

Модуль 4

- Операторы сравнения
- Условный тернарный оператор
- Логические операторы
- Побитовые операторы

Операторы сравнения

Оператор	Символ	Пример	Операция
Больше	>	$x > y$	true, если x больше y, в противном случае — false
Меньше	<	$x < y$	true, если x меньше y, в противном случае — false
Больше или равно	>=	$x >= y$	true, если x больше/равно y, в противном случае — false
Меньше или равно	<=	$x <= y$	true, если x меньше/равно y, в противном случае — false
Равно	==	$x == y$	true, если x равно y, в противном случае — false
Не равно	!=	$x != y$	true, если x не равно y, в противном случае — false

Конструкция if-else

```
if (условие)
{
    инструкции;
}
```

```
if (condition) {
// код, который исполнится, если condition истинно
} else {
// код, который исполнится, если condition ложно
}
```

Конструкция if-else

```
if (condition1) {  
    // случай, когда condition1 истинно  
} else if (condition2) {  
    // случай, когда condition1 ложно, а condition2 истинно  
} else if (condition3) {  
    // случай, когда condition1 и condition2 ложны, а condition3 истинно  
} else {  
    // случай, когда condition1, condition2 и condition3 ложны  
}
```

Сложные условия

Условия можно комбинировать с помощью логических операторов `&&` (и), `||` (или) и `!` (не).

```
int main() {
    int a, b, x;
    /* Тут должна быть логика заполнения объявленных переменных*/
    if (a <= x && x <= b) {
        // точка x лежит на отрезке [a; b]
    } else {
        // точка x лежит вне отрезка [a; b]
    }
    // то же самое можно было бы проверить так:
    if (!(x < a || x > b)) { // отрицание
        // точка x лежит на отрезке [a; b]
    } else {
        // точка x лежит вне отрезка [a; b]
    }
}
```

Логические операторы

Используются в сложных условиях

Оператор	Символ	Пример	Операция
Логическое НЕ	!	!x	true, если x — false и false, если x — true
Логическое И	&&	x && y	true, если x и y — true, в противном случае — false
Логическое ИЛИ		x y	true, если x или y — true, в противном случае — false

Логические операторы: НЕ

```
#include <iostream>

int main()
{
    int x = 5;
    int y = 7;

    if (!x == y)
        std::cout << "x does not equal y";
    else
        std::cout << "x equals y";

    return 0;
}
```

Логические операторы: ИЛИ

Логический оператор ИЛИ ()		
Левый операнд	Правый операнд	Результат
false	false	false
false	true	true
true	false	true
true	true	true

Логические операторы: И

Логический оператор И (&&)		
Левый операнд	Правый операнд	Результат
false	false	false
false	true	false
true	false	false
true	true	true

Выражение №1: `(true && true) || false`

Выражение №2: `(false && true) || true`

Выражение №3: `(false && true) || false || true`

Выражение №4: `(5 > 6 || 4 > 3) && (7 > 8)`

Выражение №5: `!(7 > 6 || 3 > 4)`

Задача

При помощи цикла найдем максимальный элемент массива.

Switch

Зачем использовать Switch?

Типы, которые могут использоваться:

- char,
- short,
- int,
- long,
- long long,
- enum

Switch

```
bool isDigit(char p)
{
    switch (p)
    {
        case '0': // если p = 0
        case '1': // если p = 1
        case '2': // если p = 2
        case '3': // если p = 3
        case '4': // если p = 4
        case '5': // если p = 5
        case '6': // если p = 6
        case '7': // если p = 7
        case '8': // если p = 8
        case '9': // если p = 9
            return true; // возвращаем true
        default: // в противном случае, возвращаем false
            return false;
    }
}
```

Switch

Когда кейс совпал (или выполняется default), то выполнение начинается с первого кейс-мента, который находится после соответствующего кейса и продолжается до тех пор, пока не будет выполнено одно из следующих условий завершения:

- Достигнут конец блока switch.
- Выполняется оператор return.
- Выполняется оператор goto.
- Выполняется оператор break.

```
switch (2)
{
    case 1: // Не совпадает!
        std::cout << 1 << '\n'; // пропускается
    case 2: // Совпало!
        std::cout << 2 << '\n'; // выполнение кода начинается здесь
    case 3:
        std::cout << 3 << '\n'; // это также выполнится
    case 4:
        std::cout << 4 << '\n'; // и это
    default:
        std::cout << 5 << '\n'; // и это
}
```

Switch

```
switch (2)
{
    case 1: // не совпадает - пропускается
        std::cout << 1 << '\n';
        break;
    case 2: // совпало! Выполнение начинается со следующего стейтмента
        std::cout << 2 << '\n'; // выполнение начинается здесь
        break; // оператор break завершает выполнение switch
    case 3:
        std::cout << 3 << '\n';
        break;
    case 4:
        std::cout << 4 << '\n';
        break;
    default:
        std::cout << 5 << '\n';
        break;
}
```

Switch

```
switch (x)
{
    case 1:
        int z; // ок, объявление разрешено
        z = 5; // ок, операция присваивания разрешена
        break;
    case 2:
        z = 6; // ок, переменная z была объявлена выше, поэтому мы можем использовать её здесь
        break;
    case 3:
        int c = 4; // нельзя, вы не можете инициализировать переменные внутри case
        break;
    default:
        std::cout << "default case" << std::endl;
        break;
}
```


Switch

```
switch (1)
{
    case 1:
        { // обратите внимание, здесь указан блок
            int z = 5; // хорошо, переменные можно инициализировать внутри блока, который находится
внутри кейса
            std::cout << z;
            break;
        }
    default:
        std::cout << "default case" << std::endl;
        break;
}
```

Условный тернарный оператор

Условный тернарный оператор — это удобное упрощение ветвления if/else, особенно при присваивании результата переменной или возврате определенного значения.

Оператор	Символ	Пример	Операция
Условный	?:	<code>с ? х : у</code>	Если <code>с</code> — ненулевое значение (<code>true</code>), то вычисляется <code>х</code> , в противном случае — <code>у</code>

Условный тернарный оператор

```
if (условие)  
    выражение;  
else  
    другое_выражение;
```



```
(условие) ? выражение : другое_выражение;
```

```
if (условие)  
    x = значение1;  
else  
    x = значение2;
```



```
x = (условие) ? значение1 : значение2;
```

```
if (x > y)  
    larger = x;  
else  
    larger = y;
```



```
larger = (x > y) ? x : y;
```

Условный тернарный оператор вычисляется как выражение

```
bool inBigClassroom = false;  
const int classSize = inBigClassroom ? 30 : 20;
```

Побитовые операторы

Оператор	Символ	Пример	Операция
Побитовый сдвиг влево	<<	$x \ll y$	Все биты в x смещаются влево на y бит
Побитовый сдвиг вправо	>>	$x \gg y$	Все биты в x смещаются вправо на y бит
Побитовое НЕ	~	$\sim x$	Все биты в x меняются на противоположные
Побитовое И	&	$x \& y$	Каждый бит в x И каждый соответствующий ему бит в y
Побитовое ИЛИ		$x y$	Каждый бит в x ИЛИ каждый соответствующий ему бит в y
Побитовое исключающее ИЛИ (XOR)	^	$x \wedge y$	Каждый бит в x XOR с каждым соответствующим ему битом в y

Побитовые операторы

Побитовый сдвиг влево (<<) и побитовый сдвиг вправо (>>)

В побитовых операциях следует использовать только **целочисленные типы данных unsigned**, так как C++ не всегда гарантирует корректную работу побитовых операторов с целочисленными типами signed.

```
3 = 0011
```

```
3 << 1 = 0110 = 6
```

```
3 << 2 = 1100 = 12
```


```
3 << 3 = 1000 = 8
```

Побитовые операторы

```
#include <iostream>
int main()
{
    unsigned int x = 4;
    x = x << 1; // оператор << используется для побитового сдвига влево
    std::cout << x; // оператор << используется для вывода данных в
консоль
    return 0;
}
```

Побитовые операторы

Побитовый оператор НЕ

Побитовый оператор НЕ () меняет каждый бит на противоположный, например, с 0 на 1 или с 1 на 0

$4 = 0100$

$\sim 4 = 1011$ (двоичное) = 11 (десятичное)

Побитовые операторы

Побитовые операторы И

Побитовые операторы И (&) и ИЛИ (|) работают аналогично **логическим операторам И и ИЛИ**. Однако, побитовые операторы применяются к каждому биту отдельно!

```
5 & 6
```

```
0 1 0 1 // 5
```

```
0 1 1 0 // 6
```

```
-----
```

```
0 1 0 0 // 4
```

Побитовые операторы

Побитовые операторы ИЛИ

5 | 6

0 1 0 1 // 5

0 1 1 0 // 6

0 1 1 1 // 7

Побитовые операторы

Побитовые операторы исключающее ИЛИ (XOR от англ. «eXclusive OR»)

6 ^ 3

0 1 1 0 // 6

0 0 1 1 // 3

0 1 0 1 // 5

x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Побитовые операторы

Оператор	Символ	Пример	Операция
Присваивание с побитовым сдвигом влево	<<=	x <<= y	Сдвигаем биты в x влево на y бит
Присваивание с побитовым сдвигом вправо	>>=	x >>= y	Сдвигаем биты в x вправо на y бит
Присваивание с побитовой операцией ИЛИ	=	x = y	Присваивание результата выражения x y переменной x
Присваивание с побитовой операцией И	&=	x &= y	Присваивание результата выражения x & y переменной x
Присваивание с побитовой операцией исключающего ИЛИ	^=	x ^= y	Присваивание результата выражения x ^ y переменной x

Например, вместо `x = x << 1;` мы можем написать `x <<= 1;`.

Побитовые операторы

Какой результат $0110 \gg 2$ в двоичной системе счисления?

Какой результат $5 \mid 12$ в десятичной системе счисления?

Какой результат $5 \& 12$ в десятичной системе счисления?

Какой результат $5 \wedge 12$ в десятичной системе счисления?

Побитовые операторы

Напишите программу, которая проверяет наличие определенного бита в заданном числе и выводит результат на экран.