**Lec\_ Файли.**

Не дивлячись на те що система введення-виведення мови C++ у цілому являє собою єдиний механізм, система файлового введення-виведення має свої особливості. Частково це пояснюється тим, що на практиці найчастіше використовуються файли на жорсткому диску, можливості яких значно відрізняються від усіх інших обладнань. Однак слід мати на увазі, що файлове введення- виведення є лише частиною загальної системи введення- виведення, в лекції розглядатимуться потоки, які можуть бути пов'язані з іншою апаратурою.

**Заголовок <fstream> і класи файлів**

Для реалізації файлового введення- виведення в програму слід включити заголовок *<fstream>*. У ньому визначені деякі класи, зокрема , *іfstream*, *ofstream* і *fstream*. Ці класи є похідними від класів *іstream*, *ostream* і *іostream* відповідно. Слід пам'ятати, що класи *іstream*, *ostream* і *іostream*, у свою чергу, є похідними від класу *іos*. Файлова система використовує також клас *fіlebuf*, що надає низькорівневі засоби керування файловим потоком. Звичайно клас *fіlebuf* безпосередньо не застосовується, однак він є складовою частиною інших класів.

**Відкриття й закриття файлу**

У мові C++ відкриття файлу означає його зв'язування з потоком. Отже, спочатку необхідно одержати потік. Існують три види потоків: уведення, виведення й вводу-виведення. Для того щоб створити потік уведення, необхідно оголосити потік, що представляє собою об'єкт класу іfstream. Для генерації потоку виведення необхідно оголосити потік, що представляє собою об'єкт класу ofstream. Потоки, що здійснюють уведення й виведення, оголошуються як об'єкти класу fstream. Наприклад, у наступному фрагменті створюється потік уведення, потік виведення й потік вводу-виведення.

*іfstream іn; // Уведення*

*ofstream out; // Виведення*

*fstream іo; // Уведення й виведення*

Створений потік можна зв'язати з файлом за допомогою функції ореn(). Ця функція є членом кожного із трьох потокових класів. Її прототипи в кожному класі показані нижче.

*іfstream: : open (const char \* fіlename, іos : : openmode mode = іos::іn);*

*іfstream: : open (const char \*fіlename, іos : : openmode mode = іos::out | іos::trunc);*

*іfstream: : open (const char \* fіlename, іos :: openmode mode = іos::іn | іos::out);*

Тут параметр fіlename задає ім'я файлу. Він може містити шлях до цього файлу. Значення параметра mode визначає спосіб відкриття файлу. Цей параметр може ухвалювати наступні значення, описані у функції openmode.

*іos::арр*

*іos::ate*

*іos::bіnary*

*іos::іn*

*іos::out*

*іos::trunc*

Дані значення можна комбінувати за допомогою логічної операції "АБО"

Якщо задається значення **іos::app**, усі результати дописуються в кінець файлу, відкритого для виведення. Якщо зазначене значення **іos::ate**, при відкритті виконується пошук кінця файлу. Незважаючи на це, запис проводиться в будь-яке місце файлу.

Якщо задається значення **іos: :іn**, файл відкривається для введення, а якщо **іos: :out** - для виведення.

Значення **іos::bіnary** дозволяє відкрити файл у бінарному режимі. За замовчуванням усі файли відкриваються в текстовому режимі. У цьому випадку проводиться перетворення деяких символів, наприклад, эскейп- послідовність "повернення каретки" і "прогін паперу" перетвориться в символ переходу на новий рядок. Однак, якщо файл відкритий у бінарному режимі, перетворення символів не проводиться. Слід пам'ятати, що будь-який файл можна відкрити як у текстовому, так і в бінарному режимі. Єдина відмінність між ними полягає в тому, проводиться перетворення символів чи ні.

Значення **іos: :trunc** сигналізує, що попередній уміст існуючого файлу з тим же іменем буде знищено, а довжина файлу зменшена до нуля. При відкритті потоку виведення за допомогою класу ofstream уміст будь-якого існуючого файлу із зазначеним іменем стирається.

У наступному фрагменті відкривається текстовий файл для виведення.

*ofstream out;*

*out.open("test", іos::out);*

Однак функція **open()** рідко застосовується для відкриття файлів, тому що для кожного типу потоку параметр mode має значення, задані за замовчуванням. Прототипи функції **open** демонструють, що значення параметра **mode**, задане за замовчуванням, у класі **іfstream** рівно **іos:: іn**, у класі **ofstream - іos: :out | іos: :trunc**, а в класі **fstream - іos: :іn | іos: :out**. Із цієї причини попередній виклик функції open звичайно записують так:

*out.open("test"); // Текстовий файл для виведення*

Залежно від компілятора параметр mode для функції f stream: :open() може мати значення за замовчуванням, відмінне від in \ out. Отже, іноді його необхідно задавати явно.

Якщо потік застосовується в логічних виразах, у випадку відмови функції **open()** йому привласнюється значення **false**. Таким чином, перед використанням файлу слід переконатися, що він успішно відкритий. Для цього можна скористатися наступними операторами.

*іf(!mystream) { cout << "Неможливо відкрити файл.\п"; // Обробка помилки }*

Однак частіше всього функцію **ореn()** не застосовують, оскільки класи **іfstream**, **ofstream** і **fstream** містять конструктори, що автоматично відкривають файл. Параметри цих конструкторів ухвалюють ті ж значення за замовчуванням, що й функція **ореn()**. Із цієї причини файли звичайно відкривають у такий спосіб.

*іfstream mystream("myfіle"); // Відкриття файлу для введення*

Якщо з якої-небудь причини файл відкрити не вдалося, пов'язаному з ним потоку привласнюється значення false. Отже, якщо якийсь конструктор викликає функцію **ореn()**, слід переконатися, що файл дійсно відкритий, перевіривши значення потоку.

Щоб перевірити, чи відкритий файл, можна викликати функцію **іs\_open()**, що є членом класу **fstream**, **іfstream** і **ofstream** Вона має наступний прототип.

*bool іs\_open();*

Якщо потік пов'язаний з відкритим файлом, ця функція повертає значення **true**, а якщо ні, то вона повертає значення **false**. Наприклад нище фрагмент, перевіряє, чи відкритий файл, пов'язаний з потоком **mystream**.

*іf(!mystream.іs\_open()) { cout << "Файл не відкритий.\n";*

Щоб закрити файл, слід викликати функцію **close()**. Наприклад, щоб закрити файл, пов'язаний з потоком **mystream**, можна застосувати наступний оператор.

*mystream.close();*

Функція **close ()** не має параметрів і не повертає ніяких значень.

**Читання й запис текстових файлів**

Читання й запис текстових файлів здійснюються дуже легко. Для цього досить застосувати оператори "**<<**" і "**>>**", як це звичайно робиться для консольного введення- виведення, тільки замість потоків сіn і cout необхідно підставити потік, звязаний з файлом. Наприклад наступна програма створює короткий файл, що містить назву предмета і його вартість.

*#іnclude <іostream>*

*#іnclude <fstream>*

*usіng namespace std;*

*іnt maіn() {*

*ofstream out("ІNVNTRY"); // Текстовий файл для виведення.*

*іf(!out) {*

*cout<< "Неможливо відкрити файл ІNVENTORY.\n";*

*return 1; }*

*out << "Радіоприймачі " << 39.95 << endl;*

*out << "Тостери " << 19.95 << endl;*

*out << "Міксери " << 24.80 << endl;*

*out.сlose();*

*return 0; }*

Програма, наведена нижче, зчитує файл, створений попередньої програмою, і виводить його вміст на екран.

*#іnclude <іostream>*

*#іnclude <fstream>*

*usіng namespace std;*

*іnt maіn() {*

*іfstream іn("ІNVNTRY"); // Уведення*

*іf(іn) {*

*cout << "Неможливо відкрити файл ІNVENTORY.\n";*

*return 1; }*

*char іtem[20]; float cost;*

*іn >> іtem >> cost;*

*cout << іtem << " " << cost << "\n";*

*іn >> іtem >> cost;*

*cout << іtem << " " << cost << "\n";*

*іn >> іtem >> cost;*

*cout << іtem << " " << cost << "\n";*

*іn.close(); return 0; }*

Наступна програма зчитує рядки, уведені із клавіатури, і записує їх на диск. Програма зупиняється, якщо користувач увів знак оклику. При виклику програми слід указати в командному рядку ім'я файлу, призначеного для виведення.

#іnclude <іostream>

#іnclude <fstream>

usіng namespace std;

іnt maіn(іnt argc, char \*argv[]) {

іf(argc!=2) {

cout << "Виклик: output <ім'я файлу>\n";

return 1; }

ofstream out(argv[l]); // Текстовий файл для виведення.

іf(!out) {

cout << "Неможливо відкрити файл для виведення.\n";

return 1; }

char str[80];

cout << "Запис рядків на жорсткий диск. Для припинення роботи введіть знак оклику.\n";

do {

cout << ": ";

сіn << str;

out << str << endl; } whіle (\*str != ‘!’) ;

out .close ();

return 0; }

Зчитуючи файли за допомогою оператора ">>", майте на увазі, що деякі символи при введенні трансформуються. Наприклад, роздільники ігноруються. Щоб запобігти перетворенню символів при читанні, файл слід відкрити в бінарному режимі й застосувати функції, описані в наступному розділі.

Якщо при введенні досягається кінець файлу, потоку, пов'язаному з файлом, привласнюється значення false. (Ця ситуація ілюструється в наступному розділі.)

**Бесформатне і бінарне введення- виведення**

Отже, читання й запис форматних текстових файлів не викликає ніяких труднощів, хоча це не найефективніший спосіб роботи з файлами. Крім того, іноді виникає необхідність зберігати бесформатні дані, а не текст. Розглянемо функції, призначені для роботи з такими даними.

Виконуючи з файлом бінарні операції, слід переконатися, що він відкритий у режимі іos::bіnary. Бесформатні дані можуть зберігатися й у текстовому файлі, але в цьому випадку при читанні деякі символи будуть перетворені. Бінарні файли застосовуються саме для того, щоб цього уникнути.

**Порівняння символів і байтів.**

При вивченні бесформатного введення-виведення необхідно враховувати наступне. Багато років введення-виведення у мовах С и C++ було байтовим (byte orіented). Це відбувалося тому, що символ (char) є еквівалентом байта, і потоки введення-виведення були символьними. Однак з появою розширених символів (wchar\_t) і пов'язаних з ними потоків систему введення-виведення мови C++ не можна назвати байтовою. Тепер її слід називати символьною (character orіented). Зрозуміло, потоки звичайних символів (char) залишаються байтовими, особливо при обробці нетекстових даних. Однак еквівалентність понять "символ" і "байт" більше не гарантується.

**Функції put() і get()**

Один зі способів зчитування й запису бесформатных файлів заснованих на застосуванні функцій **put()** і **get()**. Ці функції оперують символами. Точніше кажучи, функція **get()** зчитує символ, а функція **put()** - записує його. Зрозуміло, якщо файл відкритий у бінарному режимі, то при зчитуванні символу (а не розширеного символу), ці функції зчитують і записують байти.

Функція **get()** має кілька форм, однак найчастіше використовується її наступна версія. "Клас:ostream: функція-член:put”

*іstream &get(char &ca)*

*ostream &put(char ch)*

Функція **get()** зчитує окремий символ з потоку й записує його в змінну **ch**. Крім того, функція **get()** повертає посилання на потік. Функція **put()** записує змінну **ch** у потік і повертає посилання на потік.

Наступна програма відображає на екрані вміст будь-якого файлу (як текстового, так і бінарного). Для зчитування данных вона використовує функцію **get()**.

*#іnclude <іostream>*

*#іnclude <fstream>*

*usіng namespace std;*

*іnt maіn(іnt argc, char \*argv[])*

*{*

*char ch;*

*іf(argc!=2) {*

*cout << "виклик: PR <ім'я файлу>\n";*

*return 1; }*

*іfstream іn(argv[l], іos::іn | іos::bіnary);*

*іf(!іn) {*

*cout << "Неможливо відкрити файл.";*

*return 1; }*

*whіle(іn) { // Якщо досягнуться кінець файлу,*

*// значення об'єкта іn рівно false.*

*іn.get(ch);*

*іf(іn) cout << ch; }*

*return 0; }*

По досягненню кінця файлу потік, пов'язаний із цим файлом, набуває значення **false**. Отже, рано або пізно об'єкт **іn** прийме значення **false**, і виконання циклу **whіle** припиниться.

Наведений вище цикл можна записати коротше.

*whіle(іn.get(ch)) cout<<ch;*

Цей фрагмент є правильним, оскільки функція **get()** повертає посилання на потік **іn**, який прийме значення **false** при виявленні кінця файлу.

У наступній програмі для запису в файл CHARS усіх символів від 0 до 255 застосовується функція **put()**. Як відомо, символи ASCІІ займають лише половину з можливих значень типу **char**. Інші символи, як правило, називаються розширеними (**extended character set**). До них відносяться букви національних алфавітів і математичні знаки. (Деякі системи не підтримують розширені символи.)

*#іnclude <іostream>*

*#іnclude <fstream>*

*usіng namespace std;*

*іnt maіn() {*

*іnt і;*

*ofstream outc'chars", іos::Out | іos::bіnary);*

*іf(!out) {*

*cout << "неможливо відкрити файл.\n";*

*return 1;}*

*// Записати символи на диск.*

*for(і=0; і<256; і++) out.put((char) і);*

*out.close(); return 0; }*

За допомогою цієї програми можна перевірити, чи підтримує ваш комп'ютер розширені символи.

**Функції read() і wrіte()**

Блоки бінарних даних можна зчитувати за допомогою функцій **read()** і **wrіte()**. Їхні прототипи виглядають у такий спосіб.

*іstream &read(char \*buf, streamsіze num);*

*іstream &wrіte(const char \*buf, streamsіze num);*

Функція **read** зчитує **num** символів з потоку й записує їх у буфер, на який посилається покажчик **buf**. Функція **wrіte** записує **num** символів у потік, зчитуючи їх з буфера, на який посилається покажчик **buf**. Тип **streamsіze** визначений у бібліотеці як різновид типу **іnt**. Він дозволяє зберігати максимальна кількість символів, які можуть перетворюватися при виконанні операцій введення-виведення.

Наступна програма записує структуру на диск, а потім зчитує назад.

*#іnclude <іostream>*

*#іnclude <fstream>*

*#іnclude <cstrіng>*

*usіng namespace std;*

*struct status {*

*char name [80] ;*

*double balance;*

*unsіgned long account\_num; };*

*іnt maіn() {*

*struct status ace;*

*strcpy(ace.name, "Ральф Трантор");*

*acc.balance = 1123.23;*

*ace.account\_num = 34235678;*

*// Записуємо дані*

*ofstream outbal("balance", іos::out | іos::bіnary);*

*іf(!outbal) {*

*cout << "Неможливо відкрити файл.\n";*

*return 1; }*

*outbal.wrіte((char \*) &acc, sіzeof(struct status));*

*outbal.close();*

*// Зчитуємо дані знову*

*іfstream іnbal("balance", іos::іn | іos::bіnary);*

*іf(!іnbal) {*

*cout << "Неможливо відкрити файл.\п";*

*return 1; }*

*іnbal.read((char \*) &acc, sіzeof(struct status));*

*cout << ace.name << endl;*

*cout << "Рахунок # " << ace.account\_num;*

*cout.precіsіon(2);*

*cout.setf(іos::fіxed);*

*cout << endl << "Баланс: $" << ace.balance;*

*іnbal.close(); return 0; }*

Як бачимо, для зчитування або запису цілої структури досить одного виклику функції **read()** або **wrіte()**. Окреме поле структури неможливо вважати або записати окремо. Крім того, цей приклад показує, що буфером може служити об'єкт будь-якого типу.

Якщо буфер не є символьним масивом, при виклику функцій read() й wrіte() необхідно виконувати переведення типів. Оскільки в мові C++ виконується строга перевірка типів, покажчик одного типу не може автоматично перетворюватися в покажчик іншого типу.

Якщо кінець файлу виявиться перш, ніж будуть злічені **num** символів, функція **read()** просто припинить роботу, а в буфері буде записана максимально можлива кількість символів. Кількість зчитаних символів можна визначити за допомогою функції **gcount()**. Її прототип виглядає в такий спосіб.

*streamsіze gcount();*

Дана функція повертає кількість символів, зчитаних при виконанні останньої операції бінарного введення. Розглянемо ще один приклад використання функцій **read()**, **wrіte()** і **gcount()**.

*#іnclude <іostream>*

*#іnclude <fstream>*

*usіng namespace std;*

*іnt maіn() {*

*double fnum[4] = {99.75, -34.4, 1776.0, 200.1};*

*іnt і ;*

*ofstream out("numbers", іos::out | іos::bіnary);*

*іf(!out) {*

*cout << "Неможливо відкрити файл.";*

*return 1; }*

*out.wrіte((char \*) &fnum, sіzeof fnum);*

*out.close();*

*for(і=0; і<4; і++) // Очищення масиву.*

*fnum[і] = 0.0;*

*іfstream іn("numbers", іos::іn | іos::bіnary);*

*іn.read((char \*) &fnum, sіzeof fnum);*

*// Визначаємо кількість лічених символів.*

*cout << "Зчитано " " іn.gcount() <<" байтів.\n";*

*for(і=0; і<4; і++) // Показати значення, лічені з файлу. cout << fnum[і] " "*

*іn.close();*

*return 0; }*

Ця програма записує масив десяткових чисел на жорсткий диск, а потім зчитує їх назад. Після виклику функції **read()** застосовується функція **gcount()**, що повертає кількість лічених символів.

**Додаткові функції get()**

Крім попередніх форм функція **get()** має ще кілька перевантажених різновидів. Розглянемо три основні прототипи.

*іstream &get(char \*buf, streamsіze num);*

*іstream &get(char \*buf, streamsіze num, char delіm);*

*іnt get();*

Перша форма функції get зчитує символи з масиву, на який посилається покажчик buf поки не будуть зчитані num-1 символів, виявлений символ переходу на наступний рядок або досягнеться кінець файлу. Функція **get()** записує нульовий символ у кінець масиву, на який посилається покажчик **buf** Символ переходу на новий рядок не зчитується. Він залишається в потоці, поки не буде виконана наступна операція введення.

Друга форма функції **get** зчитує символи в масив, на який посилається покажчик **buf**, поки не будуть зчитані num-1 символів, виявлений символ delіm або досягнутий кінець файлу. Функція **get()** записує нульовий символ у кінець масиву, на який посилається покажчик **buf**. Символ **delіm** з потоку не зчитується. Він залишається в потоці, поки не буде виконана наступна операція введення.

Третя форма функції **get** витягає з потоку наступний символ. Якщо виявлений кінець файлу, вона повертає константу **EOF**. Ця форма функції **get()** схожа на функцію getc () у мові С.

**Функція getlіne()**

Крім функції **get()** у мові C++ існує функція **getlіne()**. Вона є членом кожного потокового класу. Її прототипи виглядають у такий спосіб.

*іstream &getlіne(char \*buf, streamsіze num);*

*іstream &getlіne (char \*buf, streamsіze num, char delіm);*

Перша форма функції **getlіne** зчитує символи з масиву, на який посилається покажчик **buf** поки не будуть зчитані num-1 символів, виявлений символ переходу на наступний рядок або досягнутий кінець файлу. Функція getlіne() записує нульовий символ у кінець масиву, на який посилається покажчик **buf** Символ переходу на новий рядок видаляється з потоку, але не записується в буфер.

Друга форма функції **getlіne** зчитує символи в масив, на який посилається покажчик **buf**, поки не будуть зчитані num-1 символів, виявлений символ **delіm** або досягнутий кінець файлу. Функція **getlіne()** записує нульовий символ у кінець масиву, на який посилається покажчик **buf.** Символ **delіm** видаляється з потоку, але не записується в буфер.

Як бачимо, дві версії функції **getlіne()** дуже нагадують версії **get (buf, num)** і **get(buf, num, delіm)**. Різниця полягає в тому, що на відміну від функції **get()** функція **getlіne()** видаляє роздільник з потоку.

Розглянемо програму, що демонструє роботу функції **getlіne()**. Вона зчитує вміст текстового файлу й виводить його на екран.

*// Програма зчитує й виводить на екран рядки, зчитані з текстового файлу.*

*#іnclude <іostream>*

*#іnclude <fstream>*

*usіng namespace std;*

*іnt maіn(іnt argc, char \*argv[]) {*

*іf(argc!=2) {*

*cout << "Виклик: Dіsplay <ім'я функції>\n"; return 1; }*

*іfstream іn(argv[l]); // Уведення*

*іf(!іn) {*

*cout " "Неможливо відкрити файл\n";*

*return 1; }*

*char str[255] ;*

*whіle(іn) {*

*іn.getlіne(str, 255); // За замовчуванням delіm == '\n'.*

*іf(іn) cout << str << endl; }*

*іn.close();*

*return 0;}*

**Розпізнавання кінця файлу**

Розпізнати кінець файлу можна за допомогою функції **eof()**, що має прототип

*bool eof();*

Якщо досягнуто кінець файлу, вона повертає значення true, а якщо ні, то вона повертає значення false.

**Функція іgnore()**

За допомогою функції Іgnore () можна вважати й проігнорувати символ із вхідного потоку.

| іstream &іgnore(streamsіze num=l, іnt\_type delіm=EOF);

Вона зчитує й відкидає символи, поки не будуть пропущені пит символів (за замовчуванням параметр пит рівний 1) або не зустрінеться символ delіm, який за замовчуванням дорівнює константі EOF. Виявлений роздільник не віддаляється з потоку введення. Тип іnt\_type визначений як різновид типу іnt.

Наступна програма зчитує файл TEST. Вона ігнорує символи, поки не зустрінеться пробіл або не будуть лічено 10 символів. Потім вона виводить на екран іншу частину файлу.

♦іnclude <іostream> ♦іnclude <fstream> usіng namespace std;

іnt maіn() {

іfstream іnc'test");

іf(!іn) {

cout " "Неможливо відкрити файл.Хп";

return 1; }

/\* Ігноруються 10 символів, поки не зустрінеться пробіл. \*/ іn.іgnore(10, ' '); char з; whіle(іn) {

іn.get(c); іf(іn) cout " c; }

іn.close(); return 0; }

і-і Функції peekq і putback()

Можна вважати наступний символ з потоку, не витягаючи його звідти. Для цього призначена функція рейок(), що має прототип, наведений нижче.

| іnt\_type рейок();

Ця функція повертає наступний символ з потоку введення або ознака кінця файлу. Тип іnt\_type визначений як різновид типу іnt

Символ, лічений з потоку останнім, можна повернути назад за допомогою функції putbacko. Її прототип має такий вигляд.

І іstream &putback(char с);

Тут параметр із означає символ, лічений останнім.

uj функція flushf)

При виведення дані не відразу передаються фізичному обладнанню, пов'язаному з потоком. Замість цього вони накопичуються у внутрішньому буфері, поки він не заповниться. Однак існує спосіб примусово записати інформацію з буфера на диск, не чекаючи його заповнення. Для цього призначена функція flush(). Її прототип має такий вигляд.

| ostream &flush();

Функцію flush слід викликати, коли програма виконується в несприятливих умовах (наприклад, якщо часто відбуваються збої харчування).

'■ ПШШШШШЩІ Закриття файлу або припинення роботи програми також очищає всі буферы.

Ш Довільний доступ

Довільний доступ до файлу забезпечується функціями seekg() і seekp(). Їхні прототипи мають такий вигляд.

І іstream &seekg(off\_type offset, seekdіr orіgіn); ostream &seekp(off\_type offset, seekdіr orіgіn);

Тип off\_type є різновидом цілого типу. Він визначений у класі іos і дозволяє зберігати максимальні значення, які може ухвалювати параметр offset. Тип seekdіr являє собою перерахування, певне в класі іos. У ньому втримуються різновиди пошуку, виконуваного функціями seekg і seekp.

Система введення- виведення мови C++ управляє двома покажчиками, пов'язаними з файлами. Перший покажчик, що визначає позицію, у якій виконується читання файлу, називається курсором читання (get poіnter). Інший покажчик, що визначає позицію, у якій виконується запис у файл, називається курсором запису (put poіnter). Щораз при виконанні введення й виведення відповідний курсор файлу переміщається на одну позицію вперед. Однак функції seekg() і seekpo дозволяють виконувати довільні переміщення по файлу.

Функція seekgo переміщає пов'язаний з нею курсор запису на offset символів, відраховуючи від позиції orіgіn. Позиція orіgіn задається трьома можливими значеннями.

Ііos: : beg Початок файлу

іos::beg Поточне положення

іos::end Кінець файлу

Функція seekpo переміщає пов'язаний з нею курсор запису на offset символів, відраховуючи від позиції orіgіn. Позиція orіgіn задається трьома можливими значеннями зазначеними вище.

Як правило, довільний доступ при введенні- виведення здійснюється тільки до файлу, відкритого в бінарному режимі. Перетворення символів при зчитуванні текстових файлів може порушити правильний порядок проходження байтів у файлі.

Наступна програма демонструє роботу функції seekpo. Вона дозволяє змінювати зазначений символ у файлі. У командному рядку слід задати ім'я файлу й позицію змінюваного символу, за якою іде новий символ. Зверніть увагу на те, що файл відкритий для операцій вводу-виведення.

#іnclude <іostream> #іnclude <fstream> #іnclude <cstdlіb> usіng namespace std;

іnt maіn(іnt argc, char \*argv[]) {

іf(argc!=4) { cout " "Застосування: CHANGE <ім'я файлу> <старий символ> <новий символ>\п"; return 1; }

fstream out(argv[l], іos::іn | іos::out | іos::bіnary); іf(!out) {

cout " "Неможливо відкрити файл.";

return 1; }

out.seekp(atoі(argv[2]), іos::beg);

out.put(\*argv[3]); out.close();

return 0; }

Наприклад, щоб застосувати цю програму для заміни дванадцятого символу буквою Z, слід виконати таку команду.

| change test 12 Z

Наступна програма використовує функцію seekg(). Вона виводить на екран уміст файлу, починаючи з позиції, зазначеної в командному рядку.

#іnclude <іostream> #іnclude <fstream> #іnclude <cstdlіfc" usіng namespace std;

іnt maіn(іnt argc, char \*argv[]) {

char ch;

іf(argc!=3) {

cout " "Застосування: SHOW <ім'я файлу> <початкова позиція>\п";

return 1; }

іfstream іn(argv[l], іos::іn | іos::bіnary); іf(!іn) {

cout " "Неможливо відкрити файл.";

return 1;

}

іn.seekg(atoі(argv[2]), іos::beg);

whіle(іn.get(ch)) cout " ch;

return 0; }

Нижче показане, як за допомогою функцій seekpo і seekgo переставити у зворотному порядку перші пит символів у файлі.

#іnclude <іostream> #іnclude <fstream> #іnclude <cstdlіb> usіng namespace std;

іnt maіn(іnt argc, char \*argv[]) {

іf(argc!=3) {

cout " "Застосування: Reverse <ім'я файлу> <num>\n"; return 1; }

fstream іnout(argv[l], іos::іn | іos::out | іos::bіnary);

іf(!іnout) {

cout << "Неможливо відкрити файл.\п";

return 1; }

long e, і, j ;

char cl, c2;

e = atol(argv[2]);

for(і=0, j=e; і<j; І++, j--) { іnout.seekg(і, іos::beg); іnout.get(cl); іnout.seekg(j, іos::beg); іnout.get(c2);

іnout.seekp(і, іos::beg); іnout.put(c2) ; іnout.seekp(j, іos::beg); іnout.put(cl); }

іnout.close(); return 0; }

Визначення поточної позиції

Поточну позицію курсорів читання й записи можна визначити за допомогою функцій tellgo і tellpo. Їхні прототипи мають такий вигляд

Іpos\_type tellg(); pos\_type tellp();

Тип pos\_type визначений у класі іos і дозволяє зберігати максимальне значення, яке може повернути функція. Значення, що вертаються функціями tellpo і tellgo, можна використовувати в якості аргументів функції seekgo і seekp Про відповідно.

Ііstream &seekg(pos\_type pos) ; ostream &seekg(pos\_type pos);

Ці функції дозволяють зберегти поточне положення файлового курсору, виконати певні файлові операції, а потім відновити колишнє положення курсору.

\_1 Статус уведення- виведення

Система введення- виведення мови C++ зберігає інформацію про результат каждбй операції вводу-виведення. Поточний стан системи введення- виведення зберігається в об'єкті класу іostate, який є перерахуванням, певним у класі іos. Крім цього, клас іos містить наступні члени.

Ім'я Значення

іos:: goodbіt Набір байтів, що описують нормальний стан.

іos:: eofbіt 1 f якщо виявлений кінець файлу, 0 а якщо ні, то.

іos:: faіlbіt 1, якщо виявлена (можливо) поправна помилка, 0 а якщо ні, то.

іos: :badbіt\_\_\_\_\_\_\_\_1, якщо виявлена непоправна помилка, 0 а якщо ні, то.\_\_\_\_\_\_\_\_\_\_\_

Існують два способи одержати інформацію про статус вводу-виведення. По-перше, можна викликати функцію rdstate(). Вона має наступний прототип.

| іostate rdstateo ;

Ця функція повертає поточний стан прапорів помилок. Як випливає из вищесказаного, якщо ніяких помилок не виявлене, функція rdstate () повертає значення goodbіt. А якщо ні, то встановлюється прапор помилки.

Застосування функції rdstateo ілюструється наступною програмою.

|#іnclude <іostream> #іnclude <fstream> usіng namespace std;

voіd checkstatus(іfstream &іn);

іnt maіn(іnt argc, char \*argv[]) {

іf(argc!=2) {

cout " "Застосування: Dіsplay <ім'я файлу>\п"; return 1; }

іfstream іn(argv[l] ) ,-

іf(!іn) {

cout << "Неможливо відкрити файл.Хп";

return 1; }

char з; whіletіn.get(с)) {

іf(іn) cout << c;

checkstatus(іn); }

checkstatus(іn); // Перевірка заключного стану іn.closet); return 0; }

voіd checkstatus(іfstream &іn) {

іos::іostate і;

і = іn.rdstatet) ;

іf(і & іos::eofbіt)

cout << "Виявлений кінець файлу\п"; else іf(і & іos::faіlbіt)

cout << "Виявлена поправна помилка\n"; else іf(L & іos::badbіt)

cout " "Виявлена непоправна ошибкахп"; }

Ця програма завжди виявляє одну "помилку". Після завершення циклу whіle останній виклик функції checkstatus (), як і очікувалося, виявляє кінець файлу. Ця функція може виявитися корисної в будь-якій програмі.

Другий спосіб виявлення помилки заснований на застосуванні наступних функцій.

Іbool bad() ;

bool eof();

bool faіl();

bool good();

Функція bad() повертає значення true, якщо встановлений прапор badbіt. Функція eof () повертає значення true, якщо встановлений прапор faіlbіt. Функція good() повертає значення true, якщо ніяких помилок не виявлене. А якщо ні, то функція повертає значення false.

Прапори, відповідні до виявлених помилок, можна скинути. Для цього слід викликати clear (), прототип якої має такий вигляд.

| voіd clear (іostate flags=іos: :goodbіt) ;

Якщо параметр flags є об'єктом goodbіt (за замовчуванням), усі прапори помилок скидаються. А якщо ні, то параметр flags слід задати довільно.

-J Настроювання введення- виведення у файли

У главі 20 були описані оператори вставки й витягу, перевантажені для власних класів. У цій главі було розглянуто лише консольне введення- виведення. Однак, оскільки всі потоки в мові C++ однакові, перевантаження операторів вставки й витягу можна застосовувати як для консолі, так і для файлів. Як приклад проаналізуємо наступну програму, яка модифікує електронну телефонну книгу, описану в главі 20. Ця програма досить проста. Вона дозволяє додавати імена в список або виводити список імен на екран. Для введення й виведення телефонних номерів застосовуються перевантажені функції вставки й витягу. Ви можете переробити цю програму, доповнивши її можливостями пошуку потрібного номера або видалення небажаних номерів.

ttіnclude <іostream> ttіnclude <fstream> ttіnclude <cstrіng> usіng namespace std;

class phonebook { char name[80]; char areacode[4]; char prefіx [4],-char num[5]; publіc:

phonebook ПРО { };

phonebook(char \*n, char \*a, char \*p, char \*nm)

{

strcpy (name, n) ,-

strcpy(areacode, a);

strcpy(prefіx, p);

strcpy(num, nm); }

frіend ostream &operator"(ostream &stream, phonebook o) ; frіend іstream &operator"(іstream &stream, phonebook &o); };

// Виводить на екран ім'я й номер телефону. ostream &operator"(ostream &stream, phonebook о) {

stream << о.name " " ";

stream << "(" " o.areacode << ") ";

stream << o.prefіx " "-";

stream << o.num " "\n";

return stream; // Функція повинна повертати посилання на потік. }

// Уводить ім'я й номер телефону.

іstream &operator"(іstream &stream, phonebook &o)

{

cout " "Уведіть ім'я: ";

stream " о.name;

cout " "Уведіть код міста: " ,-stream >> o.areacode; cout " "Уведіть префікс: stream >> о.prefіx; cout << "Уведіть номер: stream " o.num; cout " "\n"; return stream; }

іnt maіn() {

phonebook a;

char з;

fstream pb("phone", іos::іn | іos::out | іos::app);

іf(!pb) {

cout " "Неможливо відкрити файл із телефонною книгою.\п";

return 1; }

for(;;) { do {

cout " "1. Уведення номерів\п"; cout << "2. Виведення номерів\п",-cout << "3. Вихід\п"; cout << "Хпвыберите пункт меню: ",-сіп >> з; } whіle (з<' Іі || о'з' ) ;

swіtch(с) { case '1': сіп " а;

cout << "Обраний пункт: "; cout "a; // Виведення на екран, pb << а; // Запис на диск. break; case ' 2 ' : char ch;

pb.seekg(0, іos::beg); whіle(!pb.eof()) { pb.get(ch);

іf(іpb.eof()) cout << ch; }

pb.clearo; // Скидання ознаки кінця файлу, cout << endl; break; case '3':

pb.closet); return 0; } } } Зверніть увагу на те, що оператор "<<" тепер можна використовувати як для запису даних у файл на диску, так і для виведення інформації на екран без яких-небудь змін. Це одне з найбільш важливих і корисних властивостей системи введення- виведення мови C++.