**Звіт**

Автор: Момот Р. КІТ-119а

Дата: 10.04.2020

**ЛАБОРАТОРНА РОБОТА № 6. ПОДАННЯ РЯДКІВ У ПАМ’ЯТІ**

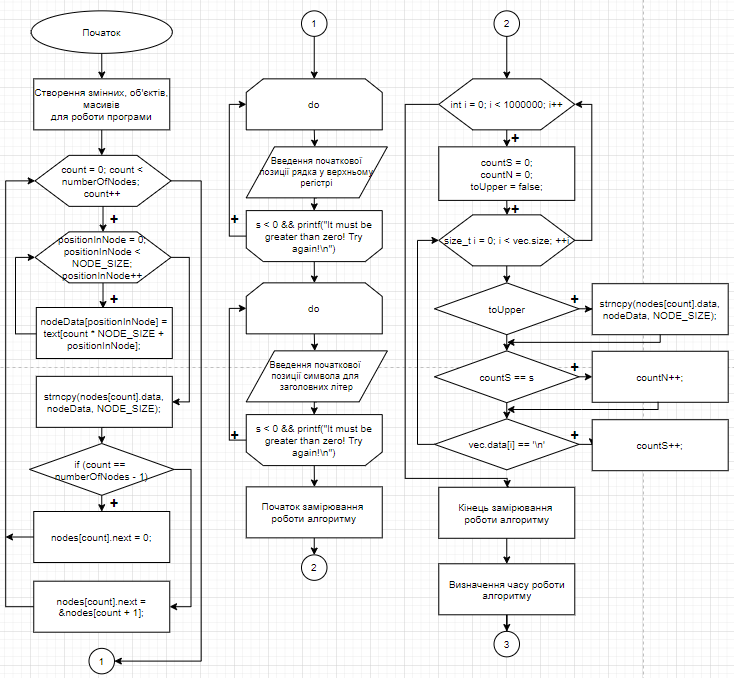
**Мета:** Отримати практичні навички та закріпити знання про можливі подання даних типу рядок та про операції над рядками.

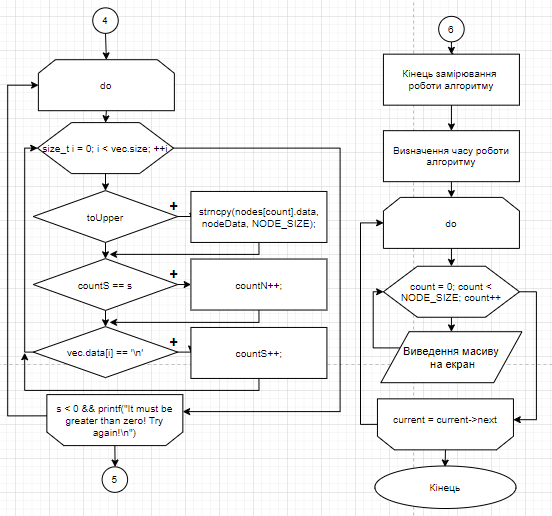
Індивідуальне завдання

Написати програму, в якій передбачити виконання вказаної операції над рядками за умови подання рядків у пам’яті двома способами. Порівняти подання рядків вказаними способами (обсяг пам’яті, час виконання функції).

Функції: Замінити в рядку s, починаючи з позиції n, всі малі літери на великі.

Спосіб подання рядка: Вектор з керованою довжиною рядка (дескриптор), блочно-зв’язне подання з фіксованою довжиною.

Блок-схема алгоритму програми

Текст програми

#define \_CRT\_SECURE\_NO\_WARNINGS

#define NODE\_SIZE 4

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <time.h>

typedef struct Vector {

char\* data;

size\_t size, capacity;

} Vector;

typedef struct Node {

char data[NODE\_SIZE];

struct Node\* next;

} Node;

void main()

{

const int textSize = 120;

int numberOfNodes = (textSize + NODE\_SIZE - 1) / NODE\_SIZE;

int count, positionInNode;

char nodeData[NODE\_SIZE];

bool toUpper = false;

int s = 2, n = 5, countS, countN;

Node\* current;

//Measuring time values to work with

clock\_t start1, start2, end1, end2;

long double cpu\_time\_used1, cpu\_time\_used2;

//Creating text data to work with

char text[] = "So\nLet's test the different methods of saving data\nThe goal is to compare time & memory usage\nLet the experiment begin!";

//Entering text data to the Vector representation

Vector vec = { text, textSize, textSize };

//Enter text data to the Node representation

Node\* nodes; //array is just for easy initialization & data entering. For all other purposes we'll use the node system itself

nodes = (Node\*)malloc(numberOfNodes \* sizeof(Node));

for (count = 0; count < numberOfNodes; count++)

{

for (positionInNode = 0; positionInNode < NODE\_SIZE; positionInNode++)

{

nodeData[positionInNode] = text[count \* NODE\_SIZE + positionInNode];

}

strncpy(nodes[count].data, nodeData, NODE\_SIZE);

if (count == numberOfNodes - 1)

nodes[count].next = 0;

else

nodes[count].next = &nodes[count + 1];

}

//User interface for entering values

printf("\t\tText: \n%s\n\n", text);

do

{

printf("Enter the starting line position for uppercasing:\t");

scanf("%d", &s);

s--; // because [0] is 1-st

} while (s < 0 && printf("It must be greater than zero! Try again!\n"));

do

{

printf("Enter the starting character position for uppercasing:\t");

scanf("%d", &n);

n--; // because [0] is 1-st

} while (n < 0 && printf("It must be greater than zero! Try again!\n"));

printf("\n\n");

//Measuring time for Vector representation:

start1 = clock();

for (int i = 0; i < 1000000; i++)

{

countS = 0;

countN = 0;

toUpper = false;

for (size\_t i = 0; i < vec.size; ++i)

{

toUpper = (countS == s && countN >= n);

if (toUpper)

vec.data[i] = toupper(vec.data[i]);

if (countS == s)

countN++;

if (vec.data[i] == '\n')

countS++;

}

}

end1 = clock();

cpu\_time\_used1 = ((double)(end1 - start1)) / CLOCKS\_PER\_SEC;

//Printing results for Vector representation

printf("\t\tVector representation\n\nUsed memory: %d bytes\nUsed time for million iterations: %.3f seconds\n\nOutput:\n", sizeof(vec) + sizeof(\*vec.data) \* vec.capacity, cpu\_time\_used1);

for (size\_t i = 0; i < vec.size; ++i)

printf("%c", vec.data[i]);

printf("\n\n\n");

//Measuring time for Node representation:

start2 = clock();

for (int i = 0; i < 1000000; i++)

{

current = &nodes[0];

countS = 0;

countN = 0;

toUpper = false;

do

{

for (count = 0; count < NODE\_SIZE; count++)

{

toUpper = (countS == s && countN >= n);

if (toUpper)

current->data[count] = toupper(current->data[count]);

if (countS == s)

countN++;

if (current->data[count] == '\n')

countS++;

}

} while (current = current->next);

}

end2 = clock();

cpu\_time\_used2 = ((double)(end2 - start2)) / CLOCKS\_PER\_SEC;

//Printing results for Node representation

printf("\t\tNode representation\n\nUsed memory: %d bytes\nUsed time for million iterations: %.3f seconds\n\nOutput:\n", sizeof(Node) \* numberOfNodes, cpu\_time\_used2);

current = &nodes[0];

do

{

for (count = 0; count < NODE\_SIZE; count++)

printf("%c", current->data[count]);

} while (current = current->next);

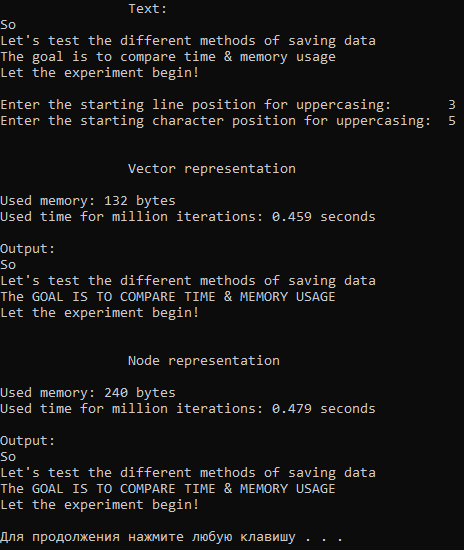
free(nodes);

printf("\n\n");

system("pause");

}

Результати роботи програми



Висновок

У результаті роботи програми було розроблено програму, в якій було передбачено заміну усіх літер з маленьких на великі, починаючи з рядка s та позиції n у рядку, за умови подання рядків у пам’яті двома способами: вектор з керованою довжиною рядка (дескриптор) та блочно-зв’язне подання з фіксованою довжиною. Час роботи функції з односпрямованим списком менше, ніж час роботи функції з двоспрямованим. Це відбувається тому, що у двоспрямованому списку на один елемент списку більше (незначущий кінцевий елемент) і тому функція працює довше.

Відповіді на питання

1. Що таке «рядок», які його властивості?

Рядок - це лінійно впорядкована послідовність символів, що належать кінцевому безлічі символів, званого алфавітом.

Рядки мають наступні важливими властивостями:

* Їх довжина, як правило, змінна, хоча алфавіт фіксований;
* Зазвичай звернення до символів рядка йде з якогось одного кінця послідовності, тобто важлива впорядкованість цієї послідовності, а не її індексація;
* Найчастіше метою доступу до рядка є не окремий її елемент (хоча це теж не виключається), а деякий ланцюжок символів в рядку.

1. Які визначені базові операції над рядками?

Базовими операціями над рядками є:

* Визначення довжини рядка.
* Присвоювання.
* Порівняння рядків.
* Конкатенація (зчеплення) рядків.
* Виділення підрядка.
* Пошуку входження.

1. За яким алгоритмом виконується порівняння рядків?

Порівняння рядків проводиться за такими правилами: порівнюються перші символи двох рядків. Якщо символи не рівні, то рядок, що містить символ, місце якого в алфавіті ближче до початку, вважається меншою. Якщо символи рівні, порівнюються другі, треті і т.д. символи. При досягненні кінця однієї з рядків, рядок меншої довжини вважається меншою. У разі рівного розподілу довжин рядків і попарним рівність всіх символів в них рядки вважаються рівними.

1. Що таке «конкатенація» рядків?

Конкатенація рядків – це операція, яка зчеплює два рядка і результатом цієї операції є рядок, довжина якого дорівнює сумарній довжині рядків-операндів, а значення відповідає значенню першого операнда, за яким безпосередньо слід значення другого операнда.

1. Які відомі способи подання рядків у пам’яті?

Векторне подання рядків:

* + - * Вектор постійної довжини;
      * Вектор змінної довжини з ознакою кінця;
      * Вектор змінної довжини з лічильником;
      * Вектор з керованою довжиною.

Зв’язне уявлення рядків;

Символьно-зв'язне уявлення рядків:

* + - * Односпрямований лінійний список;
      * Двонаправлений лінійний список.

Блочно-зв'язне уявлення рядків:

* + - * Багатосимвольні ланки фіксованої довжини;
      * Багатосимвольні ланки змінної довжини;
      * Багатосимвольні ланки з лічильником з керованою довжиною.

1. Який алгоритм видалення частини рядка?

На основі базових операцій над рядками можуть бути реалізовані і будь-які інші, навіть складні операції. Наприклад, операція видалення з рядка символів з номерами від ***n1*** до ***n2***, включно, може бути реалізована як послідовність наступних кроків:

* Виділення з вихідної рядки підрядка, починаючи з позиції ***1***, довжиною ***(n1-1)*** символів;
* Виділення з вихідної рядки підрядка, починаючи з позиції ***(n2+1)***, довжиною, що дорівнює довжині початкового рядка мінус ***n2***;
* Зчеплення підрядків, отриманих на попередніх етапах.

1. Навіщо створюється дискрипрор рядка?

Наприклад, пам'ять під вектор з керованою довжиною відводиться при створенні рядка і його розмір і розміщення залишаються незмінними весь час існування рядки. Для такого вектора створюється дескриптор (описувач), який зберігає у собі максимальну довжину рядка, поточну довжину, а також покажчик на дані рядка.

1. Які переваги та недоліки подання рядків масивом?

Подання рядків у вигляді векторів дозволяє працювати з рядками, розміщеними в статичної пам'яті. Крім того, векторне подання дозволяє легко звертатися до окремих символів рядка як до елементів вектора - за індексом.

**Вектор постійної довжини** відводить фіксовану кількість байт, в яку записуються символи рядка. Якщо рядок менше відведеного під неї вектора, то зайві місця заповнюються пробілами, а якщо рядок виходить за межі вектора, то зайві (зазвичай справа рядки) символи повинні бути відкинуті.

**Вектор змінної довжини** **з ознакою кінця** використовує на 1 символ більше для ознаки кінця.

**Вектор змінної довжини з лічильником** потребує додаткової пам’яті для лічильника. При використанні лічильника символів можливий довільний доступ до символів в межах рядка, оскільки можна легко перевірити, що звернення не виходить за межі рядка.

**Вектор з керованою довжиною** відводить пам'ять при створенні рядка і її розмір і розміщення залишаються незмінними весь час існування рядки. Для такого вектора створюється дескриптор (описувач), що збільшує кількість виділеної пам'яті.

1. Які переваги та недоліки подання рядків односпрямованим списком?

Є можливість зберігати елементи рядку по «всій» пам’яті, дефрагментуючи дані, тому що кожен елемент містить код символу і покажчик на наступний елемент, але це потребує додаткової пам’яті (на кожен символ рядка необхідний один покажчик, який зазвичай займає 2-4 байта). Також до недоліків відноситься можливість переміщення по рядку лише в одному напрямку.

1. Які переваги та недоліки подання рядків двоспрямованим списком?

Окрім переваг односпрямованого списка, є можливість двостороннього руху уздовж списку, що може значно підвищити ефективність виконання деяких строкових операцій. При цьому на кожен символ рядка необхідно два покажчика, тобто 4-8 байт.