Глубокое копирование, ссылки, константность — конспект темы

Глубокое копирование

При глубоком копировании изменения, внесённые в копию, не отражаются в оригинале.

Во время вызова функции её аргументы копируются, и функция работает с копиями. Поэтому изменение параметров внутри функции не отражается на передаваемых значениях:

```
#include <iostream>

using namespace std;

void Countdown(int n) {
    while (n >= 0) {
        cout << n << ' ';
        --n;
    }
    cout << endl;
}

int main() {
    int n = 5;
    Countdown(n); // Выведет 5 4 3 2 1 0
    cout << "n="s << n << endl; // Выведен n=5
}
```

Передача параметра по значению — передача параметров в функцию, при которой происходит копирование аргумента. В C++ по умолчанию все параметры функций передаются по значению.

Функция `SplitIntoWords` принимает параметр `text` по значению:

```
vector<string> SplitIntoWords(string text) {
    // ...
}

int main() {
    ...
    vector<string> words = SplitIntoWords(query);
    ...
}
```

Строка `SplitIntoWords` получит копию строки `query`. Копия будет доступна внутри функции под именем `text`.

Ссылки

При объявлении переменной `string word;` создаётся новая пустая строка. Если объявить строку и задать ей значение другой строки, например `string word = value;`, произойдёт глубокое копирование.

Если поставить символ амперсанд `&` после типа, вы объявите ссылку на этот тип. `string&` — ссылка на строку. Ссылка не хранит само значение.

Инициализация `string& word = value;` — не копирование данных, а создание новой связки ссылки и объекта:

```
string search_query = "how to learn c+"s;
// Ссылка search_query_ref — ещё одно имя для доступа к переменной search_query
string& search_query_ref = search_query;  // ref — сокращение от reference (ссылка)

cout << search_query << endl;     // выведет how to learn c+

// добавляем символ '+' к строке search_query, используя ссылку search_query_ref
search_query_ref += '+';

cout << search_query << endl;     // выведет how to learn c++
cout << search_query_ref << endl; // выведет how to learn c++
```

Нюансы:

1. Ссылку нельзя создать без инициализации.
2. Ссылку нельзя «перевесить» на другой объект.

Часто параметры передают по ссылке, когда функция должна изменить их значение:

```
// Удаляет последнее слово из вектора слов
void RemoveLastWord(vector<string>& words) {
    if (!words.empty()) {
        // Метод pop_back удаляет последний элемент вектора
        // и тем самым противоположен методу push_back.
        // Этот метод можно безопасно вызывать только у непустого вектора
        words.pop_back();
    }
}
```

Константные ссылки

Константность делает объект или ссылку неизменяемыми и защищает код от вторжения и случайного изменения. Константные ссылки используются, чтобы:

- передать тяжёлый объект в функцию, не копируя и не позволяя менять его внутри функции;
- перебрать такие объекты в цикле `for` с теми же условиями.

Ссылки `const string&` или `const vector<int>&` — константные.

Константные ссылки создают новое имя, по которому можно обратиться к существующему объекту и не разрешают использовать это имя для изменения объекта:

```
int main() {
    string pet_name = "Leo"s;

    const string& const_pet_name_ref = pet_name;
    // Имя const_pet_name_ref можно использовать для чтения строки
    cout << const_pet_name_ref << "'s name size is "s << const_pet_name_ref.size() << endl;
    // Выведет Leo's name size is 3

    // Следующая строка не скомпилируется, так как имя const_pet_name_ref
    // нельзя использовать для модификации объекта
    // const_pet_name_ref += "pold"s;
```

```
    // А вот саму переменную pet_name и ссылающуюся на неё ссылку pet_name_ref можно
    // использовать для изменения строки pet_name.
    pet_name += "po"s;
    string& pet_name_ref = pet_name;
    pet_name_ref += "ld"s;

    cout << pet_name_ref << "'s name size is "s << pet_name_ref.size() << endl;
    // Выведет Leopold's name size is 7
}
```

Хитрости константных ссылок

`bool`, `char`, `int` и другие простые типы лучше передавать по значению. Строки и всевозможные контейнеры — по константной ссылке.

Нельзя написать `int& ref = 50;`, так как неконстантная ссылка не может быть привязана к неконстантному объекту, а литералы — это константы. Выскочит ошибка компиляции.

Нельзя привязать ссылку на строку к строковому литералу `""s` — случится ошибка компиляции:

```
#include <string>

using namespace std;

void AddLocation(string& query) {
    query += " near central station"s;
}

int main() {
    AddLocation("white cat"s);
}
```

Если пытаться передать в `UpdateSearchQuery` строчку, только что полученную из другой функции, тоже увидите ошибку компиляции:

```
#include <iostream>
#include <string>

using namespace std;

void UpdateSearchQuery(string& query) {
    query += " free download"s;
}

string ReadSearchQuery() {
    string query;
    getline(cin, query);
    return query;
}

int main() {
    UpdateSearchQuery(ReadSearchQuery()); // error: cannot bind non-const lvalue reference of type 'std::string& ...
}
```

Невозможно поменять объект, у которого нет названия, ведь его не положили в переменную.

С константными ссылками таких проблем не бывает. Они связаны не с изменением объекта, а с передачей без копирования:

```
#include <iostream>
#include <string>

using namespace std;

string ReadSearchQuery() {
    string query;
    getline(cin, query);
    return query;
}

// Возвращает количество слов в тексте.
// Словом считается последовательность непробельных символов
int CountWords(const string& text) {
    int word_count = 0;
    char prev_char = ' ';
    for (char c : text) {
        if (prev_char == ' ' && c != ' ') {
            // Новое слово начинается на границе пробельного и непробельного символа
            ++word_count;
        }
        prev_char = c;
    }
    return word_count;
}

int main() {
    cout << CountWords("  This   text   contains  five  words  "s) << endl;
    cout << CountWords("I"s + " love "s + "C++") << endl;
    cout << CountWords(ReadSearchQuery()) << endl;
}
```

Константное всё

Тип `const string&` читается как «ссылка на `const string`», где `const string` — самостоятельный тип, неизменяемая строка.

Создание переменной типа `const string`:

```
const string query = ReadString();

// Так ОК: для константной строки можно и узнать размер, и обратиться к символу
cout << query[query.size() - 2] << endl;

// Так нельзя: += изменяет строку
query += " free download"s;
```

После создания константной строки изменить её значение нельзя.

Переменная сохраняет свой тип с момент объявления и до выхода из блока, в котором была объявлена:

```
int main() {
    // query объявлена неконстантной, чтобы в неё можно было считать значение функцией getline
    string query;
```

```
    getline(cin, query);
    // Начиная с этого места, мы хотим запретить изменять содержимое query

    int num_words = 0;
    char prev_char = ' ';
    for (char c : query) {
        if (c != ' ' && prev_char == ' ') {
            ++num_words;
        }
        prev_char = c;
    }

    cout << '\'' << query << "' contains "s << num_words << " words"s << endl;
}
```

Если во фрагменте кода нужно запретить изменять неконстантную переменную, фрагмент выносят в отдельную функцию:

```
int CountWords(const string& text) {
    int num_words = 0;
    char prev_char = ' ';
    for (char c : text) {
        if (c != ' ' && prev_char == ' ') {
            ++num_words;
        }
        prev_char = c;
    }
    return num_words;
}

int main() {
    string query;
    getline(cin, query);
    // Код подсчёта слов вынесен в отдельную функцию, которая не может изменить query
    cout << '\'' << query << "' contains "s << CountWords(query) << " words"s << endl;
}
```

Константность помогает защитить переменные от случайных модификаций. Чтобы проверять, менялась ли переменная, было проще:

1. Делайте переменную константной, если она не должна меняться после создания.
2. Делайте функции, в том числе `main`, компактными: вынесите их самостоятельные части в другие функции.
3. Помечайте имена аргументов функций, передаваемых по неконстантным ссылкам. Так принято в некоторых командах.

Ядро поисковой системы

Стоп-слова — слова (союзы и предлоги), которые встречаются во многих документах и обычно не несут смысловой нагрузки, поэтому их нужно исключить из поиска. Для их хранения подойдёт контейнер `set`.

Отдельный документ в поисковой системе будет представлен контейнером `vector<string>`, а база данных — `vector<vector<string>>`.

```
// Создадим базу данных, в которой будем хранить документы
// Каждый документ - вектор слов документа
```

```
vector<vector<string>> documents;
// Переменная document - это тот документ, который нужно добавить в базу данных
string document = "a strange brown creature was seen in the box of oranges"s;
// Подготовим множество стоп-слов
const set<string> stop_words = {"a"s, "the"s, "on"s, "end"s};

AddDocument(documents, stop_words, document);
```

Внутри функции `AddDocument`:

- документ разбит на слова;
- стоп-слова исключены из рассмотрения;
- оставшиеся слова добавятся в базу данных, которую функция `AddDocument` принимает по ссылке.

Код добавления документов:

```
vector<string> SplitIntoWordsNoStop(const string& text, const set<string>& stop_words) {
    vector<string> words;
    // проходим по всем словам из текста и проверяем, что их нет в списке стоп-слов
    for (const string& word : SplitIntoWords(text)) {
        if (stop_words.count(word) == 0) {
            words.push_back(word);
        }
    }
    // вернём результат без стоп-слов
    return words;
}

void AddDocument(vector<vector<string>>& documents, const set<string>& stop_words,
                 const string& document) {
    const vector<string> words = SplitIntoWordsNoStop(document, stop_words);
    documents.push_back(words);
}
```