

## Лабораторна робота №18

**Тема:** Динамічні масиви.

**Мета:** Навчитися використовувати динамічні масиви.

### Індивідуальне завдання

розробити функцію, яка реалізує вставку в рядок “s” другий рядок “s2” в “i”-у позицію рядка “s”. Наприклад, `insert("abrakadabra", "ТЕХТ2", 4)` повинна створити рядок “abraТЕХТ2kadabra”;

розробити функцію видалення з рядка “s” усіх символів з індексами в заданому діапазоні. Наприклад, `reduce("abrakadabra", 4, 8)` повинна створити рядок “abrara” (без підрядка kadab).

за допомогою функцій `memcpy`, `memset` створити функції додання та видалення елементів з динамічного масиву вашої прикладної області  
додати модульні тести, що демонструють коректність розроблених функцій

### Хід роботи

1. Функції для вставки та вирізки.

```

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string.h>
using namespace std;

void cut(char* mass, int indexStart, int indexEnd);
char* insert(char* mass, char* mass2, int index);

int main() {

    char mass[] = "karamba";
    char mass2[] = "TEST";
    char* result = insert(mass, mass2, 2);
    cut(result, 4, 10);

    return 0;
}

char* insert(char* mass, char* mass2, int index) { // функція для вставки

    int size = strlen(mass) + strlen(mass2);
    char* s = (char*)malloc((size + 1) * sizeof(char));

    for (int i = 0; i < index; i++)
        s[i] = mass[i];
    for (int i = index; i < strlen(mass2) + index; i++)
        s[i] = mass2[i - index];
    for (int i = strlen(mass2) + index; i < size; i++)
        s[i] = mass[i - strlen(mass2)];

    s[size] = '\0';

    cout << s << endl;

    return s;
}

void cut(char* mass, int indexStart, int indexEnd) { // функція для вирізки

    int size = strlen(mass) - (indexEnd - indexStart + 1);
    char* s = (char*)malloc((size + 1) * sizeof(char));

    for (int i = 0; i < indexStart; i++)
        s[i] = mass[i];

```

Рисунок 1.1 - Функція main і функція insert.

```

void cut(char* mass, int indexStart, int indexEnd) { // функція для вирізки

    int size = strlen(mass) - (indexEnd - indexStart + 1);
    char* s = (char*)malloc((size + 1) * sizeof(char));

    for (int i = 0; i < indexStart; i++)
        s[i] = mass[i];

    for (int i = indexEnd + 1; i < strlen(mass); i++)
        s[i - (indexEnd - indexStart + 1)] = mass[i];
    s[size] = '\0';
    cout << s ;
    return;
}

```

Рисунок 1.2 - Функція cut.

## 2. Результат коду.

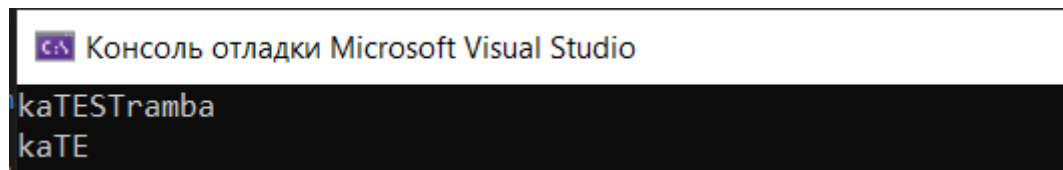


Рисунок 2 - Код працює.

## 3. Функції для додавання та видалення елементів, тести.

```

int addEmployee(struct Employee* employee, int size) { // Функція для додання елементів

    size++;
    Employee* mass = (Employee*)malloc(size * sizeof(Employee));
    memcpy(mass, employee, sizeof(Employee) * (size - 1));
    mass[size - 1] = create();
    memcpy(employee, mass, sizeof(Employee) * size);

    return size;
}

int removeEmployee(struct Employee* employee, int size, int index) { // Функція для видалення елементів

    size--;
    Employee* mass = (Employee*)malloc(size * sizeof(Employee));
    for (int i = 0; i < size; i++)
    {
        if (i < index)
        {
            mass[i] = employee[i];
        }
        else
        {
            mass[i] = employee[i + 1];
        }
    }
    memcpy(employee, mass, sizeof(Employee) * size);

    return size;
}

```

Рисунок 3.1 - Функції addEmployee та removeEmployee.

```

int testRemove() { // Тест для удаления

    printf("Here the remove test works.\n");

    int size = 3;
    Employee* mass = (Employee*)malloc(size * sizeof(Employee));
    for (int i = 0; i < size; i++)
    {
        mass[i] = create();
    }

    int tmpWorkExp0 = mass[0].workExperience;
    int tmpWorkExp2 = mass[2].workExperience;
    int Test1;

    size = removeEmployee(mass, size, 1);

    if (tmpWorkExp0 == mass[0].workExperience && tmpWorkExp2 == mass[1].workExperience && size == 2)
    {
        Test1 = 1;
        printf("Remove was succesful.\n");
    }
    else
    {
        Test1 = 0;
        printf("Remove was not succesful.\n");
    }
    return Test1;
}

```

Рисунок 3.2 - Функції testRemove.

```

int testAdd() { // Тест для добавления

    printf("Here the add test works.\n");

    int size = 2;
    Employee* mass = (Employee*)malloc(size * sizeof(Employee));
    for (int i = 0; i < size; i++)
    {
        mass[i] = create();
    }
    size = addEmployee(mass, size);
    int Test2;
    if (mass[size - 1].workExperience > 0 && size == 3)
    {
        Test2 = 1;
        printf("Add was succesful.\n");
    }
    else
    {
        Test2 = 0;
        printf("Add was not succesful.\n");
    }
    return Test2;
}

```

Рисунок 3.3 - Функції testAdd.

```
#include "pch.h"
#include "CppUnitTest.h"
using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest2
{
    TEST_CLASS(UnitTest2)
    {
    public:
        TEST_METHOD(TestMethod1)
        {
            int answer = 1;
            Assert::AreEqual(testAdd(), answer);
        }
        TEST_METHOD(TestMethod2)
        {
            int answer = 1;
            Assert::AreEqual(testRemove(), answer);
        }
    };
}
```

Рисунок 3.4 - Тести.

#### 4. Результат коду.

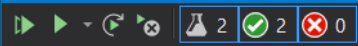
Обозреватель тестов			
			
Тестирование	Длительность	Признаки	Сообщение об ошибке
UnitTest2 (2)	< 1 мс		
UnitTest2 (2)	< 1 мс		
UnitTest2 (2)	< 1 мс		
TestMethod1	< 1 мс		
TestMethod2	< 1 мс		

Рисунок 4 - Код працює.

**Висновок:** Навчився використовувати динамічні масиви.