|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

|  |  |
| --- | --- |
| Предмет: ВП |  |
| Подготовил студент гр. ИУК4-12Б: | Бобров К.А. (https://t.me/I\_Am\_L0se) |

**Тема №12: Указатели и ссылки. Арифметика указателей. Примеры.**

Арифметика указателей означает выполнение арифметических операций над указателями. Это относится к операциям, которые допустимо выполнять с указателями.

**Ниже приведены арифметические операции, допустимые для указателей на C ++:**

1. Увеличивающие и уменьшающие указатели
2. Добавление константы к указателям
3. Вычитание константы из указателей
4. Вычитание двух указателей одного типа
5. Сравнение указателей

1. Увеличение и уменьшение указателей

Увеличение или уменьшение указателя приведет к тому, что он будет ссылаться на адрес следующих или предыдущих данных в памяти. Этот процесс отличается от увеличения или уменьшения числовых данных.

**Увеличение указателя**

Увеличение указателя будет зависеть от типа адреса переменной, хранящейся в указателе. Если указатель хранит адрес переменной целочисленного типа, то размер целочисленного указателя может составлять 4 или 8 байт в зависимости от 32-разрядной или 64-разрядной машины соответственно. Итак, теперь, если мы увеличим переменную целочисленного типа, она увеличится на 4 или 8 байт в зависимости от ее размера.

### Уменьшение указателя

Когда мы применяем операцию уменьшения к указателю, указатель уменьшается на 4 или 8 байт в зависимости от машины.

int main()

{

    int num = 27;

    // Storing address of num in num\_pointer

    int\* num\_pointer = #

    // Print size of int

    cout << "Size of int: " << sizeof(int) << endl;

    // Print the address stored at num\_pointer

    cout << "Before Increment: " << num\_pointer << endl;

    // Increment pointer

    num\_pointer++;

    cout << "After Increment: " << num\_pointer << endl;

    // Print the address stored at num\_pointer

    cout << "Before Decrement: " << num\_pointer << endl;

    // Decrement pointer

    num\_pointer--;

    cout << "After Decrement: " << num\_pointer << endl;

    return 0;

}

Вывод:

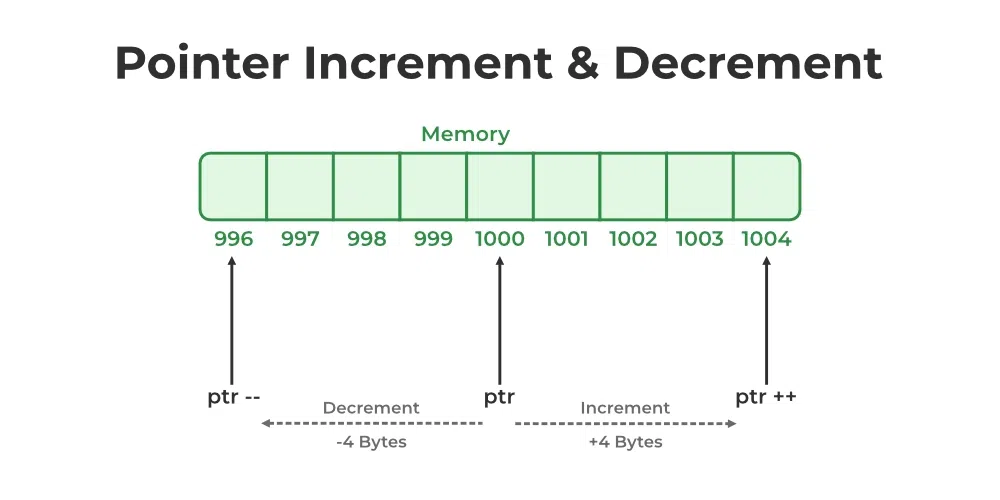
Size of int: 4

Before Increment: 0x7ffe3e7f56d4

After Increment: 0x7ffe3e7f56d8

Before Decrement: 0x7ffe3e7f56d8

After Decrement: 0x7ffe3e7f56d4



## 2. Добавление константы к указателям

Мы можем добавлять целочисленные значения к указателям, и указатель корректируется в зависимости от размера типа данных, на который он указывает. Например, если целочисленный указатель хранит адрес 1000, и мы добавляем значение 5 к указателю, он сохранит новый адрес как:

1000 + (5 \* 4(size of an integer)) = 1020

int main()

{

    int num = 20;

    int\* ptr = #

    cout << "Address stored in ptr: " << ptr << endl;

    // Adding the integer value 1 to the pointer ptr

    ptr = ptr + 1;

    cout << "Adding 1 to ptr: " << ptr << endl;

    // Adding the integer value 2 to the pointer ptr

    ptr = ptr + 2;

    cout << "Adding 2 to ptr: " << ptr << endl;

    return 0;

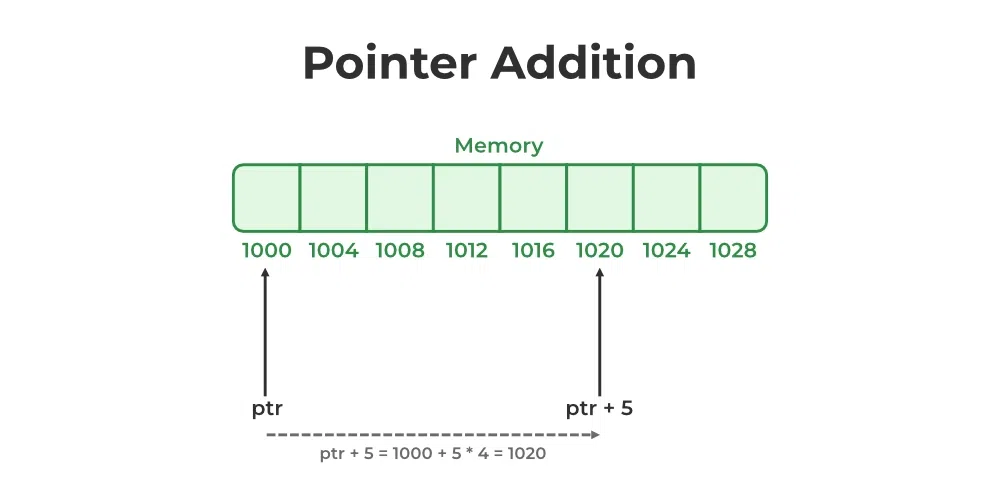
}

Вывод:

Address stored in ptr: 0x7ffdb8634a94

Adding 1 to ptr: 0x7ffdb8634a98

Adding 2 to ptr: 0x7ffdb8634aa0



## 3. Вычитание константы из указателей

Мы также можем вычитать константу из указателей, и это то же самое, что добавление константы к указателю. Например, если целочисленный указатель хранит адрес 1000, и мы вычитаем значение 5 из указателя, он сохранит новый адрес в виде:

1000 - (5 \* 4(size of an integer)) = 980

int main()

{

    int num = 100;

    int\* ptr = #

    cout << "Address stored in ptr: " << ptr << endl;

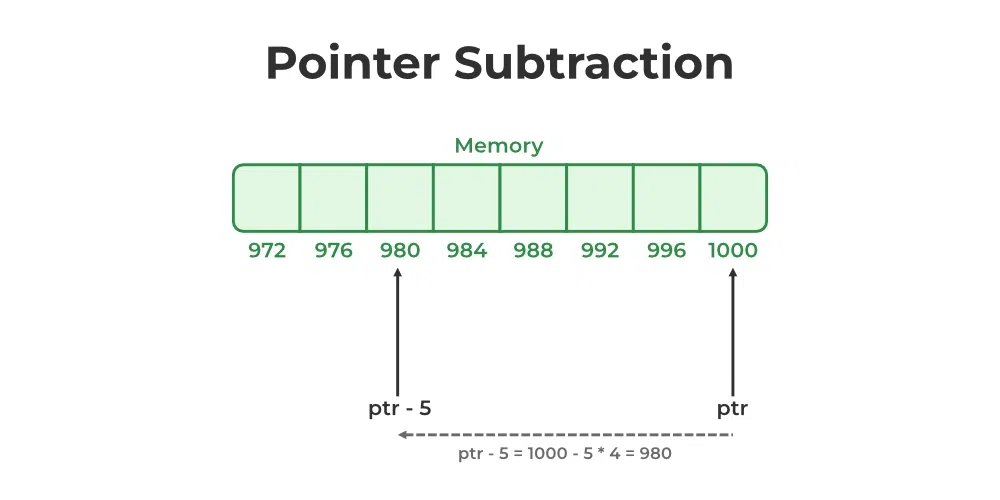
    // Subtracting the integer value 1 from pointer ptr

    ptr = ptr - 1;

    cout << "Subtract 1 from ptr: " << ptr << endl;

    return 0;

}



## 4. Вычитание двух указателей одного и того же типа данных

Вычитание двух указателей может быть выполнено только тогда, когда оба указателя имеют один и тот же тип данных. Вычитание двух указателей дает количество элементов, присутствующих между двумя указателями.

int main()

{

    int num = 45;

    int\* ptr1 = #

    // Adding 4 to ptr1 and stored in ptr2

    int\* ptr2 = ptr1 + 4;

    cout << "Address stored in ptr1:" << ptr1 << endl;

    cout << "Address stored in ptr2:" << ptr2 << endl;

    // Subtracting ptr2 from ptr1

    cout << "ptr2 - ptr1 = " << ptr2 - ptr1 << endl;

    return 0;

}

Вывод:

Address stored in ptr1:0x7ffdd3f5673c

Address stored in ptr2:0x7ffdd3f5674c

ptr2 - ptr1 = 4

## 5. Сравнение указателей

В C ++ мы можем выполнить сравнение между двумя указателями, используя операторы отношения ( **>, <, >=, <=, ==, !=**). Обычно мы используем эту операцию, чтобы проверить, указывают ли два указателя as на одну и ту же ячейку памяти или нет.

int main()

{

    // declaring some pointers

    int num = 10;

    int\* ptr1 = #

    int\*\* ptr2 = &ptr1;

    int\* ptr3 = \*ptr2;

    // comparing equality

    if (ptr1 == ptr3) {

        cout << "Both point to same memory location";

    }

    else {

        cout << "ptr1 points to: " << ptr1 << endl;

        cout << "ptr3 points to: " << ptr3 << endl;

    }

    return 0;

}

Вывод:

Both point to same memory location

### Сравнение с NULL

Мы можем сравнить указатель типа с NULL. Эта операция помогает нам определить, указывает ли данный указатель на некоторый адрес памяти или нет. Это помогает нам контролировать ошибки, такие как ошибки сегментации.

int main()

{

    int num = 10;

    // assigning null in case we dont use pointer

    int\* ptr = NULL;

    ptr = #

    // checking if the pointer is in use or not

    if (ptr == NULL) {

        cout << "No value is pointed";

    }

    else {

        cout << "The value pointed is " << \*ptr;

    }

    return 0;

}

Вывод:

The value pointed is 10

\*Данные взяты с сайта https://www.geeksforgeeks.org/cpp-pointer-arithmetic/