МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет “Львівська політехніка”

СИМВОЛЬНІ РЯДКИ В С
ВВІД ТА ВИВІД

ІНСТРУКЦІЯ

до лабораторної роботи № 4 з курсу
“Основи програмування”
для базового напрямку “Програмна інженерія”

Затверджено
На засіданні кафедри
програмного забезпечення
Протокол № від

ЛЬВІВ – 2018

1. МЕТА РОБОТИ

Мета роботи – здобути практичні навики опрацювання текстової інформації з врахуванням особливостей організації символьних рядків у мові С. Вивчити основні засоби потокового вводу/виводу в С.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ

Мова С не має спеціального типу для оголошення символьних рядків, а розглядає символьний рядок як особливий вид масиву. Елементи масиву, який називають символьним рядком, мають тип `char`, його значеннями є коди символів, з яких складається цей рядок (ASCII-коди, якщо заданий компілятор застосовує ASCII-таблицю для кодування символів). Останнім символом рядка повинен бути т. зв. нуль-символ ('\0'), код якого дорівнює 0. З кожним символьним рядком пов'язується вказівник на початок даного рядка. У всьому іншому – символьні рядки повністю зберігають властивості масивів.

2.1 Оголошення та ініціалізація символьних рядків

Рядкові константи (літерали) в мові С записуються як послідовність довільних символів взятих у подвійні лапки: "...". В оперативній пам'яті їм виділяється ділянка, обсяг якої на один байт більший за кількість символів у рядку. В цей додатковий байт автоматично записується нуль-символ (‘\0’), який надалі слугуватиме ознакою кінця рядка.

Приклад:

```
char *pst = "Hello world!";
```

Для збереження в пам’яті записаного рядка компілятор виділить 13 байтів, з них 12 байтів для символів і останній для '\0' . Адресу початку рядка отримає вказівник pst. Символьні рядки також можуть оголошуватися як звичайні масиви:

```
char імя_символьного_рядка [кількість_символів];
```

Оголошений нижче масив `str` призначений для збереження символьного рядка:

```
char str[150];
```

У `str` можна записати довільний символьний рядок, довжина якого не перевищує 149 символів, оскільки останнім записується нуль-символ – для нього треба обов'язково зарезервувати один байт. Слід також пам'ятати, що ***перевищення встановленої в оголошенні кількості символів не контролюється компілятором і може призвести до небезпечних помилок у роботі програми***.

В оголошеннях символьні рядки, як і масиви символів, можна ініціалізувати. Розглянемо декілька характерних прикладів:

```
char m1[20] = {'a', 'b', 'c', 'd', 'e', 'f'};
char m2[20] = {'a', 'b', 'c', 'd', 'e', ' f', '\0'};
char m3[20]  = "abcdef";
char m4[ ]  = "abcdef";
```

Символьні рядки `m1`, `m2` і `m3` оголошено однаково – як масиви з 20 елементів, що мають тип *char*, але ініціалізацію їх виконано різними способами. Початкові шість елементів масиву `m1` заповнено послідовністю літер, проте без ' \0' у кінці, тому цей масив не буде повноправним символьним рядком, з ним можна буде працювати тільки як із звичайним масивом символів. У масив `m2` записано таку ж послідовність літер, а після неї – нуль-символ. Фактично в `m2` занесено рядок символів `"abcdef"` . Такий же рядок записано в масив `m3` , тобто результати ініціалізації `m2` та `m3` збігаються (очевидно, що ініціалізація `m3` є простішою у записі).

Незаповнені елементи масивів m1, m2 та m3 містять"сміття" (за умови, що масиви оголошено як локальні, а в разі глобальних чи статичних масивів усі вільні елементи заповнюються нулями). Надлишкові елементи можна використовувати надалі для доповнення і розширення відповідних рядків. В оголошенні масиву m4 не вказано граничну кількість символів, тому розмірність цього масиву (символьного рядка) встановлюється за кількістю елементів-ініціалізаторів. Для наведеного прикладу розмірність m4 становитиме 7 символів: 6 перших байтів масиву заповнюються кодами літер, а сьомий – кінцевий нуль-символом. Хоча масиви m3 та m4 проініціалізовані константними рядками, елементи цих масивів можна змінювати так само, як елементи масивів m1 та m2.

2.2. Звертання до елементів символьних рядків

Процеси опрацювання символьних рядків базуються на двох основних властивостях рядків:

1) ім'я символьного рядка є константним вказівником на його перший символ;

2) кінець рядка задається нуль-символом ' \0'.

Для звертання до символів рядка застосовують як індексну, так і вказівникову систему доступу до елементів масиву

Приклад:

```
/*****************************************/
/*  Видалення заданого символа. Варіант 1       */
/*****************************+***************/
#include <conio.h>
#include <stdio.h>

int main (void)
{

      char st[] = "ABC *** XYZ *** KM*Q**RT*"; /* заданий рядок*/
      char sym = '*' ;
      int k, n;

      printf("Vkhidnyj rjadok: %s ",st);
      k = n = 0;
        while (st[k] != '\0' )
              if (st[k] == sym)
                    k++;
            else
            {
                  st[n] = st[k];
                  n++;
                  k++;
            }
      st[n] = '\0';
      printf( "Rjadok bez symvola '%c':", sym);
      printf("%s", st);

      getch();

      return 0;
     }
```

У наступному варіанті програми для звертання до елементів символьного рядка введено вказівники pk і pn. Перший вказує на символ, який перевіряється, а другий – на позицію рядка, куди повинен бути переписаний символ, що залишається в рядку. Замість циклу while у другому варіанті програми використано цикл for, а умову оператора if змінено на протилежну.

```
/*****************************************/
/*  Видалення заданого символа. Варіант 2   */
/*****************************************/

#include <conio.h>
#include <stdio.h>

int main(void)
{
     char st[] = "ABC *** XYZ *** KM*Q**RT*";     /* заданий рядок */
     char sym = '*';                             /* символ, що має бути вилучений */
     char *pk, *pn;                              /* вказівники на символи рядка */

     printf("Vkhidnyj rjadok: %s ",st);
     for (pk = pn = st; *pk != '\0'; pk++)       /* цикл по символах рядка */
     if (*pk != sym)
     *pn++ = *pk;                        /* копіювання всіх символів, крім заданого */
     *pn = '\0';                         /* фіксація кінця нового рядка */

     printf( "Rjadok bez symvola %c:", sym); printf("%s", st);

     getch();

     return 0;
}
```

У циклі `for` даної програми умовний оператор:

```
if (*pk != sym) *pn++ = *pk;
```

послідовно перевіряє кожен символ рядка `st`, звертаючись до елементів через вказівник `pk`. Якщо поточний символ не збігається з тим, який задано для вилучення `(*pk != sym)`, то він переписується у позицію, на яку вказує `pn` `(*pn++ = *pk)`. Після цього `pn` пересувається на наступний символ рядка (`pn++`).

2.3. Бібліотечні функції для роботи з символами та символьними рядками

Стандартна бібліотека мови С включає набір різнотипних функцій, що забезпечують швидку реалізацію операцій, які найчастіше зустрічаються у процесах опрацювання символьних і текстових даних.

2.3.1. Функції класифікації та перетворення символів

У заголовному файлі `<ctype. h>` оголошено групу функцій, призначених для перевірки та класифікації окремих символів. Імена цих функцій починаються префіксом `is`: `is... ()`. Усі функції мають один параметр з типом `int` – символ, що перевіряється. Функції перевірки повертають ціле ненульове значення (істина), якщо заданий символ належить до відповідної класифікаційної групи, і нульове значення (хибне), якщо символ не належить до цієї групи. Наприклад, функцію, яка перевіряє, чи заданий символ `sym` є літерою (до уваги беруться тільки великі та малі латинські літери), оголошено так:

```
int isalpha (int sym);
```

У табл.1 наведено найбільш популярні функції класифікації та зміни символів.

Дві останні функції з табл. 1: `tolower()` та `toupper()` призначені для зміни регістра символів-літер (на жаль, тільки латинських). Перша з цих функцій повертає відповідну малу літеру, якщо `sym` велика латинська літера, а друга – повертає велику літеру, якщо `sym` мала латинська літера. В інших випадках обидві функції повертають значення `sym`.

Табл. 1. Основні функції класифікації та зміни символів

Функція	Призначення
класифікації:	Перевіряє, чи символ sym є:
`int isalpha(sym)`	малою або великою латинською літерою
`int isdigit(sym)`	десятковою цифрою
`int isalnum(sym)`	латинською літерою або десятковою цифрою
`int isxdigit(sym)`	шістнадцятковою цифрою
`int isspace(sym)`	пробільним символом (символом пробілу, нового рядка, горизонтальної чи вертикальної табуляції)
`int islower(sym)`	малою латинською літерою
`int isupper(sym)`	великою латинською літерою
перетворення:	Повертає:
`int tolower(sym)`	малу латинську літеру, якщо sym велика латинська літера
`int toupper(sym)`	велику латинську літеру, якщо sym мала латинська літера

2.4. Функції операцій над символьними рядками

Прототипи бібліотечних функцій, призначених для роботи з символьними рядками, оголошені в заголовному файлі `<string.h>`. Основну групу складають функції, які починаються префіксом `str`: `str...()`. У табл. 2 описано функції, які найчастіше використовуються у процесах опрацювання символьних рядків. Щоб скоротити записи прототипів функцій у табл.2, в списку параметрів кожної функції вказано тільки імена параметрів. Типи параметрів в оголошеннях цих функцій є такими:

```
const char *s, *s1, *s2;
char *sr;
unsigned n;
int sym;
```

Символьні рядки, які опрацьовуються функціями <string.h>, мають обов'язково закінчуватись ' \0'. У разі звертання до функцій конкатенації (об'єднання) та копіювання рядків треба забезпечити, щоб розмірність масиву символів `sr` була достатньою для запису рядка результату, оскільки функції не контролюють довжин рядків. Крім цього, рядки `sr` і `s` у функціях конкатенації та копіювання не повинні перекриватись.

```
/*****************************************************/
/*  Видалення підрядка зі заданого символьного рядка */
/*****************************************************/
#include <conio.h>
#include <stdio.h>
#include <string.h>

int main (void)
{
      char str[]= "abcdef # 12345679 # uvwxyz"; /* рядок */
      char *p1, *p2;                 /* вказівники на початок і кінець підрядка */
      printf("Vkhidnyj rjadok: %s ",str);
      p1 = strchr(str, '#' );              /* перший символ '#' */
```

```
    p2 = strrchr(str, '#');              /* останній символ '#' */
    strcpy (p1, p2+1) ;                  /* перенесення кінцевої частини рядка */
    printf("\n Rjadok pislja vyluchennja: %s", str);
    getch();
    return 0;
}
```

Щоб вилучити підрядок, знаходимо його початок (перший символ `'#'` у `str`) і кінець (останній символ `'#'`). Потім переносимо, використовуючи функцію `strcpy()`, кінцеву частину основного рядка на місце підрядка, який треба видалити.

Табл. 2. Основні функції опрацювання символьних рядків

Функція	Призначення
`char* strcpy(sr, s);`	Копіює рядок s (з '\0' включно) за адресою, заданою параметром `sr`. Повертає значення `sr` - адресу скопійованого рядка.
`char* strcat(sr, s);`	Додає рядок `s` (з '\0' включно) у кінець рядка `sr`. Повертає значення `sr` – адресу доповненого рядка.
`int strcmp(s1, s2);`	Послідовно порівнює символи рядків `s1` і `s2` як дані з типом unsigned char. Повертає ціле число, значення якого <0, якщо `s1<s2`; 0, якщо `s1 == s2`; > 0, якщо `s1 > s2`.
`char* strncpy(sr, s, n);`	Аналог `strcpy()`, але з `s` копіюється не більше, ніж `n` початкових символів; якщо скопійована група символів не закінчується '\0', то нуль-символ у `sr` не заноситься.
`char* strncat (sr, s, n);`	Аналог `strcat()`, але додає до `sr` тільки n початкових символів з `s`; у кінець об'єднаного рядка заноситься '\0'.
`char* strncmp(s1, s2, n);`	Аналог `strcmp()`, але порівнює тільки n початкових символів рядків `s1` та `s2`. Якщо якийсь із рядків коротший за n, то порівняння припиняється з досягненням ' \0'.
`unsigned strlen(s);`	Повертає довжину рядка `s` у символах (' \ 0' не враховується).
`char* strchr(s, sym);`	Перевіряє, чи символ sym входить у рядок `s`. Повертає вказівник на перше входження sym у `s` або `NULL`, якщо `sym` не зустрічається в рядку `s`.
`char* strrchr(s, sym);`	Аналог `strchr()`. Повертає вказівник на останнє входження символа `sym` у рядок `s` або `NULL`, якщо `sym` не зустрічається в рядку `s`.
`char* strstr(s1, s2);`	Перевіряє, чи рядок `s2` входить як підрядок у `s1`. Повертає вказівник на перший символ рядка `s2` у `s1` або `NULL`, якщо рядок `s2` не зустрічається у рядку `s1`.
`char* strtok(sr, s);`	Виділяє в рядку `sr` лексеми, обмежені символами з рядка `s`. Повертає вказівник на виділену лексему або `NULL`.
`char* strdup(s);`	Копіює рядок `s` (з ' \0 ' включно) в динамічну пам'ять, попередньо виділивши там ділянку потрібної довжини. Повертає адресу рядка в динамічній пам'яті.

Останньою в табл. 2 записана функція `strdup()`, яка заносить копію заданого символьного рядка (з нуль-символом включно) у динамічну пам'ять та повертає адресу, за якою записано рядок. Функція `strdup()` не належить до функцій, що підтримуються стандартом мови С, хоча вона входить до складу більшості С-бібліотек.

Функція виділення лексем. Зупинимось детальніше на функції `strtok()`, оголошення якої наступне:

```
char* strtok(char* str, const char* lim);
```

Ця функція виконує поділ символьного рядка `str` на окремі лексеми, записуючи після кожної лексеми '\0'. Рядок `lim` задає набір символів, якими можуть бути обмежені лексеми рядка `str` (нуль-символ у переліку обмежувачів вказувати не треба). Для виділення всіх лексем символьного рядка функцію використовують циклічно. У першому звертанні до `strtok()` вказують адресу початку рядка, а функція повертає адресу першої знайденої лексеми. У наступних звертаннях до `strtok()` замість першого параметра записують порожній вказівник `NULL`, а функція повертає адресу наступної лексеми рядка. Коли всі лексеми виділені, функція повертає `NULL`.

```
/*********************************************/
/* Виділення слів-лексем із символьного рядка  */
/*********************************************/
#include <stdio.h>
#include <string.h>
#include <conio.h>

int main(void)
{
      char example[]="Символи, рядки (виділення слів-лексем)";
      const char *limits = " ,.;()-";      /* символи-обмежувачі лексем */
      char *pw;                            /* вказівник на лексеми */

      setlocale(0,".1251");          /* Встановлення локалізації для виводу
                                        кириличних символів */

      printf("Вхідний текст: %s" , example);
      printf( "\n      Слова:   \n");
      pw = strtok (example, limits);/* знаходження першої лексеми */
      while(pw !=NULL)
      {
            printf( "%s\n",pw);
            pw = strtok (NULL, limits);    /* пошук наступної лексеми */
      }
      _getch();
      return 0;
}
```

2.5. Функції перетворення рядків символів у числа та зворотних перетворень

Перетворення "символьний рядок => число". У багатьох задачах необхідно Перетворювати числові дані, записані у формі текстових рядків, в одну з внутрішніх форм збереження чисел. Такі перетворення реалізують стандартні бібліотечні функції, оголошені в `<stdlib.h>` (табл.3). Типи параметрів цих функцій наступні:

```
const char  *st;
char        **end;
int         base;
```

Табл. 3. Функції перетворення символьних рядків у числа

Функція	Призначення
`int atoi(st);`	Виділяє у рядку `st` перше ціле десяткове число і перетворює його у дане з типом `int`. Числу може передувати довільна кількість символів пробілу. Кінцем рядка вважається перший символ, що не належить до цифр. Повертає знайдене числове значення у разі успішного перетворення, а в разі помилки - результат не визначений.
`long atol(st) ;`	Аналог `atoi()`, але перетворює рядок у число з типом `long`.
`double atof(st);`	Аналог `atoi()`, але перетворює рядок `st` у дане з типом `double`. Число у рядку може бути записане як ціле чи як дійсне у формі з фіксованою або з плаваючою крапкою.
`long strtol(st, end, base);`	Розширений варіант `atol()`. Вказівник `end` (цей параметр є вказівником на вказівник) задає адресу змінної-вказівника, в яку буде записано адресу першого символа, що залишився неперетвореним. Параметр `base` визначає основу системи числення, в якій записано число і може приймати значення від 2 до 36: цифрами числа можуть бути арабські цифри і послідовні малі або великі латинські літери (для `base` >10). Якщо `base` дорівнює 0, то основа числа визначається формою його запису: число, що починається 0, вважається вісімковим, число з префіксом 0X чи 0x – шістнадцятковим, всі інші числа розглядаються як десяткові.
`unsigned long strtoul(st, end, base);`	Аналог `strtol()`, але повертає значення, що має тип `unsigned long`.
`double strtod(st, end);`	Розширений варіант `atof()`. Перетворює початкову частину `st` у дане з типом `double`. Додатково повертає через `end` адресу першого символа, записаного за числом.
`float strtof(st, end);`	Аналог `strtod()`, але повертає значення з типом `float`.
`long double strtold (st, end);`	Аналог `strtod()`, але повертає значення, що має тип `long double`.

Перетворення "число => рядок". Бібліотека мови програмування С додатково містить ряд функцій зворотних перетворень "число => символьний рядок серед яких:

```
char* itoa(int num, const char* str, int base);
char* ltoa(long num, const char* str, int base);
char* ultoa(unsigned long num, const char* str, int base);
```

Ці функції відрізняються між собою тільки типом параметра `num` – цілого числа, яке потрібно перетворити. Всі три функції формують із числа `num` рядок символів, що відповідає запису

цього числа в системі числення з основою `base`, і повертають вказівник на перший символ створеного рядка. Сформований рядок записується за адресою `str`. Функції `itoa()` та `ltoa()` записують від'ємні числа зі знаком мінус

2.6. Масиви символьних рядків і масиви вказівників

Практичні задачі часто вимагають опрацювання груп символьних рядків. У таких випадках створюють або масиви символьних рядків, або масиви вказівників на перші символи рядків, а самі рядки розташовують в оперативній пам'яті окремо. Розглянемо ці два підходи.

2.6.1. Масиви символьних рядків

Поширеним видом багатовимірних масивів є масиви символьних рядків, тобто масиви, кожен елемент яких – окремий символьний рядок. Наприклад, масив найменувань міст, оголошений і проініціалізований наступним чином:

```
char cities[6][20] = {"Львів", "Хмельницький", "Полтава",
                      "Рівне", "Івано-Франківськ", "Київ"};
```

є двовимірним масивом з шести рядків, кожен з яких заповнений найменуванням відповідного міста

Правила звертання до елементів масиву символьних рядків такі ж, як і для всіх інших багатовимірних масивів. Зокрема, перший рядок масиву ("Львів") виділяють вирази `*cities` та `cities[0]`.

`cities`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Л	ь	в	і	в	\0														
Х	м	е	л	ь	н	и	ц	ь	к	и	й	\0							
П	о	л	т	а	в	а	\0												
Р	і	в	н	е	\0														
І	в	а	н	о	–	Ф	р	а	н	к	і	в	с	ь	к	\0			
К	и	ї	в	\0															

2.6.2. Масиви вказівників на символи рядків

Створення масиву вказівників на перші символи набору рядків, кожен з яких зберігається в оперативній пам'яті окремо, – це альтернативний підхід, що дає змогу усунути недолік виділення зайвого місця, властивий масивам символьних рядків. Розглянемо програму, яка здійснює випадковий вибір одного міста зі заданого списку міст (імітація жеребкування). В програмі створено масив вказівників cparr, елементи якого мають тип `char*`. Масив проініціалізовано стрінговими константами і найменуваннями міст. Кожному рядку, тобто найменуванню міста, компілятор виділяє в пам'яті ділянку, обсяг якої дорівнює довжині рядка плюс нуль-символ. Адреса розміщення рядка заноситься у відповідний елемент масиву `cparr`.

```
/************************************/
/*  Вибір міста шляхом жеребкування   */
/************************************/
#include <stdio.h>
#include  <stdlib.h>
#include <conio.h>
#include <time.h>

int main(void)
```

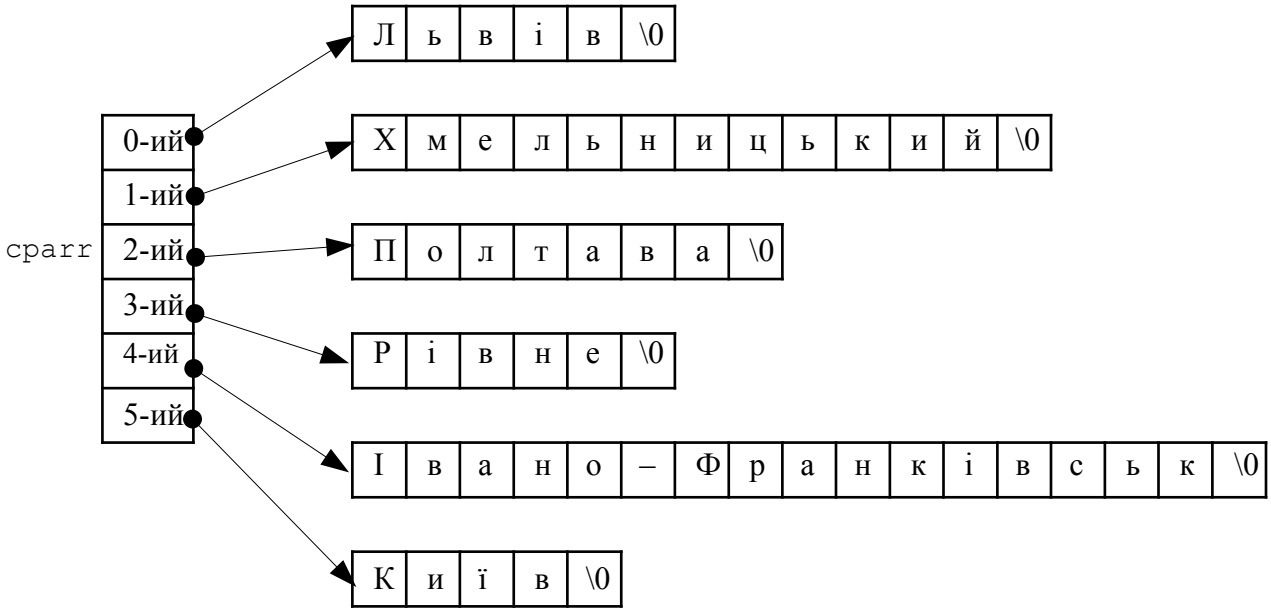
```
{
      char* cparr[] = {"Львів", "Хмельницький",     /* список міст */
                       "Полтава", "Рівне", "Івано-Франківськ", "Київ"};
      int rc;                                       /* номер вибраного міста */

      setlocale(0,".1251");

      srand(time(NULL));                            /* вибір міста */
      rc = rand() % (sizeof cparr / sizeof(char*));
      printf("\nВибране місто - %s \n", cparr[rc]);

      _getch();

      return 0;
}
```



Щоб забезпечити можливість зміни складу і кількості міст, у програмі в оголошенні масиву `cparr` не вказано розмірність цього масиву. Тому задана кількість міст обчислюється за допомогою виразу

```
sizeof cparr / sizeof(char *)
```

Значення цього виразу використано як максимальне значення генерованого числа за допомогою функції `rand()` для визначення номера вибраного міста. (Функція `rand()` генерує псевдовипадкове дійсне число в діапазоні 0.. 32767).

Перевага використання масивів вказівників, які зберігають адреси початків рядків замість масивів символьних рядків особливо відчутна, коли в програмі треба переставляти окремі рядки місцями, наприклад, потрібно впорядкувати набір рядків. У цих випадках самі символьні рядки не переписують (це важливо, адже їх розмір може бути достатньо великим), а тільки міняють місцями вказівники на початки рядків.

2.7. Ввід та вивід в мові С

Основним завданням програмування є обробка інформації, тому будь-яка мова програмування має засоби для вводу і виводу інформації. У компілятор мови С не включено спеціальних засобів для вводу/виводу даних. Тому обмін даними як зі зовнішніми (консольними) пристроями, так і з дисковими файлами реалізовано через відповідні набори бібліотечних функцій.

Бібліотеки більшості систем програмування мови С підтримують функції, що дають змогу здійснювати операції вводу/виводу даних на трьох рівнях:

- високорівневе, (потокоорієнтоване) введення/виведення;
- введення/виведення низького рівня;
- обмін даними з консольними пристроями.

Високорівневе введення/виведення використовує однаковий підхід у програмуванні обміну даними з файлами і зовнішніми пристроями та єдиний інтерфейс, незалежний від структури і способів доступу до файлу чи термінального пристрою. Всі файли та дані з пристроїв розглядаються як неструктуровані набори байтів – ***потоки***.

Функції високорівневого потокового вводу/виводу прості в програмуванні, та високоефективні завдяки внутрішній буферизації даних. Прототипи функцій потокоорієнтованого буферизованого обміну даними записані в заголовному файлі **stdio.h**.

Функції низькорівневого вводу/виводу базуються на засобах обміну даними, що властиві кожній конкретній операційній системі, тому вони не належать до стандартизованих. Ці функції не виконують форматування даних у процесах введення/виведення і не застосовують буферизації. З кожним файлом, відкритим на низькому рівні, пов'язується свій цілочисловий дескриптор (його також називають префіксом). Дескриптор зберігає номер позиції у внутрішніх таблицях операційної системи, де записано інформацію про цей файл.

Функції низькорівневого вводу/виводу блокоорієнтовані і забезпечують виграш у швидкодії лише тоді, коли обсяг блоку даних, що передається за одну операцію введення чи виведення, кратний ємності сектора диска (як правило 512 байтів).

Функції консольного вводу/виводу доповнюють і розширюють можливості високорівневих функцій щодо введення даних з клавіатури і керування формою зображення екранних повідомлень у текстових режимах виведення інформації. До цієї групи належать також функції передавання даних через порти. Консольний обмін даними найбільш залежний від операційної та апаратної платформи комп'ютера. Прототипи функцій консольного вводу/виводу оголошені в заголовному файлі *conio.h*.

Заголовний файл *stdio.h* з-поміж інших містить оголошення групи функцій форматованого вводу/виводу, схожих між собою за сигнатурою та функціональністю. Розглянемо дві найпростіші з них, які працюють зі стандартним потоком (командним рядком).

- `printf()` – для форматованого виводу інформації у стандартний потік **stdout**.
- `scanf()` – для форматованого вводу інформації зі стандартного потоку **stdin**.

2.8. Форматований вивід інформації

Функція **printf ()** призначена для форматованого виведення. Вона переводить дані в символьне представлення і записує отриману послідовність символів в стандартний потік виводу, що приводить, як правило, до виводу символів на екран. При цьому у програміста є можливість форматувати дані, тобто впливати на їх виведення на екран.

Загальна форма запису функції **printf ()**:

```
printf("Рядок формату", об'єкт0, об'єкт1, ...);
```

`Рядок формату` може містити наступні елементи:

- керуючі символи;
- текст для безпосереднього виведення;
- специфікатори, призначені для виведення значень змінних різних типів.

Об'єкти можуть бути відсутніми у випадку відсутності специфікаторів у рядку формату.

Керуючі символи не виводяться на екран, а керують розташуванням символів, що виводяться. Відмінною рисою керуючого символу є наявність зворотної косої риски (слеша) '\' перед ним.

Основні керуючі символи:

`'\n'` – новий рядок;
`'\t'` – горизонтальна табуляція;
`'\v'` – вертикальна табуляція;
`'\b'` – повернення на символ;
`'\r'` – повернення на початок рядка;
`'\a'` – звуковий сигнал.

Специфікатори потрібні для того, щоб вказувати тип об'єкта, який виводитиметься, та вигляд, в якому інформація буде подана. Відмінною рисою специфікатора є наявність символу відсоток `'%'` перед ним. Нижче наведений перелік специфікаторів (форматів) для об'єктів різних типів.

`%d або %i` – ціле число типу **int** зі знаком в десятковій системі числення;
`%u` – ціле число типу **unsigned int**;
`%x` – ціле число типу **unsigned int** зі знаком в шістнадцятковій системі числення;
`%X` – ціле число типу **unsigned int** зі знаком в шістнадцятковій системі числення (прописом);
`%o` – ціле число типу **unsigned int** зі знаком в вісімковій системі числення;
`%hd` – ціле число типу **short** зі знаком в десятковій системі числення;
`%hu` – ціле число типу **unsigned short**;
`%hx` – ціле число типу **short** зі знаком в шістнадцятковій системі числення;
`%ld` – ціле число типу **long int** зі знаком в десятковій системі числення;
`%lu` – ціле число типу **unsigned long int**;
`%lld` – ціле число типу **long long int** зі знаком в десятковій системі числення;
`%llu` – ціле число типу **unsigned long long int**;
`%lx` – ціле число типу **long int** зі знаком в шістнадцятковій системі числення;
`%llx` – ціле число типу **long long int** зі знаком в шістнадцятковій системі числення;
`%f` – дійсне число (число з плаваючою крапкою типу **float**);
`%F` – дійсне число (число з плаваючою крапкою типу **float**) (прописом);
`%lf` – дійсне число подвійної точності (число з плаваючою крапкою типу **double**);
`%e` – дійсне число в експоненційній формі (числа з плаваючою крапкою типу **float** в експоненційній формі);
`%g` – дійсне число (автоматичний вибір серед коротшого `%f` чи `%e`)
`%c` – один символ (об'єкт типу **char**);
`%s` – набір символів з останнім нульовим символом `'\0'` (нуль-термінальний рядок).
`%p` – вказівник (об'єкт типу **void***);
`%n` – нічого не виводиться. Відповідний аргумент (об'єкт) повинен мати тип **int**;
`%%` – символ %

Приклад:

```
int a = 5;
float x = 2.78;
printf("\n Значення змінної a =%d", a);
printf("\n Значення змінної x =%f", x);
```

Результат роботи програми:

```
Значення змінної a = 5
Значення змінної x = 2.780000
```

Увага:

Кількість специфікаторів в рядку формату обов'язково має співпадати із кількістю об'єктів, які йдуть аргументами опісля нього.

При використанні специфікатора можна явно вказати загальну кількість знакомісць, а для дійсних чисел – і кількість знакомісць, займаних дробовою частиною.

Приклад:

```
float y = 1.2345;
printf("\n Значення змінної y =%10.5f \n", y);
```

Результат роботи програми:

```
Значення змінної y =   1.23450
```

У наведеному прикладі 10 – загальна кількість знакомісць під значення змінної; 5 – кількість позицій після десяткового дробу. Зверніть увагу, що дробова крапка теж є символом і враховується у загальній кількості знакомісць. У зазначеному прикладі кількість знакомісць у виведеному числі менша за 10, тому вільні знакомісця зліва від числа заповнюються пробілами. Такий спосіб форматування часто використовується для побудови таблиць.

Функція `printf()` у разі успіху повертає кількість виведених символів. В іншому випадку результат буде негативний, а в глобальну змінну **errno** буде записаний код помилки.

2.9 Форматований ввід інформації

Функція форматованого вводу даних **scanf()** виконує читання даних з основного потоку вводу (зазвичай командний рядок), перетворює їх у внутрішній формат і передає результат у місце виклику. При цьому програміст задає правила інтерпретації вхідних даних за допомогою специфікації рядка формату.

Загальна форма запису функції **scanf()**:

```
scanf("Рядок формату", адреса0, адрес1, ...);
```

Рядок формату аналогічний функції `printf()`. Для формування адреси змінної використовується символ амперсанд '&':

```
адреса = &об'єкт
```

Зверніть увагу на те, що для введення нуль-термінальних рядків символ & не використовується, оскільки назва змінної (об'єкт) уже є адресою. Рядок формату і список аргументів (адрес) для функції обов'язкові.

У разі успіху функція `scanf()` повертає кількість заповнених аргументів. У випадку помилки читання чи досягнення кінця файлу функція повертає **EOF**, а в глобальну змінну **errno** буде записаний код помилки.

Приклад:

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

int main()
{
	system("chcp 1251"); // Встановлення кодування кирилиці для
командного рядка
	system("cls"); // Очищення командного рядка

	float y;
	printf("Введіть y: ");
	scanf_s("%f", &y, 1); // Введення значення змінної y
	printf("Значення змінної y = %f", y);

	getchar();getchar();
	return 0;
}
```

Результат роботи програми:

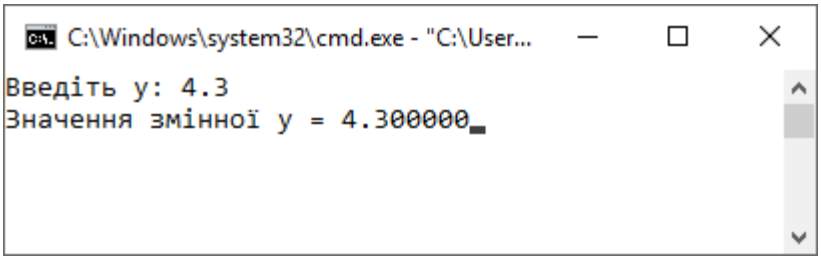


Рис. 1. Результат роботи програми

2.10. Ввід/вивід з рядків

Заголовний файл *stdio.h* також містить опис функцій для форматованого вводу/виводу не тільки зі стандартного потоку (командного рядка як правило).

Так, функції **sprintf()**, **snprintf()** та **sscanf()** призначені для виводу/вводу з рядків.

Загальна форма запису функції **sprintf()**:

```
sprintf("Рядок призначення", "Рядок формату", об'єкт0, об'єкт1, ...);
```

`Рядок призначення` вказує на місце (змінну), куди буде проводитися вивід. Решта аргументів така ж, як і у функції `printf()`.

Загальна форма запису функції **snprintf()**:

```
snprintf("Рядок призначення", "Довжина", "Рядок формату", об'єкт0,
об'єкт1, ...);
```

Аргумент **`Довжина`** має містити максимальну кількість байт, які функція може записати в рядок призначення (включно з нульовим символом). Решта аргументів аналогічні попередній функції. Ця функція вважається канонічною для перетворення інформації у символьні рядки.

Загальна форма запису функції **sscanf()**:

```
sscanf("Рядок-джерело", "Рядок формату", об'єкт0, об'єкт1, ...);
```

`Рядок-джерело` вказує на вхідний нуль-термінальний рядок, з якого функція буде читати дані для перетворення. Решта аргументів аналогічні функції `scanf()`.

2.11. Робота з файлами

Файлом вважається іменована сукупність даних, розташованих на зовнішньому носії, а також термінальні пристрої (клавіатура, принтер …).

Для уніфікації процесів файлового обміну даними у функціях високого рівня використовують поняття *потоку.*

Потік (stream) – послідовність байтів, що надходять від певного логічного пристрою (файлу) або передаються у цей файл (пристрій).

Для узагальнення обміну даними у процесах вводу/виводу здійснюється проміжна буферизація даних. Для кожного відкритого файлу в оперативній пам'яті створюється буфер обміну заданої ємності.

Перед введенням даних з конкретного файлу або записом даних у файл, необхідно створити потік, пов'язаний з цим фізичним файлом. Створення потоку реалізує функція відкриття файлу, прототип якої оголошено так:

```
FILE* fopen(char const* file_name, char const* fmode);
```

Параметрами функції є вказівниками на символьні рядки.

file_name – ім'я файлу, *fmode*– режим обміну даними.

За умови успішного відкриття потоку створюється спеціальна структура типу *FILE* і функція повертає її адресу. Якщо ж потік відкрити не вдалось, функція повертає NULL.

Основні режими відкриття файла (другий параметр *fmode* функції `fopen()`):

"r" – (від англ. read). Файл відкривається тільки для читання (повинен існувати);

"w" - (від англ. write). Файл відкривається тільки для запису (якщо існує – його вміст знищується);

"a" – (від англ. append). Файл відкривається тільки для доповнення;

"r+" - файл відкривається для читання з можливістю запису в нього;

"w+" – файл відкривається для запису з можливістю читання з нього;

"a+" - файл відкривається для доповнення з можливістю читання.

Додатково в параметрі *fmode* можна задавати текстовий (t) чи бінарний (b) режим відкриття потоку, за замовчуванням встановлюється текстовий режим.

Для закриття потоків використовується функція

```
int fclose(FILE* fp);
```

fp – вказівник на потік, який треба закрити. При успішному використанні функції вона повертає значення 0.

Стандартні потоки

На початку виконання кожної С-програми відкриваються стандартні потоки:

stdin – потік введення, який пов'язується з клавіатурою;

stdout – потік виведення даних на екран (командний рядок);

stderr – потік повідомлень про помилки.

Стандартні потоки можна перескеровувати (перепризначати), пов’язувати їх із заданим файлом чи пристроєм.

Перескерування потоків виконує функція

```
FILE *freopen (char *fname, char *fmode, FILE *fp);
```

Функція пов’язує потік *fp* з файлом *fname*. Режим доступу до даних задає параметр *fmode* .

Значення та форми завдання параметрів *fname* і *fmode* такі ж, як для функції *fopen()*. За умови успішного виконання функція повертає вказівник на створений потік, а в разі помилки – NULL.

Приклад:

Перше звертання до ***printf()*** *викликає виведення повідомлення на екран, а друге – у файл* ***example.out***.

```
#include <stdio.h>
int main()
{
      printf("Виведення на екран");
      freopen("example.out", "w", stdout);
      printf("Виведення в файл");
      return 0;
}
```

Для форматованого виводу даних у конкретний файл використовується функція:

```
int  fprintf(FILE* fp, char* format, … );
```

Параметр *fp* є вказівником потоку виводу. Решта параметрів такі ж, як і в *printf()*.

Для форматованого вводу даних з конкретного файлу використовується функція:

```
int  fscanf(FILE* fp, char* format, … );
```

Параметр *fp* є вказівником потоку вводу. Решта параметрів такі ж, як і в *scanf()*.

Для виведення одного символу використовується функція:

```
int fgetc(FILE* fp);
```

При успішному виведенні повертає код зчитаного символа з потоку введення *fp*.

Запис одного символа виконується функцією:

```
int fputc (int symb, FILE* fp);
```

При успішному виведенні повертає код записаного символа *symb* у потік виведення *fp*.

Зчитування рядка символів з потоку здійснює функція:

```
char* fgets (char* str, int max, FILE* fp);
```

str – масив символів, в який буде записано зчитаний рядок; *max* – максимальна кількість символів (з нуль-символом включно), яку може містити зчитаний рядок; *fp* – потік, з якого вводяться рядки. З потоку у ділянку оперативної пам'яті (*str*) зчитується послідовність символів до символа нового рядка чи символа кінця файла, але не більше, ніж *max-1* символів.

У разі успішного виконання функція повертає адресу першого символа введеного рядка (*str*), інакше – NULL. Розмір масиву *str* повинен бути не меншим за *max*.

Запис заданого рядка в потік виведення здійснює функція

```
int  fputs (char *str, FILE *fp);
```

Ця функція повертає ненульове значення за умови успішного виконання та EOF у разі невдачі. Функція послідовно передає у потік символи рядка *str*.

Приклад:

```
#include<stdio.h>
void main()
{
      FILE *fi;
      int age;
      fi = fopen("age.txt", "r"); /* відкриття файлу для читання */
      fscanf(fi, "%d", &age); /*читання числового значення */
      fclose(fi); /* закриття файлу */
      fi = fopen("data.txt", "a"); /* відкриття файлу для додавання
інформації в кінець */
      fprintf(fi, "Age=%d.\n", age); /* запис рядка в файл */
      fclose(fi);           /* закриття файлу */
}
```

Описані вище функції вводу/виводу використовуються для роботи із текстовими файлами. Для роботи з файлами будь-якого формату (текстовими чи бінарними) використовуються такі функції, як **fread()**, **fwrite()**, **fseek()**, **ftell()**.

2.12. Ввід/вивід у стандарті C1X

Стандартом 2011 року стандартні функції вводу/виводу, які працюють із вказівниками (наприклад, *scanf()*, *sprintf()*, *sscanf()*, *gets()* тощо), визнані застарілими. На заміну їм вводяться відповідники **scanf_s()**, **sprintf_s()**, **sscanf_s()**, **gets_s()** і т.д. Нові функції мають таку ж сигнатуру, як і старі. Проте, якщо використовувати символьний *%c* чи рядковий *%s* специфікатори, то у відповідність їм мають бути вказані пари аргументів. Першим аргументом,

як і в старих функціях, іде адреса об’єкта, другим аргументом – кількість елементів в масиві за вказаною адресою. Якщо адреса вказує на одиничний об’єкт, то значення другого аргумента в парі має бути рівним одиниці. Також стандартом вводиться макрос **_countof()**, який може автоматично підраховувати кількість об’єктів для статичних масивів.

Приклад:

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

int main(void)
{
      int       i,result;
      float     fp;
      char      c,s[80];
      result = scanf_s("%d %f %c %s ", &i, &fp, &c, 1, s,
          (unsigned) _countof(s));
      printf("The number of fields input is %d\n", result);
      printf("The contents are: %d %f %c %s\n", i, fp, c, s);
}
```

3. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Як у мові С організована робота з рядками символів?
2. Які способи ініціалізації символьних рядків Ви знаєте?
3. Які є особливості роботи з символьними рядками в С?
4. Яку роль відіграє ім’я масиву символів? Чи можна його використати для адресації довільного символа рядка через вказівник?
5. Що є ознакою кінця символьного рядка? Напишіть оператор циклу для перебору всіх символів заданого рядка через вказівник типу char.
6. Які основні функції класифікації та перетворення символів Ви знаєте?
7. За допомогою якої стандартної бібліотечної функції можна виконати порівняння двох рядків? Наведіть приклад.
8. Як перетворити числове значення в рядок символів?
9. Яка різниця між масивом символьних рядків та масивом вказівників на символи рядків?
10. Як здійснюється введення символів та рядків у С?
11. Як скопіювати один рядок в інший? Чому не можна просто присвоїти один масив символів іншому?
12. Як перевірити чи входить заданий символ у заданий рядок? Напишіть фрагмент коду, який обчислює кількість входжень символа ‘E’ у рядок символів text.
13. Що таке специфікатор? Які правила запису специфікаторів?
14. Які види специфікаторів Ви знаєте?
15. Яка функція використовується для форматованого виводу у стандартний потік?
16. Яка функція використовується для форматованого вводу зі стандартного потоку?
17. Які функції використовується для форматованого виводу у рядок?
18. Яка функція використовується для форматованого вводу з рядка?
19. Які функції використовується для форматованого виводу у файл?
20. Яка функція використовується для форматованого вводу з файлу?
21. Які специфікатори позначають цілі числа?
22. Які специфікатори позначають дійсні числа?
23. Який специфікатор позначає символи?

4. ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Ознайомитися з теоретичним матеріалом викладеним вище в даній інструкції і виконати приклади програм.
2. Одержати індивідуальне завдання.
3. Розробити алгоритм розв'язання індивідуального завдання і подати його у вигляді блок-схеми.
4. Скласти програму на мові С у відповідності з розробленим алгоритмом.
5. Виконати обчислення по програмі.
6. Самостійно ознайомитися з функціями fread(), fwrite(), fseek(), ftell().
7. Скласти програму на мові С, яка обчислюватиме значення виразу з лабораторної роботи №1 Додаток 1 "Обчислення заданих арифметичних виразів". Значення параметрів X, Y, Z прочитати із заданого бінарного файлу (вводиться користувачем з клавіатури). Результат записати у файл, теж вказаний користувачем. Формат файлу (бінарний чи текстовий) також задається користувачем. При цьому робота програми повинна бути запротокольована. Виконання основних подій, починаючи зі старту програми, має бути відображене у log-файлі з часовими мітками (розміщення вибрати самостійно). Старі записи у log-файлі мають зберігатися. Перелік подій для логування:
   - старт програми
   - відкриття файлу з параметрами
   - обчислення значення виразу
   - запис обчисленого значення у вихідний файл
   - завершення програми

   Приклад log-файлу:

```
[2016.09.01 8:30.000] Program started.
[2016.09.01 8:30.011] Parameters file "d:\data\input.bin" opened. X=3, Y=1, Z=2.4.
[2016.09.01 8:30.012] Expression calculated. Result = 4.56.
[2016.09.01 8:30.014] Output file "d:\data\output.txt" saved.
[2016.09.01 8:30.015] Program ended.
```

   Для отримання поточного часу можна скористатися функцією time() із заголовного файлу *time.h.*

11. Підготувати та здати звіт про виконання лабораторної роботи.

5. СПИСОК ЛІТЕРАТУРИ

1. Керниган Б., Ритчи Д. Язык программирования С. - М. - Финансы и статистика. - 1992. – 272 с.
2. Уэйт М., Прата С., Мартин Д. Язык С. Руководство для начинающих. - М. - Мир. - 1988. – 512 с.
3. К. Джамса. Учимся программировать на языке С++. М.: Мир, 1997. – 320 с
4. Герберт Шилдт. Полный справочник по С++. М. – С.-П.-К., Вильямс. – 2003. – 800 с.
5. Демидович Е. М. Основы алгоритмизации и программирования. Язык Си. (Учебное пособие). – Санкт-Петербург: "БХВ Петербург". – 2006. – 439 с.

6. ІНДИВІДУАЛЬНІ ЗАВДАННЯ

Вважаючи, що введене речення з клавіатури складається з довільної кількості слів, між якими є довільна кількість пробілів, і закінчується речення крапкою, написати програму для розв'язання завдання:

1. Посортувати всі слова тексту за першою буквою згідно з алфавітом і видрукувати їх у стовпчик.

2. Надрукувати введене речення трьома способами: а) великими літерами; б) починаючи кожне слово великою літерою; в) великі літери замінити малими, а малі - великими.
3. У введеному реченні визначити середню довжину слова.
4. Визначити відсоток вживання у введеному реченні кожної з голосних літер. Результат записати в спадному порядку.
5. Сформувати нове речення зі слів введеного, в яких немає вказаної користувачем літери.
6. Ввести два речення. Поміняти місцями передостаннє слово першого речення на перше слово другого речення.
7. Ввести речення і ключове слово. Надрукувати всі слова з введеного речення, які не містять літер із заданого ключового слова, або вивести повідомлення про їх відсутність.
8. Визначити і надрукувати два найкоротших слова з введеного речення.
9. Надрукувати ті слова з введеного речення, в яких є подвоєння літер, або вивести повідомлення про їх відсутність.
10. Вилучити з введеного речення всі слова, які містять задану користувачем комбінацію з двох символів.
11. Замінити у введеному реченні слово з заданим порядковим номером відповідною кількістю однакових заданих користувачем символів. У випадку, коли номер перевищує введену кількість слів друкувати відповідне повідомлення про реальну кількість слів у введеному реченні.
12. Визначити і надрукувати найкоротше та найдовше слово з введеного речення.
13. Поміняти місцями два слова з введеного речення, порядкові номери яких задає користувач.
14. Визначити і надрукувати слово з введеного речення, в якому найбільше разів зустрічається задана літера.
15. Вилучити з введеного речення слово, задане своїм порядковим номером. У випадку, коли номер перевищує введену кількість слів друкувати відповідне повідомлення про реальну кількість слів у введеному реченні.
16. Порахувати кількість різних букв у введеному реченні.
17. Надрукувати всі слова з введеного речення, які містять понад 8 літер.
18. Сформувати колонку зі слів введеного речення, роздрукувавши кожне слово в інверсному порядку (ззаду - наперед).
19. Сформувати і надрукувати нове речення зі зворотнім до введеного порядком слів.
20. Надрукувати ті слова з введеного речення, які складаються з усіх різних літер, або вивести повідомлення про їх відсутність.
21. Порахувати кількість заданих трьох літер у введеному реченні.
22. Ввести речення і ключове слово. Визначити, чи є у введеному реченні слово, яке складається з тих самих літер, що й задане ключове слово.
23. Ввести два речення. Визначити і надрукувати літери з обох речень, які не є спільними. Відповідні заголовні та малі літери вважаються однаковими.
24. Замінити у введеному реченні четверте по порядку слово на задане користувачем. У випадку, коли кількість слів менша чотирьох, друкувати відповідне повідомлення про реальну кількість слів у введеному реченні.
25. Порахувати кількість приголосних у введеному реченні.
26. Визначити і надрукувати два найдовших слова з введеного речення.