SUMY STATE UNIVERSITY

Файли

Викладач: Берест О.Б.

План

1. Три рівня введення/виведення в мові С.
2. Обмін даними з файлом.
3. Функції для роботи з файлом.

Обмін даними з файлами

У компілятор мови С не включено спецзасобів для введення/виведення даних, тому обмін даними реалізовано через бібліотечні функції.

Бібліотеки більшості систем програмування мови С підтримують функції для введення/виведення даних на трьох рівнях:

- **високорівневе** (потокоорієнтоване) введення/ виведення використовує однаковий підхід у програмуванні обміну даними з файлами та зовнішніми пристроями і єдиний інтерфейс. Всі файли та дані з пристроїв розглядаються як неструктуровані набори байтів – потоки. Прототипи функцій записані в заголовному файлі **<stdio.h>.**

Обмін даними з файлами

- **функції введення/виведення низького рівня** базуються на засобах обміну даними, що властиві конкретній операційній системі. Дані функції не виконують форматування даних і не застосовують буферизації. Функції блокоорієнтовані і забезпечують виграш у швидкодії тоді, коли обсяг блоку даних, що передається за одну операцію, кратний ємкості сектора диска (512 б). Прототипи даних функцій зберігаються в **<io.h>**.
- **функції консольного введення/виведення** доповнюють можливості високорівневих функцій щодо введення з клавіатури і керування текстовим режимом виведення інформації. Прототипи оголошені в заголовному файлі **<conio.h>**.

Обмін даними 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 введення/виведення здійснюється проміжна буферизація даних. Для кожного відкритого файлу в оперативній пам’яті створюється буфер обміну заданої ємності.

Обмін даними з файлами

Буферизація мінімізує кількість звертань до фізичних пристроїв, які найбільше гальмують процеси введення/виведення даних.

Високорівневий потоковий обмін даними – це обмін байтами з фізичними файлами і логічними пристроями через систему буферизації, що дає змогу опрацьовувати дані різних форматів і розмірів.

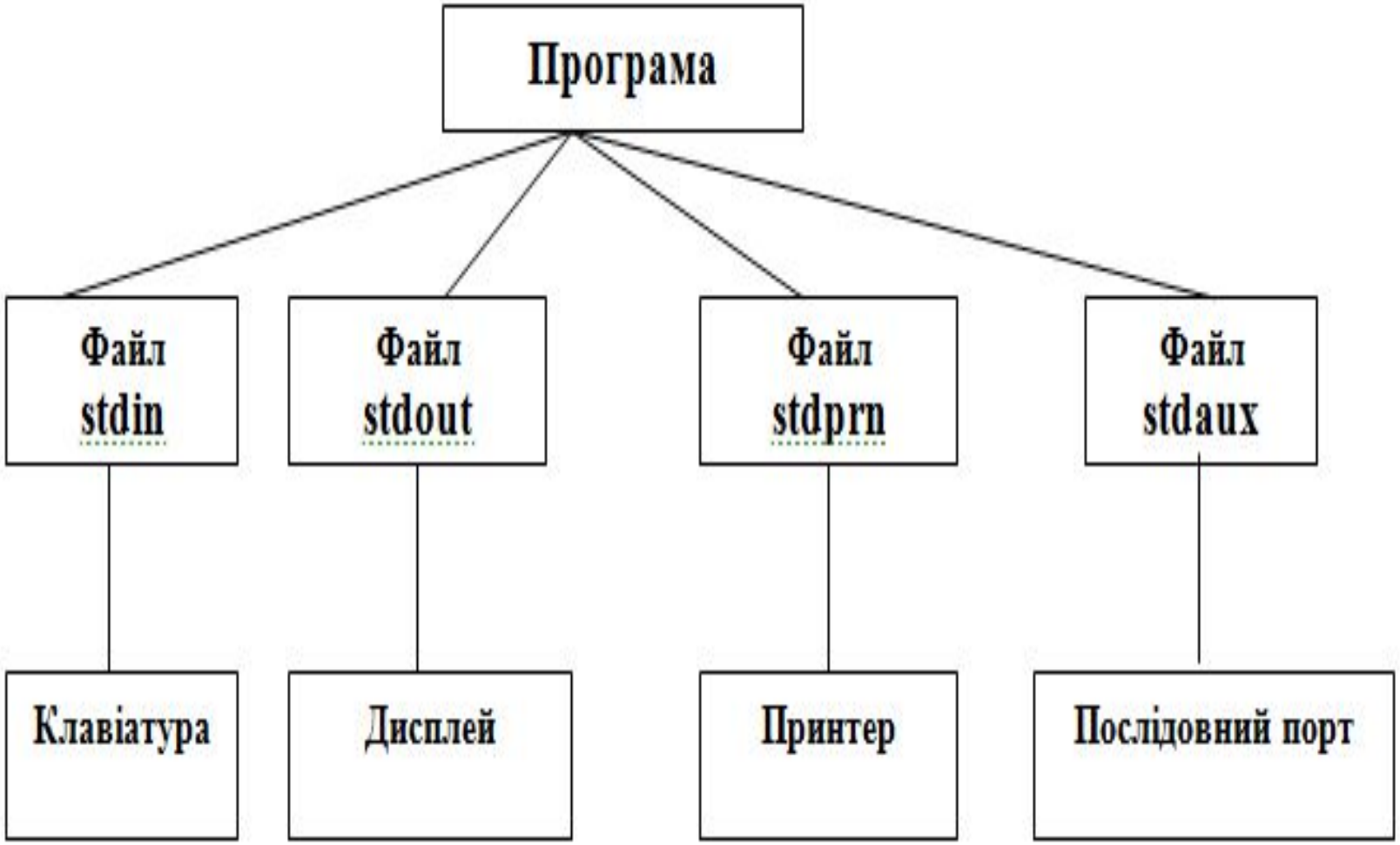
Обмін даними з файлами

Високорівневий потоковий обмін даними – це обмін байтами з фізичними файлами і логічними пристроями через систему буферизації, що дає змогу опрацьовувати дані різних форматів і розмірів.

У мові Сі програми мають зв'язок із зовнішніми пристроями тільки через файли. Для кожного зовнішнього пристрою призначений файл. Ці файли називаються стандартними. Всього їх п'ять:

- **stdin** - файл, пов'язаний за замовчуванням з клавіатурою;
- **stdout** - файл, пов'язаний за умовчанням з дисплеєм;
- **stderr** - файл для помилок. За замовчуванням також пов'язаний з дисплеєм;
- **stdprn** - файл, пов'язаний з принтером;
- **stdaux** - файл, пов'язаний з послідовним (асинхронним) портом.

Обмін даними з файлами



Обмін даними з файлами

У мові Сі файл - це структура. Вона описана у файлі stdio.h в наступному вигляді:

```
typedef struct{
    short level;   // Рівень буфера
    unsigned flags;   // Прапори статусу файлу
    char fd;        // Дескриптор файлу
    char hold;    // Попередній символ, якщо немає буфера
    short bsize;   // Розмір буфера
    unsigned char *bufer; // Буфер передачі даних
    unsigned char *curp; // Поточний активний покажчик
    short token;  // Для перевірки коректності
}FILE;
```

А stdin, stdout, stderr, stdprn, stdaux - це глобальні змінні типу FILE.

Відкриття / закриття потоків

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 *file_name, char * fmode);

Параметри функції є вказівниками на символьні рядки.

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

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

Відкриття / закриття потоків

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

- "r" - (від англ. read). Файл тільки для читання;
- "w" - (від англ. write). Файл тільки для запису;
- "a" - (від англ. append). Файл тільки для доповнення;
- "r+" - файл для читання з можливістю запису в нього;
- "w+" - файл для запису з можливістю читання з нього;
- "a+" - файл для доповнення з можливістю читання.

Додатково в параметрі **fmode** можна задавати текстовий **(t)** чи бінарний **(b)** режим відкриття потоку, за замовчуванням встановлюється текстовий режим. Різниця між двома режимами – в інтерпретації коду клавіші Enter.

Відкриття / закриття потоків

Текстовий файл складається з послідовності символів, розбитих на рядки. Кожен рядок закінчується керуючим символом '\n'. Фактично це два символи: кінець рядка і перехід на новий рядок (LF і CR). Ці два символи, що утворюють керуючий символ '\n', не зчитуються з файлу. Замість цього відбувається виконання команди, тобто перехід на новий рядок. Таким чином, в текстовому режимі цей код клавіші Enter в процесі читання замінюється символом нового рядка ‘\n’, а в разі запису навпаки – кожен символ ‘\n’ заноситься у потік як комбінація “\r\n”.

Відкриття / закриття потоків

Двійковий (бінарний) файл - це послідовність машинних слів. У вигляді машинного слова буде прочитаний і '\n'. При цьому перехід на новий рядок не здійснюється. Відбувається послідовне зчитування машинних слів (кодів) від початку до кінця файлу. Використання двійкових файлів робить програми немобільними при їх перенесенні з одного середовища в інше.

Бінарні файли мають переваги, порівняно з текстовими при зберіганні числових даних. Операції читання і запису з такими файлами виконуються швидше, так як відсутня необхідність форматування. Двійкові файли зазвичай мають менший розмір, ніж аналогічні текстові файли. В двійкових файлах можна переміщуватися в будь-яку позицію і читати або записувати дані в довільній послідовності, в той час, як в текстових файлах практично завжди виконується послідовна обробка інформації.

Відкриття / закриття потоків

Приклад:

```
FILE *p1, *p2;
p1=fopen ("ishodn.dan", "r");
p2=fopen ("rezult.dan", "w");
```

Для закриття потоків використовується функція

int fclose (FILE *fp);

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

Приклад:

```
fclose(f);
```

Стандартні потоки, перескерування потоків

На початку виконання кожної С - програми відкриваються стандартні потоки:

- **stdin** – потік введення, який пов'язується з клавіатурою;
- **stdout** – потік виведення даних на екран;
- **stderr** – потік повідомлень про помилки, скеровуються на екран.

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**.

Стандартні потоки, перескерування потоків

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 **puts()** викликає виведення повідомлення на екран, а друге – у файл.

```
#include <stdio.h>
void main(){
    puts("Виведення на екран");
    freopen ("example.xz", "w" ,stdout);
    puts("Виведення в файл");
}
```

Функції потокового введення/виведення даних

Виділяють чотири групи функцій високорівневого обміну відповідно до формату даних, що передаються за одну операцію:

- посимвольного введення/виведення;
- рядкового введення/виведення;
- блокового введення/виведення;
- форматного введення/виведення.

Всі операції виконуються, починаючи з поточної позиції файлу, яку зберігає спеціальний покажчик. У процесі введення/виведення покажчик автоматично зсувається на відповідну кількість байтів.

Посимвольний обмін даними

Введення одного символу з потоку **fp**

int fgetc (FILE *fp);

Функція повертає код зчитаного символу. Якщо ж читання з потоку недоступне, повертається макро-константа **EOF**. Такі ж дії виконує функція

int getc (FILE *fp);

, яку в Borland C оголошено як макрос.

Функція

int ungetc (int symb, FILE *fp);

дає змогу повернути в потік введення **fp** останній зчитаний символ (після повернення **symb** стає першим символом потоку).

Приклад: зчитування послідовності цифрових символів з потоку **frd** та запис за адресою **numbst.**

```
#include <stdio.h>
#include <ctype.h>
char * ReadNum (char *numbst, FILE * frd){
    char * pn = numbst;
    int symb;
    /*виявлення десяткової цифри */
    while (isdigit(symb = getc(frd)))
    /*запис цифрових символів у numbst*/
        *pn++ =symb;
    *pn ='\0' ;              /*кінець числового рядка */
    ungetc (symb, frd);      /*повернення нечислового символа*/
    return numbst;
}
```

Файловий обмін рядками символів

Зчитування рядка символів з потоку

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

str – масив символів, в який буде записано введений 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 символу нового рядка чи символу кінця файлу, але не більше, ніж **max-1** символів.

У разі успішного виконання функція повертає адресу першого символу введеного рядка (**str**), інакше – **NULL**.

Розмір масиву **str** має бути не меншим за **max**.

Запис заданого рядка в потік виведення

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

Повертає ненульове значення за умови успішного виконання та EOF у разі невдачі. Функція послідовно передає у потік символи рядка **str** до ‘\0’.

Файлове введення даних згідно зі заданим списком форматних специфікацій здійснює функція

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

Функція повертає кількість успішно введених даних. Параметр **fp** вказує на текстовий потік введення, обов’ язковий параметр **format** – задає символьний рядок з послідовністю специфікацій форматних перетворень.

Наступні параметри задають адреси змінних, куди будуть записуватись введені значення (їх кількість і типи визначаються специфікаціями). Правила форматних перетворень даних такі ж, як і для **scanf()**.

Помилки в процесі введення можна конкретизувати через функцію **feof()** – повертає нуль, якщо не встановлено ознаку кінця файлу, інакше повертає ненульове значення.

```
FILE *fin;
int numb;
while(fscanf(fin, “%d”, &numb)==1)
    printf(“%3d”, numb);
if(feof(fin))
    puts(“Зчитано всі числа файлу”);
else
    puts(“Помилка в числових даних”);
```

Для читання символьних даних використовують специфікатор **“%s”**, при цьому зчитується послідовність символів до першого роздільника.

В разі звертання до файлу **fscanf (f, “%s”, str);** у рядок **str** буде зчитане тільки одне поточне слово.

Необхідно забезпечити щоб обсяг ділянки **str** був достатнім для запису найдовшого слова файлу.

Форматне виведення в файл здійснюється функцією

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

Приклад:

```
void vyvod (FILE *f){
    int a,b,c,k;
    scanf (f, ”%d%d%d”, &a, &b, &c);
    fprintf (f, “a=%d, b=%d, c=%d);
}
```

Відкриття текстового файлу **test.txt** може мати вигляд

```
#include<stdio.h>
void main(){
    …
    FILE *f;
    if (( f=fopen("test.txt", "rt"))==NULL){
        printf("Файл не вдалося відкрити.\n");
        return;
    }
    …
    fclose(f);
    …
}
```

Форматоване введення/виведення даних

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

Обмін даними

Зчитати з потоку блок даних заданого розміру можна за допомогою функції

size_t fread (void *buf, size_t size, size_t n, FILE *fp);

що заносить у буфер **n** об’єктів розміром **size**. Тип **size_t** оголошено в **<stdio.h>** через декларацію **typedef**. У Borland C він збігається з типом int.

Функція повертає кількість реально зчитаних об’єктів.

Розглянемо приклад визначення середньоарифметичного значення дійсних чисел, бінарні коди яких зберігаються у файлі. Дані зчитуються в буфер блоками по **N** чисел. Робота завершується, коли розмір зчитаного блоку менший за **N**, що сигналізує про досягнення кінця файлу.

Запис блоку даних у потік виконує функція

size_t fwrite (void *buf, size_t size, size_t n, FILE *fp);

Параметри такі ж, як і в функції **fread()**.

Обмін даними

```
#include <stdio.h>
#define fname "sum.xz"
#define N 100
 void main(){
      int a[N], n, i;
      float s=0, k=0;
      FILE *f;
      f =fopen ( fname, "rb");
      do{
            n=fread (a, sizeof(int), N, f);
            for (i=0; i<n; i++)
                  s+=a[i];
            k+=n;
      }
      while (n==N);
            printf ("sr=%4.2f", s/k);
      fclose(f);
}
```

Обмін даними

Для організації читання даних з файлу в довільному порядку використовується "покажчик файла" (курсор), який визначає поточну позицію у файлі. При читанні даних курсор автоматично зміщується на число прочитаних байтів. Отримати поточну позицію курсору файла можна за допомогою функції

long ftell (FILE *stream);

Встановлюється поточна позиція курсору у файлі за допомогою функції fseek():

int fseek (FILE *stream, long offset, int whence);

Ця функція задає зміщення на число байтів **offset** від точки відліку, яка визначається параметром **whence**.

Константа	whence	Точка відліку
SEEK_SET	0	Початок файлу
SEEK_CUR	1	Поточна позиція
SEEK_END	2	Кінець файлу

Обмін даними

Приклад: необхідно знайти добуток матриці a[n][m] на вектор b[m]. В результаті отримаємо вектор c[n]. Значення елементів матриці та вектора вводимо з файлу ish.dan. Результати передаємо на екран та в файл rez.dan.

```
#include<stdio.h>
#include<stdlib.h>
#define  N 2
#define  M 4
void main() {
    float  a[N][M], b[M], c[N];
    int  i,  j;
    FILE *p1, *p2;
    p1=fopen("ish.dan","r");
    if(p1==0){
        puts("Файл ish.dan не открылся");
```

Обмін даними

```
        exit(1);
    }
    p2=fopen("rez.dan", "w");
    for(i=0; i<N; i++)
        for (j=0; j<M; j++)
            fscanf(p1, "%f",&a[i][j]);
    for (j=0; j<M; j++)
        fscanf(p1, "%f",&b[j]);
    for(i=0; i<N; i++){
        c[i]=0;
        for (j=0; j<M; j++)
            c[i]+=a[i][j]*b[j];
        fprintf(p2, "c[%i]=%f\n", i, c[i]);
        printf("c[%i]=%f\n", i, c[i]);
    }
    fclose(p1); fclose(p2);
}
```

Дякую за увагу

Oleg Berest, 2013