Часть 5.
Условные операторы

ОПЕРАТОРЫ С++

- Программа на языке С++ 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

- К ним относятся : ***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*** ;

(используется, когда по синтаксису оператор требуется, а по смыслу — нет)

- ***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

- К ним относятся : *условный оператор if и оператор выбора switch, они* позволяют изменить порядок выполнения операторов в программе;

Условный оператор if

- **if** используется для разветвления процесса обработки данных на два направления.
- if имеет 2 формы: *сокращенную* или *полную.*

- **Формат *сокращенного оператора if:*** **if (B) S;**

  B –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;

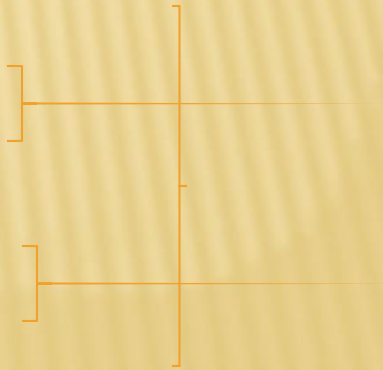
- ***SI, S2-*** один оператор: простой или составной.
- При выполнении полной формы оператора *if* сначала вычисляется выражение *B,* затем анализируется его результат: если *B* истинно, то выполняется оператор S1 а оператор *S2* пропускается; если *B* ложно, то выполняется оператор S2, а *S1* - пропускается.
- С помощью полной формы оператора *if* можно выбрать одно из двух альтернативных действий процесса обработки данных.

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Enter a number: ";
6      int a;
7      std::cin >> a;
8
9      if (a > 15)
10         std::cout << a << " is greater than 15\n";
11     else
12         std::cout << a << " is not greater than 15\n";
13
14     return 0;
15 }
```

Примеры записи условного оператора *if*.

- if (a > 0) x=y; *// сокращенная форма с простым оператором*
- if (++i>0) {x=y; y=2*z;} *// сокращенная форма с составным оператором*
- if (a > 0 || b<0) x=y; eise x=z; *//полная форма с простым оператором*
- if (i+j-1<0) { x= 0; y= 1;} else {x=1; y:=0;} *//полная форма с составными оператором*

- Операторы S1 и S2 могут являться операторами ***if***, такие операторы наз. *вложенные*;
- Ключевое слово *else* связывается с ближайшим предыдущим словом ***if***, которое еще не связано ни с одним *else.*



ВЛОЖЕННЫЕ ВЕТВЛЕНИЯ IF/ELSE

```cpp
#include <iostream>

int main()
{
    std::cout << "Enter a number: ";
    int a;
    std::cin >> a;

    if (a > 15) // внешний оператор if
        // Это плохой способ написания вложенных стейтментов if
        if (a < 25) // внутренний оператор if
            std::cout << a << " is between 15 and 25\n";

        // К какому if относится следующий else?
        else
            std::cout << a << " is greater than or equal to 25\n";

    return 0;
}
```

ВЛОЖЕННЫЕ ВЕТВЛЕНИЯ IF/ELSE

```cpp
#include <iostream>

int main()
{
    std::cout << "Enter a number: ";
    int a;
    std::cin >> a;

    if (a > 15)
    {
        if (a < 25)
            std::cout << a << " is between 15 and 25\n";
        else // относится к внутреннему оператору if
            std::cout << a << " is greater than or equal to 25\n";
    }

    return 0;
}
```

ИСПОЛЬЗОВАНИЕ ЛОГИЧЕСКИХ ОПЕРАТОРОВ В ВЕТВЛЕНИЯХ IF/ELSE

```cpp
#include <iostream>

int main()
{
    std::cout << "Enter an integer: ";
    int a;
    std::cin >> a;

    std::cout << "Enter another integer: ";
    int b;
    std::cin >> b;

    if (a > 0 && b > 0) // && - это логическое И. Проверяем, являются ли оба условия истинными
        std::cout << "Both numbers are positive\n";
    else if (a > 0 || b > 0) // || - это логическое ИЛИ. Проверяем, является ли истинным хоть одно из услови
        std::cout << "One of the numbers is positive\n";
    else
        std::cout << "Neither number is positive\n";

    return 0;
}
```

Сложные условия

Сложное условие – это условие, состоящее из нескольких простых условий (отношений), связанных с помощью **логических операций**:

`!` – НЕ (*not*, отрицание, инверсия)

`&&` – И (*and*, логическое умножение, конъюнкция, одновременное выполнение условий)

`||` – ИЛИ (*or*, логическое сложение, дизъюнкция, выполнение хотя бы одного из условий)

Простые условия (отношения)

`<` `<=` `>` `>=` `==` `!=`

равно

не равно

Сложные условия

Порядок выполнения сложных условий:

- выражения в скобках
- `!` (НЕ, отрицание)
- `<, <=, >, >=`
- `==, !=`
- `&&` (И)
- `||` (ИЛИ)

Пример:

```
        2   1      6   3     5       4
if ( !(a > b) || c != d && b == a)
   {
   ...
   }
```

Сложные условия

Истинно или ложно при `a = 2; b = 3; c = 4;`

```
!(a > b)
a < b && b < c
!(a >= b) || c == d
a < c || b < c && b < a
a > b || !(b < c)
```

1

1

1

1

0

Для каких значений x истинны условия:

```
x < 6 && x < 10
x < 6 && x > 10
x > 6 && x < 10
x > 6 && x > 10
x < 6 || x < 10
x < 6 || x > 10
x > 6 || x < 10
x > 6 || x > 10
```

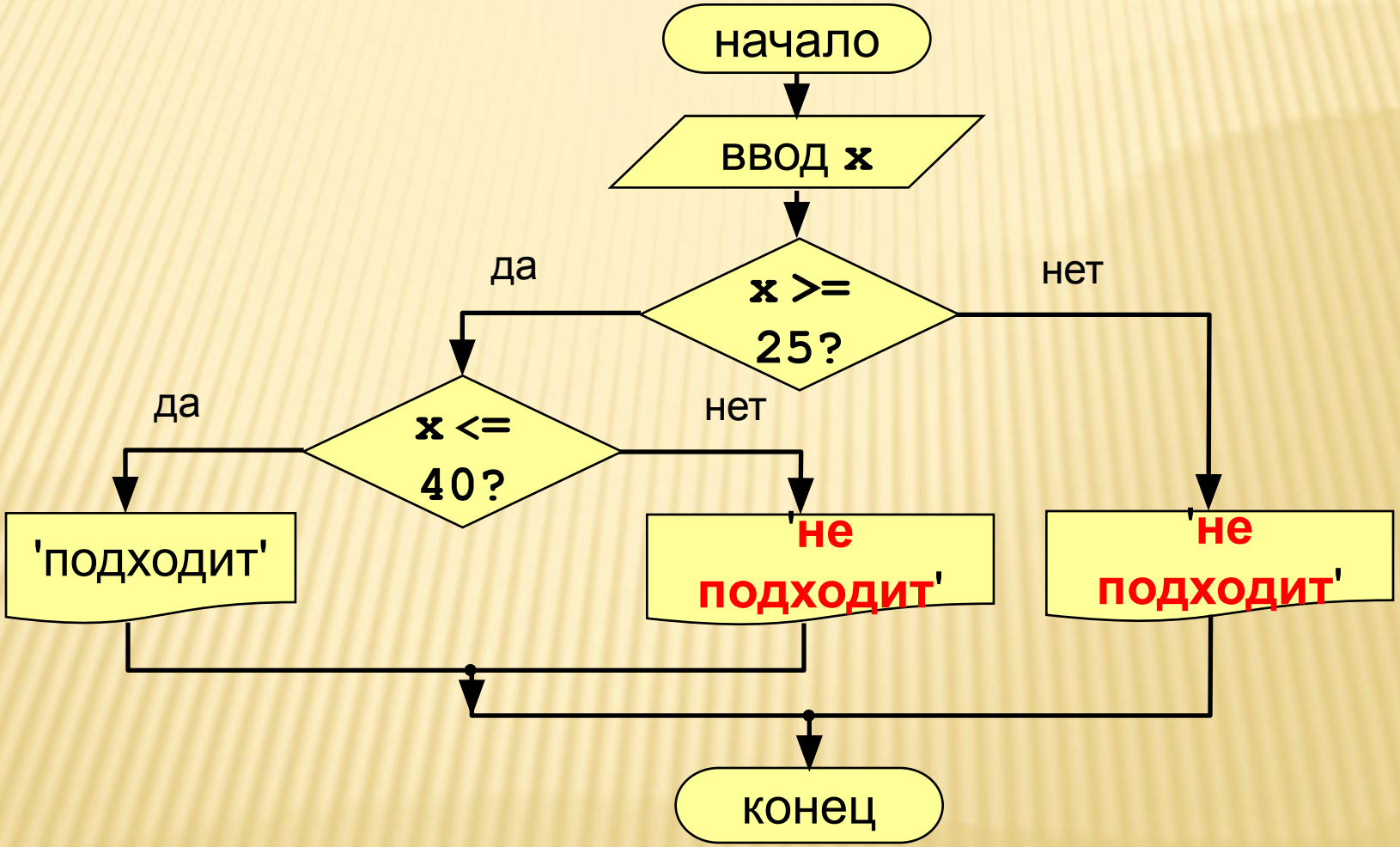
$(-\infty, 6)$	$x < 6$
$\varnothing$	
$(6, 10)$	
$(10, \infty)$	$x > 10$
$(-\infty, 10)$	$x < 10$
$(-\infty, 6) \cup (10, \infty)$	
$(-\infty, \infty)$	
$(6, \infty)$	$x > 6$

Сложные условия

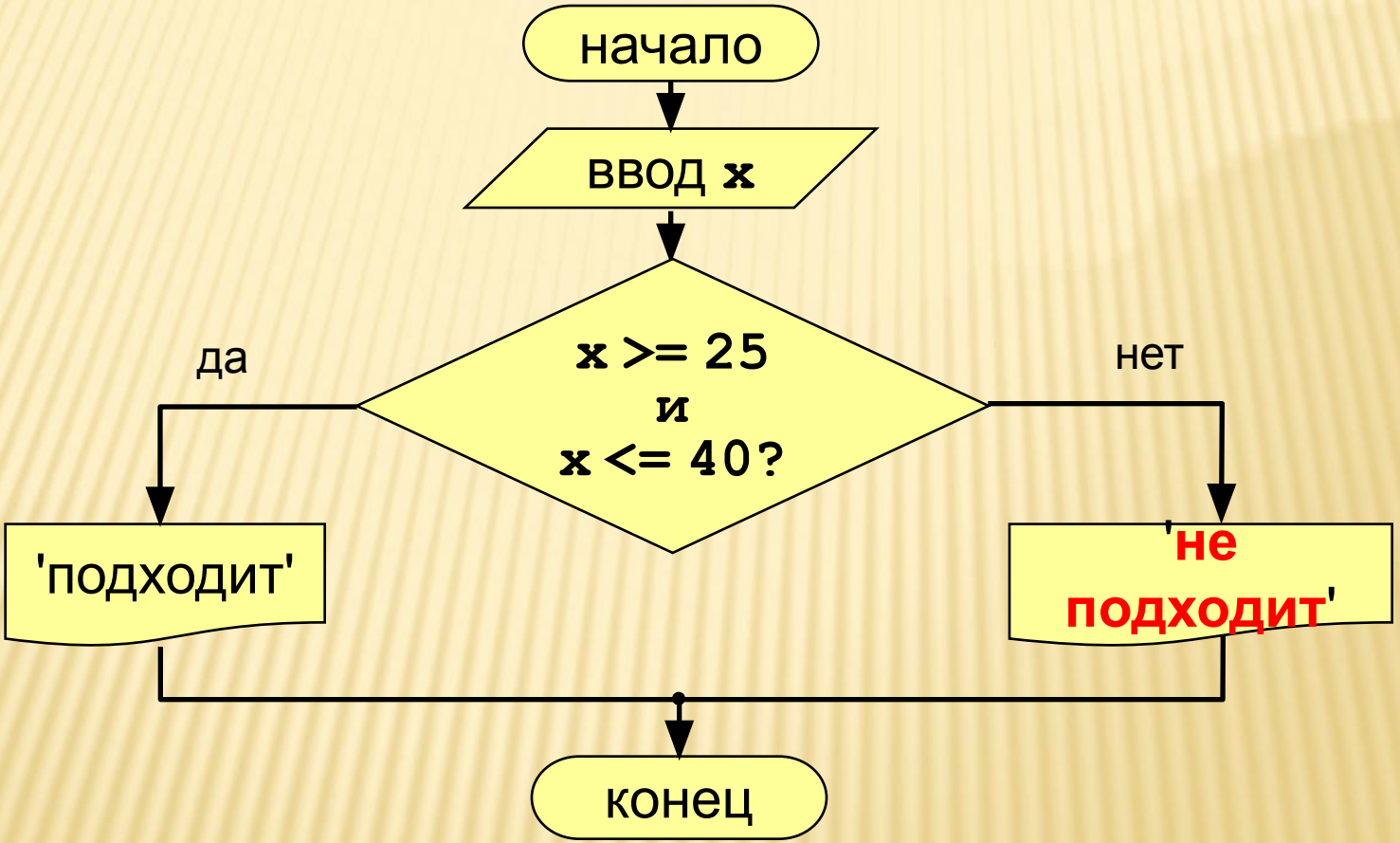
Задача. Фирма набирает сотрудников от 25 до 40 лет включительно. Ввести возраст человека и определить, подходит ли он фирме (вывести ответ «подходит» или «не подходит»).

Особенность: надо проверить, выполняются ли два условия одновременно.

Вариант 1. Алгоритм



Вариант 2. Алгоритм



ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

1. **Даны три действительных числа.**
   Возвести в куб те из них, значения которых неотрицательны.
   Результат вычисления вывести на экран монитора.
2. **Написать программу вычисления значения функции**

$$y = \begin{cases} x^2, & \text{если } 0<x<2; \\ x+4, & \text{если } -2<x\le 0; \\ 0, & \text{в остальных случаях.} \end{cases}$$

3. **Написать программу выбора наибольшего из трёх чисел.**
4. **Даны x, y. Если x и y отрицательны, то каждое значение заменить его модулем;**
   если отрицательно только одно из них, то оба значения увеличить на 0.5;
   если оба значения неотрицательны, то оба значения увеличить в 10 раз.
   Результат вывести на экран.
5. **Даны три числа a, b, c. Выяснить, верно ли, что a<b<c.**
   Ответ получить в текстовой форме: верно или неверно.
6. **Определить, является ли частное чисел a и b,**
   округлённое до ближайшего целого чётным числом.

Оператор выбора

Задача: Ввести номер месяца и вывести количество дней в этом месяце.

Решение: Число дней по месяцам:

28 дней – 2 (февраль)

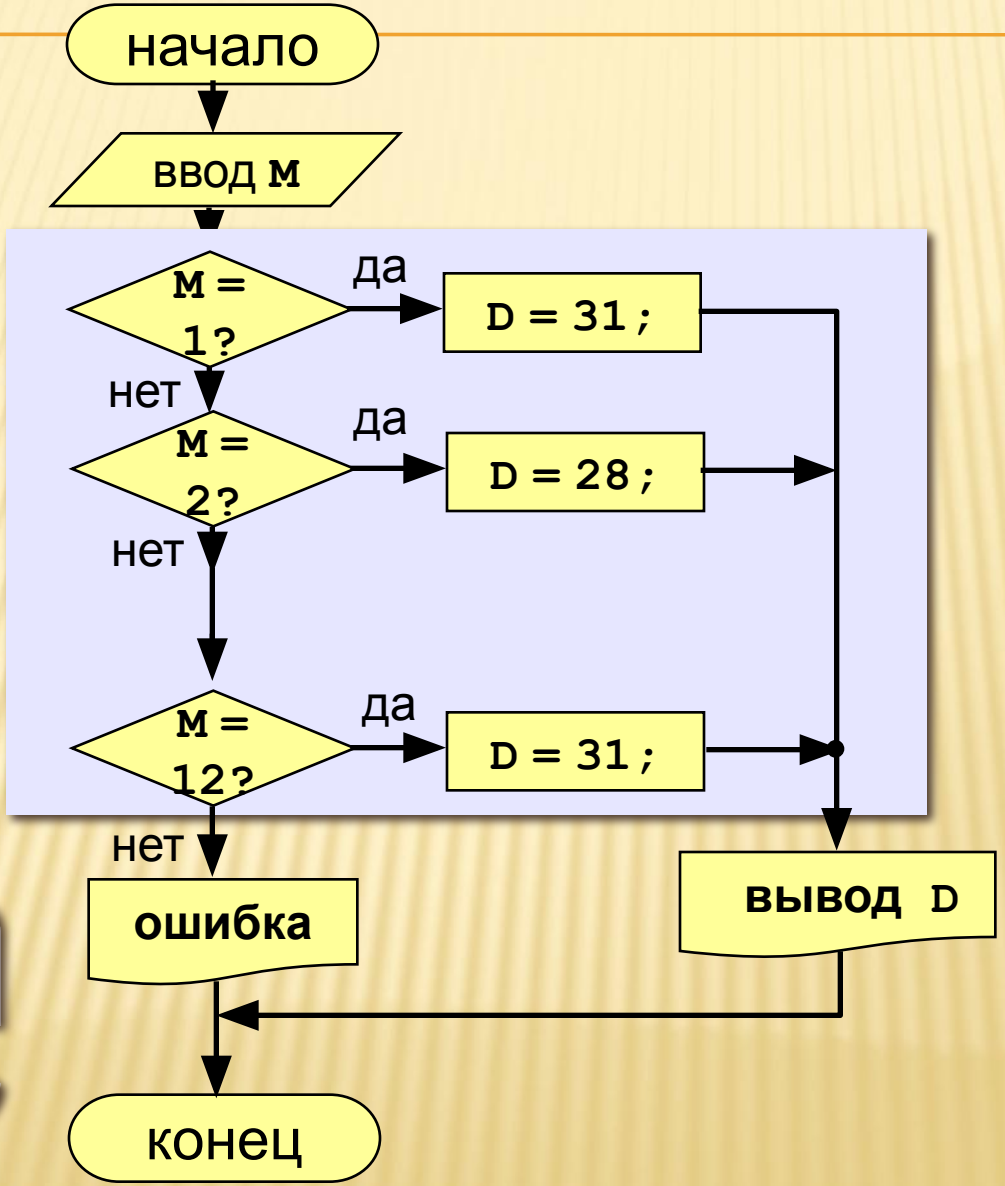
30 дней – 4 (апрель), 6 (июнь), 9 (сентябрь), 11 (ноябрь)

31 день – 1 (январь), 3 (март), 5 (май), 7 (июль), 8 (август), 10 (октябрь), 12 (декабрь)

Особенность: Выбор не из двух, а из нескольких вариантов в зависимости от номера месяца.

Можно ли решить известными методами?

Алгоритм



оператор выбора

ни один вариант не подошел

Оператор выбора *switch*

предназначен для разветвления процесса вычислений на несколько направлений.

- **Формат оператора:**

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

Выражение, стоящее за ключевым словом *switch,* должно иметь арифметический 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.*

- 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.

- все выражения case должны производить уникальные значения.
- То есть вы не сможете сделать следующее:

```
1 switch (z)
2 {
3     case 3:
4     case 3: // нельзя, значение 3 уже используется!
```

Можно использовать сразу несколько кейсов для одного выражения. Следующая функция использует несколько кейсов для проверки, соответствует ли параметр p числу из ascii-таблицы:

```
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;
    }
}
```

В случае, если `p` является числом из ASCII-таблицы, то выполнится первый стейтмент после кейса — `return true;`.

НЕСКОЛЬКО СТЕЙТМЕНТОВ ВНУТРИ БЛОКА SWITCH

- вы можете использовать несколько стейтментов под каждым кейсом, не определяя новый блок:

```
1  switch (3)
2  {
3      case 3:
4          std::cout << 3;
5          boo();
6          std::cout << 4;
7          break;
8      default:
9          std::cout << "default case\n";
10         break;
11 }
```

ОБЪЯВЛЕНИЕ ПЕРЕМЕННОЙ И ЕЁ ИНИЦИАЛИЗАЦИЯ ВНУТРИ CASE

- можете объявлять, но не инициализировать переменные внутри

```
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;
}
```

ОБЪЯВЛЕНИЕ ПЕРЕМЕННОЙ И ЕЁ ИНИЦИАЛИЗАЦИЯ ВНУТРИ CASE

- Обратите внимание, что, хотя переменная z была определена в первом кейсе, она также используется и во втором кейсе.
- Все кейсы считаются частью одной и той же области видимости, поэтому, объявив переменную в одном кейсе, мы можем спокойно использовать её без объявления и в других кейсах.

ОБЪЯВЛЕНИЕ ПЕРЕМЕННОЙ И ЕЁ ИНИЦИАЛИЗАЦИЯ ВНУТРИ CASE

- Если в кейсе нужно объявить и/или инициализировать новую переменную, то это лучше всего сделать, используя блок стейтментов внутри кейса:

```
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;
}
```

Оператор выбора

Задача: Ввести номер месяца и вывести количество дней в этом месяце.

Решение: Число дней по месяцам:

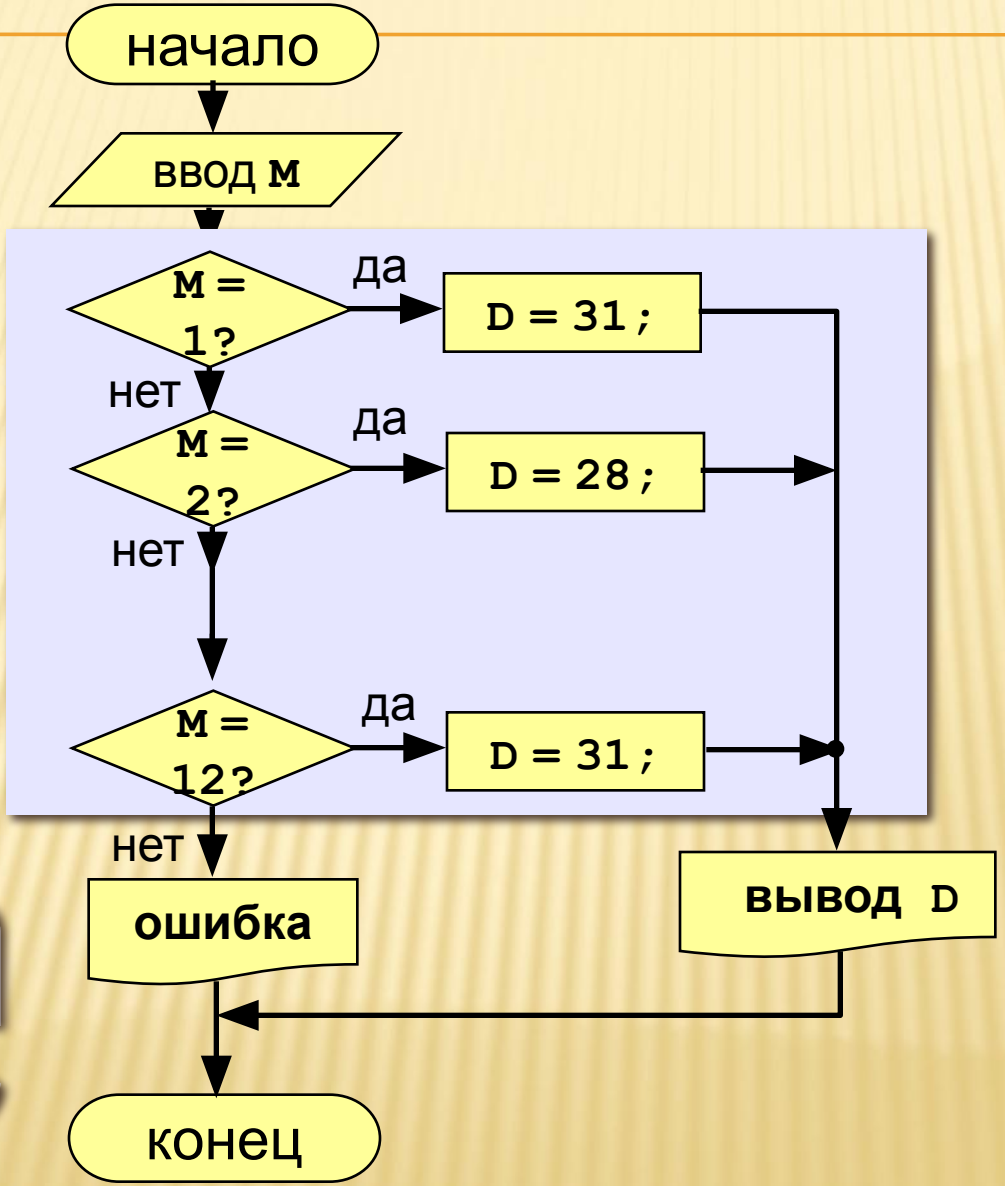
28 дней – 2 (февраль)

30 дней – 4 (апрель), 6 (июнь), 9 (сентябрь), 11 (ноябрь)

31 день – 1 (январь), 3 (март), 5 (май), 7 (июль), 8 (август), 10 (октябрь), 12 (декабрь)

Особенность: Выбор не из двух, а из нескольких вариантов в зависимости от номера месяца.

Алгоритм



оператор выбора

ни один вариант не подошел

Программа

```
main()
{
 int M, D;
 cout<<"Введите номер месяца:\n";
 cin>>M;
 switch ( M ) {
   case 2:  D = 28; break;
   case 4: case 6: case 9: case 11:
            D = 30; break;
   case 1: case 3: case 5: case 7:
   case 8: case 10: case 12:
            D = 31; break;
   default: D = -1;
   }
 if (D > 0)
      cout<<"В этом месяце"<<d <<"дней.";
 else cout<<"Неверный номер месяца";
}
```

выйти из **switch**

ни один вариант не подошел

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

```
#include <iostream>
 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;}
```

Оператор выбора

Задача: Ввести букву и вывести название животного на эту букву.

Особенность: выбор по символьной величине.

```
main()
{
 char c;
 cout<<"Введите первую букву названия животного:\n";
 cin>>c;
 switch ( c ) {
   case 'а': printf("Антилопа"); break;
   case 'б': printf("Бизон"); break;
   case 'в': printf("Волк"); break;
   default:  printf("Я не знаю!");
   }
}
```

Что будет, если везде убрать `break`?

Оператор выбора

Особенности:

- после **`switch`** может быть имя переменной или арифметическое выражение целого типа (**`int`**)

```
switch ( i+3 ) {
  case 1: a = b; break;
  case 2: a = c;
  }
```

или символьного типа (**`char`**)

- **нельзя** ставить два **одинаковых** значения:

```
switch ( x ) {
  case 1: a = b; break;
  case 1: a = c;
  }
```

- **Дан номер фигуры (1- квадрат, 2 - треугольник);**
- **по номеру фигуры запросить необходимые данные для вычисления площади;**
- **произвести вычисление площади фигуры и вывести полученные данные на экран.**

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{ int x;
  cout << “Программа подсчитывает площадь:\n1. квадрата;\n2. треугольника.\n3. выход из программы”;
  cout <<“Укажите номер фигуры или завершите работу с программой.\n”;
  cin >> x;
  switch (x)
   {case 1 :{cout <<"введите длину стороны квадрата\n";
             float a; cin >>a;
             if (a>0) cout«"Площадь квадрата со стороной" <<a <<"равна\t” <<a*a;
               else cout «"Квадрат не существует\n";
             break;}
    case 2: {cout« "введите длины сторон треугольника\n";
             float a,b,c,p, s; cin >>a >>b >>c;
             if (a+b>c && a+c>b && b+c>a)
            {p=(a+b+c)/2;   s= sqrt(p*(p-a)*(p-b)*(p-c));
             cout <<"Площадь треугольника со сторонами" <<a <<b <<c <<"равная\t" <<s;}
             else cout<<"Треугольник не существуетет\n";
             break;}
  case 3:break;
   default: cout <<"Номер фигуры указан не верно\n";}
return 0;}
```

Задача: допустим, у пользователя есть нумерованный список станций метро Барселоны. Необходимо написать код, в котором будет реализован диалог с пользователем, а именно предложено ввести номер станции метро. После ввода номера станции, надо показать на экран её название и время в пути. Если же станции с таким номером нет, сообщить об этом и предложить ввести номер снова.

```
#include <iostream>
using namespace std;

int main()
{
 setlocale(LC_ALL, "rus");

 int answer = 0; // будет хранить выбор пользователя
 bool var = true; // управляющая переменная цикла do while

 cout << "Введите номер станции метро, для расчета времени в пути (от 1 до 5): ";

 do{
 cin >> answer; // ввод значения

 switch (answer) // switch принимает переменную answer и ищет подходящий case
 {
 case 1:  // если answer равно 1, на экран выйдут сообщения этого case
 cout << "Станция метро Trinitat Nova: ";
 cout << "Время в пути - 15 мин." << endl;
 break; // выход из switch. иначе будет переход case(2)  и т.д.
 case 2: // если answer равно 2
 cout << "Станция метро Casa de l'Aigua: ";
 cout << "Время в пути - 19 мин." << endl;
 break;
 case 3:
 cout << "Станция метро Torre Baro Vallbona: ";
 cout << "Время в пути - 25 мин." << endl;
 break;
 case 4:
 cout << "Станция метро Ciutat Meridiana: ";
 cout << "Время в пути - 30 мин." << endl;
 break;
```

```
33  break;
34  case 5:
35  cout << "Станция метро Can Cuias: ";
36  cout << "Время в пути - 38 мин." << endl;
37  break;
38  default: // если ни один case не сработал, сработает default
39  cout << "Станции под номером " << answer << " нет! ";
40  cout << "Сделайте правильный выбор (от 1 до 5): ";
41  }
42  // если введено правильное значение (от 1 до 5)
43  // default не сработает и выполнится блок if
44  // переменная var примет значение false
45  // и цикл do while не повторится
46  if (answer >= 1 && answer <= 5)
47  var = false;
48  } while (var); //цикл повторится, пока var не изменит значение на false
49
50  return 0;
51 }
```

Вот как работает эта программа:

```
Введите номер станции метро, для расчета времени в пути (от 1 до 5): 77
Станции под номером 77 нет! Сделайте правильный выбор (от 1 до 5): -5
Станции под номером -5 нет! Сделайте правильный выбор (от 1 до 5): 5
Станция метро Can Cuias: Время в пути - 38 мин.
```

ЗАДАНИЕ

- Изменить код, который был рассмотрен, следующим образом:
- Пользователю для расчета времени в пути надо вводить не номера станций, а буквы (A,B,C,D,E). Необходимо предусмотреть ввод пользователем и маленьких, и больших букв (избавиться от регистрозависимости). То есть если введено или D, или d – должен сработать один и тот же case.

Задание

1. Ввести номер месяца и номер дня, вывести число дней, оставшихся до Нового года.

   Пример:

```
Введите номер месяца:
12
Введите день:
25
До Нового года осталось 6 дней.
```

2. Дан порядковый номер месяца, вывести на экран его название.

3. Дан порядковый номер дня недели, вывести на экран количество дней оставшихся до конца недели.

Задания

4. Ввести номер месяца и вывести название времени года.

Пример:

```
Введите номер месяца:
4
весна
```

5. Ввести возраст человека (от 1 до 150 лет) и вывести его вместе с последующим словом «год», «года» или «лет».

Пример:

```
Введите возраст:
24
Вам 24 года
```

```
Введите возраст:
57
Вам 57 лет
```

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*.

ОПЕРАТОР БЕЗУСЛОВНОГО ПЕРЕХОДА GOTO

- Оператор безусловного перехода *goto* имеет формат: goto <метка>;
- В теле той же функции должна присутствовать ровно одна конструкция вида:
- <метка>: <оператор>;
- Оператор *goto* передает управление на помеченный меткой оператор

ОПЕРАТОР БЕЗУСЛОВНОГО ПЕРЕХОДА GOTO

- **Пример использования оператора goto:**

```
#include <iostream>
    int main()
    {float x;
    metka: cout <<"x=";  //оператор, помеченный меткой
        cin»x;
         if (x!=0) cout<<“y=“<<1/x<<endl;
           else { cout<<"функция не определена\n";
                goto metka;}        // передача управление метке
return 0;}
```

- - при попытке ввести 0 на экран будет выведено сообщение «функция не определена», после чего управление будет передано оператору, помеченному меткой, и программа повторно попросит ввести значение x.
- *использование оператора goto затрудняет чтение больших по объему программ, поэтому использовать метки нужно только в крайних случаях.*

- Оператор goto — это оператор управления потоком выполнения программ, который заставляет центральный процессор выполнить переход из одного участка кода в другой (осуществить прыжок). Другой участок кода идентифицируется с помощью лейбла. Например:

```cpp
#include <iostream>
#include <cmath> // для функции sqrt()

int main()
{
    double z;
tryAgain: // это лейбл
    std::cout << "Enter a non-negative number: ";
    std::cin >> z;

    if (z < 0.0)
        goto tryAgain; // а это оператор goto

    std::cout << "The sqrt of " << z << " is " << sqrt(z) << std::endl;
    return 0;
}
```

В этой программе пользователю предлагается ввести неотрицательное число. Однако, если пользователь введет отрицательное число, программа, используя оператор goto, выполнит переход обратно к лейблу `tryAgain`. Затем пользователю снова нужно будет ввести число. Таким образом, мы можем постоянно запрашивать у пользователя ввод числа, пока он не введет корректное число.

Существуют некоторые ограничения на использование операторов goto. Например, вы не сможете перепрыгнуть вперед через переменную, которая инициализирована в том же блоке, что и goto:

```
1 int main()
2 {
3     goto skip; // прыжок вперед недопустим
4     int z = 7;
5 skip: // лейбл
6     z += 4; // какое значение будет в этой переменной?
7     return 0;
8 }
```

ПРАВИЛО: ИЗБЕГАЙТЕ ИСПОЛЬЗОВАНИЯ ОПЕРАТОРОВ GOTO, ЕСЛИ НА ЭТО НЕТ ВЕСКИХ ПРИЧИН.

- В целом, программисты избегают использования оператора goto в языке C++ (и в большинстве других высокоуровневых языков программирования). Основная проблема с ним заключается в том, что он позволяет программисту управлять выполнением кода так, что точка выполнения может прыгать по коду произвольно. А это, в свою очередь, создает то, что опытные программисты называют «спагетти-кодом». **Спагетти-код** — это код, порядок выполнения которого напоминает тарелку со спагетти (всё запутано и закручено), что крайне затрудняет следование порядку и понимание логики выполнения такого кода.

- В C++ goto почти никогда не используется, поскольку любой код, написанный с ним, можно более эффективно переписать с использованием других объектов в языке C++, таких как циклы, обработчики исключений или деструкторы.

```cpp
#include <iostream>
#include <cmath> // для функции sqrt()

int main()
{
    double z;

    do
    {
        std::cout << "Enter a non-negative number: ";
        std::cin >> z;
    }
    while (z < 0.0);

    std::cout << "The sqrt of " << z << " is " << sqrt(z) << std::endl;
    return 0;
}
```

ОПЕРАТОР ВЫХОДА BREAK

- 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.

Пример. Определите частное двух вещественных чисел. Исключите деление на ноль. Произведите запрос об окончании работы программы

```
1  #include <iostream>
2
3  int main()
4  {
5      float x, y, rez;
6      int s;
7  start: std::cout << "Vvedite delimoe:";
8      std::cin >> x;
9      std::cout << "Vvedite delitel:";
10     std::cin >> y;
11     if (y == 0) {
12         std::cout << "Delenie na 0\n";
13         goto vibor;
14     }
15     rez = x / y;
16     std::cout << "Chastnoe=" << rez << std::endl;
17 vibor:std::cout << "Prodolgit raboty? 1-yes, 2-no :";
18     std::cin>>s;
19     if (s == 1) goto start;
20
21     return 0;
22 }
```

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

1. Напишите программу вычисления квадратного корня из числа, введённого с клавиатуры. Исключите вычисление корня из отрицательных чисел и нуля. Снабдите программу запросом об окончании работы.

2. Напишите программу, которая выводит название дня недели по номеру дня введённому с клавиатуры. Исключите несуществующие номера дней недели и снабдите программу запросом об окончании работы.

3. Напишите программу, которая возводит действительное число, введённое с клавиатуры, в степень, введённую с клавиатуры в диапазоне от 2 до 9. Исключите несуществующие (меньше 2 и больше 9) степени, и снабдите программу запросом об окончании работы.

4. Напишите программу определения времени года по порядковому номеру недели. Снабдите программу запросом об окончании работы.

САМОСТОЯТЕЛЬНАЯ РАБОТА

- Написать программы для решения следующих задач:

1. Найти результат деления целого числа A на целое число B. Найти целую часть и остаток. Если число B=0, то попросить пользователя ввести другое число.
2. Найти корни квадратного уравнения по коэффициентам a, b, c. Если a=0, организовать запрос: продолжить работу программы с повторным вводом коэффициента a или завершить работу.