void — это такой тип данных, который используется для указания того факта, что функция не возвращает никаких данных.
void* — это нетипизированный указатель, т.е. указатель на произвольный тип данных.

Операции.

По количеству операндов операции делятся на три типа: унарные, бинарные, тернарные.

Унарные:

1. +, - при идентификации переменной
2. ~ — побитовое логическое отрицание или дополнение. Например, при дополнении возвращает целое число, биты в котором инвертированы относительно битов аргумента.

```
unsigned char c = ...;
c = c + ~c;

// результатом будет 255
```

3. ! — логическое отрицание. Аргумент целочисленный. Если его значение равно нулю, то результат операции единица. В остальных случаях результат операции — ноль.

```
!!!11 //0
!!11  //1
```

4. sizeof — возвращает размер аргумента в байтах. Аргументом может выступать имя типа, либо произвольное выражение.

```
sizeof(double) //8
```

```
short a;
sizeof(a) //2
```

5. Инкремент, декремент (++, -- соответственно) — аргументом данных операций является идентификатор переменной. Инкремент увеличивает на единицу, а декремент уменьшает. Различают префиксное и постфиксное использование этих операций. В первом случае в начале применяется операция и в вычислении выражения участвует новое значение переменной. Во втором случае выражение вычисляется со старым значение, а уже потом вычисляется значение переменной.

```
int x = 8, y;

y = ++x; или y = x++;

//в первом случае y = x = 9, во втором случае y = 8, x = 9
```

Бинарные операции делятся на следующие группы:

1. мультипликативные — умножение, деление, остаток от деления (*, /, %). В умножении и делении операнды могут быть целочисленные и вещественные, если оба аргумента целые, то аргумент операции также целого типа. Если хотя бы один аргумент вещественный, то результат будет вещественного типа.

```
14 / 5 => 2

14 / 5. => 2.8
14 / 5.0 => 2.8

14 % 5 => 4
```

2. аддитивные — сложение, вычитание(+, -). Аргументы могут быть целые или вещественный. Тип результата определяется так же, как у умножения и деления.

```
4 + 5  => 9
4 + 5. => 9.0
```

3. операции сдвига — сдвиг влево, сдвиг вправо (<<, >>). Аргументы целочисленные. Результатом является сдвиг битов первого аргумента на число позиций указанным вторым аргументом. Битовый сдвиг влево это умножение на степени двойки. Битовый сдвиг делает более эффективный код по сравнению с умножением.

```
unsigned char = 3;

c << 1 => 6  // 011 => 110
c << 2 => 12 // 011 => 1100
c << 4 => 48 // 011 => 110000

c >> 1 => 1 // 011 => 001
```

4. Операция сравнения (>, <, >=, <=, ==, !=). Аргументы — стандартные простые типы. Результат операции 0 или 1.
5. Поразрядные операции (битовые операции). Побитовые и/или, побитовое исключающее или (&, |, ^).

```
1100
1010

12 & 10 => 8  // 1000
12 | 10 => 14 // 1110
12 ^ 10 => 6  // 0110
```

7. Логические операции — логическое И и ИЛИ (&&, ||). Аргументы целочисленные. Результат 0 или 1.
8. Операции присваивания. Выражения должны быть совместимы по типу.

```
вып1 = вып2
```

Если, например, вып1 int, а вып2 float, то происходит отбрасывание дробной части. При этом компилятор генерирует предупреждение. Вып1 должно быть леводопустимо. Значением операции является значение присвоенного выражения.
Помимо простого присваивания, если присваивания выполняемые с другими выражениями.

```
x += y
x = x + y

x /= y
x = x / y // и т.д.
```

10. , — Операция последовательного вычисления. Используется для объединения двух выражений в одно, там где требуется синтаксически единое выражение.

Тернарная операция («Карманный if»):

Данная операция имеет следующий формат:

```
выр1 ? выр2 : выр3;
```

Вычисляется ***выр1***. Если его значение не равно нулю, то вычисляется ***выр2***, иначе вычисляется ***выр3***.