Занятие 7**. Знакомство с циклами**

Рассмотрим ситуацию, из которой видно, что ветвления, присваивания, ввод и вывод не позволяют эффективно разрешить проблемы, встающие перед программистом.

Пусть требуется вывести на экран одно сообщение «Привет, мир!». Мы с ходу записываем

```
cout << "Привет, мир!\n".
```

Если потребуется вывести две строки, наше решение будет таким

```
cout << "Привет, мир!\n";
cout << "Привет, мир!\n".
```

Будем увеличивать количество строк: 3, 4, 5, … . При двадцати строках мы все еще готовы идти проторенным путем:

```
cout << "Привет, мир!\n";
cout << "Привет, мир!\n";
                                  } 20 строк
      …
cout << "Привет, мир!\n".
```

Усложним задачу: пусть количество печатаемых строк равняется числу `n` (1 <= `n` <= 20). Припомнив ветвления и собрав в кулак свою волю, строим алгоритм.

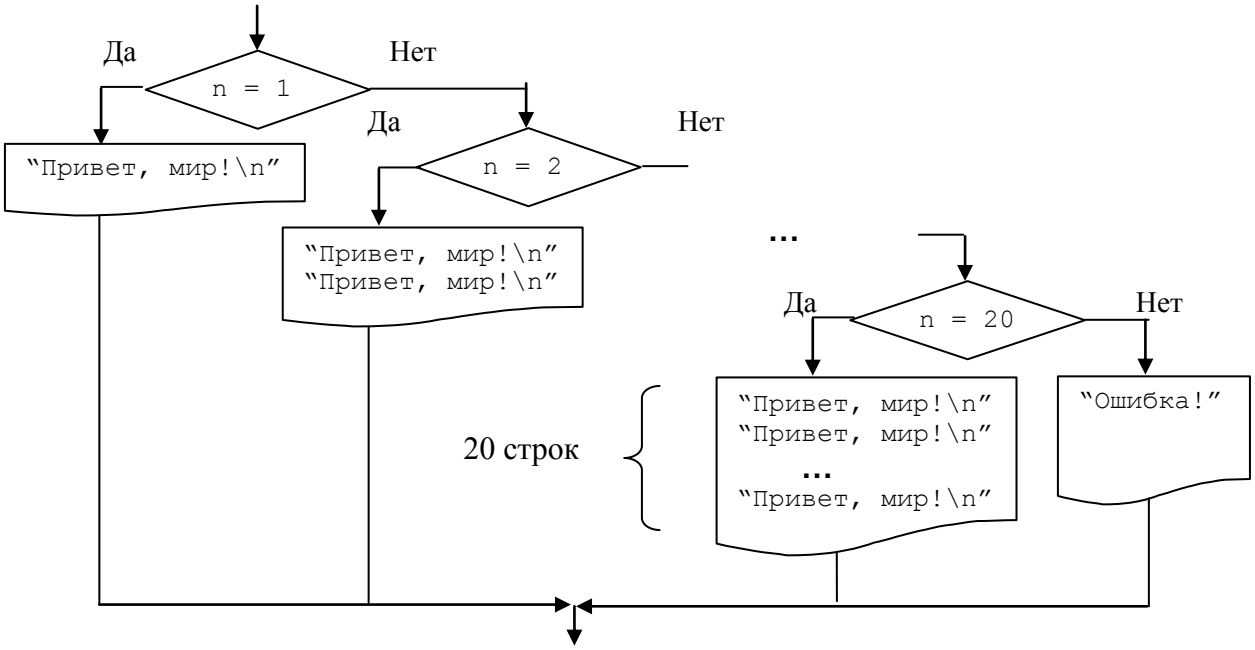


Рис. 1. Фрагмент алгоритма для вывода строки «Привет, мир!» `n` раз (1 <= `n` <= 20)

Видим, что решение с использованием известных средств возможно, но оно очень громоздкое. К тому же если потребуется увеличить n, скажем, до 40, то придется очень сильно изменить алгоритм и программу. На наше счастье существует очень эффективный способ реализации этой задачи, основанный на идее повторения.

Основным действием в нашей задаче является вывод строки «Привет, мир!». Первый вариант реализации идеи повторения приведен на рис. 2.

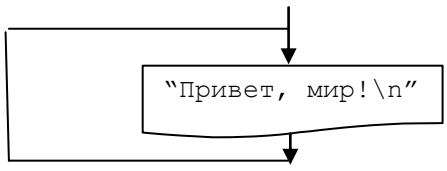


Рис. 2. Реализация идеи повторения – вариант 1

Сразу же возникает вопрос: «Как закончить вычисления?», - потому что нарисованный фрагмент алгоритма после запуска никогда не останавливается. Добавим возможность выбора.

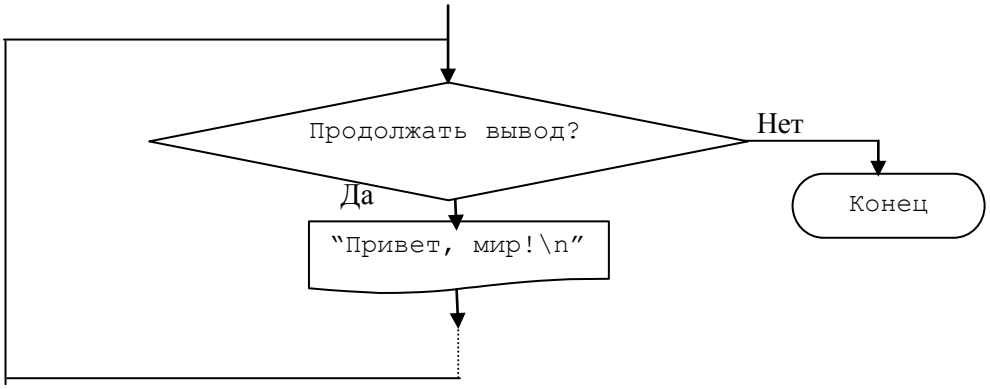


Рис. 3. Реализация идеи повторения – вариант 2

Этот вариант показывает путь, который может привести к завершению вычислений. Но решение не завершено, поскольку при первом ответе «Да» на вопрос «Продолжать вывод?» и все последующие ответы также будут равны «Да». Повторения все еще остаются бесконечными. Нужен механизм изменения ответа на вопрос «Продолжать вывод?», который, в свою очередь, тоже должен быть управляемым.

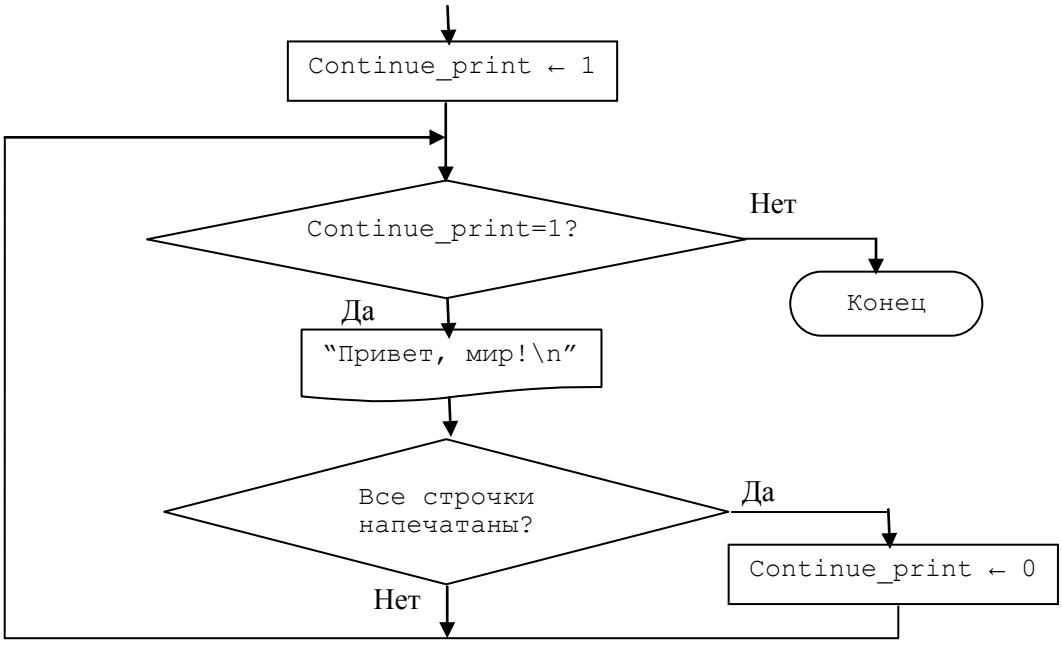


Рис. 4. Формализованный вариант реализация идеи повторения

На рис. 4 произведена формализация алгоритма: вопрос «Продолжать вывод?» заменен на анализ значения переменной `continue_print`. Эта переменная будет хранить `1` до тех пор, пока требуется продолжать вывод строчек. Когда же все строчки будут напечатаны, переменной `continue_print` будет присвоено значение `0`. Для того чтобы различать запись отношений и присваивания, на рис. 4 для операции присваивания используется символ `'←'`.

Для получения окончательного решения нужно придумать формализацию для вопроса «Все строчки напечатаны?». Нетрудно сделать, чтобы перед началом работы алгоритма с повторениями в переменной `n` было задано количество печатаемых строчек – это можно ввести с клавиатуры. После печати каждой строчки будем уменьшать значение `n` на 1 для того, чтобы новое значение `n` хранило количество еще не напечатанных строк. И в переменную `continue_print` нужно записать 0 тогда, когда количество еще не напечатанных строк станет

равно 0. На рис. 5 приведен полный алгоритм для задачи вывода на экран строчек «Привет, мир!», количество которых задается с клавиатуры.

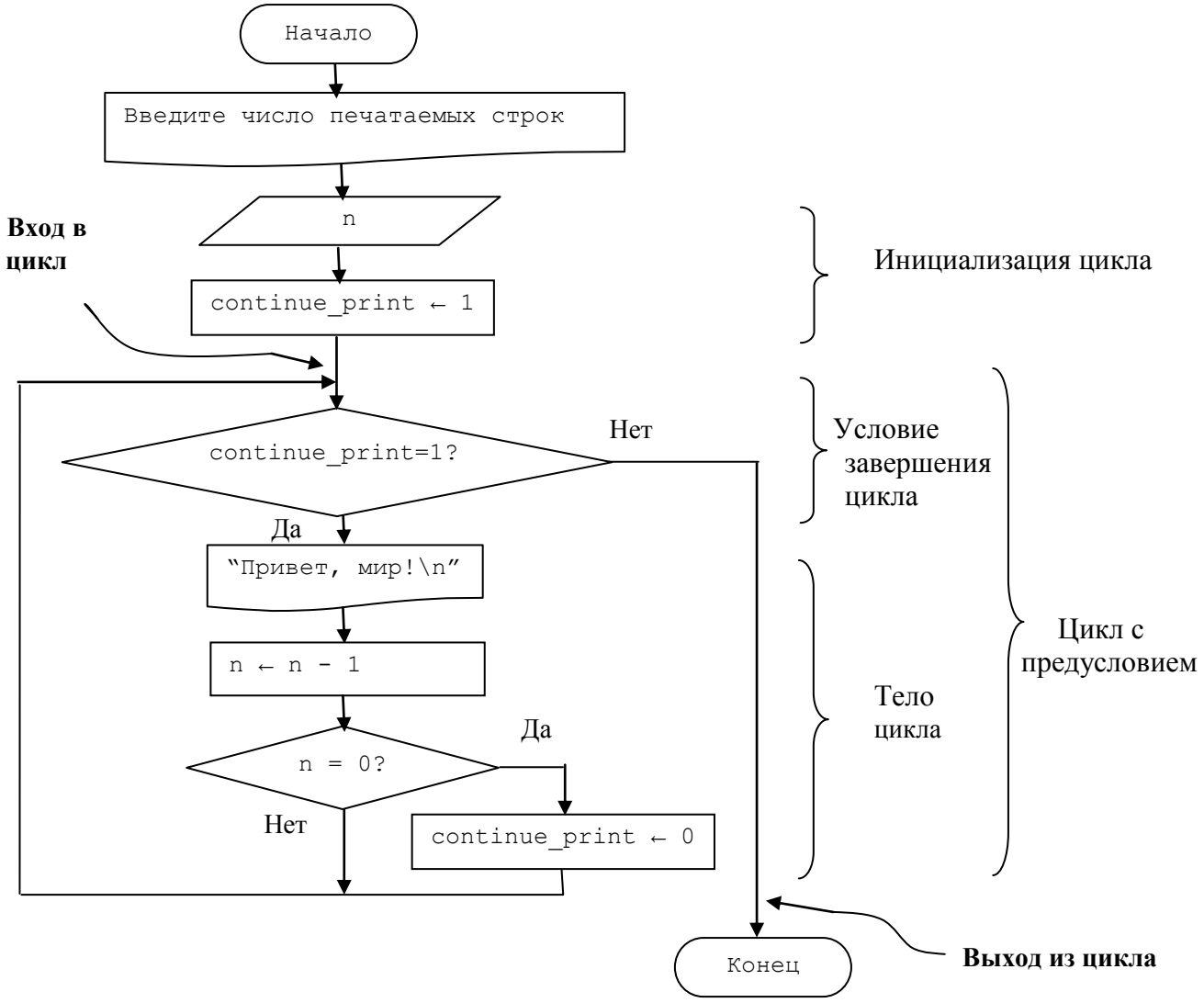


Рис. 5. Алгоритм для задачи вывода на экран строчек «Привет, мир!», количество которых задается с клавиатуры

Данный алгоритм разработан исходя из постановки задачи, которая мысленно удерживалась в голове в виде экранного диалога, который может выглядеть следующим образом:

```
Введите число печатаемых строк
3
Привет, мир!
Привет, мир!
Привет, мир!
```

Рассмотрим полученный результат подробнее. Алгоритмическая структура, в которой встречаются повторения, называется циклом. Цикл можно разделить на две части: условие, определяющее выход из цикла, и группу повторяющихся команд, составляющих тело цикла. Понятно, что хотя условие выделено отдельно, но оно также вычисляется при каждом повторении.

Каждое повторение еще называется итерацией. Приведенный на рис. 5 тип цикла называют циклом с предусловием, он имеет следующие особенности.

1. Условие завершения цикла предшествует его телу – отсюда, собственно, и возникло название. И поэтому возможна ситуация, когда операторы, входящие в тело цикла, не выполнятся ни разу. Так в примере если бы при проверке завершения цикла переменная

continue_print сразу оказалась не равной 1, то алгоритм сразу и завершил бы свою работу после передачи управления на команду «Закончить вычисления» - блок «Конец».

2. Этот цикл является конструкцией структурного проектирования программ, так он имеет только один вход и только один выход. Следовательно, он очень удобен при проектировании алгоритмов и их отладке.
3. Условие выхода из цикла нужно проектировать так, чтобы переход к телу цикла был помечен словом «Да» - это соответствует истинному значению условия. Выход из цикла должен быть помечен словом «Нет» - цикл завершается, когда значение условия становится ложным.

Важным элементом являются операторы инициализации цикла, они обеспечивают правильное выполнение алгоритма при первом прохождении по циклу. Пусть мы выполняем проверку условия «continue_print=1?» в первый раз, и при этом нам всегда *требуется напечатать хотя бы одну строку*. Тогда чтобы сразу не перейти к блоку «Конец», значение переменной continue_print должно быть равно 1, что и обеспечивается оператором инициализации. Требование *печати хотя бы одной строки* мы сформулировали только сейчас, но подразумевали его выполнение. (Так у программистов бывает часто – требования могут быть сформулированы где-нибудь в процессе составления алгоритма или программы). Если это требование изменить, то может измениться и состав, и содержание операторов инициализации.

Далее, пусть мы вошли в цикл. Оператор печати сообщения «Привет, мир!» не содержит переменных и поэтому не требует инициализации. Оператор уменьшения переменной n на 1 требует нашего внимания. Из n вычитается 1 - контролируем себя вопросом: «А чему же должно быть равно значение n в самый первый раз?». По смыслу задачи значение n – это количество еще не напечатанных строк, и мы удостоверяемся, что не забыли поставить оператор, обеспечивающий занесение значения в переменную n. Это оператор ввода.

Остальные операторы в теле цикла уже не используют новых переменных, поэтому разговор об операторах инициализации цикла закончен.

Следующий этап – кодирование алгоритма на языке С++.

```
// Ввести натуральное N<25  и N раз напечатать "Привет, мир!".
#include <iostream>
using namespace std;
int main()
{
     int n, continue_print;
     cout << "Введите число печатаемых строк\n";
     cin >> n;
     continue_print = 1;
     while (continue_print == 1)
          cout << "Привет, мир!\n";
          n = n - 1;
          if (n == 0) continue_print = 0;
     return 0;
}
```

Рис. 6. Программа на языке С++ для задачи вывода на экран строчек «Привет, мир!»

Для реализации цикла с предусловием в языке С++ существует оператор цикла while, имеющий следующую структуру:

while (<условие>) <повторяющийся_оператор>;

Здесь «while» - это ключевое слово языка С++, <условие> записывается в виде арифметического отношения (известного нам из занятия 4) или в виде логического выражения (известного нам из занятий 5 и 6). В редких случаях <условие> записывается в виде константы – такие примеры могут встретиться в изданиях по языку С++, но не в нашем учебнике. Условие

переносится в программу из блока проверки завершения цикла алгоритма; оно обязательно заключается в круглые скобки.

В качестве повторяющегося оператора, являющегося телом цикла, можно использовать все изученные ранее операторы: присваивания, ввода, вывода данных, ветвления, - а также сам оператор цикла. В последнем случае говорят, что цикл является вложенным. В нашем примере тело цикла состоит из трех операторов: вывода сообщения «Привет, мир!», уменьшения `n` на 1 (присваивание) и оператора ветвления.

Проверим теперь результат нашей работы, запустив программу на выполнение. Экран заполняется строчками «Привет, мир!», но программа не завершает работу. Дело в том, что оператор `while` имеет область действия, которая охватывает точно один оператор. Поэтому, повторяя только оператор вывода сообщения «Привет, мир!», компьютер не может изменить значение условия и завершить работу программы. Для расширения области действия оператора `while` на несколько операторов применяют фигурные скобки. Несколько операторов, заключенные в фигурные скобки, называются составным оператором. Правильно работающая программа приведена на рис. 7.

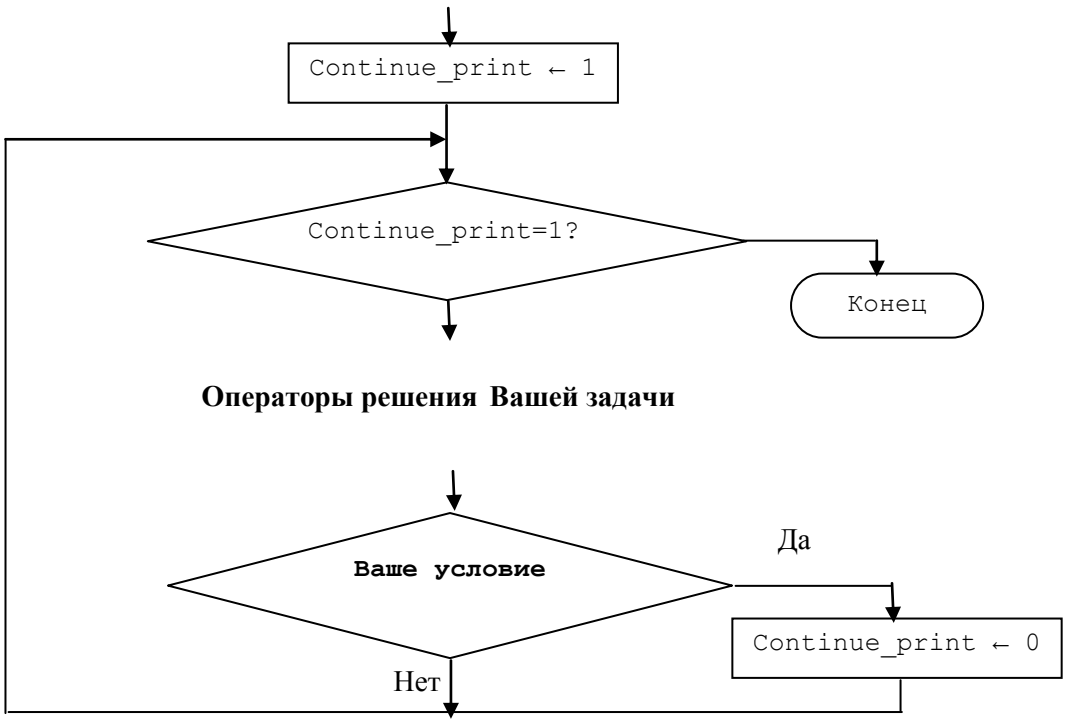
```
// Ввести натуральное N<25  и N раз напечатать "Привет, мир!".
#include <iostream>
using namespace std;
int main()
{
     int n, continue_print;
     cout << "Введите число печатаемых строк\n";
     cin >> n;
     continue_print = 1;
     while (continue_print == 1)
     {
          cout << "Привет, мир!\n";
          n = n - 1;
          if (n == 0) continue_print = 0;
     }
     return 0;
}
```

Составной оператор

Рис. 7. Программа на языке C++ для задачи вывода на экран строчек «Привет, мир!» с составным оператором

Для внимательного читателя скажем, что, во-первых, вводимое число печатаемых строк может и превышать значение 25. Только вряд ли понадобится такое число строк, которое не умещается на экране. А мы предполагаем работу с экраном, на котором размещается 25 строк по 80 символов.

Во-вторых, переменная `continue_print` принимает значение 0 именно тогда, когда и переменная `n` устанавливается в 0. Поэтому переменную `continue_print` можно исключить из программы, но мы не рекомендуем это делать при написании первых программ. Если требуется решить задачу с повторяющейся обработкой, то лучше всего воспользоваться алгоритмической заготовкой следующего вида, которая уже является решением задачи на 30-50%.



а)

```
#include <iostream>
using namespace std;
int main()
{
      int continue_print;
     …

      continue_print = 1;
      while (continue_print == 1)
      {
          Операторы решения Вашей задачи

          if (Ваше условие) continue_print = 0;
      }
      return 0;
}
```

б)

Рис. 8. Алгоритмическая (а) и программная (б) заготовка решения задачи с повторяющейся обработкой

Упражнения

1. Дано натуральное число **N**. Вывести на экран натуральные числа от 1 до **N** в порядке возрастания по одному числу в строке.
2. Дано натуральное число **N**. Вывести на экран натуральные числа от 1 до **N** в порядке убывания по одному числу в строке.