Лямбда-функции — конспект темы

Компаратор для сортировки

Компаратор — функция сравнения двух объектов в C++.

Лямбда-функции, или **безымянные функции** — синтаксический сахар. Они делают код понятнее и удобнее в использовании.

Вот как можно переписать метод `FindTopDocuments` с лямбда-функцией в качестве компаратора:

```
vector<Document> FindTopDocuments(const string& raw_query) const {
    const set<string> query_words = ParseQuery(raw_query);
    auto matched_documents = FindAllDocuments(query_words);

    sort(matched_documents.begin(), matched_documents.end(),
         [](const Document& lhs, const Document& rhs) {  // Это лямбда-функция
             return lhs.relevance > rhs.relevance;
         });
    if (matched_documents.size() > MAX_RESULT_DOCUMENT_COUNT) {
        matched_documents.resize(MAX_RESULT_DOCUMENT_COUNT);
    }
    return matched_documents;
}
```

Квадратные скобки `[]` сообщают компилятору, что это лямбда-функция. В круглых скобках идёт список параметров, как у обычной функции. После в фигурных скобках указано тело.

У лямбда-функции обычно не указывают тип возвращаемого значения. Компилятор сам выводит его по типу выражения в команде `return`.

Захват переменных по значению

Чтобы использовать внешнюю переменную внутри лямбда-функции, переменную надо явно захватить — указать её имя в квадратных скобках перед параметрами лямбда-функции:

```
int main() {
    int threshold;
    cin >> threshold;

    const vector<int> v = {1, 3, 5, 2, 6, 7, 10, 2, 3};
    cout << count_if(begin(v), end(v), [threshold](int x) { return x > threshold; }) << endl;
//                                      ^ вот здесь теперь указана переменная threshold
}
```

Из переменной, захваченной таким образом, можно только читать:

```
int main() {
    int x = 0;
    auto lambda = [x]() { ++x; };
    lambda(); // да, лямбда-функции можно присваивать переменным, а потом вызывать как обычные функции
}
```

Захват переменных по ссылке

Все локальные переменные, захваченные лямбда-функцией, копируются внутрь неё. При этом происходит глубокое копирование. Это замедляет программу. Поэтому используйте захват переменных по ссылке:

```
int CountQueryWords(const vector<string>& query, const set<string>& minus_words) {
    return count_if(
        query.begin(), query.end(),
        [&minus_words](const string& w) { // Обратите внимание на символ & перед minus_words
            return minus_words.count(w) == 0;
        }
    );
}
```

Символ `&` перед именем переменной говорит компилятору, чтобы вместо глубокого копирования переменной `minus_words` внутрь лямбды он сохранил только ссылку на неё.

Если лямбда-функция захватывает несколько переменных, способ захвата нужно прописать для каждой переменной отдельно.

Инструкция по эксплуатации

- Лямбда-функции должны быть лаконичными.
- Лучшее место для лямбда-функций — предикаты в алгоритмах `sort`, `count_if`.
- Нужно контролировать захватываемые локальные переменные.
- Переменные простых типов надо захватывать по значению, а переменные сложных типов — по ссылке.

Учёт минус-слов

Минус-слова исключают из результатов поиска документы, содержащие такие слова. Стоп-слово в запросе не нужно учитывать при поиске, даже если оно с минусом. Слова без минуса пусть называются **плюс-словами**.

Ускоряем поиск документов

Инвертированный индекс — структура данных, где каждому слову соответствует множество документов, в которых это слово встречается. Инвертированный индекс используют для эффективного поиска текста среди большого количества документов.

Ранжирование по TF-IDF

TF-IDF — мера важности слова в документе из коллекции документов. Выступает критерием релевантности документа поисковому запросу.

IDF (inverse document frequency — обратная частота документа) — величина, которая позволяет оценить полезность слов. Чем в большем количестве документов есть слово, тем ниже IDF.

Рассчитывают IDF так:

- Количество всех документов делят на количество тех, где встречается слово.
- К результату деления применяют функцию `log` из библиотеки `<cmath>`.

TF (term frequency — частота слова) — для конкретного слова и конкретного документа это доля, которую данное слово занимает среди всех.

Чтобы посчитать релевантность документа запросу:

- вычисляется IDF каждого слова в запросе,
- вычисляется TF каждого слова запроса в документе,
- IDF каждого слова запроса умножается на TF этого слова в этом документе,
- все произведения IDF и TF в документе суммируются.

Чем больше сумма, тем релевантнее документ.