Любой язык содержит следующие базовые элементы: алфавит, лексемы, выражения, операторы. Лексемы образуются из символов, выражения — из лексем и символов, операторы — из символов, выражений и лексем.

Итак,

- *Алфавит языка* — это основные неделимые знаки, с помощью которых пишутся все тексты на языке (например, в русском языке — буквы).
- *Лексема* — минимальная единица языка, имеющая самостоятельный смысл (в русском языке — слово).
- *Выражение* — правило вычисления некоторого значения (в русском языке — словосочетание).
- *Оператор* — законченное описание некоторого действия (в русском языке — предложение).

1. Базовые операторы

Во всех языках программирования существует три базовых конструкции:

Последовательные операторы Операторы выполняются последовательно в порядке написания.

Операторы ветвления В зависимости от результата некоторого условия, выполняется либо один либо другой оператор.

Операторы цикла Одно и тоже действие выполняется несколько раз до выполнения определенного условия.

Также существуют операторы управления, позволяющие прерывать выполнение операторов в зависимости от некоторых условий.

1.1. Операторы ветвления

Существуют два оператора ветвления: *условный оператор* и *оператор выбора*. Оператор ветвления позволяет выбрать из двух вариантов, оператор выбора — из нескольких.

Под условным оператором понимается конструкция, когда при истинном значении некого логического выражения выполняется один блок операторов, при ложном — другой.

Общий вид условного оператора представлен на блок-схеме (рисунок 1):

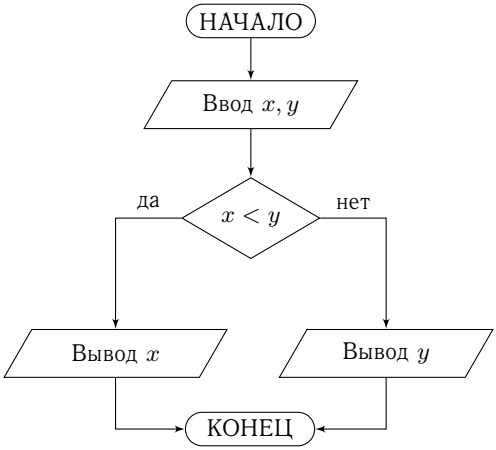

Рис. 1. Условный оператор на примере поиска наименьшего из двух чисел.

Сначала вводятся два числа. Потом вычисляется логическое выражение (сравниваются значения двух переменных `x` и `y`). В случае, если оно истинно (`x` меньше `y`), выполняются операторы, расположенные на ветке «Да» (в данном случае, вывод на экран значение переменной `x`). В случае, если логического выражение ложно (`x` больше `y`), выполняются операторы, расположенные на ветке «Нет» (вывод значения переменной `y`).

Общий вид условного оператора в языке C++:

`if` (⟨*условное выражение*⟩) ⟨*операторы «истина»*⟩ [`else` ⟨*операторы «ложь»*⟩].

Круглые скобки после ключевого слова `if` являются обязательными, так как они отделяют условное выражение от выполняемых операторов. В качестве условного выражения может выступать любое допустимое выражение, необязательно логическое. Тогда, если значение выражения равно нулю, выполняются операторы, соответствующие ветке «ложь». Если значение выражения отлично от нуля, выполняются операторы, соответствующие ветке «истина».

Ключевое слово `if` в переводе с английского означает «если», `else` — «иначе». Т. е., условный оператор звучит следующим образом:

«Если *условное выражение* истинно, выполнять *операторы «истина»*, иначе выполнять *операторы «ложь»*.»

Например, условный оператор, представленный на рисунке 1, имеет вид:

```
if (x < y)
    cout << x;
else
    cout << y;
```

Помимо полного условного оператора существует неполный условный оператор, когда отсутствует ветка `else`. Блок-схема неполного условного оператора приведена на рисунке 2 (сумма двух чисел $x + y$ вычисляется только в случае, когда $x > 0$).

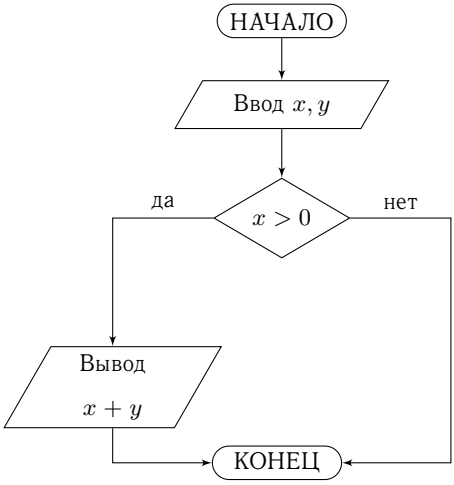

Рис. 2. Неполный условный оператор.

Условный оператор работает до первого знака «;» поэтому, если необходимо выполнить более одного действия, нужные операторы обязательно заключаются в операторные скобки `{...}`

Рассмотрим пример использования условного оператора:

Пример 0.1. Ввести два числа. Если они оба отрицательны, вывести сумму этих чисел, иначе вывести их произведение.

Листинг 1.

```
1 // Если оба числа отрицательны, вывести их сумму,
2 // иначе вывести их произведение
3 #include<iostream>
4 using namespace std;
```

```
5
6  int main(){
7     setlocale (LC_ALL, "rus");
8    int x, y;
9    cout << "x = ";
10   cin >> x;
11   cout << "y = ";
12   cin >> y;
13
14   if (x < 0 && y < 0) {
15     int z = x + y;
16     cout << z << endl;
17   }
18   else {
19     int z = x * y;
20     cout << z << endl;
21   }
22
23 }
```

Переменная **z**, описанная внутри соответствующего блока, существует только внутри этого блока, поэтому переменные, описанные в разных блоках оператора `if` (в строке 15 и строке 19 Листинга 1), являются различными переменными.

Часто встречаются вложенные условные операторы, т. е., в качестве «оператора истины» и в качестве «оператора ложь» выступают условные операторы. Степень вложенности может быть до 256 операторов, но более трех вложенных операторов приводит к практически нечитабельной программе. Ключевое слово `else` в таком случае относится к ближайшему ключевому слову `if`, которому еще не соответствует ключевое слово `else`.

Рассмотрим пример вложенных операторов:

Пример 0.2. Определить к какой четверти относится точка с координатами (x, y).

Листинг 2.

```
1 //определить к какой четверти относится точка с координатами (x,y)
2 #include <iostream>
3 #include <cmath>
4 using namespace std;
5
6 int main(){
```

```
7   setlocale (LC_ALL, "RUS");
8   int x, y;
9   cout << "x = "; //ввод 2 чисел
10  cin >> x;
11  cout << "y = ";
12  cin >> y;
13  if (x*y < 0) // внешний оператор if
14    if ( x < 0) cout << "2 четверть\n"; //внутренний 1
15    else cout << "4 четверть\n";  //внутренний 1 else
16  else       //внешний оператор else
17    if (x < 0) cout << "3 четверть\n"; //внутренний 2
18    else cout << "1 четверть\n"; //внутренний 2 else
19  system("pause");
20  return 0;
21 }
```

В Листинге 2 существует внешний условный оператор (строки 13–18), определяющий знак произведения $x \times y$. В случае, если произведение отрицательно, значит, переменные x и y имеют разные знаки. Необходимо определить какие конкретно знаки имеют эти переменные. Для этого используется внутренний условный оператор (строки 14–15): он определяет знак переменной x. Ключевое слово `else` (строка 15) относится к ближайшему `if` (строка 14).

Ключевое слово `else` (строка 16) относится к ближайшему «незанятому» `if` (строка 13). В данном случае, произведение двух переменных положительно, а это значит, что знаки переменных одинаковы. Необходимо определить какие знаки у переменных. Для этого используется внутренний условный оператор (строки 17–18): он определяет знак переменной x. Ключевое слово `else` (строка 18) относится к ближайшему `if` (строка 17).

Такой уровень вложенности встречается очень редко, так как очень сложно уловить к какому условию относится ключевое слово `else`. Чаще всего используется вложенный условный оператор вида: `if - else if - else if - ... - else`. Такой уровень вложенности более наглядный.

Рассмотрим пример вложенных операторов:

Пример 0.3. Определить взаимное расположение точки с координатами (x, y) и окружности с центром в начале координат и радиусом R.

Листинг 3.

```
//определить взаимное расположение точки с координатами (x,y)
//и окружности с центрм в начале координат и радиусом R
#include <iostream>
#include <cmath>
using namespace std;

int main(){
  setlocale (LC_ALL, "RUS");
  float x, y, R;
  cout << "x = "; //ввод координат
  cin >> x;
  cout << "y = ";
  cin >> y;
  cout << "R="; //ввод радиуса
  cin >> R;

  float z = x * x + y * y //уравнение окружности
  if (z < R * R) // внутри
    cout << "Внутри\n";
  else if (z > R * R) // вне
    cout << "Снаружи\n";
  else  //на границе
    cout << "На границе\n";
  system("pause");
  return 0;
}
```

Иногда полезно иметь возможность выбирать не из двух возможностей и большого числа вложенных циклов, а сразу выбирать из нескольких возможностей. Для этой цели служит оператор выбора.

Оператор выбора можно использовать только для выражений целых и символьных типов. Для вещественных переменных использование оператора выбора невозможно. Константные выражения должны быть различными.

Оператор `switch` имеет следующий формат:

```
switch (⟨выражение⟩) {
   case ⟨константное_выражение_1⟩ : [ ⟨блок_операторов_1⟩ ; [break;]]
   case ⟨константное_выражение_2⟩ : [ ⟨блок_операторов_2⟩ ; [break;]]
   ...
```

```
    case ⟨константное_выражение_N⟩ :[ ⟨блок_операторов_N⟩;[break;]]
    [ default:⟨операторы⟩]
}
```

Оператор выбора работает следующим образом:

1. Сначала вычисляется значение выражения.

2. Значение выражение последовательно сравнивается со всеми константными выражениями. Если найдено совпадение, то начинают выполняться операторы, начиная с соответствующего блока. Оператор `break` (завершение работы текущего оператора) служит для того, чтобы прервать работу оператора выбора.

3. Если совпадений не найдено, выполняется необязательный блок операторов, соответствующий необязательному ключевому слову `default`.

На рисунке 3 представлена блок-схема оператора выбора `switch`.

Рассмотрим пример работы оператора выбора:

Пример 0.4. Ввести символ, характеризующий транспортное средство. Вывести его возможную скорость.

Листинг 4.

```
1  //введена дата. Проверить ее корректность
2  #include <iostream>
3  #include <cmath>
4  using namespace std;
5
6  int main(){
7    setlocale (LC_ALL, "RUS");
8   char a;
9   cout << " Введите символ\n";
10  cout << " b - автобус\n";
11  cout << " c - машина \n";
12  cout << " v - велосипед\n";
13  cout << " p - самолет\n";
14
15  switch(a){
16    case 'b' : cout << "60 км/ч\n";
17    case 'c' : cout << "90 км/ч\n";
18    case 'v' : cout << "15 км/ч\n";
```

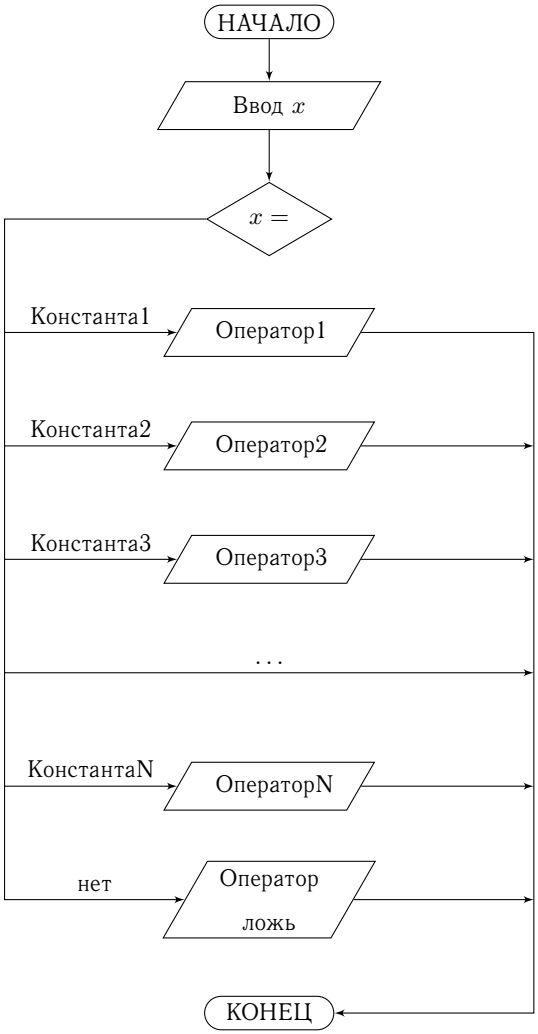

Рис. 3. Оператор выбора

```
19      case 'p' : cout << "800 км/ч\n";
20      default : cout << "90 км/ч\n";
21    }
22    system("pause");
23    return 0;
24  }
```

Если ввести `'c'`, то результат выражения (значение переменной `a` (строка 16

Листинга 4)) равен `'c'`. Совпадение будет найдено в строке 18. На экран будет выведено `"90 км/ч".` Оператор `break` прекращает работу оператора выбора.

Если оператор `break` отсутствует, то результатом работы программы будет следующая запись на экране:

```
"90 км/ч"
"15 км/ч"
"800 км/ч"
"Введен неправильный символ"
```

Может быть так, что нескольким константным выражениям соответствует один блок операторов.

Рассмотрим пример:

Пример 0.5. Введена дата в виде d, m, y. Проверить корректность ввода даты.

Листинг 5.

```
1  //введена дата. Проверить ее корректность
2  #include <iostream>
3  #include <cmath>
4  using namespace std;
5
6  int main(){
7    setlocale (LC_ALL, "RUS");
8    int d, m, y;
9    cout << "Введите дату\n";
10   cin >> d >> m >> y;
11
12   switch (m){
13     case 1 : case 3 : case 5 : case 7 :
14     case 8 : case 10 : case 12 :  //31 день
15       if (d <= 0 || d > 31)
16         cout << "Неправильно введен день\n";
17       else
18         cout << "Дата корректна\n"; break;
19     case 4 : case 6 : case 9 : case 11 : //30 дней
20       if (d <= 0 || d > 30)
21         cout << "Неправильно введен день\n";
22       else
23         cout << "Дата корректна\n"; break;
24     case 2 :
```

```
25      if (y % 400 == 0 || (y % 4 == 0 || y % 100 != 0)) //високосный год
26        if (d <= 0 || d > 29)
27          cout << "Неправильно введен день\n";
28        else
29          cout << "Дата корректна\n"; break;
30      else                                    //невисокосный
31        if (d <= 0 || d > 28)
32          cout << "Неправильно введен день\n";
33        else
34          cout << "Дата корректна\n"; break;
35    default : cout << "Введен неправильный месяц\n";
36   }
37
38   system("pause");
39   return 0;
40 }
```

1.2. Операторы цикла

Цикл — это повторяющиеся определенное количество раз (до выполнения некоторого условия) операторы.

В языке C++ существует три вида операторов цикла: с предусловием, с постусловием и оператор цикла с параметром.

На рисунке 4 представлена блок-схема оператора цикла `while`.

Оператор цикла с предусловием работает по следующему алгоритму:

1. Некоторому параметру присваивается *начальное значение*.
2. Если значение *условия* истинно, то выполняется *блок операторов*. Иначе переход к шагу 4.
3. В *блоке операторов* изменяется параметр. Переход к шагу 2.
4. Выход из цикла.

Оператор цикла с предусловием имеет следующий вид:

`while (`⟨*логическое выражение*⟩`) { `⟨*блок_операторов*⟩`;}`

Ключевое слово `while` переводится как «пока», т. е., выполнять *блок операторов* пока *логическое выражение* истинно.

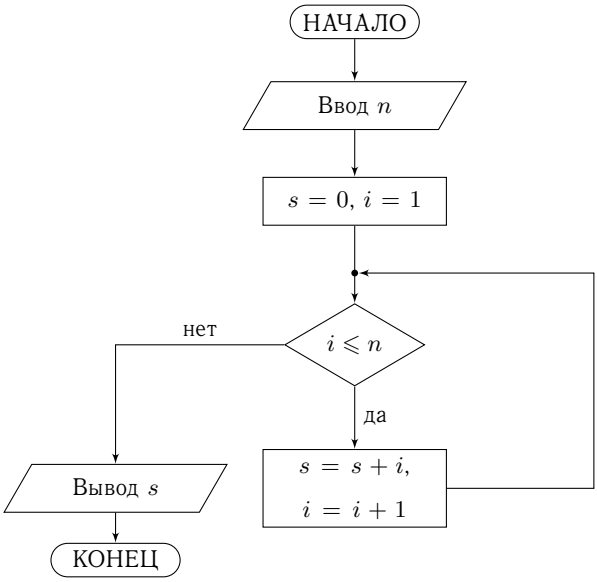

Рис. 4. Оператор цикла с предусловием

Цикл с предусловием из-за того, что сначала проверяется истинность условия, а потом, если условие истинно, выполняется блок операторов. Если условие ложно, цикл не выполнится ни разу и управление программой перейдет к оператору, следующему за циклом.

Например, найти сумму чисел от 1 до N :

```
 int S = 0, N = 10, i = 1;
while (i <= N){
    S += i ;
    i++;
}
```

Блок-схема цикла с постусловием представлена на рисунке 5:

Цикл `do while` называется циклом с постусловием и имеет следующий формат:

`do {`⟨*блок_операторов;*⟩`} while (`⟨*логическое выражение*⟩`)`

Как видно на представленной блок-схеме, оператор цикла с постусловием работает по следующему алгоритму:

1. Некоторому параметру присваивается *начальное значение*.

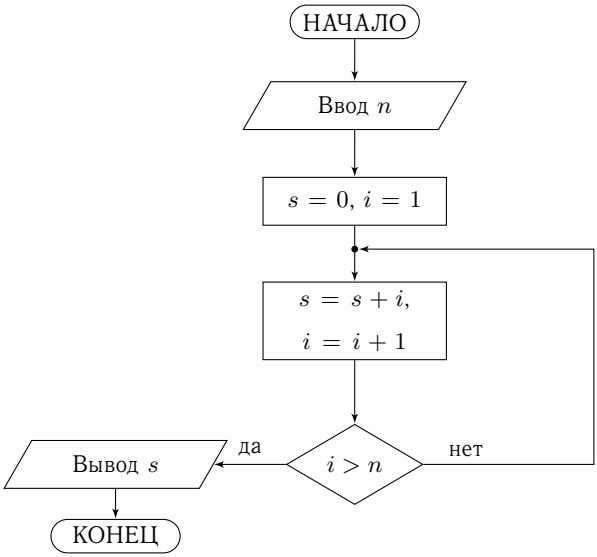

Рис. 5. Оператор цикла с постусловием

2. Выполняется *блок операторов*. В *блоке операторов* изменяется параметр.

3. Проверяется *условие*. Если оно истинное, переход к шагу 2. Иначе переход к шагу 4.

4. Выход из цикла.

В данном случае сначала выполняется блок операторов, а потом проверяется истинность условия. В отличие от цикла с предусловием, если условие ложно, блок оператор все равно выполнится один раз.

Например, найти сумму чисел от 1 до N :

```
 int S = 0, N = 10, i = 1;
do {
    S += i ;
    i++;
}
while (i <= N);
```

При работе с операторами циклов с предусловием и постусловием необходимо следить, чтобы в блоке операторов находился оператор, позволяющий изменять логическое

выражение (в приведенных выше примерах, такой оператор — это увеличение переменной i на единицу), иначе цикл может стать бесконечным.

Оператор цикла с параметром работает по тому алгоритму, что и оператор цикла с предусловием. Это наиболее общий способ организации цикла.

Цикл `for` имеет следующий формат:

```
for (⟨ инициализация; логическое выражение; блок_модификаций⟩ )
    { ⟨блок_операторов⟩; }
```

Инициализация — присвоение одному или нескольким параметрам начального значения, *логическое выражение* — проверка условия окончания работы оператора цикла, *блок модификаций* — модификация одного или нескольких параметров оператора цикла. Блок операторов выполняется до тех пор, пока логическое выражение истинно.

Любой из трех разделов (иногда и все три) может отсутствовать, обязательными являются только две точки с запятой:

```
for (; i <= N; i++)
for (int i = 1; i <= N; )
for (i = 1; ; i++)
for (; i <= N; )
for (; ;)
```

Например, найти сумму чисел от 1 до N :

```
int S = 0; for (int i = 1; i <= N; i++)
    S += i ;
```

Цикл с параметром выполняется по следующему алгоритму:

1. Сначала в разделе инициализации присваиваются начальные значения данным.

2. Проверяется истинность логического выражения. Если выражение истинно, переход к шагу 3. Иначе переход к шагу 5.

3. Выполняется блок операторов.

4. Выполняется блок модификаций. Переход к шагу 2.

5. Конец цикла.

В разделе инициализации и модификаций может быть больше одного выражения, тогда они должны быть перечислены через запятую. Например, другая запись поиска суммы чисел от 1 до N :

```
int S, i;
for (i = 1, S = 0; i <= N; S += i, i++);
```

Как в случае условных операторов, в случае операторов цикла существуют также вложенные циклы. При работе с основными алгоритмами вложенные части используются достаточно часто.

Например, фрагмент кода программы:

```
for (int i = 0; i < N; i++, cout << endl)
    for (int j = 0; j < N; j++)
        cout << j << "";
```

Алгоритм работы:

1. Сначала выполняется раздел инициализации внешнего цикла (`i = 0`). Проверяется истинность логического условия внешнего цикла (`i < N`). Если условие истинно, переход к шагу 2. Иначе переход к шагу 5.

2. Инициализация внутреннего цикла (`j = 0`). Проверка истинности логического условия внутреннего цикла (`j < N`). Если условие истинно, переход к шагу 3. Иначе переход к шагу 5.

3. Выполнение блока операторов (`cout << j << ""`) (Печать в строчку числа j и пробела.)

4. Выполнение раздела модификаций внутреннего цикла. (Увеличение j на единицу). Проверка истинности логического условия внутреннего цикла (`j < N`). Если условие истинно, переход к шагу 3. Иначе переход к шагу 5.

5. Выполнение раздела модификаций внешнего цикла. (Увеличение i на единицу и переход на следующую строку). Проверяется истинность логического условия внешнего цикла (`i < N`). Если условие истинно, переход к шагу 2. Иначе переход к шагу 6.

6. Конец цикла.

В результате будет напечатано N строк, содержащие числа от 0 до $N-1$. Например, если $N = 5$:

0 1 2 3 4

0 1 2 3 4

0 1 2 3 4

0 1 2 3 4

0 1 2 3 4

1.3. Операторы перехода

В языке C++ предусмотрены четыре оператора безусловного перехода: `goto`, `return`, `break`, `continue`. Операторы `goto` и `return` можно применять в любом месте программы, `break` и `continue` связаны с циклами, `break` используется также в операторы выбора `switch`.

Оператор безусловного перехода `return` служит для возврата управления из функции в точку вызова функции. Стандарты языка требуют, чтобы любая функция, кроме имеющих тип `void`, имела оператор `return`.

Синтаксис имеет следующий вид:

`return` ⟨*значение*⟩

Тип значения должен совпадать с типом функции.

Например,

```
int main(){
    \textit{операторы}
    return 0;
}
```

Оператор `goto` используется крайне редко, поскольку нарушает логическую последовательность программы. Для этого оператора используются метки, любые буквенные выражения или целые числа. Метка и двоеточие после нее ставится в то место программы, куда должно быть передано управление; в место, где необходимо прервать выполнение программы и вернуться к метку, ставится оператор вида: `goto`⟨*имя метки*⟩.

Придумать ситуацию, при которой нельзя обойтись без оператора `goto`, практически невозможно, однако, при осторожном и правильном использовании, например, при выходе из глубоко вложенных циклов, можно пользоваться этим оператором.

Например, с помощью оператора `goto` организован цикл, выводящий на экран числа от 1 до 100:

```
x = 1;
lab:
cout << x++ << "";
if (x < 100)goto lab;
```

Оператор прерывания циклов `break` используется при прерывании цикла, вне зависимости от истинности логического условия. В случае вложенных циклов прерывается только цикл, содержащий `break`.

Например, найти все простые числа от 2 до N :

```
int N = 20;
for (int i = 2; i <= N; i++){
    bool fl = true;
    for (int j = 2; j <= sqrt((float)i); j++)
        if (j % i == 0){
            fl = false;
            break;
        }
    if (fl)cout << i << ""
}
```

Для каждого числа (`i`) от 2 до N перебираем делители до корня из i. Если встретили делитель (`j%i == 0`), прерываем внутренний цикл.

Оператор прерывания `continue` прерывает только текущую итерацию цикла. Например, вывести на экран все числа, не кратные 5:

```
int N = 20;
for (int i = 2; i <= N; i++)
    if (i % 5 == 0)continue;
    else cout << i << "";
```