# Оператор адреса (&)

При выполнении инициализации переменной, ей автоматически присваивается свободный адрес памяти, и, любое значение, которое мы присваиваем переменной, сохраняется в этом адресе памяти.

**Оператор адреса** (**&**) позволяет узнать, какой адрес памяти присвоен определённой переменной. Всё довольно просто:

```cpp
#include <iostream>

int main()
{
    int a = 7;
    std::cout << a << '\n'; // выводим значение переменной a
    std::cout << &a << '\n'; // выводим адрес памяти переменной a

    return 0;
}
```

Результат на моём компьютере:

7

0046FCF0

# Оператор разыменования (*)

**Оператор разыменования** (*) позволяет получить значение по указанному адресу:

```
3  int main()
4  {
5      int a = 7;
6      std::cout << a << '\n'; // выводим значение переменной a
7      std::cout << &a << '\n'; // выводим адрес переменной a
8      std::cout << *&a << '\n'; /// выводим значение ячейки памяти переменной a
9
10     return 0;
11 }
```

Результат на моём компьютере:
7
0046FCF0
7

# Указатели

- **Указатель** — это переменная, значением которой является адрес (ячейка) памяти. Указатели объявляются точно так же, как и обычные переменные, только со звёздочкой между типом данных и идентификатором:

```
1 int *iPtr; // указатель на значение типа int
2 double *dPtr; // указатель на значение типа double
3 int *iPtr5, *iPtr6; // объявляем два указателя для переменных типа int
```

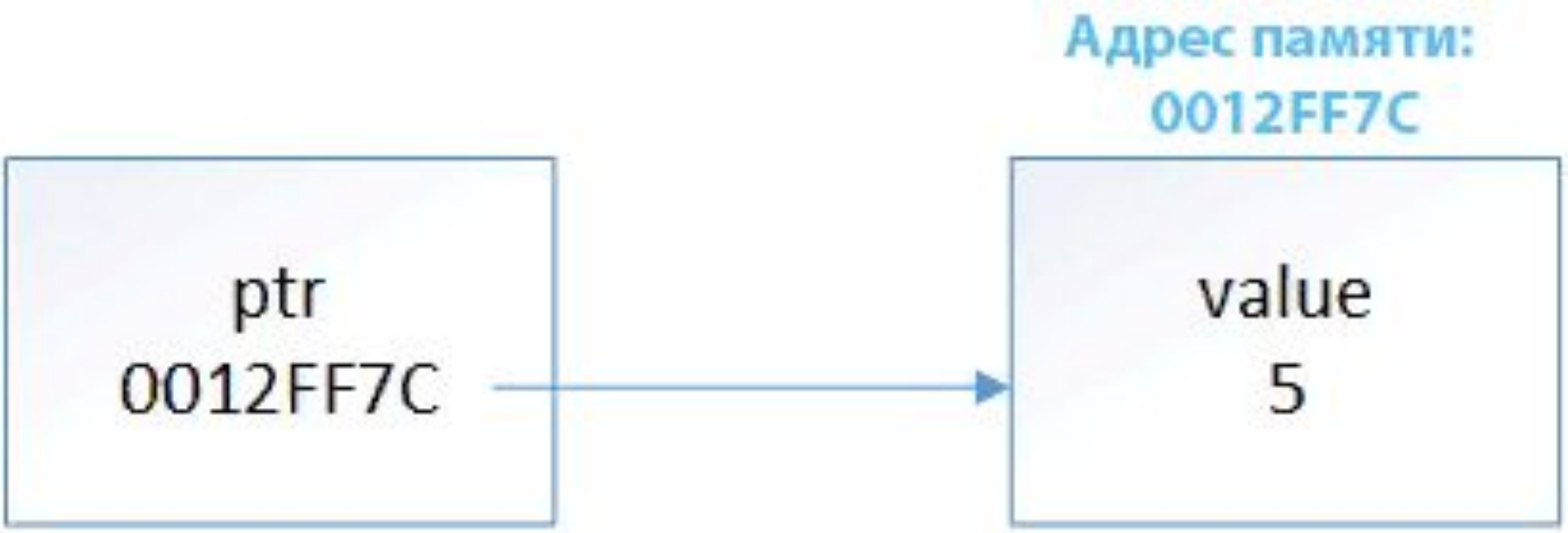
- Как и обычные переменные, указатели не инициализируются при объявлении. Содержимым неинициализированного указателя является обычный мусор

# Присваивание значений указателю

- Поскольку указатели содержат только адреса, то при присваивании указателю значения — это значение должно быть адресом. Для получения адреса переменной используется оператор адреса:

```
1 int value = 5;
2 int *ptr = &value; // инициализируем ptr адресом значения переменной
```

Приведенное выше можно проиллюстрировать следующим образом




Вот почему указатели имеют такое имя: ptr содержит адрес значения переменной value, и, можно сказать, ptr *указывает* на это значение.

- Ещё очень часто можно увидеть следующее:

```cpp
#include <iostream>

int main()
{
    int value = 5;
    int *ptr = &value; // инициализируем ptr адресом значения переменной

    std::cout << &value << '\n'; // выводим адрес значения переменной value
    std::cout << ptr << '\n'; // выводим адрес, который хранит ptr

    return 0;
}
```

Результат на моём компьютере:
003AFCD4
003AFCD4

Следующее не является допустимым:

```
1 int *ptr = 7;
```

Это связано с тем, что указатели могут содержать только адреса, а целочисленный литерал 7 не имеет адреса памяти. C++ также не позволит вам напрямую присваивать адреса памяти указателю:

```
1 double *dPtr = 0x0012FF7C;
```

# Разыменование указателей

- Как только у нас есть указатель, указывающий на что-либо, мы можем его разыменовать, чтобы получить значение, на которое он указывает. Разыменованный указатель — это содержимое ячейки памяти, на которую он указывает:

```cpp
int main()
{
    int value = 5;
    std::cout << &value << std::endl; // выводим адрес value
    std::cout << value << std::endl; // выводим содержимое value

    int *ptr = &value; // ptr указывает на value
    std::cout << ptr << std::endl; // выводим адрес, который хранится в ptr, т.е. &value
    std::cout << *ptr << std::endl; // разыменовываем ptr (получаем значение на которое указывает ptr)

    return 0;
}
```

Результат:

0034FD90

5

0034FD90

5

- Вот почему указатели должны иметь тип данных. Без типа указатель не знал бы, как интерпретировать содержимое, на которое он указывает (при разыменовании). Также, поэтому и должны совпадать тип указателя с типом переменной. Если они не совпадают, то указатель при разыменовании может неправильно интерпретировать биты (например, вместо типа double использовать тип int).

- Одному указателю можно присваивать разные значения:

```
1  int value1 = 5;
2  int value2 = 7;
3
4  int *ptr;
5
6  ptr = &value1; // ptr указывает на value1
7  std::cout << *ptr; // выведется 5
8
9  ptr = &value2; // ptr теперь указывает на value2
10 std::cout << *ptr; // выведется 7
```

Когда адрес значения переменной присвоен указателю, то выполняется следующее:

- ptr — это то же самое, что и &value;
- *ptr обрабатывается так же, как и value.

Поскольку *ptr обрабатывается так же, как и value, то мы можем присваивать ему значения так, как если бы это была бы обычная переменная. Например:

```
1 int value = 5;
2 int *ptr = &value; // ptr указывает на value
3
4 *ptr = 7; // *ptr - это то же самое, что и value, которому мы присвоили значение 7
5 std::cout << value; // выведется 7
```

- Однако, оказывается, **указатели полезны в следующих случаях**:
- **Случай №1**: Массивы реализованы с помощью указателей. Указатели могут использоваться для итерации по массиву (это мы рассмотрим в следующих уроках).
- **Случай №2**: Они являются единственным способом динамического выделения памяти в C++ (это мы рассмотрим в следующих уроках). Это, безусловно, самый распространённый вариант использования указателей.
- **Случай №3**: Они могут использоваться для передачи большого количества данных в функцию без копирования этих данных (это мы рассмотрим в следующих уроках).
- **Случай №4**: Они могут использоваться для передачи одной функции в качестве параметра другой функции.
- **Случай №5**: Они используются для достижения полиморфизма при работе с наследованием (это мы рассмотрим в следующих уроках).
- **Случай №6**: Они могут использоваться для представления одной структуры/класса в другой структуре/классе, формируя, таким образом, целые цепочки.