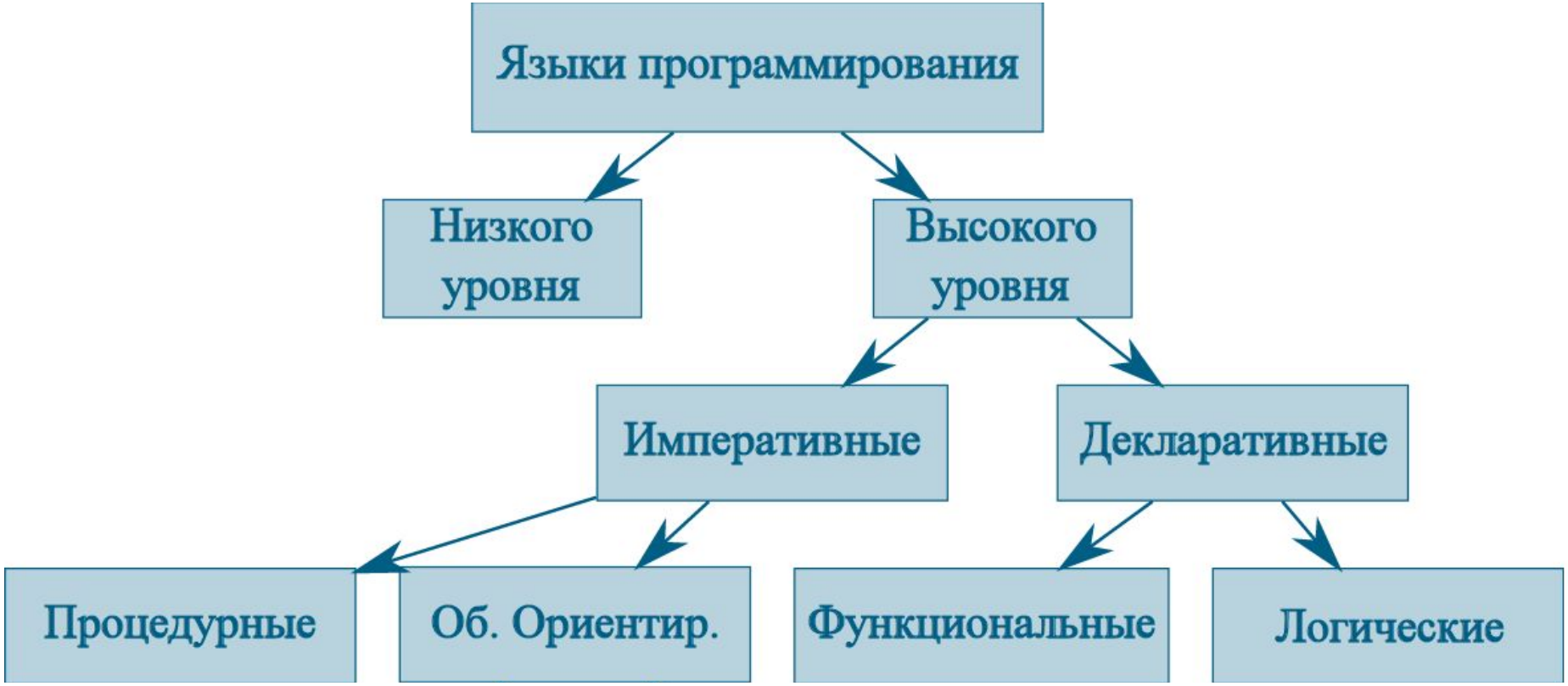
Программирование на языке Си. Часть 1

ЛЕКСЕМЫ. ПЕРЕМЕННЫЕ. КОНСТАНТЫ. ОСНОВНЫЕ ОПЕРАТОРЫ.

Классификация ЯП



Первые программы

Машинный язык — система команд (набор **кодов** операций) конкретной вычислительной машины, которая интерпретируется непосредственно процессором или микропрограммами этой вычислительной машины.

Программа «`Hello, world!`» для процессора архитектуры x86 (ОС MS DOS) выглядит следующим образом:

```
BB 11 01 B9 0D 00 B4 0E 8A 07 43 CD 10 E2 F9 CD 20 48 65
6C 6C 6F 2C 20 57 6F 72 6C 64 21
```

Ассемблеры (*assembly languages*)

- это машинно-ориентированные языки низкого уровня. Преобразование команд в машинный код выполняет специальная программа – ассемблер (сборщик).

```
.MODEL TINY
CODE SEGMENT
ASSUME CS:CODE, DS:CODE
ORG 100h
START:
        mov ah,9
        mov dx,OFFSET Msg
        int 21h
        int 20h
        Msg DB 'Hello World',13,10,'$'
CODE ENDS
END START
```

Классы языков программирования

Декларативные – программист описывает, ЧТО нужно получить на выходе. Например, язык разметки **HTML** – описывает, как должна выглядеть веб-страница. Сюда же относятся **функциональные** языки (**Haskell, Lisp,...**) и **логические** языки (**Prolog**).

Императивные – программист с помощью команд (инструкций) описывает, КАК нужно получить результат. Примеры языков: **C/C++/C#, Java, PHP, Python**….

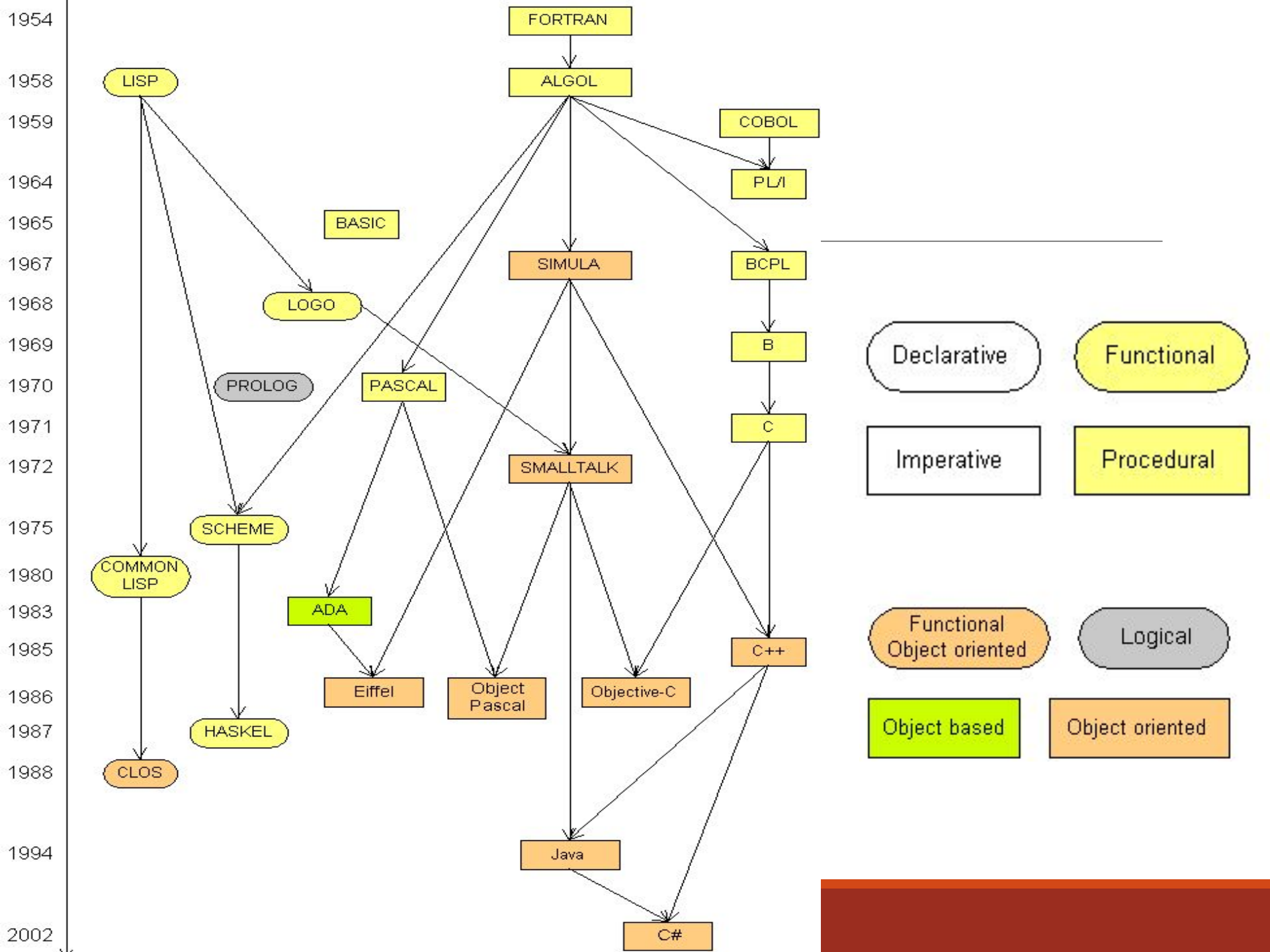
Парадигмы программирования

Процедурная : выполнение команд шаг за шагом, разбиение программ на подпрограммы (процедуры). *«Программирование с помощью глаголов».*

Структурная : разбиение процедур на более мелкие блоки, отказ от оператора goto.

Модульная : разбиение программ на связанные модули (файлы).

Объектно-ориентированная : «программирование с помощью существительных».


1954
1958
1959
1964
1965
1967
1968
1969
1970
1971
1972
1975
1980
1983
1985
1986
1987
1988
1994
2002
FORTRAN
ALGOL
LISP
COBOL
PL/I
BASIC
SIMULA
BCPL
LOGO
B
PROLOG
PASCAL
C
SMALLTALK
SCHEME
COMMON LISP
ADA
C++
Eiffel
Object Pascal
Objective-C
HASKEL
CLOS
Java
C#
Declarative
Functional
Imperative
Procedural
Functional Object oriented
Logical
Object based
Object oriented

Компиляторы и интерпретаторы

Компилятор – программа, преобразующая (*транслирующая*) исходный код на языке программирования в исполняемый файл, который содержит заголовок и машинный код для определенной платформы *(например, текстовый файл с кодом программы -> exe-файл).*

Компилируемые языки: C/C++, Pascal, Delphi, Go…

Интерпретатор – программа, которая выполняет инструкции языка по порядку.

Интерпретируемые языки: Python, PHP, JavaScript и все скриптовые.

Алгоритм

- это набор инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи за конечное число действий. Должен обладать следующими **свойствами**:

Дискретность : алгоритм – это последовательное выполнение простых шагов.

Детерминированность (определённость) : алгоритм выдаёт один и тот же результат для одних и тех же исходных данных.

Понятность : алгоритм должен включать только те команды, которые известны исполнителю.

Завершаемость (конечность) : алгоритм должен завершать работу и выдавать результат за конечное число шагов.

Массовость (универсальность) : применимость к разным наборам исходных данных.

Процесс создания ПО

1. анализ и постановка задачи,
2. построение алгоритмов,
3. проектирование программы,
4. разработка структур данных,

ПРОЕКТИРОВАНИЕ

5. написание текстов программ (кодирование),
6. отладка и тестирование программы (испытания программы),
7. документирование,
8. настройка (конфигурирование),
9. доработка и сопровождение

Язык Си

Разработчик: **Денис Ритчи** (Bell Laboratories), 1969—1973 гг.

Изначально разрабатывался для написания операционной системы Unix.

Класс: **процедурный**.

Тип исполнения: **компилируемый**.

Расширение файлов: ***.c**

Последняя версия: **C11** (2011 год)

Кроссплатформенный.

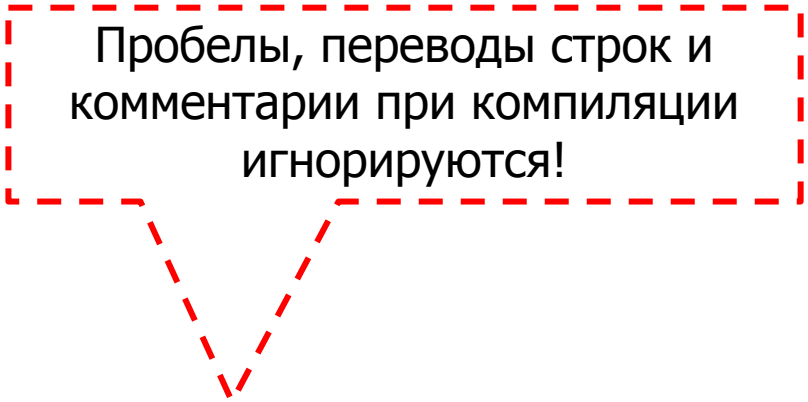
Символы языка С

- Буквы: **A…Z a…z** и знак подчеркивания **_**
- Цифры: 0…9
- Знаки: . , ; : ` " ( ) [ ] { } | + - * / % ? ! ~ < > = & ^
- Пробельные символы: символ пробела,

  \t – табуляция, \n – переход на след. строку

Лексемы в программе на С:

- идентификаторы,
- ключевые слова,
- константы,
- знаки операций,
- прочие разделители.



Идентификаторы

Идентификатор – это имя чего-либо, состоящее из последовательности символов.

В ЯП Си идентификаторами являются **типы данных, имена переменных, функций и метки.**

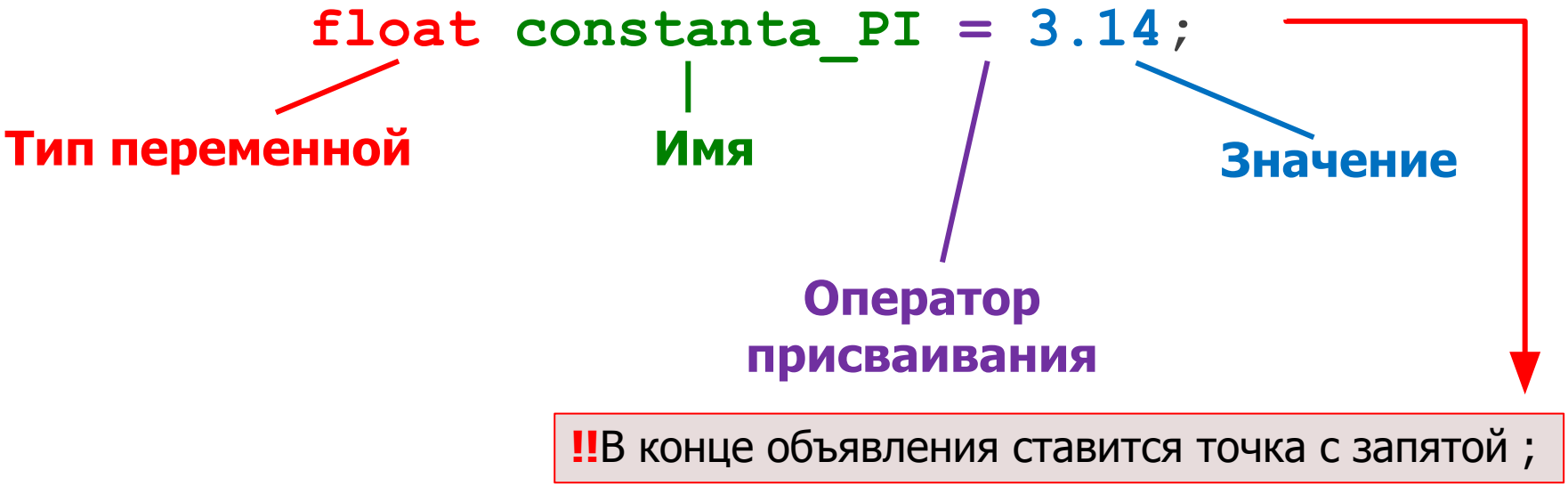
- могут включать буквы `A..Z a..z` , цифры `0..9` и символ `_` .
- не могут начинаться с цифр! ~~1Rad360~~
- Прописные и строчные буквы – это разные символы! Пример:

  `xz1, XZ1, xZ1, Xz1` – это разные идентификаторы!
- Идентификатор не должен совпадать с *ключевыми словами (см. далее)*

Переменные (П-е)

Переменная – именованная область памяти. У каждой такой области памяти есть **класс памяти**, **тип**, **адрес** и хранимое **значение**.

Объявление переменной (declaration):



Требования к именам П-ых

- **Имена должны иметь смысл**; `radius, perimeter, count.`
- Макс. длина имени – 32 символа и больше (зависит от компилятора).
- + те же, что и к идентификаторам.

Основные типы данных

Знак	Длина	Тип	Байт	Описание
signed / unsigned		char	1	один символ из локального символьного набора
	short	int	2	целое число
		int	4	
	long	int	4	
Нет		float	4	вещественное число одинарной точности с плавающей точкой
		double	8	вещественное число двойной точности с плавающей точкой
	long	double	10	вещественное число повышенной точности с плавающей точкой

Количество байт для каждого типа зависит от программной и аппаратной платформ!

Способы объявления П-ых

```
int lower, upper, step;
char c, line[1000];
```

Аналогично следующему (можно писать комментарии):

```
int lower; //Это однострочный комментарий
int upper; /*Это многострочный
комментарий*/
int step;
char c;
char line[1000];
```

Инициализация П-ых

При создании:

```
char esc = '\\';

int i = 0;

int limit = MAXLINE + 1;

float eps = 1.0e-05;
```

Во время выполнения:

```
int summa;

…

summa = a + b;
```

Ключевые слова

- это зарезервированные имена, которые имеют специальное значение для компилятора.

auto
break
case
char
const
continue
default
do
double
else
enum
extern
float
for
goto
if
inline
int
long
register
return
short
signed
sizeof
static
struct
switch
typedef
union
unsigned
void
volatile
while

Константы

1. Целочисленные
   - `0562` – восьмеричная
   - `0xFA` - шестнадцатеричная
   - `123` – десятичная
   - `123u(U)` – unsigned (без знака)
   - `123l(L)` – long

Правильно:
`0xF1uL`
`076U`
`9871231`

Неправильно:
`0F1u`
`076sL`

2. Символьные – заключаются в одинарные кавычки и содержат 1 символ: `'x'`.

Не могут содержать символ одинарной кавычки и конца строки. Вместо этого используют *escape-последовательности*:

Конец строки: `\n` Горизонтальная табуляция(Tab) `\t`

Одинарная и двойная кавычка, ?, обратная косая черта: `\' \" \? \\`

3. Вещественные с плавающей запятой

- с точкой: 3.534, -98.001
- в научной нотации:

< мантисса>E<порядок> или **<мантисса>e<порядок>**

Научная нотация	Обычная запись
0.653412E3	653,412
-3.73E-1	-0,373
+64.2e+02	6420,0
-87E2	-8700,0
.123e3	123
4037e-5	0,04037

4. Строковые (строковый литерал) – последовательность символов, заключенная в двойные кавычки: “**hello, world**” . Заканчивается символом `\0`.

!!! **"hello," "world"** эквивалентно **"hello, world"**

Операции в Си

Операция — это функция, которая выполняется над операндами и возвращает вычисленное значение — результат выполнения операции.

Операнд — это константа, переменная, выражение или вызов какой-либо определённой в программе функции.

Операции по количеству задействованных операндов делятся на:
унарные операции — операции вида **[знак операции] [операнд]**
бинарные операции **[операнд] [знак операции] [операнд]**
и **тернарные операции**.

Ассоциативность – направление выполнения (→ или ←).
z = a + b +c; // → z = a = b = 0; // ←

Унарные операции

Обозн	Название	Пример
+	(унарный плюс)	`int L = +5`
-	унарный минус	`int L = -5`
~	взятие обратного кода	`int L = ~5; //L = -6`
!	логическое отрицание	`int k = 209; int L = !k //L = 0`
&	взятие адреса	`int k; int L = &k;` `// L = 0xFFFFFFFA ;`
*****	операция разыменовывания указателя	См. тему «Указатели» далее.
sizeof	операция определения занимаемого объектом объёма памяти	`int k;` `int L = sizeof(k); //L = 4;`

Программа для определения размерности простых типов

Наберите текст в окне редактирования кода в C++ Builder, нажмите F9 и посмотрите на результат выполнения. Добавьте вывод размерности типа char.

```
int main()
{
        float i;
        double j;
        long double k;
        short a;
        int b;
        long c;
        printf("Sizes (bytes) : float = %d, double = %d, long double = %d, short = %d, int = %d, long = %d",
        sizeof(i),sizeof(j), sizeof(k), sizeof(a), sizeof(b), sizeof(c));
        getch();
        return 0;
}
```

Бинарные операции

Арифметические операции

+ - * / и % (взятие остатка от деления)

При целочисленном делении дробная часть отбрасывается!

Операция % **неприменима** к числам типа float или double.

Направление округления при операции ‘/’ или знак результата при операции ‘%’ для отрицательных аргументов зависят от системы.

Операции присваивания

= Обычный оператор присваивания

Пример:

```
a = a + 2;
x = x * (y + 1);
```

Сокращенные операторы присваивания

+= -= *= /= %=

<<= >>= &= ^= |=

Пример:

```
/*примеры выше можно записать как:*/
a += 2;
x *= ( y + 1 );
```

Операции отношения

Оп-р	Название	Пример
>	Больше	
>=	Больше или равно	
<	Меньше	
<=	Меньше или равно	
==	Проверка на равенство (**1 если операнды равны**)	
!=	Проверка на неравенство (**1 если операнды НЕ равны**)	

```
int main()
{
     int k = 0;
    printf("Enter k:\n");
    scanf("%d", &k);
    if(k>=0)
    {
        printf("K > 0!");
    }
    else if(k < 0)
    {
        printf("%d+2 = %d", k, k+2);
    }
    else printf("Ooops! Your program has
been hacked!");
        getch();
}
```

Логические операции

&&	логическое И	Возвращают в качестве результата **0** либо **число больше нуля** (соответственно, ложь или истина)
||	логическое ИЛИ	

Пример:

```
int k = 0; int L = 1;
//Если k равно 0 И L больше 0
if(k == 0 && L > 0)
{
   printf();
}
```

Поразрядные операции

&	поразрядное И
|	поразрядное ИЛИ
^	поразрядное исключающее ИЛИ (XOR)
>>	сдвиг вправо
<<	сдвиг влево
~	инверсия бит

Выполняют действия над **отдельными битами**.
Применимы только к целым числам.

Пример:

```
// обнуление всех бит, кроме последних семи.
a = a & 0x7F;

short int a = 100; // 0000 0000 0110 0100
a = a << 6; /* сдвиг на 6 бит влево, a = 1 1001 0000 0000

//Быстрое деление на степень двойки
int a = 1024 >> 1; // a = 512
//Быстрое умножение на степень двойки
int a = 1024 << 1; // a = 2048
```

Операции инкремента и декремента

Прибавление и вычитание 1 из значения переменной X можно кратко записывать следующим образом: **X++ (++X) и X-- (--X)**

X++ X-- постфиксная запись

++X --X префиксная запись

Пример:

```
int x, n = 5;
x = ++n; // x = 6, сначала к n прибавится 1, затем x приравняется к n

x = n++; // x = 5, сначала x приравняется к n, затем к n прибавится 1
// n в обоих случаях станет равно 6!
```

Эти операции применимы только к переменным. Выражения типа `(i+j)++` недопустимы.

Лексемы	Операция	Класс	Приор	Ассоциат-ть
++ ---	Инкремент и декремент	постфиксный	15	справа налево
sizeof	размер	унарный	15	справа налево
~	побитовое НЕ	унарный	15	справа налево
!	логическое НЕ	унарный	15	справа налево
- +	изменение знака, плюс	унарный	15	справа налево
&	адрес	унарный	15	справа налево
*	разыменование	унарный	15	справа налево
(имя типа)	приведение типа	унарный	15	справа налево
* / %	мультипликативные опер-и	бинарный	13	слева направо
+ -	аддитивные операции	бинарный	12	слева направо
<< >>	сдвиг влево и вправо	бинарный	11	слева направо
< > <= >=	отношения	бинарный	10	слева направо
== !=	равенство/неравенство	бинарный	9	слева направо
&	побитовое И	бинарный	8	слева направо
^	побитовое искл-ее ИЛИ	бинарный	7	слева направо
|	побитовое ИЛИ	бинарный	6	слева направо
&&	логическое И	бинарный	5	слева направо
||	логическое ИЛИ	бинарный	4	слева направо
? :	условие	тернарный	3	справа налево
= += -= *= /= %= <<= >>= &= ^= |=	присваивание	бинарный	2	справа налево
,	последовательное вычисление	бинарный	1	слева направо

Прототип:

```
int printf(const char *format[, argument, ...]);
```

Использование:

```
printf("hello, world"); /* функция вывода информации на экран*/
```

Прототип:

```
int getch(void);
```

Использование:

```
getch();/* считывает символ с клавиатуры, но не выводит его на экран. Цель использования – чтобы окно программы закрывалось только по нажатию клавиши*/
```

Оператор return

return *выражение*$_{необяз.}$;

Предназначен для возврата результата выполнения функции. После его выполнения текущая функция завершает свою работу и передает управление коду, вызвавшему функцию (в случае функции main() – операционной системе).

Если функция ничего не возвращает, то оператор return не является обязательным.

Пример1:

```
void set_property(){операция1; …}
```

Пример2:

```
int set_property(){операция1; … return 0;}
```