**Введення і виведення одного символу**

В даному розділі ми розглянемо функції, що застосовуються при введенні і виведенні. Крім того, ми розглянемо інші аспекти цього поняття. Під функціями введення-виведення маються на увазі функції, які виконують транспортування даних в програму і з неї. Ми вже використовували дві такі функції: printf( ) і scanf( ). Тепер розглянемо кілька інших можливостей, що надаються мовою Сі.

Функції вводу-виводу не входять до визначення мови Сі. Їх розробка покладена на програмістів, які реалізовують компілятор з мови Сі. З іншого боку, вигода використання стандартного набору функцій вводу-виводу на всіх системах очевидна. Це дає можливість писати програми що переносяться, які легко можна застосовувати на різних машинах. У мові Сі є багато функцій вводу-виводу такого типу, наприкладprintf( ) і scanf( ). Нижче ми розглянемо функції getchar( ) і putchar( ).

Ці дві функції здійснюють введення і виведення одного символу при кожному зверненні до них. Цей спосіб введення даних краще відповідає можливостям машини. Більш того, такий підхід є основою побудови більшості програм обробки текстів, які є послідовностями звичайних слів. Ми побачимо, як можна застосовувати ці функції в програмах, які займаються підрахунком символів, читанням і копіюванням файлів. Дізнаємося про буфери, ехо-друк і перемикання вводу-виводу.

*Функція* getchar( ) отримує один символ, що надходить з пульта терміналу, і передає його програмі , яка виконується в даний момент. *Функція* putchar( ) отримує один символ, що надходить з програми, і пересилає його для виведення на екран. Розглянемо приклад програми, яка приймає один символ з клавіатури, і виводить його на екран:

/\*введення-виведення\*/

#include <stdio.h>

int main( )

{

char ch;

ch=getchar( ); /\*\*\*1\*\*\*/

putchar(ch); /\*\*\*2\*\*\*/

}

Для більшості систем специфікація функції getchar і putchar міститься в файлі stdio.h, тому ми вказали даний файл в програмі. *Функція* getchar( ) аргументів не має, тобто при її виклику в круглі дужки не поміщається ніяка величина. Вона просто отримує черговий символ що надходить, і сама повертає його значення програмі яка виконується. Оператор, наведений в рядку 1, привласнює значення функції getchar( ) змінній ch. *Функція* putchar( ) має один аргумент. При її виклику необхідно в дужках вказати символ, який потрібно вивести на друк. Аргументом може бути одиночний символ (включаючи знаки, що подаються керуючими послідовностями), змінна або функція, значенням якої є одиночний символ. Знову ж правильним зверненням до функції putchar( ) є вказівка будь-якого з цих аргументів при її виклику:

putchar('D');

putchar('\n');

putchar('\007');

putchar(ch); /\* змінна типу char \*/

putchar(getchar( ));

Модифицируем нашу программу:

#include <stdio.h>

int main( )

{

putchar(getchar( ));

}

Такий запис дуже компактним і не вимагає введення допоміжних змінних. В результаті компіляції така програма виявляється ефективнішою.

**Буфери**

При виконанні даної програми (будь-який з двох її версій), що символ вводиться в одних обчислювальних системах негайно з'явиться на екрані (ехо-друк), в інших же обчислювальних системах нічого не відбувається до тих пір, поки ми не натиснемо клавішу Enter. Перший випадок відноситься до **небуферізованного (прямого) введення**, що означає, що виведений символ виявляється негайно доступним програмі що очікує. Другий випадок є прикладом **буферизованного введення**, коли введені символи збираються і поміщаються в деякій області тимчасової пам'яті, звану **буфером**.

Натискання клавіші Enter призводить до того, що блок символів, або один символ, стає доступним програмі. У нашій програмі застосовується тільки перший символ, оскільки функція getchar( ) викликається в ній один раз.

Навіщо потрібні буфери? По-перше, виявляється, що передачу кількох символів у вигляді одного блоку можна здійснити набагато швидше, ніж передавати їх послідовно по одному. По-друге, якщо при введенні символів допущена помилка, ми зможемо скористатися коригуючими засобами терміналу, щоб її виправити. І коли ми натиснемо клавішу Enter, буде проведена передача відкоригованої рядки. Однак, для деяких діалогових програм небуферізованне введення може виявитися прийнятним. Наприклад, в програмах обробки текстів було б бажано, щоб кожна команда вводилася, як тільки ми натискаємо відповідну клавішу. Тому як буферізованне, так і небуферізованне введення має свої переваги.

Розглянемо вивід на друк груп символів. Бажано, щоб в будь-який момент можна було зупинити роботу програми. Для цього напишемо програму так, щоб вона припиняла роботу при отриманні будь-якого спеціального символу, наприклад "!":

/\* ввід-вивід \*/

/\* ввід та друк символів до

надходження завершаючого символу \*/

#include <stdio.h>

#define STOP '!' /\*дає символу '!' символічне ім’я \*/

int main( )

{

char ch;

ch=getchar( ); /\*\*\*9\*\*\*/

while(ch != STOP) { /\*\*\*10\*\*\*/

putchar( ch); /\*\*\*11\*\*\*/

ch=getchar( ); /\*\*\*12\*\*\*/

}

}

В даному прикладі при першому проходженні тіла циклу функція putchar () отримує значення свого аргументу в результаті виконання оператора, розташованого в рядку 9. Надалі, аж до завершення роботи циклу, значенням цього аргументу є символ, який передається програмі функцією getchar, розташованої в рядку 12 . Цикл while буде здійснювати читання і друк символів до тих пір, поки не надійде ознака STOP.

|  |  |
| --- | --- |
| ! | Програма, приведена нижче, робить те ж саме, але стиль її написання краще відповідає стилю мови Сі:  /\* ввід-вивід \*/  #include <stdio.h>  #define STOP '!'  int main( )  {  char ch;  while ((ch=getchar( )) != STOP) /\*\*\*8\*\*\*/  putchar(ch);  }  Один рядок 8 цієї програми замінює рядки 9, 10, 12 попередньої програми. |

### Читання одного рядка

*Ускладнимо приклад введення-виведення*:

/\* підрахунок символів \*/

#include <stdio.h>

#define STOP '!'

int main( )

{

char ch;

/\*ініціалізація лічильників символів 0 \*/

int count = 0;

while ((ch=getchar( )) != STOP) {

putchar(ch);

count++; /\* додати 1 до лічильника \*/

}

printf("\n Всього було прочитано %d символа.\n",

count);

}

Якщо ми хочемо просто підрахувати число введених символів без відображення їх на екрані, функцію putchar () можна опустити.

Замінимо ознаку закінчення введення даних, використовуємо символ новий рядок \ n. Для цього потрібно перевизначити ознаку STOP:

#define STOP '\n'

Символ новий рядок пересилається при натисканні клавіші Enter. Припустимо, що ми внесли зазначену зміну в програму "підрахунок символів", а потім при виконанні ввели наступний рядок:

На екрані тридцять чотири символи.[Enter]

У відповідь на екрані з'являться наступні рядки:

Всього було прочитано 34 символи.

Ознака, що з'являється в результаті натискання клавіші Enter, не входить в число символів 34, підрахованих програмою, оскільки підрахунок здійснюється всередині циклу. Тепер у нас є програма, яка може прочитати один рядок.

### Читання файлу

Якщо нам потрібно читати великі порції даних, наприклад з файлу, яким повинен бути ознака STOP? Це повинен бути такий символ, який зазвичай не використовується в тексті і, отже, не призводить до ситуації, коли він випадково зустрінеться при введенні, і робота програми буде зупинена раніше, чим би ми хотіли. Файлом можна назвати ділянку пам'яті, в який поміщена деяка інформація. Зазвичай файл зберігається в деякій довготривалій пам'яті, наприклад на гнучких або жорстких дисках або на магнітній стрічці. Щоб відзначати, де кінчається один файл і починається інший, корисно мати спеціальний символ, який вказує на кінець файлу, щоб відзначати кінець файлу і починати інший. Це повинен бути символ, який не може з'явитися десь в середині файлу. Рішенням зазначеної проблеми є введення спеціального ознаки, званого " End-of-File ", кінець файлу, або EOF. Вибір конкретної ознаки EOF залежить від типу системи. Він може складатися навіть з кількох символів. Зазвичай визначення EOF міститься в файлі <stdio.h>. Загальновживаним є визначення

#define EOF (-1)

Приклад:

/\* ввід-вивід\_ф \*/

#include <stdio.h>

int main( )

{

int ch;

while ((ch = getchar( )) != EOF)

putchar(ch);

}

**Це потрібно пам’ятати**:

1. Не потрібно самим визначати ознаку EOF. Вона описана в файлі <stdio.h>.
2. Ми можемо не цікавитися фактичним значенням символу EOF, оскільки директива #define, що міститься в файлі <stdio.h>, дозволяє нам використовувати його символічне уявлення.
3. Ми змінили в нашій програмі тип змінної ch з char на int. Це ми зробили, тому що значенням змінних типу char є ціле без знака в діапазоні від 0 до 255, а ознака EOF може мати числове значення -1. Ця величина є неприпустимою для змінної типу char. Функція getchar () фактично повертає значення типу int, тому вона в змозі прочитати символ EOF.
4. Змінна ch цілого типу ніяк не може вплинути на роботу функції putchar (). Вона просто виводить на друк символьний еквівалент значення аргументу.
5. При роботі з даною програмою, коли знаки вводяться з клавіатури, необхідно вміти вводити ознаку EOF. У більшості реалізацій операційної системи UNIX, наприклад, введення [CTRL/d] (натиснути на клавішу [d], тримаючи клавішу [CTRL]) інтерпретується як ознака EOF. У багатьох комп'ютерах для тієї ж мети використовується знак [CTRL/z].

Нехай ми ввели фразу з клавіатури. Наведемо результат роботи програми "вхід-вивод\_ф" в системі, з буферизованним введенням:

Попит на високопрофесійних фахівців з інформаційних технологій  
Попит на високопрофесійних фахівців з інформаційних технологій  
росте як з боку державних, так і  
приватних компаній зростає як з боку  
державних, так і приватних компаній [CTRL/z]

### Кожен раз при натисканні клавіші Enter проводиться обробка символів, які потрапили в буфер, і копія рядка виводиться на друк. Це продовжується до тих пір, поки ми не введемо ознаку EOF. Програма "ввід-вивід\_ф" здійснює вивід на екран символів незалежно від того, звідки вони надходять. Наша програма могла б переглядати вміст файлів, створювати нові файли і отримувати копії файлів. Вирішення цих проблем - в управлінні введенням і виведенням.

### Перемикання та робота з файлами

***Поняття введення-виведення*** включає в себе функції, дані і пристрої. Розглянемо, наприклад, програму "вхід-вивод\_ф". У ній використовується функція getchar (), що здійснює введення, причому пристрій введення - клавіатура (відповідно до наших припущеннь), а вихідні дані - окремі символи. Змінимо джерело надходження в програму даних. За замовчуванням Сі-програма розглядає стандартне введення як джерело надходження даних.

***Стандартним вводом*** називається пристрій, прийнятий в якості звичайного засобу введення даних в машину. Це може бути пристрій читання даних з магнітної стрічки телетайпа або термінал. Ми можемо самі вибирати пристрій даних з будь-якого джерела. Ну, наприклад, ми можемо написати в програмі, що джерело вхідних даних - файл, а не клавіатура.

Існують два способи написання програм, що працюють з файлами. Перший спосіб полягає в явному використанні спеціальних функцій, які відкривають і закривають файли, організовують читання і запис даних і т.д. Це питання ми будемо обговорювати в 15 лекції. Другий спосіб полягає в тому, щоб використовувати програму, спроектовану спочатку в припущенні, що дані в неї вводяться з клавіатури і виводяться на екран, але переключити введення і виведення на інші інформаційні канали, наприклад, з файлу в файл.

Цей спосіб в деяких відносинах має менші можливості, ніж перший, але зате набагато простіший у використанні. Операція перемикання - це засіб OC UNIX, а не самої мови Сі. Але вона виявилася настільки корисною, що при перенесенні компілятора з мови Сі на інші обчислювальні системи часто разом з ним переноситься і ця операція. Багато з новостворених операційних систем, таких, як MS-DOS 2, включають в себе певний засіб. Спочатку ми обговоримо можливості цієї операції в OC UNIX, а потім і в інших системах.

**Перемикання виведення.** Припустимо, ми здійснили компіляцію програми "вхід-вивод\_ф" і помістили виконуваний об'єктний код в файл з ім'ям get\_put. Потім, щоб запустити цю програму, ми вводимо з терміналу тільки ім'я файлу

get\_put

і програма виконується так, як було описано вище, тобто отримує в якості вхідних даних символи, що вводяться з клавіатури. Тепер припустимо, що ми хочемо подивитися, як наша програма працює з текстовим файлом з ім'ям words. **Текстовий файл** - це файл, який містить певний текст, тобто дані у вигляді символів. Це може бути, наприклад, розповідь або програма на мові Сі. Файл, що містить команди на машинному мові, не є текстовим. Так як наша програма займається обробкою символів, то вона повинна використовуватися разом з текстовим файлом. Для цього треба ввести наступну команду:

get\_put < words

Символ <служить позначенням операції перемикання, що використовується в OC UNIX. Виконання зазначеної операції призводить до того, що вміст файлу words буде направлено в файл з ім'ям get\_put. Сама програма "вхід-вивод\_ф" не знає, що вхідні дані надходять з деякого файлу, а не з терміналу. На її вхід просто надходить потік символів, вона читає їх, і послідовно, по одному, виводить на друк до тих пір, поки не зустріне ознака EOF. Якщо ми наберемо команду

get\_put < words

то в результаті на екрані може з'явитися, наприклад, наступний текст:

Аналіз даних, прогнози, організація зв'язку,  
створення програмного забезпечення, побудова  
моделей процесів - ось далеко не повний список  
областей застосування знань для комп'ютерних  
фахівців.

якщо цей текст знаходиться в текстовому файлі words.

#### Переключення вводу

Тепер розглянемо випадок, коли нам потрібно ввести текст з клавіатури в файл з ім'ям my\_words. Для цього ми повинні ввести команду

get\_put > my\_words

і почати введення символів. Символ> служить позначенням операції перемикання, що використовується в OC UNIX. Її виконання призводить до того, що створюється новий файл з ім'ям my\_words, а потім результат роботи програми "вхід-вивод\_ф", що представляє собою копію символів, що вводять, направляється в даний файл. Якщо файл з ім'ям my\_words вже існує, він зазвичай знищується, і замість нього створюється новий. На екрані з'являються вводяться слова. Їх копії будуть направлені в зазначений файл. Щоб закінчити роботу, ми вводимо EOF, в OC UNIX це зазвичай [CTRL/d].

#### Комбіноване перемикання

Припустимо, що ми хочемо створити копію файлу my\_words і назвати його my\_words2. Для цього потрібно ввести команду:

get\_put < my\_words > my\_words2

І завдання буде виконане.

Команда

get\_put > my\_words2 < my\_words

#### **призведе до такого ж результату, оскільки порядок вказівки операцій перемикання не має значення. Не можна в одній команді використовувати один і той же файл і для введення, і для виведення.**

*Операційні системи, відмінні від OC UNIX*

Чим відрізняються інші операційні системи від OC UNIX? Всі відмінності можна розділити на дві групи:

1. В інших операційних системах реалізована операція перемикання.
2. Компілятори з мови Сі надають можливість використовувати операцію перемикання.

У нас немає можливості розглянути всі компілятори з мови Сі. Однак в п'яти з шести версій компілятора, призначених для мікрокомп'ютерів, для вказівки операції перемикання використовуються символи <>. Операція перемикання відрізняється від аналогічної операції в двох аспектах:

1. Зазначена операція виконується при роботі програм, написаних тільки на Сі, в той час як в OC UNIX вона може використовуватися при роботі будь-якої програми.
2. Між ім'ям файлу з відкомпільованої програми і знаком операції повинен бути один пробіл, а між знаком операції та ім'ям файлу пробіл повинен бути відсутнім. наприклад:

get\_put <words