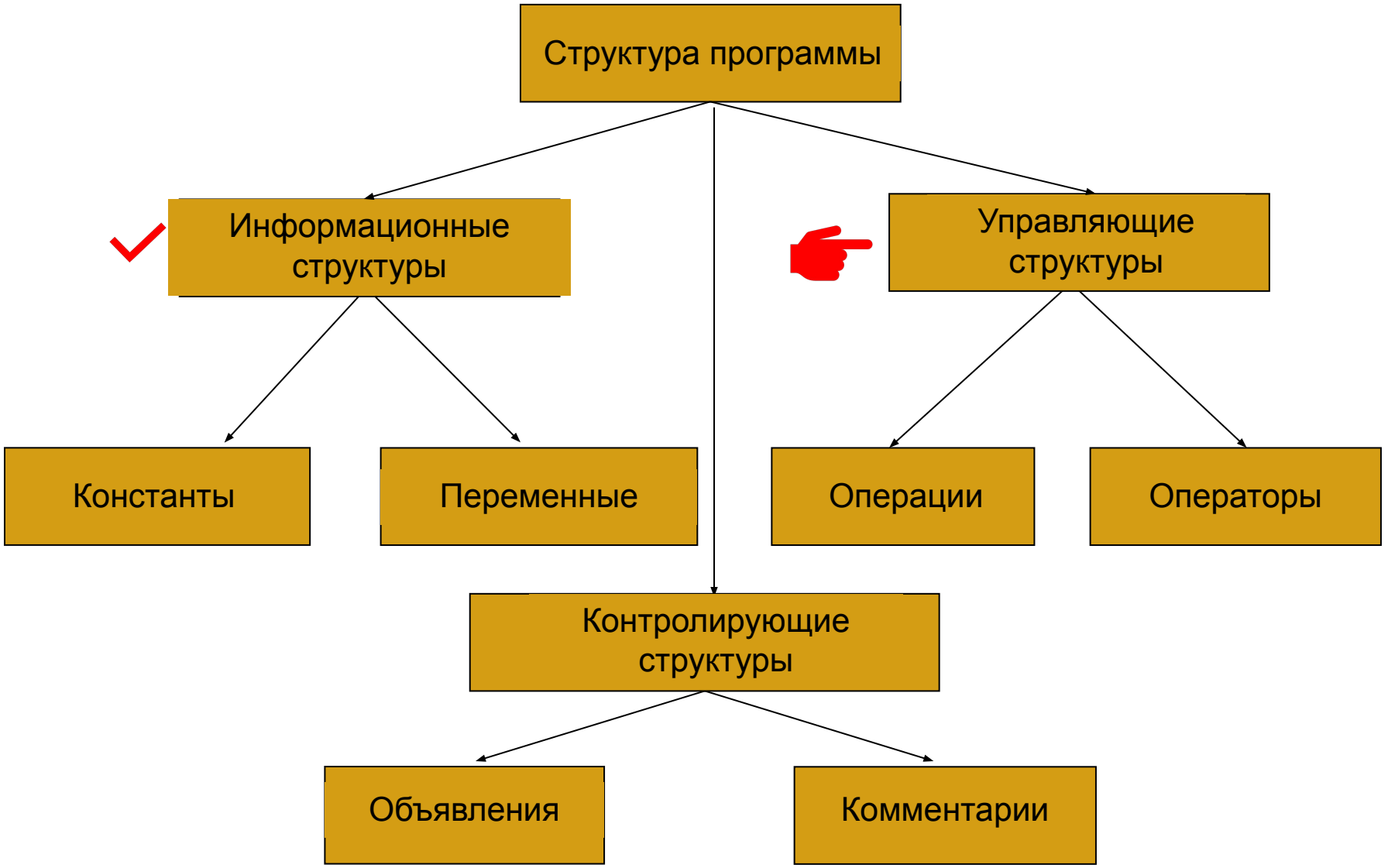
Тема 2-2.

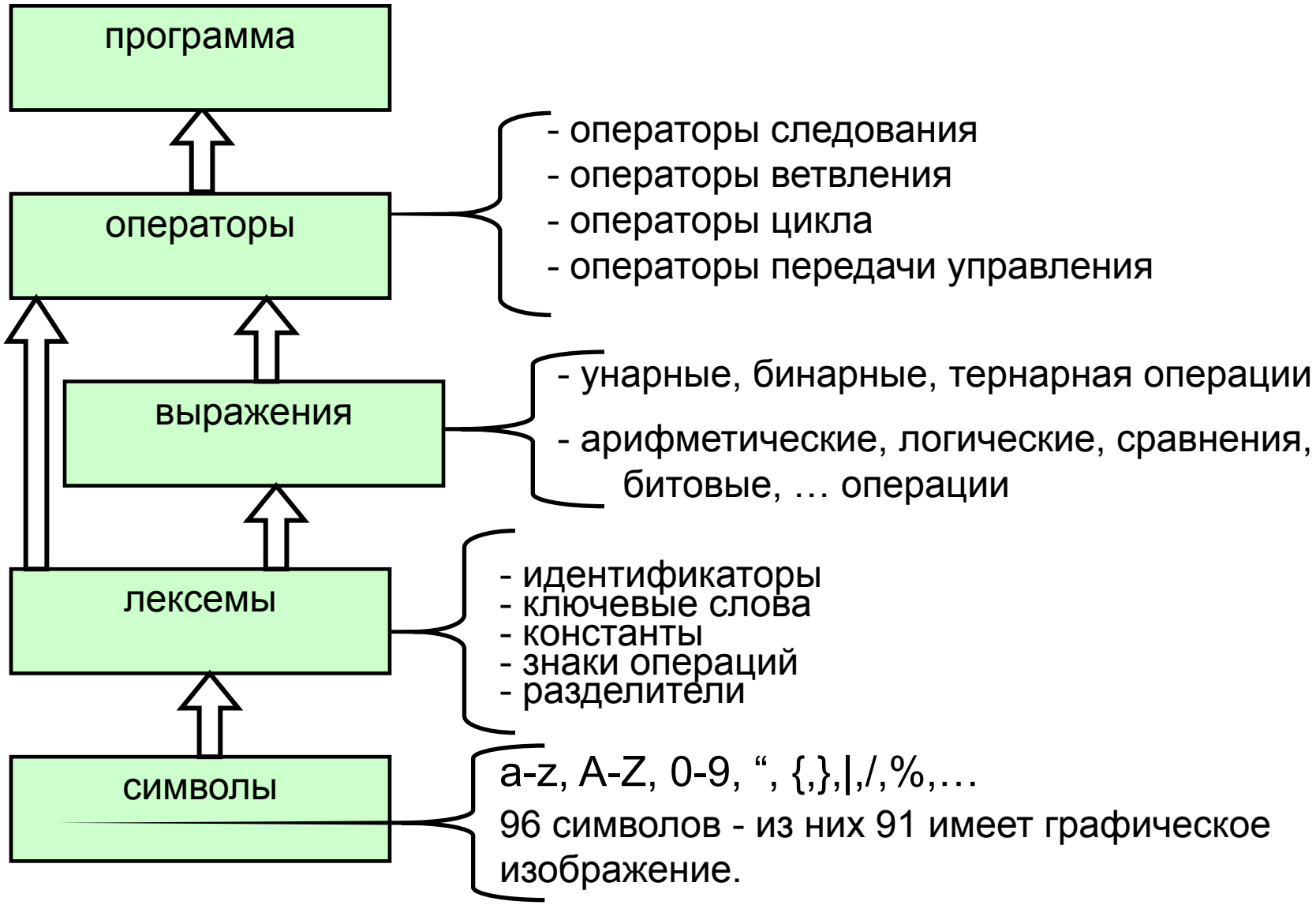
Операции и операторы

для АСУб и ЭВМб

Структурный аспект программы C++



«Языковой» аспект программы C++



Операции и операторы

- Русскому термину «**операция**» соответствует английский — «**operator**», из-за чего возникает путаница: «operator» переводят как «оператор».
- *Оператор* — это наименьшая исполняемая единица программы. Различают операторы выражения, действие которых состоит в вычислении заданных выражений; операторы объявления; составные операторы; пустые операторы; операторы метки; цикла и т. д.
- Для обозначения конца оператора в C/C++ используется символ «;» (в конце составного оператора — набора операторов, помещенных между открывающей ({) и закрывающей (}) фигурными скобками — точка с запятой не ставится).
- Таким образом, операция может быть составной частью оператора, но не наоборот, а русский термин «оператор» обозначает то же, что и «инструкция» (**statement**).

Операции и операторы: примеры

- c = a + b; —это оператор, а sin(c)+d — операция, но не оператор.
- Путаница усугубляется ещё и тем, что в C/C++ **присваивание** и **инкремент/декремент** являются как операторами, так и операциями. Например, a = sin(b) может быть операцией в составе условного оператора
  - if ((a = sin(b)) == 0) {…}
- a может быть самостоятельным оператором:
  - a = sin(b);

Операции: основные 4 типа

Арифметические операции

Операция	Символ	Пример	Операция
Унарный - и +	-	- y	Значение с обратным знаком
Сложение	+	x + y	x плюс y
Вычитание	−	x − y	x минус y
Умножение	*	x * y	x умножить на y
Деление	/	x / y	x разделить на y
Деление с остатком	%	x % y	Остаток от деления x на y

Инкремент и декремент

Операция	Символ	Пример	Операция
Префиксный инкремент (пре-инкремент)	++	++x	Инкремент x, затем вычисление x
Префиксный декремент (пре-декремент)	−−	−−x	Декремент x, затем вычисление x
Постфиксный инкремент (пост-инкремент)	++	x++	Вычисление x, затем инкремент x
Постфиксный декремент (пост-декремент)	−−	x−−	Вычисление x, затем декремент x

Инкремент и декремент

С операциями инкремента/декремента версии префикс всё просто. Значение переменной сначала увеличивается/уменьшается, а затем уже вычисляется.

Например

```
1  int x = 5;
2  int y = ++x; // x = 6 и 6 присваивается переменной y
```

Инкремент и декремент

А вот с операторами инкремента/декремента версии постфикс несколько сложнее. Компилятор создает временную копию переменной x, увеличивает или уменьшает оригинальный x (не копию), а затем возвращает копию. Только после возврата копия x удаляется. Например:

```
1
2   int x = 5;
    int y = x++; // x = 6, но переменной y присваивается 5
```

Унарные операции

Операции увеличения и уменьшения увеличивают или уменьшают значение операнда на единицу.

```
int t=1, s=2, z, f;
z=(t++)*5;
```

Вначале происходит умножение t*5, а затем увеличение t.

В результате получится z=5, t=2.

Бинарные операции

Операция деления (/) выполняет деление первого операнда на второй. Если две целые величины не делятся нацело, то результат округляется в сторону нуля.

При попытке деления на ноль выдается сообщение во время выполнения.

Операция остаток от деления (%) дает остаток от деления первого операнда на второй. Знак результата совпадает со знаком делимого. Если второй операнд равен нулю, то выдается сообщение.

Бинарная операция + для **строк** означает сцепление двух строк и присвоение результата третьей строке.

Сравнительные и логические

Используются для сравнения значения в двух переменных, или между переменной и константой

Знак операции	Операция	Пример	Значение
<	Меньше	3 < 2	false
>	Больше	3 > 2	true
<=	Меньше или равно	3 <= 2	false
>=	Больше или равно	3 >= 2	true
==	Тождество	3 == 2	false
!=	Не равно	3 != 2	true

Сравнительные и логические

Знак операции	Логическая операция	Пример	Значение
&&	И	3 && 2	true
||	ИЛИ	3 || 2	true
!	НЕ	!(0)	true

Пример: **if (a>10) && (a<20)**

Выражения, в которых используются логические операции возвращают **false** в случае ложного высказывания, и **true** в случае истинного

Логические (Битовые)

Обрабатывают данные после представления чисел в их бинарную форму (Битовое представление)

AND (NUM1 & NUM2)	Возвращает 1 если оба операнда 1
OR (NUM1 | NUM2)	Возвращает 1 если биты любого из двух операндов 1
NOT (~ NUM1)	Отрицание бита своего операнда
XOR (NUM1 ^ NUM2)	Возвращает 1 если любой из битов в операнде 1 но не оба

Логические (Битовые)

Пример:

10 & 15□ 1010 & 1111□1010 □ 10

10 | 15□ 1010 | 1111□1111 □ 15

10 ^ 15□ 1010 ^ 1111□0101 □ 5

~ 10 □ ~1010 □1011 □ -11 (из-за специфика представления отрицательных целых чисел)

ОПЕРАЦИИ ПРИСВАИВАНИЯ

- формат операция простого присваивания (=):
- опреанд_1 = операнд_2

пример: a=b=c=100, это выражение выполняется справа налево, результатом выполнения c=100, является число 100, которое затем присвоиться переменной b, потом a.

- *Составные операции присваивания*:
- (*=) – умножение с присв-ем,
- (/=) - деление с присв-ем
- (%=)- остаток от деления с присв-ем,
- (+=) –сложение с присв.,
- (-=) - вычит.с присв.
- *пример:* к операнду _1 прибавляется операнд_2 и результат записывается в операнд_2
- *т.е.* c = c + a, тогда компактная запись c += a

Составные операции присваивания

int X=5;

X*=5;

X-=10;

string S="Hello, ";

S+="World";

Приоритет арифметических операций

- Приоритет устанавливает иерархию одной группы операций над другой
- Это влияет на порядок, в котором вычисляются операции
- Приоритет операций может быть изменен с помощью заключения нужного выражения в скобки

Класс операций	Арифметические операции	Ассоциативность
Унарная	- ++ --	Справа налево
Бинарная	* / %	Слева направо
Бинарная	+ -	Слева направо
Бинарная	=	Справа налево

Приоритет арифметических операций

Всегда вычисляется слева направо

Приоритет для логических

Приоритет	Знак операции
1	!
2	&&
3	||

Примите во внимание следующее выражение:

False || True && ! False && True

Это выражение вычисляется, как показано ниже:

False || True && [! False] && True

! имеет высший приоритет.

False || [True && True]&& True

&& является оператором с высоким приоритетом и операторы с одинаковы
приоритетом вычисляются **СЛЕВА НА ПРАВО**

False || [True && True]

[False || True]

True

Приоритет среди разных операций

Приоритет	Тип оператора
1	Арифметический
2	Сравнение
3	Логический

Приоритет среди разных операций: пример

Рассмотрим следующий пример:

2*3+4/2 > 3 && 3<5 || 10<9

Ход вычислений показан ниже:

[2*3+4/2] > 3 && 3<5 || 10<9

Первыми вычисляются **арифметические операции**:

[[2*3]+[4/2]] > 3 && 3<5 || 10<9

[6+2] >3 && 3<5 || 10<9

[8 >3] && [3<5] || [10<9]

Приоритет среди разных операций: пример

[8 >3] && [3<5] || [10<9]

Следующими вычисляются **операции сравнения**, а поскольку они все имеют одинаковый приоритет, они вычисляются **СЛЕВА НА ПРАВО**

True && True || False

Последними вычисляются вычисляются логические операции.

[True && True] || False

True || False

True

Смена приоритета

- Скобки ( ) имеют высший приоритет
- Приоритет операций может быть изменен используя скобки ( )
- Операции низкого приоритета заключенные в скобки получают высший приоритет и исполняются первыми
- В случае вложенных скобок ((( ))), внутренние скобки будут исполняться первыми
- В случае вложенных скобок вычисления выполняются **СЛЕВА НАПРАВО**

Смена приоритета: пример

Рассмотрим следующий пример:

5+9*3^2-4 > 10 && (2+2^4-8/4 > 6 || (2<6 && 10>11))

Решение:

1. 5+9*3^2-4 > 10 && (2+2^4-8/4 > 6 || (True && False))

Внутренние скобки обладают приоритетом над другими операциями, а действия вне скобок выполняются по обычным правилам

2. 5+9*3^2-4 > 10 && (2+2^4-8/4 > 6 || False)

Смена приоритета: пример

3. 5+9*3^2-4 >10 && (2+16-8/4 > 6 || False)

   Затем выполняются действия во внешних скобках

4. 5+9*3^2-4 > 10 && (2+16-2 > 6 || False)
5. 5+9*3^2-4 > 10 && (18-2 > 6 || False)
6. 5+9*3^2-4 > 10 && (16 > 6 || False)
7. 5+9*3^2-4 > 10 && (True O||R False)
8. 5+9*3^2-4 > 10 && True

Смена приоритета: пример

9. 5+9*9-4>10 && True

Выражение слева выполняется по обычным правилам

10. 5+81-4>10 && True
11. 86-4>10 && True
12. 82>10 && True
13. True && True
14. **True**

ОПЕРАЦИИ ПРИСВАИВАНИЯ

- формат операция простого присваивания (=):
- опреанд_1 = операнд_2

пример: a=b=c=100, это выражение выполняется справа налево, результатом выполнения c=100, является число 100, которое затем присвоиться переменной b, потом a.

- *Сложные операции присваивания*:
- (*=) – умножение с присв-ем,
- (/=) - деление с присв-ем
- (%=)- остаток от деления с присв-ем,
- (+=) –сложение с присв.,
- (-=) - вычит.с присв.
- *пример:* к операнду _1 прибавляется операнд_2 и результат записывается в операнд_2
- *т.е.* c = c + a, тогда компактная запись c += a

Операция запятая

- **<выражение>, <выражение>, <выражение>, . . .**
- Пара выражений, разделенных запятой, вычисляется слева направо
- Результатом всего выражения будет результат последней части
- Чаще всего операция запятая используется в заголовках циклов, в местах, где синтаксис разрешает записать только одно выражение, а надо записать несколько. Тогда выражения объединяются в одно при помощи операции запятая.
- Запятая также используется и как **РАЗДЕЛИТЕЛЬ** списка объектов

Операция размер (унарная)

- **sizeof <выражение>**

или

- **sizeof (<тип>)**

- sizeof (4 + 1) ; // значение 4
- sizeof 4 + 1; // значение 5
- sizeof (int); // значение 4

Тернарная операция

- <условие> ? <выражение_если_истина> : <выражение если ложно>

- Пример тернарной операции:

string result = (myInteger < 10) ? "Меньше 10" : "Больше или равно 10";

ТЕРНАРНАЯ ОПЕРАЦИЯ

- *Условная операция (? :)*
- Формат условной операции: операнд_1 ? операнд_2 ? : операнд_3
- Операнд_1 это логическое или арифметич-ое выражение;
- Оценивается с точки зрения эквивалентности константам true и false;
- Если результат вычисления операнда_1 равен true,то результат условной операции будет значение операнда_2, иначе операнда_3;
- Тип может различаться;
- Условная операция является сокращ. формой услов-го оператора if;

Пример:

```
#include <iosteram>
   int main()
   using nanespace std;
   { int x, y,max;
     cin >>x>>y;
    (x>y)? cout <<x: cout<<y<<endl;     //1
    max=(x>y)? x:y;                      //2
    cout<<max<<endl;
    return 0;}
```

нахождение наибольшего значения для двух целых чисел;

Результат:

для x=11 и y=9
11
11

-

Общий список в справочниках

- https://docs.microsoft.com/ru-ru/cpp/cpp/cpp-built-in-operators-precedence-and-associativity?view=msvc-160

Структура программы: операторы

- Программа на языке C++ состоит из последовательности операторов, каждый из них определяет значение некоторого действия;
- Все операторы разделены на 4 группы:

- операторы следования;
- операторы ветвления;
- операторы цикла;
- операторы передачи управления.

ОПЕРАТОРЫ СЛЕДОВАНИЯ

- ***оператор выражение и составной оператор.***
- ***Выражение,*** завершающееся точкой с запятой, рассматривается как оператор (вычислении значения выражения или выполнении законченного действия);

  ++i; *//оператор инкремента*

  x+=y; *//оператор сложение с присваиванием*

  f(a, b) //вызов функции

  x= max (a, b) + a*b; *//вычисление сложного выражения*
- Частным случаем оператора выражения является ***пустой оператор*** *;*
- ***Составной оператор*** или ***блок*** представляет собой последовательность операторов, заключенных в фигурные скобки.
- **Блок обладает собственной *областью видимости:*** объявленные внутри блока имена доступны только внутри блока;
- Составные операторы применяются в случае, когда правила языка предусматривают наличие только одного оператора, а логика программы требует нескольких операторов.

ОПЕРАТОРЫ ВЕТВЛЕНИЯ

Условный оператор if

- **if** используется для разветвления процесса обработки данных на два направления.
- if имеет формы: *сокращенную* или *полную.*

- **Формат *сокращенного оператора if:*** **if (B) S;**
  B –УСЛОВИЕ: логич. или арифметич. выражение, истинность которого проверяется;
  S **-** один оператор: простой или составной.
- При выполнении сокращенной формы оператора if сначала вычисляется выражение *B,* затем проводится анализ его результата: если *B* истинно, то выполняется оператор S; если *B* ложно, то оператор *S* пропускается.
- С помощью сокращенной формы оператора If можно либо выполнить оператор S, либо пропустить его.
- **Формат *полного оператора if:*** **if (B) S1 ; else S2;**
  SI, S2- один оператор: простой или составной.
- При выполнении полной формы оператора *if* сначала вычисляется выражение *B,* затем анализируется его результат: если *B* истинно, то выполняется оператор S1 а оператор *S2* пропускается; если *B* ложно, то выполняется оператор S2, а *S1* - пропускается.
- С помощью полной формы оператора *if* можно выбрать одно из двух альтернативных действий процесса обработки данных.

Оператор if

- Допускается использование вложенных операторов if.
- Оператор if может быть включен в конструкцию if или в конструкцию else другого оператора if.
- Чтобы сделать программу более читабельной, рекомендуется группировать операторы и конструкции во вложенных операторах if, используя **фигурные скобки**.
- Если же фигурные скобки опущены, то компилятор связывает каждое ключевое слово else с наиболее близким if, для которого нет else.

Оператор if

Пример:

```
   {
int t=2, b=7, r=3;
   if (t>b)
   {
    if (b < r)  r=b;
   }
   else
    r=t;
}
```

В результате выполнения этой программы r станет равным 2.

Оператор if

Если же в программе опустить фигурные скобки, стоящие после оператора if, то программа будет иметь следующий вид:

```
{ int t=2,b=7,r=3;
if ( t>b)
if ( b < r ) t=b;
else  r=t;}
```

В этом случае r получит значение равное 3, так как ключевое слово else относится ко второму оператору if, который не выполняется, поскольку не выполняется условие, проверяемое в первом операторе if.

Вычисления по короткой схеме (*Short-circuit evaluation*)

- это стратегия в некоторых языках программирования, при которой второй логический оператор выполняется или оценивается только в том случае, если первого логического оператора недостаточно для определения значения выражения.
- Таким образом, после того, как результат выражения становится очевидным, его вычисление прекращается
- **Операции в c++** &&, ||, ?

- if (n!=0) && (k/n>17)

- Чтобы вычислить всё условие можно **на свой страх и риск** использовать побитовые операции

Оператор выбора *switch*

предназначен для разветвления процесса вычислений на несколько направлений.

- **Формат оператора:**

switch (<выражение>)
{case <константное_выражение_1>: [<оператор 1>]
case <константное_выражение_2>: [<оператор 2>]
……..
case <константное_выражение_n>: [<оператор n>]
[default: <оператор>]}

- Выражение, стоящее за ключевым словом *switch,* должно иметь арифметич. тип или тип указатель.

- Все константные выражения должны иметь разные значения, но совпадать с типом выражения, стоящим после *switch.*

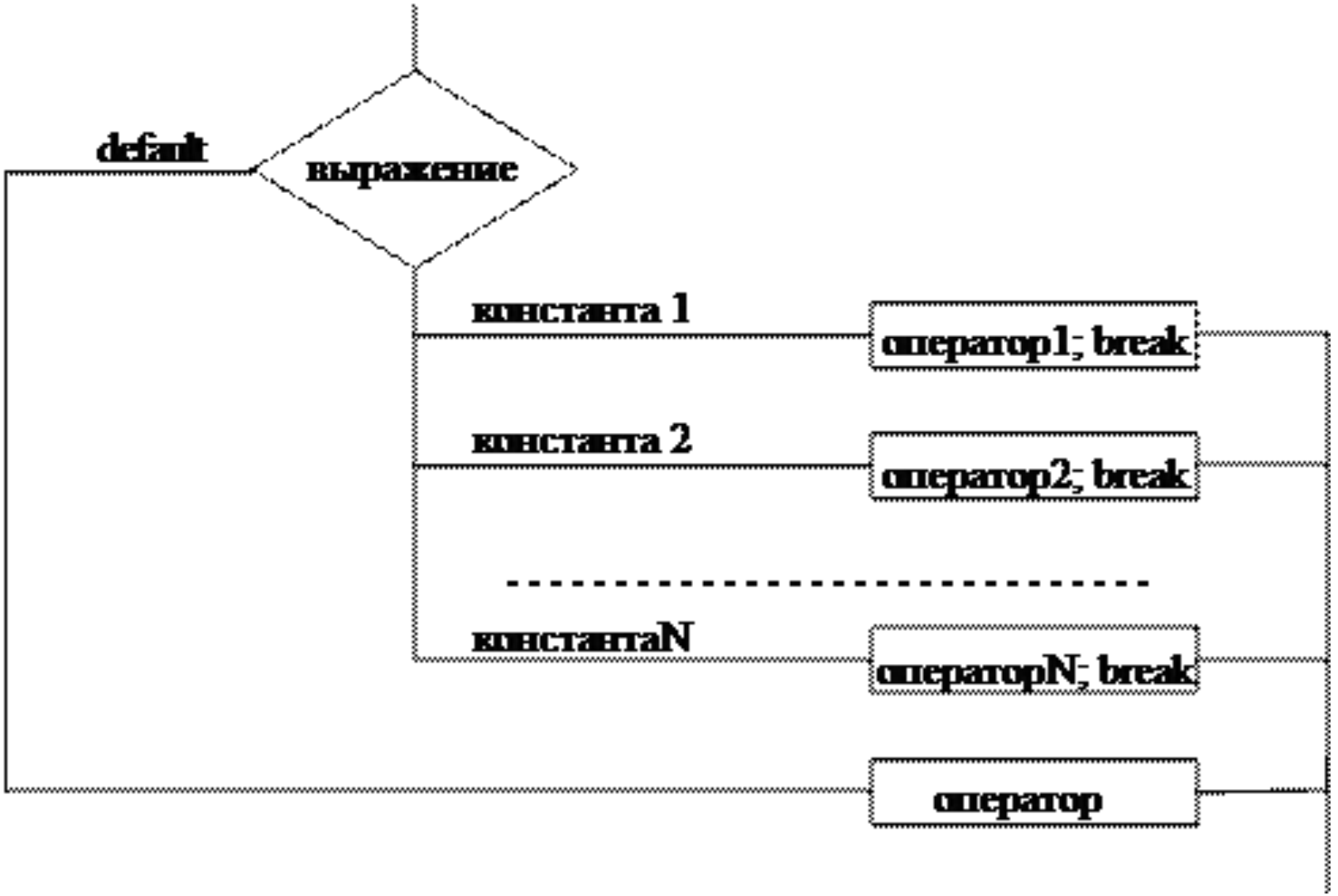
- Ключевое слово *case* и расположенное после него константное выражение называют также меткой *case.*

Разветвляющийся процесс

ВЫБОР ПО переменная
Значение_1: Действия
Значение_2: Действия
Значение_n: Действия
*: Действия
ВЫБОР ВСЕ

* означает, что переменная не равна ни одному значению

Чтобы избежать вложенных условия, используется конструкция ВЫБОР. Она позволяет иметь несколько ветвей для проверки равенства переменной одному из многих значений.


default
выражение
константа 1
оператор1; break
константа 2
оператор2; break
константаN
операторN; break
оператор

```
case low ... high:

case 'A' ... 'Z':

case 1 ... 5:        !!!пробелы     case 1...5:
```

ОПЕРАТОРЫ ЦИКЛА

- *Операторы цикла используются для организации многократно повторяющихся вычислений.*

- цикл с предусловием while,
- цикл с постусловием do while
- цикл с параметром for.

Цикл с предусловием while:

- Оператор цикла *while* организует выполнение одного оператора (простого или составного) неизвестное заранее число раз.
- ***Формат цикла while:*** **while (B) S;**
- ***B*** - выражение, истинность которого проверяется (условие завершения цикла);
- ***S*** - тело цикла: один оператор (простой или составной).

ОПЕРАТОРЫ ЦИКЛА

Перед каждым выполнением тела цикла анализируется значение выражения *B:*

- если оно истинно, то выполняется тело цикла, и управление передается на повторную проверку условия B;
- если значение B ложно - цикл завершается и управление передается на оператор, следующий за оператором S.
- если результат выражения B окажется ложным при первой проверке, то тело цикла не выполнится ни разу

если условие B во время работы цикла не будет изменяться, то возможна ситуация зацикливания, то есть невозможность выхода из цикла

Внутри тела должны находиться операторы, приводящие к изменению значения выражения B так, чтобы цикл мог завершиться.

ЦИКЛ С ПОСТУСЛОВИЕМ DO WHILE

- В отличие от цикла *while* условие завершения цикла проверяется после выполнения тела цикла.

- **Формат цикла *do while:*** **do S while (B);**

B - выражение, истинность которого проверяется (условие завершения цикла);

S - тело цикла: один оператор (простой или блок).

- Сначала выполняется оператор *S,* а затем анализир-ся значение выражения *B:*

- если оно истинно, то управление передается оператору *S,*

- если ложно - цикл заверш. и управление передается на оператор, следующий за условием *B*.

Пример(*do*)

Программа вывода на экран целых чисел из интервала от 1 до *n*.

- #include <iostream>
- using namespace std:
- int main()
- { intn, i=1;
- cout <<"n="; cin >>n;
- **while (i<=n)** ***//пока i меньше или равно n*** *Результаты работы программы:*
- { cout«i«"\t"; *//выводим на экран значение i*
- ++i;} *//увеличиваем i на единицу*
- return 0;}

n	ответ
10	12345678910

Пример(*do while*)

Программа вывода на экран целых чисел из интервала от 1 до *n*.

- #include <iostream>
- using namespace std;
- int main()
- {int n, i=1;
- cout <<"n="; cin >>n;
- **do** *//выводим на экран i, а замет увеличиваем*
- **cout<<i++<<"\t";** //ее *значении на единицу*
- **whiie (i<=n)**; *//до тех пор пока i меньше или равна n*
- return 0;}

Результаты работы программы:

n
10

ответ
1 2 3 4 5 6 7 8 9 10

ЦИКЛ С ПАРАМЕТРОМ FOR

- Цикл с параметром имеет следующую структуру:
- **for (<инициализация>; <выражение>; <модификации>) <оператор>;**
- ***Инициализация*** *используется для объявления и присвоения начальных значений величинам, используемым в цикле.*
- В этой части можно записать несколько операторов, разделенных запятой. Областью действия переменных, объявленных в части инициализации цикла, является цикл и вложенные блоки.
- ***Выражение*** *определяет условие выполнения цикла:*

- *если его результат истинен, цикл выполняется.*

- *Истинность* выражения проверяется перед каждым выполнением тела цикла, таким образом, цикл с параметром реализован как цикл с предусловием.
- ***Модификации*** *выполняются после каждой итерации цикла и служат обычно для изменения параметров цикла.*
- В части модификаций можно записать несколько операторов через запятую.
- ***Оператор*** (простой или составной) представляет собой тело цикла.

Пример (*for*)

- Любая из частей оператора *for* (инициализация, выражение, модификация, оператор) может отсутствовать, но точку с запятой, определяющую позицию пропускаемой части, надо оставить.

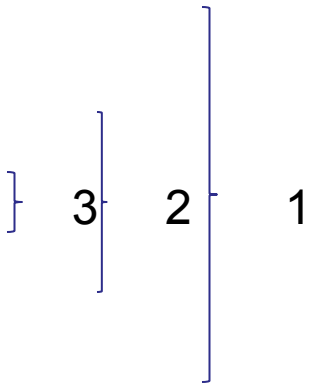
- #include <iostream>
- using namespace std;
- int main()
- { intn; cout <<"n="; cin >>n; ***Результаты работы программы***:
- **for (int i=1; j<=n; i++)** *//для i от 1 до n с шагом 1* ответ
- cout<<i<<"\t"; *//выводить на экран значение i* 1^{n}0123456789 10
- return 0;}

- *Замечание*. Используя операцию постфиксного инкремента при выводе данных на экран, цикл *for* можно преобразовать следующим образом:
- ***for (int i=l ;i<=n;) cout<<i+ + <<"\t";***
- В этом случае в заголовке цикла *for* отсутствует блок модификации.

ВЛОЖЕННЫЕ ЦИКЛЫ

- Циклы могут быть простые или вложенные (кратные, циклы в цикле).
- Вложенными могут быть циклы любых типов: *while, do while, for.*
- Структура вложенных циклов на примере *типа for* приведена ниже:

```
for(i=1;i<ik;i++)
{…
   for (j=10; j>jk;j- -)
      {...for(k=1;k<kk;j+=2){...}
      ...}
   ...}
```



Каждый внутренний цикл должен быть полностью вложен во все внешние циклы.

--- «Пересечения» циклов не допускается.

ОПЕРАТОРЫ БЕЗУСЛОВНОГО ПЕРЕХОДА

- В *C++* есть четыре оператора, изменяющие естественный порядок выполнения операторов:
- оператор безусловного перехода *goto,*
- оператор выхода *break*,
- оператор перехода к следующей итерации цикла *continue*,
- оператор возврата из функции *return.*

Break и Continue

- Оператор ***break*** используется внутри операторов ветвления и цикла для обеспечения перехода в точку программы, находящуюся **непосредственно за оператором**, внутри которого находится *break.*
- Оператор ***break*** применяется также для выхода из оператора *switch,* аналогичным образом он может применяться для выхода из других операторов.
- Оператор перехода к следующей итерации цикла ***continue*** пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующей итерации (повторение тела цикла).

- Выполнение оператора начинается с вычисления выражения, расположенного за ключевым словом *switch.*

- Полученный результат сравнивается с меткой *case.*
- Если результат выражения соответствует метке *case,* то выполняется оператор, стоящий после этой метки.
- Затем последовательно выполняются **ВСЕ ОПЕРАТОРЫ ДО КОНЦА ОПЕРАТОРА** *switch* , если только их выполнение не будет прервано с помощью оператора передачи управления *break*
- При использование оператора *break* происходит выход из *switch* и управление переходит к первому после него оператору.
- Если совпадения выражения ни с одной меткой *case* не произошло, то выполняется оператор, стоящий после слова *default,* а при его отсутствии управление передается следующему за *switch* оператору.

- Пример. Известен порядковый номер дня недели. Вывести на экран его название.

```
#include <iostream>
using namespace std;
 int main()
{int x; cin >>x;
switch (x)
{ case 1: cout <<"понедельник";
    break;
  case 2: cout <<"вторник"; break;
  case 3: cout <<"среда"; break;
  case 4: cout <<"четверг"; break;
  case 5: cout <<"пятница"; break;
   case 6: cout <<"суббота"; break;
   case7: cout <<"
    воскресенье";break;
   default: cout <<"вы ошиблись";}
return 0;}
```

Пример continue

- #include <iostream>
- using namespace std;
- int main()
- { for (int i=1; i<100; ++i) *//перебираем все числа от 1 до 99*
  {if (i % 2) continue; *//если число нечетное, то переходим к следующей итерации*
  cout<<i<<"\t";} *//выводим число на экран*
- return 0;}

- В результате данной программы на экран будут выведены только четные числа из интервала от 1 до 100, т.к. для нечётных чисел текущая итерация цикла прерывалась и команда *cout<<i<<"\t"* не выполнялась.
- ***Оператор возврата** из **функции** **return:***