# Лабораторная работа № 1 «Начало работы»

## Цель работы

Изучение объектно-ориентированного программирования и получение навыков написания программ, содержащих файловый ввод-вывод, контейнеры, итераторы и регулярные выражения на примере языка программирования C++.

## Задание

Написать программу, позволяющую пользователю извлекать из текстового документа, написанного с использованием языка разметки, определённое содержимое. Язык разметки и извлекаемое содержимое выбираются в соответствии с вариантом (таблица 1.1).

Имя файла передаётся как аргумент командной строки.

Для создания и управления коллекциями объектов использовать контейнеры. Для обхода контейнеров использовать итераторы.

Поиск производить посредством регулярных выражений.

### Пример ожидаемого результата выполнения программы (Вариант 1)

**Input:**

D:\...\lw01.exe filename.html

**Output:**

header Библиотека регулярных выражений

paragraph Библиотека регулярных выражений предоставляет класс, представляющий регулярные выражения, которые являются своего рода мини-языком, используемым для выполнения сопоставления шаблонов в строках. Почти все операции с регулярными выражениями можно охарактеризовать как работу с несколькими из следующих объектов:

ulst\_item Целевая последовательность (Target sequence).

ulst\_item Шаблон (Pattern).

ulst\_item Совпавший массив (Matched array).

ulst\_item Строка замены (Replacement string).

**Таблица 1.1 — Задание**

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Язык разметки** | **Извлекаемое содержимое** |
| 1 | HTML | заголовки,  абзацы,  неупорядоченные одноуровневые списки |
| 2 | Markdown | заголовки,  абзацы,  неупорядоченные одноуровневые списки |
| 3 | AsciiDoc | заголовки,  абзацы,  неупорядоченные одноуровневые списки |
| 4 | HTML | заголовки,  абзацы,  упорядоченные одноуровневые списки |
| 5 | Markdown | заголовки,  абзацы,  упорядоченные одноуровневые списки |
| 6 | AsciiDoc | заголовки,  абзацы,  упорядоченные одноуровневые списки |
| 7 | HTML | заголовки,  абзацы,  ссылки |
| 8 | Markdown | заголовки,  абзацы,  ссылки |
| 9 | AsciiDoc | заголовки,  абзацы,  ссылки |
| 10 | HTML | заголовки,  абзацы,  жирный текст |
| 11 | Markdown | заголовки,  абзацы,  жирный текст |
| 12 | AsciiDoc | заголовки,  абзацы,  жирный текст |
| 13 | HTML | заголовки,  абзацы,  текст курсивом |
| 14 | Markdown | заголовки,  абзацы,  текст курсивом |
| 15 | AsciiDoc | заголовки,  абзацы,  текст курсивом |

## Теория

### Язык разметки Markdown

Markdown — это легкий и простой язык разметки, который используется для создания структурированного текста.

#### Заголовки

# Заголовок первого уровня

## Заголовок второго уровня

...

###### Заголовок шестого уровня

#### Абзацы

Обычный текст в Markdown выглядит так же, как и в текстовом редакторе.

#### Неупорядоченные одноуровневые списки

Неупорядоченные (маркированные) списки используют звездочки, плюсы и дефисы (\*, + и -) в качестве маркеров списка. Эти три маркера взаимозаменяемы.

\* Пункт списка.

- Пункт списка.

+ Пункт списка.

#### Упорядоченные одноуровневые списки

В упорядоченных (нумерованных) списках в качестве маркеров списка используются обычные числа, за которыми следуют точки.

1. Первый пункт списка

2. Второй пункт списка

3. Третий пункт списка

#### Ссылки

Markdown поддерживает два стиля для создания ссылок: встроенный (inline) и ссылочный. Ссылки в стиле inline используют скобки сразу после текста ссылки.

Это [пример ссылки](http://example.com/).

#### Жирный текст (сильно акцентированный текст)

#### Используйте две звёздочки для \*\*сильного акцентирования\*\*.

#### Или, если хотите, \_\_используйте вместо этого два подчеркивания\_\_.

#### Текст курсивом (акцентированный текст)

\*Акцентированный текст\*.

\_Также акцентированный текст\_.

### Контейнеры

Контейнеры в C++ представляют собой классы, предназначенные для хранения и управления коллекциями объектов. Они предоставляют разработчикам мощные средства для работы с динамическими массивами данных, такими как векторы, списки, множества и ассоциативные массивы. Эти классы обеспечивают автоматическое управление памятью, быструю обработку и удобство использования.

#### Пример с использованием std::list

std::list — это контейнер, который поддерживает вставку и удаление элементов из любого места контейнера за постоянное время. Быстрый произвольный доступ не поддерживается. Обычно он реализуется как двусвязный список. По сравнению с std::forward\_list этот контейнер обеспечивает возможность двунаправленной итерации, будучи менее эффективным по пространству.

Добавление, удаление и перемещение элементов в списке или между несколькими списками не делает итераторы или ссылки недействительными. Итератор становится недействительным только при удалении соответствующего элемента.

**Source code:**

#include <algorithm>

#include <iostream>

#include <list>

int main()

{

// Создать список, содержащий целые числа

std::list<int> lst = { 7, 5, 16, 8 };

// Добавить целое число в начало списка

lst.push\_front(25);

// Добавить целое число в конец списка

lst.push\_back(13);

// Вставьте целое число перед 16, выполнив поиск

auto it = std::find(lst.begin(), lst.end(), 16);

if (it != lst.end())

lst.insert(it, 42);

// Вывести список

std::cout << "lst = { ";

for (int n : lst)

{

std::cout << n << ", ";

}

std::cout << "};\n";

return 0;

}

**Output:**

lst = { 25, 7, 5, 42, 16, 8, 13, };

### Итераторы

Итераторы — это обобщение указателей, позволяющее программе C++ единообразно работать с различными структурами данных (например, контейнерами и диапазонами (начиная с C++20)).

Поскольку итераторы являются абстракцией указателей, их семантика является обобщением большей части семантики указателей в C++. Это гарантирует, что каждый шаблон функции, принимающий итераторы, будет работать также и с обычными указателями.

#### Пример обхода std::ist

**Source code:**

#include <iostream>

#include <list>

int main()

{

std::list<int> lst;

for (int count = 0; count < 6; ++count)

{

lst.push\_back(count);

}

std::list<int>::const\_iterator it; // объявить итератор

it = lst.cbegin(); // присвоить it на начало списка

while (it != lst.cend()) // пока it не дошел до конца

{

std::cout << \*it << ' '; // вывести значение элемента, на который указывает it

++it; // и перейти к следующему элементу

}

std::cout << '\n';

return 0;

}

**Output:**

0 1 2 3 4 5

### Регулярные выражения

В стандартной библиотеке C++ есть библиотека регулярных выражений. Библиотека регулярных выражений предоставляет класс, представляющий регулярные выражения, которые являются своего рода мини-языком, используемым для сопоставления шаблонов в строках.

Поскольку регулярные выражения медленнее ручной проверки строк, их следует использовать только в том случае, если производительность (время компиляции и время выполнения) не имеет значения или ручная проверка слишком обременительна.

#### Пример с регулярными выражениями

**Source code:**

#include <iostream>

#include <iterator>

#include <regex>

#include <string>

int main()

{

std::string s = "Some people, when confronted with a problem, think "

"\"I know, I'll use regular expressions.\" "

"Now they have two problems.";

std::regex self\_regex("REGULAR EXPRESSIONS",

std::regex\_constants::ECMAScript | std::regex\_constants::icase);

if (std::regex\_search(s, self\_regex))

std::cout << "Text contains the phrase 'regular expressions'\n";

std::regex word\_regex("(\\w+)");

auto words\_begin =

std::sregex\_iterator(s.begin(), s.end(), word\_regex);

auto words\_end = std::sregex\_iterator();

std::cout << "Found "

<< std::distance(words\_begin, words\_end)

<< " words\n";

const int N = 6;

std::cout << "Words longer than " << N << " characters:\n";

for (std::sregex\_iterator i = words\_begin; i != words\_end; ++i)

{

std::smatch match = \*i;

std::string match\_str = match.str();

if (match\_str.size() > N)

std::cout << " " << match\_str << '\n';

}

std::regex long\_word\_regex("(\\w{7,})");

std::string new\_s = std::regex\_replace(s, long\_word\_regex, "[$&]");

std::cout << new\_s << '\n';

return 0;

}

**Output:**

Text contains the phrase 'regular expressions'

Found 20 words

Words longer than 6 characters:

confronted

problem

regular

expressions

problems

Some people, when [confronted] with a [problem], think "I know, I'll use [regular] [expressions]." Now they have two [problems].