**7. Введення і виведення**

Засоби введення / виводу не є складовою частиною мови "с", так що ми не виділяли їх в нашій попередній лекції. Однак реальні програми взаємодіють зі своїм довкіллям набагато більш складним чином, ніж ми бачили досі. У цій лекції буде описана "стандартна бібліотека вводу / виводу", тобто набір функцій, розроблених для забезпечення стандартної системи введення / виводу для "с" - програм.

Ці функції призначені для зручності програмного інтерфейсу, і все ж відображають тільки ті операції, які можуть бути забезпечені на більшості сучасних операційних систем. Процедури досить ефективні для того, щоб користувачі рідко відчували необхідність обійти їх "заради ефективності", як би не була важлива конкретна задача.

І, нарешті, ці процедури задумані бути "переносними" в тому сенсі, що вони повинні існувати в сумісному вигляді на будь-якій системі, де є мова "с", і що програми, які обмежують свої взаємодії з системою можливостями, наданими стандартної бібліотекою, можна буде перенесена з однієї системи на іншу по суті без змін.

Ми тут не будемо намагатися описати всю бібліотеку введення / виведення; ми більш зацікавлені в тому, щоб продемонструвати сутність написання "с" - програм, які взаємодіють зі своєю операційною середовищем.

**7.1. Звернення до стандартної бібліотеки**

Кожен вихідний файл, який звертається до функції зі стандартної бібліотеки, повинен поблизу початку містити рядок

#include <stdio.h>

в файлі stdio.h визначаються деякі макроси і змінні, використовувані бібліотекою вводу / виводу. Використання кутових дужок замість звичайних подвійних лапок - вказівка компілятору шукати цей файл в довіднику, що містить заголовки стандартної інформації (на системі UNIX зазвичай /usr/include ).

Крім того, при завантаженні програми може виявитися необхідним вказати бібліотеку явно; на системі PDP-11 UNIX, наприклад, команда компіляції програми мала б вигляд:

cc вихідні файли і т.д. -ls

де -ls вказує на завантаження з стандартної бібліотеки.

#### 7.2. Стандартне введення і виведення - функції GETCHAR і PUTCHAR

Найпростіший механізм введення полягає в читанні по одному символу за раз з "стандартного введення", зазвичай з терміналу користувача, за допомогою функції getchar. функція getchar () при кожному до неї зверненні повертає наступний символ, що вводиться. У більшості середовищ, які підтримують мову "с", термінал може бути замінений деяким файлом за допомогою позначення <: якщо деяка програма prog використовує функцію getchar то командний рядок

prog<infile

призведе до того, що prog буде читати з файлу infile, а не з терміналу. Перемикання введення робиться таким чином, що сама програма prog не помічає зміни; зокрема рядок "<infile" не включається в командний рядок аргументів на argv. Перемикання введення виявляється непомітним і в тому випадку, коли висновок надходить з іншої програми за допомогою поточного (pipe) механізму; командний рядок

otherprog | prog

проганяє дві програми, otherprog і prog, і організовує так, що стандартним вводом для prog служить стандартний висновок otherprog.

функція getchar повертає значення EOF, коли вона потрапляє на кінець файлу, який би ввивід вона при цьому не зчитувала. Стандартна бібліотека вважає символічну константу EOF рівній -1 (за допомогою #define в файлі stdio.h), але перевірки слід писати в термінах EOF, а не -1, щоб уникнути залежності від конкретного значення.

Висновок можна здійснювати за допомогою функції putchar (c), помещающей символ 'с' в "стандартний висновок", який за замовчуванням є терміналом. Висновок можна направити в певний файл за допомогою позначення> якщо prog використовує putchar, то командний рядок

prog>outfile

приведе до запису стандартного виводу в файл outfile, а не на термінал. На системі UNIX можна також використовувати потоковий механізм. Рядок

prog | anotherprog

поміщає стандартний висновок prog в стандартне введення anotherprog. І знову prog НЕ буде поінформована про зміну напряму.

Висновок, який здійснюється функцією printf, також надходить в стандартний висновок, і звернення до putchar і printf можуть чергуватися.

Вражаюча кількість програм читає тільки з одного вхідного потоку і пише тільки в один вихідний потік; для таких програм введення і виведення за допомогою функцій getchar, putchar і printf може виявитися цілком адекватним і для початку виразно достатнім. Це особливо справедливо тоді, коли є можливість вказівки файлів для введення та виведення і потоковий механізм для зв'язку виведення однієї програми з введенням інший. Розглянемо, наприклад, програму lower, яка перетворює великі літери зі свого введення в рядкові:

#include <stdio.h>

main() /\* convert input to lower case \*/

{

int c;

while ((c = getchar()) != EOF)

putchar(isupper(c) ? tolower(c) : c);

}

"Функції" isupper і tolower насправді є макросами, визначених у stdio.h. Макрос isupper перевіряє, чи є його аргумент літерою з верхнього регістру, і повертає нульове значення, якщо це так, і нуль в іншому випадку. Макрос tolower перетворює букву з верхнього регістру в ту ж букву нижнього регістра. Незалежно від того, як ці функції реалізовані на конкретній машині, їх зовнішня поведінка абсолютно однаково, так що використовують їх програми позбавлені знання символьного набору.

Якщо Ви бажаєте перевести кілька файлів, то можна зібрати ці файли за допомогою програми, подібної утиліти cat системи *UNIX*,

cat file1 file2 ... | lower>output

і уникнути тим самим питання про те, як звернутися до цих файлів з програми. (Програма cat наводиться пізніше в цій лекції).

Крім того зазначимо, що в стандартній бібліотеці введення / виведення "функції" getchar і putchar насправді можуть бути макросами. Це дозволяє уникнути накладних витрат на звернення до функції для обробки кожного символу. В "лекції №8" ми продемонструємо, як це робиться.

#### 7.3. Форматний висновок - функція printf

#### Дві функції: printf для виведення і scanf для введення (наступний розділ) дозволяють перетворювати чисельні величини в символьне уявлення і назад. Вони також дозволяють генерувати й інтерпретувати форматні рядки. Ми вже всюди в попередніх лекціях неформально використовували функцію printf; тут наводиться більш повний і точний опис. Функція

printf(control, arg1, arg2, ...)

перетворює, визначає формат і друкує свої аргументи в стандартний висновок під керуванням рядка control. Керуючий рядок містить два типи об'єктів: звичайні символи, які просто копіюються у вихідний потік, і специфікації перетворень, кожна з яких викликає перетворення і друк чергового аргументу printf.

Кожна Специфікація перетворення починається з символу% і закінчується символом перетворення. Між% і символом перетворення можуть перебувати:

* знак мінус, який вказує про вирівнювання перетвореного аргументу по лівому краю його поля.
* Рядок цифр, що задає мінімальну ширину поля. Перетворене число буде надруковано в поле принаймні цієї ширини, а якщо необхідно, то і в більш широкому. Якщо перетворений аргумент має менше символів, ніж зазначена ширина поля, то він буде доповнений зліва (або справа, якщо було вказано вирівнювання по лівому краю) заповнюють символами до цієї ширини. Заповнює символом зазвичай є пробіл, а якщо ширина поля вказується з лідируючим нулем, то цим символом буде нуль (лідируючий нуль в даному випадку не означає восьмеричної ширини поля).
* Точка, яка відокремлює ширину поля від наступного рядка цифр.
* Рядок цифр (точність), яка вказує максимальне число символів рядка, які повинні бути надруковані, або число друкованих праворуч від десяткового дробу цифр для змінних типу float або double.
* Модифікатор довжини l, який вказує, що відповідний елемент даних має тип long, а не int.

Нижче наводяться символи перетворення і їх зміст:

* d - *аргумент* перетвориться до десяткового виду.
* o - *аргумент* перетвориться в беззнакову восьмеричну форму (без лідируючого нуля).
* x - *аргумент* перетвориться в беззнакову шестнадцатеричную форму (без лідируючих 0x ).
* u - *аргумент* перетвориться в беззнакову десяткову форму.
* c - *аргумент* розглядається як окремий символ.
* s - *аргумент* являеться рядком: символи рядка друкуються до тих пір, поки не буде досягнуто нульовий символ, чи не буде надруковано кількість символів, вказане в специфікації точності.
* e - *аргумент*, розглянутий як змінна типу float або double, перетворюється в десяткову форму у вигляді [-] m.nnnnnne [+ -] xx, де довжина рядка з n визначається зазначеної точністю. Точність за замовчуванням дорівнює 6.
* f - *аргумент*, розглянутий як змінна типу float або double, перетворюється в десяткову форму у вигляді [-] mmm.nnnnn, де довжина рядка з n визначається зазначеної точністю. Точність за замовчуванням дорівнює 6. Відзначимо, що ця точність не визначає кількість друкованих в форматі f значущих цифр.
* g - Використовується або формат% e або% f, який коротше; незначущі нулі не друкуються. Якщо йде за% символ не є символом перетворення, то друкується сам цей символ; отже, символ% можна надрукувати, вказавши %%.

Більшість з форматних перетворень очевидно і було проілюстровано в попередніх лекціях. Єдиним винятком є те, як точність взаємодіє з рядками. Наступна таблиця демонструє вплив завдання різних специфікацій на друк "hello, world" (12 символів). Ми помістили двокрапки навколо кожного поля для того, щоб ви могли бачити його протяжність.

:%10s: :hello, world:

:%10-s: :hello, world:

:%20s: : hello, world:

:%-20s: :hello, world :

:%20.10s: : hello, wor:

:%-20.10s: :hello, wor :

:%.10s: :hello, wor:

Застереження: printf використовує свій перший аргумент для визначення числа наступних аргументів і їх типів. Якщо кількість аргументів виявиться недостатнім або вони будуть мати невідповідні типи, то виникне плутанина і ви отримаєте безглузді результати.

**Вправа 7-1**

Напишіть програму, яка буде друкувати розумним чином довільний введення. Як мінімум вона повинна друкувати неграфічні символи в вісімковому або шістнадцятковому вигляді (відповідно до прийнятих у вас звичаями) і складати довгі рядки.

#### 7.4. Форматный ввод - функция SCANF

Функція що дійснює введення scanf є аналогом printf і дозволяє проводити в зворотному напрямку багато з тих же самих перетворень. Функція

scanf(control, arg1, arg2, ...)

читає символи зі стандартного вводу, інтерпретує їх відповідно до формату, зазначеному в аргументі control, і поміщає результати в інші аргументи. Керуючий аргумент описується нижче; інші аргументи, кожен з яких повинен бути покажчиком, визначають, куди слід помістити відповідним чином перетворений введення.

Керуючий рядок зазвичай містить специфікації перетворення, які використовуються для безпосередньої інтерпретації вхідних послідовностей. Керуючий рядок може містити:

* прогалини, табуляції або символи нового рядка ("символи порожніх проміжків"), які ігноруються.
* Звичайні символи (НЕ%), які передбачаються повинні збігатися з наступними відмінними від символів порожніх проміжків символами вхідного потоку.
* Специфікації перетворення, що складаються з символу%, необов'язкового символу придушення присвоювання \*, необов'язкового числа, що задає максимальну ширину поля і символу перетворення.

Специфікація перетворення управляє перетворенням наступного поля введення. Нормально результат поміщається в змінну, яка вказується відповідним аргументом. Якщо, проте, за допомогою символу \* вказано придушення привласнення, то це поле введення просто пропускається і ніякого привласнення не проводиться. Поле введення визначається як рядок символів, які відмінні від символів простих проміжків; воно триває або до наступного символу чистого місця, або поки не буде вичерпана ширина поля, якщо вона вказана. Звідси випливає, що при пошуку потрібного їй введення, функція scanf буде перетинати кордони рядків, оскільки символ нового рядка входить в число порожніх проміжків.

Символ перетворення визначає інтерпретацію поля введення; згідно з вимогами заснованої на виклик за значенням семантики мови "с" відповідний аргумент повинен бути покажчиком. Допускаються наступні символи перетворення:

* d - на вводі очікується десяткове ціле; відповідний аргумент повинен бути покажчиком на ціле.
* o - На введенні очікується вісімкове ціле (з лідируючим нулем або без нього); відповідний аргумент повинен бути покажчиком на ціле.
* x - На введенні очікується шестнадцатерічне ціле (з лідируючими 0x або без них); відповідний аргумент повинен бути покажчиком на ціле.
* h - На введенні очікується ціле типу short; відповідний аргумент повинен бути покажчиком на ціле типу short.
* c - Очікується окремий символ; відповідний аргумент повинен бути покажчиком на символи; Наступний введений символ поміщається в вказане місце. Звичний пропуск символів порожніх проміжків в цьому випадку пригнічується; для читання наступного символу, який не є символом чистого місця, користуйтеся специфікацією перетворення %1s.
* s - Очікується символьний рядок; відповідний аргумент повинен бути покажчиком символів, який вказує на масив символів, який досить великий для прийняття рядка і додається в кінці символу \0.
* f - Очікується число з плаваючою крапкою; відповідний аргумент повинен бути покажчиком на змінну типу float.
* Е - символ перетворення e є синонімом для f. Формат введення змінної типу float включає необов'язковий знак, рядок цифр, можливо містить десяткову точку і необов'язкове поле експоненти, що складається з букви e, за якою слідує ціле, можливо має знак.

Перед символами перетворення d, o і x може стояти l, яка означає, що в списку аргументів повинен знаходитися покажчик на змінну типу long, а не типу int. Аналогічно, буква l може стояти перед символами перетворення e або f, кажучи про те, що в списку аргументів повинен знаходитися покажчик на змінну типу double, а не типу float.

Наприклад, звернення

int i; float x; char name[50]; scanf("%d %f %s", &i, &x, name);

з рядком на введені

25 54.32e-1 thompson

призводить до присвоєння i значення 25, x - значення 5.432 і name - рядки "thompson", належним чином закінченою символом \ 0. Ці три поля введення можна розділити стількома пробілами, табуляціями і символами нових рядків, скільки ви забажаєте. Звернення

int i;

float x;

char name[50];

scanf("%2d %f %\*d %2s", &i, &x, name);

з введеням

56789 0123 45a72

присвоїть i значення 56, x - 789.0, пропустить 0123 і помістить в name рядок "45". При наступному зверненні до будь-якої процедури введення розгляд почнеться з a. У цих двох прикладах name є покажчиком і, отже, перед ним не потрібно поміщати знак &.

Як інший приклад перепишемо тепер елементарний калькулятор з "лекції №4", використовуючи для перетворення введення функцію scanf:

#include <stdio.h>

main() /\* rudimentary desk calculator \*/

{

double sum, v;

sum =0;

while (scanf("%lf", &v) !=EOF)

printf("\t%.2f\n", sum += v);

}

виконання функції scanf закінчується або тоді, коли вона вичерпує свій керуючий рядок, або коли певний елемент введення не збігається з керуючою специфікацією. В якості свого значення вона повертає число правильно співпадаючих і привласнених елементів введення.

Це число може бути використано для визначення кількості знайдених елементів введення. При виході на кінець файлу повертається EOF; підкреслимо, що це значення відмінно від 0, що наступний символ, що вводиться не задовольняє першої специфікації в керуючої рядку. При наступному зверненні до scanf пошук поновлюється безпосередньо за останнім введеним символом.

Заключне застереження: аргументи функції scanf повинні бути покажчиками. Безсумнівно найбільш поширена помилка полягає в написанні

scanf("%d", n);

замість

scanf("%d", &n);

#### 7.5. Форматне перетворення в пам'яті

Від функції scanf і printf походять функції sscanf і sprintf, які здійснюють аналогічні перетворення, але оперують з рядком, а не з файлом. Звернення до цих функцій мають вигляд:

sprintf(string, control, arg1, arg2, ...)

sscanf(string, control, arg1, arg2, ...)

Як і раніше, функція sprintf перетворює свої аргументи arg1, arg2 і т.д. Відповідно до формату, зазначеним в control, але поміщає результати в string, а не в стандартний висновок. Звісно, рядок string повинен бути досить великий, щоб прийняти результат. Наприклад, якщо name - це символьний масив, а n - ціле, то

sprintf(name, "temp%d", n);

створює в name рядок виду tempnnn, де nnn - значення n.

функція sscanf виконує зворотні перетворення - вона переглядає рядок string відповідно до формату в аргументі control і поміщає результуючі значення в аргументи arg1, arg2 і т.д. Ці аргументи повинні бути покажчиками. В результаті звернення

sscanf(name, "temp%d", &n);

змінна n отримує значення рядка цифр, наступних за temp в name.

**Вправа 7-2**

Перепишіть настільний калькулятор з "лекції №4", використовуючи для введення і перетворення чисел scanf і / або sscanf.

#### 7.6. Доступ до файлів

Всі досі написані програми читали зі стандартного вводу і писали в стандартний висновок, щодо яких ми припускали, що вони магічним чином надані програмою місцевої операційною системою.

Наступним кроком в питанні введення-виведення є написання програми, яка працює з файлом, який не пов'язаний заздалегідь з програмою. Однією з програм, яка явно демонструє потребу в таких операціях, є cat, яка об'єднує набір з декількох іменованих файлів в стандартний висновок. Програма cat використовується для виведення файлів на термінал і в якості універсального збирача введення для програм, які не мають можливості звертатися до файлів по імені. Наприклад, команда

cat x.c,y.c

друкує вміст файлів x.c і y.c в стандартний висновок.

Питання полягає в тому, як організувати читання з іменованих файлів, тобто, як зв'язати зовнішні імена, якими мислить користувач, з фактично читають дані операторами.

Ці правила прості. Перш ніж можна зчитувати з деякого файлу або записувати в нього, цей файл повинен бути відкритий за допомогою функції fopen зі стандартної бібліотеки. функція fopen бере зовнішнє ім'я (подібне x.c або y.c), проводить деякі обслуговуючі дії і переговори з операційною системою (деталі яких не повинні нас стосуватися) і повертає внутрішнє ім'я, яке повинно використовуватися при наступних читаннях з файлу або записах в нього.

Це внутрішнє ім'я, зване "покажчиком файлу", фактично є покажчиком структури, яка містить інформацію про файл, таку як місце розміщення буфера, поточна позиція символу в буфері, чи відбувається читання з файлу або запис в нього тощо. Його користувачі не зобов'язані знати ці деталі, тому що серед визначень для стандартного вводу-виводу, що отримуються з файлу stdio.h, міститься визначення структури з ім'ям file. Єдина необхідна для покажчика файлу опис демонструється прикладом:

file \*fopen(), \*fp;

Тут йдеться про те, що fp є покажчиком на file і fopen повертає покажчик на file. Зверніть увагу, що file є ім'ям типу, подібним int, а не ярлику структури; це реалізовано як typedef. (Подробиці того, як все це працює на системі UNIX, наведені в "лекції №8" ).

Фактичне звернення до функції fopen в програмі має вигляд:

fp=fopen(name,mode);

Першим аргументом функції fopen є "ім'я" файлу, яке задається в вигляді символьного рядка. Другий аргумент mode ("режим") також є символьним рядком, яка вказує, як цей файл буде використовуватися. Допустимими режимами є: читання ("r"), запис ("w") і додавання ("a").

Якщо ви відкриєте файл, який ще не існує, для запису або додавання, то такий файл буде створений (якщо це можливо).

Відкриття існуючого файлу на запис призводить до відкидання його старого вмісту. Спроба читання неіснуючого файлу є помилкою. Помилки можуть бути обумовлені і іншими причинами (наприклад, спробою читання з файлу, не маючи на те дозволу). При наявності будь-якої помилки функція повертає нульове значення покажчика null (яке для зручності також визначається у файлі stdio.h ).

Іншою необхідною річчю є спосіб читання або запису, якщо файл вже відкритий. Тут є кілька можливостей, з яких getc і putc є найпростішими. Функція getc повертає наступний символ з файлу; їй необхідний покажчик файлу, щоб знати, з якого файлу читати. Таким чином,

c=getc(fp)

поміщає в "C" наступний символ з файлу, вказаного за допомогою fp, і EOF, якщо досягнуто кінець файлу.

Функція putc, що є зверненням до функції getc,

putc(c,fp)

поміщає символ "c" в файл fp і повертає "c". Подібно до функцій getchar і putchar, getc і putc можуть бути макросами, а не функціями.

При запуску програми автоматично відкриваються три файли, які забезпечені певні покажчиками файлів. Цими файлами є стандартний ввід, стандартний висновок і стандартний висновок помилок; відповідні покажчики файлів називаються stdin, stdout і stderr. Зазвичай всі ці покажчики пов'язані з терміналом, але stdin і stdout можуть бути перенаправлені на файли або в потік (pipe), як описувалося в розділі 7.2. "лекциії №7" .

Функції getchar і putchar можуть бути визначені в терміналах getc, putc, stdin і stdout наступним чином: #define getchar () getc (stdin) #define putchar (c) putc (c, stdout). При роботі з файлами для форматного введення і виведення можна використовувати функції fscanf і fprintf. Вони ідентичні функціям scanf і printf, за винятком того, що першим аргументом є покажчик файлу, що визначає той файл, який буде читатися або куди буде вестися запис; керуючий рядок буде другим аргументом.

Покінчивши з попередніми зауваженнями, ми тепер в змозі написати програму cat для конкатенації файлів. Використовувана тут основна схема виявляється зручною у багатьох програмах: якщо є аргументи у командному рядку, то вони обробляються послідовно. Якщо такі аргументи відсутні, то обробляється стандартне введення. Це дозволяє використовувати програму як самостійно, так і як частина більшої завдання.

#include <stdio.h>

main(argc, argv) /\*cat: concatenate files\*/

int argc;

char \*argv[];

{

file \*fp, \*fopen();

if(argc==1) /\*no args; copy standard input\*/

filecopy(stdin);

else

while (--argc > 0)

if ((fp=fopen(\*++argv,"r"))==null) {

printf("cat:can't open %\n",\*argv);

break;

} else {

filecopy(fp);

fclose(fp);

}

}

filecopy(fp) /\*copy file fp to standard output\*/

file \*fp;

{

int c;

while ((c=getc(fp)) !=EOF)

putc(c, stdout);

}

покажчики файлів stdin і stdout заздалегідь визначені в бібліотеці вводу-виводу як стандартний ввід і стандартний висновок; вони можуть бути використані в будь-якому місці, де можна використовувати об'єкт типу file \*. Вони однак є константами, а не змінними, так що не намагайтеся їм що-небудь привласнювати.

функція fclose є зворотною по відношенню до fopen; вона розриває зв'язок між покажчиком файлу і зовнішнім ім'ям, встановлену функцією fopen, і вивільняє покажчик файлу для іншого файлу. Більшість операційних систем мають деякі обмеження на число одночасно відкритих файлів, якими може розпоряджатися програма. Тому, то як ми вчинили в cat, звільнивши непотрібні нам більш об'єкти є хорошою ідеєю. Є й інша причина для застосування функції fclose до вихідного файла - вона викликає видачу інформації з буфера, в якому putc збирає висновок. (При нормальному завершенні роботи програми функція fclose викликається автоматично для кожного відкритого файла).

#### 7.7. Обробка помилок- STDERR і EXIT

Обробка помилок в cat неідеальна. Незручність полягає в тому, що якщо один з файлів з деякої причини виявляється недоступним, діагностичне повідомлення про це друкується в кінці об'єднаного виведення. Це прийнятно, якщо висновок надходить на термінал, але не годиться, якщо висновок надходить в деякий файл або через поточний (pipeline) механізм в іншу програму.

Щоб краще обробляти таку ситуацію, до програми точно таким же чином, як stdin і stdout, приєднується другий вихідний файл під назвою stderr. Якщо це взагалі можливо, висновок, записаний у файлі stderr, з'являється на терміналі користувача, навіть якщо стандартний висновок направляється в інше місце.

Давайте переробимо програму cat таким чином, щоб повідомлення про помилки писалися в стандартний файл помилок.

"include <stdio.h>

main(argc,argv) /\*cat: concatenate files\*/

int argc;

char \*argv[];

{

file \*fp, \*fopen();

if(argc==1) /\*no args; copy standard input\*/

filecopy(stdin);

else

while (--argc > 0)

if((fp=fopen(\*++argv,"r#))==null) {

fprintf(stderr,

"cat: can't open,%s\n", argv);

exit(1);

} else {

filecopy(fp);

}

exit(0);

}

Програма повідомляє про помилки двома способами. діагностичне повідомлення, що видається функцією printf, надходить в stderr і, таким чином, виявляється на терміналі користувача, а не зникає в потоці (pipeline) або в вихідному файлі.

Програма також використовує функцію exit зі стандартної бібліотеки, звернення до якої викликає завершення виконання програми. Аргумент функції exit доступний будь-якій програмі, що звертається до даної функції, так що успішне або невдале завершення даної програми може бути перевірено іншою програмою, що використовує цю як підзадачі. За угодою величина 0 в якості значення, що повертається свідчить про те, що все в порядку, а різні ненульові значення є ознаками ненормальних ситуацій.

Функція exit викликає функцію fclose для кожного відкритого вихідного файлу, з тим щоб вивести всю вміщену в буфери вихідну інформацію, а потім викликає функцію \_exit. Функція \_exit призводить до негайного завершення без очищення будь-яких буферів; звичайно, при бажанні до цієї функції можна звернутися безпосередньо.

#### 7.8. Введення і виведення рядків

Стандартна бібліотека містить функцію fgets, абсолютно аналогічну функції getline, яку ми використовували на всьому протязі книги. В результаті звернення

fgets(line, maxline, fp)

наступний рядок введення (включаючи символ нового рядка) зчитується з файлу fp в символьний масив line; найбільше maxline-1 символ буде прочитаний. Рядок закінчується символом \ 0. Нормально функція fgets повертає line; в кінці файлу вона повертає null. (Наша функція getline повертає довжину рядка, а при виході на кінець файлу - нуль).

Призначена для виведення функція fputs записує рядок (яка має утримувати символ нового рядка) в файл:

fputs(line, fp)

Щоб показати, що у функціях типу fgets і fputs немає нічого таємничого, ми наводимо їх нижче, скопійованими безпосередньо зі стандартної бібліотеки вводу-виводу:

#include <stdio.h>

char \*fgets(s,n,iop) /\*get at most n chars from iop\*/

char \*s;

int n;

register file \*iop;

{

register int c;

register char \*cs;

cs = s;

while(--n>0&&(c=getc(iop)) !=EOF)

if ((\*cs++ = c)=='\n')

break;

\*cs = '\0';

return((c==EOF && cs==s) ? null : s);

}

fputs(s,iop) /\*put string s on fils iop\*/

register char \*s;

register file \*iop;

{

register int c;

while (c = \*s++)

putc(c,iop);

}

**Вправа 7-3**

Напишіть програму порівняння двох файлів, яка буде друкувати перший рядок і позицію символу, де вони розрізняються.

**Вправа 7-4**

Переробіть програму пошуку заданої комбінації символів з "лекції №5" таким чином, щоб в якості введення використовувався набір іменованих файлів або, якщо ніякі файли не вказані як аргументи, стандартне введення. Чи слід друкувати ім'я файлу при знаходженні відповідної рядка?

**Вправа 7-5**

Напишіть програму друку набору файлів, яка починає кожен новий файл з нової сторінки і друкує для кожного файлу заголовок і лічильник поточних сторінок.

#### 7.9. Кілька різноманітних функцій

Стандартна бібліотека надає безліч різноманітних функцій, деякі з яких виявляються особливо корисними. Ми вже згадували функції для роботи з рядками: strlen, strcpy, strcat і strcmp. Ось деякі інші.

##### 7.9.1. Перевірка виду символів і перетворення

Деякі макроси виконують перевірку символів і перетворення:

isalpha (c) НЕ 0, якщо "c" алфавітний символ,

0 - якщо ні.

isupper (c) Чи не 0, якщо "c" буква верхнього регістру,

0 - якщо ні.

islower (c) Чи не 0, якщо "c" буква нижнього регістра,

0 - якщо ні.

isdigit (c) Чи не 0, якщо "c" цифра,

0 - якщо ні.

isspace (c) Чи не 0, якщо "c" пробіл, табуляція

або новий рядок, 0 - якщо ні.

toupper (c) Перетворює "c" в велику літеру.

tolower (c) Перетворює "c" в малу літеру.

##### 7.9.2. Функція UNGETC

Стандартна бібліотека містить досить обмежену версію функції ungetch, написаної нами в "лекції №4"; вона називається ungetc. В результаті звернення

ungetc(c,fp)

символ "C" повертається в файл fp. Дозволяється повертати в кожен файл тільки один символ. Функція ungetc може бути використана в будь-який з функцій введення і з макросами типу scanf, getc або getchar.

##### 7.9.3. Звернення до системи

функція system (s) виконує команду, що міститься в символьному рядку s, і потім відновлює виконання поточної програми. Вміст s сильно залежить від операційної системи. Як тривіального прикладу, зазначимо, що на системі UNIX рядок

system("date");

призводить до виконання програми date, яка друкує дату і час дня.

##### 7.9.4. Керування пам’яттю

функція calloc вельми схожа з функцією alloc, використаної нами в попередніх лекціях. В результаті звернення

calloc(n, sizeof(object))

повертається або покажчик простору, достатнього для розміщення n об'єктів зазначеного розміру, або null, якщо запит не може бути задоволений. Відведена пам'ять ініціалізується нульовими значеннями.

Покажчик має потрібне для розглянутих об'єктів вирівнювання, але йому слід приписувати відповідний тип, як в

char \*calloc();

int \*ip;

ip=(int\*) calloc(n,sizeof(int));

функція cfree (p) звільняє простір, на яке вказує "p", причому покажчик "p" спочатку повинен бути отриманий в результаті звернення до calloc. Тут немає ніяких обмежень на порядок звільнення простору, але буде неприємно помилкою звільнити що-небудь, що не було отримано зверненням до calloc.

Реалізація програми розподілу пам'яті, подібної calloc, в якій розміщені блоки можуть звільнятися в довільному порядку, продемонстрована в "лекції №8" .