ФГОБУ ВПО "СибГУТИ"
Кафедра вычислительных систем

ПРОГРАММИРОВАНИЕ
ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Состав высокоуровневого языка программирования СИ

Преподаватель:
Доцент Кафедры ВС, к.т.н.
Поляков Артем Юрьевич

Состав языка СИ

Алфавит языка СИ

- прописные и строчные буквы латинского алфавита:

  A ... Z, a ... z

- цифры:

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- специальные знаки:

  " , { } | [ ] ( ) + - / % \ ; ' . : ? > < = _ & ! * # ~ ^

- неотображаемые символы:

  пробел, табуляция, переход на новую строку

- В комментариях, строках и символьных константах могут использоваться другие литеры (например, русские буквы).

Состав языка СИ

Естественный язык	Язык программирования
Алфавит	Символы
Слова	Лексемы
Словосочетания	Выражения
Предложения	Операторы

Лексемы языка СИ

Из символов алфавита составляются лексемы - минимальные единицы языка, имеющие самостоятельный смысл:

- константы;
- имена объектов;
- ключевые слова;

} **Идентификаторы**

- знаки операций;
- разделители (скобки, точка, запятая, пробельные символы).

Целые константы

Формат	Примеры
Десятичные константы: последовательность десятичных цифр (0…9), не начинающаяся с нуля (если это не ноль).	**5, 0, 65535**
Восьмеричная константа: ноль, за которым следуют восьмеричные цифры (0…7)	**05, 065535**
Шестнадцатеричная константа: 0x или 0X, за которым следуют шестнадцатеричные цифры (0..9, A..F)	**0x5, 0x65535, 0xFA15, 0xfcab**

Целые константы (вопросы)

x_{10}	x_2
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111

В какой системе счисления записаны следующие константы:

1) 0x120;
2) 121;
3) 0122.

Запись в программе указанных значений в восьмеричной СС:

1) $0011\ 1111_2 = ?$
2) $1010\ 1010_2 = ?$
3) $0010\ 0100_2 = ?$

Целые константы (ответы)

x_{10}	x_2
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111

В какой системе счисления записаны следующие константы:

1) 0x120; - шестнадцатеричное
2) 121; - десятичное
3) 0122. - восьмеричное

Запись в программе указанных значений в восьмеричной СС:

1) $0011\ 1111_2 = 00\ 111\ 111_2 = 077$
2) $1010\ 1010_2 = 10\ 101\ 010_2 = 0252$
3) $0010\ 0100_2 = 00\ 100\ 100_2 = 044$

Вещественные константы

Формат	Примеры
Десятичный формат: <целая часть> **.** <дробная часть> дробная ИЛИ целая часть могут быть опущены	**2.5 , 2. , .65**

Экспоненциальный формат:

<целое> **.** <дробн.> {E | e} <степень>

$$a.b\mathrm{E}c = a.b \cdot 10^{c}$$

0.5e2 = 50

100E-1 = 10

Могут быть опущены либо целая часть, либо дробная, но не обе сразу.

Вещественные константы (вопросы)

1. Запишите следующие числа в экспоненциальном формате:

1) 2,5;
2) 1,635;
3) 0,00000001;
4) 1,000000001;
5) 15000000000;

2. Запишите следующие числа в десятичном формате:

1) 50E-4;
2) 5E3;
3) 1.345E5;
4) 1.1E-5;

Вещественные константы (ответы)

1. Запишите следующие числа в экспоненциальном формате:

1) 2,5; - 2.5E0 , 0.25E1, 25E-1
2) 1,635; - 1635E-3, .1635E1
3) 0,00000001; - 1E-8
4) 1,000000001; - 1,000000001E0
5) 15000000000; - 1.5E10

2. Запишите следующие числа в десятичном формате:

1) 50E-4; - 0.005
2) 5E3; - 5000
3) 1.345E5; - 134500
4) 1.1E-5; - 0.000011

Символьные константы

Формат	Примеры
Символьные константы: Несколько символов, заключенных в одинарные апострофы. Символьная константа соответствует записи в таблице символьной кодировки! Символ обратной косой черты используется для представления: • кодов, не имеющих графического изображения (например, \n – перевод на нов. строку); • символов апострофа, обратной косой черты, кавычки; • любого символа с помощью его 16-ричного или 8-ричного кода, например, \073, \0xF5.	**'a', 'B',** **'\n',** **'\''** **'\\'** **'\"'** **'\x30' ~ '0'** **'\061' ~ '1'**

Последовательности, начинающиеся с '\' называют управляющими, или escape-последовательностями.

Перевод символьного представления в численное

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
	. . .															
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	. . .															

- Коды цифр располагаются непрерывно друг за другом
- Если известно, что переменная C содержит код цифры, то получить числовое представление цифры можно отняв от значения C код 0-ля (**тип char – целочисленный!**):

```
int i = C – '0'; // или i = C – 0x30; i = C – 48;
```

Первый вариант **наиболее предпочтителен**, так как обладает **большей информативностью**

Символьные константы (вопросы)

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1.	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2.		!	"	#	$	%	&	'	(	)	*	+	,	—	.	/
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5.	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7.	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Какие символы закодированы следующими escape-последовательностями:

1) \x65; 2) \011; 3) \x3C;

Символьные константы (вопросы)

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1.	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2.		!	"	#	$	%	&	'	(	)	*	+	,	—	.	/
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5.	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7.	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Какие символы закодированы следующими escape-последовательностями:

1) \x65; - 'e' 2) \011; - TAB 3) \x3C; - '<'

Строковые константы

Формат	Примеры
Строковые константы: Последовательность символов, заключенная в кавычки Строковые константы, отделенные только пробельными символами, при компиляции объединяются в одну. **"test " "string" → "test string"** Длинную строковую константу можно разместить на нескольких строках. Для экранирования переноса на новую строку используется символ '\': **"this is a ** **test string" → "this is a test string"**	**"D=%d"** **"test"** **"Hello world"**

Идентификаторы

Идентификатором называется имя, сопоставленное объекту языка (например: *переменной*, *функции*) или ключевым словам.

Идентификатор представляет собой последовательность строчных, прописных букв латинского алфавита, цифр и знака подчеркивания. Причем идентификаторы должны начинаться либо с буквы, либо со знака подчеркивания.

Строчные и прописные буквы рассматриваются компиляторами языка Си как несовпадающие.

Примеры идентификаторов:

- `int, float`
- `a, b, id`
- `_test`

Идентификаторы (вопросы)

Почему на идентификаторы накладывается следующее ограничение: " идентификаторы должны начинаться либо с буквы, либо со знака подчеркивания **" ?**

Идентификаторы (ответы)

Почему на идентификаторы накладывается следующее ограничение: " идентификаторы должны начинаться либо с буквы, либо со знака подчеркивания **" ?**

Пусть первый символ идентификатора может быть цифрой. В этом случае возможны конфликты между именами идентификаторов и константами:

1) лексема 1abc 0abc – не конфликтуют, так как в десятичной СС (константы начинающиеся с цифр 1-9) и восмеричной СС (константы, начинающиеся с 0) нет букв.
2) лексема 0xaaa – может быть шестнадцатеричной константой!;
3) лексема 0123 – может быть восьмеричной константой;
4) лексема 123 – может быть десятичной константой;

Служебные слова

Служебные слова представляют собой **идентификаторы**, имеющие специальное значение для компилятора языка Си.

Служебные слова нельзя использовать как имя переменной. Они применяются для использования определенных свойств языка.

asm*	double	goto	struct
auto	else	if	switch
break	end	int	typedef
case	entry*	long	union
char	enum	register	unsigned
const	exit	return	unix*
continue	extern	short	void
default	float	sizeof	volatile*
do	for	static	while

*Зависит от компилятора; для некоторых компиляторов может не быть служебным словом.

Операции (классификация по числу аргументов)

- Унарные (работают с одним аргументом):
  - **`sizeof(var)`** – возвращает размер **`var`** в байтах;
  - **`z++`** – увеличивает значение **`z`** на единицу;
  - **`-z`** – изменение знака **`z`**.
- Бинарные (требуют 2 аргумента):
  - **`a - b`** – разность значений **`a`** и **`b`** (не унарная!)
  - **`a > b`** – сравнение значений **`a`** и **`b`**, результат – истина, если условие выполняется и лож – в противном случае.
- Тренарные (требуют 3 аргумента):
  - **`a > 0 ? 1 : 2`** – если a > 0 то операция возвращает значение 1, в противном случае – 2.

Операции (классификация по типу)

Тип операции	унарные	бинарные	тренарные
Арифметические	`+, -`	`+, - , *, / , %`	
Логические	`!`	`||, &&`	
Сравнение		`<, >, >=, <=, ==, !=`	
Условные			`x ? y : z`
Битовые	`~`	`&, |, ^, >>, <<`	
Присваивание	`++, --`	`=, +=, -=, *=, /=, %=`	
Последовательность выражений		, (запятая)	
Преобразование типа	`(тип)`		

Арифметические операции

Класс	Опер.	Описание	Пример
Унарная	`-`	Изменение знака	`int k = -z;`
	`+`	Изменение знака (исп. редко)	`-`
Бинарная	`+`	Аналогичны математическим операциям	`z = a + b;`
	`-`		`z = a – b;`
	`*`		`z = a * b;`
	`/`	Деление (для целых – целая часть от деления)	`z = a / b;`
	`%`	остаток от деления (только целые)	`z = a % b;`

Пример для операций деления:

```
1  int i, j, k = 5, l = 2;
2  i = k / l; // i == 2
3  j = k % l; // j == 1, k = (i * l) + j
```

Операции сравнения

Класс	Опер.	Описание	Пример
Бинарная	<	Меньше	a < b
	>	Больше	a > b
	>=	Больше или равно	a >= b
	<=	Меньше или равно	a <= b
	==	Равно	a == b
	!=	Не равно	a != b

a	b	a>b	a<b	a>=b	a<=b	a==b	a!=b
10	10	0	0	1	1	1	0
10	5	1	0	1	0	0	1
5	10	0	1	0	1	0	1

Логические операции

Класс	Опер.	Описание	Пример
Унарная	`!`	Логическое НЕ	`z = !a;`
Бинарная	`&&`	Логическое И	`z = a && b;`
	`||`	Логическое ИЛИ	`z = a || b;`

Для хранения значений ИСТИНА и ЛОЖЬ используются целый тип:

- ИСТИНА ~ **любому ненулевому значению**, например 1, 10, -5;
- ЛОЖЬ ~ **нулевому значению**: 0.

a	b	!a	a && b	a || b
1	1	0	1	1
1	0	0	0	1
0	1	1	0	1
0	0	1	0	0
10	5	0	1	1

Логические операции (вопросы)

Опер.	Описание
`!`	Логическое НЕ
`&&`	Логическое И
`||`	Логическое ИЛИ

a	b	!a	a && b	a || b
1	1	0	1	1
1	0	0	0	1
0	1	1	0	1
0	0	1	0	0

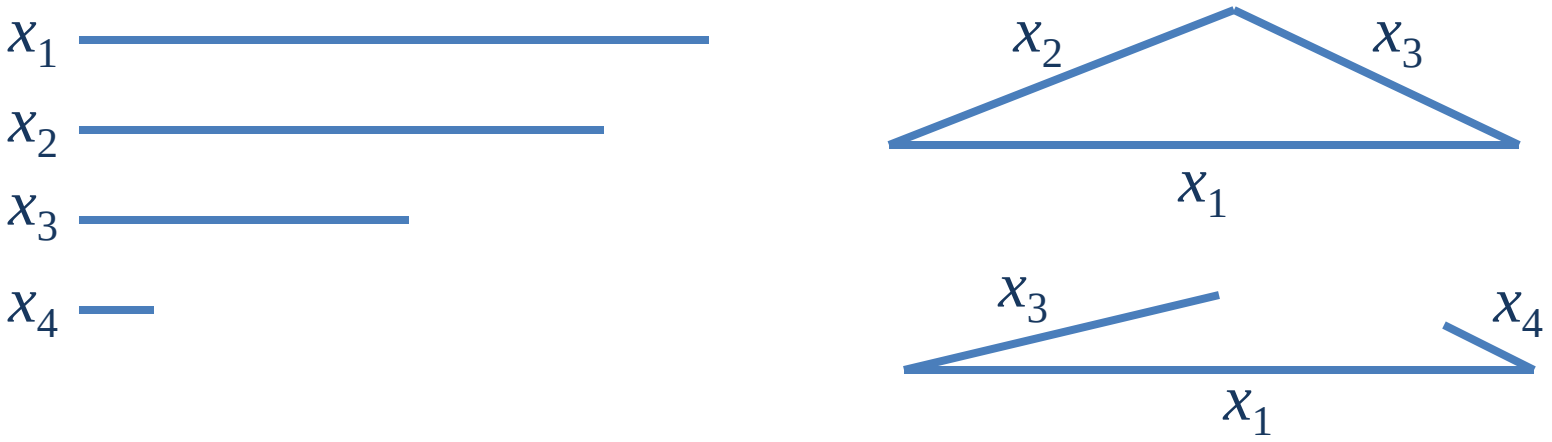
Даны длины отрезков a, b и c. Записать логическое выражение, которое будет истинно, если :

1) из данных отрезков можно построить треугольник (правило треугольника: сумма двух любых сторон больше третьей);
2) из данных отрезков можно построить равнобедренный треугольник;
3) из данных отрезков можно построить прямоугольный треугольник.

Логические операции (ответы)

Даны длины отрезков a, b и c. Записать логическое выражение, которое будет истинно, если из данных отрезков можно построить треугольник (правило треугольника: сумма двух любых сторон больше третьей);

(a+b) > c && (a+c) > b && (b+c) > a



Логические операции (ответы)

Даны длины отрезков a, b и c. Записать логическое выражение, которое будет истинно, если из данных отрезков можно построить равнобедренный треугольник.

((a == b) && 2*a > c) || ((a == c) && 2*a > b) ||

((b == c) && 2*b > a)

Если $a = b$, то какой бы ни была длина сторона стороны с $(a + c)$ всегда больше b и наоборот $(b + c) > a$. Поэтому достаточно проверить выполнение неравенства $(a + b) < c$, а так как $a = b$, то $(a + b) = (a + a) = 2*a$.

Аналогичные рассуждения применяются

Логические операции (ответы)

Даны длины отрезков a, b и c. Записать логическое выражение, которое будет истинно, если из данных отрезков можно построить равнобедренный треугольник.

((a == b) && 2*a > c) || ((a == c) && 2*a > b) ||

((b == c) && 2*b > a)

Логические операции (ответы)

Даны длины отрезков a, b и c. Записать логическое выражение, которое будет истинно, если из данных отрезков можно построить прямоугольный треугольник.

((a*a + b*b == c*c) || ((a*a + c*c == b*b) ||

((c*c + b*b == a*a)

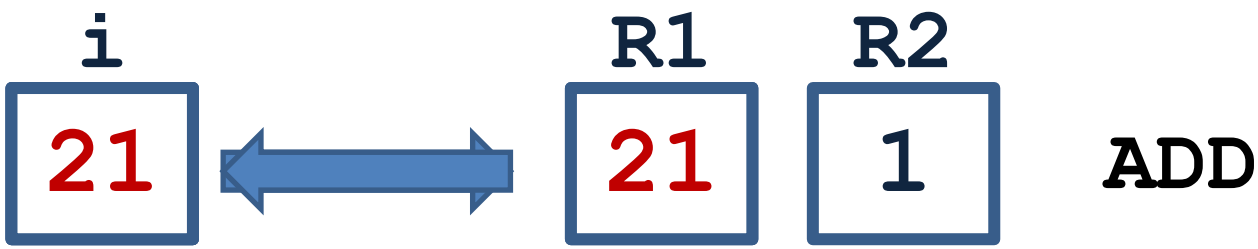
Операции присваивания

<table>
<tr><th>Класс</th><th>Опер.</th><th>Описание</th><th>Пример</th></tr>
<tr><td rowspan="4">Унарная</td><td rowspan="2">++</td><td>префиксный инкремент</td><td>++i;</td></tr>
<tr><td>постфиксный инкремент</td><td>i++;</td></tr>
<tr><td rowspan="2">--</td><td>префиксный декремент</td><td>--i;</td></tr>
<tr><td>префиксный декремент</td><td>i--;</td></tr>
<tr><td rowspan="7">Бинарная</td><td>=</td><td>присваивание</td><td>i = j;</td></tr>
<tr><td>+=</td><td rowspan="6">присваивание вида:
a <OP>= b;
трактуется как
a = a <OP> b;
например:
a += b; эквив. a = a + b;</td><td>i += j;</td></tr>
<tr><td>-=</td><td>i -= j;</td></tr>
<tr><td>*=</td><td>i *= j;</td></tr>
<tr><td>/=</td><td>i /= j;</td></tr>
<tr><td>%=</td><td>i %= j;</td></tr>
<tr><td>...</td><td>...</td></tr>
</table>

Операции присваивания (примеры)

Инкремент – увеличение на единицу, **декремент** – уменьшение.

```
1 int i = 20;
2 i++;
```



Префиксная запись предусматривает использование **измененного значения**, а постфиксная – **исходного**.

Например:

```
int i = 10, j, k;
j = ++i; // i = 11, j = 11
k = i++; // i = 12, k = 11
```

Операции присваивания (вопросы)

Какие результаты будут распечатаны при выполнении приведенной ниже программы:

```
int i=2, j, k=3;
j = ++i;
printf("i=%d, j=%d\n", i, j);
k += i+++j++;
printf("i=%d, j=%d, k = %d\n", i, j, k);
j -= k--- --i;
printf("i=%d, j=%d, k = %d\n", i, j, k);
```

Операции присваивания (ответы)

Какие результаты будут распечатаны при выполнении приведенной ниже программы:

```
int i=2, j, k=3;
j = ++i;
printf("i=%d, j=%d\n", i, j);
```

i=3, j=3

```
k += i+++j++;
printf("i=%d, j=%d, k = %d\n", i, j, k);
```

i=4, j=4, k = 9

```
j -= k--- --i;
printf("i=%d, j=%d, k = %d\n", i, j, k);
```

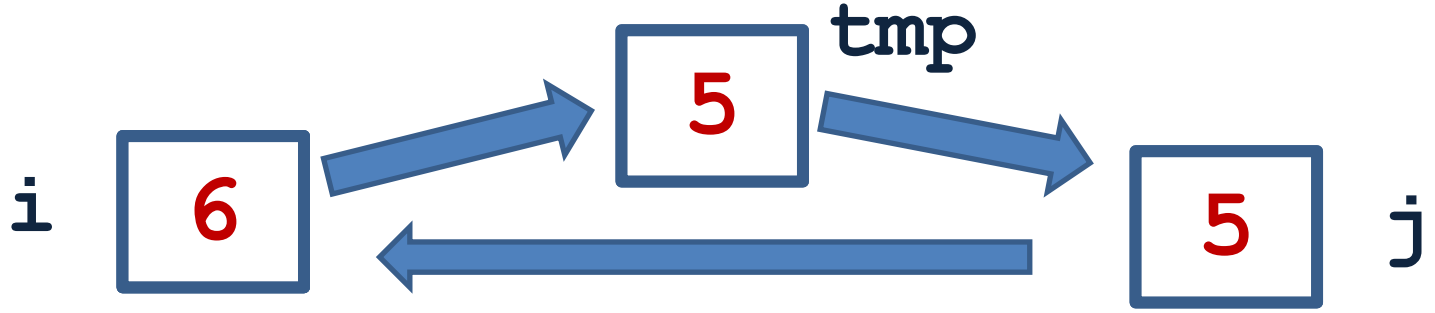
i=3, j=-2, k = 8

Обмен значений двух ячеек памяти

Часто встречающаяся задача в программировании – обменять значения двух переменных.

Для решения этой задачи можно использовать вспомогательную переменную:

```
1 int i = 5, j = 10, tmp;
2 tmp = i;
3 i = j;
4 j = tmp;
```

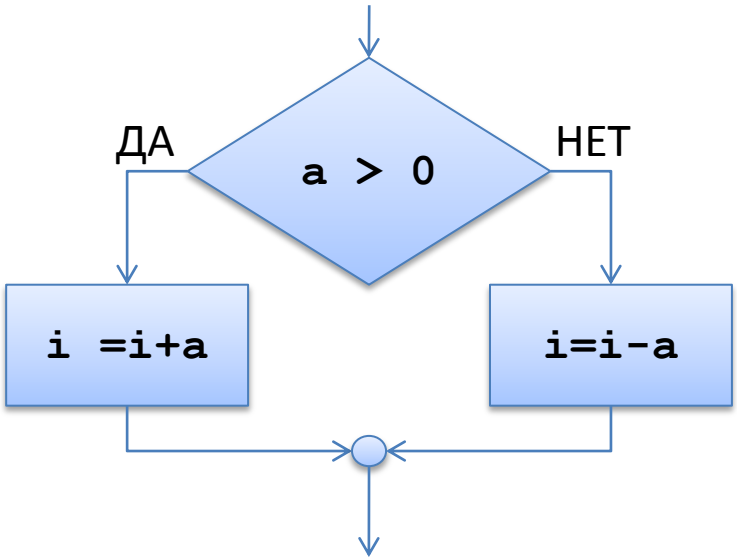


Условная операция

Единственная тренарная операция в языке Си. Реализует упрощенную запись конструкции ветвления. Например:

```
i = (a>0) ? i + a : i – a; // i = i + |a|
```

Данная запись эквивалентна следующей конструкции ветвления:



Читать следующим образом:
Если a больше нуля, то увеличить значение **i** на значение **a**. В противном случае – уменьшить значение **i** на значение **a**.

Лексемы языка СИ (дубл. слайд)

Из символов алфавита составляются лексемы - минимальные единицы языка, имеющие самостоятельный смысл:

- константы;
- имена объектов;
- ключевые слова;

} **Идентификаторы**

- знаки операций;
- разделители (скобки, точка, запятая, пробельные символы).

Вычисление суммы двух целых чисел

```
#include <stdio.h>

int main()
{
    int a = 5, b = 10, c;

    c = a + b;
    printf("a + b = %d\n", c );

    return 0;
}
```

Вычисление суммы двух целых чисел (лексемы - константы)

```
#include <stdio.h>

int main()
{
    int a = 5, b = 10, c;

    c = a + b;
    printf("a + b = %d\n", c );

    return 0;
}
```

Вычисление суммы двух целых чисел (лексемы - идентификаторы)

```
#include <stdio.h>

int main()
{
    int a = 5, b = 10, c;

    c = a + b;
    printf("a + b = %d\n", c );

    return 0;
}
```

Вычисление суммы двух целых чисел (лексемы - операции)

```
#include <stdio.h>

int main()
{
    int a = 5, b = 10, c;

    c = a + b;
    printf("a + b = %d\n", c );

    return 0;
}
```

Вычисление суммы двух целых чисел (лексемы - разделители)

```
#include <stdio.h>

int main()
{
    int a = 5, b = 10, c;

    c = a + b;
    printf("a + b = %d\n", c );

    return 0;
}
```

Состав языка СИ

Выражения

Выражения строятся из **первичных выражений** и **знаков операций**.

Первичные выражения вычисляются в первую очередь, они имеют наивысший приоритет. К первичным выражениям относятся:

- идентификатор;
- лексическая константа;
- выражение в скобках;
- оператор sizeof .

((a + b) / c – 0x21) / (z*z)

Описание конструкций языка

- Под синтаксисом языка программирования понимают правила построения корректных конструкций данного языка.
- Наиболее распространенными способами описания синтаксиса языков программирования являются:
  - форма Бэкуса-Наура (БНФ);
  - расширенная форма Бэкуса-Наура (РБНФ).

В рамках данного курса для описания синтаксиса будет использоваться

расширенная форма Бэкуса-Наура (РБНФ).

Символы РБНФ

- Служебные слова языка, разделители и операции называются **терминальными символ** и записываются в кавычках. Из РБНФ они переносятся в программу без изменений.

Например, **"int"** – имя целочисленного типа данных.

- Конструкции языка называются **нетерминальными символами** и записываются слитно русскими и латинскими буквами.

Например, **Идентификатор**.

Правила РБНФ

Правило РБНФ имеет следующий вид:

ИмяСимвола = выражение.

где **ИмяСимвола** — имя нетерминального символа, а выражение — комбинация терминальных и нетерминальных символов и специальных знаков, удовлетворяющая требованиям РБНФ.
Точка в конце — специальный символ, указывающий на завершение правила.

Выражения РБНФ

A = B|C|D. обозначает, что символ A может принимать значения B, либо C, либо D.

- **[A]** определяет, что выражение **A** может повторяться 0 или 1 раз ("AB"["C"] допускает AB или ABC);
- **{A}** определяет, что выражение **A** может повторяться 0 или множество раз. Например, "AB"{"C"} допускает конструкции AB, ABC, ABCC и т.д.
- **(A)** используются для группировки выражений. Например A = ("B"|"C")("D"|"E") допускает наличие следующих конструкций: BD, BE, CD, CE.

Выражения РБНФ (2)

A = B|C|D. обозначает, что символ A может принимать значения B, либо C, либо D.

- **[A]** определяет, что выражение **A** может повторяться 0 или 1 раз ("AB"["C"] допускает AB или ABC);
- **{A}** определяет, что выражение **A** может повторяться 0 или множество раз. Например, "AB"{"C"} допускает конструкции AB, ABC, ABCC и т.д.
- **(A)** используются для группировки выражений. Например A = ("B"|"C")("D"|"E") допускает наличие следующих конструкций: BD, BE, CD, CE.

Выражения РБНФ (примеры)

Идентификатор представляет собой последовательность строчных, прописных букв латинского алфавита, цифр и знака подчеркивания. Причем идентификаторы должны начинаться либо с буквы, либо со знака подчеркивания.

Эквивалентное определение идентификатора с помощью РБНФ:

```
Идентификатор =
    (Буква | "_"){ Буква | Цифра | "_" }
Буква =
"A"|"B"|...|"Y"|"Z"|"a"|"b"|...|"y"|"z"
Цифра = "0"|"1"|...|"9"
```

Выражения РБНФ (вопрос)

```
Идентификатор =
    (Буква | "_"){ Буква | Цифра | "_" }
Буква =
"A"|"B"|...|"Y"|"Z"|"a"|"b"|...|"y"|"z"
Цифра = "0"|"1"|...|"9"
```

Проверить, соответствуют ли следующие строки правилу РБНФ:

1. `i`
2. `Test_string`
3. `~test`
4. `test$`
5. `i`
6. `my-var`
7. `____aaa`
8. `1test`

Выражение языка СИ (РБНФ)

```
Выражение =
   ( Выражение { БинарнОперация, Выражение } ) |
   ( ЛевыйОперанд, Присваивание, Выражение ) |
   Операнд.

Операнд = [ ПрефОп ] Переменная | Переменная ПостфОп |
    [ ПрефОп1 ] ( Константа | sizeof(Тип) ) |
    [ ПрефОп1 ] "(" Выражение ")".
ЛевыйОперанд = Переменная | АдресноеВыражение.
ПрефОп1 = + | - | ! | ~ | (тип).
ПрефОп = ПрефикснОп1 | ++ | --.
ПостфОп = ++ | --.
```

БинарнОперация – все бинарные операции кроме группы присваивания.

Присваивание – все операции присваивания.

Выражения (примеры)

```
Выражение =
  (Выражение {БинарнОперация, Выражение})|...| Операнд.
Операнд = [ ПрефОп ] Переменная | Переменная ПостфОп |
    [ ПрефОп1 ] ( Константа | sizeof(Тип) ) |
    [ ПрефОп1 ] "(" Выражение ")".
ПрефОп1 = + | - | ! | ~ | (тип).
ПрефОп = ПрефикснОп1 | ++ | --.
ПостфОп = ++ | --.
```

БинарнОперация – все бинарные операции кроме группы присваивания.

ЛОГИЧЕСКОЕ ВЫРАЖЕНИЕ возвращает значение ИСТИНА (1) или ЛОЖЬ (0)

```
1) y + 1
2) -(y + 1)
3) 1++
4) (y + 1)++
5) (y + 1)*(z-1)-k
6) i++ + ++j
7) (k == a + b*c)
9) (a < b) && (a < c)
```

Выражения (примеры)

```
Выражение = (Выражение { БинарнОперация, Выражение })|
  (ЛевыйОперанд, Присваивание, Выражение) | Операнд.
Операнд = [ ПрефОп ] Переменная | Переменная ПостфОп |
    [ ПрефОп1 ] ( Константа | sizeof(Тип) ) |
    [ ПрефОп1 ] "(" Выражение ")".
ЛевыйОперанд = Переменная | АдресноеВыражение.
ПрефОп1 = + | - | ! | ~ | (тип).
ПрефОп = ПрефикснОп1 | ++ | --.
ПостфОп = ++ | --.
```

БинарнОперация – все бинарные операции кроме группы присваивания.

Присваивание – все операции присваивания.

```
1) x = y + 1;
2) x = -(y + 1);
3) 1 = x;
4) (y + 1) = 12;
5) x = (y+1)*(z-1)-k;
```

```
6) x *= i++ + ++j;
7) x += (a<b) && (a<c);
8) x += j -= (k=a+b*c);
9) x = (a*=2,a++,a+z);
```

Приоритеты и ассоциативность

Группы по убыванию приоритета (сверху – наивысший)

Операция	Ассоциативн.
`++, --`	Слева направо
`++, --, +, -, !, ~, (тип), *, &, sizeof`	**Справа налево**
`*, /, % (бинарные)`	Слева направо
`+, - (бинарные)`	Слева направо
`<<, >>`	Слева направо
`<, <=, >, >=`	Слева направо
`== !=`	Слева направо
`&`	Слева направо
`^`	Слева направо
`|`	Слева направо
`&&`	Слева направо
`||`	Слева направо
`?:`	**Справа налево**
`=, +=, -=, *=, /=, ...`	**Справа налево**
`,`	Слева направо

Ассоциативность слева направо

Ассоциативность внутри одной группы

Операция	Ассоциативность
`*, /, % (бинарные)`	Слева направо

Например:

```
i * j / k % s  <->  ( (i * j) / k ) % s
```

Ассоциативность и приоритеты

Операция	Ассоциативность
`++, --`	Слева направо
`*, /, % (бинарные)`	Слева направо

Например:

```
++i * j / ++k % s ~ ( ((++i) * j)/(++k) ) % s
```

Ассоциативность справа налево

Операция	Ассоциативн.
`=, +=, -=, *=, /=, ...`	**Справа налево**

Например:

```
1 int i = 5, j = 12 , k;
2 i += j -= k = 2;
```

В данном примере строка 2 эквивалентна следующей программе:

```
1 k = 2;
2 j -= k;
3 i += j;
```

Различные приоритеты и ассоциативность

Операция	Ассоциативн.
`*, /, % (бинарные)`	Слева направо
`+, - (бинарные)`	Слева направо
`<, <=, >, >=`	Слева направо
`=, +=, -=, *=, /=, ...`	**Справа налево**

```
1 int i= 5 , j, k, z = 1;
2 i += j = 5 + (k = 2) * (z < 5);
```

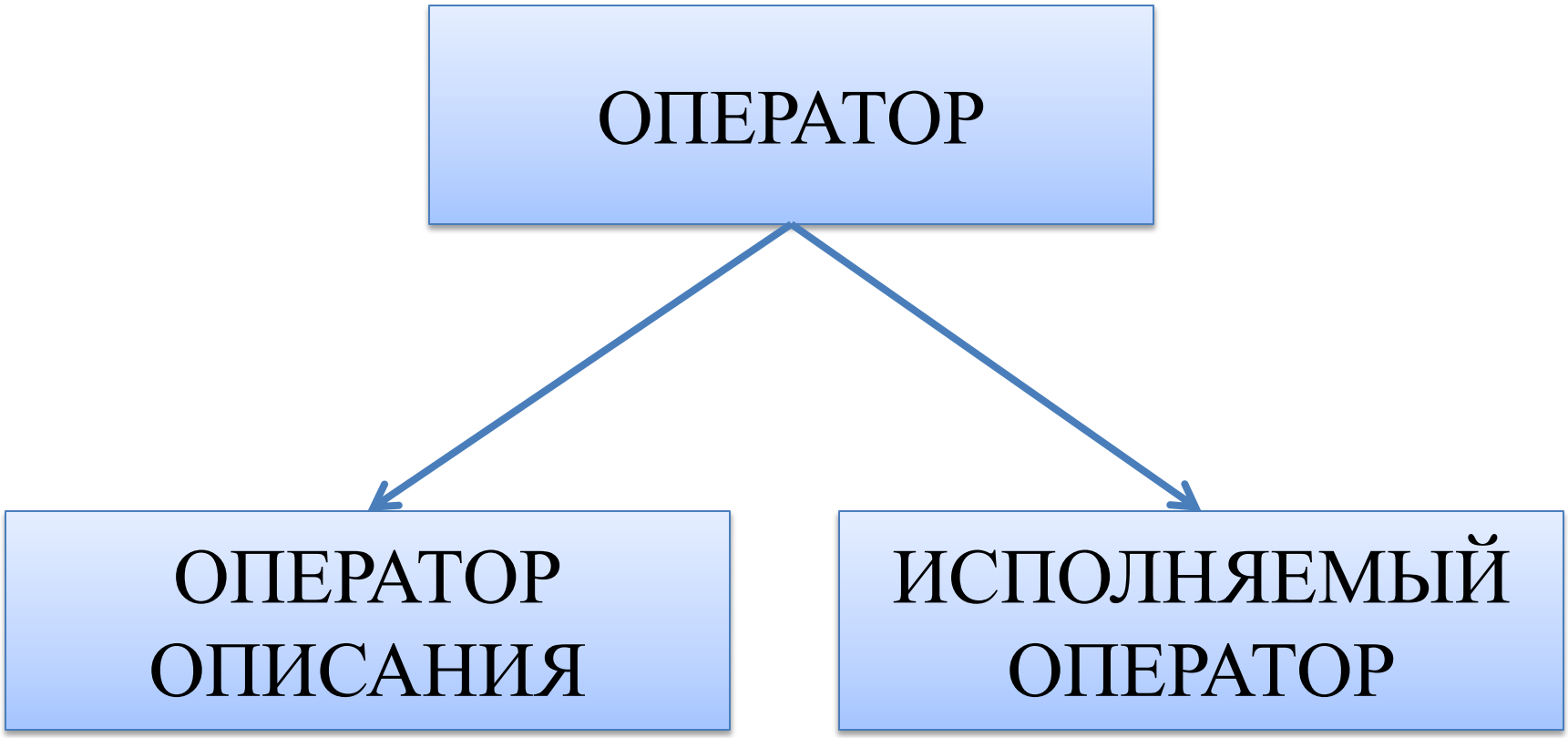
№	Действие	Результат	№	Действие	Результат
1	k = 2	2	4	5 + 2	7
2	z < 5	1	5	j = 7	7
3	2 * 1	2	6	i += 7	12

Состав языка СИ

Естественный язык	Язык программирования
Алфавит	Символы
Слова	Лексемы
Словосочетания	Выражения
Предложения	Операторы

Операторы в языке СИ

Выражение, оканчивающееся точкой с запятой, является оператором, т.е. наименьшей элементарной частью программы.



Операторы описания

Оператор описания позволяет определить данные, над которыми в программе выполняются действия.

```
ОператорОписания = Тип " " Объект {"," Объект} ";".
Объект = Идентификатор ["=" Выражение].
Тип = ЦелыйТип | ВеществТип | СложнТип.
ЦелыйТип = ["unsigned"]("char"|"short"|"int"|"long").
ВеществТип = "float"| ["long"] "double".
```

Инициализация

```
1) int i;
2) char z; x; y;
3) int j = 5, m = 8, k = 10;
4) short (y + 1) = 12;
5) long double k, n, q;
```

Операторы описания (2)

```
ОператорОписания = Тип " " Объект {"," Объект} ";".
Объект = Идентификатор ["=" Выражение].
Тип = ЦелыйТип | ВеществТип | СложнТип.
ЦелыйТип = ["unsigned"]("char"|"short"|"int"|"long").
ВеществТип = "float"| ["long"] "double".
```

```
1  int i;
2  float j, k = 5;
3  j=k*3;
```

Примером описания строки **1** на естественном языке может служить предложение:
"В памяти следует отвести область для хранения целого числа, для обращения к данной области будет использоваться имя i".

Целочисленные типы данных

Знаковые целые

Идентификатор	Размер, байт	Диапазон значений
`char`	1	[-128; 127]
`short`	2	[-32768; 32767]
`int`	4	[-2147483648; 2147483647]
`long`	4 или 8	$[-2^{63}; 2^{63} - 1]$ (8 байт)

Беззнаковые целые

Идентификатор	Размер, байт	Диапазон значений
`unsigned char`	1	[0; 255]
`unsigned short`	2	[0; 65535]
`unsigned int`	4	[0; 4294967295]
`unsigned long`	4 или 8	$[0; 2^{64} - 1]$ (8 байт)

Целочисленные типы данных (примеры)

Знаковые целые

```
1  int i;
2  char c;
3  short z = 10;
4  long l;
```

Инициализация

Беззнаковые целые

```
1  unsigned int i;
2  unsigned char c;
3  unsigned short z = 10;
4  unsigned long l;
```

Вещественные типы данных

Идентификатор	Размер, байт	Диапазон значений
`float`	4	от $3.4 \cdot 10^{-38}$ до $3.4 \cdot 10^{38}$ (~ 7 значащих цифр)
`double`	8	от $1.7 \cdot 10^{-308}$ до $1.7 \cdot 10^{308}$ (~ 15 значащих цифр)
`long double`	12	от $1.2 \cdot 10^{-4932}$ до $1.2 \cdot 10^{4932}$ (~ 30 значащих цифр)

Вещественные типы данных (примеры)

```
1  float f;
2  double d = 1.5, d1 = 10.1245;
3  long double ld, ld1, _ld12bytes;
4  f = 1.8;
5  d = f + d1;
```

Исполняемый оператор следования

```
Оператор = ( ОператорПрисв | ВызовФункции) ";" |
{ "{" {ОператорОписания} Оператор {Оператор} "}"}.
ОператорПрисв = (++|--)Переменная | Переменная(++|--)
    | ЛевыйОперанд, Присваивание, Выражение.
```

Например:

В начале блока операторов могут находиться операторы описания

```
x = 10;
y = x + 15 * x;

{
  int x = 10;
  y = x + 15 * x;
}
```

```
{
  x = 10;
  {
    y = x + 15 * x;
    z = 10;
  }
}
```

Блок операторов

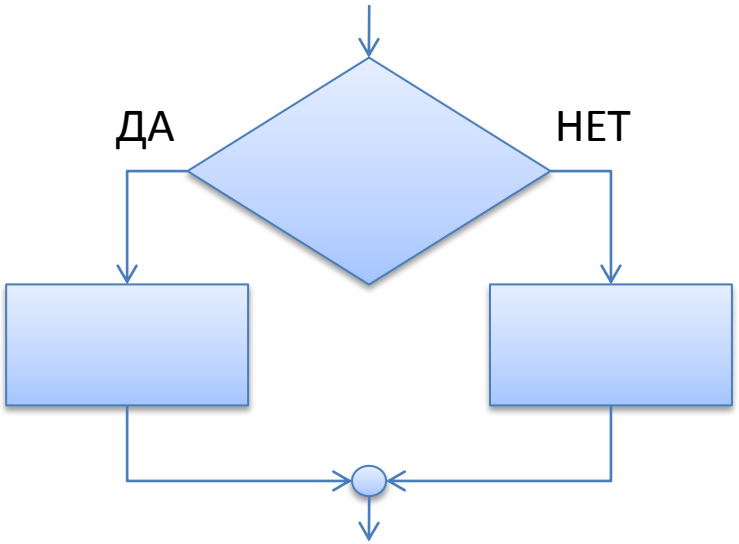
Несколько операторов может быть заключено в фигурные скобки. В этом случае они образуют блок операторов. С точки зрения синтаксиса блок рассматривается как один (составной) оператор.

Группировка операторов необходима для того, чтобы указать компилятору:

- какие операторы принадлежат функции (например **`main`**);
- какие операторы следует выполнять в случае выполнения (**`if`**) или невыполнения (**`else`**) некоторого условия в условном операторе.
- какие операторы формируют тело цикла (операторы **`for`**, **`while`**, **`do-while`**).

Оператор ветвления (условный оператор)

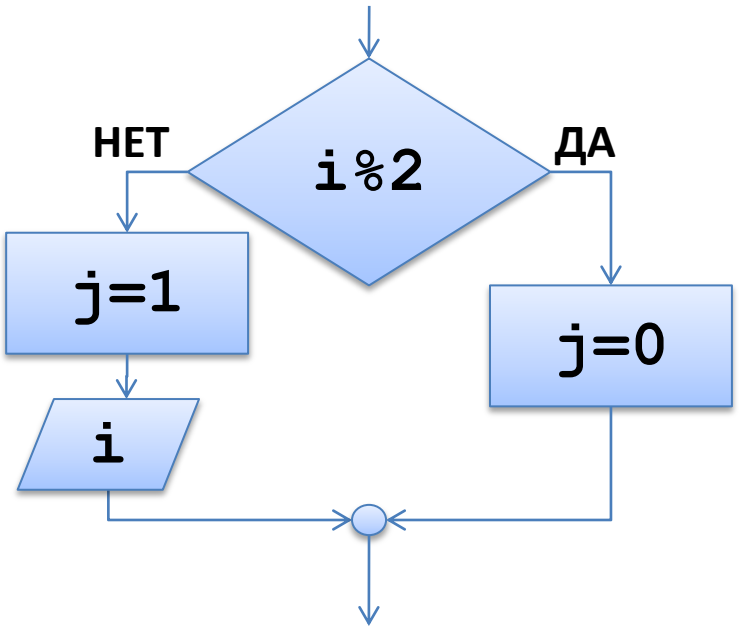
```
УсловныйОператор =
     "if" "(" Выражение ")"
        Оператор1
     [ else
        Оператор2 ].
```



Выражение рассматривается как логическое:
нулевое значение – ЛОЖЬ,
иначе – ИСТИНА
Оператор1 выполняется если **Выражение** ИСТИННО,
иначе – **Оператор2**.

Настройка флага четности

```
УсловныйОператор =
     "if" "(" Выражение ")"
        Оператор1
     [ else
        Оператор2 ].
```

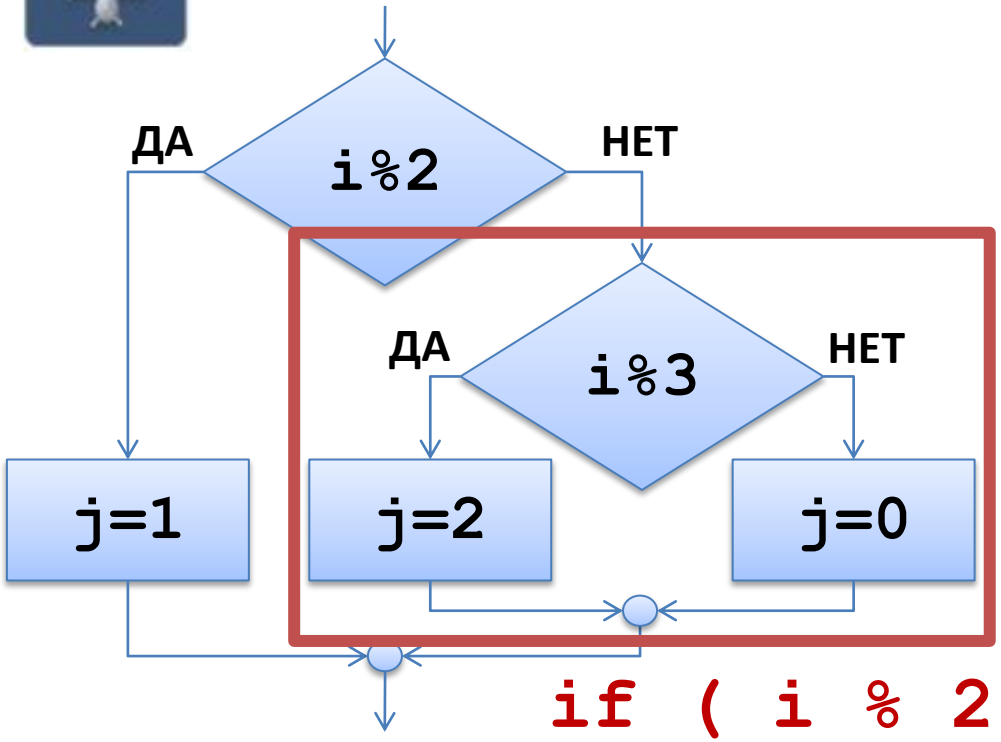



```
if ((i % 2) == 0){
   j = 1;
   printf("i-четное");
} else
   j = 0;
```

Оператор1 данном случае составной и содержит 2 простых оператора.

Оператор2 – простой.

Вложение условных операторов

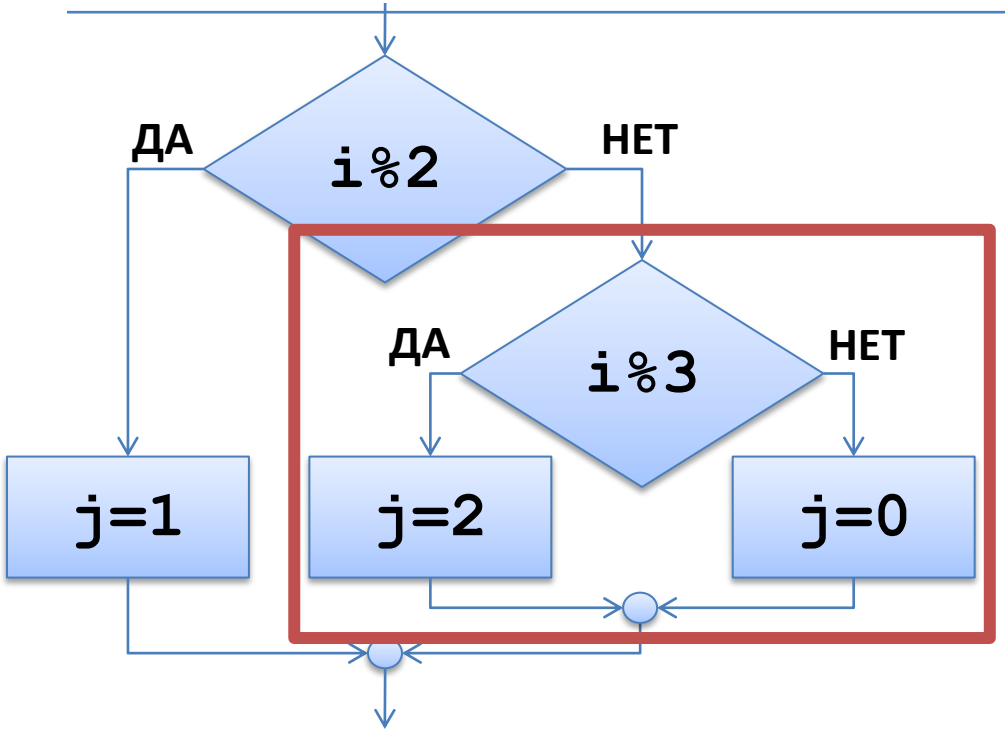



```
if ( i % 2 ){
   j = 1;
} else{
   if( i % 3 )
      j = 2;
   else
      j = 0;
}
```

```
if ( i % 2 )
   j = 1;
else if( i % 3 )
      j = 2;
else
      j = 0;
```

Вложение условных операторов

```
УсловныйОператор =
      "if" "(" Выражение ")"
         Оператор1
      [ else
         Оператор2 ].
```



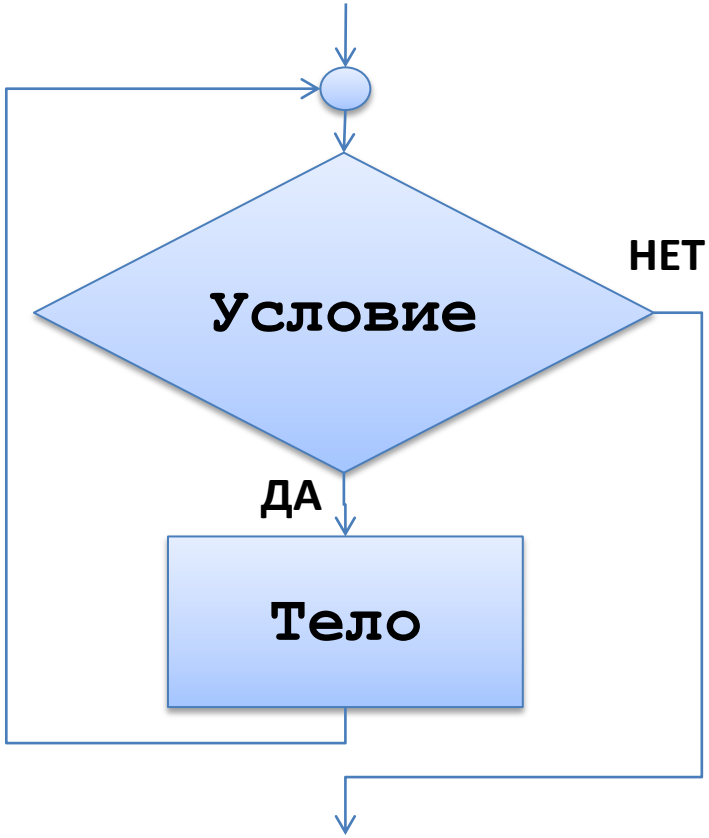
Проверить программу по РБНФ

```
if ( i % 2 )
   j = 1;
else if( i % 3 )
     j = 2;
else
     j = 0;
```

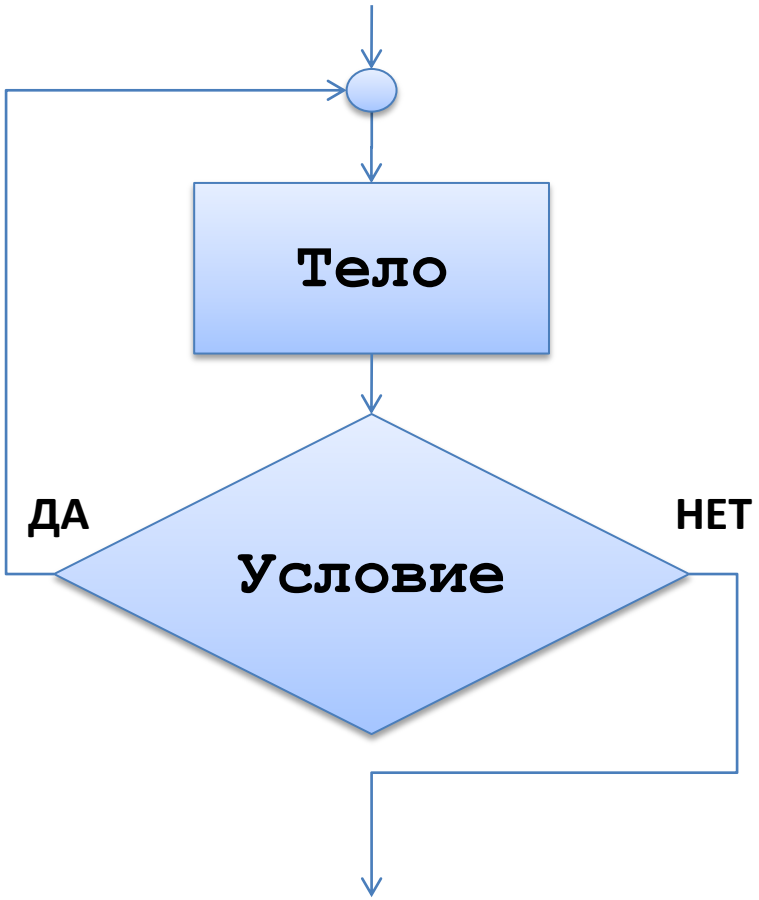
Оператор присваивания внутри оператора ветвления

```
1   if ( (D = b*b - 4*a*c) < 0 ){
2     printf("Действительных корней нет\n");
3   }else if( D == 0 ){
4     printf("Корень один: %f\n", -b/(2*a) );
5   }else{
6     float x1 = (-b - sqrt(D) )/(2*a);
7     float x2 = (-b + sqrt(D) )/(2*a);
8     printf("Корней два: %f, %f\n", x1, x2);
9   }
```

Циклические конструкции



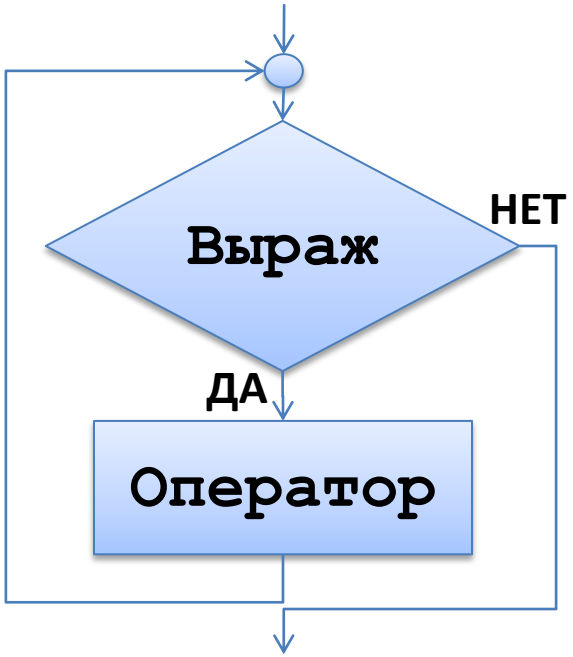
Цикл с предусловием
(**while**, **for**)



Цикл с постусловием
(**do-while**)

Цикл while

```
ОператорWhile =
    "while" "(" Выраж ")" Оператор.
```



Выражение – условие продолжения цикла. Рассматривается как логическое:
нулевое значение – ЛОЖЬ,
иначе – ИСТИНА

Оператор выполняется до тех пор, пока **Выражение** является ИСТИННЫМ.

Цикл while (бесконечный цикл)

Оператор **while** может быть использован для организации бесконечного цикла. Такая задача возникает при разработке, например серверных программ.

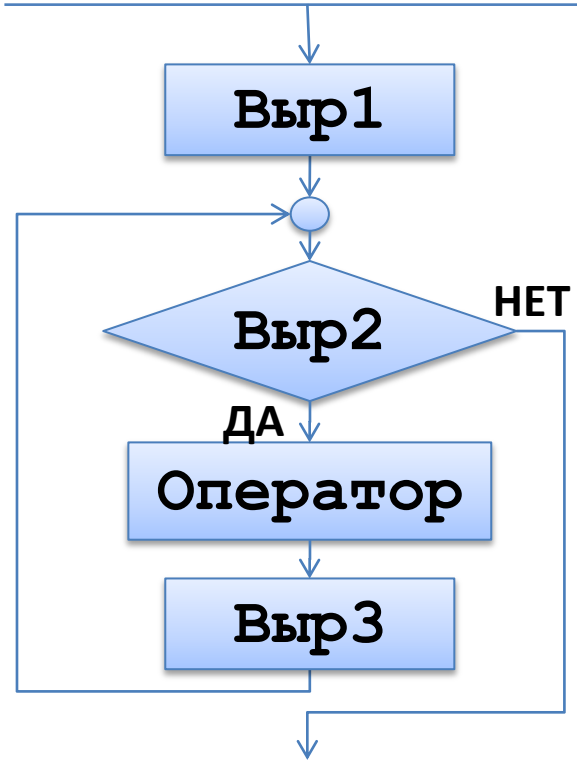
Для организации бесконечного цикла достаточно в качестве **Выражения** указать ненулевую константу, например – 1.

Структура бесконечного цикла на основе оператора **while** приведена ниже:

```
1  while( 1 ){
2     ТелоЦикла
3  }
```

Цикл for

```
ОператорFor =
 "for" "(" [Выр1] ";" [Выр2] ";" [Выр3] ")"
    Оператор.
```



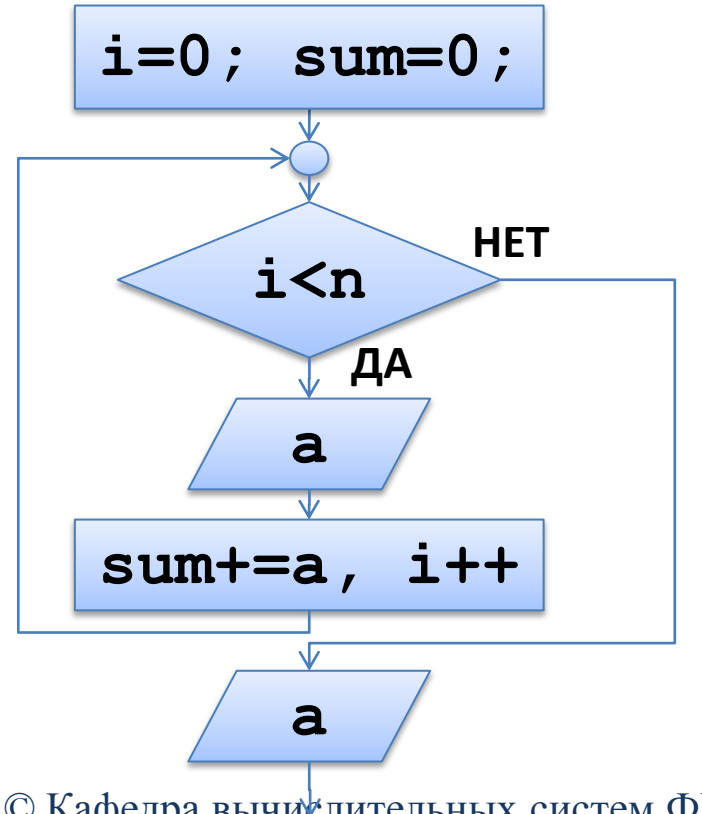
Выр1 выполняется ОДИН РАЗ до начала цикла
Выр2 – условие продолжения цикла, рассматривается как логическое
Выр3 выполняется ПОСЛЕ КАЖДОЙ итерации.

Оператор выполняется до тех пор, пока **Выр2** является ИСТИННЫМ.

Цикл for
(сумма n входных значений)

ОператорFor =

"for" "(" [Выр1] ";" [Выр2] ";" [Выр3] ")"
Оператор.




```
1  int n, i, sum, a;
2  scanf("%d",&n);
3  for(i=0,sum=0; i<n; i++){
4     scanf("%d",&a);
5     sum += a;
6  }
7  printf("sum = %d\n", sum);
```

Цикл for (особенности)

```
ОператорFor =
 "for" "(" [Выр1] ";" [Выр2] ";" [Выр3] ")"
 Оператор.
```

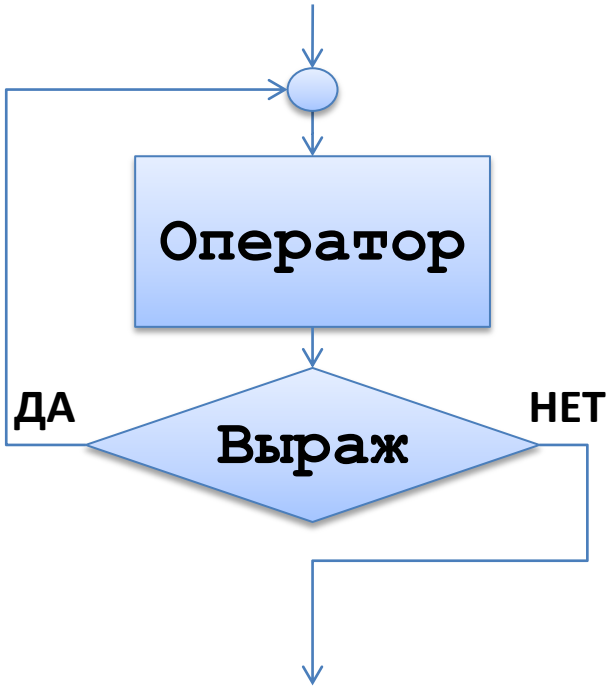
Любое управляющее выражение цикла **for** (**Выр1, Выр2, Выр3**) может отсутствовать! Наличие ";" обязательно!
В случае отсутствия **Выр2** считается, что условие продолжения цикла всегда ИСТИННО!

Пример бесконечного цикла на основе конструкции **for**:

```
1  for( ; ; ){
2     ТелоЦикла
3  }
```

Цикл do-while

```
ОператорDoWhile =
"do" Оператор "while" "(" Выраж ")" ";".
```



Оператор выполняется КАК МИНИМУМ один раз

Выраж – условие продолжения цикла, рассматривается как логическое

Оператор выполняется до тех пор, пока **Выраж** является ИСТИННЫМ.

Цикл do-while

```
ОператорDoWhile =
"do" Оператор "while" "(" Выраж ")" ";".
```

```
1    int n, i, sum, a;
2    scanf("%d",&n);
3    if(n > 0){
4      do{
5        scanf("%d",&a);
6        sum += a;
7        i++;
8      }while(i<=n);
9    }
10   printf("sum = %d\n", sum);
```

Вложенность циклов

```
1   int i, j;
2   for(i=1;i<=4;i++){
3     printf("i=%d, j=",i);
4     for(j=1;j<=i;j++){
5       printf("%d ",j);
6     }
7     printf("\n");
8   }
```

Результат работы программы:

```
i=1, j=1
i=2, j=1 2
i=3, j=1 2 3
i=4, j=1 2 3 4
```

ДОПОЛНИТЕЛЬНЫЙ МАТЕРИАЛ

Операции присваивания (побочные эффекты)

Основной эффект операции: вычисление значения
Побочный эффект: изменение объектов.

Рассмотрим более конкретный пример:

```
int i, j;
i = 5 + (j = 2);
```

Основной эффект `j = 2` результат вычисления выражения справа. Он подставляется во внешнее выражение:

```
i = 5 + (j = 2).
i = 5 + 2
```

Побочный эффект - изменение значения ячейки `j`.

Побочные эффекты неопределенное поведение

В стандарте Си между двумя **точками последовательных вычислений** (ТПВ) изменение значения переменной возможно **не более одного раза**.

Пример 1

```
1  y = 4;     // ТПВ №1
2  x = x + y;     // ТПВ №2
3  printf("y=%d,x=%d\n",y,x);
```

Пример 2

```
1   int x=0;    // ТПВ №1
2   x = x++;    // ТПВ №2
3   printf("x = %d\n", x);
```

```
gcc 3.4.4: x = 0
gcc 4.4.5: x = 1
```

http://alenacpp.blogspot.com/2005/11/sequence-points.html
http://cmcmsu.no-ip.info/download/c.expressions.pdf
http://msdn.microsoft.com/en-us/library/8a425116(v=vs.80).aspx

Побочные эффекты переносимость

```
1  int a = 1, b, c;
2  b = (a++)+(++a);
3  printf("a=%d,b=%d\n",a,b);
4  a = 1;
5  c = (a++)+(a++)+(++a);
6  printf("a=%d,c=%d\n",a,c);
```

Результат работы:

Компилятор	MS Visual C 2008	GCC 4.4.5
Результаты	a = 3, b = 4	a = 3, b = 4
	a = 4, c = 6	a = 4, c = 4

http://gcc.gnu.org/bugzilla/show_bug.cgi?id=11751

Побочные эффекты переносимость (2)

```
1    c = (a++)+(a++)+(++a);
```

MS Visual C:

Пусть a = 1

1.++a(a=a+1)	**(a=2)**
2.c=a+a	**(c=4)**
3.c=c+a	**(c=6)**
4.a++ (a=a+1)	**(a=3)**
5.a++ (a=a+1)	**(a=4)**

GCC:

Пусть a = 1

1.c=a+a	**(c=2)**
2.++a (a=a+1)	**(a=2)**
3.c=c+a	**(c=4)**
4.a++ (a=a+1)	**(a=3)**
5.a++ (a=a+1)	**(a=4)**

http://gcc.gnu.org/bugzilla/show_bug.cgi?id=11751

Последовательность выражений операция ","

```
1  int i,j,k;
2  i = 5, j = 3, k = i + j;
3  printf("1: i=%d,j=%d,k=%d\n",i,j,k);
4  k = ++j, i+j;
5  printf("2: i=%d, j=%d, k=%d\n",i,j,k);
6  k = (++j, i+j);
7  printf("3: i=%d, j=%d, k=%d\n",i,j,k);
```

```
$ ./sequence
1: i = 5, j = 3, k = 8
2: i = 5, j = 4, k = 4
3: i = 5, j = 5, k = 10
```

СПАСИБО ЗА ВНИМАНИЕ!