

4.2. Работа с текстовыми файлами

4.2.1. Потоки в языке C++

Стандартная библиотека обеспечивает гибкий и эффективный метод обработки целочисленного, вещественного, а также символьного ввода через консоль, файлы или другие потоки. А также позволяет гибко расширять способы ввода для типов, определенных пользователем.

Существуют следующие базовые классы:

istream поток ввода (cin)

ostream поток вывода (cout)

iostream поток ввода/вывода

Все остальные классы, о которых пойдет речь далее, от них наследуются.

Классы, которые работают с файловыми потоками:

ifstream для чтения (наследник istream)

ofstream для записи (наследник ostream)

fstream для чтения и записи (наследник iostream)

4.2.2. Чтение из потока построчно

Чтение из потока производится с помощью оператора ввода (`>>`) или функции `getline`, которая позволяет читать данные из потока построчно.

Пусть заранее создан файл со следующим содержимым:

```
hello world!  
second line
```

Для работы с файлами нужно подключить библиотеку `fstream`:

```
#include <fstream>
```

Чтобы считать содержимое файла следует объявить экземпляр класса `ifstream`:

```
ifstream input("hello.txt");
```

В качестве аргумента конструктора указывается путь до желаемого файла.

Далее можно создать строковую переменную, в которую будет записан результат чтения из файла:

```
string line;
```

Функция `getline` первым аргументом принимает поток, из которого нужно прочитать данные, а вторым — переменную, в которую их надо записать. Чтобы проверить, что все работает, можно вывести переменную `line` на экран:

```
getline(input, line);
cout << line << endl;
```

Чтобы считать и вторую строчку, можно попробовать запустить следующий код:

```
getline(input, line);
cout << line << endl;

getline(input, line);
cout << line << endl;
```

Если вызвать `getline` в третий раз, то она не изменит переменную `line`, так как уже достигнут конец файла и из него ничего не может быть прочитано:

```
getline(input, line);
cout << line << endl;

getline(input, line);
cout << line << endl;

getline(input, line);
cout << line << endl;
```

Чтобы избежать таких ошибок, следует помнить, что `getline` возвращает ссылку на поток, из которого берет данные. Поток можно привести к типу `bool`, причем `false` будет в случае, когда с потоком уже можно дальше не работать.

Переписать код так, чтобы он выводил все строчки из файла и ничего лишнего, можно так:

```
ifstream input("hello.txt");
string line;
while (getline(input, line)) {
    cout << line << endl;
}
```

Следует обратить внимание, что переводы строки при выводе добавлены искусственно. Это связано с тем, что функция `getline`, на самом деле, считывает данные до некоторого разделителя, причем по умолчанию до символа перевода строки, который в считанную строку не попадает.

4.2.3. Обработка случая, когда указанного файла не существует

Рассмотрим ситуацию, когда по некоторым причинам неверно указано имя файла или файла с таким именем не может существовать в файловой системе. Например, внесем опечатку:

```
ifstream input("helol.txt");
```

При запуске этого кода оказывается, что он работает, ничего не выводит, но никак не сигнализирует о наличии ошибки.

Вообще говоря, желательно, чтобы программа не умалчивала об этом, а явно сообщала, что файла не существует и из него нельзя прочитать данные.

У файловых потоков существует метод `is_open`, который возвращает `true`, если файловый поток открыт и готов работать. Программу, таким образом, следует переписать так:

```
ifstream input("helol.txt");
string line;

if (input.is_open()){
    while (getline(input, line)) {
        cout << line << endl;
    }
    cout << "done!" << endl;
} else {
    cout << "error!" << endl;
}
```

Следует также отметить, что файловые потоки можно приводить к типу `bool`, причем значение `true` соответствует тому, что с потоком можно работать в данный момент. Другими словами, код можно переписать в следующем виде:

```
ifstream input("helol.txt");
string line;

if (input){
    while (getline(input, line)) {
        cout << line << endl;
    }
    cout << "done!" << endl;
} else {
    cout << "error!" << endl;
}
```

4.2.4. Чтение из потока до разделителя

Научимся считывать данные с помощью `getline` поблочно с некоторым разделителем. Например, в качестве разделителя может выступать символ «минус». Допустим, считать нужно дату из следующего текстового файла `date.txt`:

2017–01–25

Для этого создадим:

```
ifstream input("date.txt");
```

Объявим строковые переменные `year`, `month`, `day`.

```
string year;
string month;
string day;
```

Нужно считать файл таким образом, чтобы соответствующие части файла попали в нужную переменную. Воспользуемся функцией `getline` и укажем разделитель:

```
if (input) {
    getline(input, year, '-');
    getline(input, month, '-');
    getline(input, day, '-');
}
```

Чтобы проверить, что все работает, выведем переменную на экран через пробел:

```
cout << year << ' ' << month << ' ' << day << endl;
```

4.2.5. Оператор чтения из потока

Решим ту же самую задачу с помощью оператора чтения из потока (`>>`). Записывать считанные данные будем в переменные типа `int`.

```
ifstream input("date.txt");
int year = 0;
int month = 0;
int day = 0;
if (input) {
    input >> year;
    input.ignore(1);
    input >> month;
```

```
    input.ignore(1);
    input >> day;
    input.ignore(1);
}
cout << year << ' ' << month << ' ' << day << endl;
```

После того, как из потока будет считан год, следующим символом будет «минус», от которого нужно избавиться. Это можно сделать с помощью метода ignore, который принимает целое число — сколько символов нужно пропустить. Аналогичночитываются месяц и день. Получается такой же результат.

То, каким методом пользоваться, зависит от ситуации. Иногда бывает удобнее сперва считать всю строку целиком.

4.2.6. Оператор записи в поток. Дозапись в файл.

Данные в файл можно записывать с помощью класса ofstream:

```
const string path = "output.txt";

ofstream output(path);
output << "hello" << endl;
```

После проверим, что записалось в файл, открыв его и прочитав содержимое:

```
ifstream input(path);
if (input) {
    string line;
    while (getline(input, line)) {
        cout << line << endl;
    }
}
```

Чтобы избежать дублирования кода, имеет смысл создать переменную path.

Для удобства можно создать функцию, которая будет считывать весь файл:

```
void ReadAll(const string& path) {
    ifstream input(path);
    if (input) {
        string line;
        while (getline(input, line)) {
            cout << line << endl;
```

```
    }
}
}
```

Предыдущая программа примет вид:

```
const string path = "output.txt";

ofstream output(path);
output << "hello" << endl;

ReadAll(path);
```

Следует отметить, что при каждом запуске программы файл записывается заново, то есть его содержимое удалялось и запись начиналась заново.

Для того, чтобы открыть файл в режиме дозаписи, нужно передать специальный флагок `ios::app` (от англ. append):

```
ofstream output(path, ios::app);
output << " world!" << endl;
```

4.2.7. Форматирование вывода. Файловые манипуляторы.

Допустим, нужно в определенном формате вывести данные. Это могут быть имена колонок и значения в этих колонках.

Сохраним в векторе `names` имена колонок и после этого создадим вектор значений:

```
vector<string> names = {"a", "b", "c"};
vector<double> values = {5, 0.01, 0.000005};
```

Выведем их на экран:

```
for (const auto& n : names) {
    cout << n << ' ';
}
cout << endl;
for (const auto& v : values) {
    cout << v << ' ';
}
cout << endl;
```

При этом читать значения очень неудобно.

Для того, чтобы решить такую задачу, в языке C++ есть файловые манипуляторы, которые работают с потоком и изменяют его поведение. Для того, чтобы с ними работать, нужно подключить библиотеку iomanip.

fixed Указывает, что числа далее нужно выводить на экран с фиксированной точностью.

```
cout << fixed;
```

setprecision Задает количество знаков после запятой.

```
cout << fixed << setprecision(2);
```

setw (set width) Указывает ширину поля, которое резервируется для вывода переменной.

```
cout << fixed << setprecision(2);
cout << setw(10);
```

Этот манипулятор нужно использовать каждый раз при выводе значения, так как он сбрасывается после вывода следующего значения:

```
for (const auto& n : names) {
    cout << setw(10) << n << ' ';
}
cout << endl;
cout << fixed << setprecision(2);
for (const auto& v : values) {
    cout << setw(10) << v << ' ';
}
```

Здесь колонки были выведены в таком же формате.

setfill Указывает, каким символом заполнять расстояние между колонками.

```
cout << setfill(' ');
```

left Выравнивание по левому краю поля.

```
cout << left;
```

Для удобства напишем функцию, которая будет на вход принимать вектора имен и значений, и выводить их в определенном формате:

```
void Print(const vector<string>& names,
           const vector<double>& values, int width) {
    for (const auto& n : names) {
        cout << setw(width) << n << ' ';
    }
    cout << endl;
    cout << fixed << setprecision(2);
    for (const auto& v : values) {
        cout << setw(width) << v << ' ';
    }
    cout << endl;
}
```

Покажем как пользоваться манипуляторами `setfill` и `left`:

```
cout << setfill('.');
cout << left;
Print(names, values, 10);
```