КІТ-119а

Автор: Чугунов В.Ю

Дата : 01.05.2020

**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.

**Індивідуальне завдання**

Написати програму, в якій передбачити виконання вказаної операції над рядками за умови подання рядків у пам’яті двома способами. Вектор змінної довжини з ознакою кінця рядка та блочно-зв’язне подання з фіксованою довжиною. Порівняти рядки *s1* та *s2*.

**Текст програми**

#include <iostream>

#include <chrono>

using namespace std;

const int BlkSize = 8; // число символов в блоке

typedef struct Block\* BlkPtr; // указатель на блок

struct Block // блок

{

char chFirst, chLast; // номера 1-го и последнего символов

char StrData[BlkSize]; // символы

BlkPtr next; // указатель на следующий блок

};

// Представление строк вектором с управляемой длиной

typedef struct StrNode\* String; // тип - СТРОКА ПЕРЕМЕННОЙ ДЛИНЫ

struct StrNode // дескриптор строки

{

unsigned short MaxLen, Len; // максимальная и текущая длины

char\* str; // указатель на данные строки

BlkPtr first, last; // указ.на 1-й и последний блоки

};

// \*\*\* Внутр. функция-выделение нового блока

BlkPtr NewBlock()

{

BlkPtr curr = new Block; // выделение памяти

curr->next = NULL; // начальные значения

curr->chFirst = 0;

curr->chLast = 0;

return curr;

}

//\*\*\* Внутр.функция - освобождение цепочки блока, начиная со значения bl

BlkPtr DeleteBlock(BlkPtr bl)

{

BlkPtr buf;

while (bl) // движение по цепочке с освобождением памяти

{

buf = bl;

bl = bl->next;

delete buf;

buf = NULL;

}

return NULL; // всегда возвращает NULL

}

// \*\*\* создание строки

String NewStr2()

{

String str = new StrNode; // выделение памяти для дескриптора

str->Len = 0; // занесение в дескриптор начальных значений

str->first = str->last = NULL;

return str;

}

// \*\*\* уничтожение строки

void DeleteStr2(String str)

{

str->first = DeleteBlock(str->first); // уничтожение блоков

delete str; // уничтожение дескриптора

}

// \*\*\* Сравнение строк - возвращает:

int CompStr2(String s1, String s2)

{

char bi1, bi2;

BlkPtr cp1, cp2;

cp1 = s1->first;

cp2 = s2->first;

bi1 = cp1->chFirst;

bi2 = cp2->chFirst;

while (cp1 && cp2)// цикл, пока не будет достигнут конец одной из строк

{ // если соответств. символы не равны, ф-ция заканчивается

if (cp1->StrData[bi1] > cp2->StrData[bi2]) return 1;

else if (cp1->StrData[bi1] < cp2->StrData[bi2]) return -1;

bi1++; // к следующему символу в s1

if (bi1 > cp1->chLast)

{

cp1 = cp1->next;

bi1 = cp1->chFirst;

}

bi2++; // к следующему символу в s2

if (bi2 > cp2->chLast)

{

cp2 = cp2->next;

bi2 = cp2->chFirst;

}

}

// мы дошли до конца одной из строк,

// строка меньшей длины считается меньшей

if (s1->Len < s2->Len)

{

cout << "s1 < s2" << endl;

return -1;

}

else if (s1->Len > s2->Len)

{

cout << "s1 > s2" << endl;

return 1;

}

else

{

cout << "s1 = s2" << endl;

return 0;

}

}

/////////////////////////////////////////////ВТОРОЙ СПОСОБ ПРЕДСТАВЛЕНИЯ///////////////////////////////////////////////////

// Создание строки; len - максимальная длина строки;

// ф-ция возвращает указатель на дескриптор строки

String NewStr(int len)

{

String s = new StrNode; // выделение памяти для дескриптора

s->MaxLen = len; // занесение в дескриптор начальных значений

s->Len = 0;

s->str = new char[len + 1]; // выделение памяти для данных

return s;

}

// уничтожение строки

void DeleteStr(String s)

{

if (s)

{

if (s->Len > 0) delete[] s->str; // уничтожение данных

delete s; // уничтожение дескриптора

}

}

void CopyStr2(String dest, String source)

{

BlkPtr cp1, cp2, // адреса текущих блоков для dest и source

pp; // адрес предыдущего блока для dest

cp1 = dest->first;

cp2 = source->first;

pp = NULL;

if (source->Len == 0) // если s2 пустая, освобождается вся s1

{

dest->first = DeleteBlock(dest->first);

dest->last = NULL;

}

else

{

while (cp2) // перебор блоков source

{

if (cp1 == NULL) // если в dest больше нет блоков

{

cp1 = NewBlock(); // выделяется новый блок для dest

if (dest->first == NULL) dest->first = cp1;

else if (pp) pp->next = cp1;

}

\*cp1 = \*cp2; // копирование блока

// к следующему блоку

pp = cp1; cp1 = cp1->next; cp2 = cp2->next;

}

dest->last = pp; // последний блок

// если в s1 остались лишние блоки - освободить их

pp->next = DeleteBlock(pp->next);

} // end else

dest->Len = source->Len;

}

void CopyStr(String dest, string source)

{

int len = 0;

while (len < dest->MaxLen && source[len])

{

dest->str[len] = source[len];

len++;

}

dest->str[len] = '\0';

dest->Len = len;

}

// Сравнение строк - возвращает: 0, если s1=s2; 1 - если s1>s2;

// -1 - если s1<s2

char CompStr(String s1, String s2)

{

int i = 0; // индекс текущего символа

// цикл, пока не будет достигнут конец одной из строк

while (i < s1->Len && i < s2->Len)

{

// если i-е символы не равны, функция заканчивается

if (s1->str[i] > s2->str[i])

{

cout << "\nstrings are not equal\n";

break;

}

else if (s1->str[i] < s2->str[i])

{

cout << "\nstrings are not equal\n";

break;

}

i++; // переход к следующему символу

}

// если выполнение дошло до этой точки, то найден конец одной из

// строк, и все сравненные до сих пор символы были равны;

// строка меньшей длины считается меньшей

if (s1->Len < s2->Len)

{

cout << "length of s1 > length of s2" << endl;

return -1;

}

else if (s1->Len > s2->Len)

{

cout << "s1 < s2" << endl;

return 1;

}

else

{

cout << "s1 = s2" << endl;

return 0;

}

}

int main()

{

string s1 = "ppppp";

string s2 = "ppppp";

cout << "String s1:" << s1 << endl;

cout << "String s2:" << s2 << endl;

cout << "Compare two strings s1 & s2 with vector way:" << endl;

String S1 = NewStr(s1.length());

String S2 = NewStr(s2.length());

CopyStr(S1, s1);

CopyStr(S2, s2);

auto begin = chrono::steady\_clock::now();

CompStr(S1, S2);

auto end = chrono::steady\_clock::now();

auto elapsed\_ms = chrono::duration\_cast<chrono::nanoseconds>(end - begin);

cout << "\nFunction work time is vector way: " << elapsed\_ms.count() << "ns" << endl;

cout << "===============================================================";

DeleteStr(S1);

DeleteStr(S2);

cout << endl << endl;

/////////////////////////////////////////////////////////////////////////////////////

cout << "Compare two strings s1 & s2 with Fixed-length block view:" << endl;

cout << "String s1:" << s1 << endl;

cout << "String s2:" << s2 << endl;

S1 = NewStr2();

S2 = NewStr2();

String S3 = NewStr2();

String S4 = NewStr2();

CopyStr(S1, s1);

CopyStr(S2, s2);

CopyStr2(S3, S1);

CopyStr2(S4, S2);

auto begin2 = chrono::steady\_clock::now();

CompStr2(S3, S4);

auto end2 = chrono::steady\_clock::now();

auto elapsed\_ms2 = chrono::duration\_cast<chrono::nanoseconds>(end2 - begin2);

cout << "\nFunction work time with Fixed-length block view : " << elapsed\_ms.count() << "ns" << endl;

cout << "===============================================================";

DeleteStr2(S1);

DeleteStr2(S2);

cout << endl << endl;

return 0;

}

Схема алгоритму програми

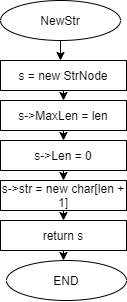


Рисунок 1 – Функція створення списку векторним способом

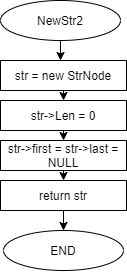


Рисунок 2 – Функція створення списку блочно-зв’язним способом

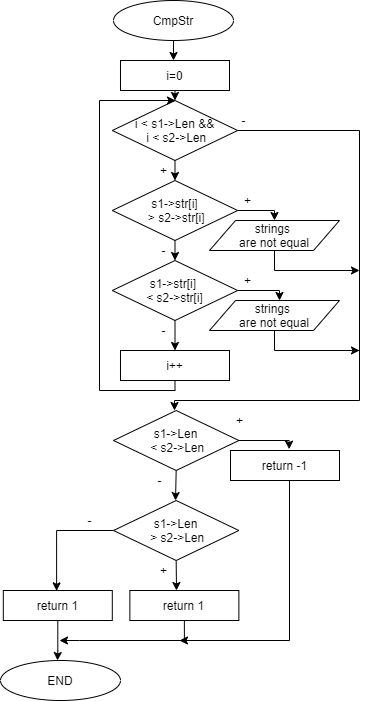


Рисунок 3– Функція порівняння рядків першим способом

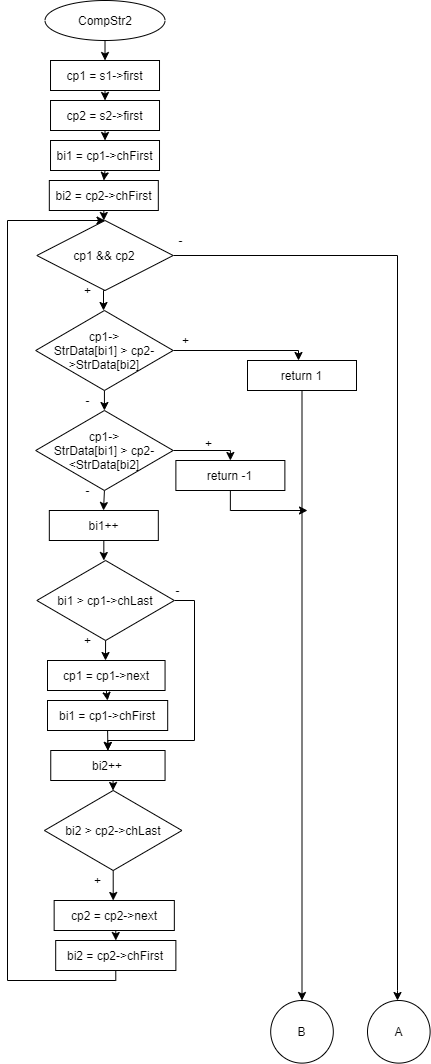
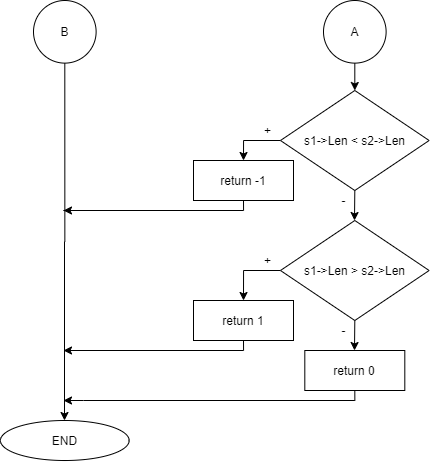


Рисунок 4– Функція порівняння рядків другим способом



Продовження рис. 4

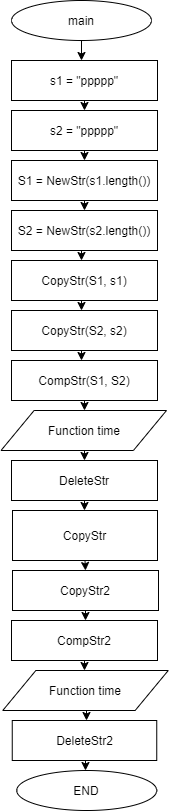
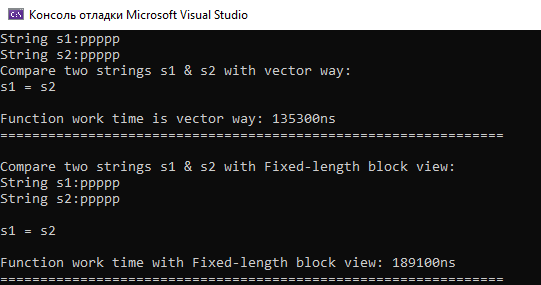


Рисунок 5– Функція main

Результат виконання роботи



**Висновок**

Під час виконання лабораторної роботи було розроблену програму, яка порівнює рядки s1 та s2. Рядки подавали двома способами. Час виконання функції рядка вектора змінної довжини з ознакою кінця менше ніж рядка блочно-зв’язного подання. Це пояснюється тим, що тут змушені обробляти як зв'язкові (списки блоків), так і векторні (масив символів в кожному блоці) структури.

**Контрольні питання**

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

Рядок - це лінійно впорядкована послідовність символів, що належать кінцевій множині символів, яка зветься алфавітом.

Рядки мають важливі властивості:

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 (зчеплення) рядків – це операція, яка зчеплює два рядки і її результатом 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?
   1. Векторне подання рядків:
      * + Вектор постійної довжини;
        + Вектор змінної довжини з ознакою кінця;
        + Вектор змінної довжини з лічильником;
        + Вектор з керованою довжиною.
   2. Зв’язне уявлення рядків;
   3. Символьно-зв'язне уявлення рядків:
      * + Односпрямований лінійний список;
        + Двонаправлений лінійний список.
   4. Блочно-зв'язне уявлення рядків:
      * + Багатосимвольні ланки фіксованої довжини;
        + Багатосимвольні ланки змінної довжини;
        + Багатосимвольні ланки з лічильником з керованою довжиною.
2. Який алгоритм видалення частини рядка?

Операція видалення з рядка символів з номерами від ***n1*** до ***n2***, включно, може бути реалізована як послідовність наступних кроків:

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

1. Навіщо створюється дескриптор рядка?

Наприклад, під вектор з керованою довжиною відводиться пам’ять при створенні рядка і його розмір і розміщення залишаються незмінними весь час існування рядка.

Для такого вектора створюється дескриптор , який зберігає у собі максимальну довжину рядка, поточну довжину, а також покажчик на дані рядка.

1. Які переваги та недоліки подання рядків масивом?

**Переваги:** подання рядків у вигляді векторів дозволяє працювати з рядками, розміщеними в статичній 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 нього вектора, то зайві місця заповнюються пробілами, а якщо рядок виходить за межі вектора, то зайві (зазвичай справа рядку) 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 його розмір і розміщення залишаються незмінними весь час існування рядка. Для такого вектора створюється дескриптор, що збільшує кількість виділеної пам'яті.

1. Які переваги та недоліки подання рядків односпрямованим списком?

**Переваги:**

* Можливість зберігання елементів рядку по “всій” пам’яті, тому що кожен елемент містить код символу і покажчик на наступний елемент.

**Недоліки:**

* Це потребує додаткової пам’яті (на кожен символ рядка необхідний один покажчик, який зазвичай займає 2-4 байта).
* Можливість переміщення по рядку лише у одному напрямку.

1. Які переваги та недоліки подання рядків двоспрямованим списком?

**Переваги:**

* Ті ж, що і в односпрямованому списку.
* Можливість двостороннього руху уздовж списку, що може значно підвищити ефективність виконання деяких строкових операцій.

**Недоліки:**

При цьому на кожен символ рядка необхідно два покажчика, тобто 4-8 байт.