 while (*index* > 0) {  
 if (!p)  
 return 0;  
 p = p->next;  
 *index*--;  
 }  
 p->prev->next = p->next;  
 if (p->next)  
 p->next->prev = p->prev;  
 else  
 *ls*->tail = p->prev;  
 free(p);  
 return 1;  
}  
  
struct List \*list\_new()  
{  
 struct List \*ls = malloc(sizeof(struct List));  
 ls->head = NULL;  
 ls->tail = NULL;  
 return ls;  
}  
  
void list\_show(struct List \**ls*, int *reverse*) {  
 if (!reverse)  
 for (struct Node \*node = ls->head; node != NULL; node = node->next)  
 printf("%15c\n", node->a);  
 else  
 for (struct Node \*node = ls->tail; node != NULL; node = node->prev)  
 printf("%15c\n", node->a);  
 printf("\n");  
}  
  
  
char \* DeleteStr(char\* *string*, int *start\_position*, int *end\_position*) {  
 if ((*end\_position* - *start\_position*) == 0 && *start\_position* >= strlen(*string*)) {  
 printf("nothing to delete");  
 return 0;  
 }  
 char \* new\_string = malloc((strlen(*string*)) \* sizeof(char ));  
 memcpy(new\_string, *string*, *start\_position* \* sizeof(char));  
 memcpy(new\_string + *start\_position*, *string* + *end\_position*, (strlen(*string*) - (*end\_position* - *start\_position*)) \* sizeof(char ));  
 printf("New string: %s\n", new\_string);  
 return new\_string;  
}  
  
  
int main() {  
 char \*str = malloc(128 \* sizeof (char));  
 printf("Enter the string: ");  
 scanf("%s", str);  
 printf("Enter position: ");  
 int start\_pos;  
 scanf("%d", &start\_pos);  
 int len;  
 printf("Enter length: ");  
 scanf("%d", &len);  
 int end\_pos = start\_pos + len;  
 clock\_t start = clock();  
 DeleteStr(str, start\_pos, end\_pos);  
 clock\_t end = clock();  
 double time\_spent1 = (double) (end - start) / CLOCKS\_PER\_SEC;  
 printf("Час затрачений на виконання функції по видаленю підрядка з рядка у векторі з керованою довжиною рядка: %.10f second(s)\n\n", time\_spent1);  
  
 int n;  
 printf("Enter count of elements: ");  
 scanf("%d", &n);  
 struct List \*ls = list\_new();  
 for (int i = 0; i<n; i++) {  
 char symb;  
 printf("Enter symbol: ");  
 scanf("%s", &symb);  
 list\_add\_end(ls, symb);  
 }  
 int side;  
 printf("Enter the direction of the list (0 or 1): ");  
 scanf("%d", &side);  
 printf("Original list: \n");  
 list\_show(ls, side);  
 int index;  
 printf("Enter the index of the element to be deleted: ");  
 scanf("%d", &index);  
 clock\_t begin = clock();  
 list\_remove(ls, index);  
 printf("New list: \n");  
 list\_show(ls, side);  
 clock\_t finish = clock();  
 double time\_spent2 = (double)(finish - begin) / CLOCKS\_PER\_SEC;  
 printf("Час затрачений на виконання функції по видаленю підрядка з рядка у символьно-зв’язаному поданню двоспрямованого списку: %.10f second(s) \n", time\_spent2);  
  
}

**Результати виконання програми**

За алгоритмом коду задаємо рядки для роботи з ними, та виконуємо фунцію з індивідуального завдання(див. рис. 1).

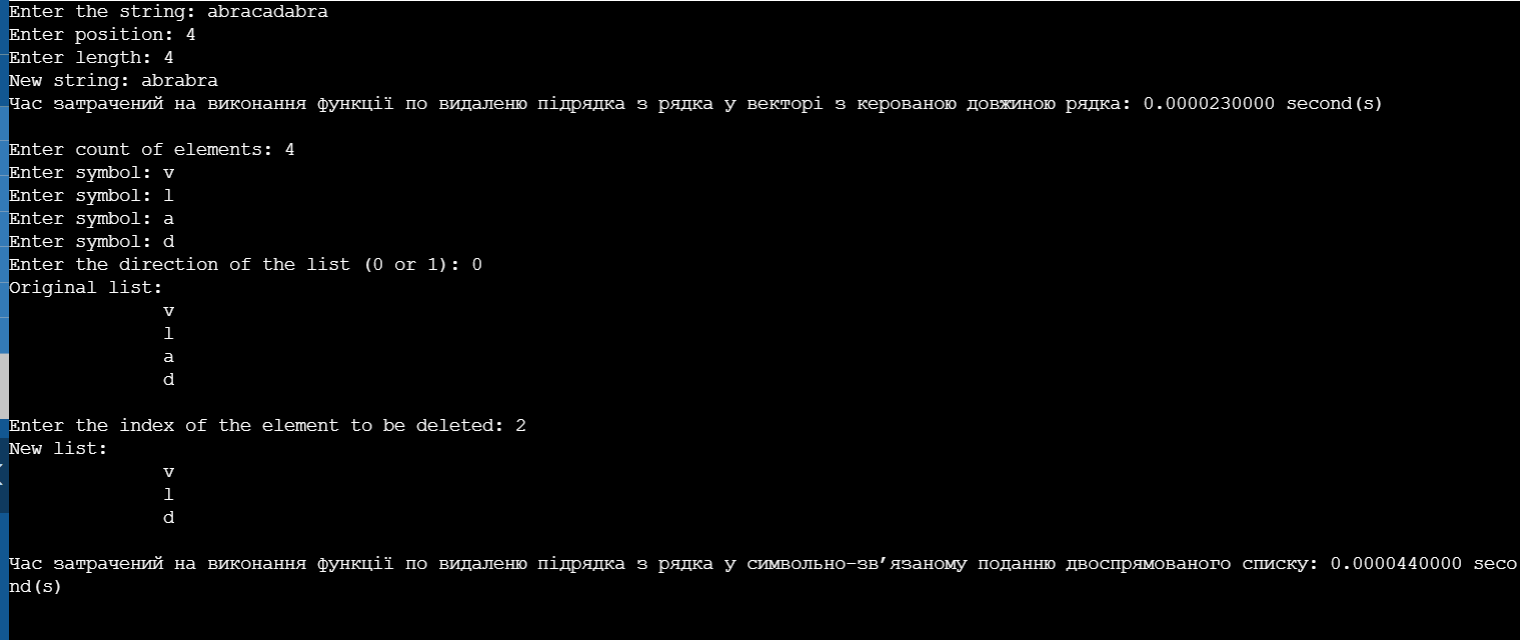


Рисунок 1 – Результати роботи першого кроку

**3 Порівняння часу**

Порівняємо часу виконання функції у векторі(зліва) та у списку(справа) за допомогою функції *clock()* (див. рис.2).

|  |  |  |
| --- | --- | --- |
|  | **<** |  |

Рисунок 2 – Порівняння часу

Як бачимо з (рис.2) задана функція працює бистріше для вектора, а не для списку.

**Висновок:** на цій лабораторній роботі ми отримали практичні навички та закріпили знання про можливі подання даних типу рядок та про операції над рядками.