stringstream

Чтобы использовать stringstream, нужно подключить заголовочный файл sstream. Чтобы занести данные в stringstream мы можем использовать оператор вставки (<<):

```cpp
#include <iostream>
#include <sstream> // для stringstream

int main()
{
    std::stringstream myString;
    myString << "Hello, world" << std::endl; // вставляем "Hello, world" в stringstream
}
```

Или функцию str(string):

```cpp
#include <iostream>
#include <sstream> // для stringstream

int main()
{
    std::stringstream myString;
    myString.str("Hello, world"); // присваиваем буферу stringstream значение "Hello, world"
}
```

stringstream

Чтобы получить данные обратно из stringstream, мы можем использовать функцию str():

```cpp
#include <iostream>
#include <sstream> // для stringstream

int main()
{
    std::stringstream myString;
    myString << "336000 12.14" << std::endl;
    std::cout << myString.str();
}
```

stringstream

Чтобы получить данные из stringstream, можно использовать также оператор извлечения >>

```cpp
#include <iostream>
#include <sstream> // для stringstream

int main()
{
    std::stringstream myString;
    myString << "336000 12.14"; // вставляем (числовую) строку в поток

    std::string part1;
    myString >> part1;

    std::string part2;
    myString >> part2;

    // выводим числа
    std::cout << part1 << " and " << part2 << std::endl;
}
```

Конвертация чисел в строки

```cpp
#include <iostream>
#include <sstream> // для stringstream

int main()
{
    std::stringstream myString;

    int nValue = 336000;
    double dValue = 12.14;
    myString << nValue << " " << dValue;

    std::string strValue1, strValue2;
    myString >> strValue1 >> strValue2;

    std::cout << strValue1 << " " << strValue2 << std::endl;
}
```

Конвертация строк в числа

```cpp
#include <iostream>
#include <sstream> // для stringstream

int main()
{
    std::stringstream myString;
    myString << "336000 12.14"; // вставляем (числовую) строку в поток
    int nValue;
    double dValue;

    myString >> nValue >> dValue;

    std::cout << nValue << " " << dValue << std::endl;
}
```

Очистка stringstream для повторного использования

```cpp
#include <iostream>
#include <sstream> // для stringstream

int main()
{
    std::stringstream myString;
    myString << "Hello ";

    myString.str(""); // очищаем буфер

    myString << "Hello ";

    myString.str(std::string()); // очищаем буфер
}
```

Файловый вывод

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib> // для использования exit()

int main()
{
    using namespace std;

    // ofstream используется для записи данных в файл
    // Создаём файл SomeText.txt
    ofstream outf("SomeText.txt");

    // Если мы не можем открыть этот файл для записи данных в него
    if (!outf)
    {
        // То выводим сообщение об ошибке и выполняем exit()
        cerr << "Uh oh, SomeText.txt could not be opened for writing!" << endl;
        exit(1); // окончание работы программы
    }

    // Записываем в файл следующие две строчки
    outf << "See line #1!" << endl;
    outf << "See line #2!" << endl;

    return 0;

    // Когда outf выйдет из области видимости, то деструктор класса ofstream автоматически закроет наш файл
}
```

Файловый ввод

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib> // для использования exit()

int main()
{
    using namespace std;

    // ifstream используется для чтения содержимого файлов
    // Мы попытаемся прочитать содержимое файла SomeText.txt
    ifstream inf("SomeText.txt");

    // Если мы не можем открыть файл для чтения его содержимого
    if (!inf)
    {
        // То выводим следующее сообщение об ошибке и выполняем exit()
        cerr << "Uh oh, SomeText.txt could not be opened for reading!" << endl;
        exit(1);
    }

    // Пока есть, что читать
    while (inf)
    {
        // То перемещаем то, что можем прочитать, в строку, а затем выводим эту строку на экран
        string strInput;
        getline(inf, strInput);
        cout << strInput << endl;
    }

    return 0;
    // Когда inf выйдет из области видимости, то деструктор класса ifstream автоматически закроет наш файл
}
```

Режимы открытия файлов

app — открывает файл в режиме добавления;

ate — переходит в конец файла перед чтением/записью;

binary — открывает файл в бинарном режиме (вместо текстового режима);

in — открывает файл в режиме чтения (по умолчанию для ifstream);

out — открывает файл в режиме записи (по умолчанию для ofstream);

trunc — удаляет файл, если он уже существует.

Режимы открытия файлов

```cpp
#include <iostream>
#include <cstdlib> // для использования exit()
#include <fstream>

int main()
{
    using namespace std;

    /* Передаём флаг ios:app, чтобы сообщить fstream, что мы собираемся добавить
    свои данные к уже существующим данным файла,
    мы не собираемся перезаписывать файл. Нам не нужно передавать флаг ios::out,
    поскольку ofstream по умолчанию работает в режиме ios::out */
    ofstream outf("SomeText.txt", ios::app);

    outf << "See line #3!" << endl;
    outf << "See line #4!" << endl;

    return 0;
    // Когда outf выйдет из области видимости, то деструктор класса ofstream автоматически закроет наш файл
}
```

Произвольный доступ к файлу

Каждый класс файлового ввода/вывода содержит файловый указатель, который используется для отслеживания текущей позиции чтения/записи данных в файл. Любая запись в файл или чтение содержимого файла происходит в текущем местоположении файлового указателя. По умолчанию, при открытии файла для чтения или записи, файловый указатель находится в самом начале этого файла. Однако, если файл открывается в режиме добавления, то файловый указатель перемещается в конец файла, чтобы пользователь имел возможность добавить данные в файл, а не перезаписать его.

Произвольный доступ к файлу осуществляется путём манипулирования файловым указателем с помощью методов seekg() (для ввода) и seekp() (для вывода).

inf.seekg(15, ios::cur); // перемещаемся вперёд на 15 байтов относительно текущего местоположения файлового указателя
inf.seekg(-17, ios::cur); // перемещаемся назад на 17 байтов относительно текущего местоположения файлового указателя
inf.seekg(24, ios::beg); // перемещаемся к 24-ому байту относительно начала файла
inf.seekg(25); // перемещаемся к 25-ому байту файла
inf.seekg(-27, ios::end); // перемещаемся к 27-ому байту от конца файла
inf.seekg(0, ios::beg); // перемещаемся в начало файла
inf.seekg(0, ios::end); // перемещаемся в конец файла

Произвольный доступ к файлу

Есть ещё две другие полезные функции: tellg() и tellp(), которые возвращают абсолютную позицию файлового указателя. Это полезно при определении размера файла:

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

int main()
{
    std::ifstream inf("SomeText.txt");
    inf.seekg(0, std::ios::end); // перемещаемся в конец файла
    std::cout << inf.tellg();
}
```

Одновременное чтение и запись в файл с помощью fstream

```cpp
#include <iostream>
#include <fstream>
#include <string>

int main()
{
    using namespace std;

    // Мы должны указать как in, так и out, поскольку используем fstream
    fstream iofile("SomeText.txt", ios::in | ios::out);

    string strInput;
    getline(iofile, strInput); // прочитать строку
    cout << strInput << endl;

    char chChar;
    iofile.get(chChar); // прочитать один символ
    cout << chChar;

    iofile.put('0'); // записать один символ

    return 0;
}
```

Работа с файловой системой

```cpp
#include <experimental/filesystem>
#include <iomanip>
#include <iostream>

int main(int argc, char ** argv)
{
    system("chcp 1251");
    std::experimental::filesystem::path file = "./main.cpp";

    // проверка существования файла в файловой системе
    if (!std::experimental::filesystem::exists(file)) {
        std::cerr << "Path " << file << " does not exist.\n";
        return EXIT_FAILURE;
    }
    else {
        // для вывода пути файла в виде строки нужно использовать метод string()
        std::cout << "Path " << file.string() << " exists." << std::endl;
    }
    // конвертирует путь в абсолютный путь, не содержащий символов . (текущая директория)
    // и .. (родительская директория)
    std::cout << std::experimental::filesystem::canonical(file).string() << std::endl;
    std::experimental::filesystem::path path = "././main.cpp";
    std::cout <<
        // путь к текущему файлу
        "current " << std::experimental::filesystem::current_path().string() << std::endl <<
        // абсолютный путь
        "absolute " << std::experimental::filesystem::absolute(path).string() << std::endl << std::endl;
}
```

Работа с файловой системой

```cpp
#include <experimental/filesystem>
#include <iomanip>
#include <iostream>

int main(int argc, char ** argv)
{
    system("chcp 1251");
    auto path = std::experimental::filesystem::current_path() / "main.cpp";
    for (const auto & part : path)
    {
        std::cout << part.string() << std::endl;
    }
    std::cout << std::boolalpha <<
        "exists() = " << std::experimental::filesystem::exists(path) << std::endl <<
        "root_name() = " << path.root_name().string() << std::endl <<
        "root_directory() = " << path.root_directory().string() << std::endl <<
        "root_path() = " << path.root_path().string() << std::endl <<
        "relative_path() = " << path.relative_path().string() << std::endl <<
        "parent_path() = " << path.parent_path().string() << std::endl <<
        "filename() = " << path.filename().string() << std::endl <<
        "stem() = " << path.stem().string() << std::endl <<
        "extension() = " << path.extension().string() << std::endl;
    try {
        std::cout << "canonical() = " << std::experimental::filesystem::canonical(path).string() << std::endl;
    }
    catch (const std::experimental::filesystem::filesystem_error & exception) {
        std::cout << "exception: " << exception.what() << std::endl;
    }
}
```

Работа с файловой системой. Создание директорий

```cpp
#include <experimental/filesystem>
#include <fstream>
#include <iomanip>
#include <iostream>

int main(int argc, char ** argv)
{
    //system("chcp 1251");
    auto path = std::experimental::filesystem::current_path();
    // создание директории
    std::experimental::filesystem::create_directory(path / "directory_1");
    std::experimental::filesystem::create_directory(path / "directory_1", path);
    // создание вложенных директорий
    std::experimental::filesystem::create_directories(path / "directory_2" / "directory_3");
    std::experimental::filesystem::create_directories("A/B/C");
    // открытие файла и запись символа 'a'
    std::fstream("A/file_1.txt", std::ios::out).put('a');
    // копирование содержимого файла file_1.txt в файл file_2.txt
    std::experimental::filesystem::copy("A/file_1.txt", "A/file_2.txt");
    std::experimental::filesystem::copy("A/B", "A/BB"); // копирование директорий
    // копирование директории со всеми файлами в ней
    std::experimental::filesystem::copy("A", "AA", std::experimental::filesystem::copy_options::recursive);
    // переименование директории
    std::experimental::filesystem::rename("AA", "new_name");
}
```

Работа с файловой системой. Размер файлов

```cpp
#include <experimental/filesystem>
#include <fstream>
#include <iomanip>
#include <iostream>

auto compute_file_size(const std::experimental::filesystem::path & path)
{
    if (std::experimental::filesystem::exists(path) && std::experimental::filesystem::is_regular_file(path)){
        auto file_size = std::experimental::filesystem::file_size(path);
        if (file_size != static_cast <uintmax_t> (-1)) {
            return file_size;
        }
    }
    return static_cast <uintmax_t> (-1);
}

auto compute_directory_size(const std::experimental::filesystem::path & path)
{
    uintmax_t size = 0ULL;
    if (std::experimental::filesystem::exists(path) && std::experimental::filesystem::is_directory(path)) {
        for (auto const & entry : std::experimental::filesystem::recursive_directory_iterator(path)) {
            if (std::experimental::filesystem::is_regular_file(entry.status()) ||
             std::experimental::filesystem::is_symlink(entry.status())) {
                auto file_size = std::experimental::filesystem::file_size(entry);
                if (file_size != static_cast <uintmax_t> (-1)) {
                    size += file_size;
    }   }   }   }
    return size;
}
```

Работа с файловой системой. Размер файлов

```cpp
void view_directory(const std::experimental::filesystem::path & path)
{
    if (std::experimental::filesystem::exists(path) && std::experimental::filesystem::is_directory(path))
    {
        for (const auto & entry : std::experimental::filesystem::directory_iterator(path))
        {
            auto file_name = entry.path().filename().string();
            std::cout << file_name << std::endl;
        }
    }
}

int main(int argc, char ** argv)
{
    system("chcp 1251");
    std::cout << compute_file_size("./main.cpp") << std::endl;
    std::cout << compute_directory_size(std::experimental::filesystem::current_path()) << std::endl;
    view_directory(std::experimental::filesystem::current_path());
}
```

Формат JSON

JSON (JavaScript Object Notation) - формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Он основан на подмножестве языка программирования JavaScript.

JSON основан на двух структурах данных:

- Коллекция пар ключ/значение (map в C++).
- Упорядоченный список значений (vector в C++).

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address": {
       "streetAddress": "Московское ш., 101, кв.101",
       "city": "Ленинград",
       "postalCode": 101101
   },
   "phoneNumbers": [
       "812 123-1234",
       "916 123-4567"
   ]
}
```