Потоки и система ввода-вывода

Базовые типы для работы с потоками

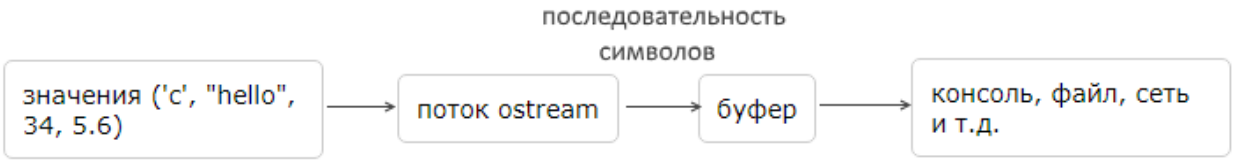
Все инструменты для работы с системой ввода-вывода и потоками в языке C++ определены в стандартной библиотеке. Заголовочный файл iostream определяет следующие базовые типы для работы с потоками:

- istream и wistream: читают данные с потока
- ostream и wostream: записывают данные в поток
- iostream и wiostream: читают и записывают данные в поток

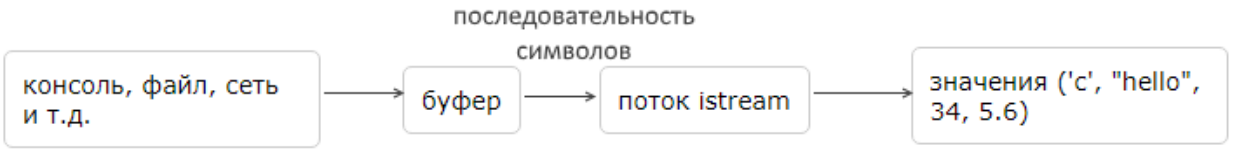
Для каждого типа определен его двойник, который начинается на букву w и который предназначен для поддержки данных типа wchar_t.

Эти типы являются базовыми для других классов, управляющих потоками ввода-вывода.

Объект типа ostream получает значения различных типов, преобразует их в последовательность символов и передает их через буфер в определенное место для вывода (консоль, файл, сетевые интерфейсы и т.д.)



Поток istream получает через буфер из определенного места последовательности символов (с консоли, из файла, из сети и т.д.) и преобразует эти последовательности в значения различных типов. То есть когда мы вводим данные (с той же клавиатуры в консоли), сначала данные накапливаются в буфере и только затем передаются объекту istream.



По умолчанию в стандартной библиотеке определены объекты этих классов - cout, cin, cerr, которые работают с консолью.

Запись в поток

Для записи данных в поток ostream применяется оператор <<. Этот оператор получает два операнда. Левый операнд представляет объект типа ostream, а правый операнд - значение, которое надо вывести в поток.

К примеру, по умолчанию стандартная библиотека C++ предоставляет объект cout, который представляет тип ostream и позволяет выводить данные на консоль:

```
#include <iostream>

int main()
{
    std::cout << "Hello" << std::endl;
}
```

Так как оператор << возвращает левый операнд - cout, то с помощью цепочки операторов мы можем передать на консоль несколько значений:

```
std::cout << "Hello" << " world" << std::endl;
```

Чтение данных

Для чтения данных из потока применяется оператор ввода >>, который принимает два операнда. Левый операнд представляет поток istream, с которого производится считывание, а правый операнд - объект, в который считываются данные.

Для чтения с консоли применяется объект cin, который представляет тип istream.

```
#include <iostream>

int main()
{
    int age;
    double weight;
    std::cout << "Input age: ";
    std::cin >> age;
    std::cout << "Input weight: ";
    std::cin >> weight;
    std::cout << "Your age: " << age << "\t your weight: " << weight << std::endl;
}
```

Однако такой способ не очень подходит для чтения строк с консоли особенно когда считываемая строка содержит пробельные символы. В этом случае лучше использовать встроенную функцию getline(), которая в качестве параметра принимает поток istream и переменную типа string, в которую надо считать данные:

```
#include <iostream>

int main()
{
    std::string name;
    std::cout << "Input name: ";
```

```
    getline(std::cin, name);
    //std::cin >> name;
    std::cout << "Your name: " << name <<std::endl;
}
```

Пример работы программы:

```
Input name: Tom Smit
Your name: Tom Smit
```

По умолчанию признаком окончания ввода служит перевод на другую строку, например, с помощью клавиши Enter. Но также можно задать свой признак окончания ввода с помощью дополнительного параметра функции `getline()`. Для этого надо передать символ, который будет служить окончанием ввода:

```
#include <iostream>

int main()
{
    std::string text;
    std::cout << "Input text: " << std::endl;
    getline(std::cin, text, '*');   // окончанием ввода будет служить символ *
    std::cout << "\nYour text:" <<std::endl;
    std::cout << text <<std::endl;
}
```

В данном случае ввод завершится, когда пользователь введет символ *. Таким образом, мы можем ввести многострочный текст, но при вводе звездочки ввод завершится. Пример работы программы:

```
Input text:
Hello World
Good bye world*

Your text:
Hello World
Good bye world
```

Вывод ошибок

Для вывода сообщения об ошибке на консоль применяется объект cerr, который представляет объект типа ostream:

```
#include <iostream>

int main()
{
    std::cerr << "Error occured" << std::endl;
}
```

Потоки символов wchar_t

Для работы с потоками данных типов wchar_t в стандартной библиотеке определены объекты wcout (тип wostream), wcerr (тип wostream) и wcin (тип wistream), которые являются аналогами для объектов cout, cerr и cin и работают аналогично

```
#include <iostream>

int main()
{
    int age;
    double weight;
    std::wcout << "Input age: ";
    std::wcin >> age;
    std::wcout << "Input weight: ";
    std::wcin >> weight;
    if (age <= 0 || weight <= 0)
        std::wcerr << "Invalid data" << std::endl;
    else
        std::wcout << "Your age: " << age << "\t your weight: " << weight << std::endl;
}
```

Файловые потоки. Открытие и закрытие

Для работы с файлами в стандартной библиотеке определен заголовочный файл fstream, который определяет базовые типы для чтения и записи файлов. В частности, это:

- ifstream: для чтения с файла
- ofstream: для записи в файл
- fstream: совмещает запись и чтение

Для работы с данными типа wchar_t для этих потоков определены двойники:

- wifstream
- wofstream
- wfstream

Открытие файла

При операциях с файлом вначале необходимо открыть файл с помощью функции open(). Данная функция имеет две версии:

- `open(путь)`
- `open(путь, режим)`

Для открытия файла в функцию необходимо передать путь к файлу в виде строки. И также можно указать режим открытия. Список доступных режимов открытия файла:

- ios::in: файл открывается для ввода (чтения). Может быть установлен только для объекта ifstream или fstream

- ios::out: файл открывается для вывода (записи). При этом старые данные удаляются. Может быть установлен только для объекта ofstream или fstream
- ios::app: файл открывается для дозаписи. Старые данные не удаляются.
- ios::ate: после открытия файла перемещает указатель в конец файла
- ios::trunc: файл усекается при открытии. Может быть установлен, если также установлен режим `out`
- ios::binary: файл открывается в бинарном режиме

Если при открытии режим не указан, то по умолчанию для объектов ofstream применяется режим `ios::out`, а для объектов ifstream - режим `ios::in`. Для объектов fstream совмещаются режимы `ios::out` и `ios::in`.

```
std::ofstream out;          // поток для записи
out.open("hello1.txt"); // окрываем файл для записи

std::ofstream out2;
out2.open("hello2.txt", std::ios::app); // окрываем файл для дозаписи

std::ofstream out3;
out2.open("hello3.txt", std::ios::out | std::ios::trunc); // установка нескольких режимов

std::ifstream in;       // поток для чтения
in.open("hello4.txt"); // окрываем файл для чтения

std::fstream fs;        // поток для чтения-записи
fs.open("hello5.txt"); // окрываем файл для чтения-записи
```

Однако в принципе необязательно использовать функцию open для открытия файла. В качестве альтернативы можно также использовать конструктор объектов-потоков и передавать в них путь к файлу и режим открытия:

```
fstream(путь)
fstream(путь, режим)
```

При вызове конструктора, в который передан путь к файлу, данный файл будет автоматически открываться:

```
std::ofstream out("hello.txt");
std::ifstream in("hello.txt");
std::fstream fs("hello.txt", std::ios::app);
```

В данном случае предполагается, что файл "hello.txt" располагается в той же папке, где и файл программы.

Вообще использование конструкторов для открытия потока является более предпочтительным, так как определение переменной, представляющей файловой поток, уже преполагает, что этот поток будет открыт для чтения или записи. А использование конструктора избавит от ситуации, когда мы забудем открыть поток, но при этом начнем его использовать.

В процессе работы мы можем проверить, окрыт ли файл с помощью функции is_open(). Если файл открыт, то она возвращает true:

```
std::ifstream in;       // поток для чтения
in.open("hello.txt"); // окрываем файл для чтения
// если файл открыт
if (in.is_open())
{
}
```

Закрытие файла

После завершения работы с файлом его следует закрыть с помощью функции close(). Также стоит отметить, то при выходе объекта потока из области видимости, он удаляется, и у него автоматически вызывается функция close.

```
#include <iostream>
#include <fstream>

int main()
{
    std::ofstream out;          // поток для записи
    out.open("hello.txt"); // окрываем файл для записи
    out.close();            // закрываем файл

    std::ifstream in;       // поток для чтения
    in.open("hello.txt"); // окрываем файл для чтения
    in.close();             // закрываем файл

    std::fstream fs;        // поток для чтения-записи
    fs.open("hello.txt"); // окрываем файл для чтения-записи
    fs.close();             // закрываем файл
}
```

Чтение и запись текстовых файлов

Потоки для работы с текстовыми файлами представляют объекты, для которых не задан режим открытия ios::binary.

Запись в файл

Для записи в файл к объекту ofstream или fstream применяется оператор << (как и при выводе на консоль) (файл **write_in_file.cpp**):

```
#include <iostream>
#include <fstream>

int main()
{
    std::ofstream out;          // поток для записи
    out.open("hello.txt");      // открываем файл для записи
    if (out.is_open())
```

```
    {
        out << "Hello World!" << std::endl;
    }
    out.close();
    std::cout << "File has been written" << std::endl;
}
```

Здесь предполагается, что файла "hello.txt" располагается в одной папке с файлом программы. Данный способ перезаписывает файл заново. Если надо дозаписать текст в конец файла, то для открытия файла нужно использовать режим ios::app:

```
std::ofstream out("hello.txt", std::ios::app);
if (out.is_open())
{
    out << "Welcome to C++" << std::endl;
}
out.close();
```

Чтение из файла

Если надо считать всю строку целиком или даже все строки из файла, то лучше использовать встроенную функцию getline(), которая принимает поток для чтения и переменную, в которую надо считать текст (файл **read_from_file.cpp**):

```
#include <iostream>
#include <fstream>
#include <string>      // для std::getline

int main()
{
    std::string line;

    std::ifstream in("hello.txt"); // окрываем файл для чтения
    if (in.is_open())
    {
        while (std::getline(in, line))
        {
            std::cout << line << std::endl;
        }
    }
    in.close();     // закрываем файл
}
```

Также для чтения данных из файла для объектов ifstream и fstream может применяться оператор >> (также как и при чтении с консоли) (файл **file_class.cpp**):

```
#include <iostream>
#include <fstream>
#include <vector>

struct Point
{
```

```
    Point(double x, double y): x{x}, y{y} {}
    double x;
    double y;
};
int main()
{
    std::vector<Point> points{ Point{0, 0}, Point{4, 5}, Point{-5, 7}};

    std::ofstream out("points.txt");

    if (out.is_open())
    {
        // записываем все объекты Point в файл
        for (const Point& point: points)
        {
            out << point.x << " " << point.y << std::endl;
        }
    }
    out.close();

    std::vector<Point> new_points;
    std::ifstream in("points.txt"); // окрываем файл для чтения
    if (in.is_open())
    {
        double x, y;
        while (in >> x >> y)
        {
            new_points.push_back(Point{x, y});
        }
    }
    in.close();

    for (const Point& point: new_points)
    {
        std::cout << "Point X: " << point.x << "\tY: " << point.y << std::endl;
    }
}
```

Здесь вектор структур Point записывается в файл.

```
for (const Point& point: points)
{
    out << point.x << " " << point.y << std::endl;
}
```

При чем при записи значений переменных файл они отделяются пробелом. В итоге будет создаваться файл в формате

```
0 0
4 5
-5 7
```

Используя оператор >>, можно считать последовательно данные в переменные x и y и ими инициализировать структуру.

```
double x, y;
while (in >> x >> y)
{
   new_points.push_back(Point{x, y});
}
```

Но стоит отметить, что это ограниченный способ, поскольку при чтении файла поток in использует пробел для отделения одного значения от другого и таким образом считывает эти значения в переменные x и y. Если же нам надо записать и затем считать строку, которая содержит пробелы, и какие-то другие данные, то такой способ, конечно, не сработает.

Переопределение операторов ввода и вывода

Операторы ввода >> и вывода << прекрасно работают для примитивных типов данных, таких как int или double. В то же время для использования их с объектами классов необходимо переопределять эти операторы.

Оператор <<

Стандартный выходной поток cout имеет тип std::ostream. Поэтому первый параметр (левый операнд) операции << представляет ссылку на неконстантный объект ostream. Данный объект не должен представлять константу, так как запись в поток изменяет его состояние. Причем параметр представляет именно ссылку, так как нельзя копировать объект класса ostream.

Второй параметр оператора определяется как ссылка на константный объекта класса, который надо вывести в поток.

Для совместимости с другими операторами переопределяемый оператор должен возвращать значение параметра std::ostream.

Также следует отметить, что операторы ввода и вывода не должны быть членами в классе, а определяются вне класса как обычные функции.

```
#include <iostream>

class Person
{
public:
   Person(std::string name, unsigned age): name{name}, age{age} {}
   std::string getName() const {return name;}
   unsigned getAge() const {return age;}

   void setName(std::string personName){ name = personName;}
   void setAge(unsigned personAge){ age = personAge;}
private:
   std::string name;
   unsigned age;
};
std::ostream& operator << (std::ostream &os, const Person &person)
```

```
{
    return os << person.getName() << " " << person.getAge();
}
int main()
{
    Person tom{"Tom", 38};
    std::cout << tom << std::endl;

    Person bob{"Bob", 42};
    std::cout << bob << std::endl;
}
```

В данном случае оператор вывода определяется для объектов структуры Person. Сам оператор по сути просто выводит имя и возраст пользователя через пробел. Консольный вывод программы:

```
Tom 38
Bob 42
```

Оператор >>

Первый параметр оператора >> представляет ссылку на объект istream, с которого осуществляется чтение. Второй параметр представляет ссылку на неконстантный объект, в который надо считать данные. В качестве результата оператор возвращают ссылку на поток ввода istream из первого параметра. Пример: файл **in_operator.cpp**.

```
#include <iostream>

class Person
{
public:
    Person(std::string name, unsigned age): name{name}, age{age} {}
    std::string getName() const {return name;}
    unsigned getAge() const {return age;}

    void setName(std::string personName){ name = personName;}
    void setAge(unsigned personAge){ age = personAge;}
private:
    std::string name;
    unsigned age{};
};
std::istream& operator >> (std::istream& in, Person& person)
{
    std::string name;
    unsigned age;
    in >> name >> age;
    person.setName(name);
    person.setAge(age);
    return in;
}
int main()
{
```

```
    Person bob{"",0};
    std::cout << "Input name and age: ";
    std::cin >> bob;
    std::cout << "Name: " << bob.getName() << "\tAge: " << bob.getAge() << std::endl;
}
```

Оператор ввода последовательно считывает из потока данные в переменные name и age и затем использует их для установки имени и возраста пользователя.

```
std::istream& operator >> (std::istream& in, Person& person)
{
    std::string name;
    unsigned age;
    in >> name >> age;
    person.setName(name);
    person.setAge(age);
    return in;
}
```

При этом в данном случае предполагается, что имя представляет одно слово. Если надо считать сложное имя, которое состоит из нескольких слов, или имя и фамилию, то естественно надо определять более сложную логику.

Пример работы программы:

```
Input name and age: Bob 42
Name: Bob       Age: 42
```

Однако что если мы введем для возраста вместо числа строку? В этом случае переменная age получит неопределенное значение. Существуют различные варианты, как обрабатывать подобные ситуации. Но в качестве примера мы можем в случае некорректного ввода устанавливать значение по умолчанию:

```
std::istream& operator >> (std::istream& in, Person& person)
{
    std::string name;
    unsigned age;
    in >> name >> age;
    if (in)
    {
        person.setName(name);
        person.setAge(age);
    }
    return in;
}
```

С помощью выражения `if(in)` проверяем, является ли ввод удачным. Если он завершился успешно, то устанавливаем введенные значения. Если же ввод не удался, у объекта Person остаются те значения, которые у него было до ввода.

Чтение и запись файла

Определив операторы ввода и выводы, мы можем их использовать также и для чтения и записи файла (файл **file_stream_class.cpp**):

```
#include <iostream>
#include <fstream>
#include <vector>

class Person
{
public:
    Person(std::string name, unsigned age): name{name}, age{age} {}
    std::string getName() const {return name;}
    unsigned getAge() const {return age;}

    void setName(std::string personName){ name = personName;}
    void setAge(unsigned personAge){ age = personAge;}
private:
    std::string name;
    unsigned age{};
};
std::ostream& operator << (std::ostream &os, const Person &person)
{
    return os << person.getName() << " " << person.getAge();
}
std::istream& operator >> (std::istream& in, Person& person)
{
    std::string name;
    unsigned age;
    in >> name >> age;
    // если ввод не удался, устанавливаем некоторые значения по умолчанию
    if (in)
    {
        person.setName(name);
        person.setAge(age);
    }
    return in;
}
int main()
{
    // начальные данные - вектор объектов Person
    std::vector<Person> people =
    {
        Person{"Tom", 23},
        Person{"Bob", 25},
        Person{"Alice", 22},
        Person{"Kate", 31}
    };
    // запись данных в файл
    std::ofstream out("people.txt");
    if (out.is_open())
    {
```

```
        for (const Person& person: people)
        {
            out << person << std::endl;
        }
    }
    out.close();
    // вектор для считываемых данных
    std::vector<Person> new_people;
    // чтение ранее записанных данных из файла
    std::ifstream in("people.txt");
    if (in.is_open())
    {
        Person person{"",0};
        while (in >> person)
        {
            new_people.push_back(person);
        }
    }
    in.close();
    // вывод считанных данных на консоль
    std::cout << "All people:" << std::endl;
    for (const Person& person: new_people)
    {
        std::cout << person << std::endl;
    }
}
```

Здесь для класса Person определены операторы ввода и вывода. С помощью оператора вывода данные будут записываться в файл users.txt, а с помощью оператора ввода - считываться из файла. В конце считанные данные выводятся на консоль:

Результат работы программы:

```
All users:
Tom 23
Bob 25
Alice 22
Kate 31
```