**Глава 4. Представление типов данных в программе**

**4.1 Типы данных**

Действия в программе происходят данными, которые в программировании называются *операндами*. Каждый операнд может быть как переменной, так и постоянной величиной – константой, а также выражением или функцией.

В программировании принято говорить, что каждый операнд имеет свой *тип данных*. Данные разных типов по-разному хранятся в памяти компьютера, по-разному обрабатываются, отличаются разным набором операций и функций, разрешенных для каждого типа, а также отличаются множеством значений, разрешенных для величин каждого типа.

Типы данных делятся на *основные* и *составные* (рисунок 4.1). Существует шесть основных типов данных, на основе которых можно создавать составные типы.



Рисунок 4.1 – Типы данных

Основные типы данных представляют собой базовые типы, предоставляемые языком программирования (рисунок 4.2). Они включают в себя следующие:

Целочисленный тип данных – int – позволяет хранить целые числа без десятичной части [14].

Символьный тип – char. Он предназначен для хранения отдельных символов, таких как буквы, цифры или специальные символы.

Логический тип – bool. Он может принимать два значения: true или false. Логические типы данных используются для представления логических состояний и условий.

Вещественные типы данных: например, float и double. Они используются для хранения чисел с плавающей запятой, которые могут иметь десятичную часть.

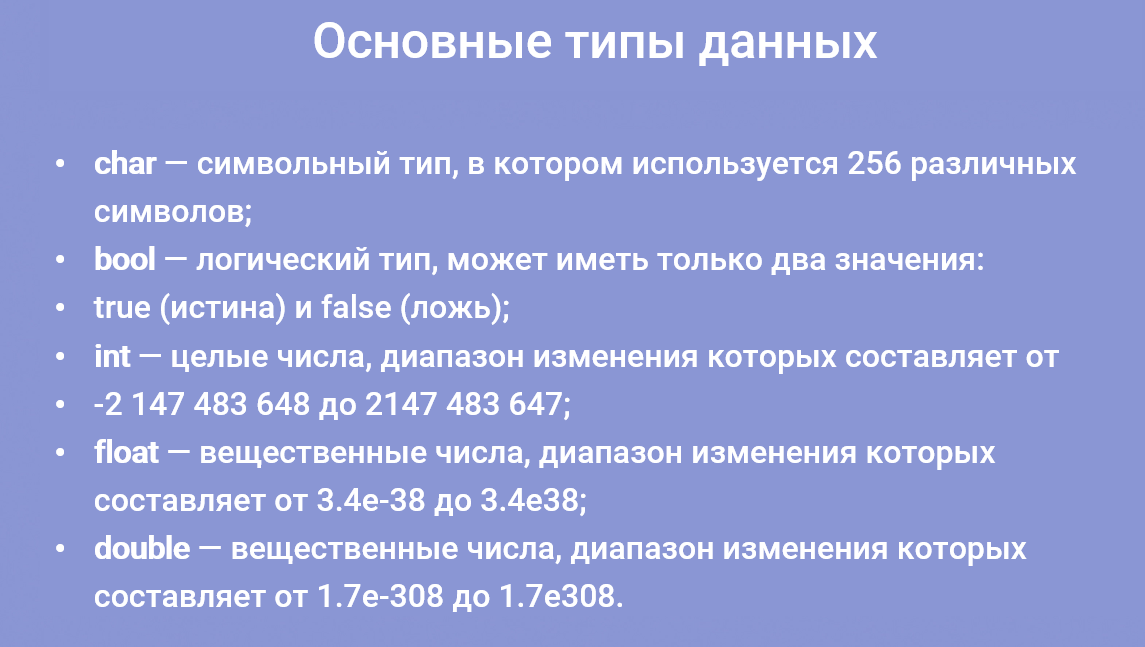


Рисунок 4.2 – Основные типы данных

Кроме основных типов данных, существуют также составные типы данных, включающие массивы, функции, структуры, ссылки, указатели, классы, объединения и перечисления. Они позволяют объединять переменные и элементы в более сложные структуры данных [15].

**4.2 Объявление переменных**

Для использования переменных в программировании необходимо их объявить. Объявление переменных выполняется с помощью определенного синтаксиса (рисунок 4.3). Сначала указывается тип переменной, а затем ее наименование. Кроме того, при объявлении можно сразу же инициализировать переменную, то есть присвоить ей начальное значение.

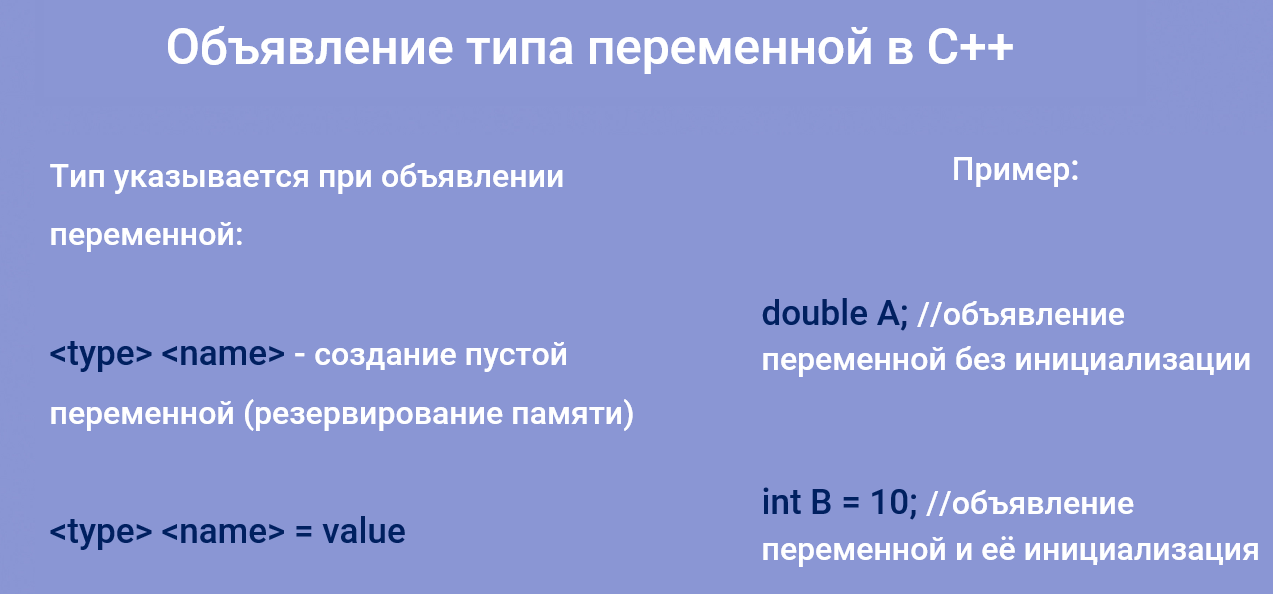


Рисунок 4.3 – Объявление типа переменной

При использовании переменных в программировании необходимо объявлять их до того, как они будут использованы.Это обусловлено тем, что компилятор языка обрабатывает код программы сверху вниз.

**4.3 Спецификаторы типов**

В языке программирования C++ существуют *спецификаторы типов*, которые позволяют изменять диапазон значений стандартных числовых типов данных.

signed: спецификатор указывает на то, что переменная может хранить отрицательные значения (со знаком –) и положительные значения (со знаком+). Данный спецификатор указан по умолчанию, то есть его не нужно прописывать специально.

unsigned: спецификатор указывает, что переменная может хранить только положительные значения или ноль, то есть она не имеет знака.

short: спецификатор уменьшает диапазон значений в два раза по сравнению с обычным типом.

long: спецификатор увеличивает диапазон значений в два раза по сравнению с обычным типом благодаря увеличенному резервированию памяти.

Данные спецификаторы можно комбинировать и получать необходимые переменные (рисунок 4.4) [17].

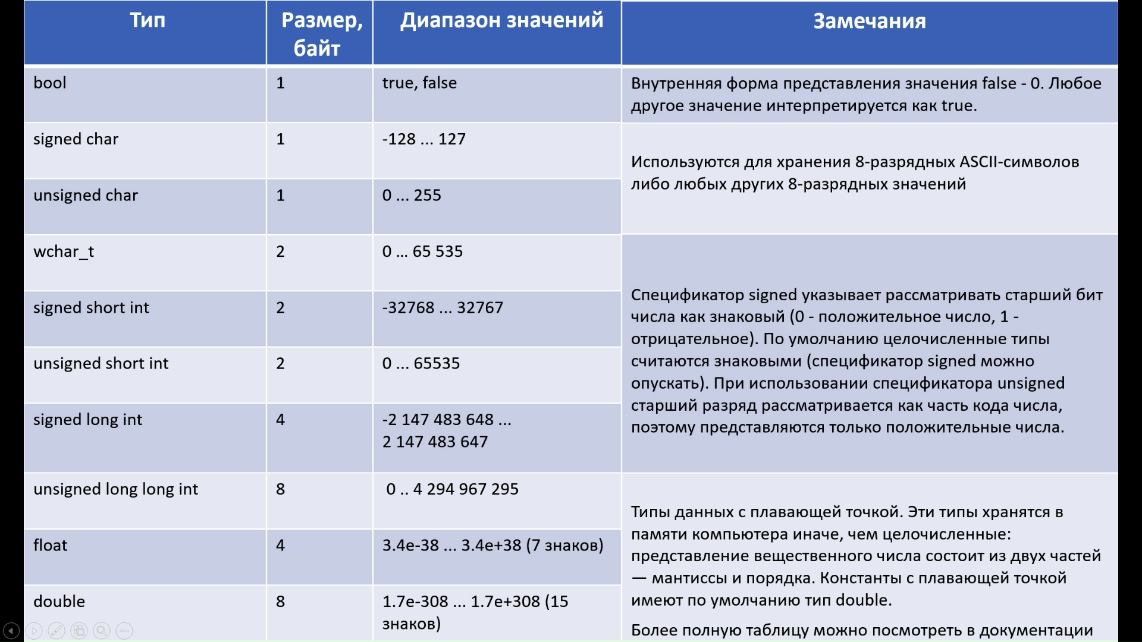


Рисунок 4.4 – Спецификаторы типов данных

**4.4 Константы**

При работе с переменными в программировании, помимо изменяемых значений, существуют также неизменяемые величины, которые называются *константами*.

Константы имеют следующие особенности:

1. Они обладают фиксированным значением, которое не может быть изменено во время выполнения программы. Значение инициализируются только один раз.

2. Константы могут использоваться в операциях и выражениях, но результат таких операций нельзя сохранить в области памяти, где хранится значение константы.

3. Константы часто используются для задания фиксированных значений, таких как числовые значения, значения символов, логические значения и другие неизменяемые данные.

При объявлении констант используется ключевое слово *const* вместе с синтаксисом, аналогичным объявлению обычных переменных. Важной особенностью констант является обязательная инициализация при их объявлении. Пример объявления константы представлен на рисунке 4.5

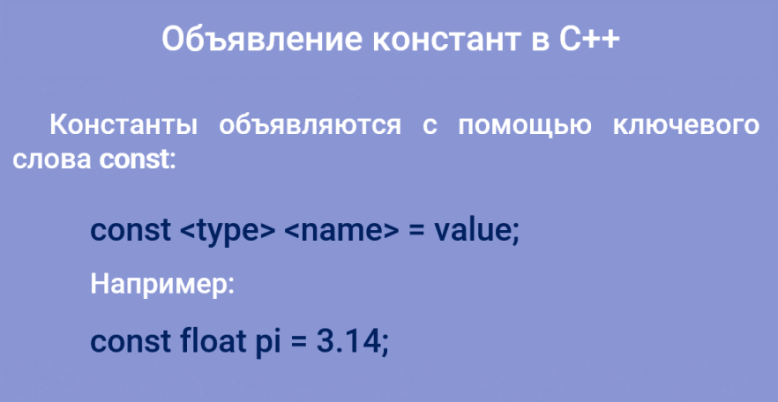


Рисунок 4.5 – Объявление константы

**4.5 Указатели и ссылки**

*Указатель* - это переменная, которая содержит адрес в памяти другой переменной. Он позволяет получить доступ к значению этой переменной по ее адресу.

*Ссылка* - это псевдоним для переменной. Она предоставляет ей альтернативное имя, по которому можно обращаться к тому же значению. Ссылка всегда должна быть инициализирована при объявлении и не может быть переопределена.

Для объявления указателя используется "\*", а также синтаксис переменной. Для ссылки используется "&" и синтаксис переменной. Данные символы используются не только для объявления, но и прямой работы с оперативной памятью. "&" нужен для взятия адреса переменной, а "\*" для разыменования и разадресации указателей (рисунок 4.6).

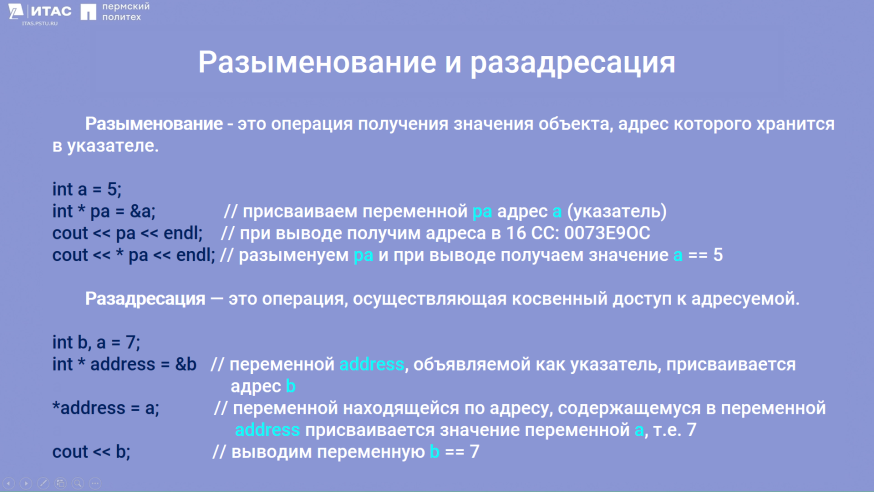


Рисунок 4.6 – Понятия разыменования и разадресации

Дополнительные примеры, демонстрирующие принцип работы указателей и ссылок, представлены на рисунках 4.7, 4.8 и 4.9.

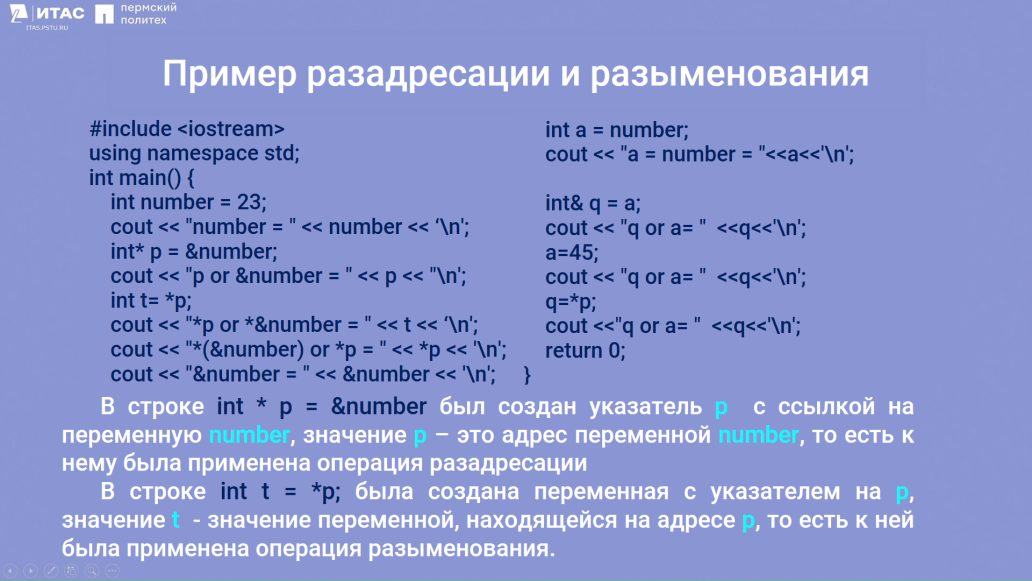


Рисунок 4.7 – Пример использования разадресации и разыменования

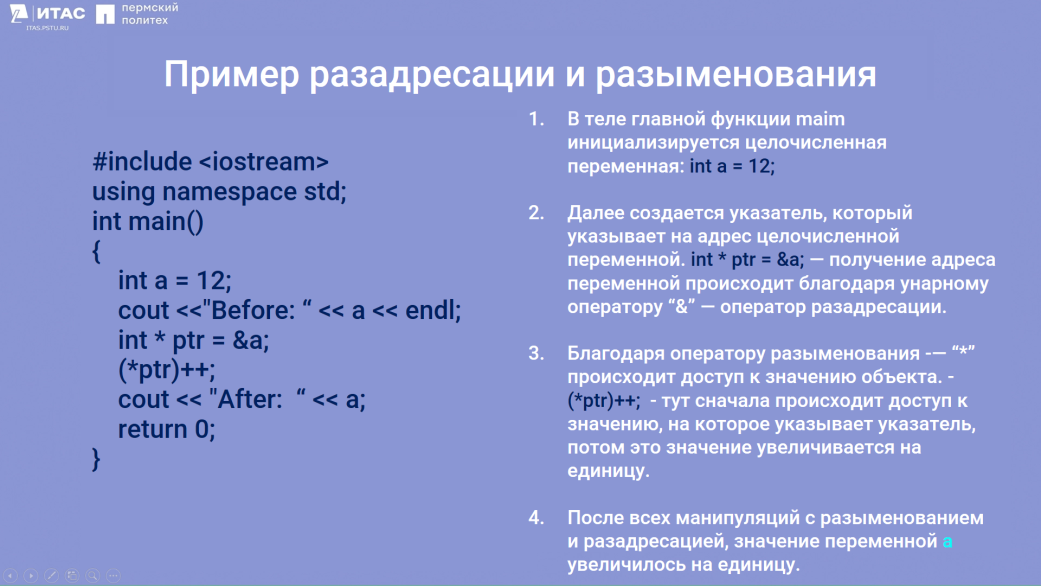


Рисунок 4.8 – Пример использования разадресации и разыменования

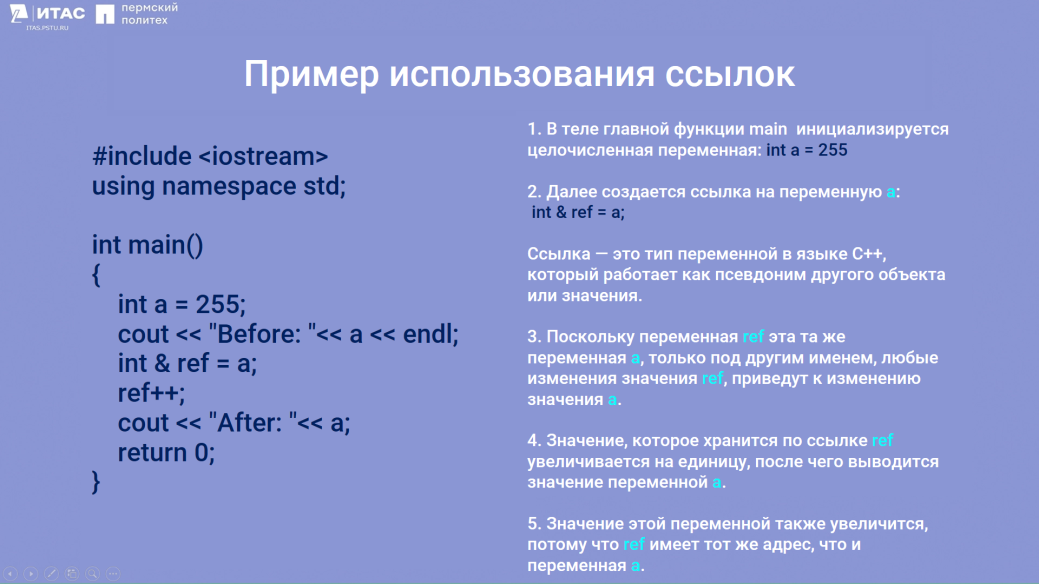


Рисунок 4.9 – Пример использования ссылок

**4.6 Классы памяти**

В языке С++ существует понятие *класс памяти*. Но для понимания этой информации необходимо ввести некоторые понятия:

*Время жизни* – это период выполнения программы, в течение которого существует переменная или функция. Время жизни может быть *постоянным*(в течение всей программы) и *временным* (в течение выполнения блока, который заключен в фигурные скобки).

*Область видимости* - часть текста программы, из которой доступна область памяти, которая связана с данным идентификатором. Область действий делится на локальную и глобальную.

*Локальная область видимости* - то что включено в фигурные скобки. Переменная действует от места ее объявления, до закрытия фигурных скобок.

*Глобальная область видимос*ти - то что находится за пределами каких-либо фигурных скобок. Определена вне блока программы. Переменная действует от места объявления до конца программы.

Пример, демонстрирующий принцип работы время жизни переменной и её области видимости, представлен на рисунке 4.10.

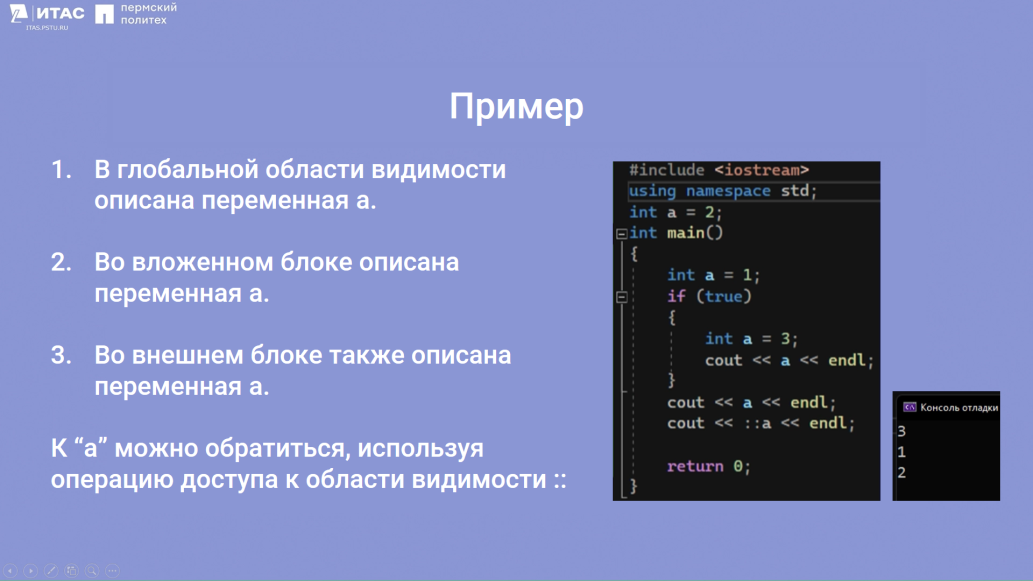


Рисунок 4.10 – Время жизни и область действия переменной

На примере видно, как работает область видимости. В первом случае выводится 3, потому что данное число находится в области видимости. Далее выводится 1, так как 3 уже вышла из области видимости блока и время ее жизни прекратилось. С помощью двух двоеточий происходит обращение к глобальной области видимости, поэтому на консоли 2.

Класс памяти - определяет время жизни, область видимости, а также место в оперативной памяти. Если не указывать класс памяти явно, то компилятор укажет его самостоятельно. Существует 4 типа классов памяти: auto, extern, static, register.

*Auto* (автоматическая переменная):

1. Время жизни этой переменной: от описания до конца блока.
2. Не используется для глобальных переменных.
3. При выходе из блока память, где находится описание переменной, освобождается.
4. Для локальных переменных auto применяется по умолчанию, поэтому задавать его явно смысла не имеет.

*Register:*

Данный спецификатор повторяет auto во всем, но переменные с таким классом памяти помещаются на прямую в регистры процессора. Это может сильно ускорить работу программы, если данная переменная будет использоваться очень часто при вычислениях.

*Static* (сегмент данных) – статическая переменная:

1. Инициализация – один раз при первом выполнении оператора,  
который содержит определение переменных.

2. Могут быть глобальными – видны во всех файлах программы и локальными – видны только в файле‚ в котором описаны. Если задать переменную static:

Вне блока: переменная будет глобальной (время жизни – постоянное (вся программа)).

В блоке: переменная будет локальной (время жизни – до конца программы).

С помощью модификатора static можно описать статические поля и методы класса. Их можно рассматривать как глобальные переменные или функции, доступные только в пределах области класса. Пример, демонстрирующий разницу между такими классами памяти как auto и static, представлен на рисунке 4.11.

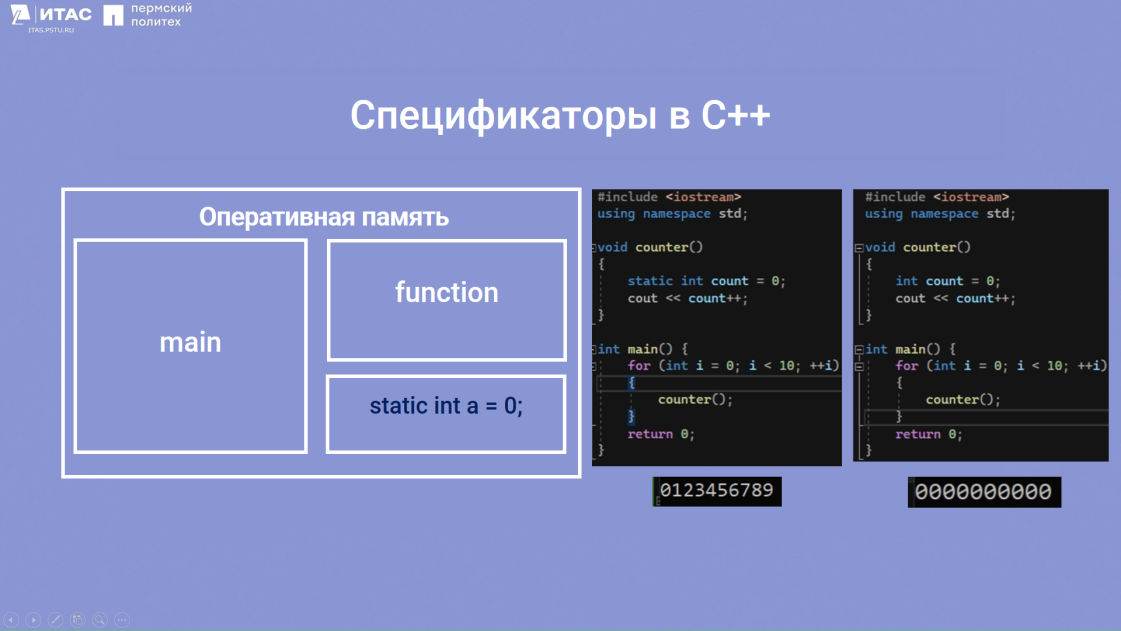


Рисунок 4.11­­­­– Отличие классов памятиauto и static

*Extern*: объявление самой глобальной переменной (связывает переменную с конкретной областью памяти и описывает свойства переменной).

1. Переменная определяется в любом месте программы, может даже в другом файле.
2. Используется для создания переменных, доступных во всех файлах программы.
3. Если переменная при описании инициализируется (начальным значением), то extern игнорируется.

Пример, в котором используется extern, представлен на рисунке 4.12.

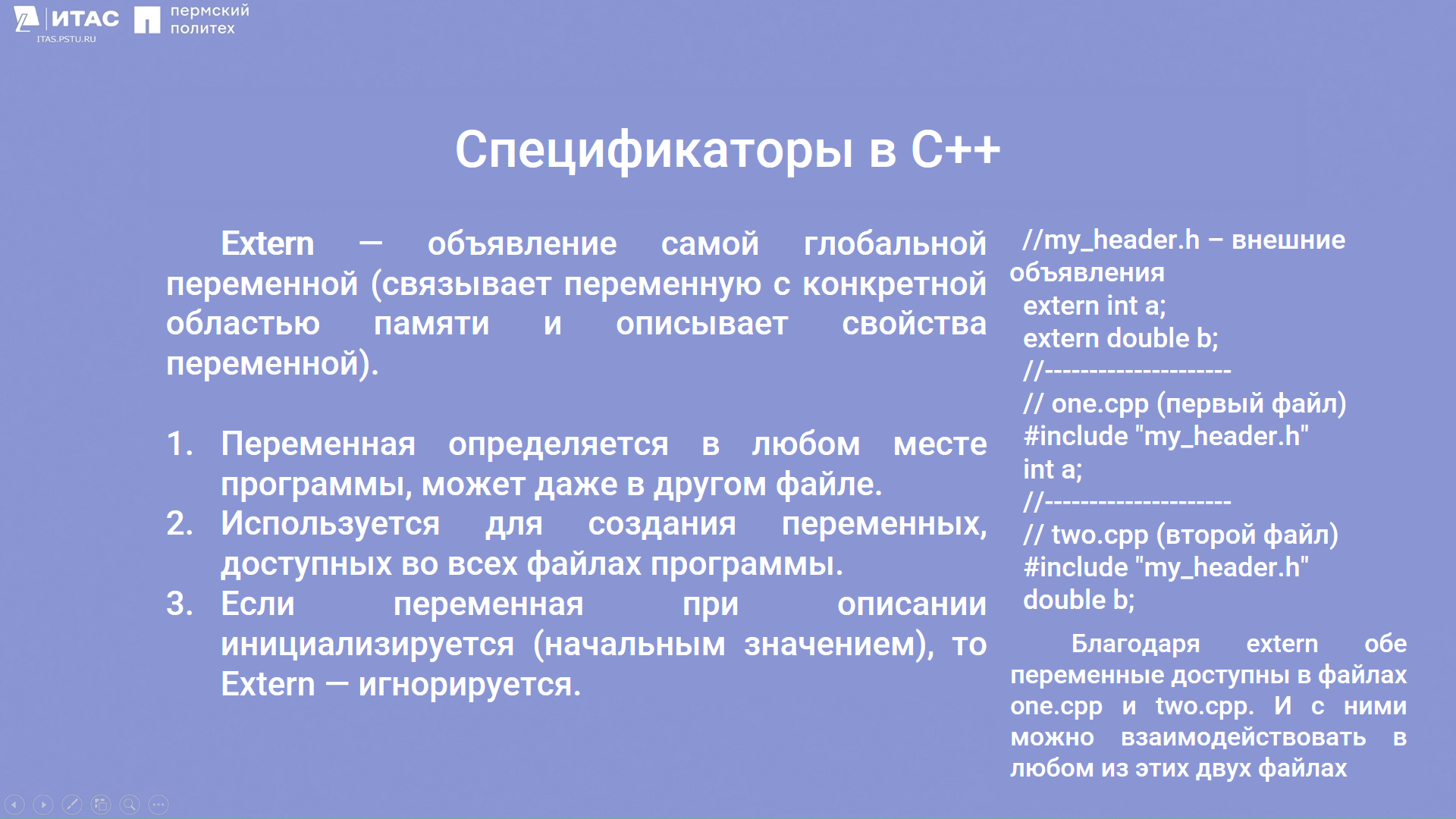


Рисунок 4.12 – Пример работы с классом памяти extern

Благодаря extern обе переменные доступны в файлах one.cpp и two.cpp. И с ними можно взаимодействовать.